Heuristiques

C. Recherche de la meilleure branche

- **. Branch And Bound**
- **. Branch and Bound avec sous estimations**
- **. Branch and Bound avec programmation dynamique**
- **. A***

Heuristiques

Recherche de la meilleure branche

- **utilisées quand on donne une importance primaire au coût de la branche.**
- **Pour les méthodes précédentes, on ne tient pas compte du passif. Ici, il est tenu compte du passif. On attribue donc un coût à toute la branche parcourue depuis la racine.**

Heuristiques

Branch and Bound

- **Tenir compte du passif (attribuer une valeur à chaque branche)**
- **Utilise une liste (ou file d'attente avec priorité)**

1. Placer le nœud début de longueur 0 dans la liste.

2. Répéter jusqu'à ce que liste vide ou nœud recherché trouvé :

a) Si la première branche contient le nœud recherché, fin avec succès.

c) Sinon

- supprimer la branche de la liste et former des branches nouvelles en étendant la branche supprimée d'une étape.

- calculer les coût cumulés des branches et les ajouter dans la liste de telle sorte que la liste soit triée en ordre croissant.

3. Autrement " pas d'élément"

Heuristiques

Branch and Bound : Exemple

- **Pour la sortie du labyrinthe, On cumule les distances des branches parcourues.**

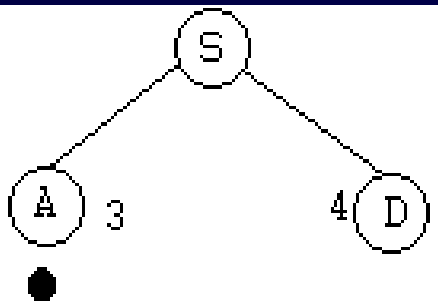
- **Remarque :**

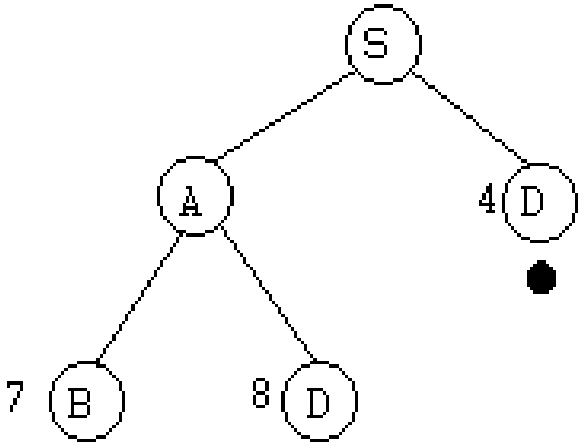
- **Donc pas d'estimation sur la distance restante.**
- **L'heuristique réside dans le fait qu'on choisit une branche arbitraire qui ne peut être la meilleure.(on est guidé par une choix).**

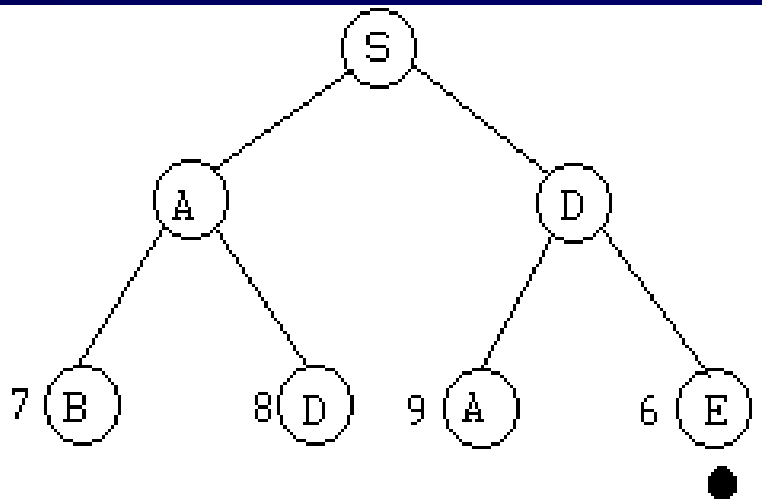
- **On trouve donc toujours la solution optimale.**

