# Introduction à la Complexité et NP-complétude

# Qu’est-ce que la complexité?

- Question centrale de l’informatique théorique: quelles sont les limites des ordinateurs? Existe-t-il des limites fondamentales qui sont indépendantes de la technologie?
- Peut-on facilement identifier les problèmes de calcul qui sont ainsi hors de portée?

# Indécidabilité

- Les limites des ordinateurs telles qu'on les comprend au début du XXe siècle.
- Turing (et d'autres) formalisent la notion de calcul automatique et d'algorithme et démontrent qu'il existe des *langages indécidables* c'est-à-dire des problèmes de calcul qui ne peuvent être résolus par aucun algorithme.

# Complexité

- En pratique, un algorithme n'est utile que s'il consomme une quantité de ressources *raisonnable*.
- Ressources de calcul: nombre d'opérations effectuées, temps, quantité de mémoire utilisée, nombre de processeurs, communication (dans un calcul distribué), taille d'une puce, etc.
- La complexité d'un problème de calcul est la quantité minimale de ressources de calcul nécessaire à sa résolution.

# Pourquoi la complexité?

- Pour développer notre compréhension des difficultés inhérentes au calcul.
- Parce que c'est un outil important pour diriger l'algorithmique.
- Parce que la sécurité de certains protocoles cryptographiques repose sur l'impossibilité (par exemple) de factoriser rapidement de grands entiers.

# Complexite : concepts

- On se concentre surtout sur le temps de calcul $\approx$ nombre d'opérations.

Les principaux facteurs déterminant le temps nécessaire à un calcul sont la taille de l'entrée, l'ordinateur utilisé et le langage de programmation utilisé.

On veut développer une théorie robuste qui fait abstraction des deux derniers facteurs. $\Rightarrow$ nécessité de s'entendre sur des modèles de calcul.

# Classes de complexité

Premier objectif: regrouper ensemble des problèmes résolubles avec une quantité de ressources donnée.

- P: problèmes de décision résolubles en temps polynomial.
- NP: résolubles en temps polynomial avec un algorithme non-déterministe.
- EXP: résolubles en temps exponentiel.

Rôle central de ces deux premières classes.

# Complexite : concepts

Pour placer un problème dans une classe de complexité donnée, il suffit de fournir un algorithme opérant dans les contraintes de ressources correspondantes. ⇒ algorithmique

Idéalement, on voudrait un moyen de placer chaque problème dans la *plus petite classe de complexité possible.* Cela correspondrait à trouver un algorithme optimal.

# Réductions

- Il est très difficile de démontrer formellement qu'un problème nécessite une quantité minimale de ressources.
- Il existe par contre de bons outils pour comparer la complexité de deux problèmes.

# Complexité : concepts

Le problème de calcul A se réduit au problème B s'il existe un algorithme « efficace » qui permet de résoudre A en utilisant des appels procéduraux à B.

Ceci est dénoté $A \leq_T B$. On peut avoir à la fois $A \leq_T B$ et $B \leq_T A$. Dans ce cas, on considère que A et B sont des problèmes de complexité équivalente.

Avantage: Si un problème qui nous intéresse est de complexité équivalente à des problèmes pour lesquels aucun algorithme efficace n'est connu, on a une forte indication que ce problème est très complexe.

# Complétude

- Un problème A est complet pour la classe de complexité **C** si
  1. $A \in \mathbf{C}$
  2. Pour tout $B \in \mathbf{C}$ on a $B \leq_T A$.

C’est-à-dire que A est le problème le plus dur de sa classe.

# Complexité : concepts

- Si A est complet pour la classe **C** alors soit A n'admet pas d'algorithme efficace, soit il en admet un et alors *tous* les problèmes de **C** en admettent un également.
- Par contraposé, s'il existe un seul problème de **C** qui n'admet pas d'algorithme efficace, alors A n'admet pas d'algorithme efficace.

## Exemple : Problème du voyageur de commerce(TSP):

Étant donné un réseau de villes et un coût de déplacement $d_{ij}$ entre les villes i et j.

On cherche à trouver un circuit fermé qui visite chaque ville une seule fois en empruntant le parcours de plus bas coût possible.

## Exemple : Problème du voyageur de commerce(TSP):

Plusieurs types de problèmes de calcul.

- Problèmes de décision: réponse est oui/non. (ou 0/1)
  - TSP, existe-t-il un circuit de coût $\leq$ t?
- Problèmes de recherche ou d'optimisation: recherche d'une solution à un ensemble de contraintes
  - TSP, quel est le circuit optimal?
- Problèmes d'évaluation
  - TSP, quel est le coût du circuit optimal?

# Remarques sur TSP

- On peut facilement évaluer le coût d'un circuit donné.
- Même pour un petit nombre de villes, il y a un nombre astronomique de circuits possibles, donc la fouille systématique n'est pas envisageable.
- Deux caractéristiques typiques des problèmes d'*optimisation combinatoire* rencontrés très fréquemment en algorithmique.

## Exemple : Problème du voyageur de commerce (TSP):

TSP est NP-complet. Cela semble indiquer qu'il n'existe pas d'algorithme efficace pour le résoudre.

Alors que conseiller au voyageur de commerce ?

Rep : le même problème se pose pour un très grand nombre de problèmes d'optimisation.

# NP-complétude

La NP-complétude est un des outils les plus utilisés pour donner une indication forte qu'un problème n'admet pas d'algorithme efficace.

- Exemples choisis: TSP, couvertures de graphes, coloriage de graphe, problème du sac à dos, etc.
- Aperçu des techniques utiles pour établir ce genre de résultat.

# Contourner la NP-complétude

- Considérer des variantes plus simples du problèmes.
  - TSP est-il plus simple si on considère que $d_{ij} \in \{0,1\}$?
  - TSP est-il plus simple si on considère que $d_{ij} + d_{jk} \geq d_{ik}$?

# Contourner la NP-complétude

- Algorithmes **d'approximation**
  - Existe-t-il un algorithme efficace qui obtient une solution de TSP qui est au pire 50% plus chère que la solution optimale?
  - Peut-on formellement démontrer qu'un tel algorithme n'existe pas à moins que TSP admette lui-même un algorithme efficace?

# Contourner la NP-complétude

- Algorithmes probabilistes
  - Peut-on trouver un algorithme efficace qui aura 99% de chances de trouver une solution optimale à chaque instance?
- **Heuristiques**
  - Peut-on trouver un algorithme efficace qui trouve une solution optimale pour 99% des instances qui nous intéressent?

# Contourner la NP-complétude

- Complexité paramétrée
  - Raffiner l'analyse du temps d'exécution d'un algorithme pour confiner l'explosion du temps de calcul à un paramètre qui reste petit dans les applications pratiques.
  - Limites de ce paradigme.

# P = NP?

- Beaucoup de problèmes sont NP-complets. Donc si P ≠ NP, aucun de ceux-ci n'admet d'algorithme qui s'exécute en temps polynomial.
- Malheureusement, on ne sait pas comment démontrer P ≠ NP.
- Et si P = NP?

# Heuristique

# Heuristique : Définition

une méthode approchée conçue pour un problème particulier
pour produire
des solutions non nécessairement optimales (avec un temps de calcul raisonnable)

# Métaheuristique

Métaheuristique

- une stratégie (règle) de choix pilotant une heuristique
- un schéma de calcul heuristique, général et adaptable à un ensemble de problèmes différents

# Métaheuristique

Métaheuristique

- une stratégie (règle) de choix pilotant une heuristique
- un schéma de calcul heuristique, général et adaptable à un ensemble de problèmes différents

# Métaheuristique

Métaheuristique

- une stratégie (règle) de choix pilotant une heuristique
- un schéma de calcul heuristique, général et adaptable à un ensemble de problèmes différents

## 1 exemple d'heuristique :

Recherche de chemins de coût minimal avec l’algorithme A*

# Objectif à réaliser sous forme de mini-projet

**Objectif :** Déterminer, pour un agent* donné, un chemin de coût minimum depuis un sommet source vers un sommet destination au sein d'un graphe orienté.

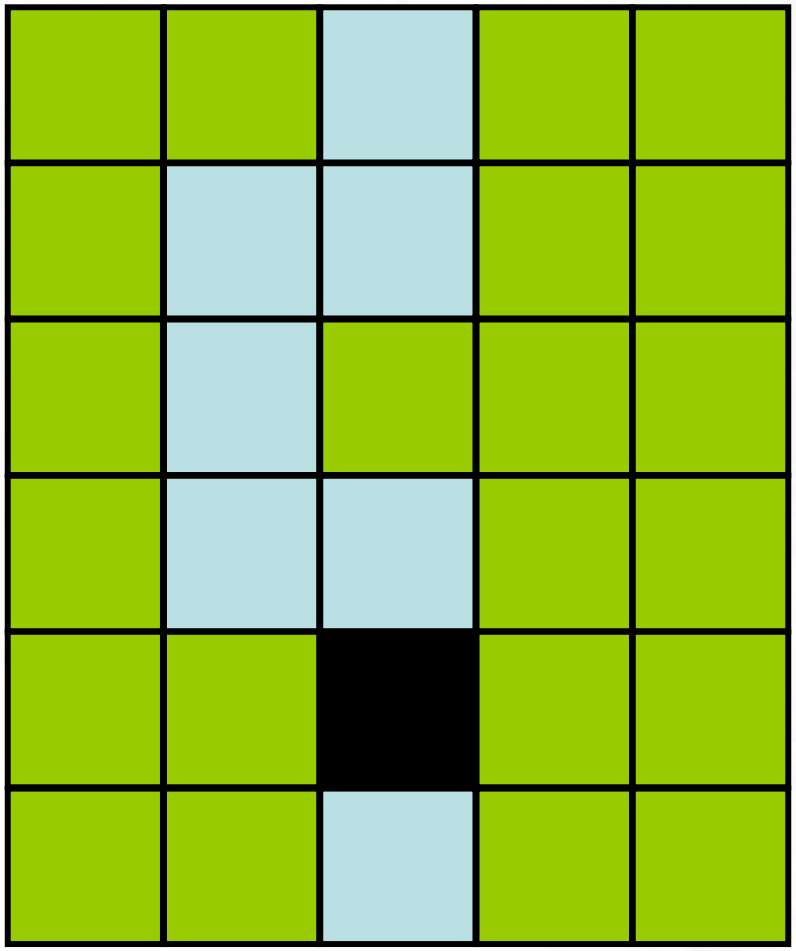
Un ***agent*** est un objet informatique autonome utilisé pour représenter une entité mobile dotée d'un comportement (humain, animal, véhicule, …)

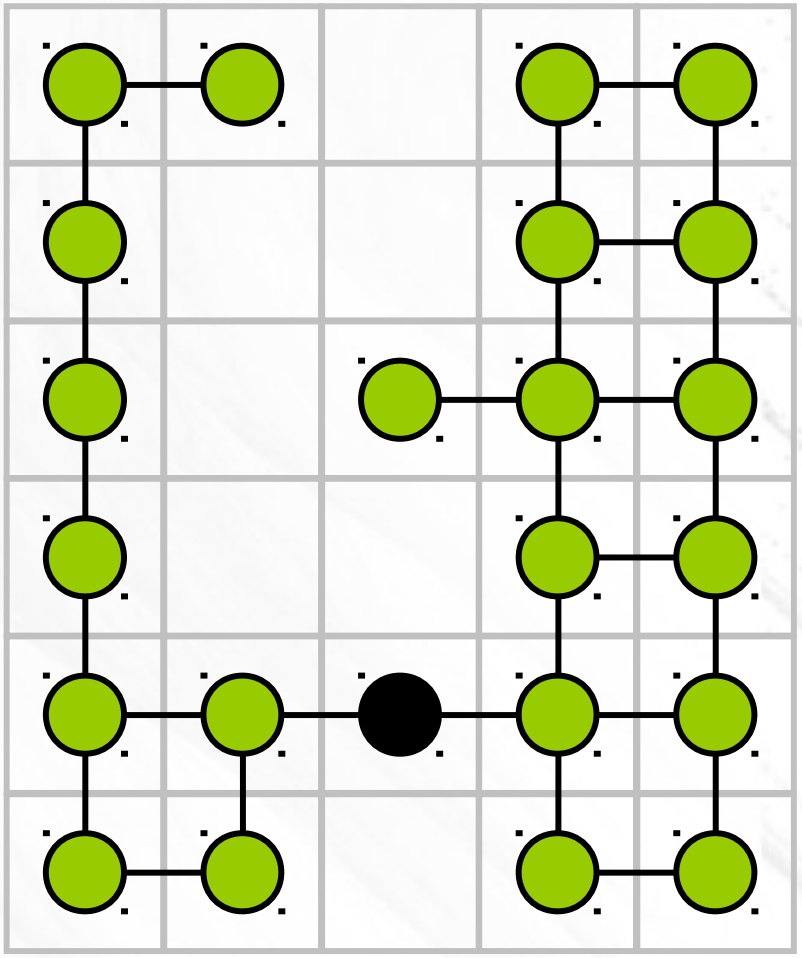
# Applications

- Jeux vidéo
  - Animation des personnages non joueurs
  - Déplacement réaliste d'un personnage contrôlé par le joueur vers un objectif désigné par le joueur
- Simulation – vie artificielle
  - Etude du comportement d'une foule, du traffic automobile, …
  - Effets spéciaux (scènes de bataille, …)

