HMIN233 - Algorithmes d'exploration et de mouvement

Recherche de chemins

Suro François
(adaptation des cours de Jacques Ferber)

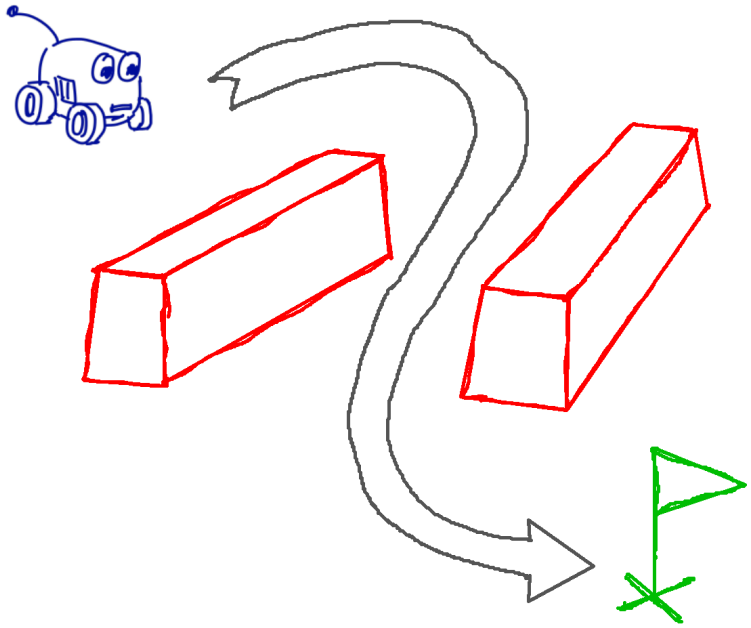
Université de Montpellier
Laboratoire d'informatique, de robotique
et de microélectronique de Montpellier

Janvier 2021

Comment planifier un chemin pour atteindre un objectif ?

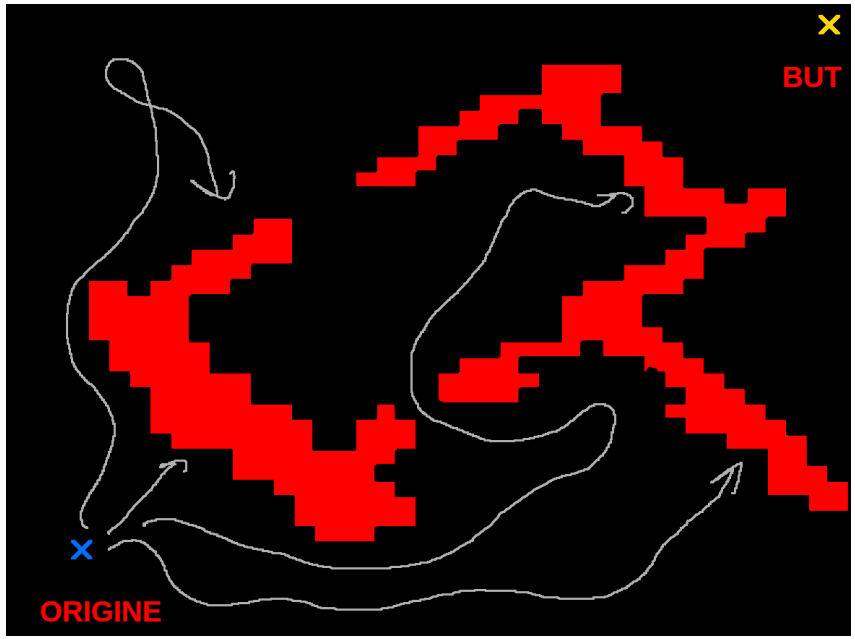
Quel algorithme utiliser pour planifier un chemin pour aller d'un point à un autre ?

- Différent de la situation réactive (agents situés) où l'agent avance et trouve son chemin au fur et à mesure.
- On prendra en compte un environnement défini sous forme de grille.



