Intelligence Artificielle

Résolution de Problèmes

Emmanuel ADAM

Université Polytechnique des Hauts-De-France

UPHF/INSA HdF

Introduction

Résolution de Problèmes

- Définir les problèmes
  - Nature
- Représentation des problèmes
- Algorithmes de résolution

Introduction

Résolution de Problèmes

- Définir les problèmes
  - Nature
- Représentation des problèmes
  - Logique
- Algorithmes de résolution

Introduction

Résolution de Problèmes

- Définir les problèmes
  - Nature
- Représentation des problèmes
  - Logique
  - Arbres et Graphes d'états
- Algorithmes de résolution

Introduction

Résolution de Problèmes

- Définir les problèmes
  - Nature
- Représentation des problèmes
  - Logique
  - Arbres et Graphes d'états
- Algorithmes de résolution

Introduction

Résolution de Problèmes

- Définir les problèmes
    - Nature
- Représentation des problèmes
    - Logique
    - Arbres et Graphes d'états
- Algorithmes de résolution

Introduction

Résolution de Problèmes

- Définir les problèmes
  - Nature
- Représentation des problèmes
  - Logique
  - Arbres et Graphes d'états
- Algorithmes de résolution

Introduction

Résolution de Problèmes

- Définir les problèmes
  - Nature
- Représentation des problèmes
  - Logique
  - Arbres et Graphes d'états
- Algorithmes de résolution

Définition d'un problème I

Exemple de problème

Se rendre de son domicile (Onnaing) à un lieu précis (Arrivée) d'une autre ville (Théâtre Lille), le plus rapidement possible, avant une date donnée (obligation de passer par une ville étape B).
(exemple : de Onnaing au Théâtre L'antre 2 de Lille 2 avant 20h)

Définitions

Espace d'états : description (abstraction) du monde réel définissant le problème

Exemple : réseau des bus, trams, metro . . .

Buts : sous ensemble de l'espace d'états

Exemple : Lille-Théâtre19h50

Action (opérateurs) : déplacement dans l'espace d'états

Définition d'un problème II

Définitions

Solution du problème : la séquence d'action menant de l'état initial à l'état objectif

Algorithme de Recherche : procédure qui calcule une (ou plusieurs) solution à partir d'un problème (état initial, actions, états objectifs).

Types de problèmes

Les types de problèmes

à état unique : déterministe, l'effet de chaque action est connu.

à états multiples : non déterministe, mais l'effet des actions est globalement connu

à connaissances incompletes : monde partiellement connu ou évolutif (besoin de "capter" les états)

d'exploration : espace d'états inconnu, à découvrir par exploration

Types de problèmes

Les types de problèmes

à état unique : déterministe, l'effet de chaque action est connu.

à états multiples : non déterministe, mais l'effet des actions est globalement connu

à connaissances incompletes : monde partiellement connu ou évolutif (besoin de "capter" les états)

d'exploration : espace d'états inconnu, à découvrir par exploration

Types de problèmes

Les types de problèmes

à état unique : déterministe, l'effet de chaque action est connu.

à états multiples : non déterministe, mais l'effet des actions est globalement connu

à connaissances incompletes : monde partiellement connu ou évolutif (besoin de "capter" les états)

d'exploration : espace d'états inconnu, à découvrir par exploration

Types de problèmes

Les types de problèmes

à état unique : déterministe, l'effet de chaque action est connu.

à états multiples : non déterministe, mais l'effet des actions est globalement connu

à connaissances incompletes : monde partiellement connu ou évolutif (besoin de "capter" les états)

d'exploration : espace d'états inconnu, à découvrir par exploration

Types de problèmes

Les types de problèmes

à état unique : déterministe, l'effet de chaque action est connu.

à états multiples : non déterministe, mais l'effet des actions est globalement connu

à connaissances incompletes : monde partiellement connu ou évolutif (besoin de "capter" les états)

d'exploration : espace d'états inconnu, à découvrir par exploration

Espace d'états de l'exemple

Espace d'états de l'exemple

Action	*Etat*	*Action*	*Etat*
A00 :Onnaing-bus18h00	E0.0 :Valenciennes18h20	A10 :train18h00	E1.0 :Lille18h40
A01 :Onnaing-bus18h10	E0.1 :Valenciennes18h30	A11 :train18h30	E1.1 :Lille19h10
A02 :Onnaing-bus18h20	E0.2 :Valenciennes18h40	A12 :train19h00	E1.2 :Lille19h40
A03 :Onnaing-bus18h30	E0.3 :Valenciennes18h50	A13 :train19h30	E1.3 :Lille20h10
A04 :Onnaing-bus18h40	E0.4 :Valenciennes19h00	A14 :train20h00	E1.4 :Lille20h40
A05 :Onnaing-bus18h50	E0.5 :Valenciennes19h10	A15 :train20h30	E1.5 :Lille21h10

Action	*Etat*
A20 :metro18h50	E2.0 :Lille-Theatre19h10
A21 :metro19h05	E2.1 :Lille-Theatre19h25
A22 :metro19h20	E2.2 :Lille-Theatre19h40
A23 :metro19h35	E2.3 :Lille-Theatre19h55
A24 :metro19h50	E2.4 :Lille-Theatre20h10
A25 :metro20h05	E2.5 :Lille-Theatre20h25

Etat initial E0 = Onnaing ; **Etat Final** désiré EF = E2.3 :Lille-Theatre19h55

Types de problèmes

Retour sur les types de problèmes

- **à état unique** : recherche évidente
- **à états multiples** : recherche facile
- **à connaissances incompletes** : *le type le plus courant* ; recherche de solution moins évidente, besoin de percevoir l'état en cours ; possibilité de ramener en partie aux types précédents
- **d'exploration** : alternance de phases de génération d'états / recherche

Types de problèmes

Retour sur les types de problèmes

à état unique : recherche évidente

à états multiples : recherche facile

à connaissances incompletes : *le type le plus courant* ; recherche de solution moins évidente, besoin de percevoir l'état en cours ; possibilité de ramener en partie aux types précédents

d'exploration : alternance de phases de génération d'états / recherche

Types de problèmes

Retour sur les types de problèmes

à état unique : recherche évidente

à états multiples : recherche facile

à connaissances incompletes : *le type le plus courant* ; recherche de solution moins évidente, besoin de percevoir l'état en cours ; possibilité de ramener en partie aux types précédents

d'exploration : alternance de phases de génération d'états / recherche

Types de problèmes

Retour sur les types de problèmes

à état unique : recherche évidente

à états multiples : recherche facile

à connaissances incompletes : *le type le plus courant* ; recherche de solution moins évidente, besoin de percevoir l'état en cours ; possibilité de ramener en partie aux types précédents

d'exploration : alternance de phases de génération d'états / recherche

Types de problèmes

Retour sur les types de problèmes

à état unique : recherche évidente

à états multiples : recherche facile

à connaissances incompletes : *le type le plus courant* ; recherche de solution moins évidente, besoin de percevoir l'état en cours ; possibilité de ramener en partie aux types précédents

d'exploration : alternance de phases de génération d'états / recherche

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, . . .

Toys Problems

Exemple de Toys Problems

Les Toys Problems sont des problèmes types utilisés pour tester des algorithmes. Parmi ceux-ci :

- le taquin
- le chien, la chèvre et le chou
- le Wumpus
- l'aspirateur
- le voyageur de commerce
- le labyrinthe
- les mots croisés
- l'arithmétique cryptée
- les jeux d'échecs, de dames, ...

Une méthode de résolution

Une méthode de résolution ?

- Il n'existe de pas de méthode générale de résolution de problème
- Nécessité
  - de décrire formellement le problème, les états, les actions
- Résolution constructive = Proposer + Evaluer une solution

Une méthode de résolution

Une méthode de résolution ?

- Il n'existe de pas de méthode générale de résolution de problème
- Nécessité
  - de décrire formellement le problème, les états, les actions
  - de procédure d'évaluation de solution
- Résolution constructive = Proposer + Evaluer une solution

Une méthode de résolution

Une méthode de résolution ?

- Il n'existe de pas de méthode générale de résolution de problème
- Nécessité
  - de décrire formellement le problème, les états, les actions
  - de procédure d'évaluation de solution
- Résolution constructive = Proposer + Evaluer une solution

Une méthode de résolution

Une méthode de résolution ?

- Il n'existe de pas de méthode générale de résolution de problème
- Nécessité
  - de décrire formellement le problème, les états, les actions
  - de procédure d'évaluation de solution
- Résolution constructive = Proposer + Evaluer une solution

Une méthode de résolution

Une méthode de résolution ?

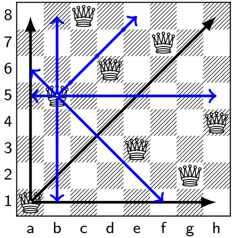
- Il n'existe de pas de méthode générale de résolution de problème
- Nécessité
  - de décrire formellement le problème, les états, les actions
  - de procédure d'évaluation de solution
- Résolution constructive = Proposer + Evaluer une solution

Une méthode de résolution

Une méthode de résolution ?

- Il n'existe de pas de méthode générale de résolution de problème
- Nécessité
  - de décrire formellement le problème, les états, les actions
  - de procédure d'évaluation de solution
- Résolution constructive = Proposer + Evaluer une solution

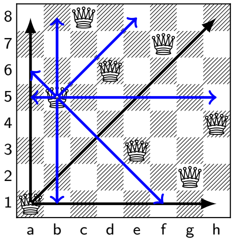
Le problème des 8 reines



Le problème des 8 reines

- **But** : placer 8 reines sur un échiquier de 8×8, sans attaque possible
- **état initial** : échiquier vide
- **actions/opérateurs** : ajouter une reine sur l'échiquier
- Résoudre des solutions partielles :
  - Placer la première reine, puis placer les suivantes dans les cases restantes de l'échiquier
- **Complexité** : $O(n!)$

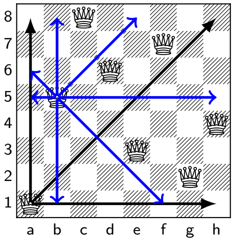
Le problème des 8 reines



Le problème des 8 reines

- **But** : placer 8 reines sur un échiquier de 8×8, sans attaque possible
- **état initial** : échiquier vide
- **actions/opérateurs** : ajouter une reine sur l'échiquier
- Résoudre des solutions partielles :
  - Placer la première reine, puis placer les suivantes dans les cases restantes de l'échiquier
- **Complexité :** $O(n!)$

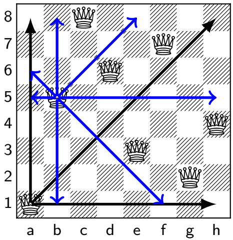
Le problème des 8 reines



Le problème des 8 reines

- **But** : placer 8 reines sur un échiquier de 8×8, sans attaque possible
- **état initial** : échiquier vide
- **actions/opérateurs** : ajouter une reine sur l'échiquier
- Résoudre des solutions partielles :
  - Placer la première reine, puis placer les suivantes dans les cases restantes de l'échiquier
- **Complexité :** $O(n!)$

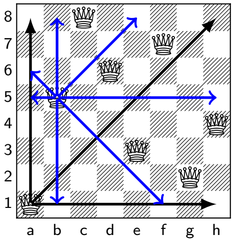
Le problème des 8 reines



Le problème des 8 reines

- **But** : placer 8 reines sur un échiquier de 8×8, sans attaque possible
- **état initial** : échiquier vide
- **actions/opérateurs** : ajouter une reine sur l'échiquier
- Résoudre des solutions partielles :
  - Placer la première reine, puis placer les suivantes dans les cases restantes de l'échiquier
- **Complexité :** $O(n!)$

Le problème des 8 reines



Le problème des 8 reines

- **But** : placer 8 reines sur un échiquier de 8×8, sans attaque possible
- **état initial** : échiquier vide
- **actions/opérateurs** : ajouter une reine sur l'échiquier
- Résoudre des solutions partielles :
  - Placer la première reine, puis placer les suivantes dans les cases restantes de l'échiquier
- **Complexité :** $O(n!)$

Le taquin-8

4	3	5		1	2	3
1	6	2	$\rightarrow$	4	5	6
7	8	.		7	8	.