# Représentation du graphe à partir d’informations topographiques

# Relations d’adjacence : grille carrée



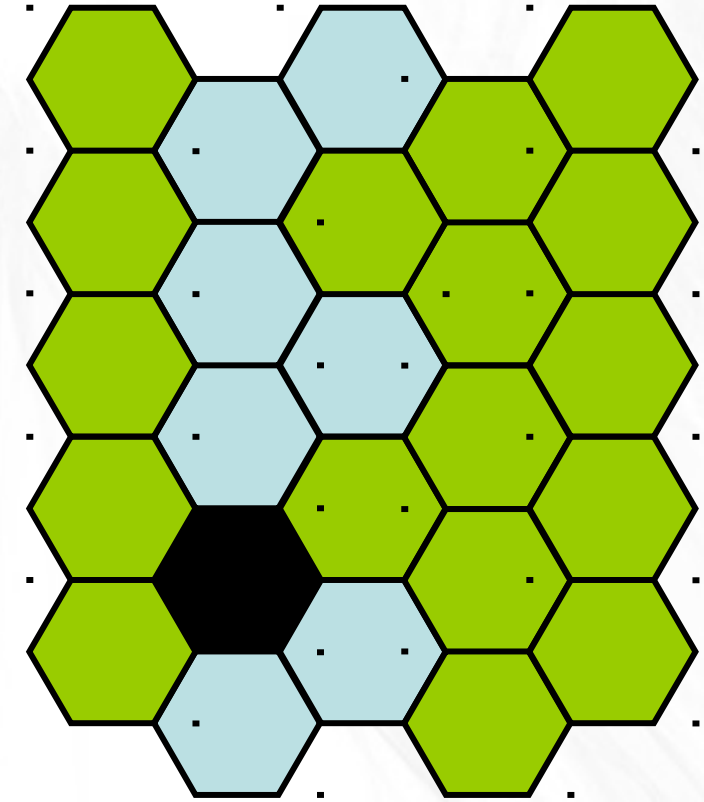


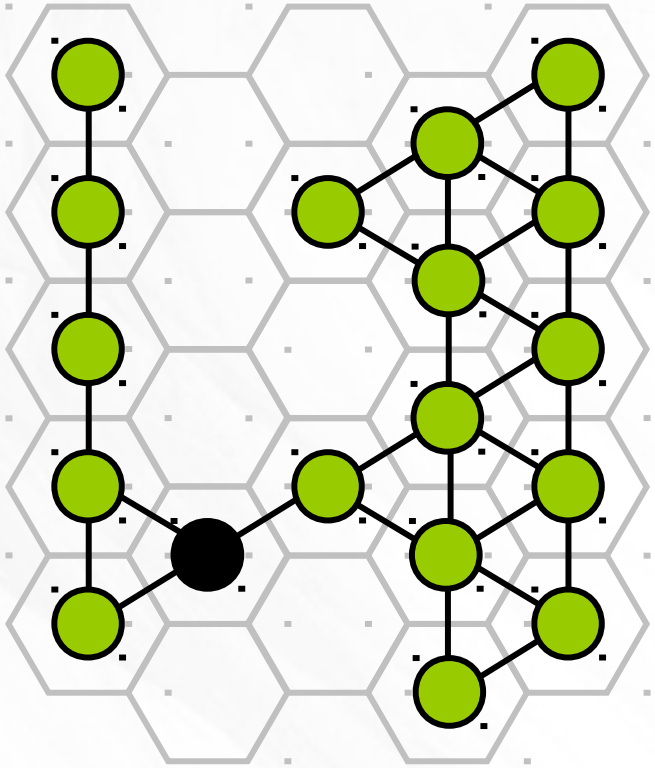
Prairie

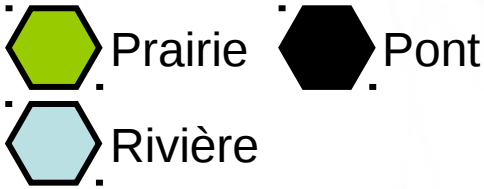
Pont

Rivière

# Relations d’adjacence : grille hexagonale



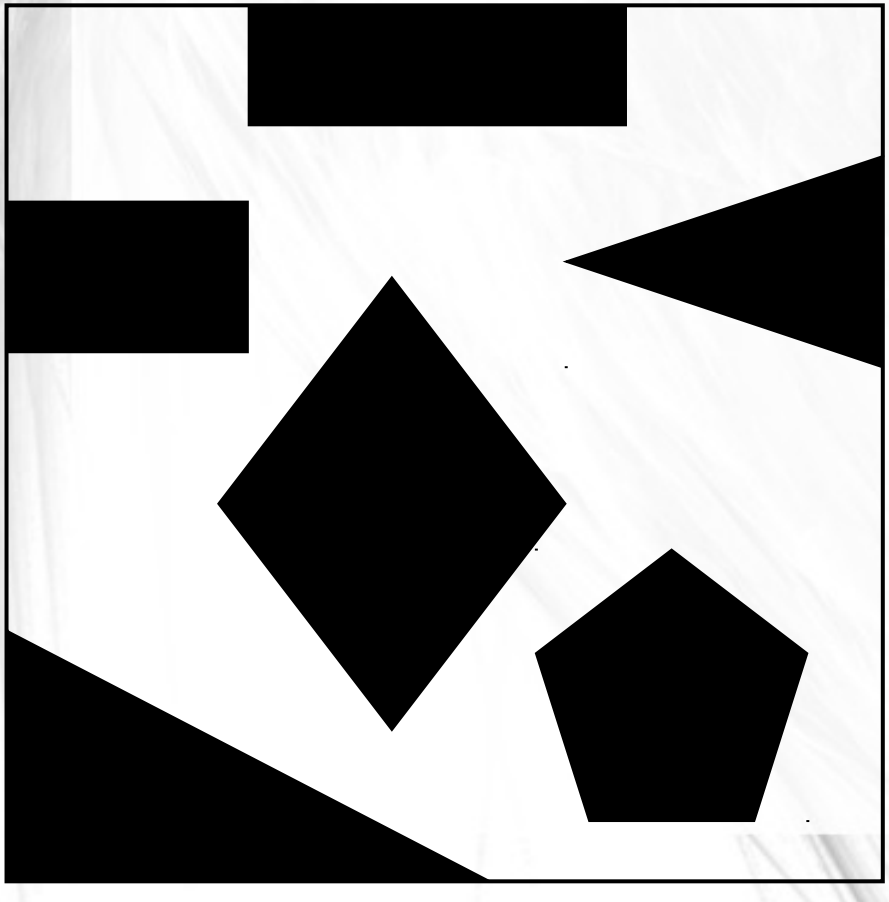


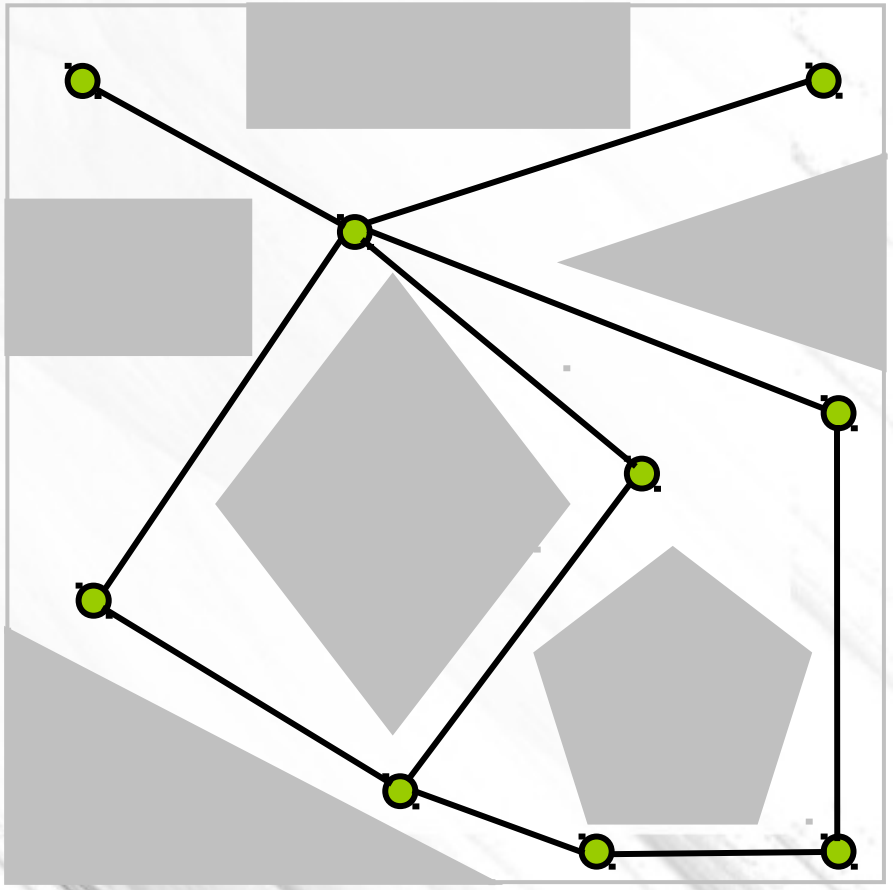


# Relations d’adjacence : points visibles

Obstacles

Couloirs



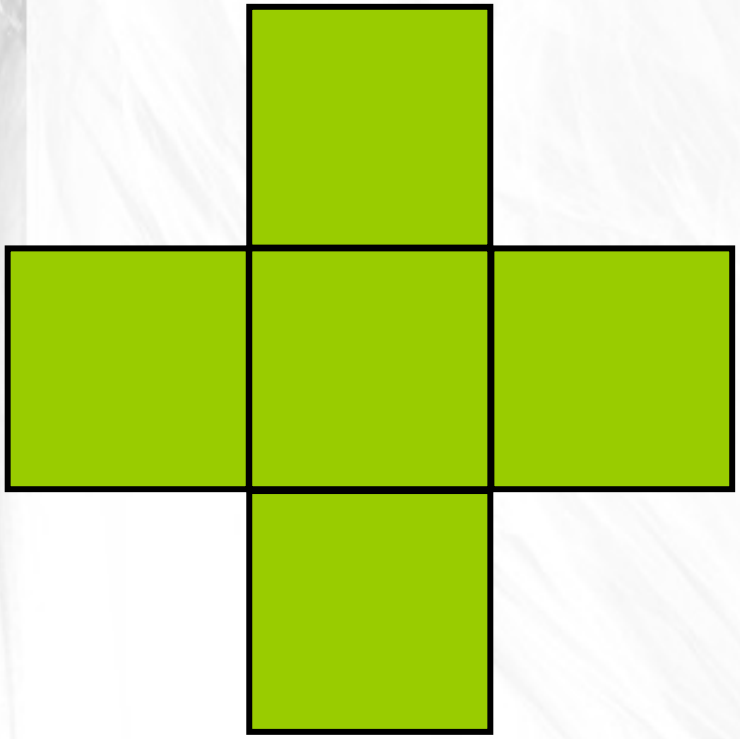


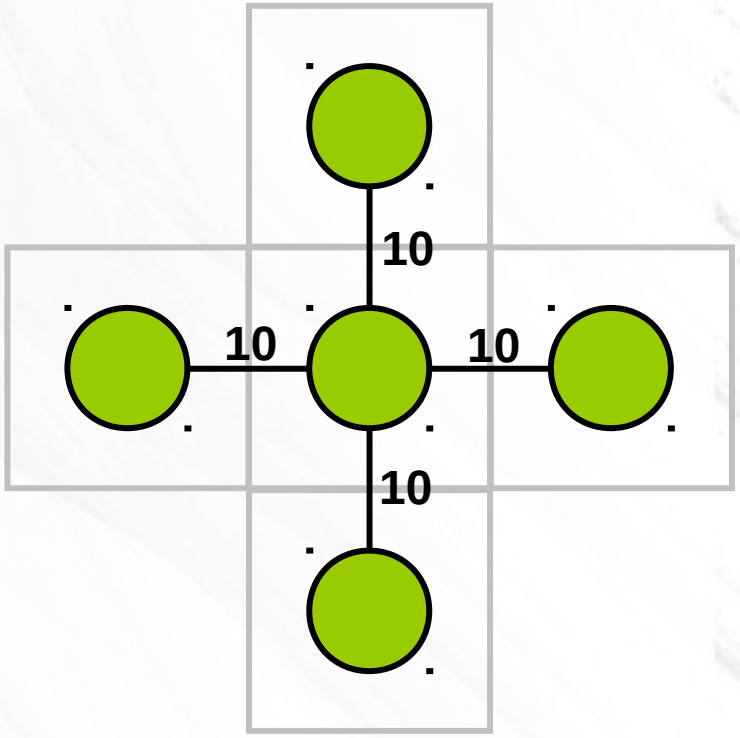
# Coût des arcs

Signification du coût d’un arc :

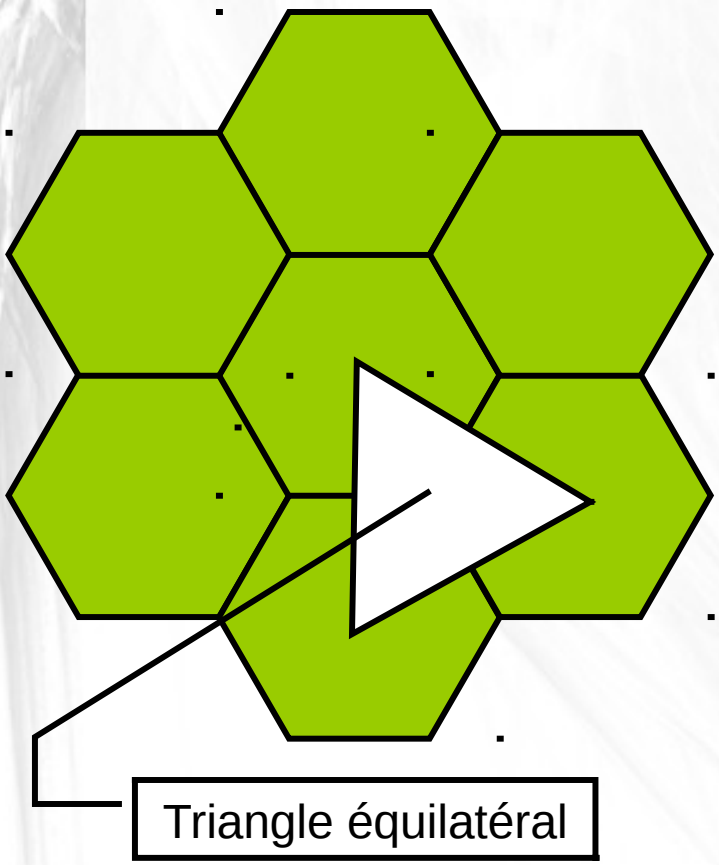
- Distance kilométrique
  - Recherche de chemins de longueur minimale
- Temps (nécessaire au franchissement de l’arc)
  - Recherche de chemins en temps minimum
- Consommation de carburant
  - Rechercher de chemins « économes »

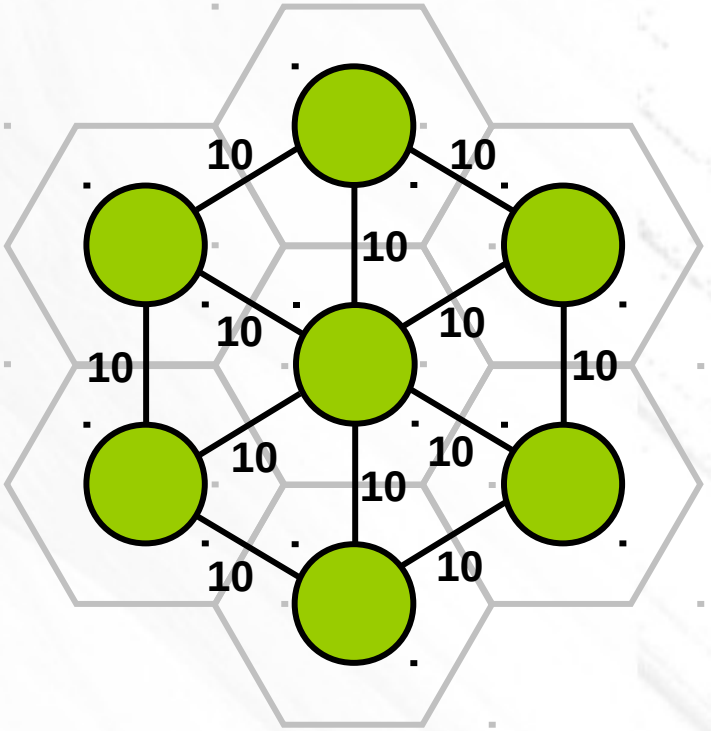
# Coût des arcs : grille carrée



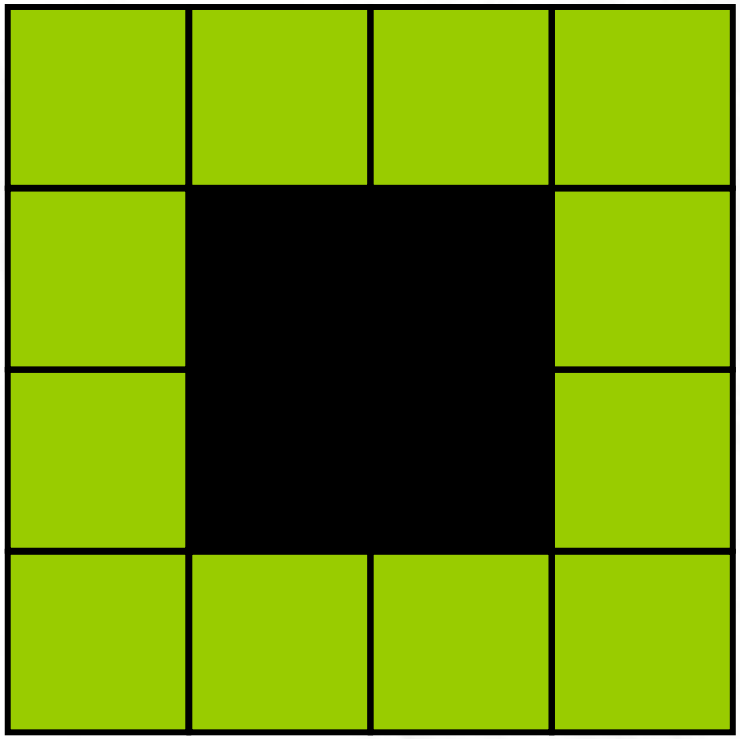


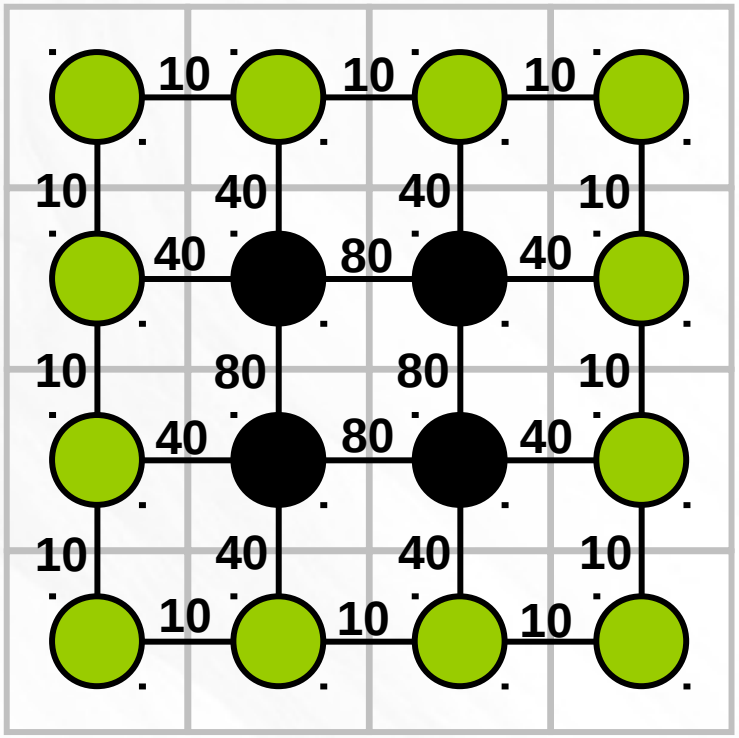
# Coût des arcs : grille hexagonale



# Coût des arcs : pondération en fonction de la nature de l'environnement

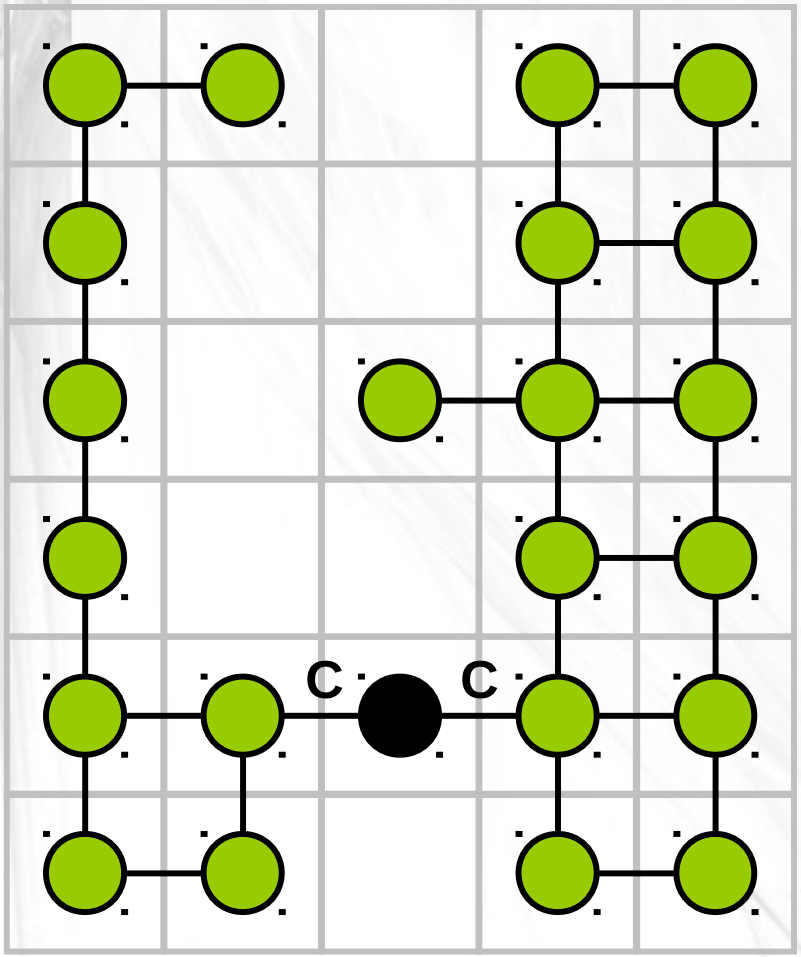





Prairie

Montagne

# Coût des arcs : pondération en fonction de la nature de l'agent




Coût du franchissement d'un pont

**C = 10** pour un humain.

**C = 50** pour une voiture.

**C = 500** pour un semi-remorque.

# Algorithme A*

# Principe général : évaluation du coût total d'un sommet

Coût total (**F**) = Coût depuis la source (**G**) +
Coût vers la destination (**H**)

- **G** : Coût depuis la source
  - Algorithmes classiques (Ford, Bellman, Dijkstra)
  - $G_i = \min G_j + C_{ij}$ / i prédecesseur de j
    $C_{ij}$ coût de l'arc (i,j)

- **H** : Coût vers la destination
  - Difficile puisque le reste du chemin (vers la destination) est encore inconnu.

# Coût vers la destination

**Pourquoi évaluer un coût vers la destination ?**

Afin de resserrer l'ensemble des sommets à explorer en privilégiant les sommets « qui semblent » nous rapprocher de la destination.

**Remarque**

Dans le cas d'un algorithme de recherche plus classique (Dijsktra), on effectue une recherche exhaustive parmi **TOUS** les sommets.

**Conséquence**

l'algorithme A* est plus performant que n'importe quel autre algorithme puisqu'il diminue l'ensemble des sommets à explorer.

# Coût vers la destination

**Comment évaluer un coût vers la destination ?**

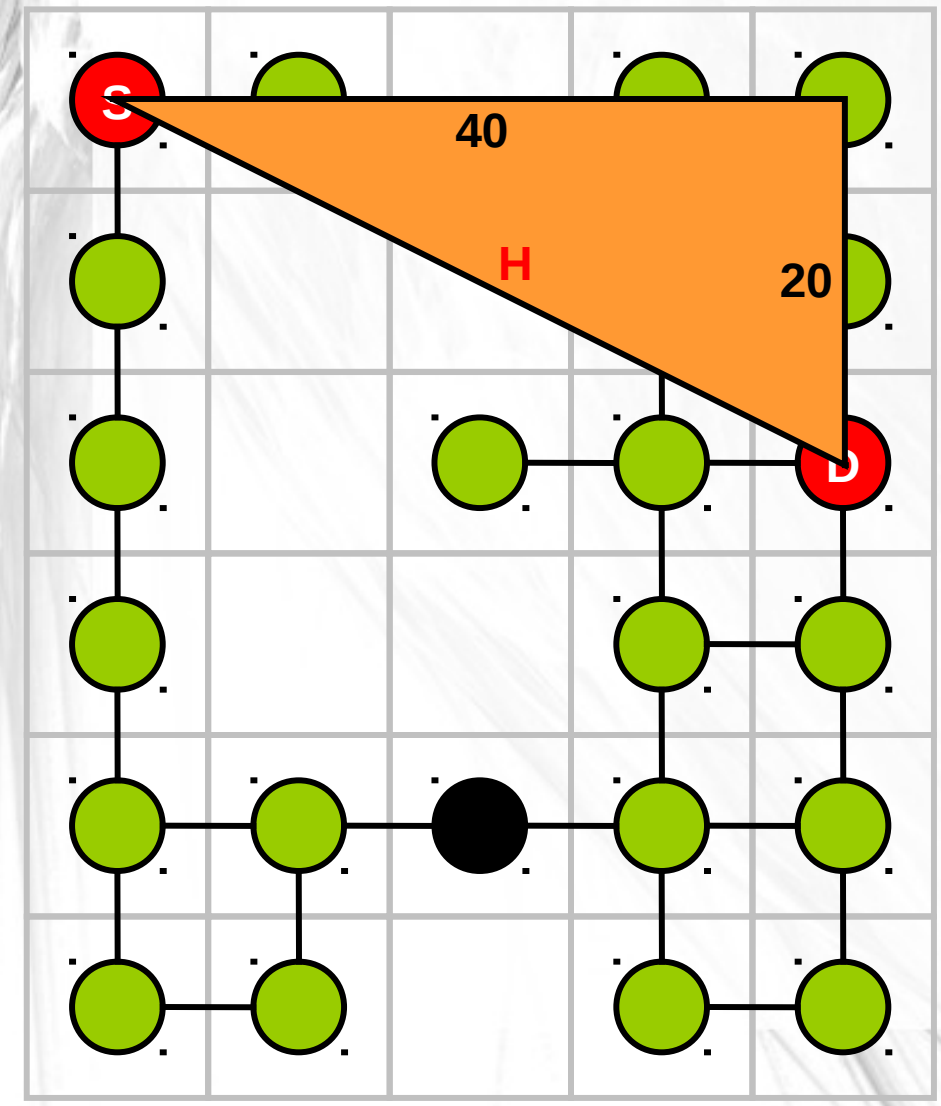
En utilisant des heuristiques (prédictions) afin d'évaluer un coût vers la destination **INFERIEUR** au coût réel (encore inconnu).

A ce titre, A* est un algorithme ***optimiste***.

**Remarque**

Si l'heuristique était supérieur au coût réel, on risquerait de générer un chemin qui ne soit pas minimal.

# Distance euclidienne




**Théorème de Pythagore**

$H^2$ = (Coté oppose) $^2$ +

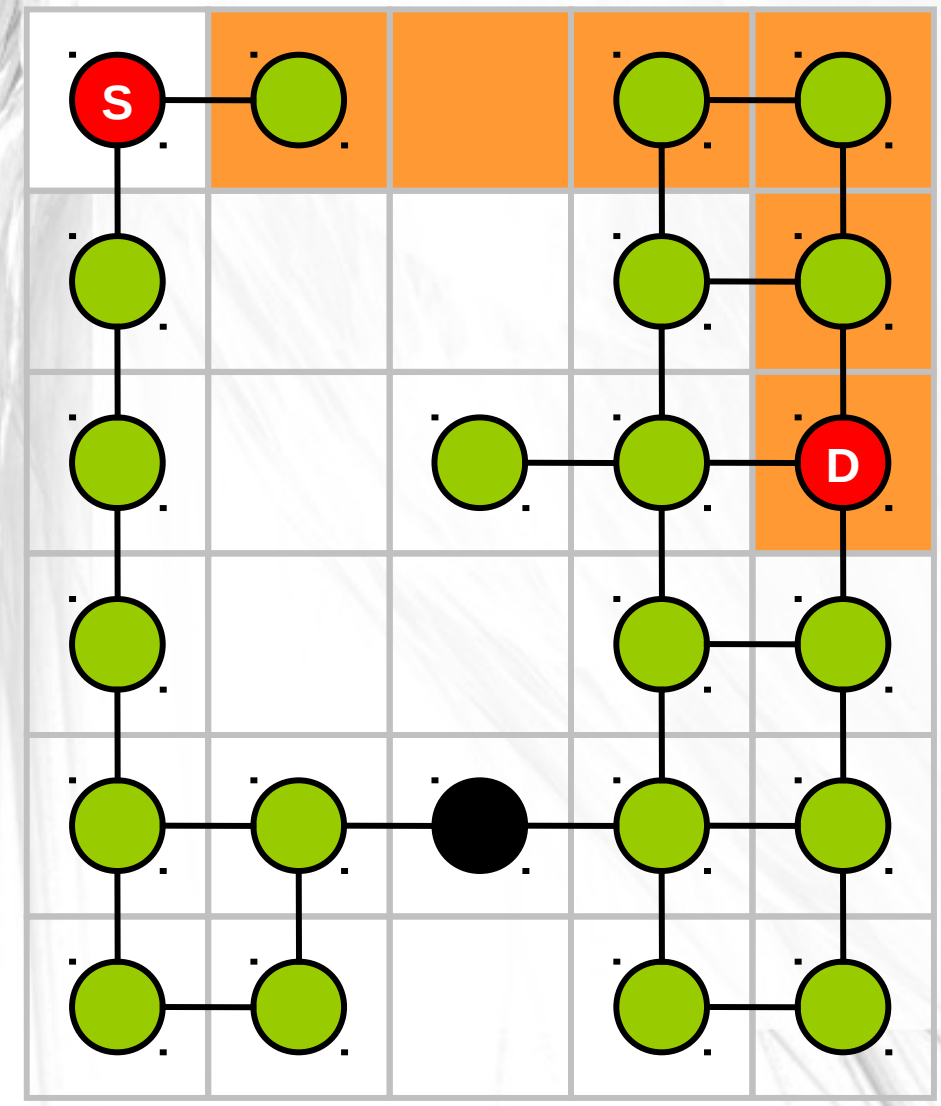
(Coté adjacent) $^2$

$H^2 = 40^2 + 20^2$

$= 2000$

$H = 20 \times (5)^{1/2}$

# Distance de Manhattan




Nombre de cellules, en **horizontal** et en **vertical** entre la source et la destination.

