Projet *Labyrinthe*

Algorithmes et Structures de Donnée

Juan-Carlos Barros et Daniel Kessler

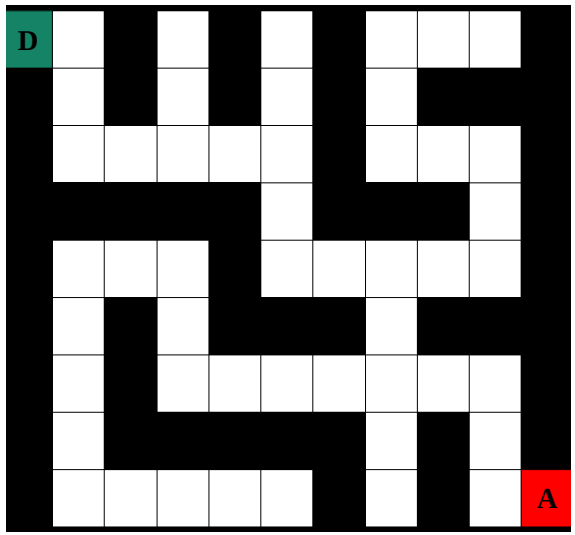
14 mai 2021

Cours d'*Algorithmes* et *Structures de donnée*

Cours d'*Algorithmes* et *Structures de donnée*

Projet *Labyrinthe*

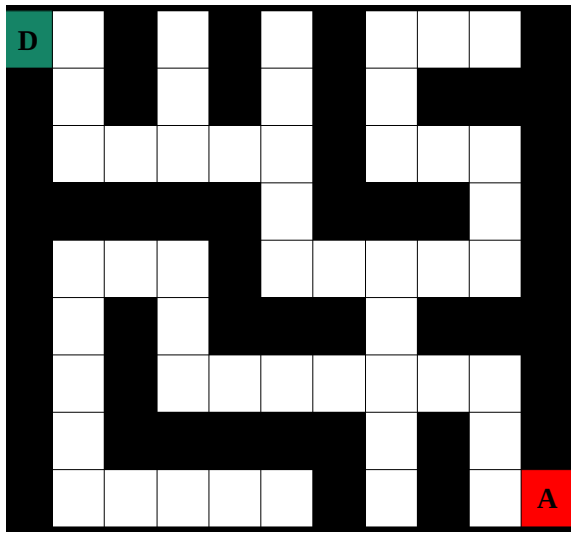
- Algorithme
- Structure de Donnée



Cours d'*Algorithmes* et *Structures de donnée*

Projet *Labyrinthe*

- Algorithme
  A*
- Structure de Donnée



Cours d'*Algorithmes* et *Structures de donnée*

Projet *Labyrinthe*

- Algorithme
  A*
- Structure de Donnée
  Priority Queue

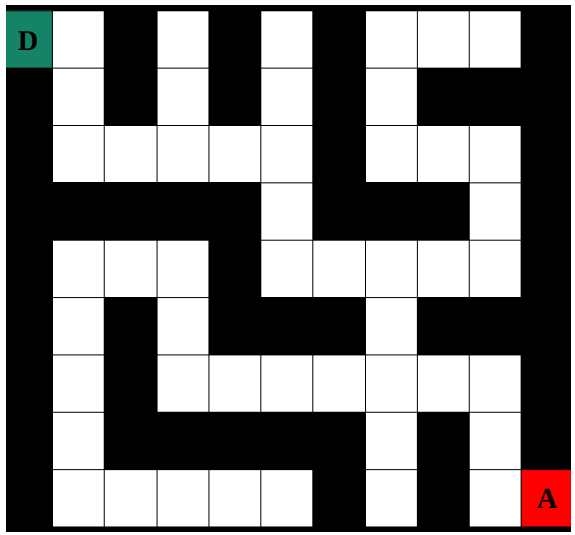


Table des matières

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - On veut le chemin le plus court, pour peut-être le réutiliser.

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - **On veut le chemin le plus court**, pour peut-être le réutiliser.

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - **On veut le chemin le plus court**, pour peut-être le réutiliser.
- Connait-on les coordonnées de la sortie dès le départ ?

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - **On veut le chemin le plus court**, pour peut-être le réutiliser.
- Connait-on les coordonnées de la sortie dès le départ ?
  - Oui, et cette information pourra nous aider.
  - Non, le lieu de la sortie partie des inconnues.

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - **On veut le chemin le plus court**, pour peut-être le réutiliser.
- Connait-on les coordonnées de la sortie dès le départ ?
  - **Oui, et cette information pourra nous aider.**
  - Non, le lieu de la sortie partie des inconnues.

Un problème, plusieurs solutions

- Breadth-First Search
  - garantit de trouver une solution si elle existe
  - solution optimale si tous les pas sont égaux

Un problème, plusieurs solutions

- Breadth-First Search
  - garantit de trouver une solution si elle existe
  - solution optimale si tous les pas sont égaux
- Dijkstra
  - choisit où explorer selon les distances déjà parcourues
  - garantit de trouver le plus court chemin

Un problème, plusieurs solutions

- Breadth-First Search
  - garantit de trouver une solution si elle existe
  - solution optimale si tous les pas sont égaux
- Dijkstra
  - choisit où explorer selon les distances déjà parcourues
  - garantit de trouver le plus court chemin
- A*
  - nécessite de connaître les coordonnées de la sortie
  - choisit où explorer selon les distances déjà parcourues et la distance à la sortie

Algorithme A*

pseudo-code

Démarrer une file d'attente avec la cellule de départ D, une liste de prédécesseurs avec $\{D : Nil\}$ et une liste de coûts d'accès avec $\{D : 0\}$.

Algorithme A*

pseudo-code

Démarrer une file d'attente avec la cellule de départ D, une liste de prédécesseurs avec $\{D : Nil\}$ et une liste de coûts d'accès avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

Algorithme A*

pseudo-code

Démarrer une file d'attente avec la cellule de départ D, une liste de prédécesseurs avec $\{D : Nil\}$ et une liste de coûts d'accès avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente

Algorithme A*

pseudo-code

Démarrer une file d'attente avec la cellule de départ D, une liste de prédécesseurs avec $\{D : Nil\}$ et une liste de coûts d'accès avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente
- si C est la cellule d'arrivée A, retourner le chemin qui y amène (via backtracking sur les prédecesseurs)

Algorithme A*

pseudo-code

Démarrer une file d'attente avec la cellule de départ D, une liste de prédécesseurs avec $\{D : Nil\}$ et une liste de coûts d'accès avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente
- si C est la cellule d'arrivée A, retourner le chemin qui y amène (via backtracking sur les prédecesseurs)
- sinon, pour chaque vosin V de C qui n'est pas déjà accessible à moindre co'ut

Algorithme A*

pseudo-code

Démarrer une file d'attente avec la cellule de départ D, une liste de prédécesseurs avec $\{D : Nil\}$ et une liste de coûts d'accès avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente
- si C est la cellule d'arrivée A, retourner le chemin qui y amène (via backtracking sur les prédecesseurs)
- sinon, pour chaque vosin V de C qui n'est pas déjà accessible à moindre co'ut
  - mémoriser le prédecesseur de V et le coût d'accès à V

Algorithme A*

pseudo-code

Démarrer une file d'attente avec la cellule de départ D, une liste de prédécesseurs avec $\{D : Nil\}$ et une liste de coûts d'accès avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente
- si C est la cellule d'arrivée A, retourner le chemin qui y amène (via backtracking sur les prédecesseurs)
- sinon, pour chaque vosin V de C qui n'est pas déjà accessible à moindre co'ut
  - mémoriser le prédecesseur de V et le coût d'accès à V
  - ajouter V à la file d'attente

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût estimé d'un chemin complet passant par cette cellule.

priorité $= \text{coût}_r\acute{e}el(\text{D}->\text{C}) + co\hat{u}t_estimé(\text{C}->\text{S})$

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût estimé d'un chemin complet passant par cette cellule.

priorité $= \text{coût}_r\acute{e}el(\text{D}->\text{C}) + co\hat{u}t_estimé(\text{C}->\text{S})$

La distance restante depuis une cellule jusqu'à l'arrivée doit être estimée *sans jamais surestimer*.

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût estimé d'un chemin complet passant par cette cellule.

priorité $= \text{coût}_r\acute{e}el(\text{D}->\text{C}) + co\hat{u}t_estimé(\text{C}->\text{S})$

La distance restante depuis une cellule jusqu'à l'arrivée doit être estimée *sans jamais surestimer*.

- La **distance de Manhattan** $|\Delta x| + |\Delta y|$ est un bon estimateur si les mouvements permis sont horizontaux et verticaux.

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût estimé d'un chemin complet passant par cette cellule.

priorité $= \text{coût}_réel(\text{D}->\text{C}) + coût_estimé(\text{C}->\text{S})$

La distance restante depuis une cellule jusqu'à l'arrivée doit être estimée *sans jamais surestimer*.

- La **distance de Manhattan** $|\Delta x| + |\Delta y|$ est un bon estimateur si les mouvements permis sont horizontaux et verticaux.
- Une **heuristique nulle** ramène A* à l'algorithme de Dijkstra (ou Breadth-First Search sur grille carrée).

File d'attente : "Priority Queue"

- Structure permettant insertion avec priorité et "pop" rapide de l'élément prioritaire
- Implémentation en Python en tant que **binary heap** avec le module *heapq*
- Dans cette implémentation, vérifier si vide en $O(1)$, insertion et "pop" en $O(\log(n))$ où n est le nombre d'objets en attente [1]

1. cf. https ://www.cs.princeton.edu/ wayne/kleinberg-tardos/pdf/BinomialHeaps.pdf

Priority Queue pour A*

- La priorité d'un noeud est le **coût heuristique** pour aller de l'entrée à la sortie en passant par ce noeud.

Priority Queue pour A*

- La priorité d'un noeud est le **coût heuristique** pour aller de l'entrée à la sortie en passant par ce noeud.
- Un élément de la queue a comme attributs l'identité du noeud, le **coût réel** pour y accéder et son prédecesseur

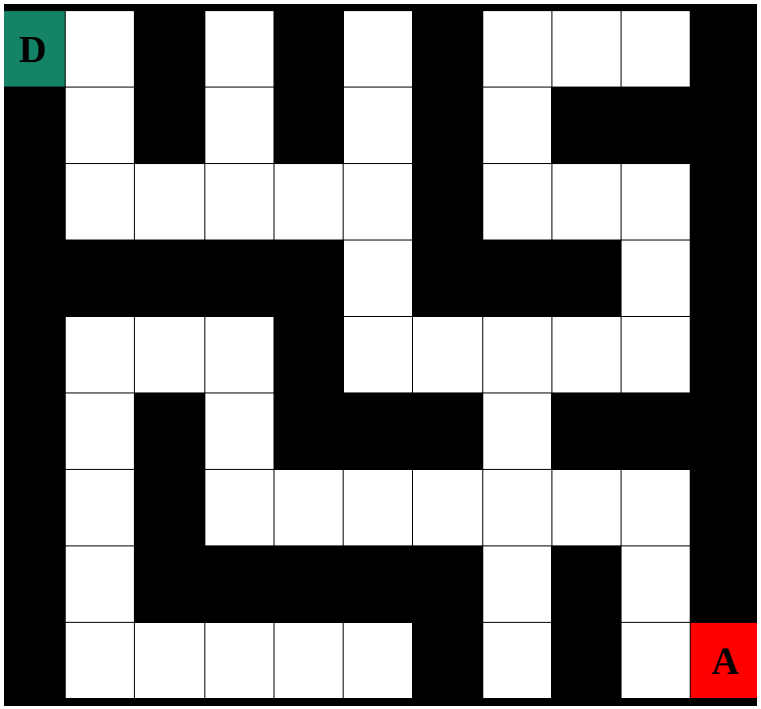
Priority Queue pour A*

- La priorité d'un noeud est le **coût heuristique** pour aller de l'entrée à la sortie en passant par ce noeud.
- Un élément de la queue a comme attributs l'identité du noeud, le **coût réel** pour y accéder et son prédecesseur
- La queue est initialisée avec le noeud de départ et un coût nul.