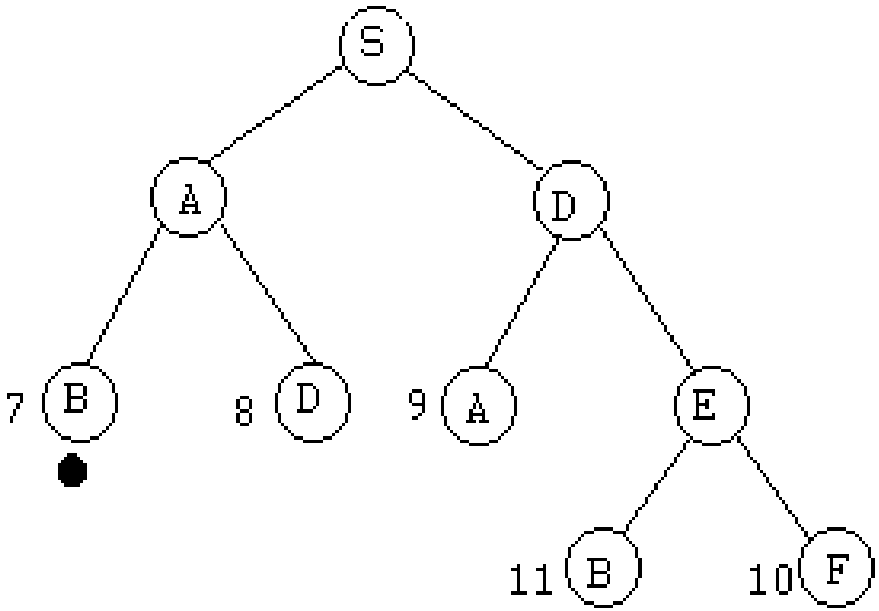
Heuristiques

Branch and Bound : Exemple









Heuristiques

Branch and Bound avec sous-estimations

- **Tenir compte du passif et de l'estimation restante.**
- **Algorithme**

1. Placer le nœud début de longueur 0 dans la liste.

2. Répéter jusqu'à ce que liste vide ou nœud recherché trouvé :

a) Si la première branche contient le nœud recherché, fin avec succès.

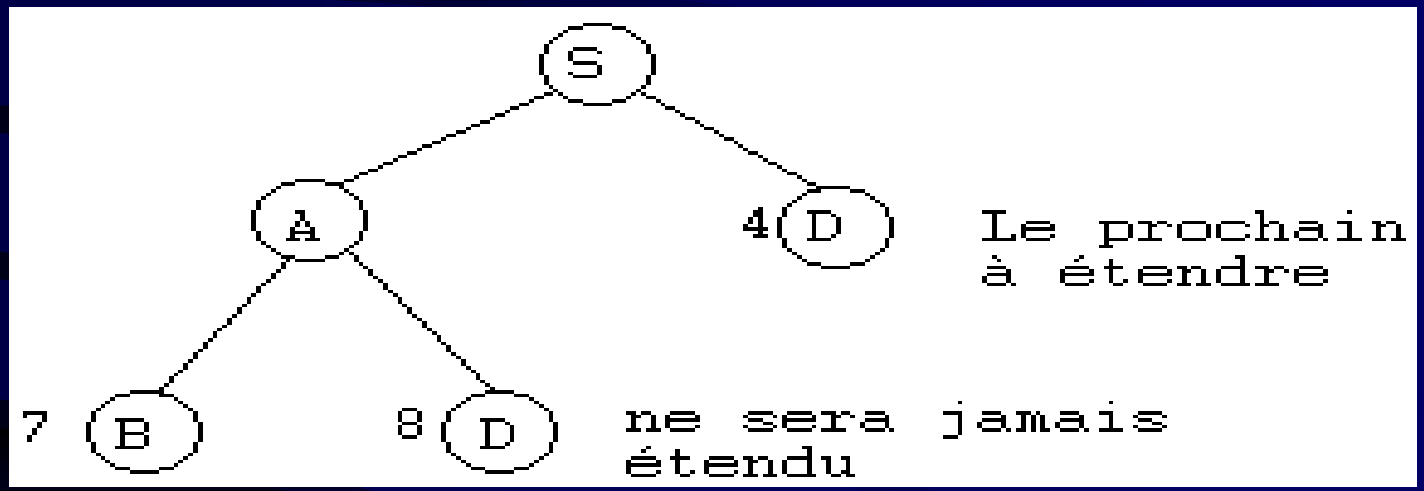
b) Sinon

- supprimer la branche de la liste et former des branches nouvelles en étendant la branche supprimée d'une étape.