Le problème du taquin 8

- **But** : agencer les pièces en ordre croissant en un coût minimum
- **état initial** grille de 8 pièces désordonnées, plus une case vide
- **Actions/opérateurs** : déplacer la case vide à gauche (L), à droite (R), en haut (U), en bas (D)
- **coût** : 1 point par déplacement
- Résoudre par solutions partielles
- **complexité** : *NP* (polynomial non-déterministe)

Le taquin-8

4	3	5		1	2	3
1	6	2	$\rightarrow$	4	5	6
7	8	.		7	8	.

Le problème du taquin 8

- **But** : agencer les pièces en ordre croissant en un coût minimum
- **état initial** grille de 8 pièces désordonnées, plus une case vide
- **Actions/opérateurs** : déplacer la case vide à gauche (L), à droite (R), en haut (U), en bas (D)
- **coût** : 1 point par déplacement
- Résoudre par solutions partielles
- **complexité** : *NP* (polynomial non-déterministe)

Le taquin-8

4	3	5		1	2	3
1	6	2	$\rightarrow$	4	5	6
7	8	.		7	8	.

Le problème du taquin 8

- **But** : agencer les pièces en ordre croissant en un coût minimum
- **état initial** grille de 8 pièces désordonnées, plus une case vide
- **Actions/opérateurs** : déplacer la case vide à gauche (L), à droite (R), en haut (U), en bas (D)
- **coût** : 1 point par déplacement
- Résoudre par solutions partielles
- **complexité** : *NP* (polynomial non-déterministe)

Le taquin-8

4	3	5		1	2	3
1	6	2	$\rightarrow$	4	5	6
7	8	.		7	8	.

Le problème du taquin 8

- **But** : agencer les pièces en ordre croissant en un coût minimum
- **état initial** grille de 8 pièces désordonnées, plus une case vide
- **Actions/opérateurs** : déplacer la case vide à gauche (L), à droite (R), en haut (U), en bas (D)
- **coût** : 1 point par déplacement
- Résoudre par solutions partielles
- **complexité** : *NP* (polynomial non-déterministe)

Le taquin-8

4	3	5		1	2	3
1	6	2	$\rightarrow$	4	5	6
7	8	.		7	8	.

Le problème du taquin 8

- **But** : agencer les pièces en ordre croissant en un coût minimum
- **état initial** grille de 8 pièces désordonnées, plus une case vide
- **Actions/opérateurs** : déplacer la case vide à gauche (L), à droite (R), en haut (U), en bas (D)
- **coût** : 1 point par déplacement
- Résoudre par solutions partielles
- **complexité** : *NP* (polynomial non-déterministe)

Le taquin-8

4	3	5		1	2	3
1	6	2	$\rightarrow$	4	5	6
7	8	.		7	8	.

Le problème du taquin 8

- **But** : agencer les pièces en ordre croissant en un coût minimum
- **état initial** grille de 8 pièces désordonnées, plus une case vide
- **Actions/opérateurs** : déplacer la case vide à gauche (L), à droite (R), en haut (U), en bas (D)
- **coût** : 1 point par déplacement
- Résoudre par solutions partielles
- **complexité** : *NP* (polynomial non-déterministe)

Le taquin-8

4	3	5		1	2	3
1	6	2	$\rightarrow$	4	5	6
7	8	.		7	8	.

Le problème du taquin 8

- **But** : agencer les pièces en ordre croissant en un coût minimum
- **état initial** grille de 8 pièces désordonnées, plus une case vide
- **Actions/opérateurs** : déplacer la case vide à gauche (L), à droite (R), en haut (U), en bas (D)
- **coût** : 1 point par déplacement
- Résoudre par solutions partielles
- **complexité** : *NP* (polynomial non-déterministe)

Deux méthodes de résolution constructive

Deux méthodes de résolution constructive

- Avancer par étapes, par solutions partielles
  - Exemple : pour le voyage : trouver les états liés directement à l'arrivée, trouver les états menant à ces états, ..., jusqu'à l'état initial
- Décomposer en sous-problèmes

Deux méthodes de résolution constructive

Deux méthodes de résolution constructive

- Avancer par étapes, par solutions partielles
  - Exemple : pour le voyage : trouver les états liés directement à l'arrivée, trouver les états menant à ces états, . . ., jusqu'à l'état initial
- Décomposer en sous-problèmes
  - Exemple : Tours de Hanoi

Deux méthodes de résolution constructive

Deux méthodes de résolution constructive

- Avancer par étapes, par solutions partielles
  - Exemple : pour le voyage : trouver les états liés directement à l'arrivée, trouver les états menant à ces états, . . ., jusqu'à l'état initial
- Décomposer en sous-problèmes
  - Exemple : Tours de Hanoï

Deux méthodes de résolution constructive

Deux méthodes de résolution constructive

- Avancer par étapes, par solutions partielles
  - Exemple : pour le voyage : trouver les états liés directement à l'arrivée, trouver les états menant à ces états, ..., jusqu'à l'état initial
- Décomposer en sous-problèmes
  - Exemple : Tours de Hanoï

Deux méthodes de résolution constructive

Deux méthodes de résolution constructive

- Avancer par étapes, par solutions partielles
  - Exemple : pour le voyage : trouver les états liés directement à l'arrivée, trouver les états menant à ces états, . . ., jusqu'à l'état initial
- Décomposer en sous-problèmes
  - Exemple : Tours de Hanoï

Représentation par graphe d'états

Graphe d'états

Représentation par graphe des états du problème :

- Les **nœuds** représentent les états
- Un **arc** (i,j) représentent l'opération/l'action permettant de l'état i à l'état j
- **Solution** = chemin entre l'état initial et l'état final
- **Recherche** de Solution = Recherche du/d'un chemin entre l'état initial et l'état final

Représentation par graphe d'états

Graphe d'états

Représentation par graphe des états du problème :

- Les **nœuds** représentent les états
- Un **arc** (i,j) représentent l'opération/l'action permettant de l'état i à l'état j
- **Solution** = chemin entre l'état initial et l'état final
- **Recherche** de Solution = Recherche du/d'un chemin entre l'état initial et l'état final

Représentation par graphe d'états

Graphe d'états

Représentation par graphe des états du problème :

- Les **nœuds** représentent les états
- Un **arc** (i,j) représentent l'opération/l'action permettant de l'état i à l'état j
- **Solution** = chemin entre l'état initial et l'état final
- **Recherche** de Solution = Recherche du/d'un chemin entre l'état initial et l'état final

Représentation par graphe d'états

Graphe d'états

Représentation par graphe des états du problème :

- Les **nœuds** représentent les états
- Un **arc** (i,j) représentent l'opération/l'action permettant de l'état i à l'état j
- **Solution** = chemin entre l'état initial et l'état final
- **Recherche** de Solution = Recherche du/d'un chemin entre l'état initial et l'état final

Représentation par graphe d'états

Graphe d'états

Représentation par graphe des états du problème :

- Les **nœuds** représentent les états
- Un **arc** (i,j) représentent l'opération/l'action permettant de l'état i à l'état j
- **Solution** = chemin entre l'état initial et l'état final
- **Recherche** de Solution = Recherche du/d'un chemin entre l'état initial et l'état final

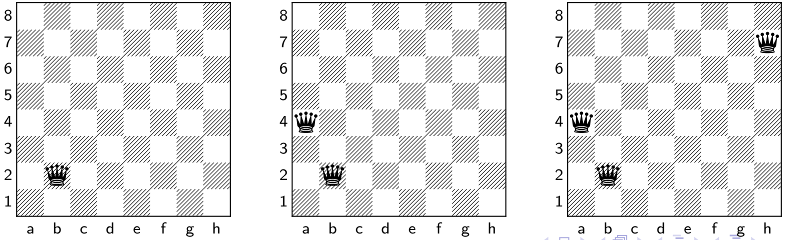
Importance du choix de représentation

Choix de Graphe d'états

Reprenons le problèmes des 8 reines sur un échiquier 8×8

Si 1 état = 1 case

- $posReine_i \in [1, 64]$ donne $1,78.10^{14}$ états possibles
  $64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57 = 64! - 56!$
- Action/Opérateur : placerReine($reine_i$, $ligne_j$, $colonne_k$)
- **Test** : validationFinale(*echiquier*)



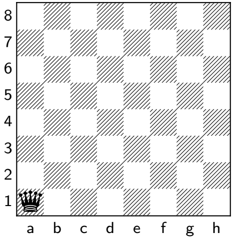
Importance du choix de représentation

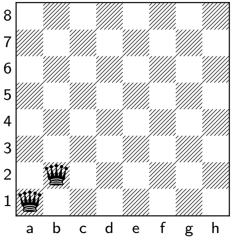
Choix de Graphe d'états

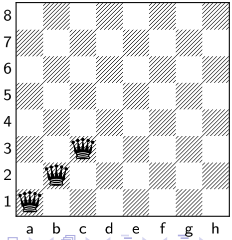
Reprenons le problèmes des 8 reines sur un échiquier 8×8

Si 1 état = 1 position dans une ligne

- $\mathit{ligneReine}_i \in [1, 8]$ donne 40320 états possibles : $8 \times 7 \times 6 \times \cdots \times 1 = 8!$
- Action/Opérateur : placerReineDansLigne(reine_i, ligne_j)
- **Test** : validationFinale(*echiquier*)







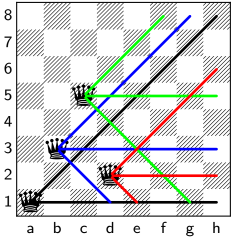
Importance du choix de l'action

Choix de l'action

Sur le problèmes des 8 reines sur un échiquier 8×8

Avec 1 état = 1 position dans une ligne

- Action/Opérateur : placerReineDansLigne**Libre**($reine_i$, $ligne_j$)
- **Test** : placementImpossible($reine_i$), toutesReinesPlacees()



Evaluation de la recherche

Critères d'évaluation

Complétude : si une solution existe, la méthode de recherche la trouve toujours

Complexité en temps : temps nécessaire pour trouver la solution

Complexité en espace : espace mémoire nécessaire pour effectuer la recherche

Optimalité : la solution retenue est la meilleure

Evaluation de la recherche

Critères d'évaluation

Complétude : si une solution existe, la méthode de recherche la trouve toujours

Complexité en temps : temps nécessaire pour trouver la solution

Complexité en espace : espace mémoire nécessaire pour effectuer la recherche

Optimalité : la solution retenue est la meilleure

Evaluation de la recherche

Critères d'évaluation

Complétude : si une solution existe, la méthode de recherche la trouve toujours

Complexité en temps : temps nécessaire pour trouver la solution

Complexité en espace : espace mémoire nécessaire pour effectuer la recherche

Optimalité : la solution retenue est la meilleure

Evaluation de la recherche

Critères d'évaluation

- **Complétude** : si une solution existe, la méthode de recherche la trouve toujours
- **Complexité en temps** : temps nécessaire pour trouver la solution
- **Complexité en espace** : espace mémoire nécessaire pour effectuer la recherche
- **Optimalité** : la solution retenue est la meilleure

Evaluation de la recherche

Critères d'évaluation

- **Complétude** : si une solution existe, la méthode de recherche la trouve toujours
- **Complexité en temps** : temps nécessaire pour trouver la solution
- **Complexité en espace** : espace mémoire nécessaire pour effectuer la recherche
- **Optimalité** : la solution retenue est la meilleure

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.
  - Remarque : il est possible d'avoir $m = \infty$