Idée de preuve

Si l'heuristique sous-estime le coût du chemin futur, alors on arrive à un noeud toujours par un chemin de distance minimale d'abord.
[à expliquer]

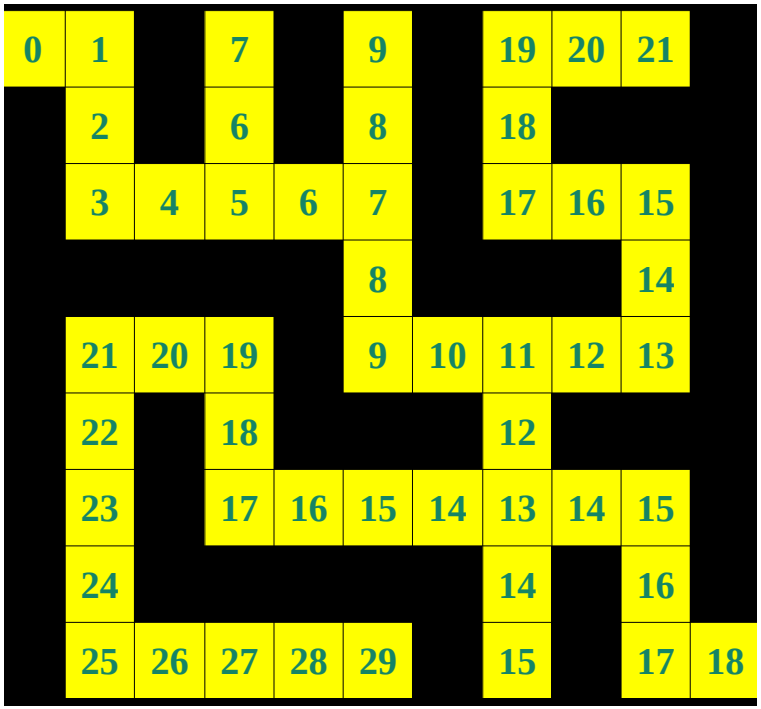
Exemple de résolution



Objectif

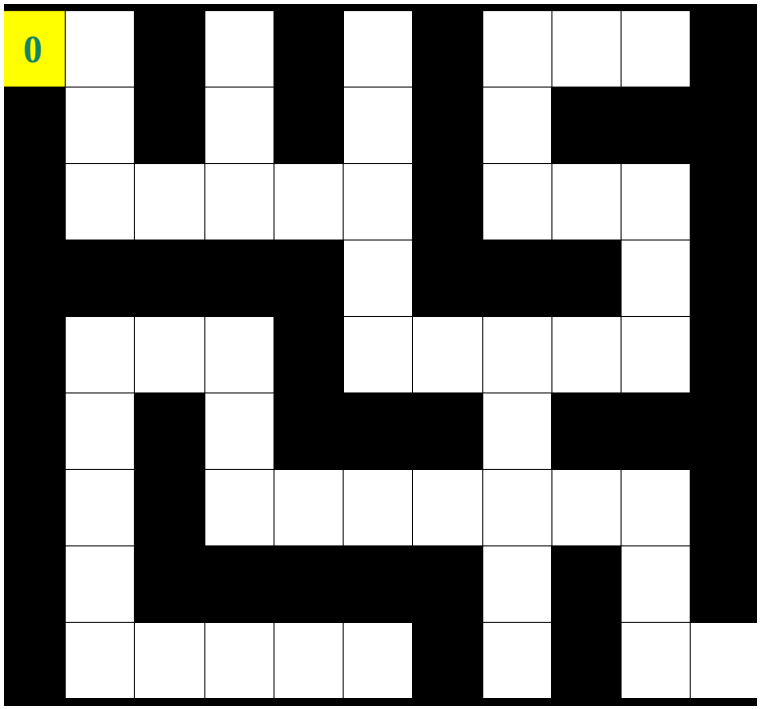
Trouver le chemin le plus court entre le **D**épart et l'**A**rrivée

Exemple de résolution



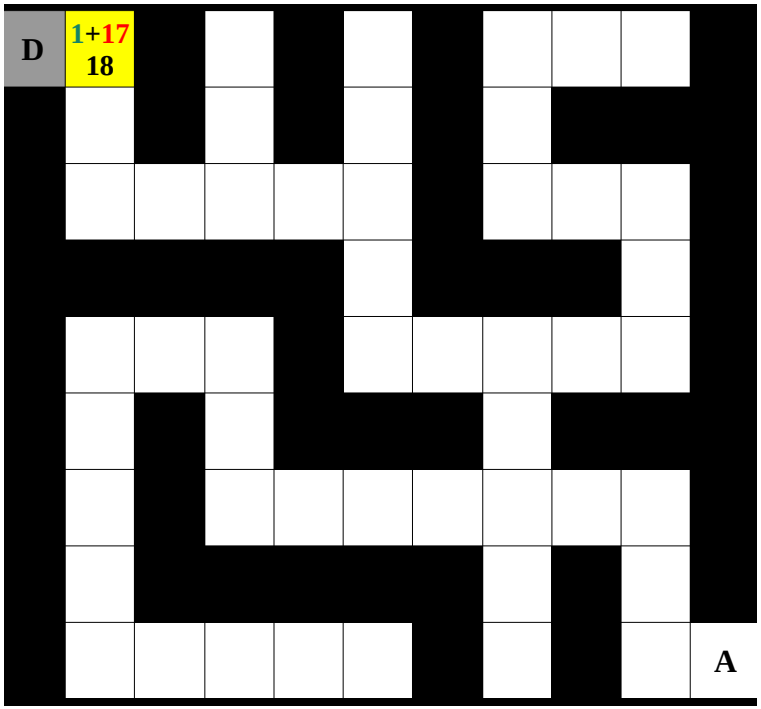
Chaque noeud est à une distance réelle de l'arrivée, mais ces distances ne sont pas encore connues.

Exemple de résolution



Le départ est mis en file d'attente, avec une priorité 0.

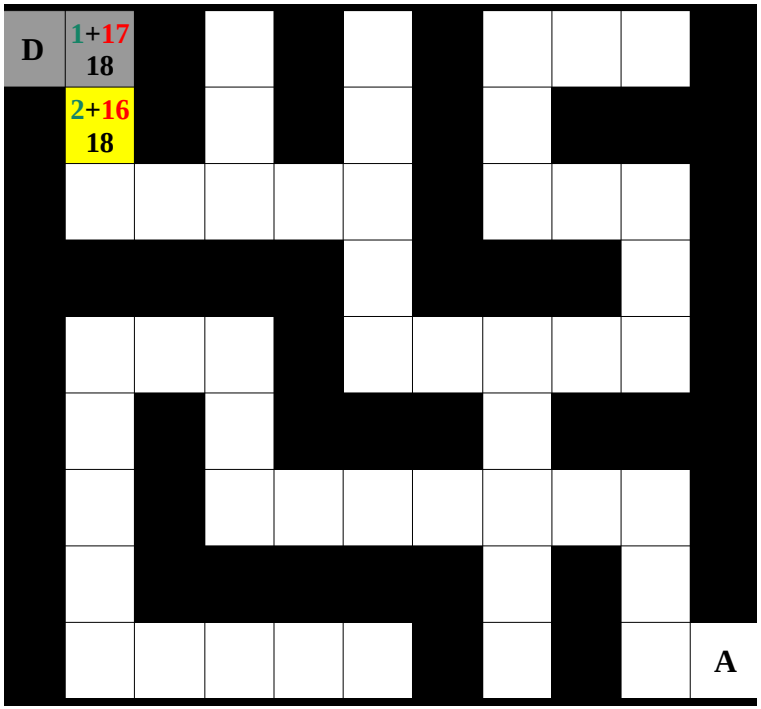
Exemple de résolution



Le seul voisin est évalué :

- coût réel pour y accéder : 1
- coût heuristique pour la suite : 17
- coût heuristique total (priorité) : 18