Plus conforme à la nature des déplacements autorisés (haut, bas, gauche, droite)

# Algorithme A*

**Initialisation**

Sommet source (S)

Sommet destination (D)

Liste des sommets à explorer (E) : sommet source S

Liste des sommets visités (V) : vide

Tant que (la liste E est non vide) et (D n'est pas dans E) Faire

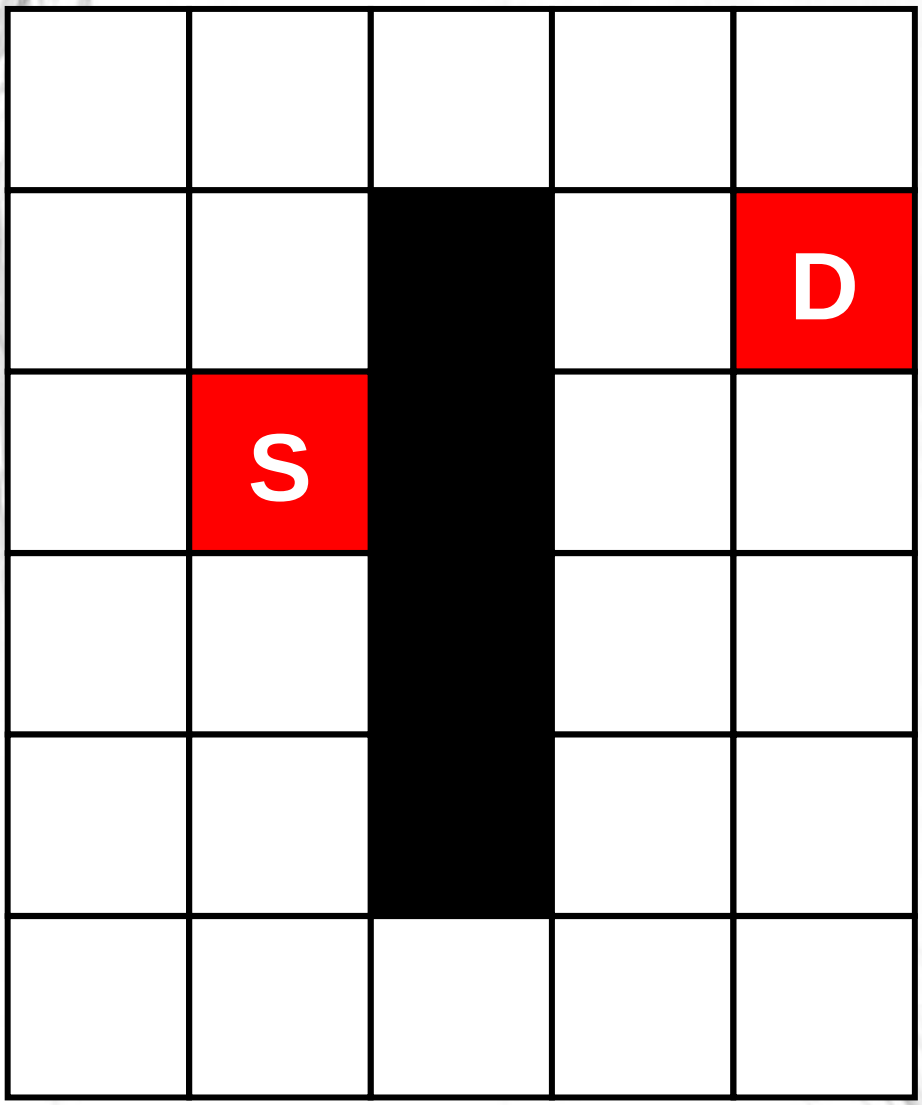
+ Récupérer le sommet X de coût total F minimum.
+ Ajouter X à la liste V
+ Ajouter les successeurs de X (non déjà visités) à la liste E en évaluant leur coût total F et en identifiant leur prédécesseur.

+ Si (un successeur est déjà présent dans E) et
  (nouveau coût est inférieur à l'ancien) Alors
  Changer son coût total
  Changer son prédécesseur