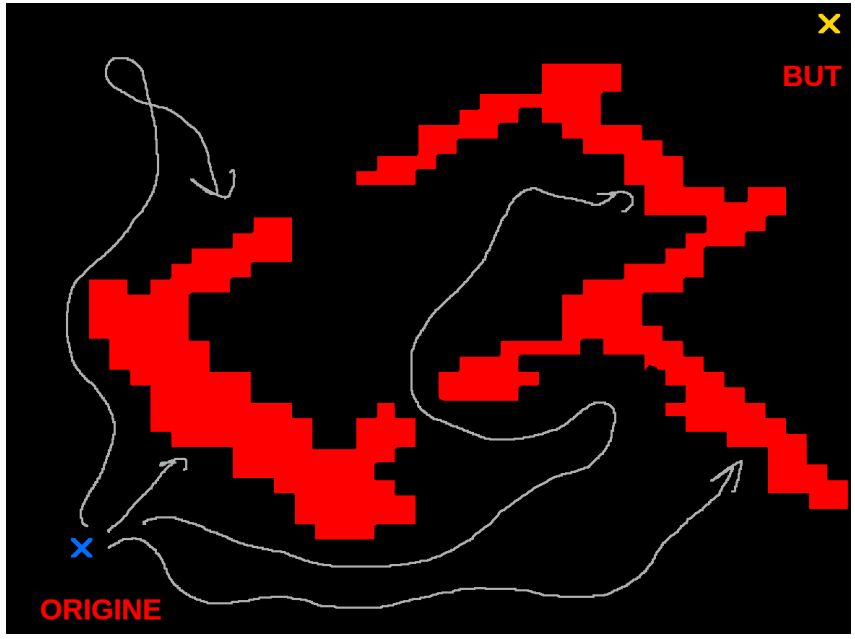
Aller d'un point à un autre

- l'agent (croix bleu en bas à gauche) cherche à aller au but (croix jaune en haut à droite).
- Comment trouver sa route ?
- Comment trouver le "meilleur" chemin ou tout du moins un "bon" chemin.



Les difficultés

1. Trouver le but le plus rapidement possible.
2. Ne pas rester bloqué par les obstacles.
3. Optimiser le chemin une fois trouvé le but.

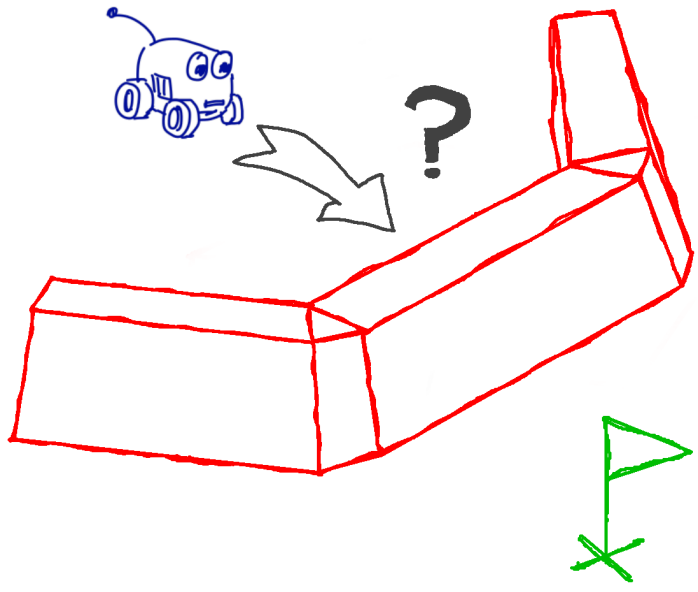


1: Trouver le but

- Idée: aller dans la direction où l'on se rapproche du but.
- Heuristique: Comme si on utilisait une boussole.
- Sinon on explore jusqu'à ce qu'on trouve.

2: Ne pas rester bloqué

- En allant directement vers le but, l'agent est bloqué.
- Il peut être coincé dans un minimum local.



3. Optimiser le chemin à parcourir

- Éviter de refaire tous les détours inutiles.
- Évidemment par rapport aux lieux que l'on a visités.