Exemple de résolution

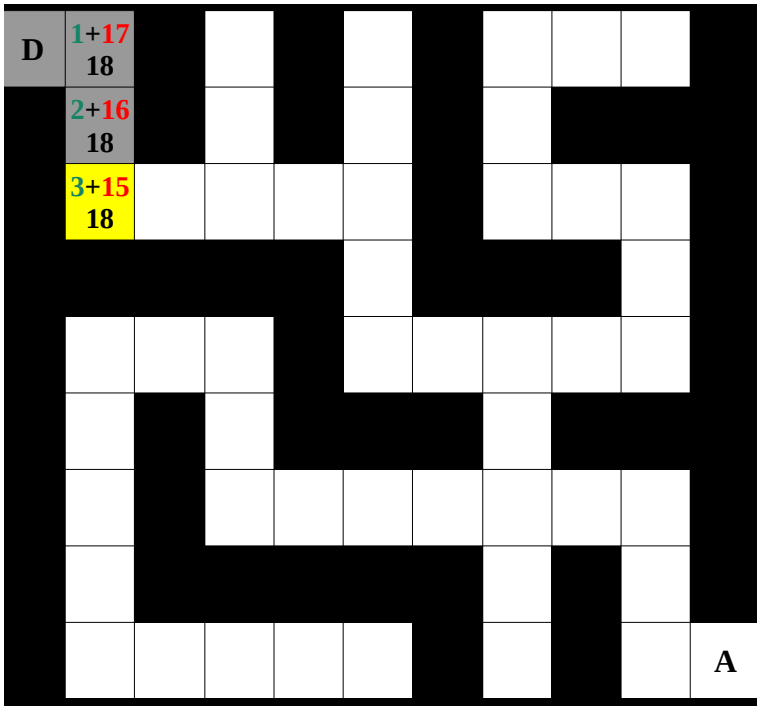


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

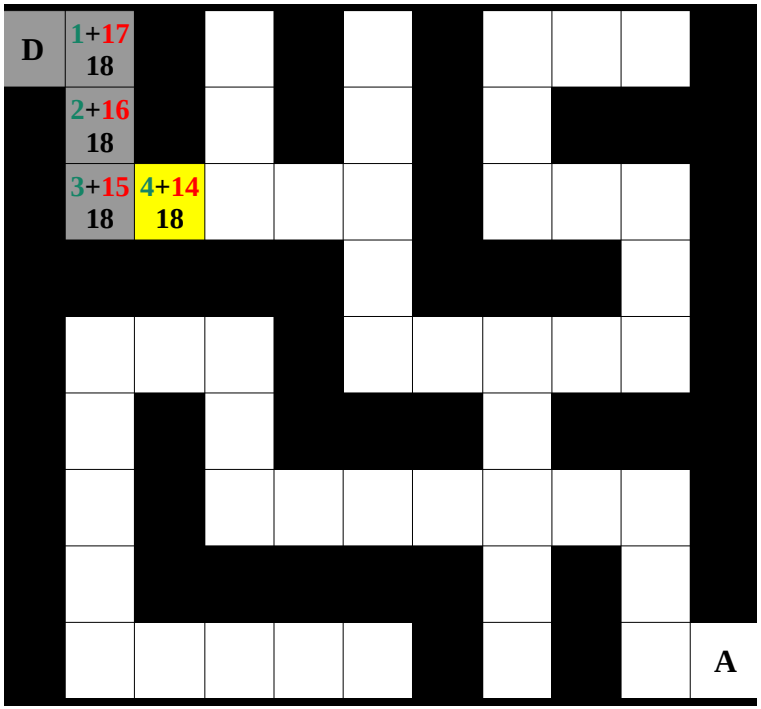


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

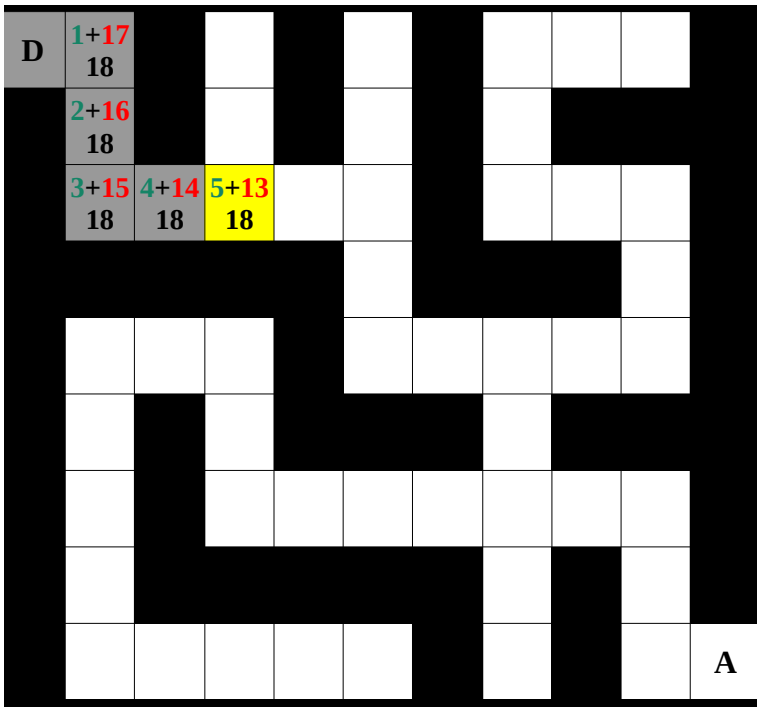


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

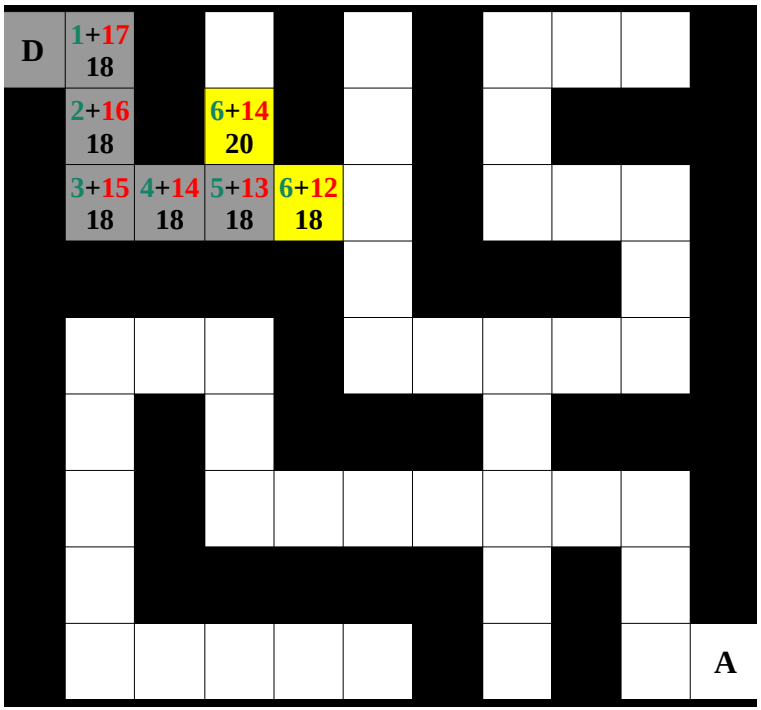


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

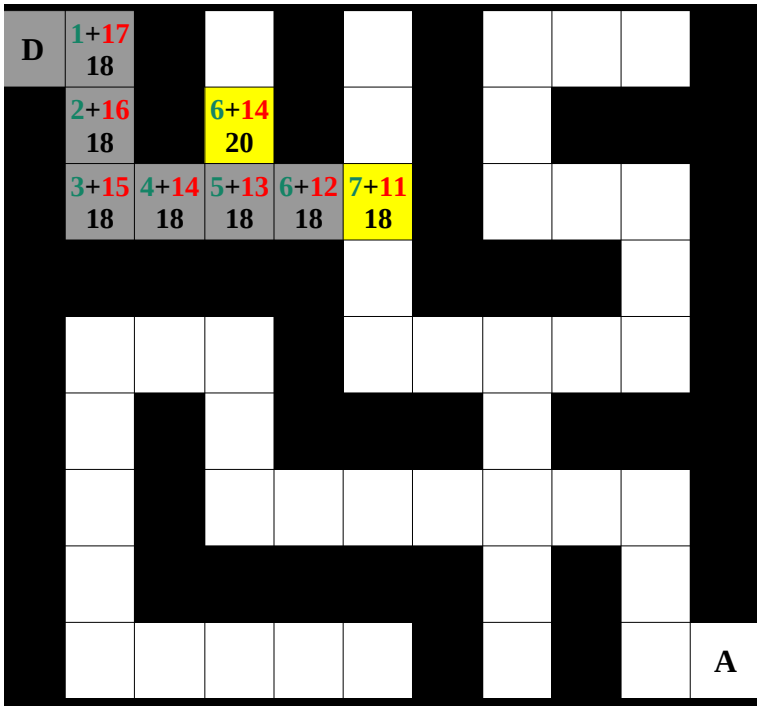


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

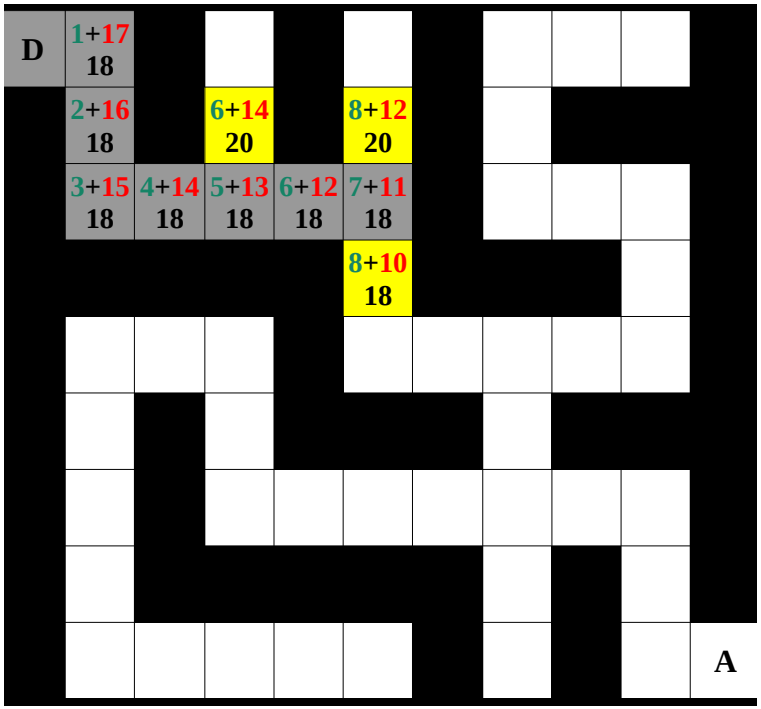


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

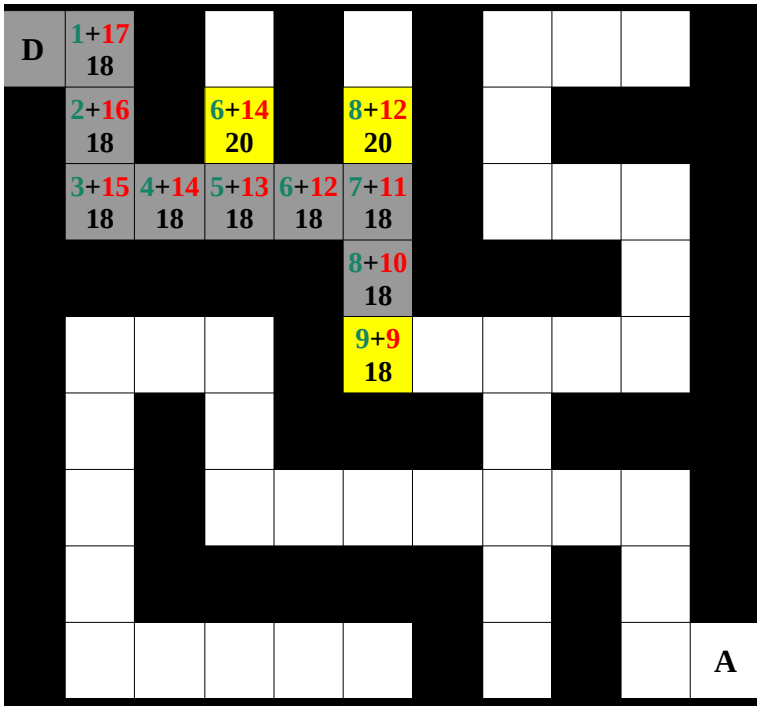


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

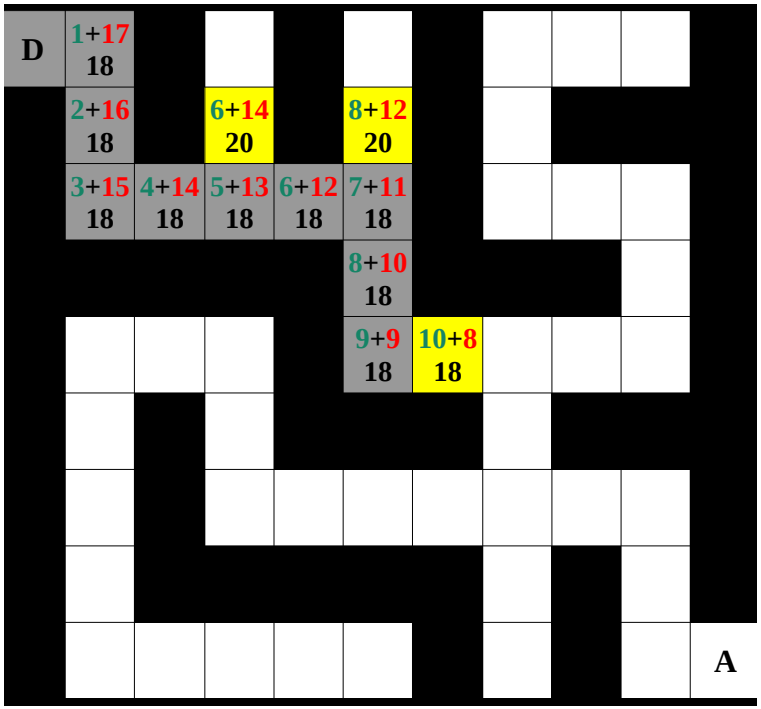


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