FinSi

FinFaire

# Exemple 1

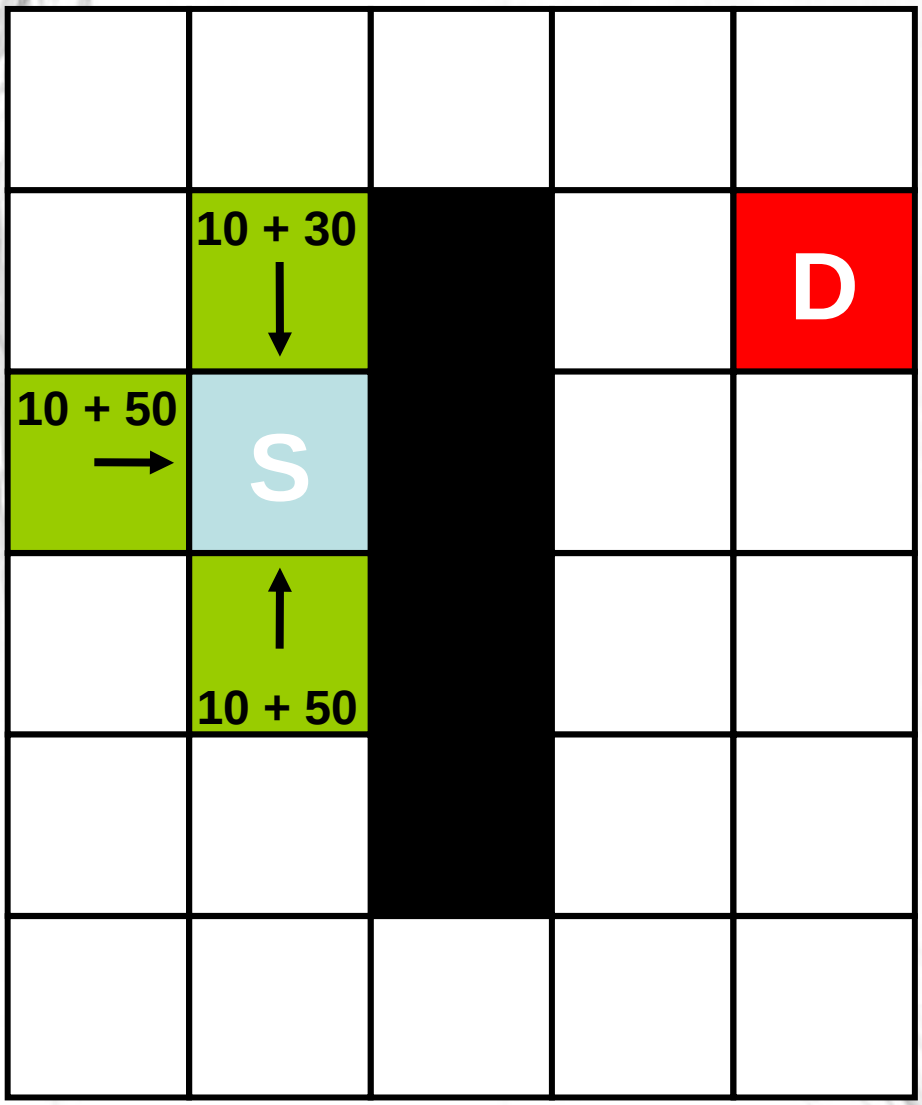


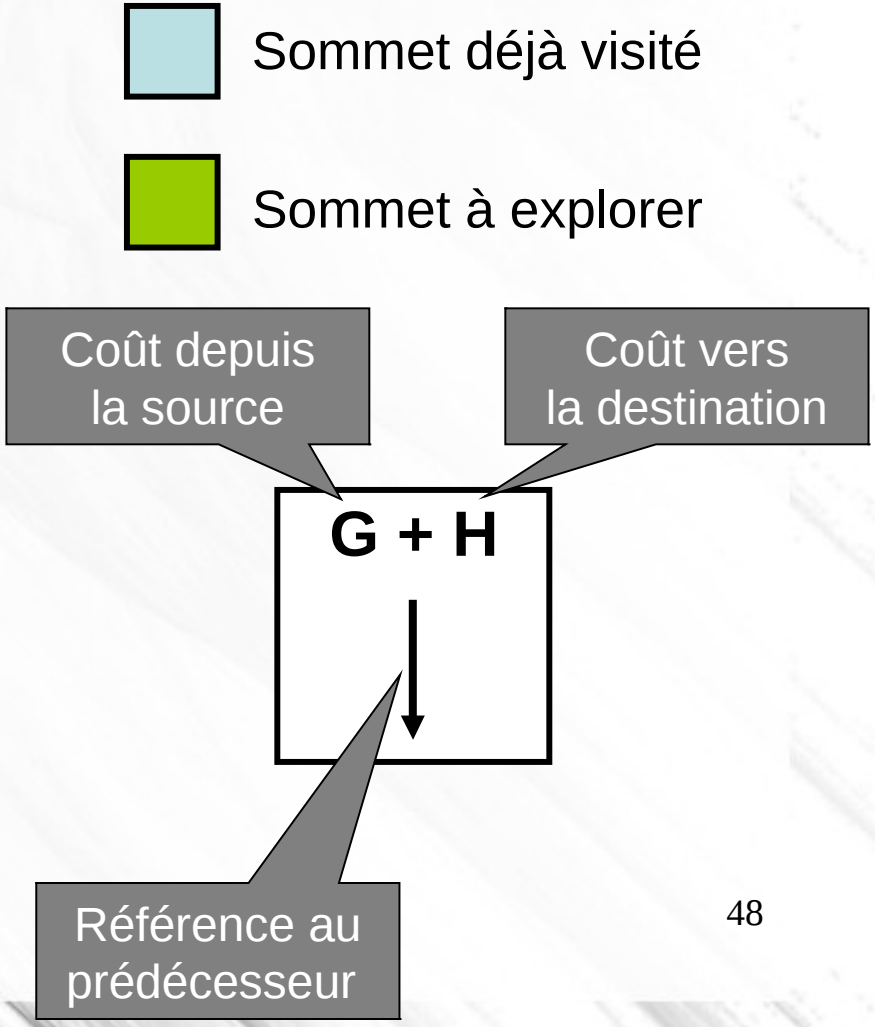

S Sommet source

D Sommet destination

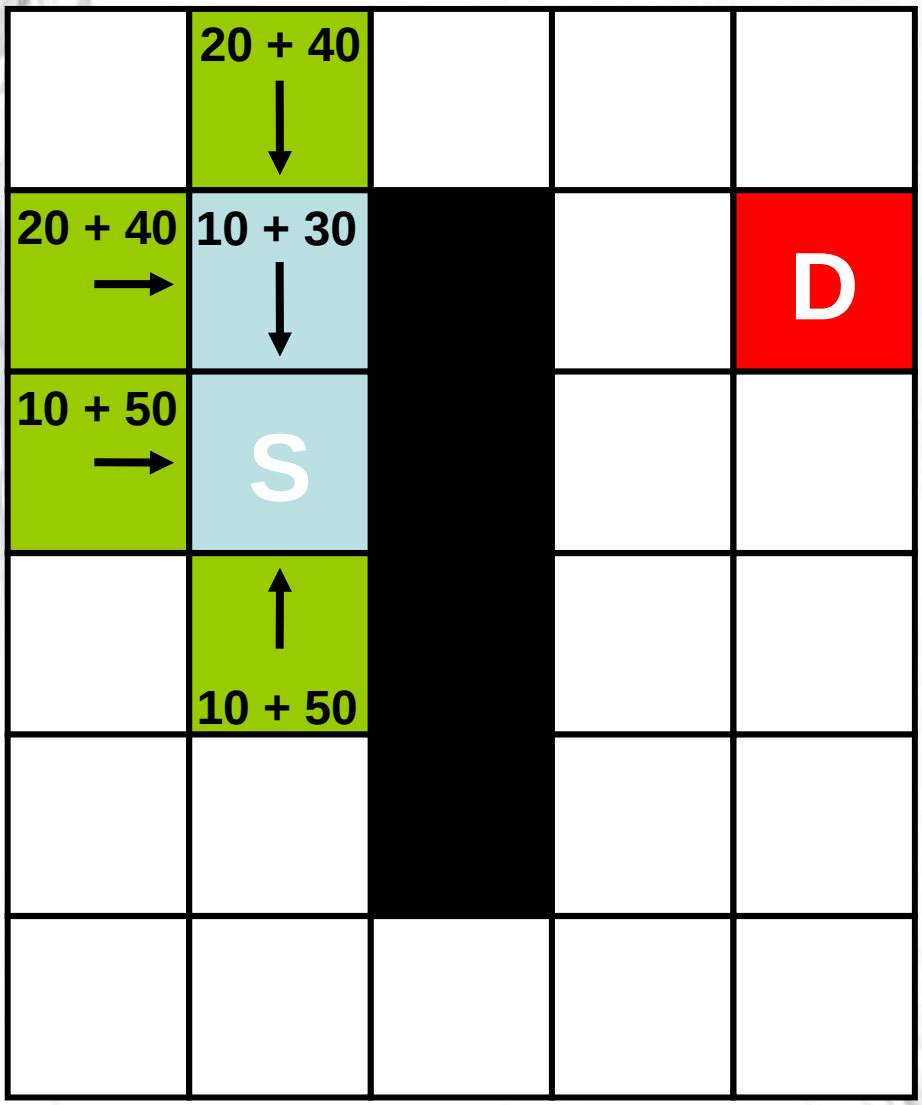
Obstacle

# Exemple 1


10 + 30
D
10 + 50
S
10 + 50


Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 1




Sommet déjà visité

Sommet à explorer

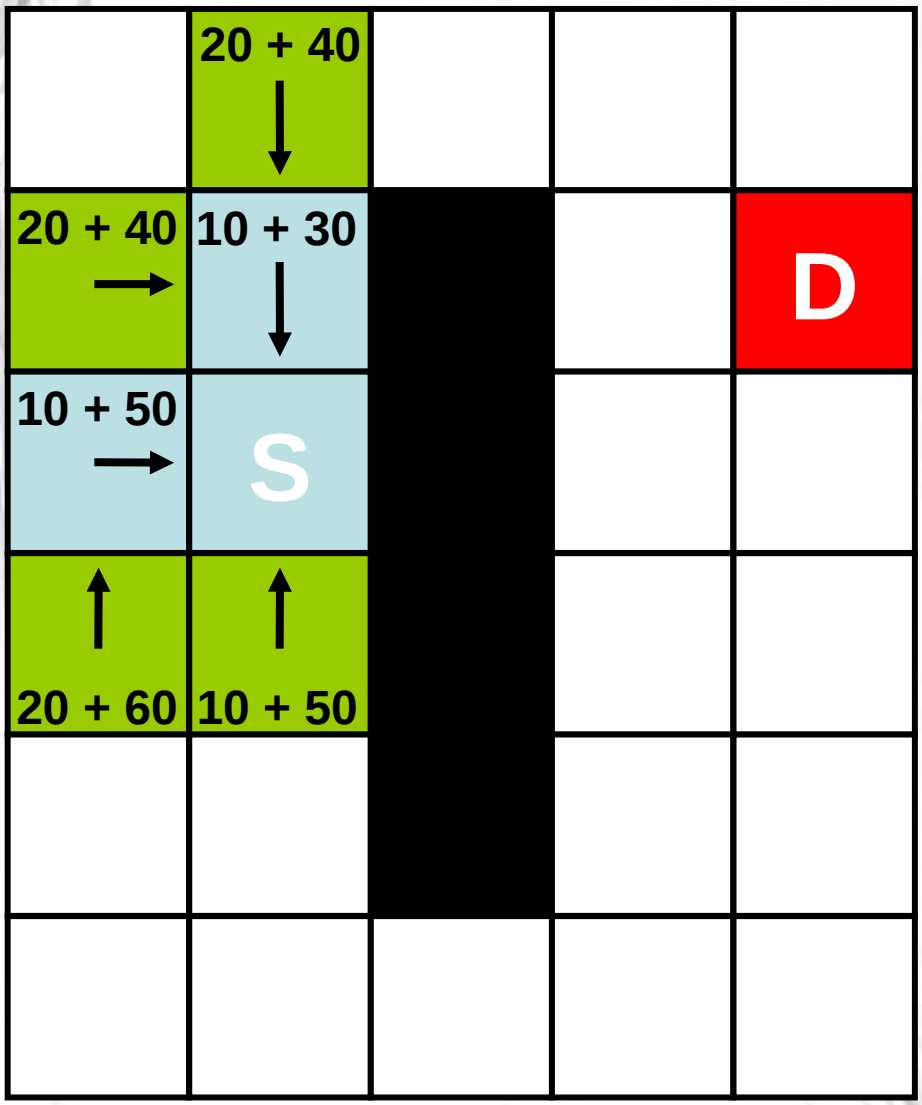
Coût depuis la source

Coût vers la destination

G + H

Référence au prédécesseur

# Exemple 1




Sommet déjà visité

Sommet à explorer

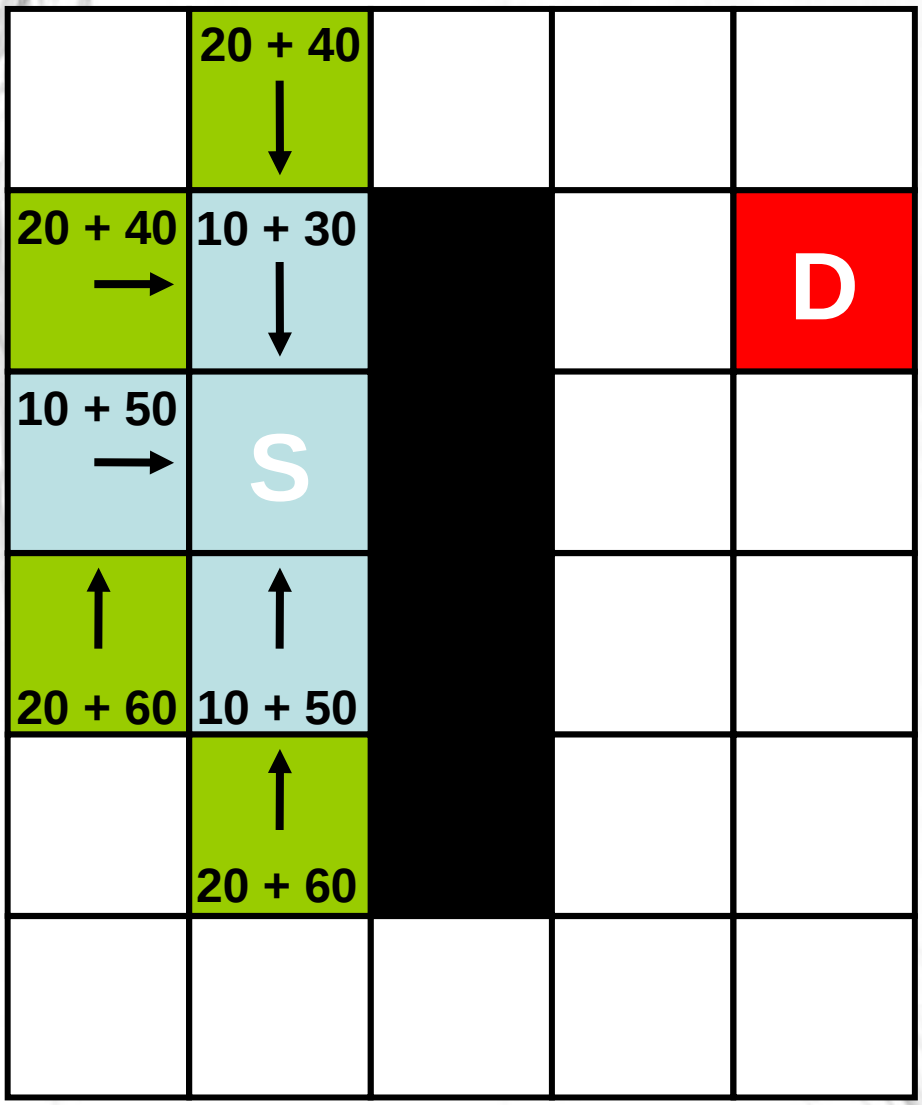
Coût depuis la source

Coût vers la destination

G + H

Référence au prédécesseur

# Exemple 1




Sommet déjà visité

Sommet à explorer

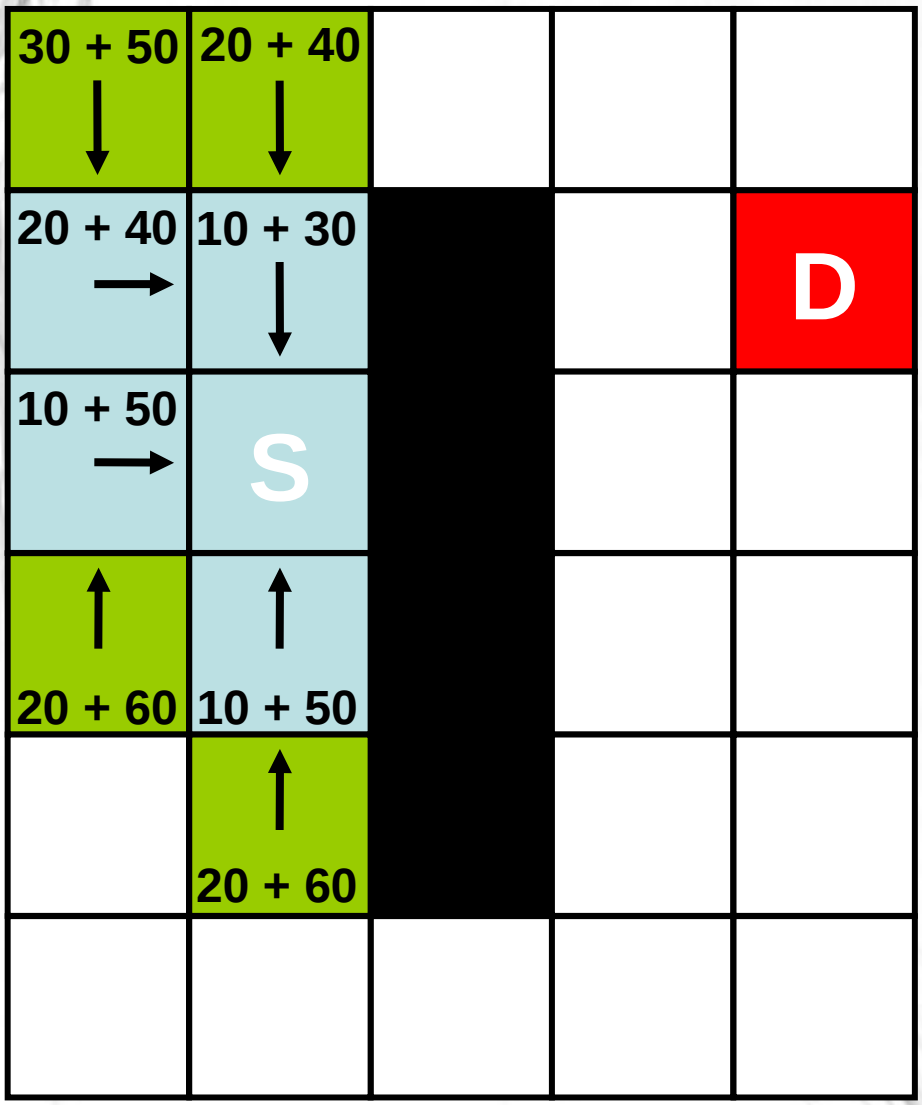
Coût depuis la source

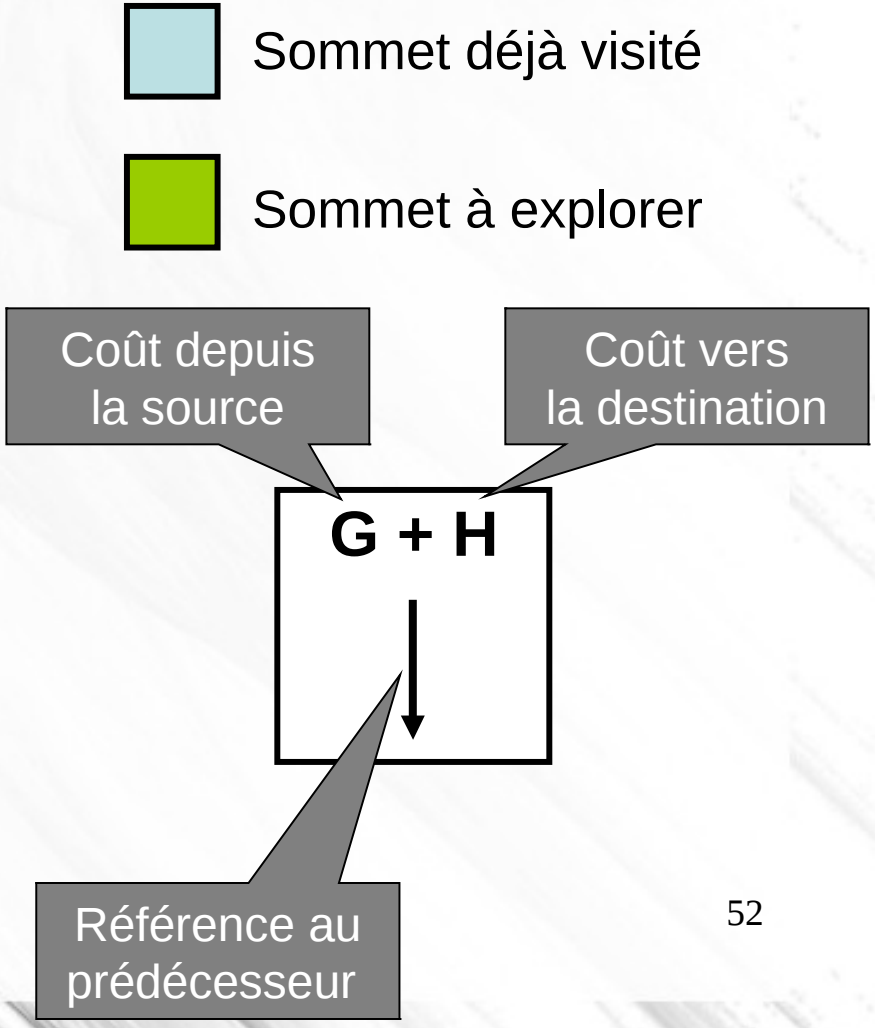
Coût vers la destination

G + H

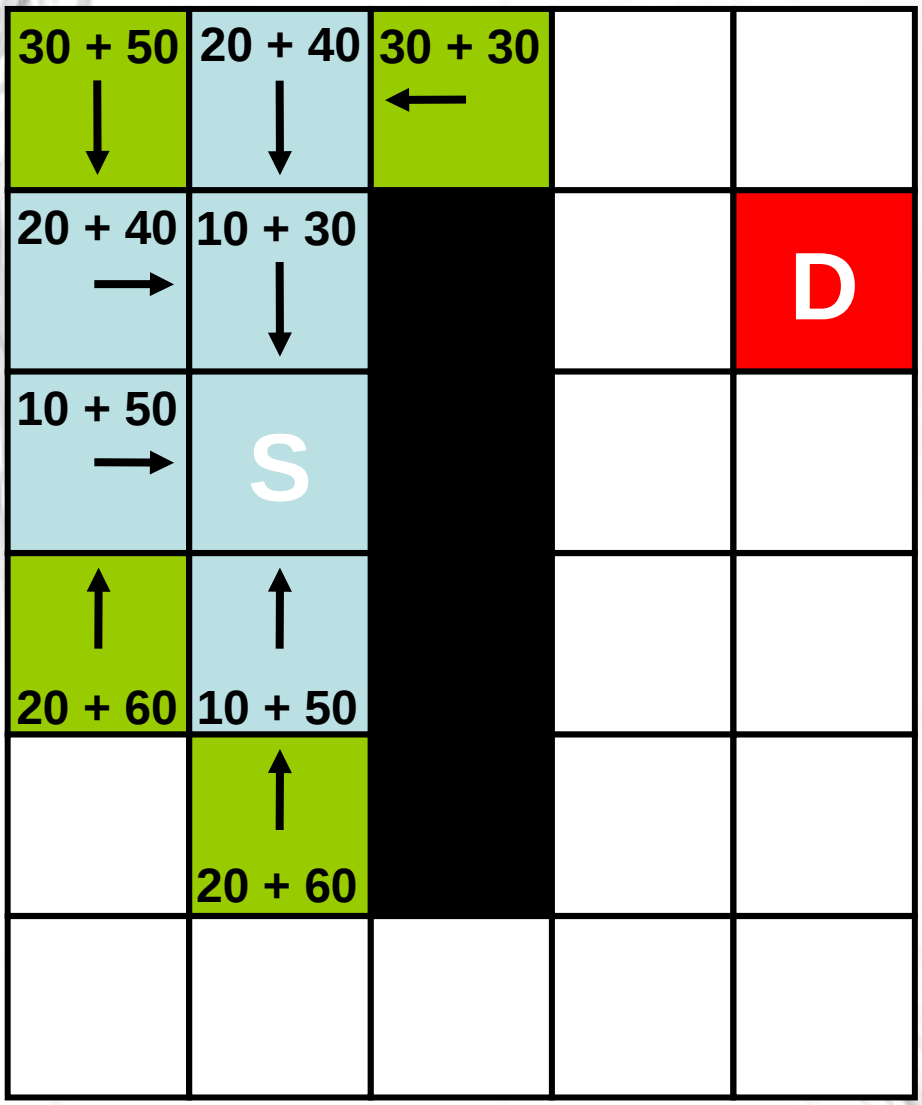
Référence au prédécesseur

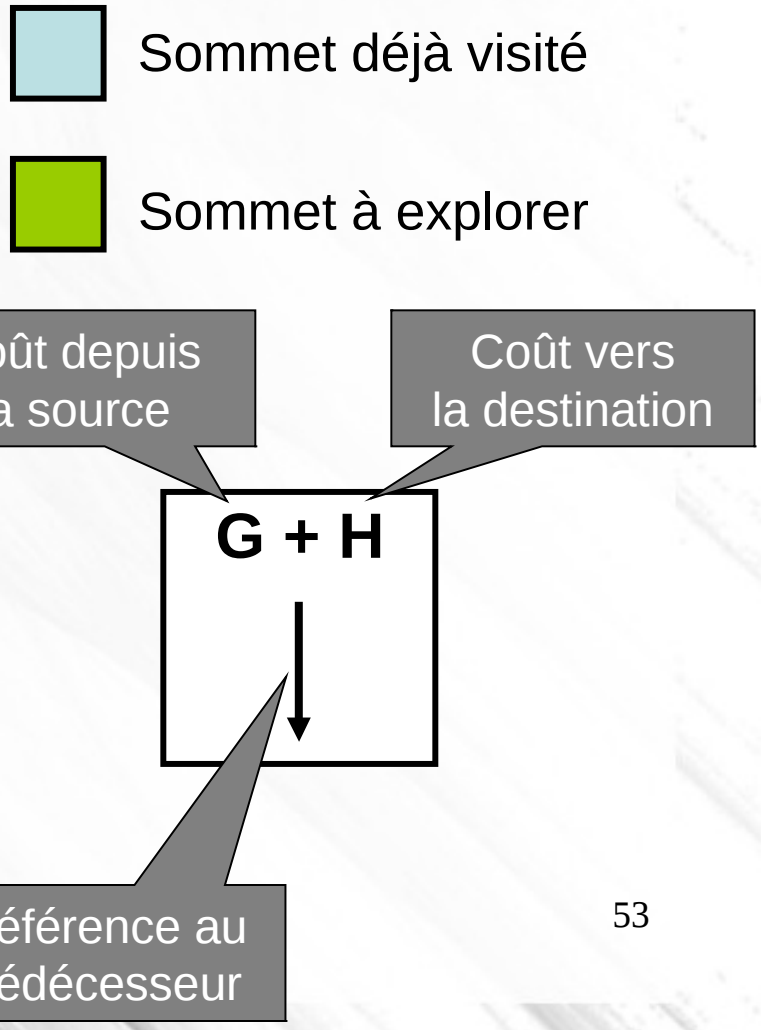
# Exemple 1


30 + 50
20 + 40
20 + 40