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.
  - Remarque : il est possible d'avoir $m = \infty$

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.
    - Remarque : il est possible d'avoir $m = \infty$

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.
  - Remarque : il est possible d'avoir $m = \infty$

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.
  - Remarque : il est possible d'avoir $m = \infty$

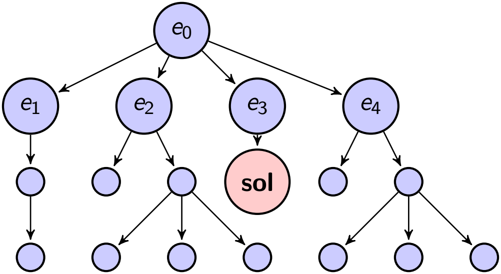
Ici, $b = 4$, $d = 2$ et $m = 3$

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.
    - Remarque : il est possible d'avoir $m = \infty$



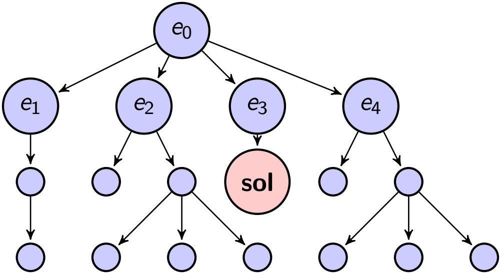
Ici, $b = 4$, $d = 2$ et $m = 3$

Analyse de l'arbre de recherche

Complexité

Les complexités en temps et en espace dépendent de :

- b = facteur de branchement maximum de l'arbre de recherche,
- d = profondeur à laquelle se trouve le (meilleur) nœud-solution,
- m = profondeur maximum de l'espace de recherche.
  - Remarque : il est possible d'avoir $m = \infty$



Ici, $b = 4$, $d = 2$ et $m = 3$

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendamment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul

Ordre général de croissance de l'arbre de recherche

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendamment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendamment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendamment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

- Le temps de calcul $T(n)$ est fonction du taux de croissance de l'arbre de recherche

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendemment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

- Le temps de calcul $T(n)$ est fonction du taux de croissance de l'arbre de recherche

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendamment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

- Le temps de calcul $T(n)$ est fonction du taux de croissance de l'arbre de recherche
- ex : pour un arbre doublant ses branches à chaque étape, $T(n) = O(2^n)$
- ex : pour une recherche dichotomique, $T(n) = O(\log(n))$

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendemment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

- Le temps de calcul $T(n)$ est fonction du taux de croissance de l'arbre de recherche
- ex : pour un arbre doublant ses branches à chaque étape, $T(n) = O(2^n)$
- ex : pour une recherche dichotomique, $T(n) = O(log_2(n))$

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendamment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

- Le temps de calcul $T(n)$ est fonction du taux de croissance de l'arbre de recherche
- ex : pour un arbre doublant ses branches à chaque étape, $T(n) = O(2^n)$
- ex : pour une recherche dichotomique, $T(n) = O(log_2(n))$

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendamment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

- Le temps de calcul $T(n)$ est fonction du taux de croissance de l'arbre de recherche
- ex : pour un arbre doublant ses branches à chaque étape, $T(n) = O(2^n)$
- ex : pour une recherche dichotomique, $T(n) = O(log_2(n))$

Evaluation de la recherche

Questions sur la complexité

Comment évaluer le temps indépendemment de la machine ?

- On note $T(n)$ le temps de calcul d'un algorithme en fonction de la taille des données d'entrées.
- utiliser de préférence :
  - l'ordre de grandeur du temps de calcul
  - son évolution en fonction des données entrées
  - le pire cas

Ordre général de croissance de l'arbre de recherche

- Le temps de calcul $T(n)$ est fonction du taux de croissance de l'arbre de recherche
- ex : pour un arbre doublant ses branches à chaque étape, $T(n) = O(2^n)$
- ex : pour une recherche dichotomique, $T(n) = O(log_2(n))$

Evolution du temps de recherche

On suppose 1 action $= 1\mu s = 10^{-6} s$.

$T(n) \mid n$	10	20	30	40	50	60
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$
$\log n$	$2.3\mu s$	$3\mu s$	$3.4\mu s$	$3.7\mu s$	$3.9\mu s$	$4.1\mu s$
$\log_2 n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
$n \log n$	$23\mu s$	$60\mu s$	$102\mu s$	$147\mu s$	$195\mu s$	$245\mu s$
n^2	$100\mu s$	$400\mu s$	$900\mu s$	1.6 ms	2.5 ms	3.6 ms
n^3	1 ms	8 ms	27 ms	64 ms	125 ms	216 ms
2^n	1 ms	1 s	18 mn	13 jours	36 ans	366 siècles

Pour un algo de complexité 2^n, comme celui résolvant les “Tours de Hanoï”,
si le déplacement d'un disque coûte $1\mu s$,
cela nécessite 13 jours pour déplacer la pile de 40 disques !

Type de recherches

Type de recherches

- Méthodes de recherche aveugles, sans utilisation de connaissances sur le problème :
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche informées (heuristiques)

Type de recherches

Type de recherches

- Méthodes de recherche aveugles, sans utilisation de connaissances sur le problème :
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche informées (heuristiques)

Type de recherches

Type de recherches

- Méthodes de recherche aveugles, sans utilisation de connaissances sur le problème :
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche informées (heuristiques)

Type de recherches

Type de recherches

- Méthodes de recherche aveugles, sans utilisation de connaissances sur le problème :
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche informées (heuristiques)

Type de recherches

Type de recherches

- Méthodes de recherche aveugles, sans utilisation de connaissances sur le problème :
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche informées (heuristiques)

Type de recherches

Type de recherches

- Méthodes de recherche aveugles, sans utilisation de connaissances sur le problème :
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche informées (heuristiques)

Type de recherches

Type de recherches

- Méthodes de recherche aveugles, sans utilisation de connaissances sur le problème :
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche informées (heuristiques)

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,

- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,

- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des nœuds libres
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1^{er} nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1^{er} nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1^{er} nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1^{er} nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1er nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1^{er} nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1er nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

Principe de la résolution

- La trame générale de résolution repose sur
  - des *nœuds libres* : nœuds à partir desquels des actions sont réalisables
    nœuds dont on peut générer des nœuds fils potentiels
  - des *nœuds clos* : nœuds déjà visités et dont on ne peut plus générer de descendance
- Le principe est simple,
  1. placer le nœud initial dans la liste des *nœuds libres*
  2. Tant qu'il existe un nœud dans *nœuds libres* ou que le but n'est pas atteint
  3. retirer le 1er nœud de *nœuds libres*
  4. le placer dans la liste *nœuds clos*
  5. générer ses descendants et ajouter aux *nœuds libres* ceux qui sont nouveaux (ni déjà présents en libres ou en clos)
- *L'ajout se fait en fin de liste ou en début de liste selon la méthode choisie*

Algorithme de résolution générique

```
procedure RECHERCHE
    noeudsLibres ← etatInitial
    noeudsClos ← ∅
    succes ← faux
    while noeudsLibres ≠ ∅ ∧ (non(succes)) do
        n ← PRENDREPREMIER(noeudsLibres)
        if ESTFINAL(n) then
            succes ← vrai
        else
            noeudsLibres ← noeudsLibres − {n}
            noeudsClos ← noeudsClos ∪ {n}
            for all s ∈ n.successeurs do
                if s ∉ noeudsLibres ∨ s ∉ noeudsClos then
                    AJOUTERNOEUDLIBRE(noeudsLibres, s)
                    s.pere ← n
                end if
            end for
        end if
    end while
    if succes then
        AFFICHERCHEMIN(n)
    end if
end procedure
```

Recherche en largeur

Recherche en largeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en fin de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue niveau par niveau
- *Complétude* : **Oui** (si b (branchement) est fini)
- *Complexité* : $1 + b + b \times b + \cdots + b^d \rightarrow$ en $O(b^d)$
- *Optimalité* : **Oui** le chemin trouvé de l'état initial à l'état final est le plus court

Recherche en largeur

Recherche en largeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en fin de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue niveau par niveau
- *Complétude* : **Oui** (si b (branchement) est fini)
- *Complexité* : $1 + b + b \times b + \cdots + b^d \rightarrow$ en $O(b^d)$
- *Optimalité* : **Oui** le chemin trouvé de l'état initial à l'état final est le plus court

Recherche en largeur

Recherche en largeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en fin de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue niveau par niveau
- *Complétude* : **Oui** (si b (branchement) est fini)
- *Complexité* : $1 + b + b \times b + \cdots + b^d \rightarrow$ en $O(b^d)$
- *Optimalité* : **Oui** le chemin trouvé de l'état initial à l'état final est le plus court

Recherche en largeur

Recherche en largeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en fin de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue niveau par niveau
- *Complétude* : **Oui** (si b (branchement) est fini)
- *Complexité* : $1 + b + b \times b + \cdots + b^d \rightarrow$ en $O(b^d)$
- *Optimalité* : **Oui** le chemin trouvé de l'état initial à l'état final est le plus court

Recherche en largeur

Recherche en largeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en fin de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue niveau par niveau
- *Complétude* : **Oui** (si b (branchement) est fini)
- *Complexité* : $1 + b + b \times b + \cdots + b^d \rightarrow$ en $O(b^d)$
- *Optimalité* : **Oui** le chemin trouvé de l'état initial à l'état final est le plus court

Recherche en largeur

Recherche en largeur

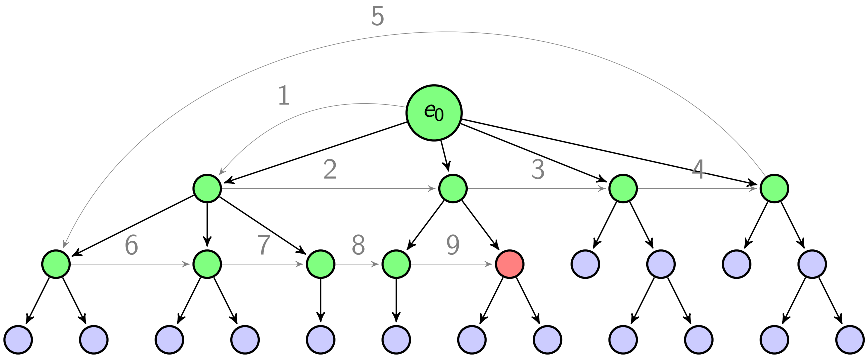
- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en fin de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue niveau par niveau
- *Complétude* : **Oui** (si b (branchement) est fini)
- *Complexité* : $1 + b + b \times b + \cdots + b^d \rightarrow$ en $O(b^d)$
- *Optimalité* : **Oui** le chemin trouvé de l'état initial à l'état final est le plus court

Recherche en largeur

Recherche en largeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en fin de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue niveau par niveau
- *Complétude* : **Oui** (si b (branchement) est fini)
- *Complexité* : $1 + b + b \times b + \cdots + b^d \rightarrow$ en $O(b^d)$
- *Optimalité* : **Oui** le chemin trouvé de l'état initial à l'état final est le plus court

Recherche en largeur : Exemple d'arbre de recherche



Recherche en profondeur

Recherche en profondeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au bout de l'arbre
- *Complétude* : Oui (*si espaces d'états finis* et acycliques)
- *Complexité* : en $O(b^m)$
- *Optimalité* : **non**

Recherche en profondeur

Recherche en profondeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au bout de l'arbre
- *Complétude* : Oui (*si espaces d'états finis* et acycliques)
- *Complexité* : en $O(b^m)$
- *Optimalité* : **non**

Recherche en profondeur

Recherche en profondeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au bout de l'arbre
- *Complétude* : Oui (*si espaces d'états finis* et acycliques)
- *Complexité* : en $O(b^m)$
- *Optimalité* : **non**