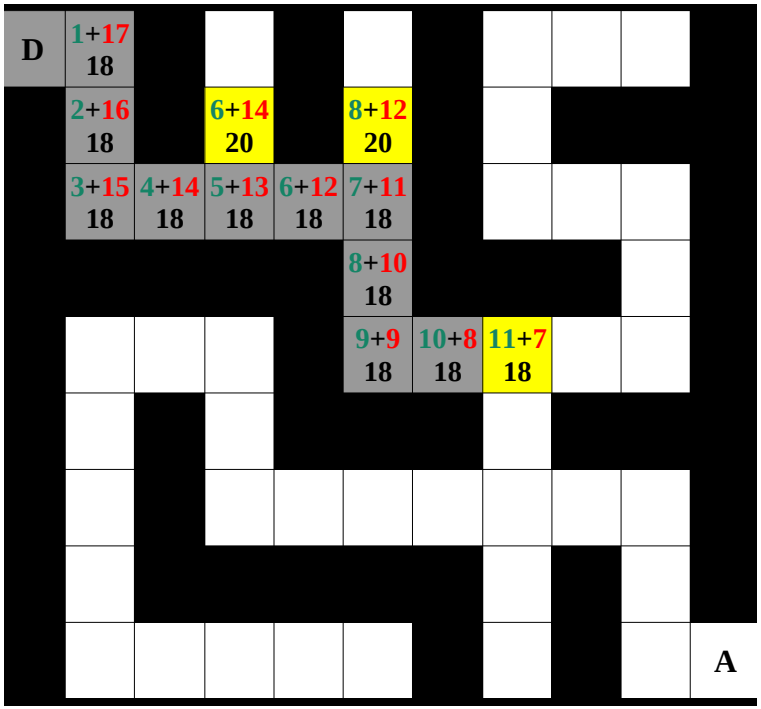


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

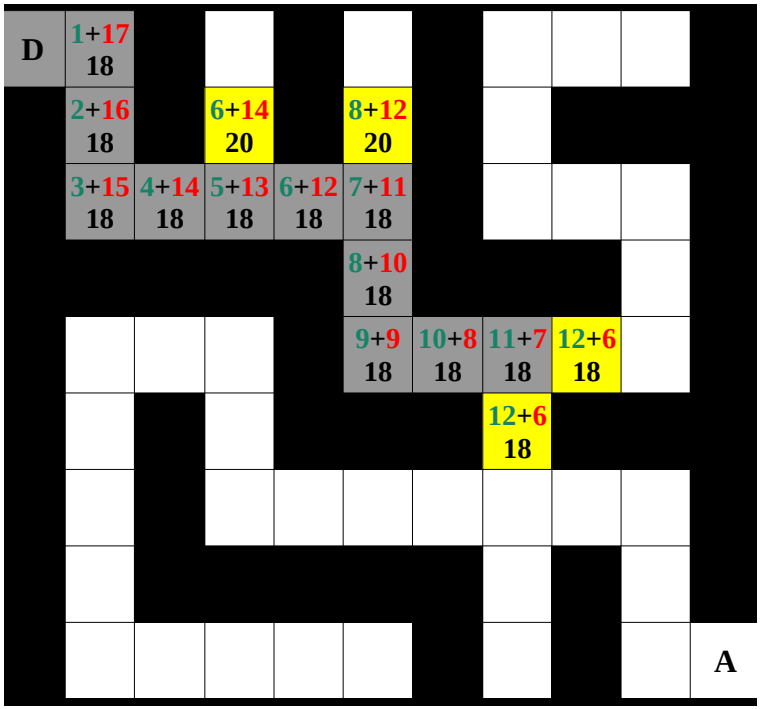


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

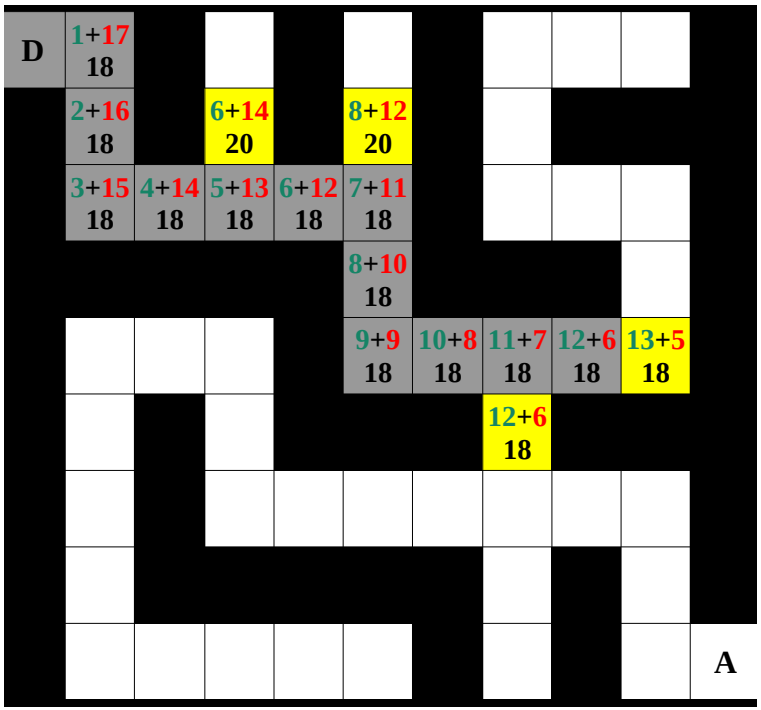


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.
Parfois deux choix ont la même priorité.

Exemple de résolution

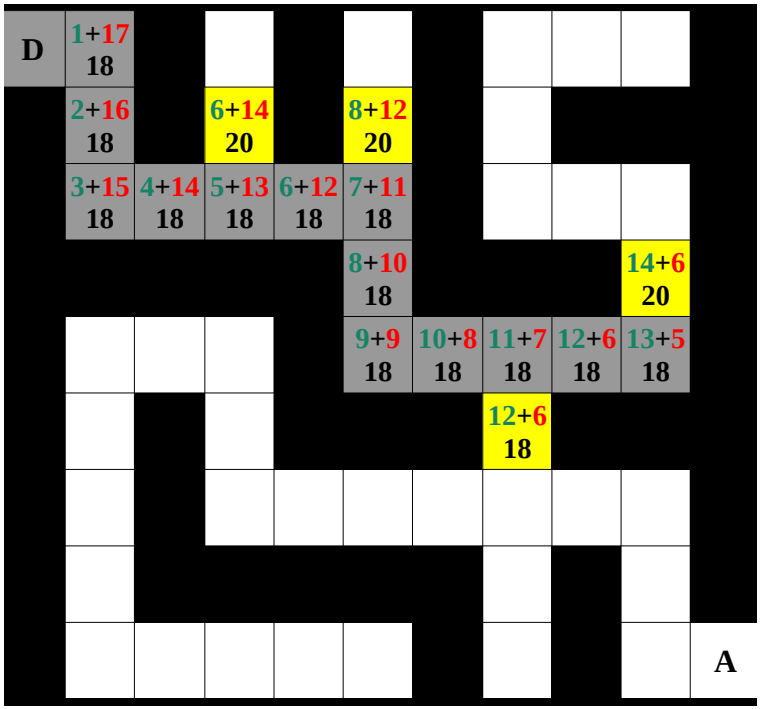


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

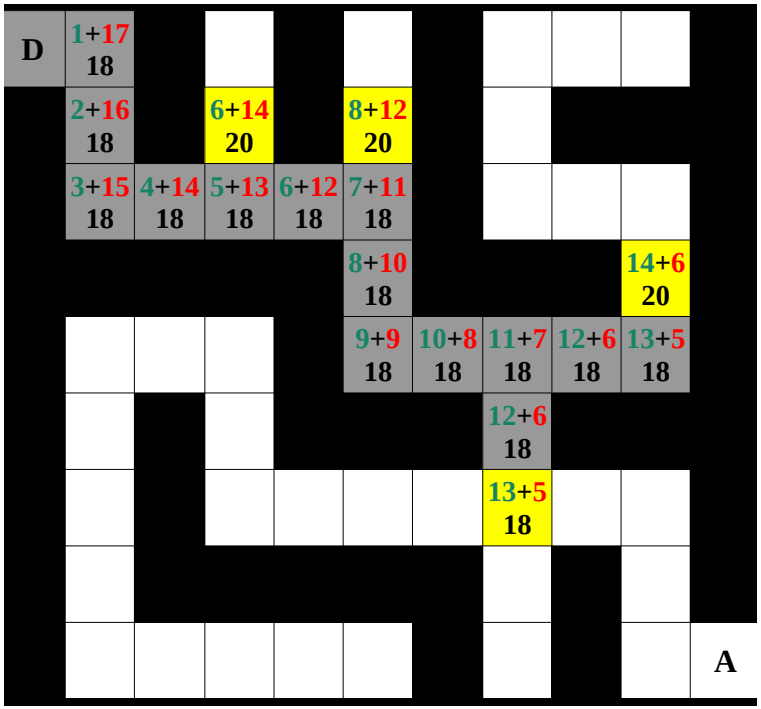


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

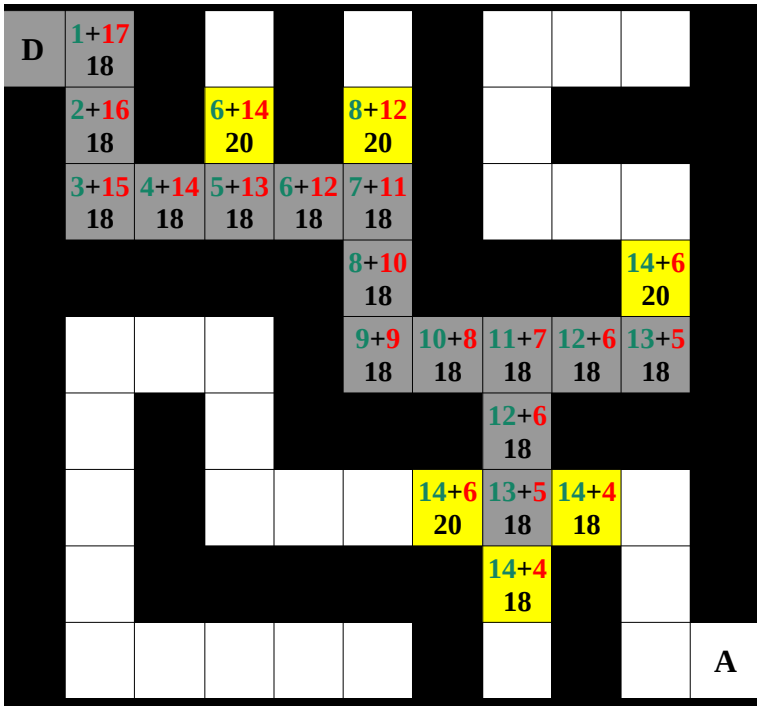


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

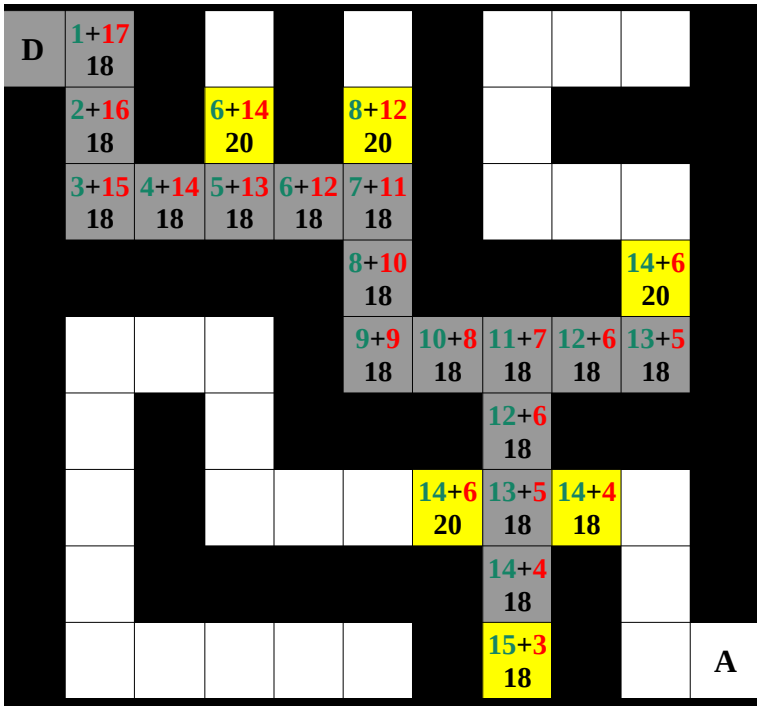


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.
Parfois deux choix ont la même priorité.