10 + 30
D
10 + 50
S
20 + 60
10 + 50
20 + 60

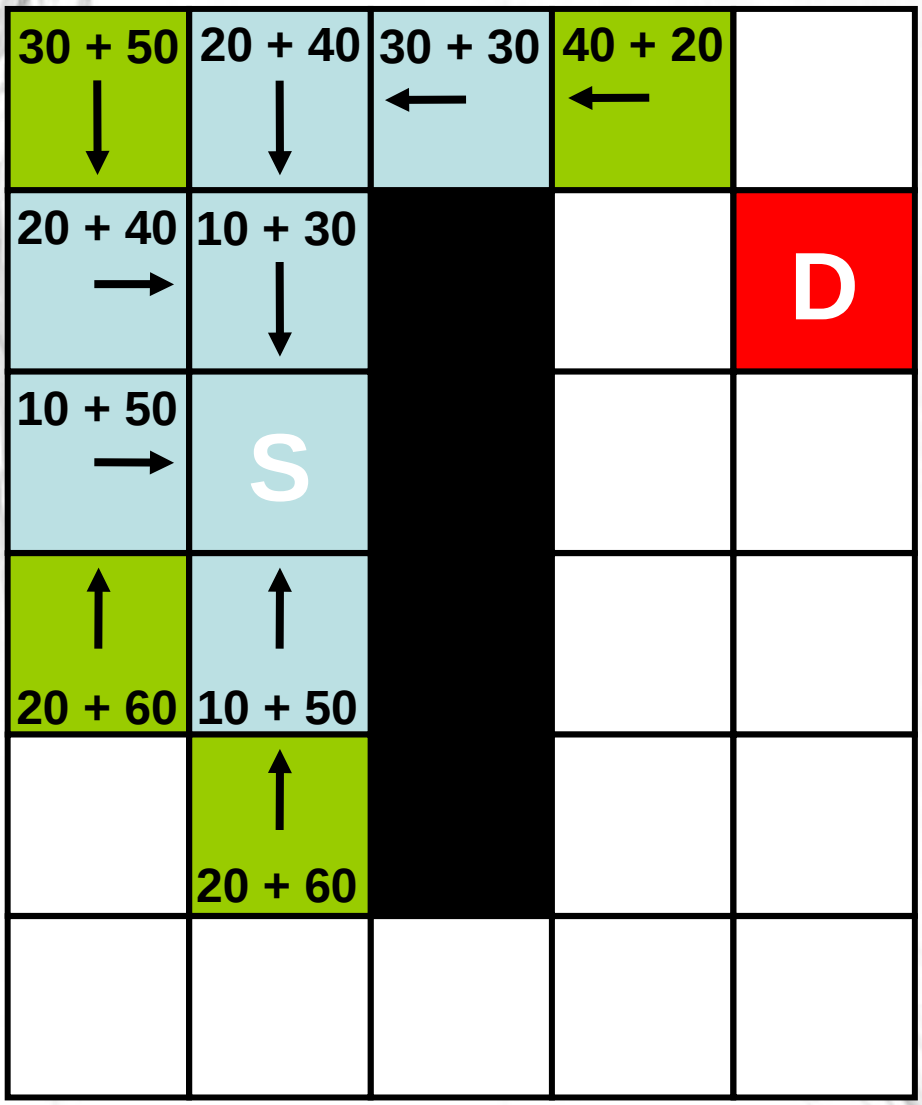

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 1


30 + 50
20 + 40
30 + 30
20 + 40
10 + 30
D
10 + 50
S
20 + 60
10 + 50
20 + 60

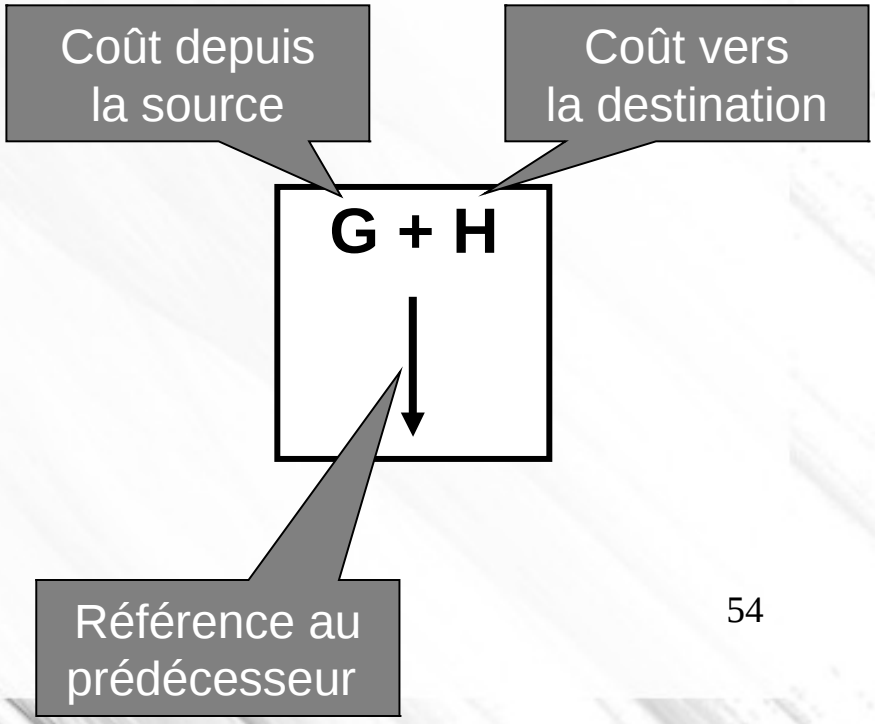

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 1

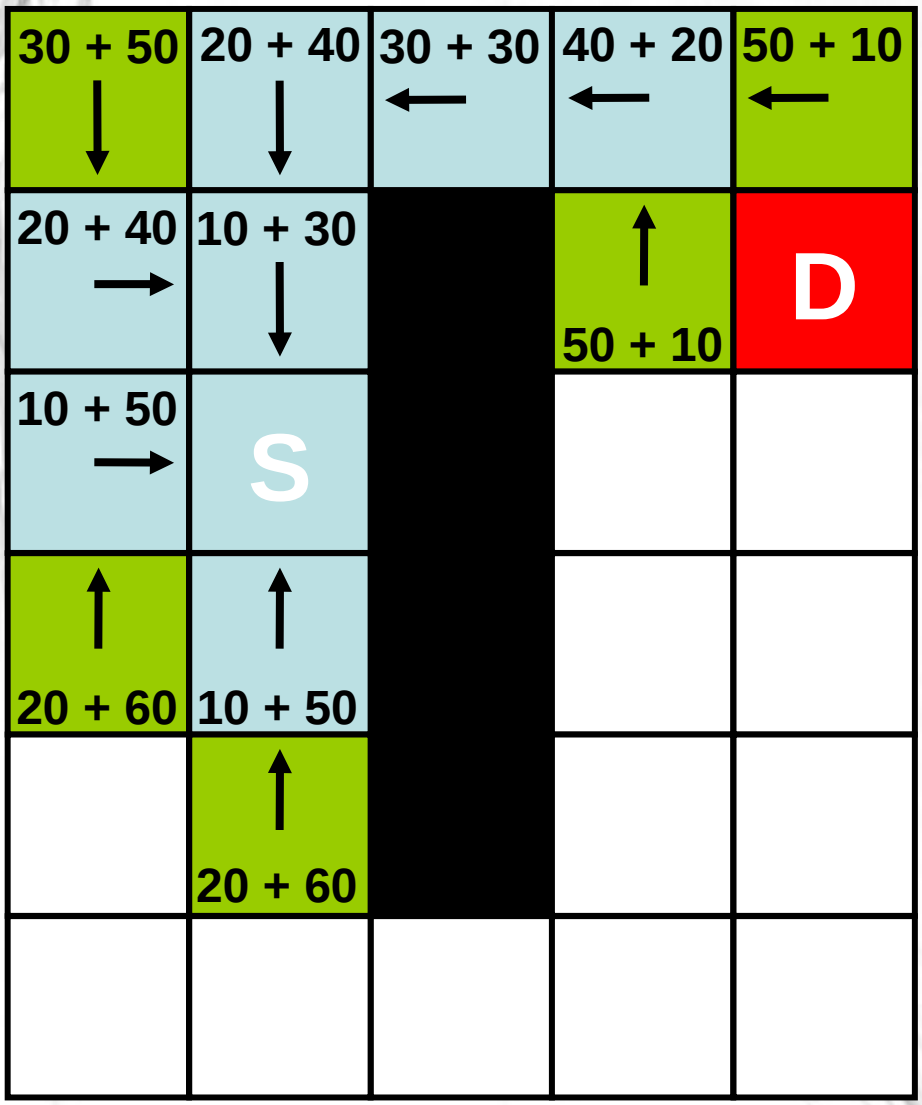


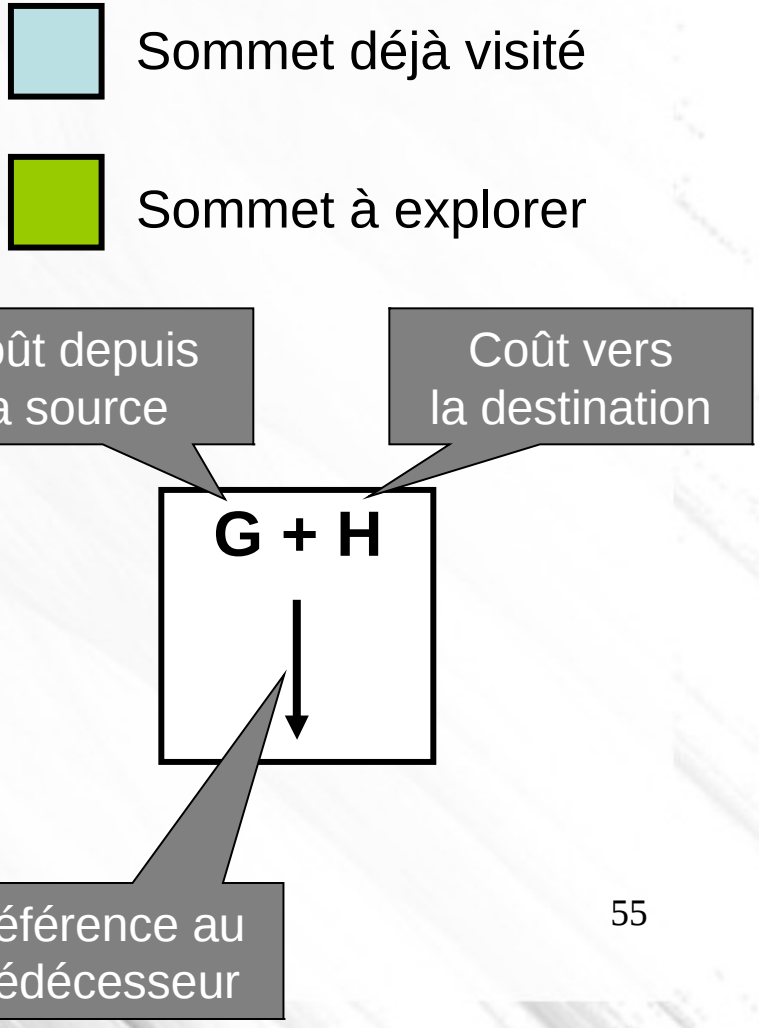

Sommet déjà visité

Sommet à explorer

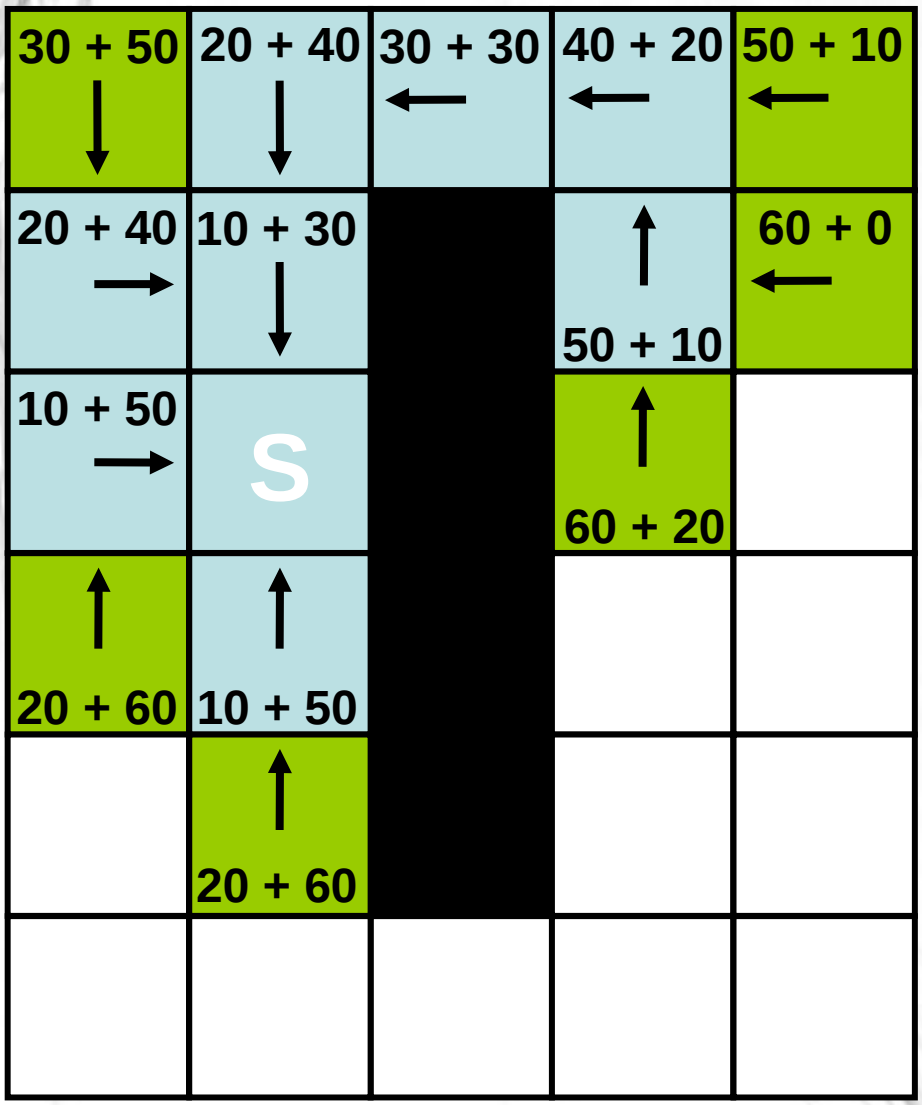


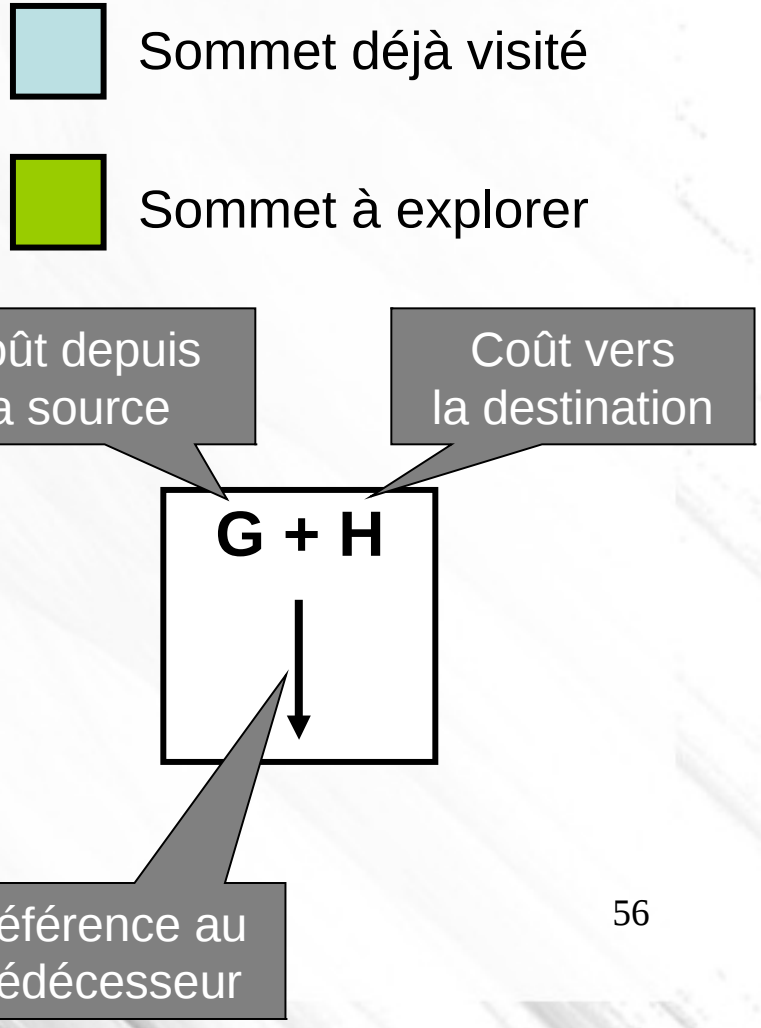
# Exemple 1


30 + 50
20 + 40
30 + 30
40 + 20
50 + 10
20 + 40
10 + 30
50 + 10
D
10 + 50
S
20 + 60
10 + 50
20 + 60

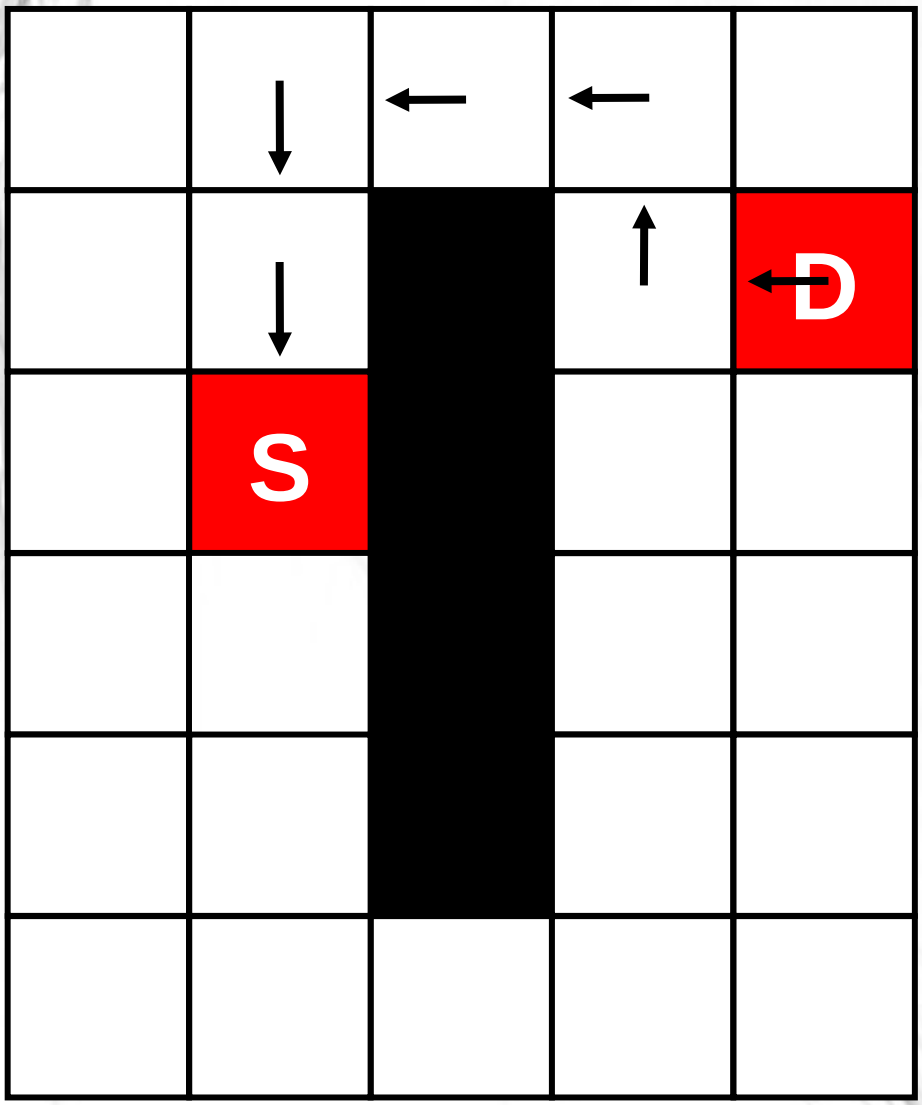

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 1

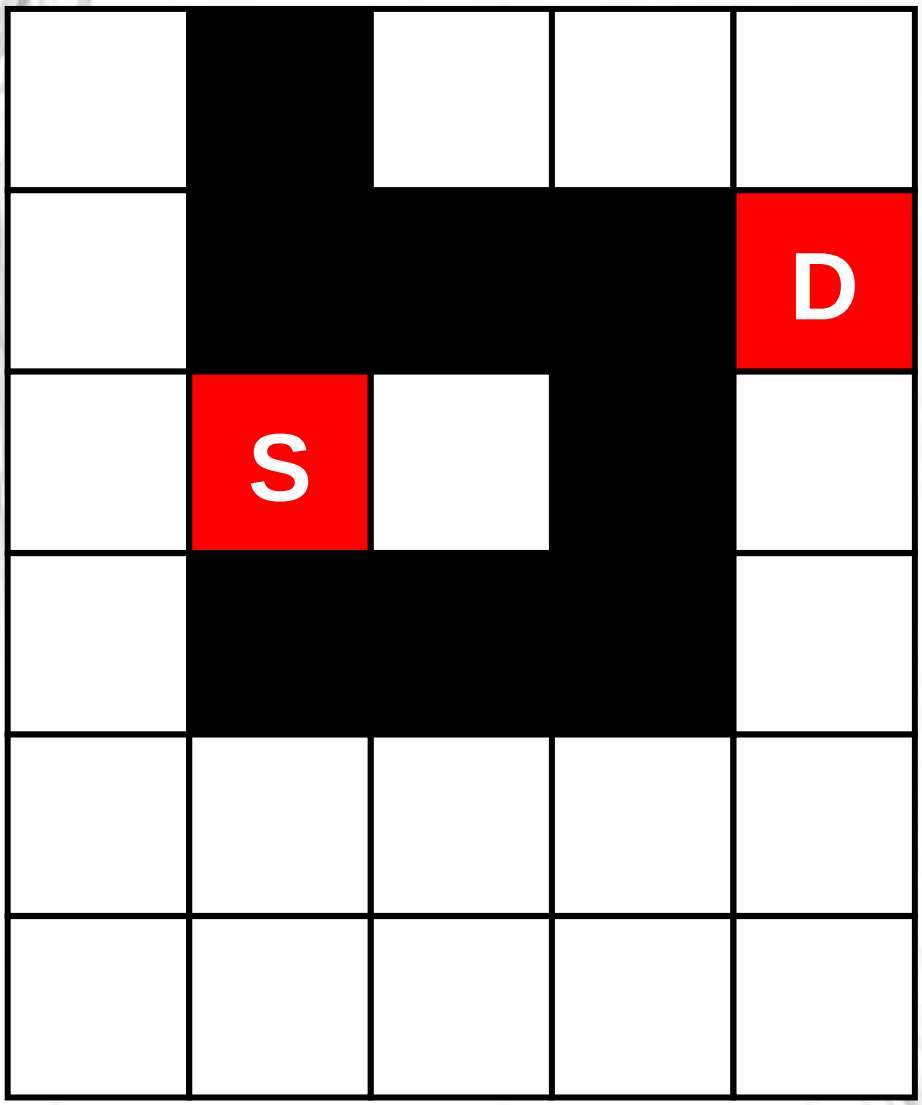

30 + 50
20 + 40
30 + 30
40 + 20
50 + 10
20 + 40
10 + 30
50 + 10
60 + 0
10 + 50
S
60 + 20
20 + 60
10 + 50
20 + 60


Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 1



# Exemple 2

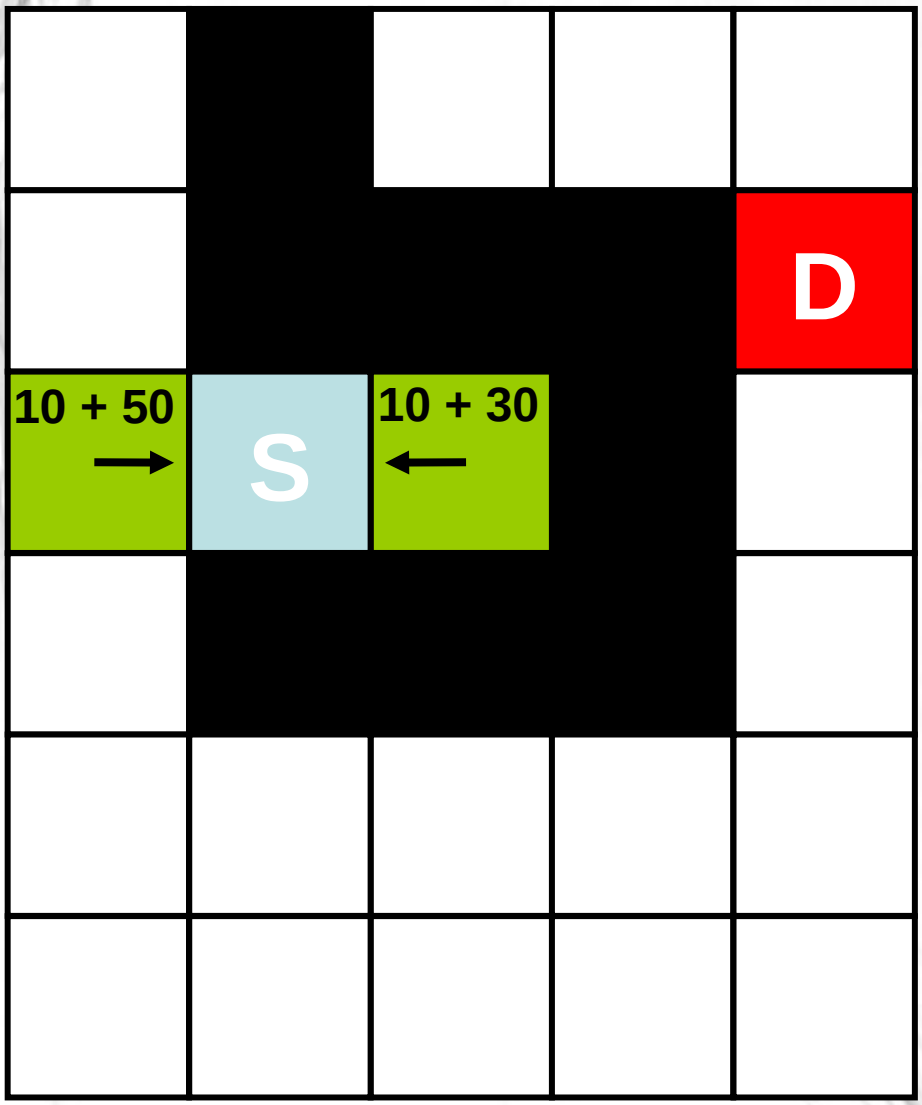