Recherche en profondeur

Recherche en profondeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au bout de l'arbre
- *Complétude* : Oui (*si espaces d'états finis* et acycliques)
- *Complexité* : en $O(b^m)$
- *Optimalité* : **non**

Recherche en profondeur

Recherche en profondeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au bout de l'arbre
- *Complétude* : Oui (*si espaces d'états finis* et acycliques)
- *Complexité* : en $O(b^m)$
- *Optimalité* : **non**

Recherche en profondeur

Recherche en profondeur

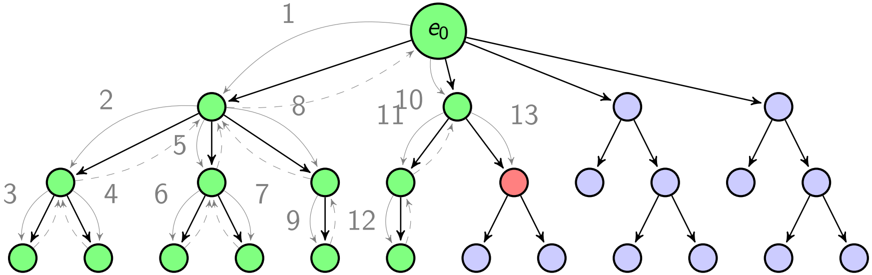
- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au bout de l'arbre
- *Complétude* : Oui (*si espaces d'états finis* et acycliques)
- *Complexité* : en $O(b^m)$
- *Optimalité* : **non**

Recherche en profondeur

Recherche en profondeur

- Stratégie :
  - prendre un nœud, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au bout de l'arbre
- *Complétude* : Oui (*si espaces d'états finis* et acycliques)
- *Complexité* : en $O(b^m)$
- *Optimalité* : **non**

Recherche en profondeur : Exemple d'arbre de recherche



Recherche en profondeur limitée

Recherche en profondeur limitée

- Stratégie :
  - prendre un nœud, si son niveau est $< L$, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au niveau L
- *Complétude* : Non (solution trouvée ssi $L \leq d$)
- *Complexité* : en $O(b^L)$
- *Optimalité* : **non**

Recherche en profondeur limitée

Recherche en profondeur limitée

- Stratégie :
  - prendre un nœud, si son niveau est $< L$, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au niveau L
- *Complétude* : Non (solution trouvée ssi $L \leq d$)
- *Complexité* : en $O(b^L)$
- *Optimalité* : **non**

Recherche en profondeur limitée

Recherche en profondeur limitée

- Stratégie :
  - prendre un nœud, si son niveau est $< L$, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au niveau L
- *Complétude* : Non (solution trouvée ssi $L \leq d$)
- *Complexité* : en $O(b^L)$
- *Optimalité* : **non**

Recherche en profondeur limitée

Recherche en profondeur limitée

- Stratégie :
  - prendre un nœud, si son niveau est $< L$, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au niveau L
- *Complétude* : Non (solution trouvée ssi $L \leq d$)
- *Complexité* : en $O(b^L)$
- *Optimalité* : **non**

Recherche en profondeur limitée

Recherche en profondeur limitée

- Stratégie :
  - prendre un nœud, si son niveau est $< L$, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au niveau L
- *Complétude* : Non (solution trouvée ssi $L \leq d$)
- *Complexité* : en $O(b^L)$
- *Optimalité* : **non**

Recherche en profondeur limitée

Recherche en profondeur limitée

- Stratégie :
  - prendre un nœud, si son niveau est $< L$, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au niveau L
- *Complétude* : Non (solution trouvée ssi $L \leq d$)
- *Complexité* : en $O(b^L)$
- *Optimalité* : **non**

Recherche en profondeur limitée

Recherche en profondeur limitée

- Stratégie :
  - prendre un nœud, si son niveau est $< L$, en générer les fils et les ajouter en tête de la liste des nœuds libres
  - ainsi le parcours de l'arbre s'effectue en profondeur jusqu'au niveau L
- *Complétude* : Non (solution trouvée ssi $L \leq d$)
- *Complexité* : en $O(b^L)$
- *Optimalité* : **non**

Recherche par approfondissement itératif

Recherche par approfondissement itératif

- Stratégie :
  - recherche en profondeur limité en incrémentant L dès que la profondeur L est atteinte
  - combine donc recherche en largeur et recherche en profondeur
- *Complétude* : **Oui**
- *Complexité* : en $O(b^d)$
- *Optimalité* : oui (si $L \leq d$)

Recherche par approfondissement itératif

Recherche par approfondissement itératif

- Stratégie :
  - recherche en profondeur limité en incrémentant L dès que la profondeur L est atteinte
  - combine donc recherche en largeur et recherche en profondeur
- *Complétude* : **Oui**
- *Complexité* : en $O(b^d)$
- *Optimalité* : oui (si $L \leq d$)

Recherche par approfondissement itératif

Recherche par approfondissement itératif

- Stratégie :
  - recherche en profondeur limité en incrémentant L dès que la profondeur L est atteinte
  - combine donc recherche en largeur et recherche en profondeur
- *Complétude* : **Oui**
- *Complexité* : en $O(b^d)$
- *Optimalité* : oui (si $L \leq d$)

Recherche par approfondissement itératif

Recherche par approfondissement itératif

- Stratégie :
  - recherche en profondeur limité en incrémentant L dès que la profondeur L est atteinte
  - combine donc recherche en largeur et recherche en profondeur
- *Complétude* : **Oui**
- *Complexité* : en $O(b^d)$
- *Optimalité* : oui (si $L \leq d$)

Recherche par approfondissement itératif

Recherche par approfondissement itératif

- Stratégie :
  - recherche en profondeur limité en incrémentant L dès que la profondeur L est atteinte
  - combine donc recherche en largeur et recherche en profondeur
- *Complétude* : **Oui**
- *Complexité* : en $O(b^d)$
- *Optimalité* : oui (si $L \leq d$)

Recherche par approfondissement itératif

Recherche par approfondissement itératif

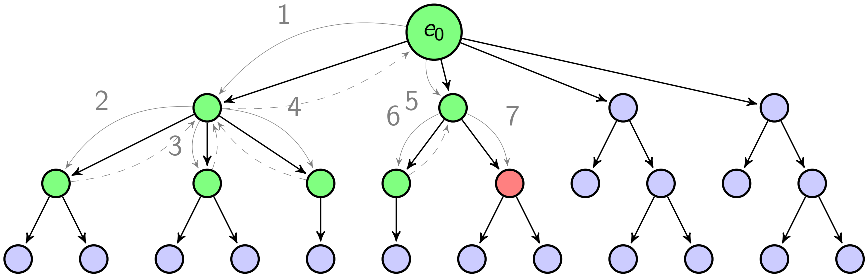
- Stratégie :
  - recherche en profondeur limité en incrémentant L dès que la profondeur L est atteinte
  - combine donc recherche en largeur et recherche en profondeur
- *Complétude* : **Oui**
- *Complexité* : en $O(b^d)$
- *Optimalité* : oui (si $L \leq d$)

Recherche par approfondissement itératif

Recherche par approfondissement itératif

- Stratégie :
  - recherche en profondeur limité en incrémentant L dès que la profondeur L est atteinte
  - combine donc recherche en largeur et recherche en profondeur
- *Complétude* : **Oui**
- *Complexité* : en $O(b^d)$
- *Optimalité* : oui (si $L \leq d$)

Recherche par approfondissement itératif : Exemple d'arbre de recherche avec L=2



Heuristiques

Notion d'heuristiques

- Méthodes en aveugle trop gourmandes en mémoire et/ou en temps
- Une solution : orienter la recherche par une information heuristique
- Une *heuristique* doit guider le choix des états à tester et les ordonner selon leurs 'promesses de rapprocher d'un but'.

Heuristiques

Notion d'heuristiques

- Méthodes en aveugle trop gourmandes en mémoire et/ou en temps
- Une solution : orienter la recherche par une information heuristique
- Une *heuristique* doit guider le choix des états à tester et les ordonner selon leurs 'promesses de rapprocher d'un but'.
  - Une heuristique dépend fortement du problème à traiter

Heuristiques

Notion d'heuristiques

- Méthodes en aveugle trop gourmandes en mémoire et/ou en temps
- Une solution : orienter la recherche par une information heuristique
- Une *heuristique* doit guider le choix des états à tester et les ordonner selon leurs 'promesses de rapprocher d'un but'.
  - Une heuristique dépend fortement du problème à traiter
  - Une heuristique *pauvre* basée sur des propriétés trop simples du problème sera peu efficace,
  - Une heuristique *riche* basée sur des propriétés approfondies du problème sera efficace, mais est difficile à établir.

Heuristiques

Notion d'heuristiques

- Méthodes en aveugle trop gourmandes en mémoire et/ou en temps
- Une solution : orienter la recherche par une information heuristique
- Une *heuristique* doit guider le choix des états à tester et les ordonner selon leurs 'promesses de rapprocher d'un but'.
  - Une heuristique dépend fortement du problème à traiter
  - Une heuristique *pauvre* basée sur des propriétés trop simples du problème sera peu efficace,
  - Une heuristique *riche* basée sur des propriétés approfondies du problème sera efficace, mais est difficile à établir.

Heuristiques

Notion d'heuristiques

- Méthodes en aveugle trop gourmandes en mémoire et/ou en temps
- Une solution : orienter la recherche par une information heuristique
- Une *heuristique* doit guider le choix des états à tester et les ordonner selon leurs 'promesses de rapprocher d'un but'.
  - Une heuristique dépend fortement du problème à traiter
  - Une heuristique *pauvre* basée sur des propriétés trop simples du problème sera peu efficace,
  - Une heuristique *riche* basée sur des propriétés approfondies du problème sera efficace, mais est difficile à établir.

Heuristiques

Notion d'heuristiques

- Méthodes en aveugle trop gourmandes en mémoire et/ou en temps
- Une solution : orienter la recherche par une information heuristique
- Une *heuristique* doit guider le choix des états à tester et les ordonner selon leurs 'promesses de rapprocher d'un but'.
  - Une heuristique dépend fortement du problème à traiter
  - Une heuristique *pauvre* basée sur des propriétés trop simples du problème sera peu efficace,
  - Une heuristique *riche* basée sur des propriétés approfondies du problème sera efficace, mais est difficile à établir.

Heuristiques

Notion d'heuristiques

- Méthodes en aveugle trop gourmandes en mémoire et/ou en temps
- Une solution : orienter la recherche par une information heuristique
- Une *heuristique* doit guider le choix des états à tester et les ordonner selon leurs 'promesses de rapprocher d'un but'.
  - Une heuristique dépend fortement du problème à traiter
  - Une heuristique *pauvre* basée sur des propriétés trop simples du problème sera peu efficace,
  - Une heuristique *riche* basée sur des propriétés approfondies du problème sera efficace, mais est difficile à établir.