Exemple de résolution

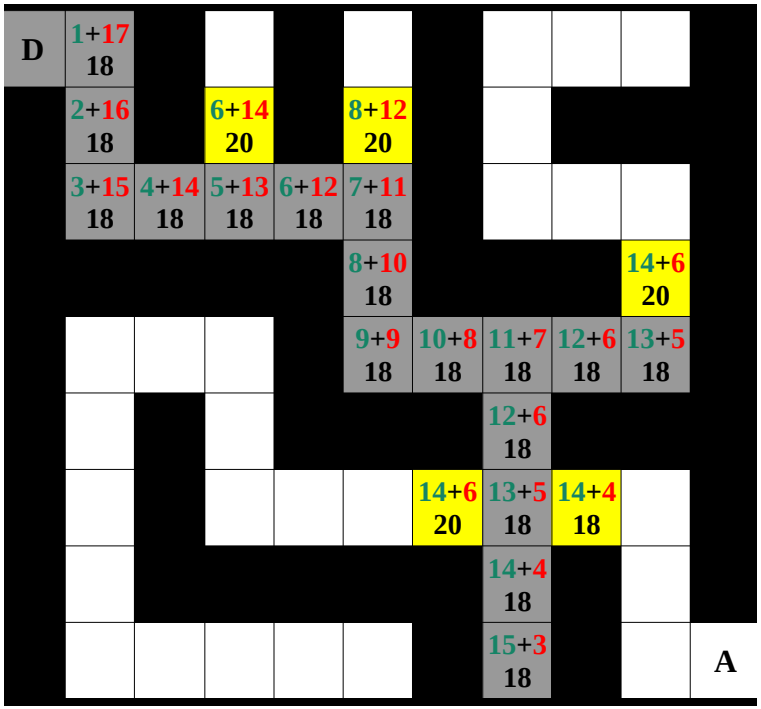


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.
Parfois deux choix ont la même priorité.