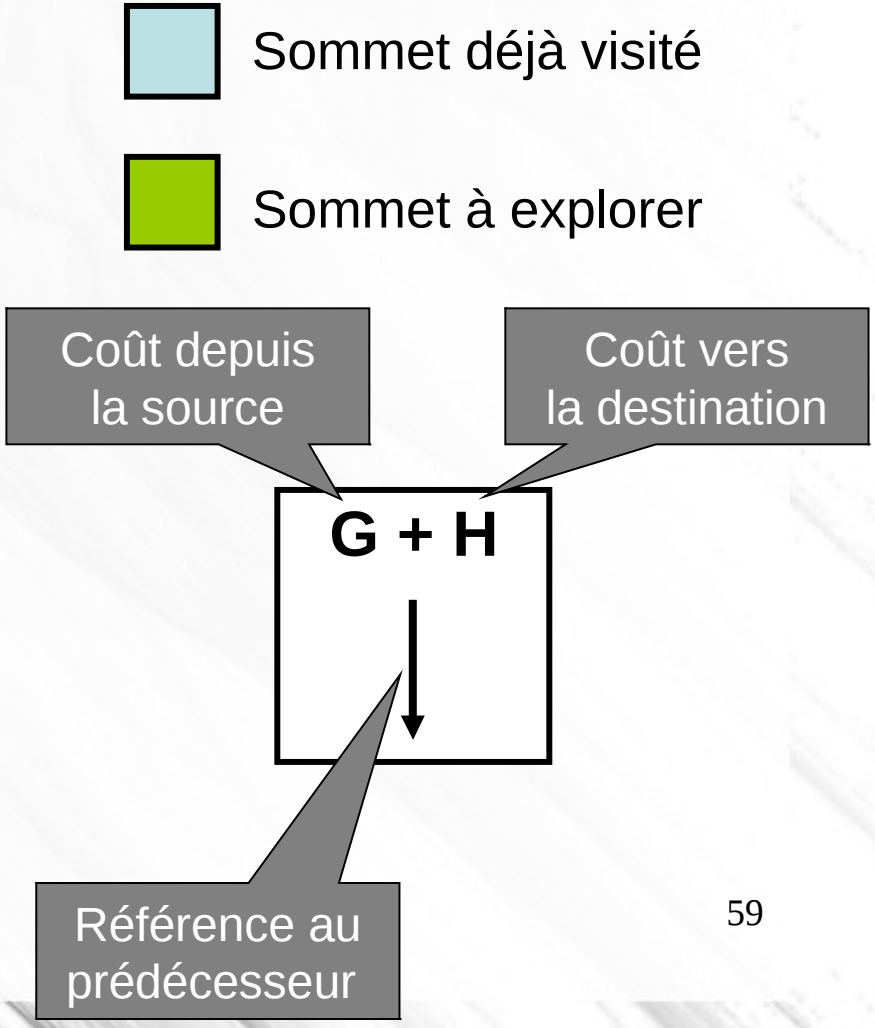

S Sommet source

D Sommet destination

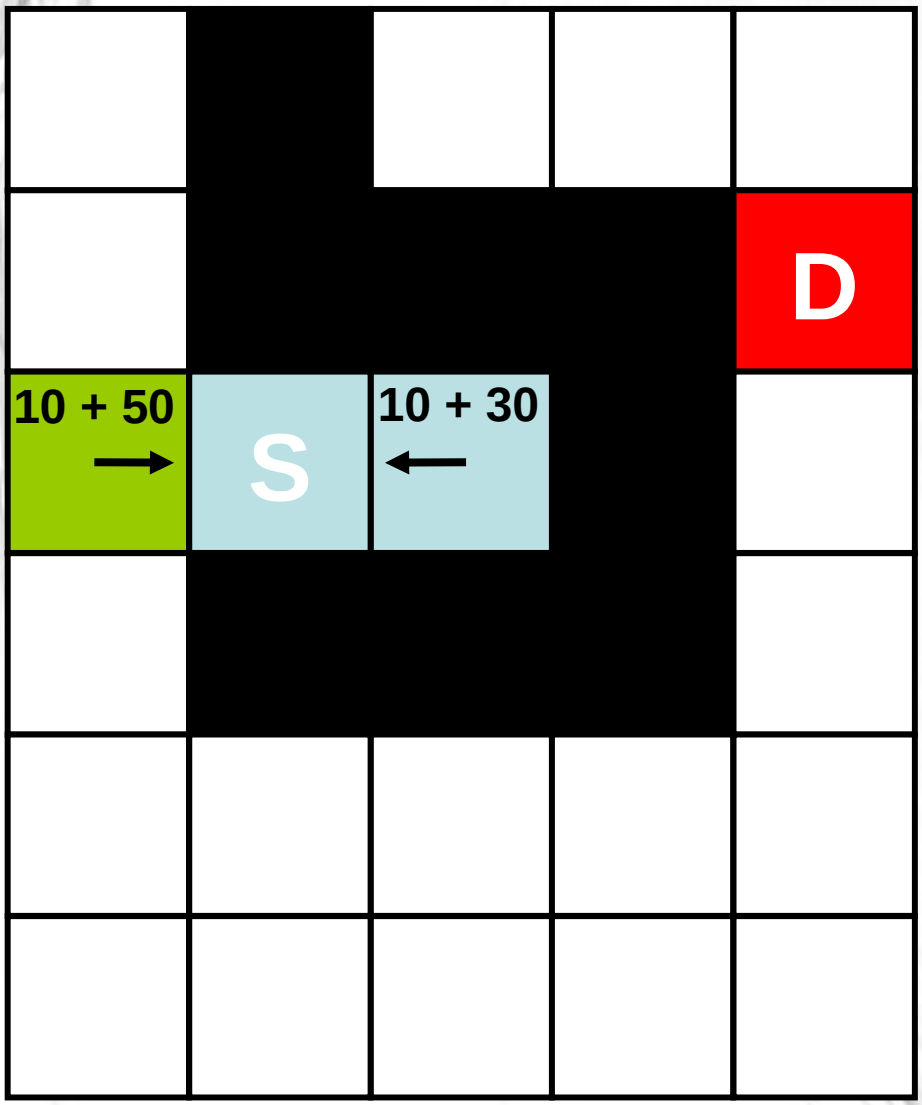
Obstacle

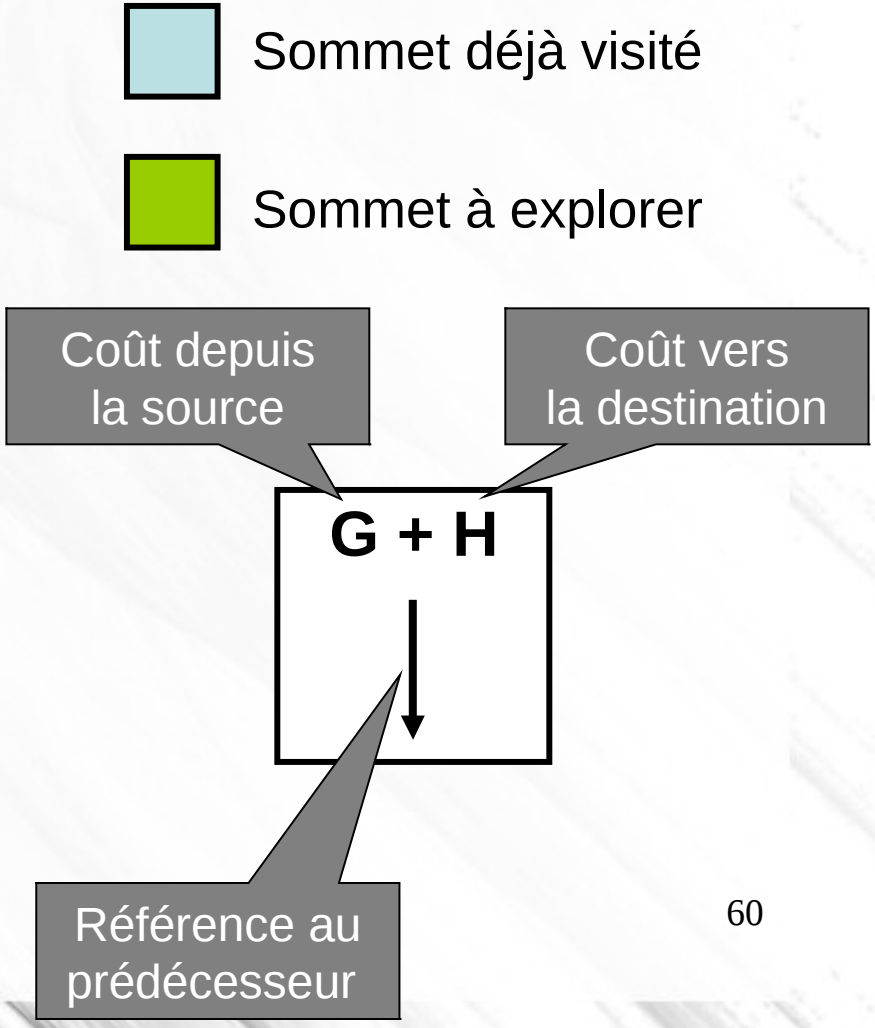
# Exemple 2


D
10 + 50
S
10 + 30

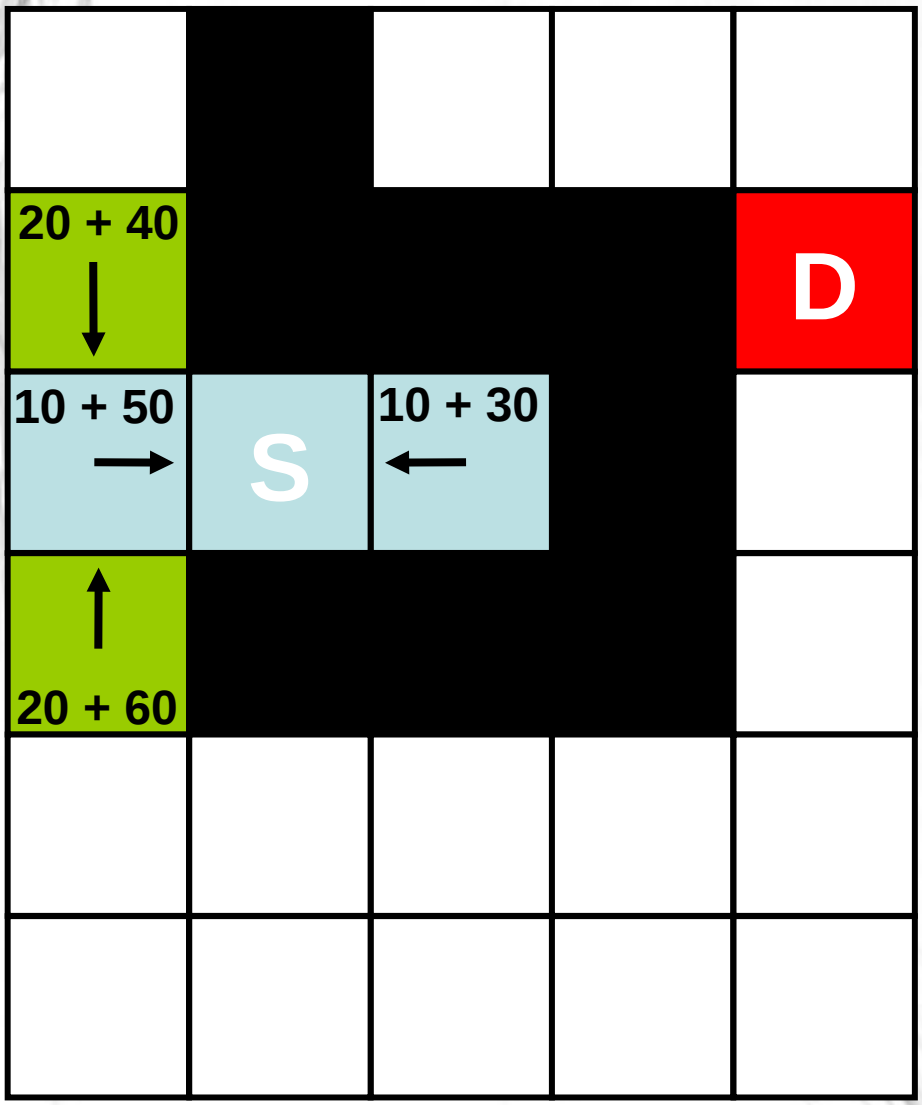

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

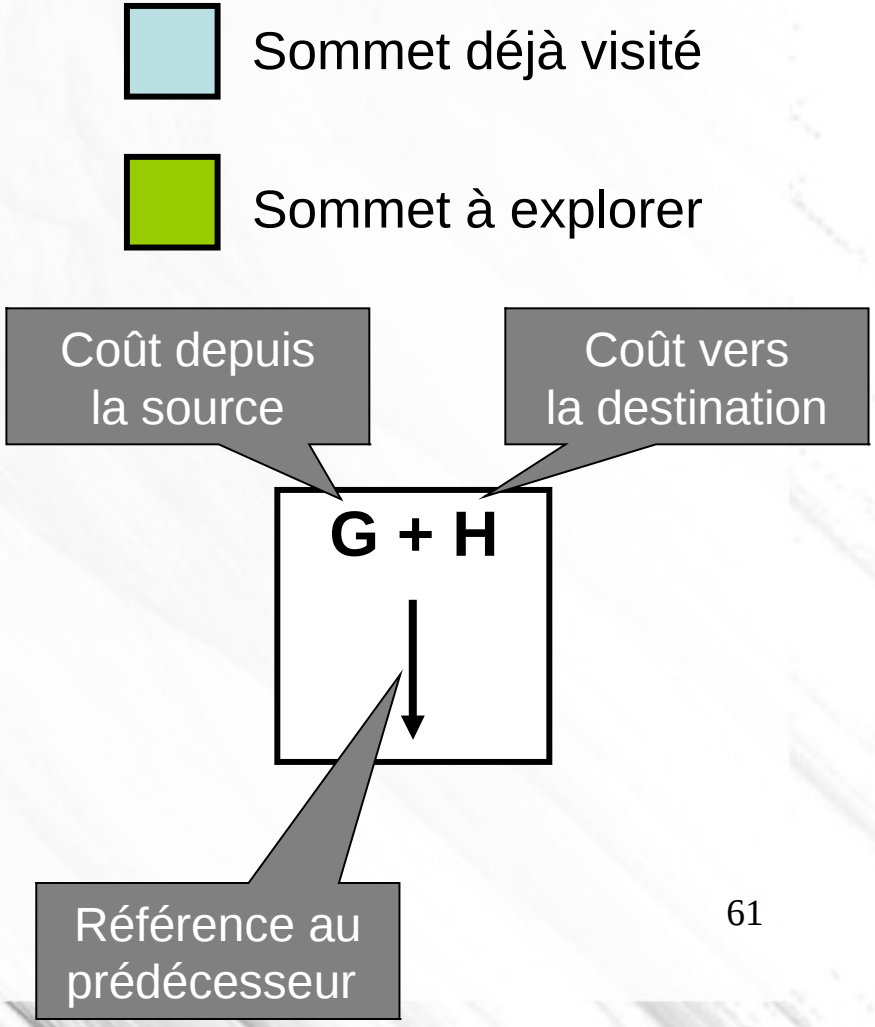
# Exemple 2


D
10 + 50
S
10 + 30

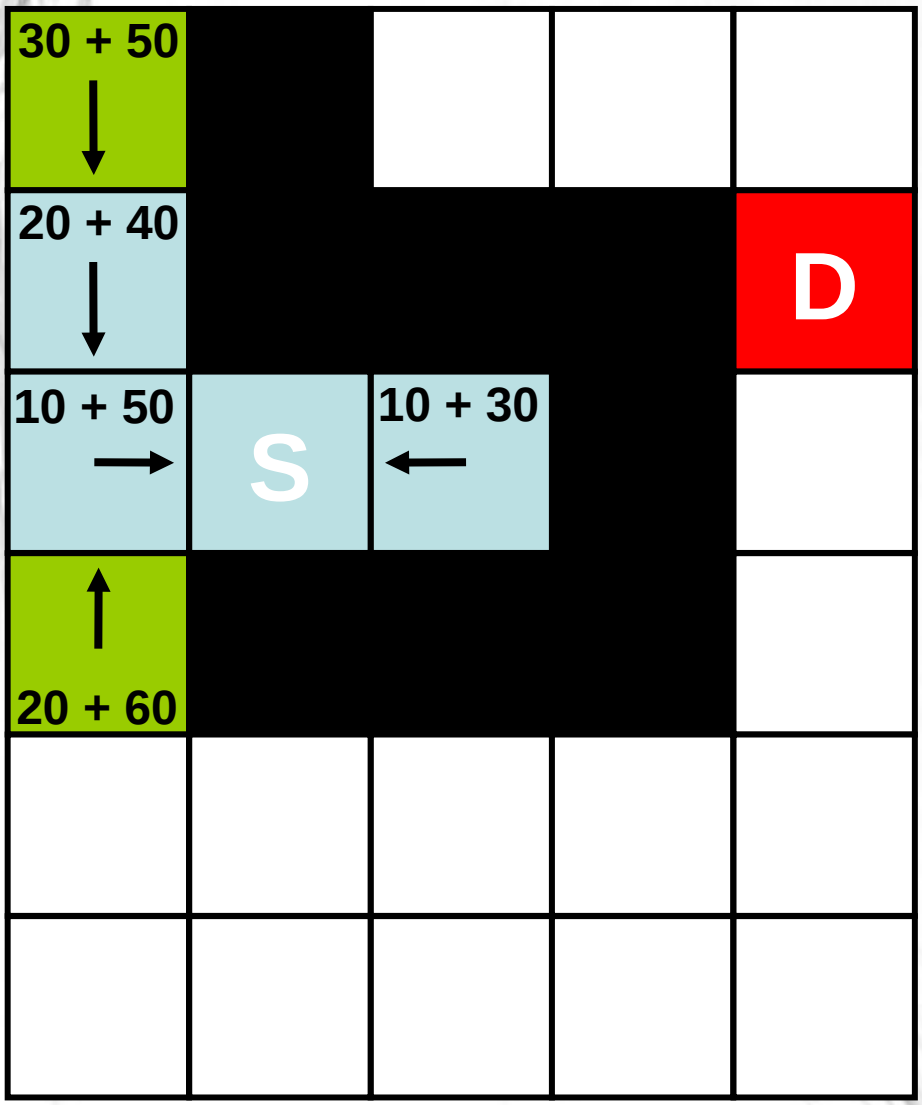

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

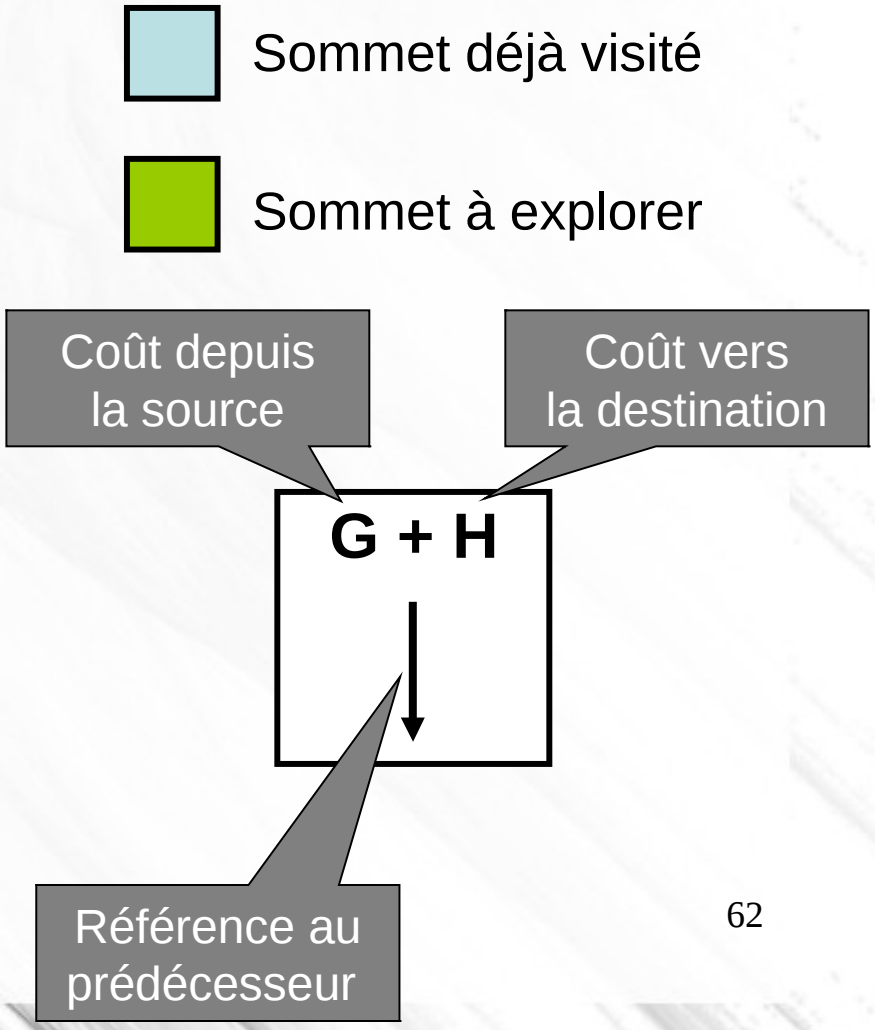
# Exemple 2


20 + 40
D
10 + 50
S
10 + 30
20 + 60

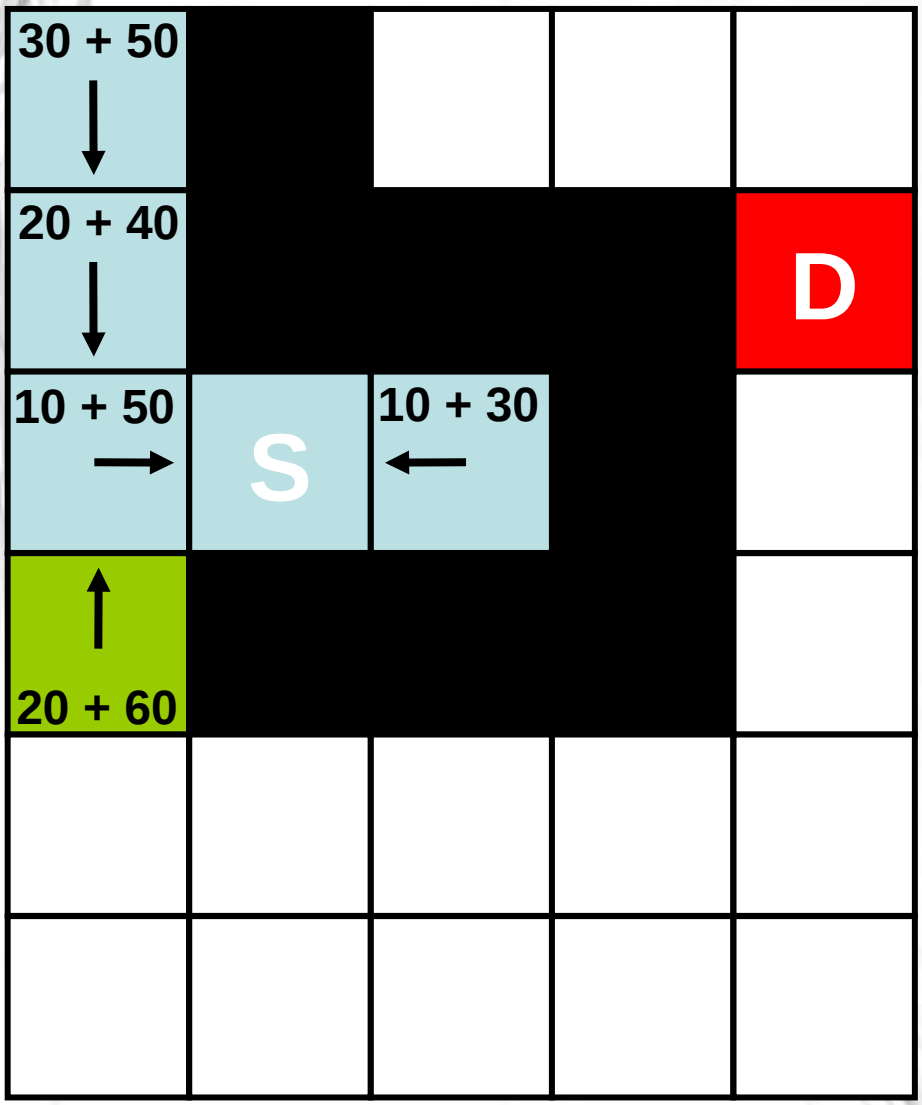

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

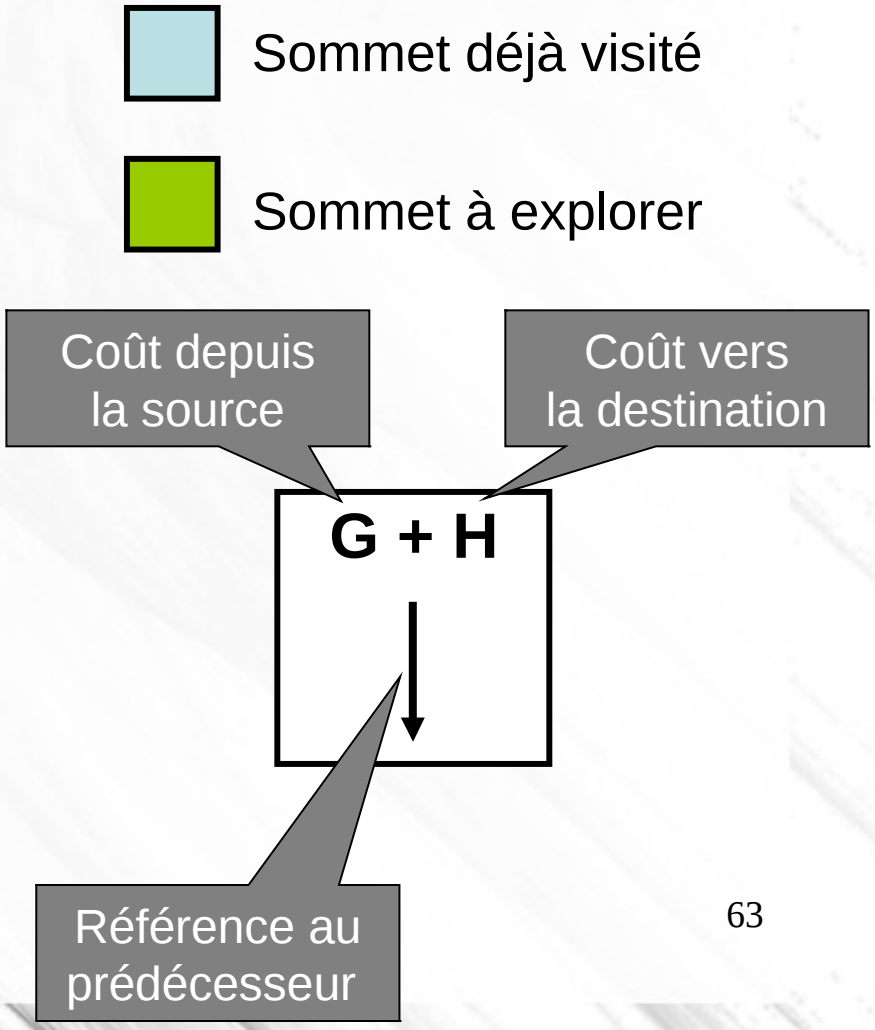
# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
20 + 60

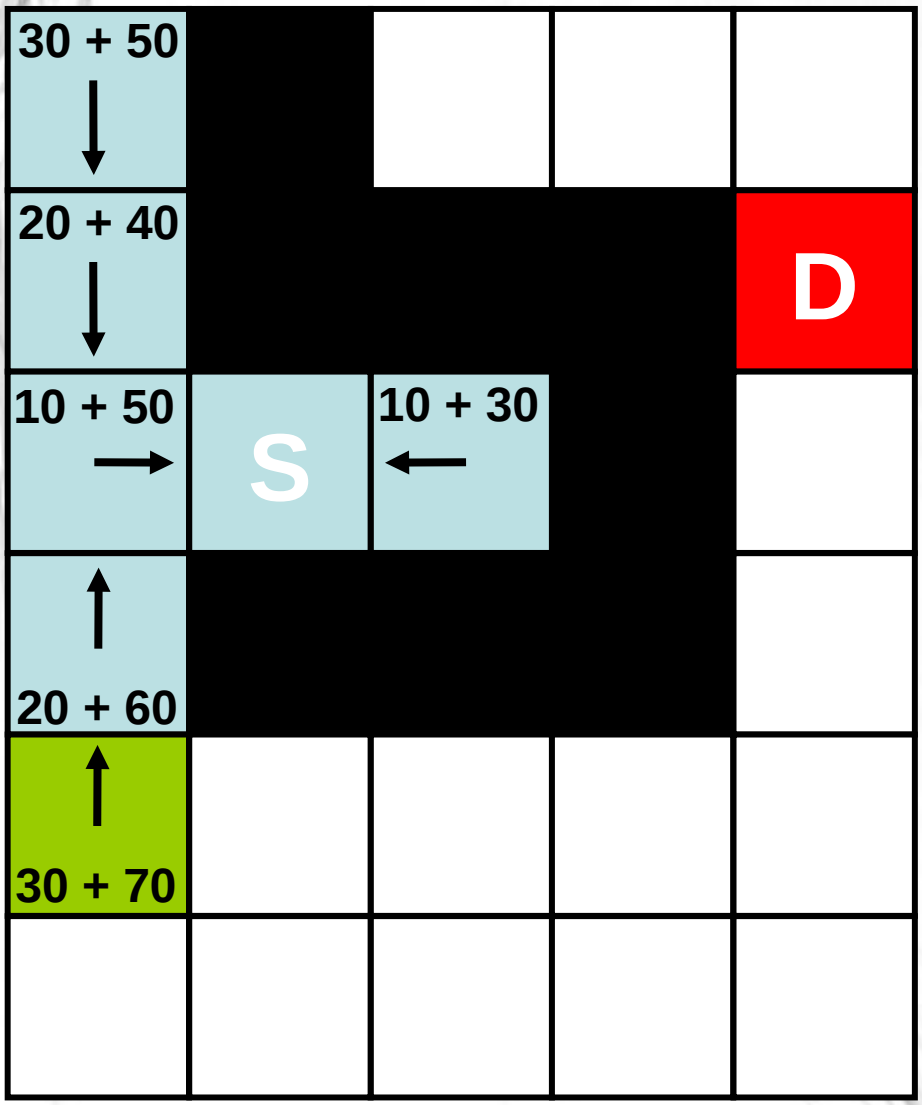

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

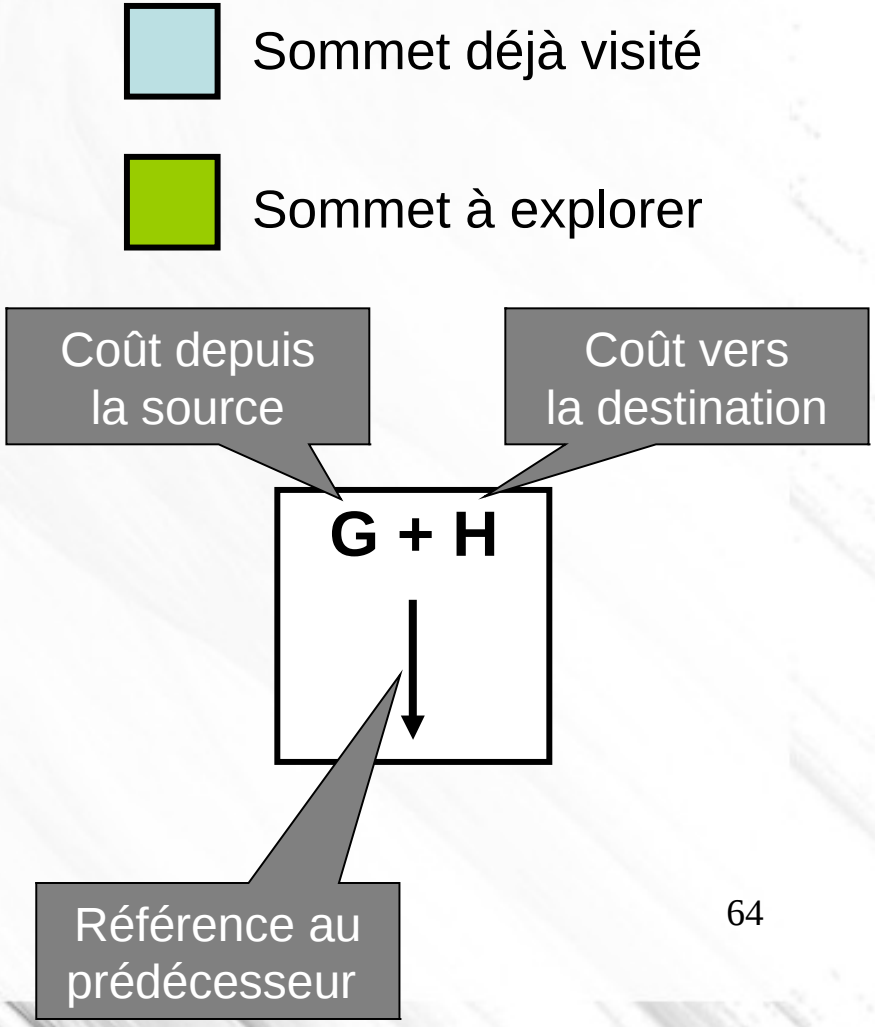
# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
20 + 60