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :
  - Transformer un nb rationnel en somme de nb rationnels de la forme 1/n (fractions unitaires, égyptiennes)

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :
  - Transformer un nb rationnel en somme de nb rationnels de la forme 1/n (fractions unitaires, égyptiennes)
  - Maximiser le nb d'activités réalisables en un temps donné
  - Rendre la monnaie en minimisant le nb de pièces/billets

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :
  - Transformer un nb rationnel en somme de nb rationnels de la forme 1/n (fractions unitaires, égyptiennes)
  - Maximiser le nb d'activités réalisables en un temps donné
  - Rendre la monnaie en minimisant le nb de pièces/billets

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :
  - Transformer un nb rationnel en somme de nb rationnels de la forme 1/n (fractions unitaires, égyptiennes)
  - Maximiser le nb d'activités réalisables en un temps donné
  - Rendre la monnaie en minimisant le nb de pièces/billets

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :
  - Transformer un nb rationnel en somme de nb rationnels de la forme 1/n (fractions unitaires, égyptiennes)
  - Maximiser le nb d'activités réalisables en un temps donné
  - Rendre la monnaie en minimisant le nb de pièces/billets

Algorithme glouton (*greedy*)

Avance par gourmandise

- A chaque nœud, on choisit le nœud suivant offrant le meilleur gain
- Il faut donc pouvoir évaluer les nœuds
- Algorithme optimal, *dans certains cas*
- Exemples classiques :
  - Transformer un nb rationnel en somme de nb rationnels de la forme 1/n (fractions unitaires, égyptiennes)
  - Maximiser le nb d'activités réalisables en un temps donné
  - Rendre la monnaie en minimisant le nb de pièces/billets

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
- Exemple : transformer $n = \left(\frac{14}{15}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$
  - $\frac{1}{2} \leq \frac{14}{15}$, on note $\frac{1}{2}$ et $n \leftarrow \left(\frac{14}{15} - \frac{1}{2}\right) = \frac{13}{30}$
  - $\frac{1}{3} \leq \frac{13}{30}$, on note $\frac{1}{3}$ et $n \leftarrow \left(\frac{13}{30} - \frac{1}{3}\right) = \frac{1}{10}$
  - La suite est évidente, $\frac{1}{4} \ldots \frac{1}{10}$ sont testées
  - Au final, il est écrit que $\frac{14}{15} = \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{10}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$
  - $\frac{1}{2} \leq \frac{14}{15}$, on note $\frac{1}{2}$ et $n \leftarrow \left(\frac{14}{15} - \frac{1}{2}\right) = \frac{13}{30}$
  - $\frac{1}{3} \leq \frac{13}{30}$, on note $\frac{1}{3}$ et $n \leftarrow \left(\frac{13}{30} - \frac{1}{3}\right) = \frac{1}{10}$
  - La suite est évidente, $\frac{1}{4}, \ldots, \frac{1}{10}$ sont testées
  - Au final, il est écrit que $\frac{14}{15} = \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{10}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$
  - $\frac{1}{2} \leq \frac{14}{15}$, on note $\frac{1}{2}$ et $n \leftarrow \left(\frac{14}{15} - \frac{1}{2}\right) = \frac{13}{30}$
  - $\frac{1}{3} \leq \frac{13}{30}$, on note $\frac{1}{3}$ et $n \leftarrow \left(\frac{13}{30} - \frac{1}{3}\right) = \frac{1}{10}$
  - La suite est évidente, $\frac{1}{4}, \ldots, \frac{1}{10}$ sont testées
  - Au final, il est écrit que $\frac{14}{15} = \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{10}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$
  - $\frac{1}{2} \leq \frac{14}{15}$, on note $\frac{1}{2}$ et $n \leftarrow \left(\frac{14}{15} - \frac{1}{2}\right) = \frac{13}{30}$
  - $\frac{1}{3} \leq \frac{13}{30}$, on note $\frac{1}{3}$ et $n \leftarrow \left(\frac{13}{30} - \frac{1}{3}\right) = \frac{1}{10}$
  - La suite est évidente, $\frac{1}{4}, \ldots, \frac{1}{10}$ sont testées
  - Au final, il est écrit que $\frac{14}{15} = \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{10}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$
  - $\frac{1}{2} \leq \frac{14}{15}$, on note $\frac{1}{2}$ et $n \leftarrow \left(\frac{14}{15} - \frac{1}{2}\right) = \frac{13}{30}$
  - $\frac{1}{3} \leq \frac{13}{30}$, on note $\frac{1}{3}$ et $n \leftarrow \left(\frac{13}{30} - \frac{1}{3}\right) = \frac{1}{10}$
  - La suite est évidente, $\frac{1}{4}, \ldots, \frac{1}{10}$ sont testées
  - Au final, il est écrit que $\frac{14}{15} = \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{10}\right)$

Algorithme glouton (*greedy*)

Fractions égyptiennes

- *Objectif* : $n = \left(\frac{a}{b}\right) = \sum_i \left(\frac{1}{d_i}\right)$
- *Objectif transformé* : $n - \sum_i \left(\frac{1}{d_i}\right) = 0$
- *Algorithme glouton* :
  - *meilleur choix* = plus grande fraction unitaire $\leq n$
  - retirer de n cette fraction,
  - répéter jusqu'à $n = 0$
- Exemple : transformer $n = \left(\frac{14}{15}\right)$
  - $\frac{1}{2} \leq \frac{14}{15}$, on note $\frac{1}{2}$ et $n \leftarrow \left(\frac{14}{15} - \frac{1}{2}\right) = \frac{13}{30}$
  - $\frac{1}{3} \leq \frac{13}{30}$, on note $\frac{1}{3}$ et $n \leftarrow \left(\frac{13}{30} - \frac{1}{3}\right) = \frac{1}{10}$
  - La suite est évidente, $\frac{1}{4}, \ldots, \frac{1}{10}$ sont testées
  - Au final, il est écrit que $\frac{14}{15} = \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{10}\right)$

Algorithme glouton (*greedy*)

Maximiser le nb d'actes

- *Objectif* : A partir d'une liste d'actes de dates et de durées variables,
  maximiser le nb d'actes réalisables sans chevauchement
- *Algorithme glouton* :
  - trier les actes par date de départ
- Exemple : à partir de l'ensemble d'actes initial, 3 actes sont réalisables sans chevauchement

Algorithme glouton (*greedy*)

Maximiser le nb d'actes

- *Objectif* : A partir d'une liste d'actes de dates et de durées variables,
  maximiser le nb d'actes réalisables sans chevauchement
- *Algorithme glouton* :
  - trier les actes par date de départ
  - *meilleur choix* = prendre le plus court réalisable au plus tôt
- Exemple : à partir de l'ensemble d'actes initial, 3 actes sont réalisables sans chevauchement

Algorithme glouton (*greedy*)

Maximiser le nb d'actes

- *Objectif* : A partir d'une liste d'actes de dates et de durées variables,
  maximiser le nb d'actes réalisables sans chevauchement
- *Algorithme glouton* :
  - trier les actes par date de départ
  - *meilleur choix* = prendre le plus court réalisable au plus tôt
- Exemple : à partir de l'ensemble d'actes initial, 3 actes sont réalisables sans chevauchement

Algorithme glouton (*greedy*)

Maximiser le nb d'actes

- *Objectif* : A partir d'une liste d'actes de dates et de durées variables,
  maximiser le nb d'actes réalisables sans chevauchement
- *Algorithme glouton* :
  - trier les actes par date de départ
  - *meilleur choix* = prendre le plus court réalisable au plus tôt
- Exemple : à partir de l'ensemble d'actes initial, 3 actes sont réalisables sans chevauchement

Algorithme glouton (*greedy*)

Maximiser le nb d'actes

- *Objectif* : A partir d'une liste d'actes de dates et de durées variables,
  maximiser le nb d'actes réalisables sans chevauchement
- *Algorithme glouton* :
  - trier les actes par date de départ
  - *meilleur choix* = prendre le plus court réalisable au plus tôt
- Exemple : à partir de l'ensemble d'actes initial, 3 actes sont réalisables sans chevauchement

Algorithme glouton (*greedy*)

Maximiser le nb d'actes

- *Objectif* : A partir d'une liste d'actes de dates et de durées variables,
  maximiser le nb d'actes réalisables sans chevauchement
- *Algorithme glouton* :
  - trier les actes par date de départ
  - *meilleur choix* = prendre le plus court réalisable au plus tôt
- Exemple : à partir de l'ensemble d'actes initial, 3 actes sont réalisables sans chevauchement

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
    - retirer de s les pièces les plus élevées
- Exemple :
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i$, $p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :
  - si $p_i \in \{4, 3, 1\}$ et $s = 6$,

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :
  - si $p_i \in \{4, 3, 1\}$ et $s = 6$,
  - l'algo donne $6 = 1 \times (4) + 2 \times (1)$ donc 3 pièces
  - mais l'optimal est $6 = 2 \times (3)$ donc 2 pièces

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :
  - si $p_i \in \{4, 3, 1\}$ et $s = 6$,
  - l'algo donne $6 = 1 \times (4) + 2 \times (1)$ donc 3 pièces
  - mais l'optimal est $6 = 2 \times (3)$ donc **2 pièces**

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i, p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :
  - si $p_i \in \{4, 3, 1\}$ et $s = 6$,
  - l'algo donne $6 = 1 \times (4) + 2 \times (1)$ donc 3 pièces
  - mais l'optimal est $6 = 2 \times (3)$ donc **2 pièces**

Algorithme glouton (*greedy*)

Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i$, $p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :
  - si $p_i \in \{4, 3, 1\}$ et $s = 6$,
  - l'algo donne $6 = 1 \times (4) + 2 \times (1)$ donc 3 pièces
  - mais l'optimal est $6 = 2 \times (3)$ donc **2 pièces**

Algorithme glouton (*greedy*)

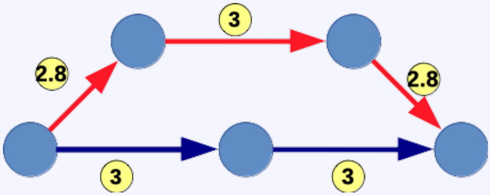
Non optimal avec le rendu de monnaie

- *Objectif* : donner le minimum de pièces/billets correspondant à une somme $s = \sum_i p_i$, $p_i \in \{10, 5, 2, 1\}$
- *Algorithme glouton* : $s - \sum_i p_i = 0$
  - retirer de s les pièces les plus élevées
- Exemple :
  - $36 = 3 \times (10) + 1 \times (5) + 1 \times (1)$, optimal de 5 pièces atteint
- *MAIS algo non optimal selon les pièces/billets possibles*
- Exemple :
  - si $p_i \in \{4, 3, 1\}$ et $s = 6$,
  - l'algo donne $6 = 1 \times (4) + 2 \times (1)$ donc 3 pièces
  - mais l'optimal est $6 = 2 \times (3)$ donc **2 pièces**

Algorithme glouton (*greedy*)

Non optimal avec les graphes

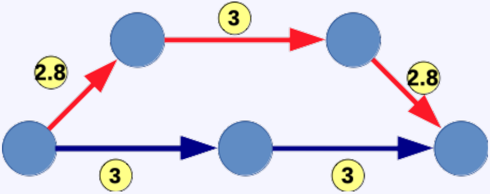
- *Objectif* : trouver le chemin le plus court
- *Algorithme glouton* : à partir d'un point donné, prendre l'arc le plus court
    - La figure montre un exemple simple la non optimalité de l'algorithme glouton pour ce cas



Algorithme glouton (*greedy*)

Non optimal avec les graphes

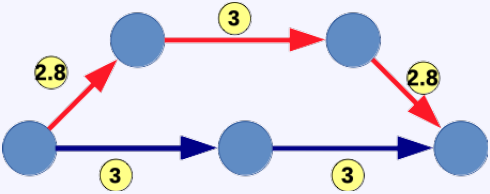
- *Objectif* : trouver le chemin le plus court
- *Algorithme glouton* : à partir d'un point donné, prendre l'arc le plus court
  - La figure montre un exemple simple la non optimalité de l'algorithme glouton pour ce cas
  - le plus court chemin a une longueur de 6 unités
  - l'algo glouton trouve une longueur de 8.6 unités !



Algorithme glouton (*greedy*)

Non optimal avec les graphes

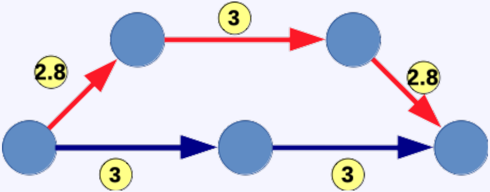
- *Objectif* : trouver le chemin le plus court
- *Algorithme glouton* : à partir d'un point donné, prendre l'arc le plus court
  - La figure montre un exemple simple la non optimalité de l'algorithme glouton pour ce cas
  - le plus court chemin a une longueur de 6 unités
  - l'algo glouton trouve une longueur de 8.6 unités !