Exemple de résolution

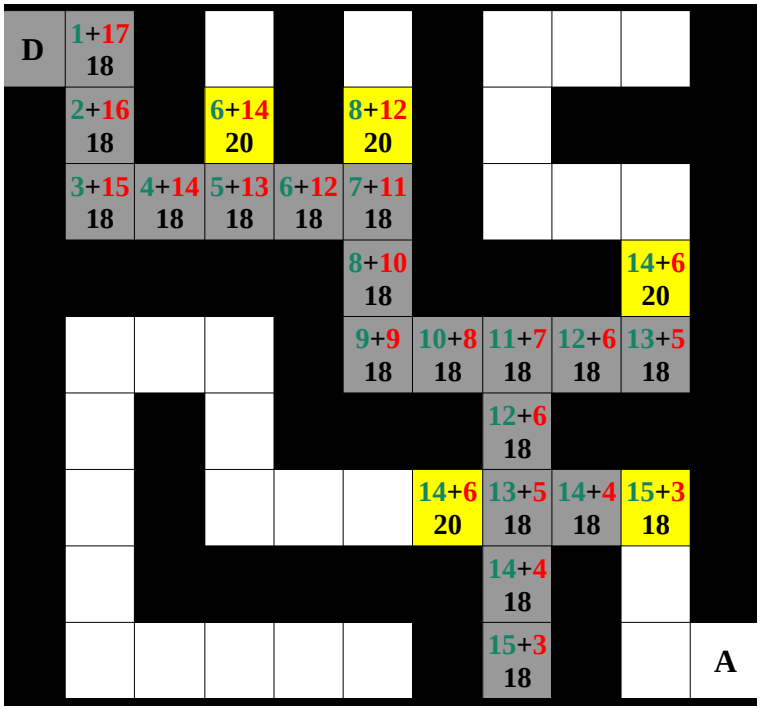


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

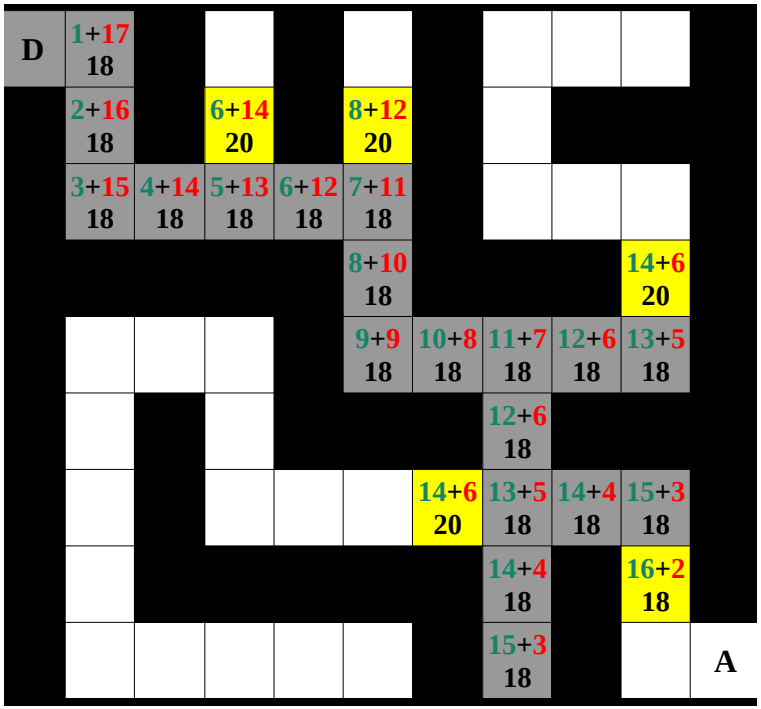


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

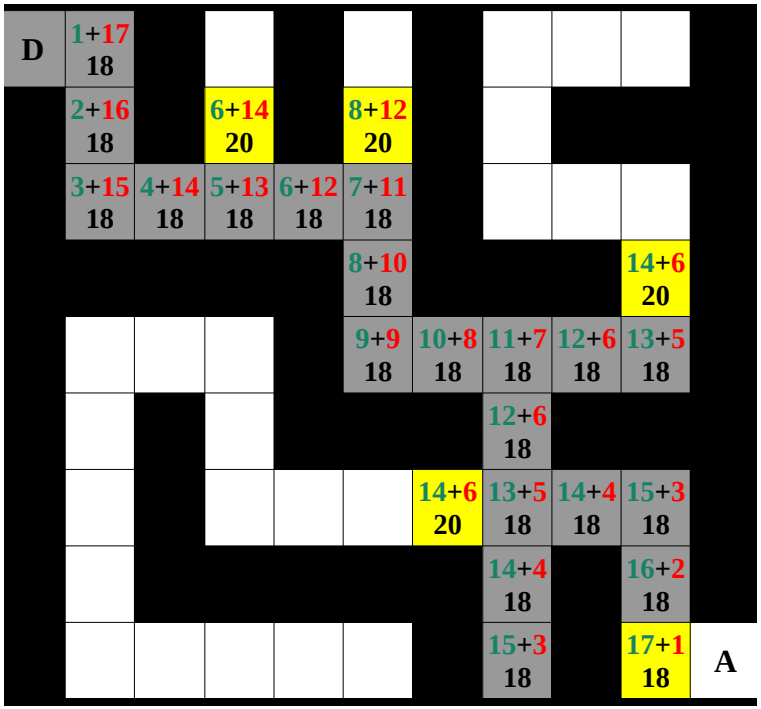


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

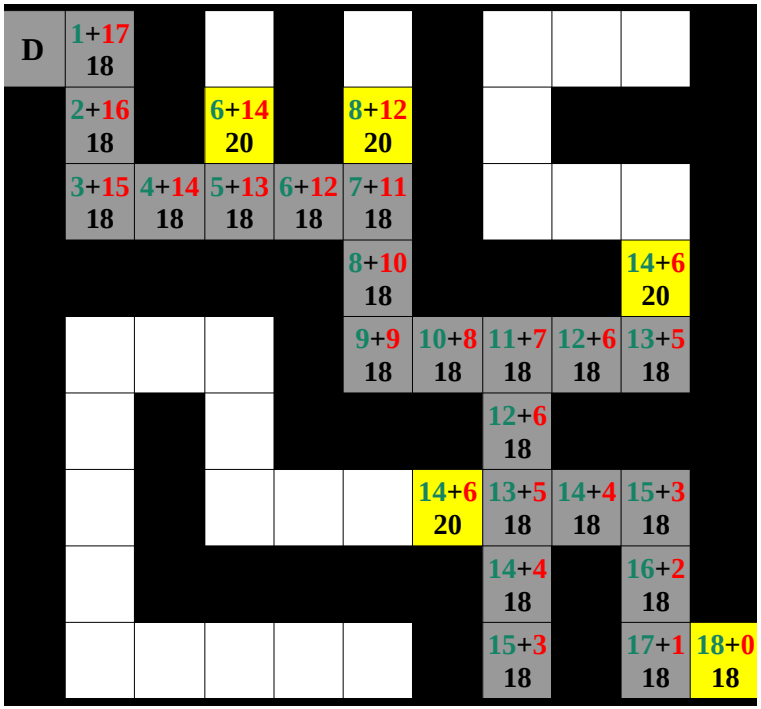


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

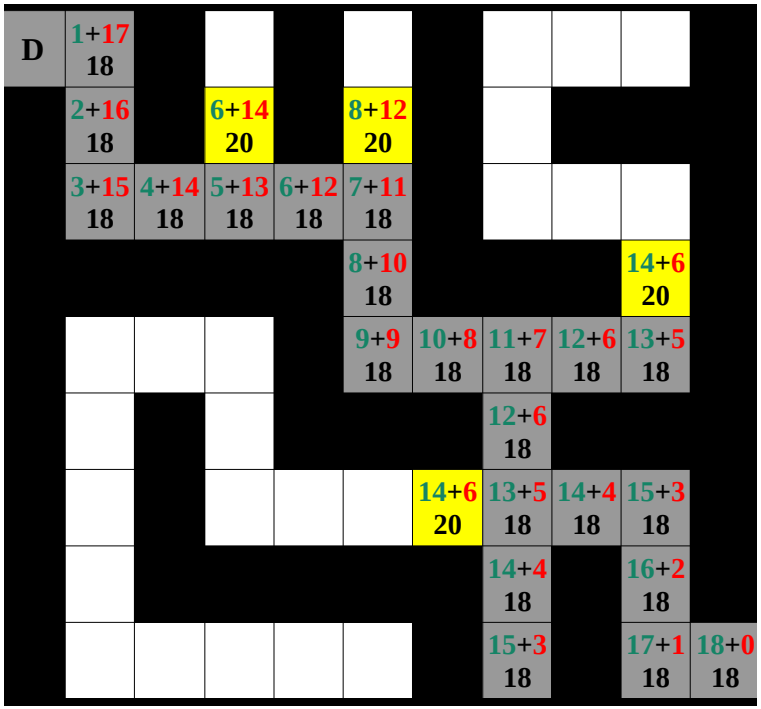


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

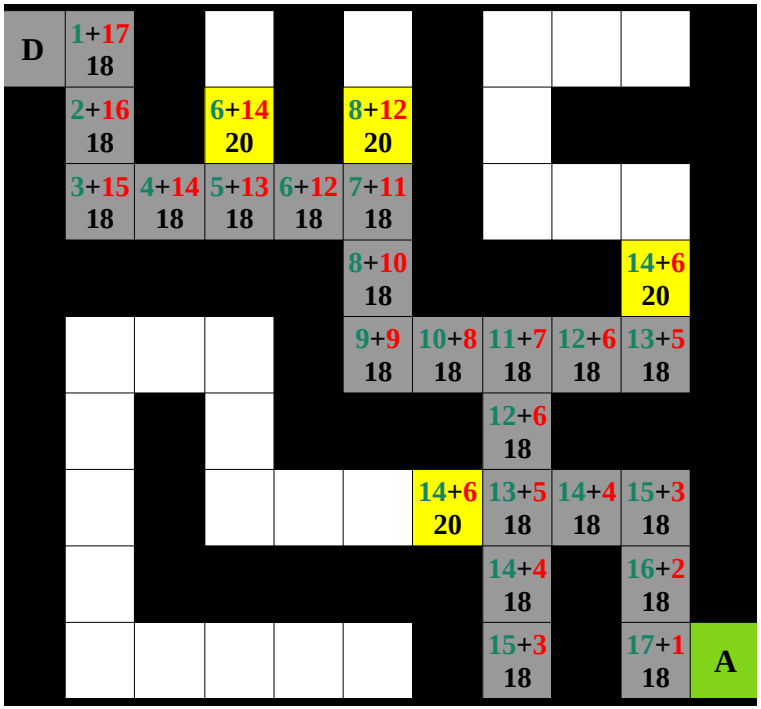


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

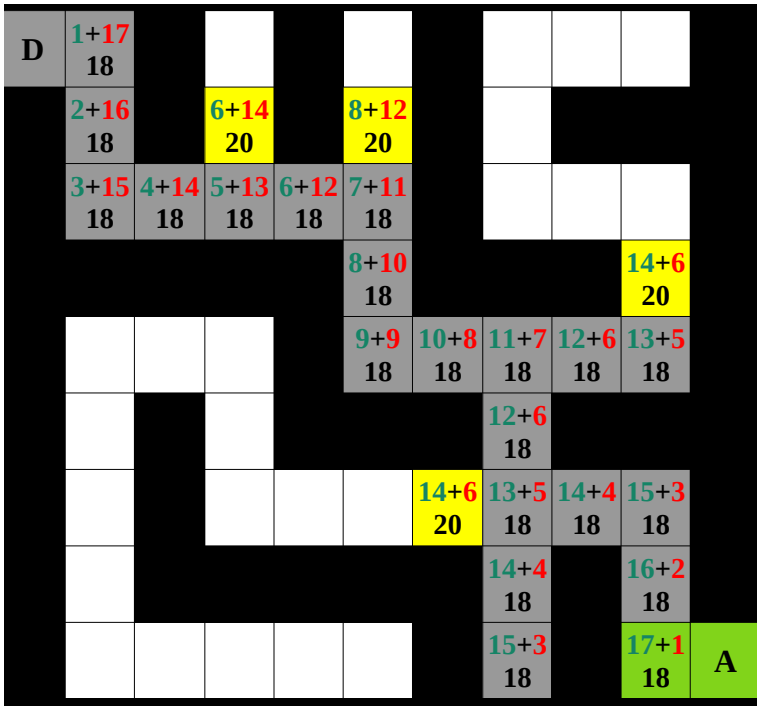
L'algorithme poursuit son chemin.

Exemple de résolution



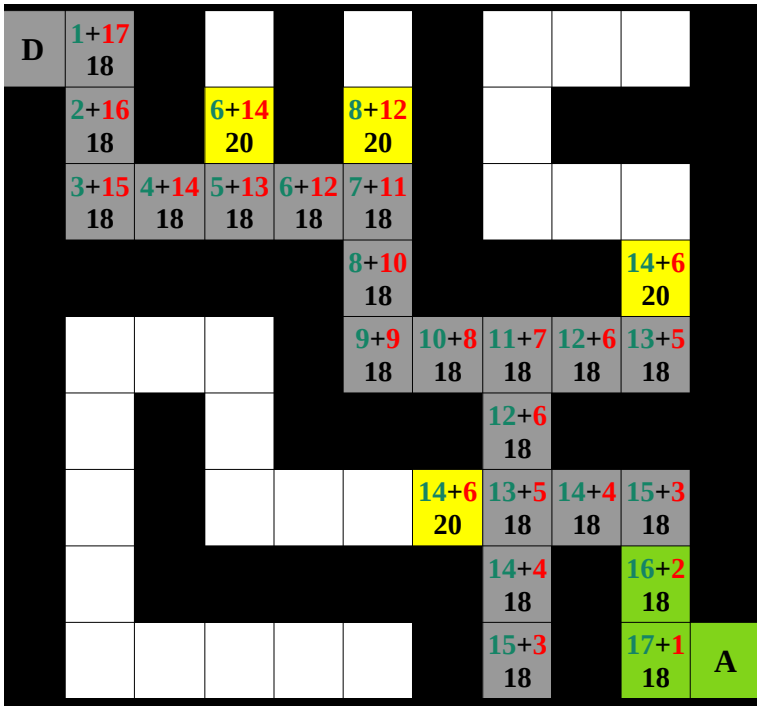
Un chemin vers la sortie a été trouvé !

Exemple de résolution



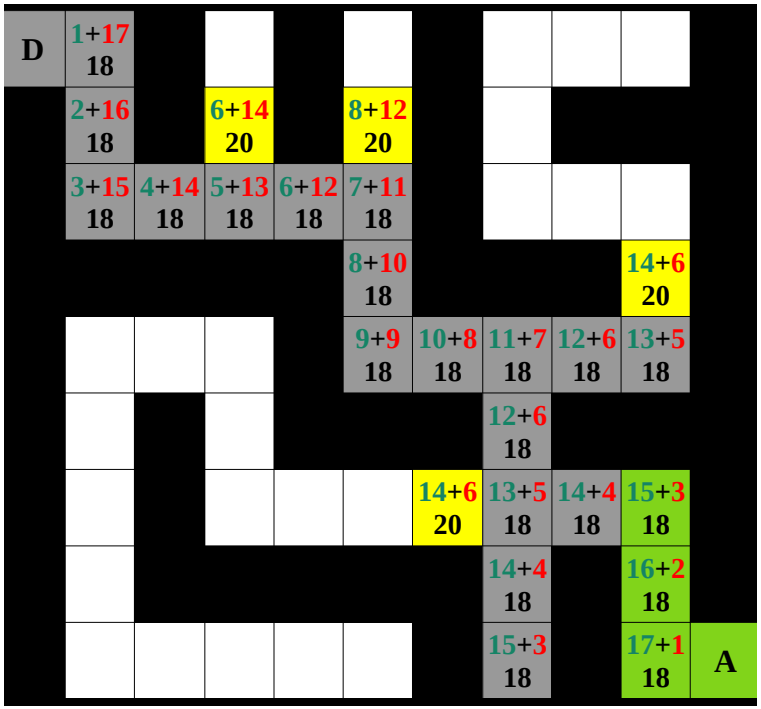
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



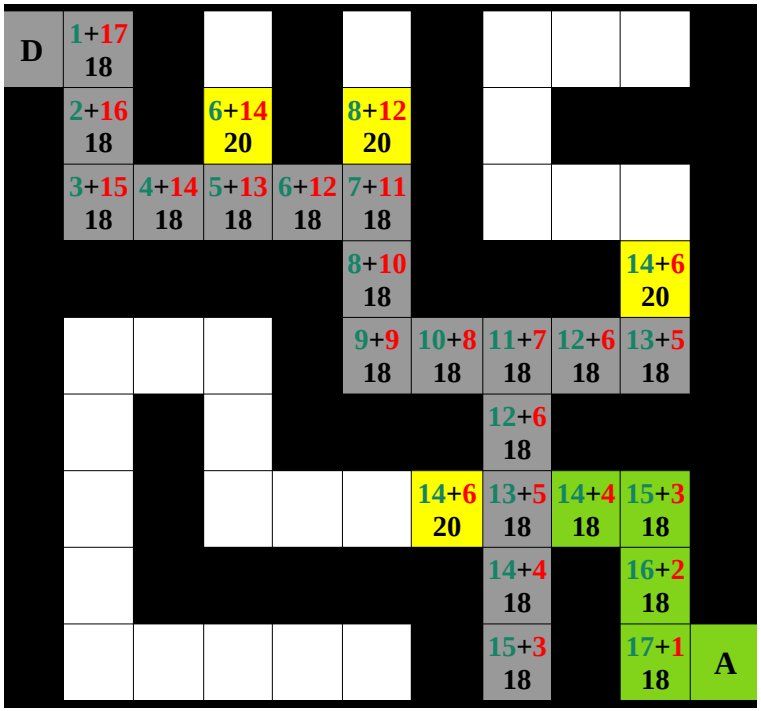
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



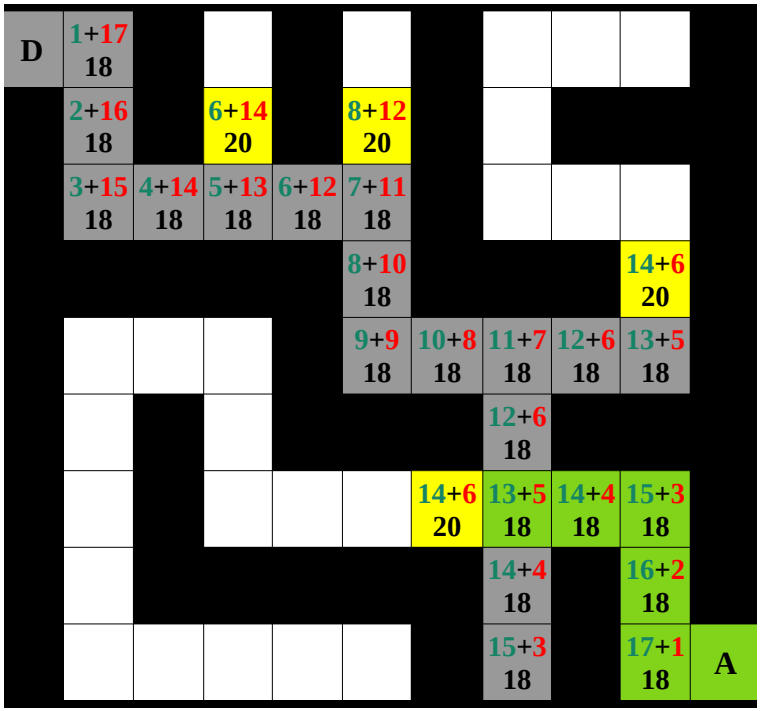
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



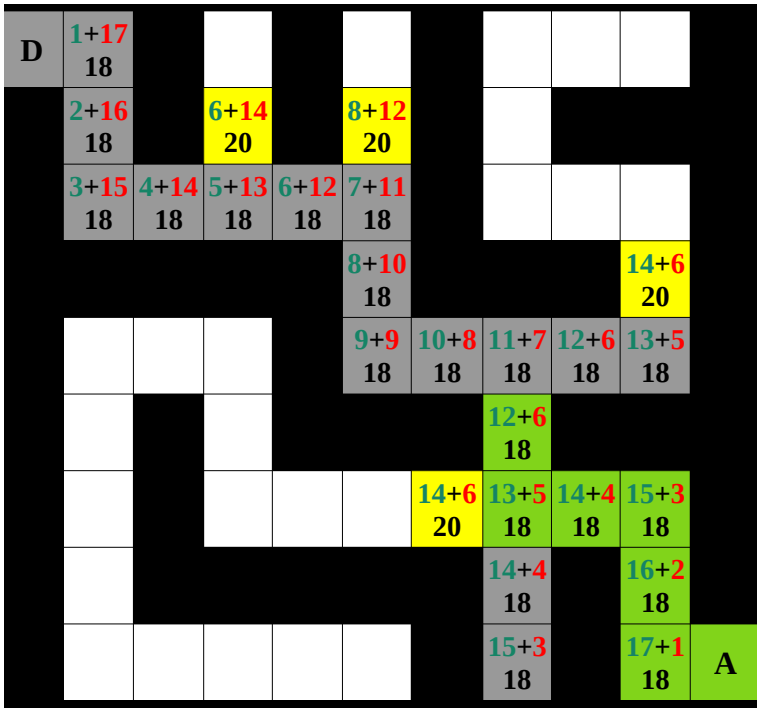
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