- calculer la *somme du coût cumulé et la limite inférieure de l'estimation de la distance restante* des branches et les ajouter dans la liste de telle sorte que la liste soit triée en ordre croissant.

Heuristiques

Branch and Bound avec sous-estimations



Les valeurs à coté des nœuds désignent les distances accumulées.

Heuristiques

Branch and Bound avec programmation dynamique

1. Placer le nœud début de longueur 0 dans la liste.

2. Répéter jusqu'à ce que liste vide ou nœud recherché trouvé :

a) Si la première branche contient le nœud recherché, fin avec succès.

b) Sinon

- supprimer la branche de la liste et former des branches nouvelles en étendant la branche supprimée d'une étape.

- calculer les coûts cumulés des branches et les ajouter dans la liste de telle sorte que la liste soit triée en ordre croissant.

- Si deux ou plusieurs branches possèdent un nœud en commun, éliminer toutes ces branches sauf celle avec un coût minimal.

Heuristiques

A*

- **C'est branch and bound amélioré. Il combine l'estimation de la distance restante avec la programmation dynamique.**

Heuristiques

A*

1. Placer le nœud début de longueur 0 dans la liste.

2. Répéter jusqu'à ce que liste vide ou nœud recherché trouvé :

a) Si la première branche contient le nœud recherché, fin avec succès.

b) Sinon

- supprimer la branche de la liste et former des branches nouvelles en étendant la branche supprimée d'une étape.

- calculer la *somme du coût cumulé et la limite inférieure de l'estimation de la distance restante* des branches et les ajouter dans la liste de telle sorte que la liste soit triée en ordre croissant.

- Si deux ou plusieurs branches possèdent un nœud en commun, éliminer toutes ces branches sauf celle avec un coût minimal.

Heuristiques

Autre version de l'algorithme A*.

- **On maintient deux listes Ouvert(O) et Fermé(F) dont l'une est une file d'attente avec priorité(Ouvert).**
- **g : coût réel au nœud n**
- **h: estimation de la distance restante**
- **f = g + h**

Heuristiques

Autre version de l'algorithme A*.

1. O <-- nœud initial s.

2. Retirer un nœud n de Ouvert qui a la plus petite valeur de f(n). Si c'est le nœud but, stop avec succès. Autrement

- F <-- n

- Générer les successeurs n'

- Pour chaque successeur n' :

Si n' non dans O et n' non dans F :

. Calculer f(n') = g(n') + h(n')

. O <-- n'

. attacher un pointeur arrière vers n

Fsi

Heuristiques

Autre version de l'algorithme A*.

Si n' est dans O ou n' est dans F :

. Changer le chaînage arrière selon le coût de la plus petite branche (g)

Fsi

Si n' est dans F et son chaînage arrière est modifié:

. l'enlever de Fermé

. Enlever tous les nœuds de la descendance de n' qui sont dans O et dans F.

. O <-- n'.

Fsi

Finpour

3. allera 2

Heuristiques

Autre version de l'algorithme A*.

- Remarques à propos de g :

— La fonction g nous laisse choisir quel nœud élargir sur la base de tout le chemin depuis la racine. Ce nœud peut ne pas être le meilleur.

— Si nous voulons arriver à une solution d'une façon ou d'une autre, il suffit de donner à g la valeur 0.(Cas des algorithmes gloutons)

— Si nous voulons trouver un chemin (solution) avec le plus petit nombre de pas possible, il suffit de donner à g la valeur 1.

— Si nous voulons trouver le chemin le moins coûteux, il suffit de donner à g le coût du trajet(ou de l'opération).

— Ainsi A* peut être utilisé pour trouver un chemin au moindre coût ou un chemin quelconque aussi rapide possible.

Heuristiques

Autre version de l'algorithme A*.

- **Remarques à propos de h :**

— **Si h' est parfait, A* converge immédiatement vers la solution sans recherche. Meilleur est h' plus nous approchons de plus près de la solution.**

— **Si h' vaut 0, la recherche sera contrôlée uniquement par g. Si g vaut 0, la recherche est aléatoire. Si g vaut toujours 1, la recherche se fera en largeur.**

— ***** Si h' ne surestime jamais h(distance réel), A* est assuré de trouver un chemin optimal vers un but s'il existe.(propriété d'admissibilté)***