Algorithme glouton (*greedy*)

Non optimal avec les graphes

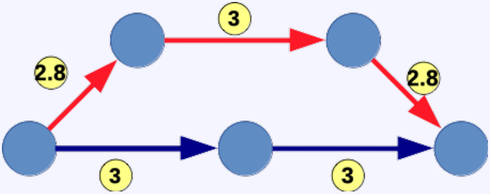
- *Objectif* : trouver le chemin le plus court
- *Algorithme glouton* : à partir d'un point donné, prendre l'arc le plus court
    - La figure montre un exemple simple la non optimalité de l'algorithme glouton pour ce cas
    - le plus court chemin a une longueur de 6 unités
    - l'algo glouton trouve une longueur de 8.6 unités !



Algorithme glouton (*greedy*)

Non optimal avec les graphes

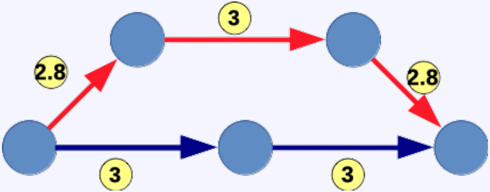
- *Objectif* : trouver le chemin le plus court
- *Algorithme glouton* : à partir d'un point donné, prendre l'arc le plus court
  - La figure montre un exemple simple la non optimalité de l'algorithme glouton pour ce cas
  - le plus court chemin a une longueur de 6 unités
  - l'algo glouton trouve une longueur de 8.6 unités !



Algorithme glouton (*greedy*)

Non optimal avec les graphes

- *Objectif* : trouver le chemin le plus court
- *Algorithme glouton* : à partir d'un point donné, prendre l'arc le plus court
  - La figure montre un exemple simple la non optimalité de l'algorithme glouton pour ce cas
  - le plus court chemin a une longueur de 6 unités
  - l'algo glouton trouve une longueur de 8.6 unités !



Recherche Heuristique par A*

Définitions pour la recherche par heuristique

Soit n un nœud du graphe

- $g^*(n)$ est le coût minimum entre le nœud de départ n_0 et le nœud n
- $h^*(n)$ est le coût minimal des chemins du nœud n à un nœud solution n_s.
- $f^*(n) = g^*(n) + h^*(n)$ est le coût du chemin solution optimal de n_0 à n_s passant par n.

Il est donc nécessaire de définir :

- une heuristique h(n) qui *estime* $h^*(n)$
- g(n) le coût effectif du meilleur chemin connu pour aller de n_0 à n
- pour poser $f(n) = g(n) + h(n)$, la fonction d'évaluation du nœud n

Recherche Heuristique par A*

Définitions pour la recherche par heuristique

Soit n un nœud du graphe

- $g^*(n)$ est le coût minimum entre le nœud de départ n_0 et le nœud n
- $h^*(n)$ est le coût minimal des chemins du nœud n à un nœud solution n_s.
- $f^*(n) = g^*(n) + h^*(n)$ est le coût du chemin solution optimal de n_0 à n_s passant par n.

Il est donc nécessaire de définir :

- une heuristique h(n) qui *estime* $h^*(n)$
- g(n) le coût effectif du meilleur chemin connu pour aller de n_0 à n
- pour poser $f(n) = g(n) + h(n)$, la fonction d'évaluation du nœud n

Recherche Heuristique par A*

Définitions pour la recherche par heuristique

Soit n un nœud du graphe

- $g^*(n)$ est le coût minimum entre le nœud de départ n_0 et le nœud n
- $h^*(n)$ est le coût minimal des chemins du nœud n à un nœud solution n_s.
- $f^*(n) = g^*(n) + h^*(n)$ est le coût du chemin solution optimal de n_0 à n_s passant par n.

Il est donc nécessaire de définir :

- une heuristique h(n) qui *estime* $h^*(n)$
- g(n) le coût effectif du meilleur chemin connu pour aller de n_0 à n
- pour poser $f(n) = g(n) + h(n)$, la fonction d'évaluation du nœud n

Recherche Heuristique par A*

Définitions pour la recherche par heuristique

Soit n un nœud du graphe

- $g^*(n)$ est le coût minimum entre le nœud de départ n_0 et le nœud n
- $h^*(n)$ est le coût minimal des chemins du nœud n à un nœud solution n_s.
- $f^*(n) = g^*(n) + h^*(n)$ est le coût du chemin solution optimal de n_0 à n_s passant par n.

Il est donc nécessaire de définir :

- une heuristique h(n) qui *estime* $h^*(n)$
- g(n) le coût effectif du meilleur chemin connu pour aller de n_0 à n
- pour poser $f(n) = g(n) + h(n)$, la fonction d'évaluation du nœud n

Recherche Heuristique par A*

Définitions pour la recherche par heuristique

Soit n un nœud du graphe

- $g^*(n)$ est le coût minimum entre le nœud de départ n_0 et le nœud n
- $h^*(n)$ est le coût minimal des chemins du nœud n à un nœud solution n_s.
- $f^*(n) = g^*(n) + h^*(n)$ est le coût du chemin solution optimal de n_0 à n_s passant par n.

Il est donc nécessaire de définir :

- une heuristique h(n) qui *estime* $h^*(n)$
- g(n) le coût effectif du meilleur chemin connu pour aller de n_0 à n
- pour poser $f(n) = g(n) + h(n)$, la fonction d'évaluation du nœud n

Recherche Heuristique par A*

Définitions pour la recherche par heuristique

Soit n un nœud du graphe

- $g^*(n)$ est le coût minimum entre le nœud de départ n_0 et le nœud n
- $h^*(n)$ est le coût minimal des chemins du nœud n à un nœud solution n_s.
- $f^*(n) = g^*(n) + h^*(n)$ est le coût du chemin solution optimal de n_0 à n_s passant par n.

Il est donc nécessaire de définir :

- une heuristique h(n) qui *estime* $h^*(n)$
- g(n) le coût effectif du meilleur chemin connu pour aller de n_0 à n
- pour poser $f(n) = g(n) + h(n)$, la fonction d'évaluation du nœud n

Recherche Heuristique par A*

Définitions pour la recherche par heuristique

Soit n un nœud du graphe

- $g^*(n)$ est le coût minimum entre le nœud de départ n_0 et le nœud n
- $h^*(n)$ est le coût minimal des chemins du nœud n à un nœud solution n_s.
- $f^*(n) = g^*(n) + h^*(n)$ est le coût du chemin solution optimal de n_0 à n_s passant par n.

Il est donc nécessaire de définir :

- une heuristique h(n) qui *estime* $h^*(n)$
- g(n) le coût effectif du meilleur chemin connu pour aller de n_0 à n
- pour poser $f(n) = g(n) + h(n)$, la fonction d'évaluation du nœud n

Exemple d'algorithme A* I

procedure RECHERCHEA*
  neudsLibres ← *etatInitial*
  noeudsClos ← ∅
  succes ← *faux*
  while *noeudsLibres* ≠ ∅ ∧ (non(succes)) **do**
      ▷ *Choisir n dans les noeudsLibres tel que f(n) est minimum*
    n ← CHOISIRNŒUDHEURISTIQUE(noeudsLibres)
    if ESTFINAL(n) **then**
      succes ← vrai
    else
      noeudsLibres ← *noeudsLibres* − $\{n\}$
      noeudsClos ← *noeudsClos* ∪ $\{n\}$
      for all $s \in n.successeurs$ **do**
        if $s \notin noeudsLibres \vee s \notin noeudsClos$ **then**
          noeudsLibres ← *noeudsLibres* ∪ $\{s\}$
          $s.pere \leftarrow n$
          $g(s) \leftarrow g(n) + cout(n, s)$
        else

Exemple d'algorithme A* II

if $g(s) \geq g(n) + cout(n,s)$ **then**
    $s.pere \leftarrow n$
    $g(s) \leftarrow g(n) + cout(n,s)$
    if $s \in n\sphericalangle udsClos$ **then**
        $noeudsLibres \leftarrow noeudsLibres \cup \{s\}$
        $noeudsClos \leftarrow noeudsClos - \{s\}$
    end if
end if
end if
end for
end if
end while
end procedure

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

- h est dite *minorante* si $\forall n, h(n) \leq h^*(n)$

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

- h est dite *minorante* si $\forall n, h(n) \leq h^*(n)$
- h est dite *monotone* si $\forall n, \forall s \in successeur(n), h(n) - h(s) \leq cout(n, s)$
- si h est *minorante*, alors A^* est *admissible*
- si h est *monotone* alors la première rencontre d'un noeud fournit le meilleur chemin pour y arriver (on est dispensé des mises à jour dans l'algorithme)

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

- h est dite *minorante* si $\forall n, h(n) \leq h^*(n)$
- h est dite *monotone* si $\forall n, \forall s \in successeur(n), h(n) - h(s) \leq cout(n, s)$
- si h est *minorante*, alors A* est *admissible*
- si h est *monotone* alors la première rencontre d'un nœud fournit le meilleur chemin pour y arriver (on est dispensé des mises à jour dans l'algorithme).

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

- h est dite *minorante* si $\forall n, h(n) \leq h^*(n)$
- h est dite *monotone* si $\forall n, \forall s \in successeur(n), h(n) - h(s) \leq cout(n, s)$
- si h est *minorante*, alors A* est *admissible*
- si h est *monotone* alors la première rencontre d'un nœud fournit le meilleur chemin pour y arriver (on est dispensé des mises à jour dans l'algorithme).

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

- h est dite *minorante* si $\forall n, h(n) \leq h^*(n)$
- h est dite *monotone* si $\forall n, \forall s \in successeur(n), h(n) - h(s) \leq cout(n, s)$
- si h est *minorante*, alors A* est *admissible*
- si h est *monotone* alors la première rencontre d'un nœud fournit le meilleur chemin pour y arriver (on est dispensé des mises à jour dans l'algorithme).

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

Propriétés

- h est dite *minorante* si $\forall n, h(n) \leq h^*(n)$
- h est dite *monotone* si $\forall n, \forall s \in successeur(n), h(n) - h(s) \leq cout(n, s)$
- si h est *minorante*, alors A* est *admissible*
- si h est *monotone* alors la première rencontre d'un nœud fournit le meilleur chemin pour y arriver (on est dispensé des mises à jour dans l'algorithme).

Propriétés de la recherche par heuristique

Particularités

- Cas particuliers :
  - si $\forall n, h(n) = 0$, alors f(n)=g(n) ; l'algorithme est équivalent à une recherche en largeur ;
  - si $\forall n, f(n) = h(n)$, l'algorithme est équivalent à une recherche en profondeur

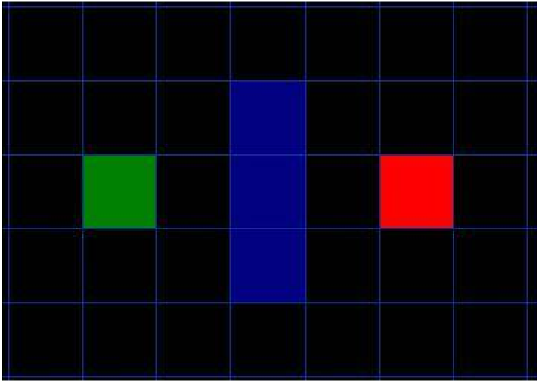
Propriétés

- h est dite *minorante* si $\forall n, h(n) \leq h^*(n)$
- h est dite *monotone* si $\forall n, \forall s \in successeur(n), h(n) - h(s) \leq cout(n, s)$
- si h est *minorante*, alors A* est *admissible*
- si h est *monotone* alors la première rencontre d'un nœud fournit le meilleur chemin pour y arriver (on est dispensé des mises à jour dans l'algorithme).

Recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

But : Se rendre de la case verte à la case rouge en évitant les case bleues



Initialisation de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Analyse du problème

- Définir l'espace d'états
  - Décomposition de la zone en grille, l'espace est constitué des cellules
- Etat initial = cellule verte A en c(1,2)
- Etat but/final = cellule rouge B en c(5,2)
- Branchement = 8 :chaque cellule est liée à ses 8 cellules voisines
- Action = se déplacer d'une case (H, B, D, G, HG, HD, BD, BG) si possible

Initialisation de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Analyse du problème

- Définir l'espace d'états
  - Décomposition de la zone en grille, l'espace est constitué des cellules
- Etat initial = cellule verte A en c(1,2)
- Etat but/final = cellule rouge B en c(5,2)
- Branchement = 8 :chaque cellule est liée à ses 8 cellules voisines
- Action = se déplacer d'une case (H, B, D, G, HG, HD, BD, BG) si possible

Initialisation de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Analyse du problème

- Définir l'espace d'états
  - Décomposition de la zone en grille, l'espace est constitué des cellules
- Etat initial = cellule verte A en c(1,2)
- Etat but/final = cellule rouge B en c(5,2)
- Branchement = 8 :chaque cellule est liée à ses 8 cellules voisines
- Action = se déplacer d'une case (H, B, D, G, HG, HD, BD, BG) si possible

Initialisation de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Analyse du problème

- Définir l'espace d'états
  - Décomposition de la zone en grille, l'espace est constitué des cellules
- Etat initial = cellule verte A en c(1,2)
- Etat but/final = cellule rouge B en c(5,2)
- Branchement = 8 :chaque cellule est liée à ses 8 cellules voisines
- Action = se déplacer d'une case (H, B, D, G, HG, HD, BD, BG) si possible

Initialisation de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Analyse du problème

- Définir l'espace d'états
  - Décomposition de la zone en grille, l'espace est constitué des cellules
- Etat initial = cellule verte A en c(1,2)
- Etat but/final = cellule rouge B en c(5,2)
- Branchement = 8 :chaque cellule est liée à ses 8 cellules voisines
- Action = se déplacer d'une case (H, B, D, G, HG, HD, BD, BG) si possible

Initialisation de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Analyse du problème

- Définir l'espace d'états
  - Décomposition de la zone en grille, l'espace est constitué des cellules
- Etat initial = cellule verte A en c(1,2)
- Etat but/final = cellule rouge B en c(5,2)
- Branchement = 8 :chaque cellule est liée à ses 8 cellules voisines
- Action = se déplacer d'une case (H, B, D, G, HG, HD, BD, BG) si possible

Initialisation de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Analyse du problème

- Définir l'espace d'états
  - Décomposition de la zone en grille, l'espace est constitué des cellules
- Etat initial = cellule verte A en c(1,2)
- Etat but/final = cellule rouge B en c(5,2)
- Branchement = 8 :chaque cellule est liée à ses 8 cellules voisines
- Action = se déplacer d'une case (H, B, D, G, HG, HD, BD, BG) si possible

Coût et Heuristiques de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Définir les coûts et heuristiques

- Définir $f = g + h$
  - $g(C)$: détermine le coût du trajet de A vers C en suivant le chemin généré
  - $h(C)$: coût estimé du mouvement de C vers le but B
- Choix effectué pour

Coût et Heuristiques de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Définir les coûts et heuristiques

- Définir $f = g + h$
  - $g(C)$: détermine le coût du trajet de A vers C en suivant le chemin généré
  - $h(C)$: coût estimé du mouvement de C vers le but B
- Choix effectué pour

Coût et Heuristiques de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Définir les coûts et heuristiques

- Définir $f = g + h$
  - $g(C)$: détermine le coût du trajet de A vers C en suivant le chemin généré
  - $h(C)$: coût estimé du mouvement de C vers le but B
- Choix effectué pour
  - $g(C)$: coût du trajet entre cases adjacentes $= 10$, en diagonale $= 14$ ($int(10 \times \sqrt{2})$)

Coût et Heuristiques de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Définir les coûts et heuristiques

- Définir $f = g + h$
  - $g(C)$: détermine le coût du trajet de A vers C en suivant le chemin généré
  - $h(C)$: coût estimé du mouvement de C vers le but B
- Choix effectué pour
  - $g(C)$: coût du trajet entre cases adjacentes $= 10$; en diagonale $= 14$ ($int(10 \times \sqrt{2})$)
  - $h(C)$: $|x_C - x_B| + |y_C - y_B|$

Coût et Heuristiques de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Définir les coûts et heuristiques

- Définir $f = g + h$
  - $g(C)$: détermine le coût du trajet de A vers C en suivant le chemin généré
  - $h(C)$: coût estimé du mouvement de C vers le but B
- Choix effectué pour
  - $g(C)$: coût du trajet entre cases adjacentes $= 10$; en diagonale $= 14$ $(int(10 \times \sqrt{2}))$
  - $h(C)$: $|x_C - x_B| + |y_C - y_B|$

Coût et Heuristiques de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Définir les coûts et heuristiques

- Définir $f = g + h$
  - $g(C)$: détermine le coût du trajet de A vers C en suivant le chemin généré
  - $h(C)$: coût estimé du mouvement de C vers le but B
- Choix effectué pour
  - $g(C)$: coût du trajet entre cases adjacentes $= 10$; en diagonale $= 14$ $(int(10 \times \sqrt{2}))$
  - $h(C)$: $|x_C - x_B| + |y_C - y_B|$

Coût et Heuristiques de la recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

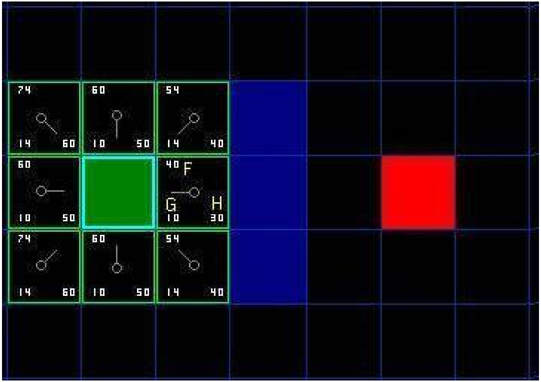
Définir les coûts et heuristiques

- Définir $f = g + h$
  - $g(C)$: détermine le coût du trajet de A vers C en suivant le chemin généré
  - $h(C)$: coût estimé du mouvement de C vers le but B
- Choix effectué pour
  - $g(C)$: coût du trajet entre cases adjacentes $= 10$; en diagonale $= 14$ $(int(10 \times \sqrt{2}))$
  - $h(C)$: $|x_C - x_B| + |y_C - y_B|$

Recherche du plus court chemin par A*
d'après "A* Pathfinding for Beginners" (Patrick Lester)

Valeurs de f, g, h autour de A

valeur de f dans le coin gauche-haut, valeur de g dans le coin gauche-bas et valeur de h dans le coin bas-droite



Déroulement de l'algorithme A*
d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
  - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) :
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*
d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
  - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) ;
  - ces cases sont déjà dans la liste des cases libres ou closes
  - on teste alors si l'accès par la nouvelle case est plus intéressant que l'accès déjà calculé à partir de A
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
  - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) ;
  - ces cases sont déjà dans la liste des cases libres ou closes
  - on teste alors si l'accès par la nouvelle case est plus intéressant que l'accès déjà calculé à partir de A
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
  - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) ;
  - ces cases sont déjà dans la liste des cases libres ou closes
  - on teste alors si l'accès par la nouvelle case est plus intéressant que l'accès déjà calculé à partir de A
    - ex: la case au Sud possède déjà $g(S) = 14$ (coût de A vers celle-ci), en passant par D on a $g(D) + coût(D, S) = 20$: plus coûteux donc n'est pas retenu
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
  - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) ;
  - ces cases sont déjà dans la liste des cases libres ou closes
  - on teste alors si l'accès par la nouvelle case est plus intéressant que l'accès déjà calculé à partir de A
    - ex. la case au Sud possède déjà $g(S) = 14$ (coût de A vers celle-ci), en passant par D on a $g(D) + cout(D, S) = 20$: plus coûteux donc n'est pas retenu
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
  - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) ;
  - ces cases sont déjà dans la liste des cases libres ou closes
  - on teste alors si l'accès par la nouvelle case est plus intéressant que l'accès déjà calculé à partir de A
    - ex. la case au Sud possède déjà $g(S) = 14$ (coût de A vers celle-ci), en passant par D on a $g(D) + cout(D, S) = 20$: plus coûteux donc n'est pas retenu
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
  - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
  - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) ;
  - ces cases sont déjà dans la liste des cases libres ou closes
  - on teste alors si l'accès par la nouvelle case est plus intéressant que l'accès déjà calculé à partir de A
    - ex. la case au Sud possède déjà $g(S) = 14$ (coût de A vers celle-ci), en passant par D on a $g(D) + cout(D, S) = 20$: plus coûteux donc n'est pas retenu
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Déroulement de l'algorithme A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Déroulement de l'algorithme A*

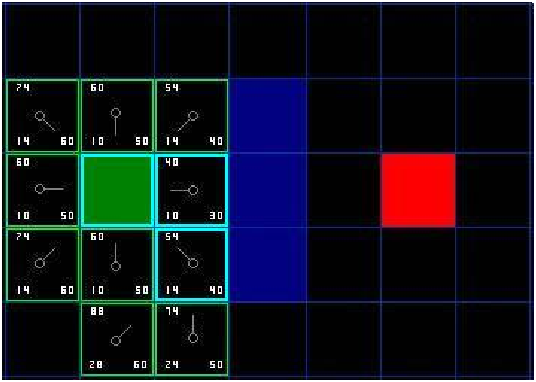
- Les nœuds libres sont les cases voisines de A
- Choix du meilleur nœud, D, à droite de A ($f = 40$, avec $g = 10$ et $h = 30$)
    - on la retire des nœuds libres et la place dans les nœuds clos
- Analyse des cases voisines à la case retenue D
    - seules 5 cases accessibles (par Haut, Bas, HautGauche, BasGauche, Gauche) ;
    - ces cases sont déjà dans la liste des cases libres ou closes
    - on teste alors si l'accès par la nouvelle case est plus intéressant que l'accès déjà calculé à partir de A
        - ex. la case au Sud possède déjà $g(S) = 14$ (coût de A vers celle-ci), en passant par D on a $g(D) + cout(D, S) = 20$: plus coûteux donc n'est pas retenu
- Aucune voisine à D ne convient, choix alors d'un autre meilleur nœud (ici un des nœuds avec $f = 54$)

Recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Sélection du second nœud intéressant

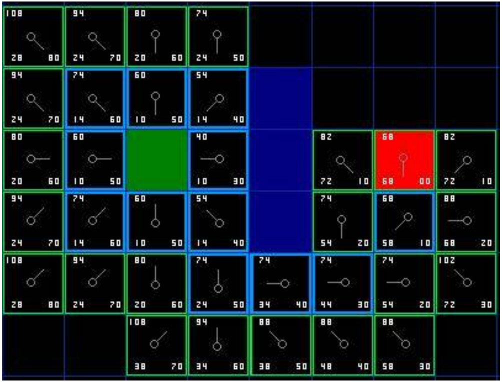
Seuls 2 nœuds libres ajoutés (la diagonale bas-droite est impossible à cause du mur)



Recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

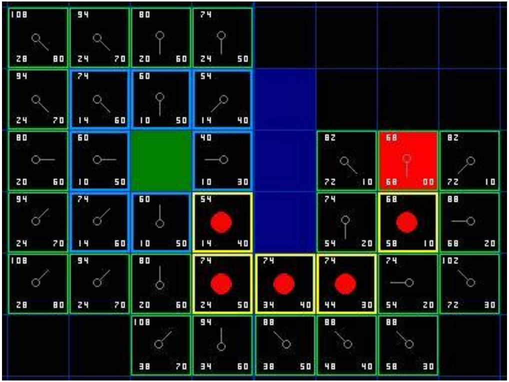
Répétition de l'algorithme jusqu'au but



Recherche du plus court chemin par A*

d'après "A* Pathfinding for Beginners" (Patrick Lester)