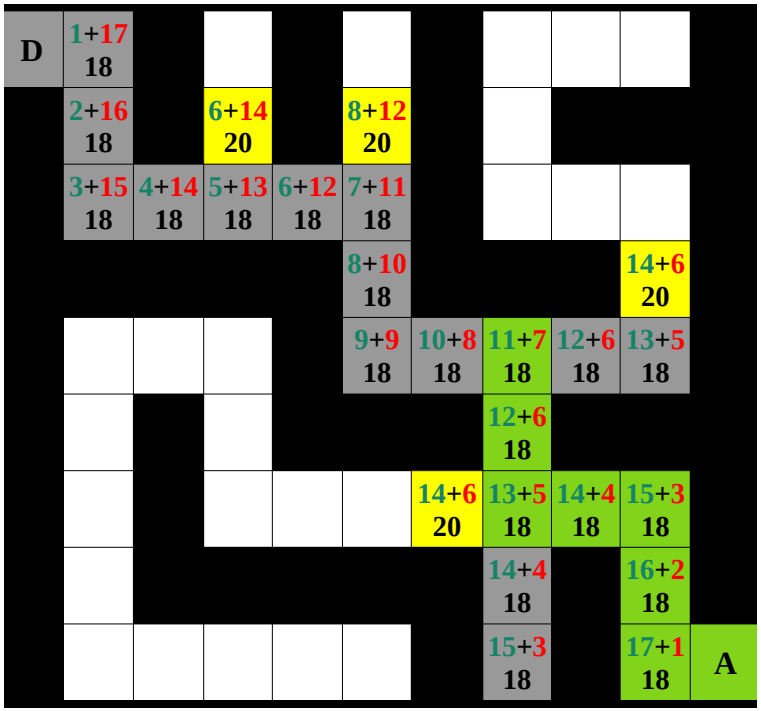
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



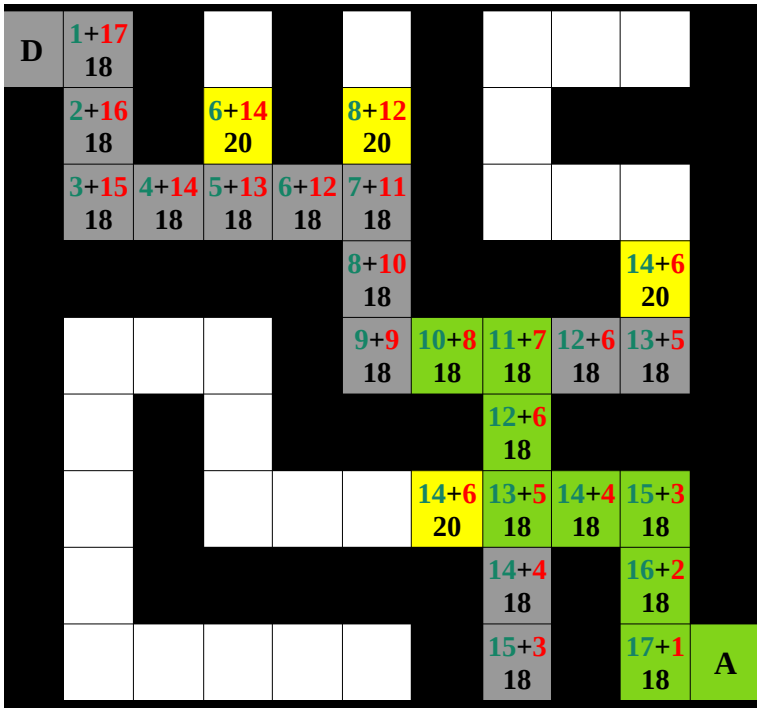
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



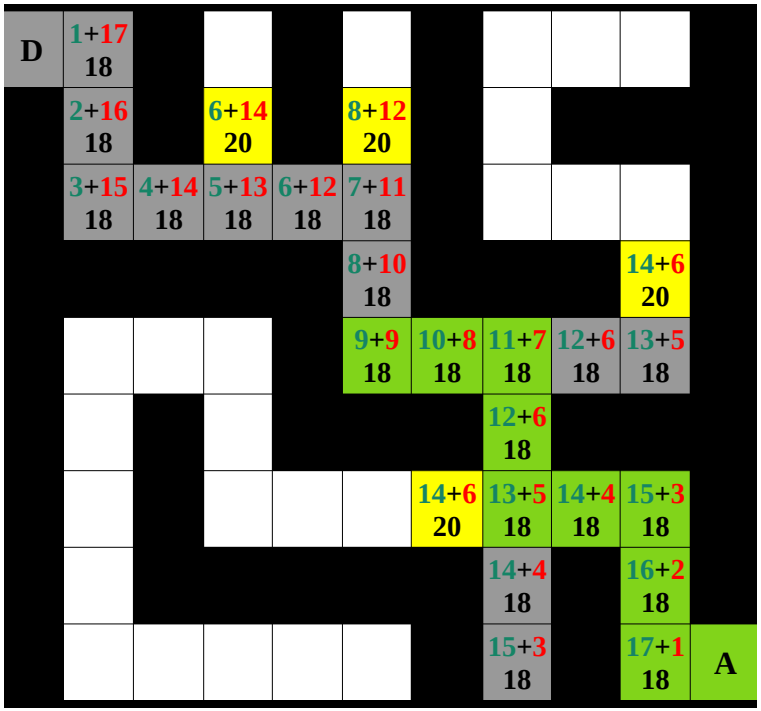
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



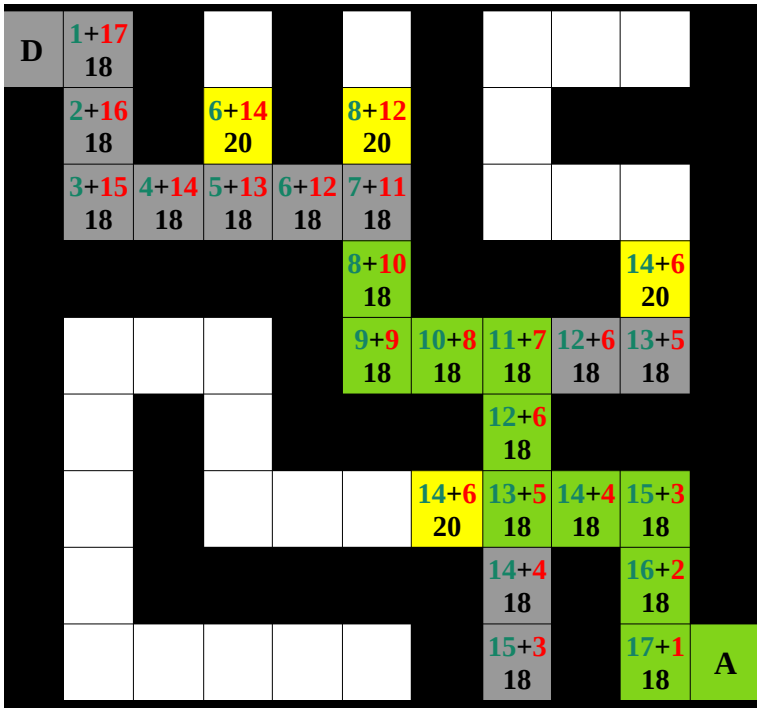
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



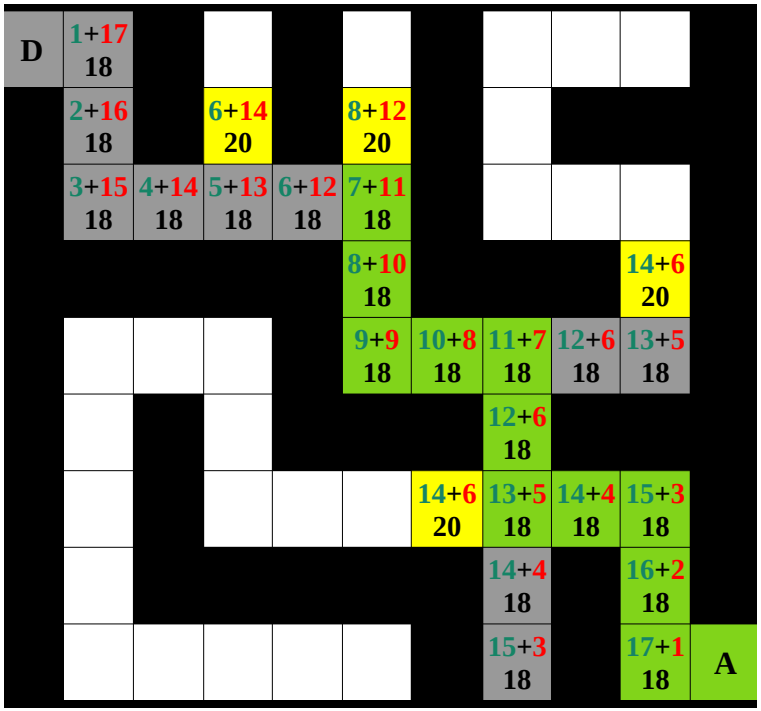
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



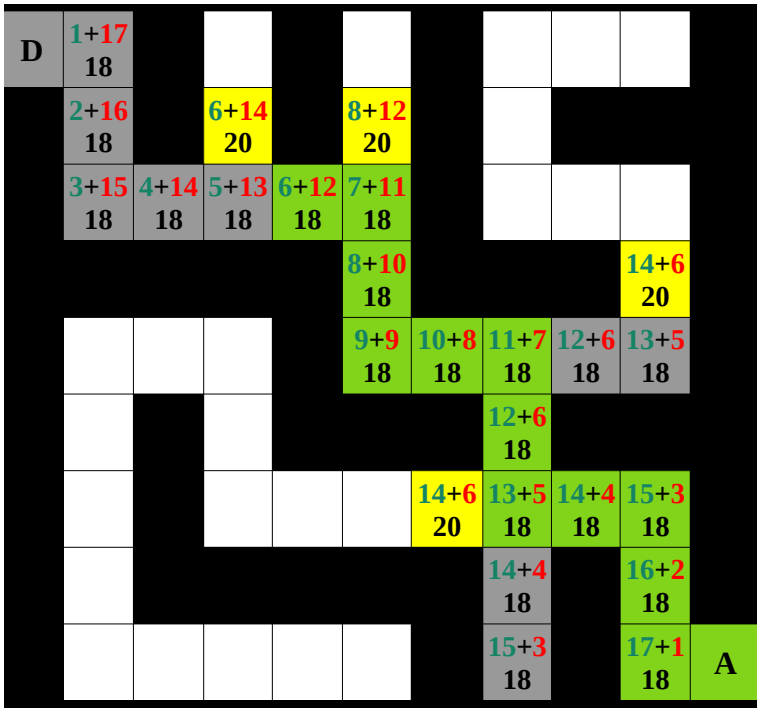
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



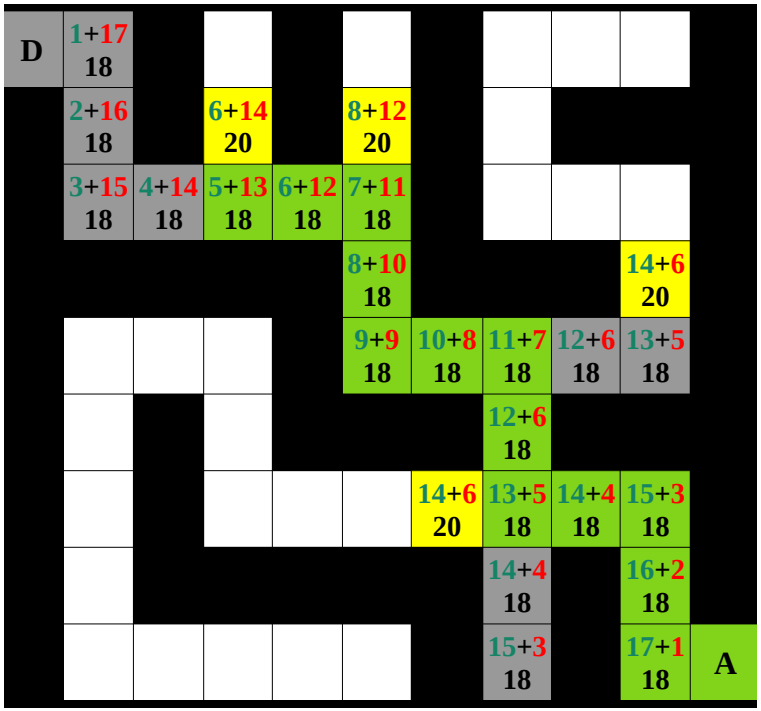
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