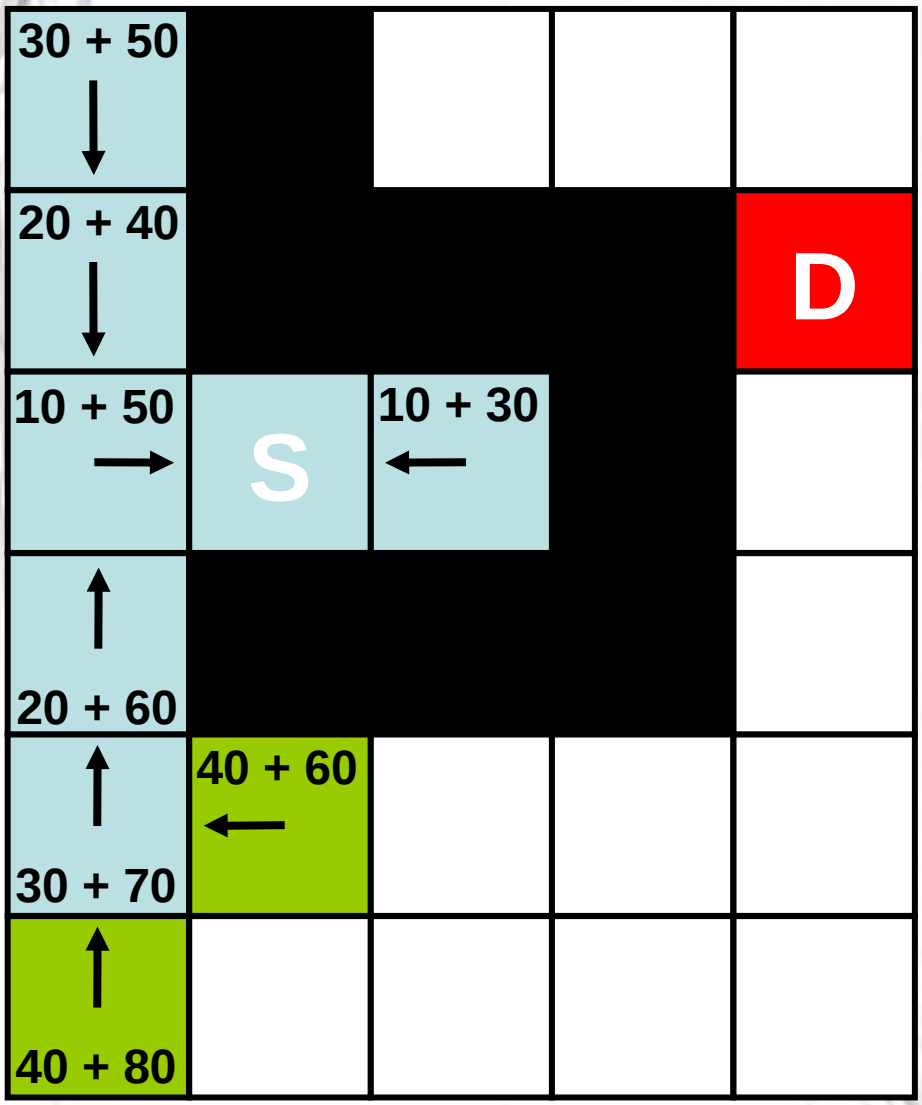

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

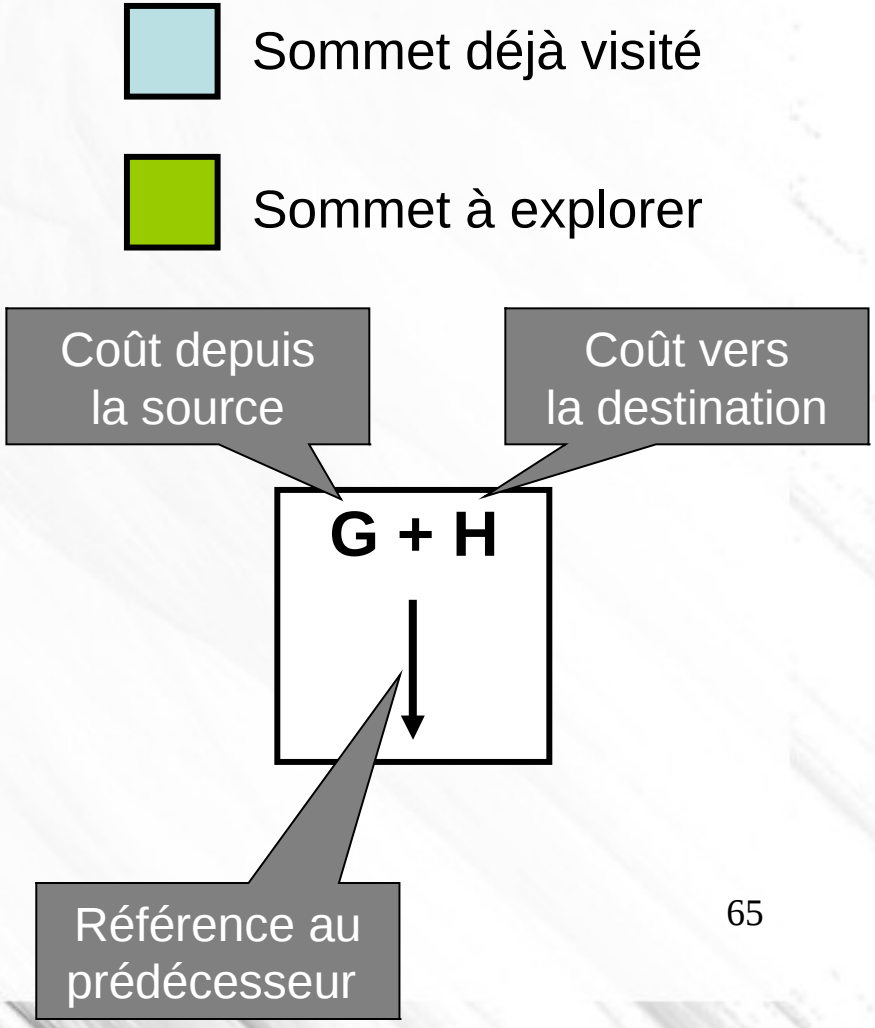
# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
20 + 60
30 + 70

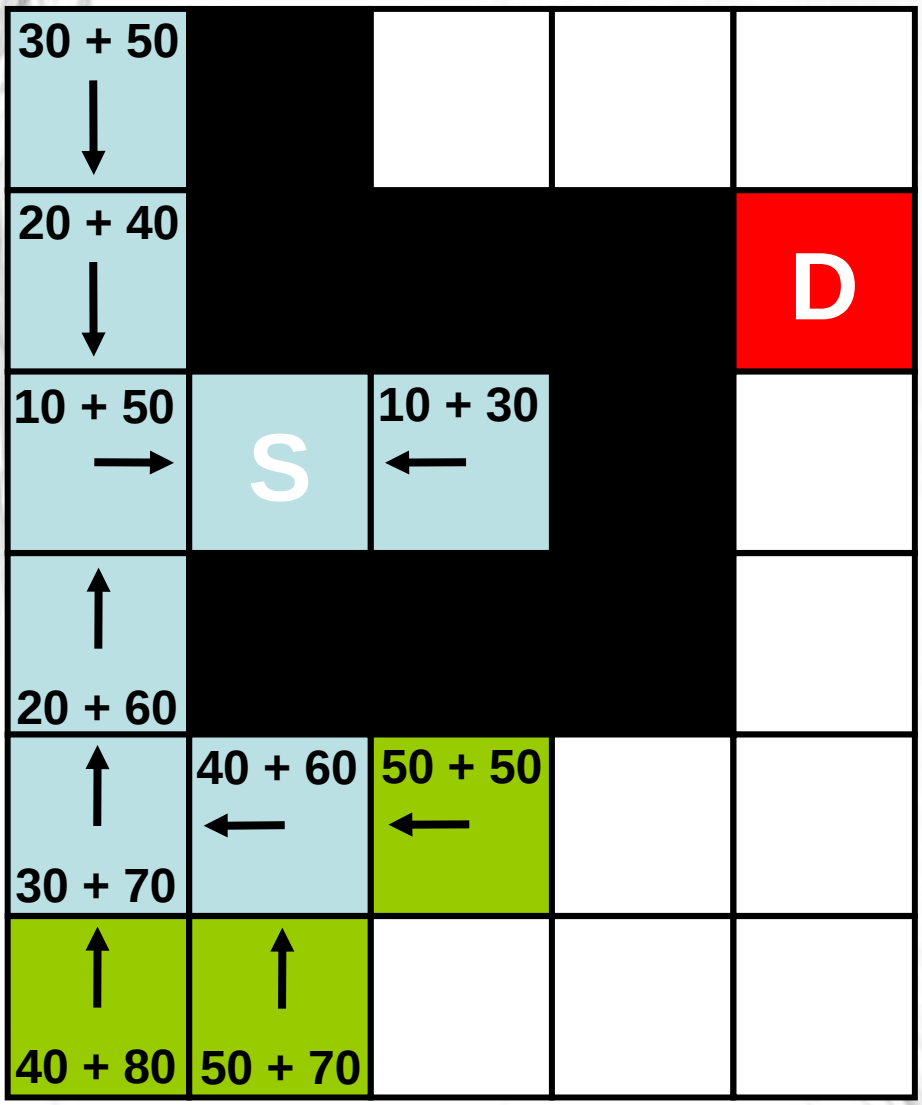

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
20 + 60
40 + 60
30 + 70
40 + 80