Etablissement de la solution à partir de l'arrivée par suivi du nœud père



Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :
  - 1 état = placement des N reines

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :
  - 1 état = placement des N reines,
  - action = échanger deux reines en conflit
  - heuristique = nb de conflits

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :
  - 1 état = placement des N reines,
  - action = échanger deux reines en conflit
  - heuristique = nb de conflits

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :
  - 1 état = placement des N reines,
  - action = échanger deux reines en conflit
  - heuristique = nb de conflits

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :
  - 1 état = placement des N reines,
  - action = échanger deux reines en conflit
  - heuristique = nb de conflits

Autres exemples avec A*

Application de A*

- Résolution du Taquin (N-Puzzle) :
  - 1 état = grille de cases,
  - action = déplacer la case vide,
  - heuristique = distance de manhatan des cases mal placées
- Résolution des N-Reines :
  - 1 état = placement des N reines,
  - action = échanger deux reines en conflit
  - heuristique = nb de conflits

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus d'erreur
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ *5s pour 1000 reines*)

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et non oubliée (perte de noeud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx 5s$ *pour 1000 reines*)

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et *non oubliée* (perte de nœud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ *5s pour 1000 reines*)

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et *non oubliée* (perte de nœud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ 5*s pour 1000 reines*)

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et *non oubliée* (perte de nœud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ *5s pour 1000 reines*)

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et *non oubliée* (perte de nœud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ 5s *pour 1000 reines*)

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et *non oubliée* (perte de nœud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ 5*s pour 1000 reines*)

Recherche Locale

Concept de Recherche Locale, ou Tabou

- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et *non oubliée* (perte de nœud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ 5*s pour 1000 reines*)

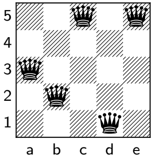
Recherche Locale

Concept de Recherche Locale, ou Tabou

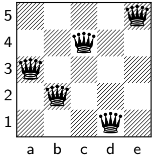
- Cet algorithme se base sur une proposition de solution imparfaite
- Tant que le but n'est pas atteint
  - recherche de la variable permettant de corriger le plus l'erreur
  - recherche d'une meilleure valeur ($\neq$ de la précédente) pour cette variable
    si pas de meilleure valeur et pas d'autre variable aussi problématique,
    recherche de la moins pire des autres valeurs possibles
  - évite de retomber sur une situation rencontrée et *non oubliée* (perte de nœud clos pour libérer la mémoire)
- non complet, possibilité de minima local (correction possible par mémorisation)
- non optimal
- très rapide pour large problème ($\approx$ 5*s pour 1000 reines*)

Recherche Locale, exemple sur n-reines

Placement de 5 reines au hasard

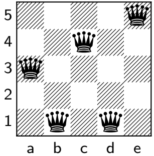


Comptage des contraintes non respectées :
$Q_a = 2$; $Q_b = 2$; $Q_c = 2$; $Q_d = 0$; $Q_e = 2$;
Meilleurs gains possibles aux autres positions pour $Q_a, Q_b, Q_c, Q_e,$:
$Q_a = 1$; $Q_b = 0$; $Q_c = 2$; $Q_e = 1$;
Q_c prend une meilleure valeur

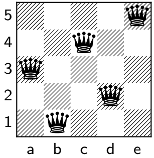


Recherche Locale, exemple sur n-reines

Comptage des contraintes non respectées :
$Q_a = 1$; $Q_b = 2$; $Q_c = 0$; $Q_d = 0$; $Q_e = 1$;
Meilleurs gains possibles aux autres positions pour Q_a, Q_b, Q_e, :
$Q_a = 0$; $Q_b = 1$; $Q_e = 0$;
seule b peut s'ôter 1 pb, elle se déplace



Comptage des contraintes non respectées : $Q_b = 1$; $Q_d = 1$;
Meilleurs gains possibles : $Q_b = 0$, $Q_d = 1$: Q_d est choisie



FIN !

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\infty$

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
  - Il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres
    (nb états potentiels $= 88\ 473\ 600$
    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$ et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
    - chaque nombre généré par une action est ajouté à L
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres

    (nb états potentiels = 88 473 600

    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
    - chaque nombre généré par une action est ajouté à L
    - chaque nombre utilisé par une action est retiré de L
    - une action peut être utilisée plusieurs fois
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres
    (nb états potentiels $= 88\ 473\ 600$
    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
    - chaque nombre généré par une action est ajouté à L
    - chaque nombre utilisé par une action est retiré de L
    - une action peut être utilisée plusieurs fois
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres
    (nb états potentiels = 88 473 600
    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
    - chaque nombre généré par une action est ajouté à L
    - chaque nombre utilisé par une action est retiré de L
    - une action peut être utilisée plusieurs fois
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres
    (nb états potentiels = 88 473 600
    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
    - chaque nombre généré par une action est ajouté à L
    - chaque nombre utilisé par une action est retiré de L
    - une action peut être utilisée plusieurs fois
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres
    (nb états potentiels $= 88\ 473\ 600$
    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
    - chaque nombre généré par une action est ajouté à L
    - chaque nombre utilisé par une action est retiré de L
    - une action peut être utilisée plusieurs fois
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres
    (nb états potentiels $= 88\ 473\ 600$
    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Concept de Recherche Aléatoire

- Pour certains problèmes, l'heuristique est inconnue ou trop complexe
- L'espace d'états est trop important pour balayer le tout
- La stratégie consiste alors à tenter des solutions au hasard
- Exemple : Le compte est bon
  - l'espace d'état est : $+\aleph$
  - à partir d'une liste L de 6 nombres
    $nb_i \in values = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$
  - du choix de deux nombres $(a, b) \in L^2$
    et de 4 actions : $action \in \{a + b, a - b, a \times b, a \div b\}$
    - chaque nombre généré par une action est ajouté à L
    - chaque nombre utilisé par une action est retiré de L
    - une action peut être utilisée plusieurs fois
  - il s'agit d'obtenir ou de se rapprocher au mieux d'un nombre de 3 chiffres
    (nb états potentiels $= 88\ 473\ 600$
    $= 6 \times 4 \times 5 \times 5 \times 4 \times 4 \times 4 \times 4 \times 3 \times 3 \times 4 \times 2 \times 2 \times 4 \times 1$)

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
  - Exécution n°2 : but atteint après 5508 tests
  - Exécution n°3 : but atteint après 730 tests
  - Exécution n°4 : but atteint après 574 tests

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
  - Exécution n°3 : but atteint après 730 tests
  - Exécution n°4 : but atteint après 574 tests

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
  - Exécution n°4 : but atteint après 574 tests

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
  - Exécution n°4 : but atteint après 574 tests

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
    - $25 \times 6 = 150; 75 - 1 = 74; 2 + 7 = 9; 74 + 150 = 224; 9 \times 224 = 2016$
  - Exécution n°4 : but atteint après 574 tests

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
    - $25 \times 6 = 150; 75 - 1 = 74; 2 + 7 = 9; 74 + 150 = 224; 9 \times 224 = 2016$
  - Exécution n°4 : but atteint après 574 tests

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
    - $25 \times 6 = 150; 75 - 1 = 74; 2 + 7 = 9; 74 + 150 = 224; 9 \times 224 = 2016$
  - Exécution n°4 : but atteint après 574 tests
    - $25 - 1 = 24; 2 \times 7 = 14; 24 \times 14 = 336; 6 \times 336 = 2016$

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
    - $25 \times 6 = 150; 75 - 1 = 74; 2 + 7 = 9; 74 + 150 = 224; 9 \times 224 = 2016$
  - Exécution n°4 : but atteint après 574 tests
    - $25 - 1 = 24; 2 \times 7 = 14; 24 \times 14 = 336; 6 \times 336 = 2016$

Recherche Aléatoire

Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
    - $25 \times 6 = 150; 75 - 1 = 74; 2 + 7 = 9; 74 + 150 = 224; 9 \times 224 = 2016$
  - Exécution n°4 : but atteint après 574 tests
    - $25 - 1 = 24; 2 \times 7 = 14; 24 \times 14 = 336; 6 \times 336 = 2016$

Recherche Aléatoire

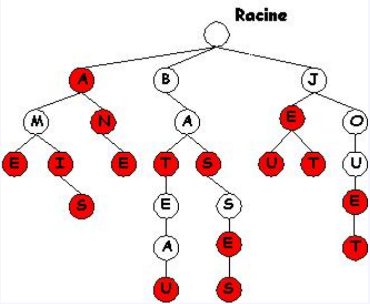
Exemple de résolution de compte

- Etat initial : $[1, 2, 6, 7, 25, 75]$
- Etat Final : 2016
  - Exécution n°1 : but atteint après 718 tests
    - $25 - 1 = 24; 2 \times 6 = 12; 12 \times 7 = 84; 84 \times 24 = 2016$
  - Exécution n°2 : but atteint après 5508 tests
    - $2 + 7 = 9; 75 + 9 = 84; 25 - 1 = 24; 24 \times 84 = 2016$
  - Exécution n°3 : but atteint après 730 tests
    - $25 \times 6 = 150; 75 - 1 = 74; 2 + 7 = 9; 74 + 150 = 224; 9 \times 224 = 2016$
  - Exécution n°4 : but atteint après 574 tests
    - $25 - 1 = 24; 2 \times 7 = 14; 24 \times 14 = 336; 6 \times 336 = 2016$

Recherche Exhaustive

Concept de Recherche Exhaustive

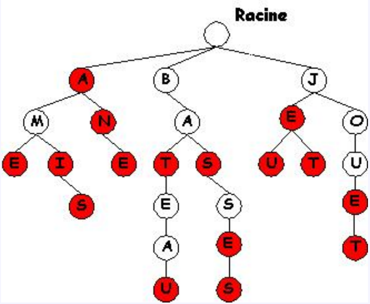
- Pour certains problèmes, l'espace d'états est important, mais il est nécessaire de balayer l'ensemble des combinaisons
- Exemple : Le Scrabble
  - Pour le scrable, utilisation d'un arbre lexicographique généralisé (cf. http://lvh.free.fr/pages/algo/minmax/scrabble.html)



Recherche Exhaustive

Concept de Recherche Exhaustive

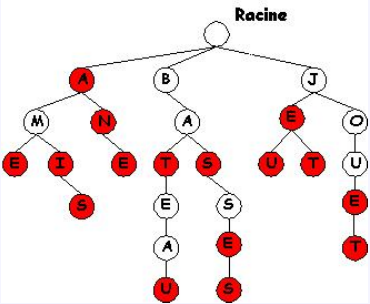
- Pour certains problèmes, l'espace d'états est important, mais il est nécessaire de balayer l'ensemble des combinaisons
- Exemple : Le Scrabble
  - Pour le scrable, utilisation d'un arbre lexicographique généralisé (cf. http ://lwh.free.fr/pages/algo/minmax/scrabble.htm)



Recherche Exhaustive

Concept de Recherche Exhaustive

- Pour certains problèmes, l'espace d'états est important, mais il est nécessaire de balayer l'ensemble des combinaisons
- Exemple : Le Scrabble
  - Pour le scrable, utilisation d'un arbre lexicographique généralisé (cf. http ://lwh.free.fr/pages/algo/minmax/scrabble.htm)



Recherche Exhaustive

Concept de Recherche Exhaustive

- Pour certains problèmes, l'espace d'états est important, mais il est nécessaire de balayer l'ensemble des combinaisons
- Exemple : Le Scrabble
  - Pour le scrable, utilisation d'un arbre lexicographique généralisé (cf. http ://lwh.free.fr/pages/algo/minmax/scrabble.htm)