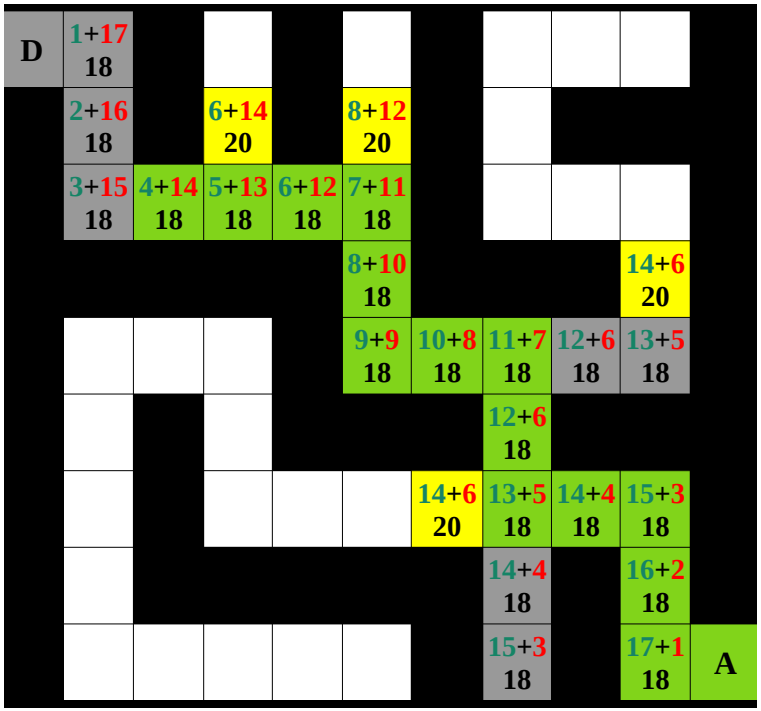
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



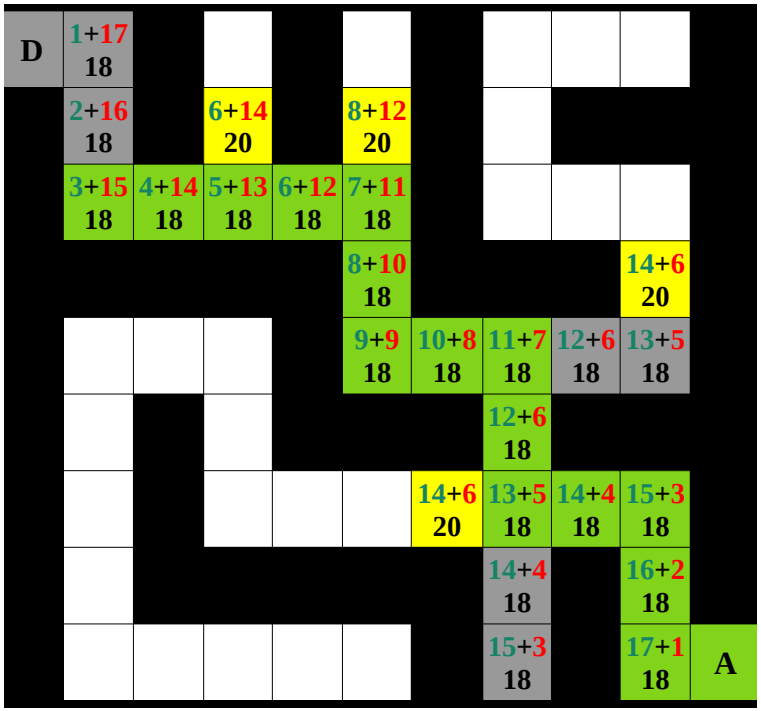
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



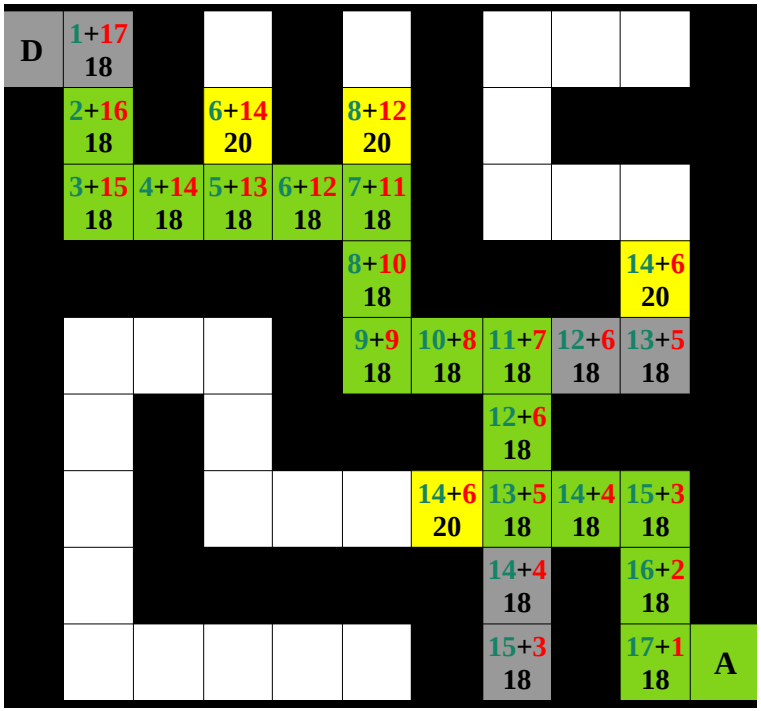
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



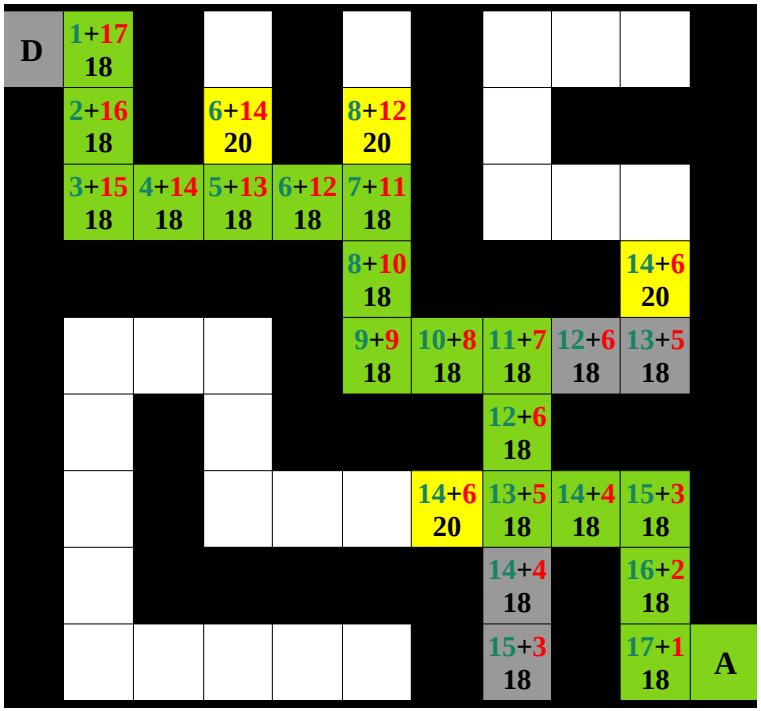
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



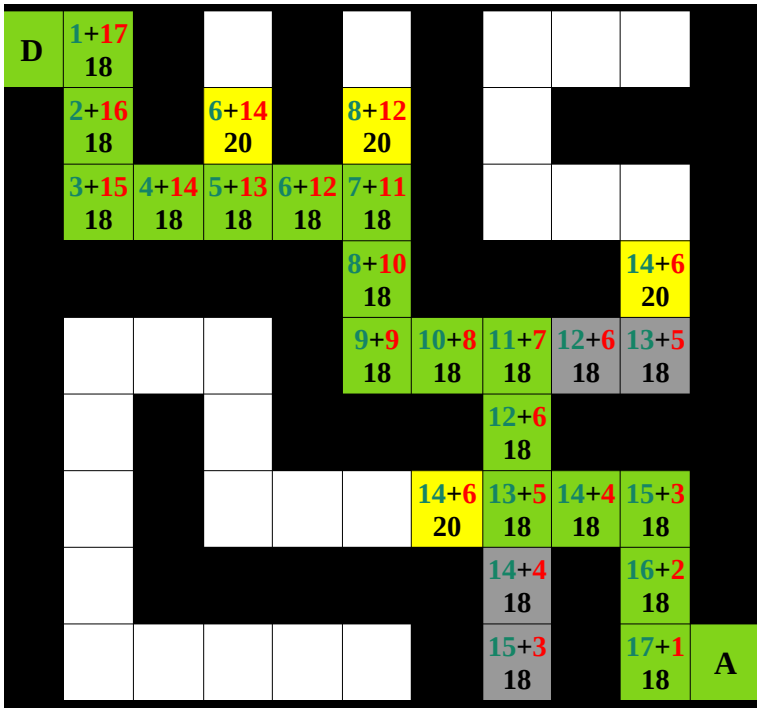
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



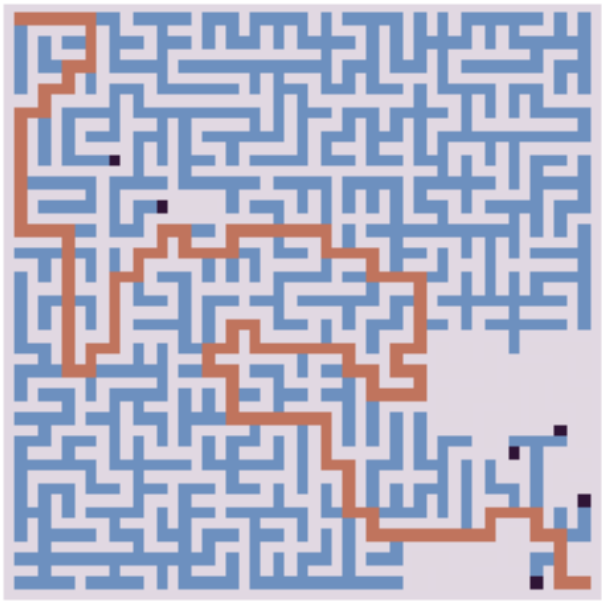
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Complexité

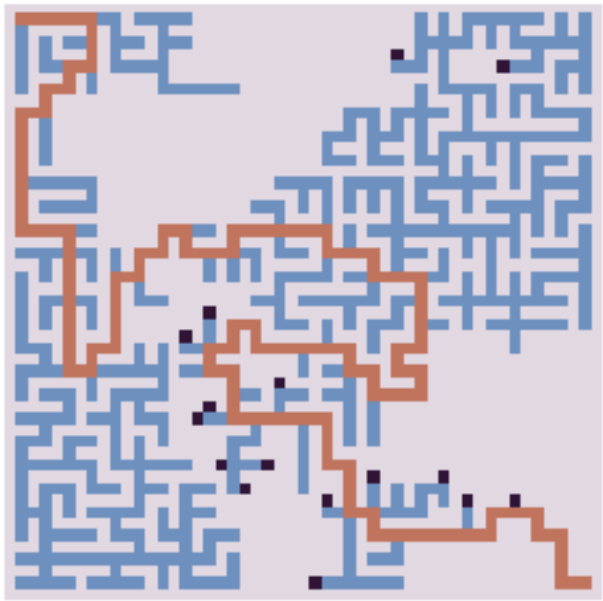
Le pire des cas sera réalisé par un labyrinthe dont le meilleur chemin revient souvent en arrière (s'éloigne de l'arrivée). Dans ce cas, aucun gain n'est réalisé par rapport à l'algorithme de Dijsktra.

Tests avec Python

A* with null heuristic



A* with Manhattan heuristic



Conclusion

- . . .
- Robot-Aspirateur