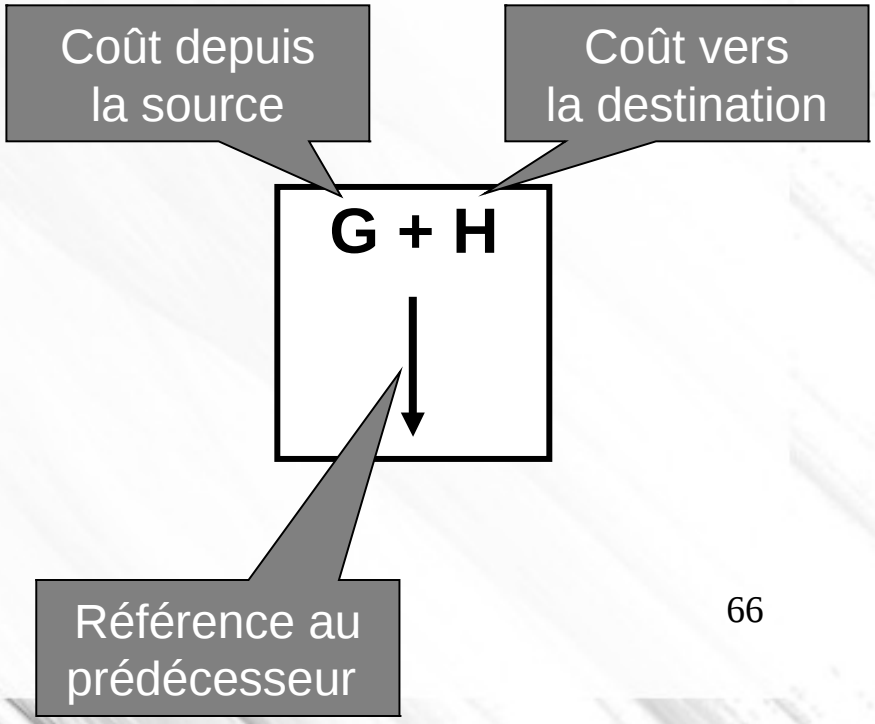

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 2

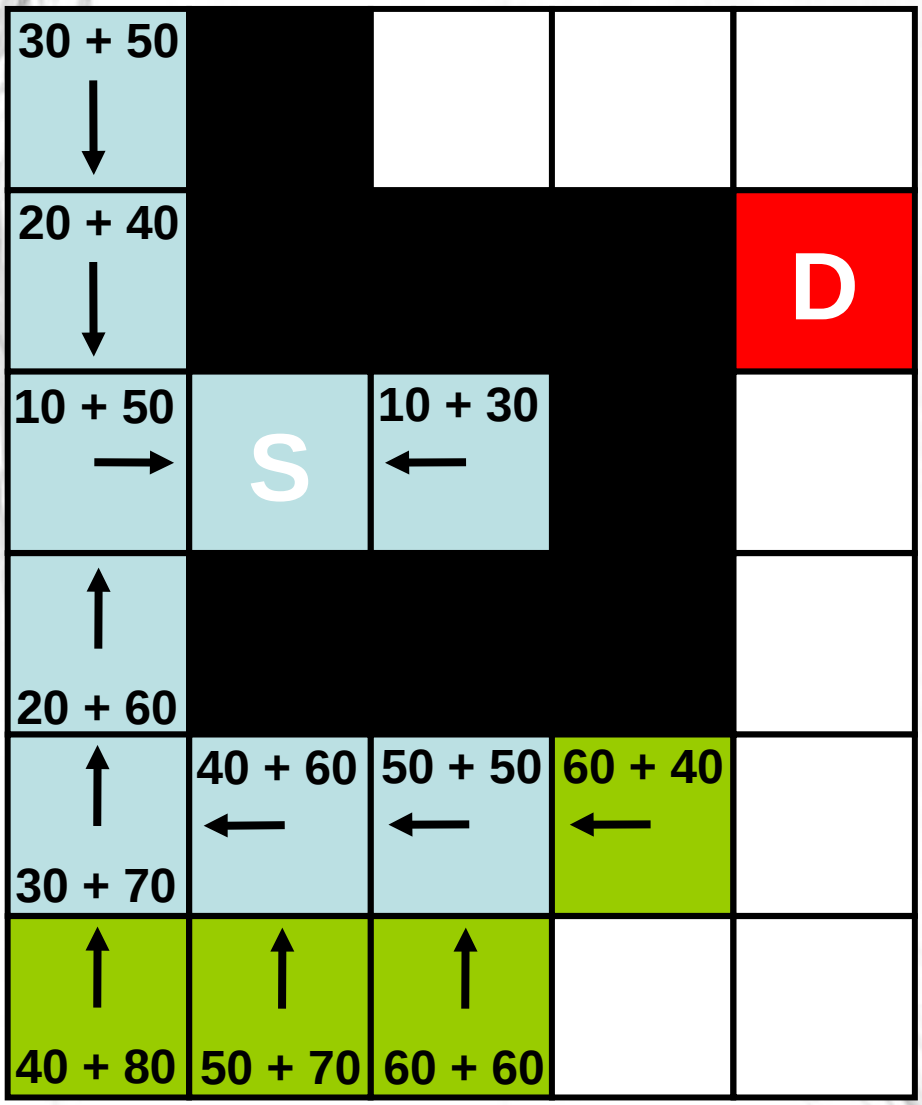


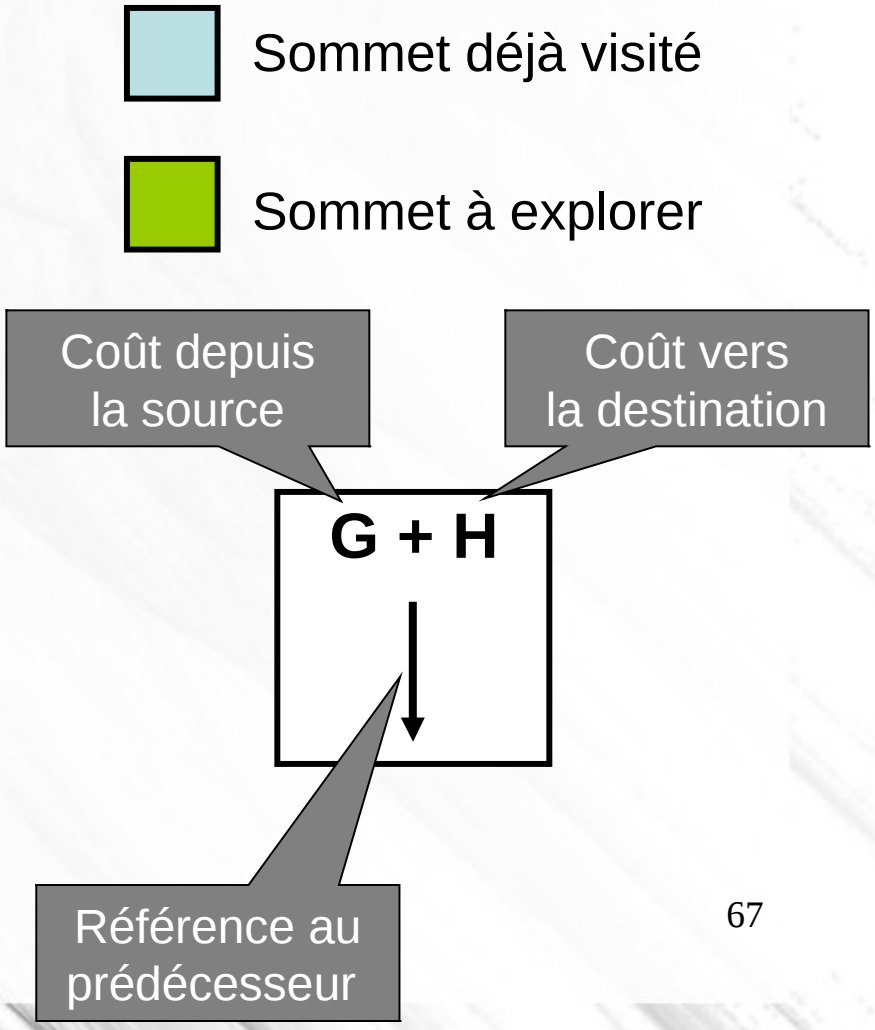

Sommet déjà visité

Sommet à explorer

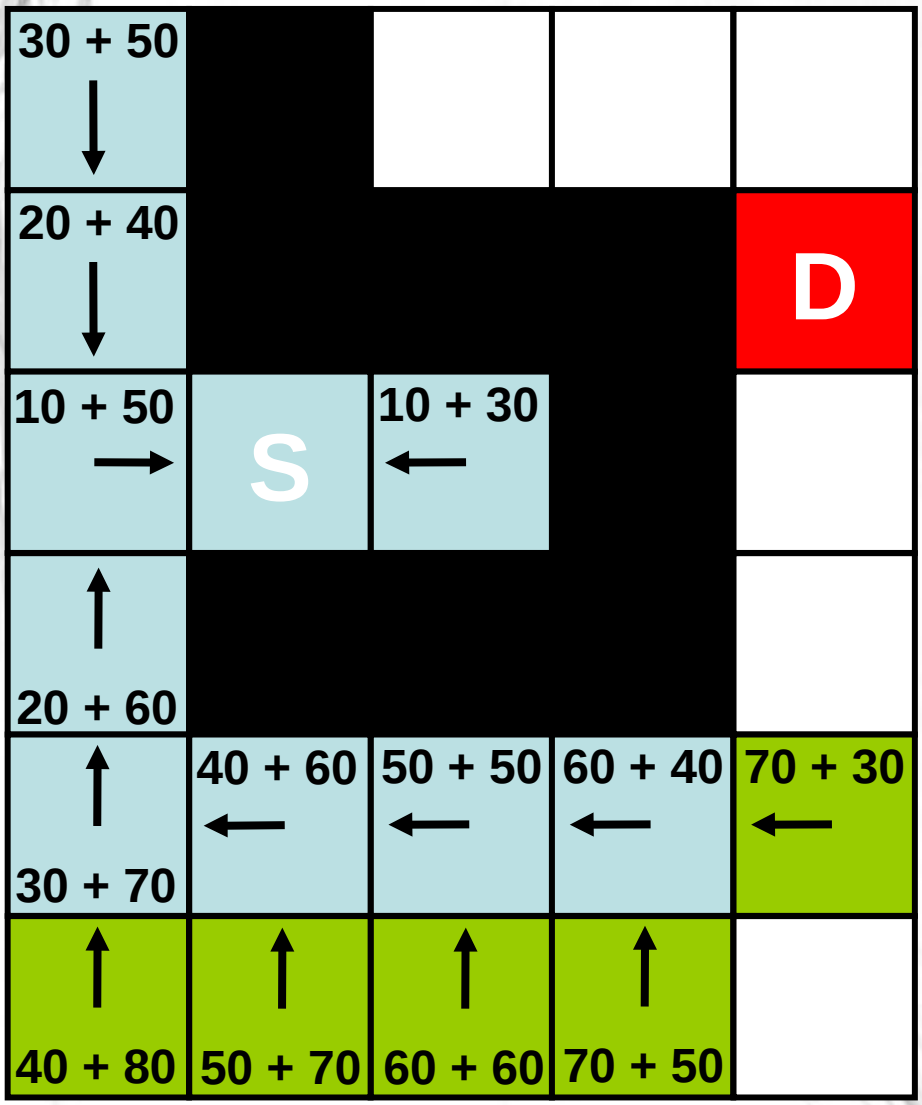


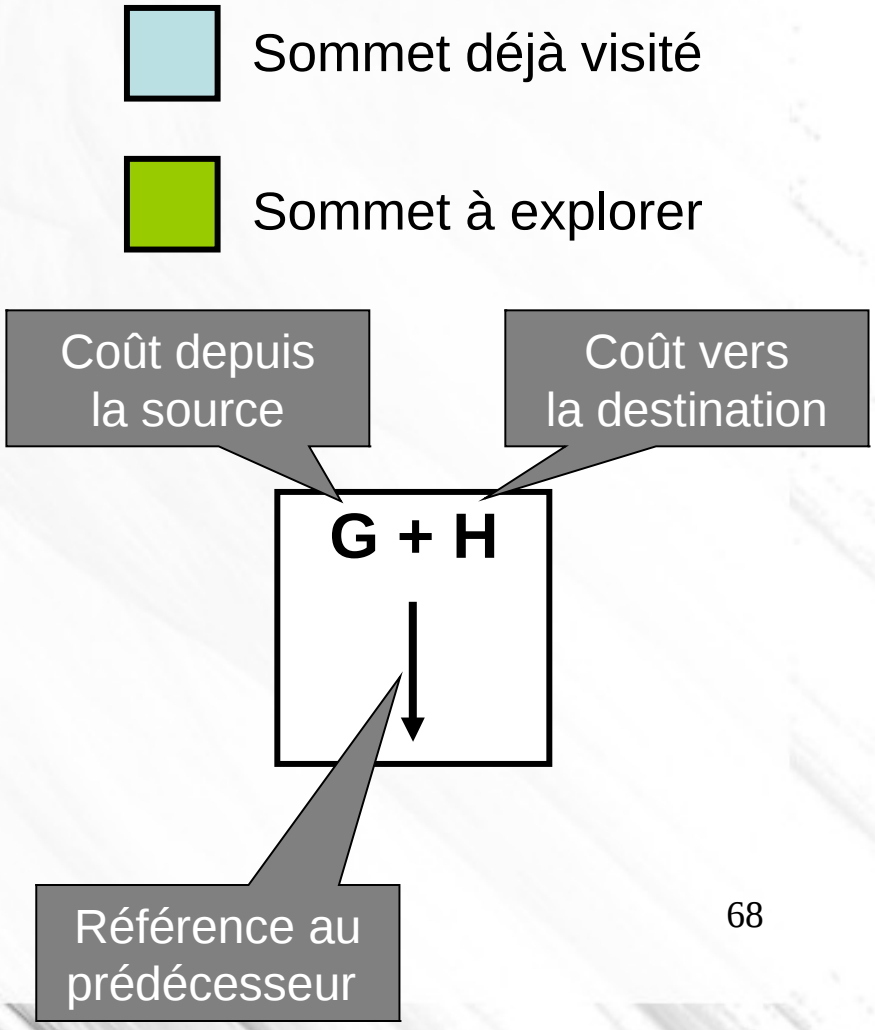
# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
20 + 60
40 + 60
50 + 50
60 + 40
30 + 70
40 + 80
50 + 70
60 + 60

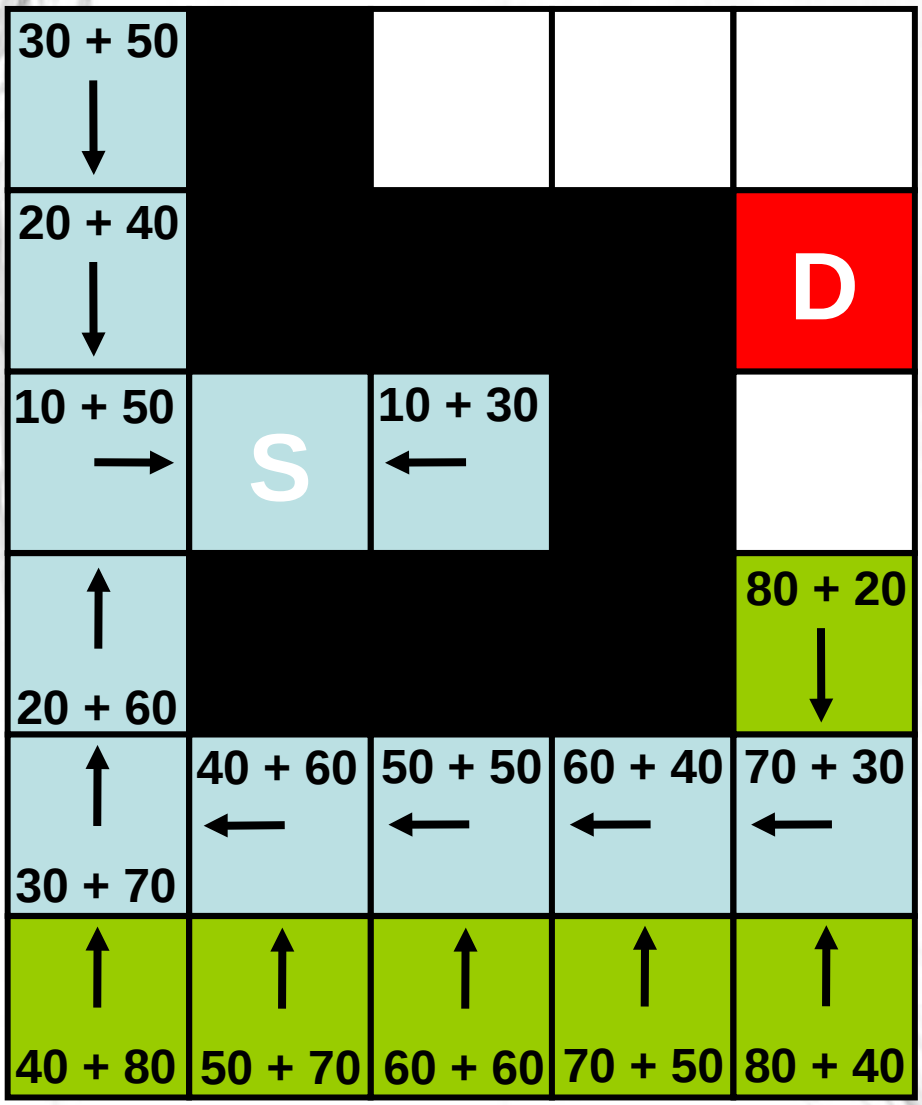

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

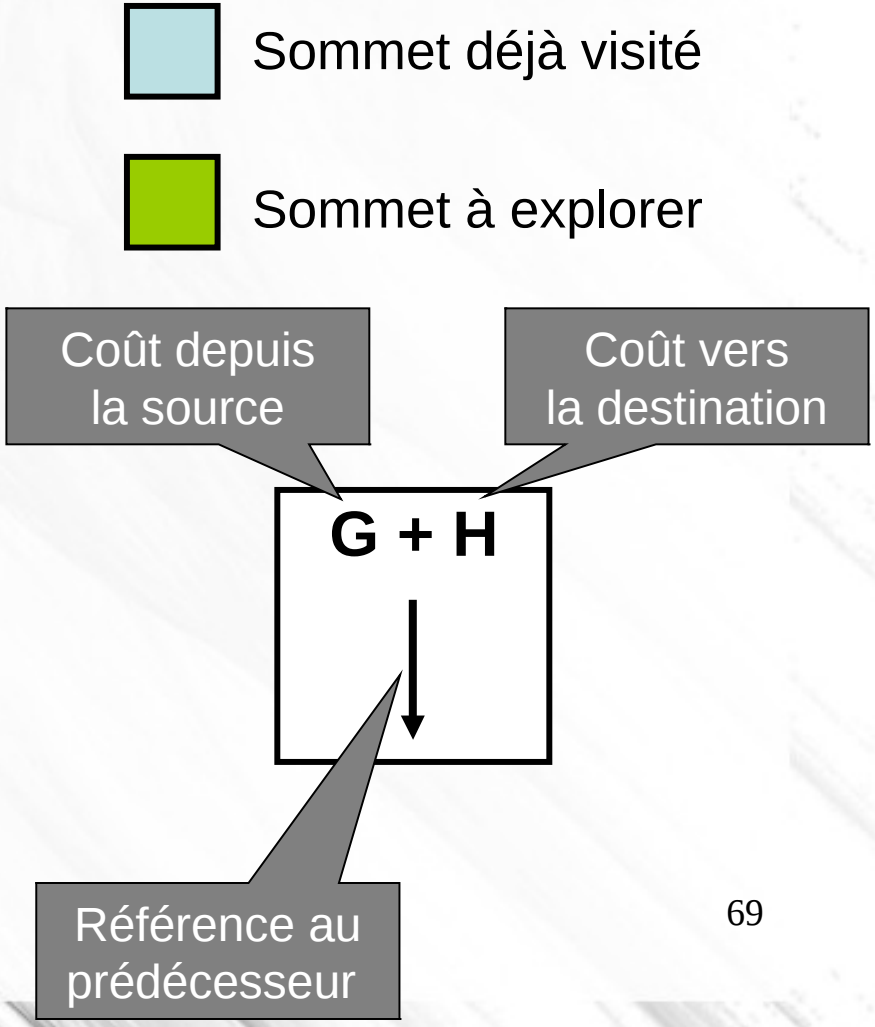
# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
20 + 60
30 + 70
40 + 60
50 + 50
60 + 40
70 + 30
40 + 80
50 + 70
60 + 60
70 + 50

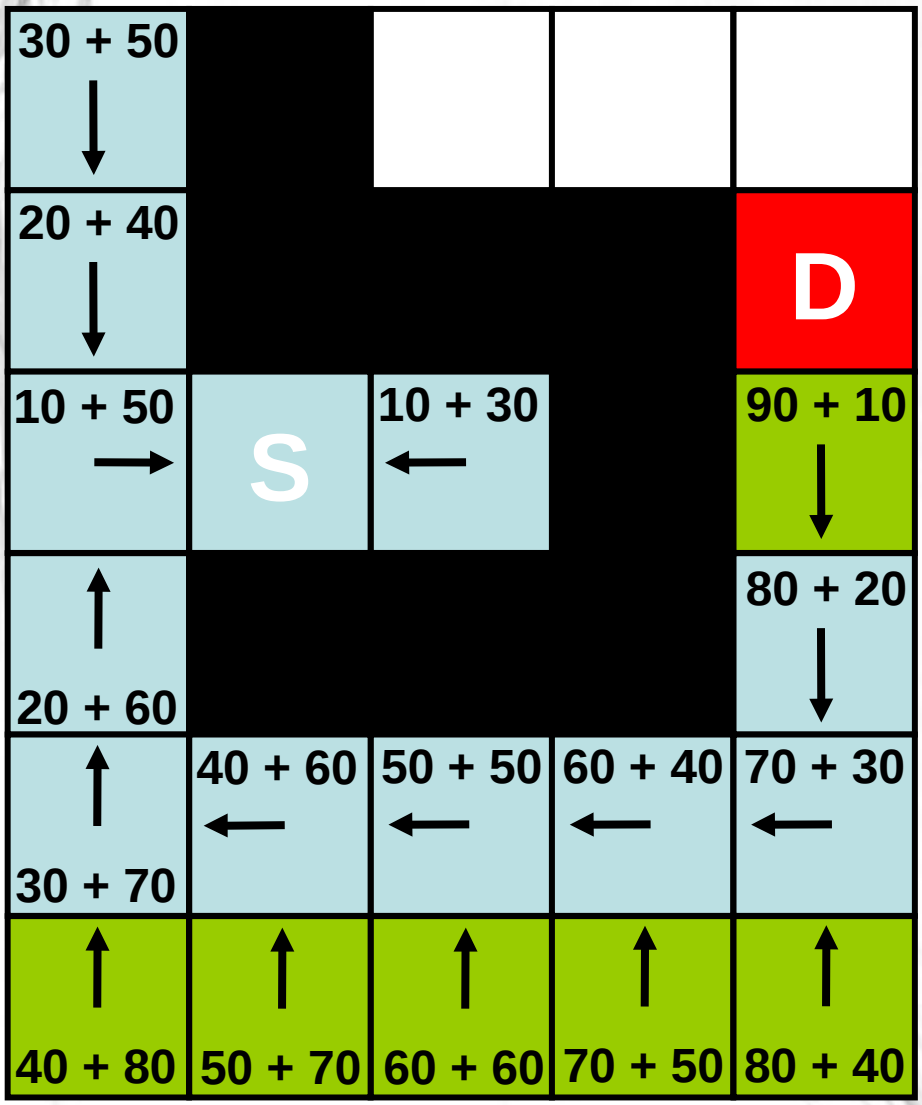

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

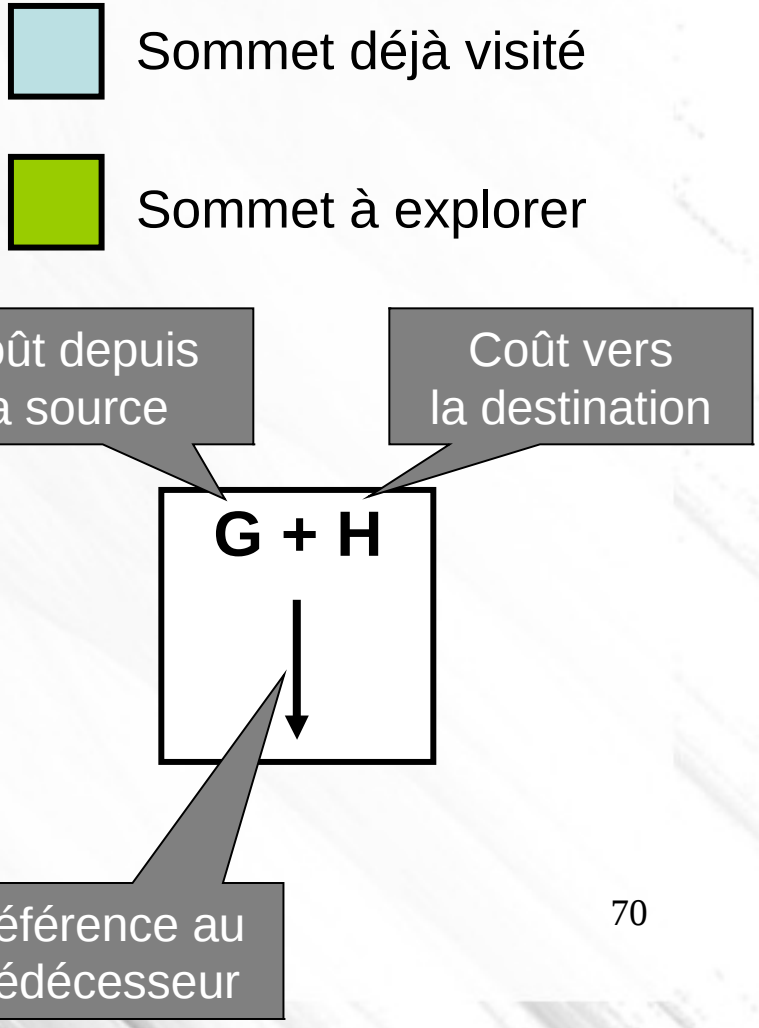
# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
80 + 20
20 + 60
40 + 60
50 + 50
60 + 40
70 + 30
30 + 70
40 + 80
50 + 70
60 + 60
70 + 50
80 + 40

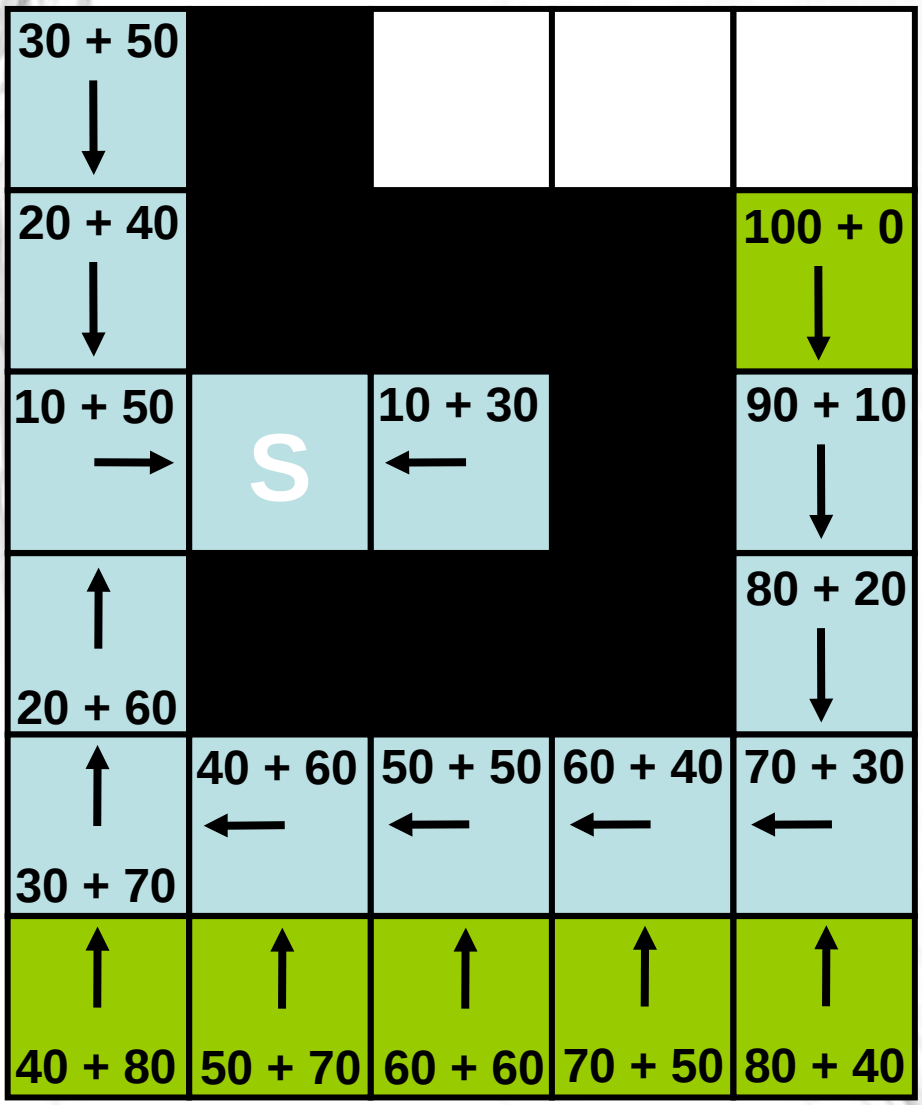

Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

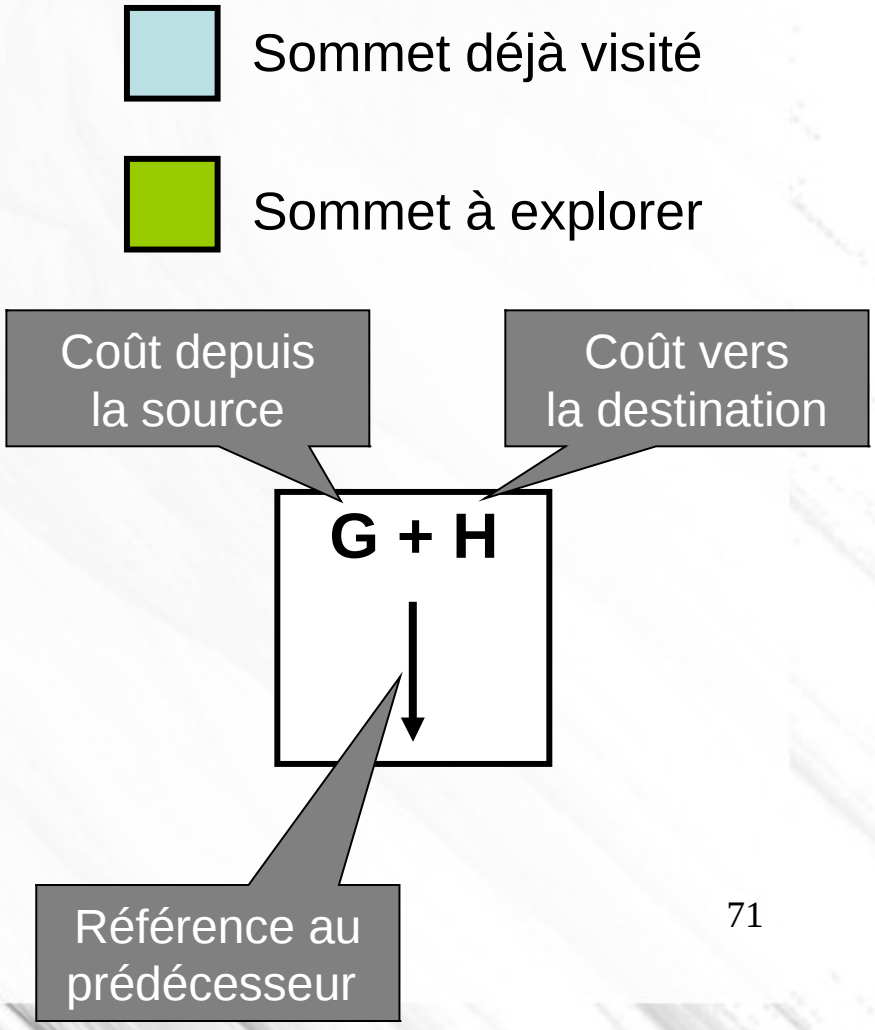
# Exemple 2


30 + 50
20 + 40
D
10 + 50
S
10 + 30
90 + 10
80 + 20
20 + 60
40 + 60
50 + 50
60 + 40
70 + 30
30 + 70
40 + 80
50 + 70
60 + 60
70 + 50
80 + 40


Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 2


30 + 50
20 + 40
100 + 0
10 + 50
S
10 + 30
90 + 10
80 + 20
20 + 60
40 + 60
50 + 50
60 + 40
70 + 30
30 + 70
40 + 80
50 + 70
60 + 60
70 + 50
80 + 40


Sommet déjà visité
Sommet à explorer
Coût depuis la source
Coût vers la destination
G + H
Référence au prédécesseur

# Exemple 2