Trois possibilités

Algorithme de Dijkstra

Explorer l'espace et essayer de trouver le "meilleur" chemin à partir de ce que l'on a exploré.

Algorithme de "meilleur d'abord" (best-first)

Essayer d'aller vers le but en utilisant une heuristique (se rapprocher du but).

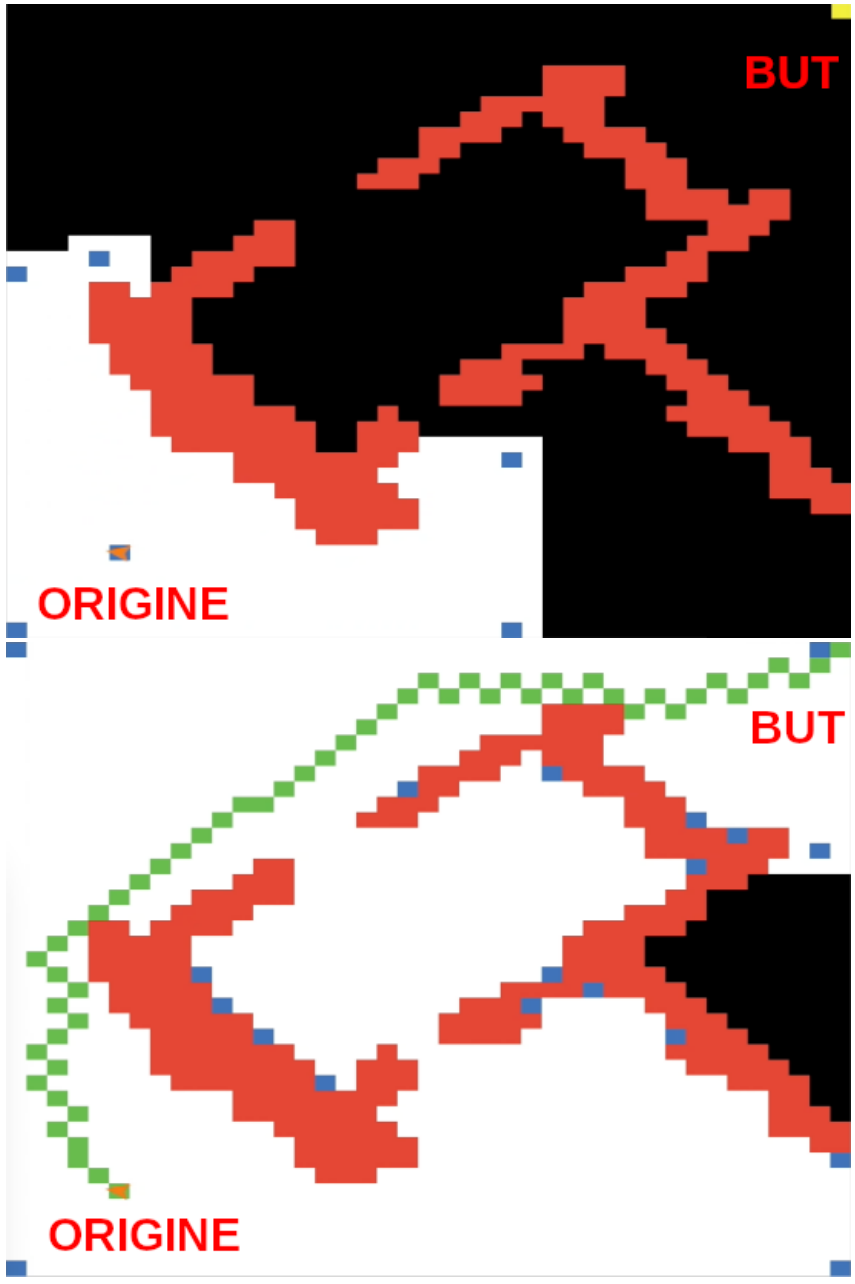
Algorithme A*

Essayer de combiner ce qu'il y de mieux dans ces deux approches.

Algorithme de Dijkstra

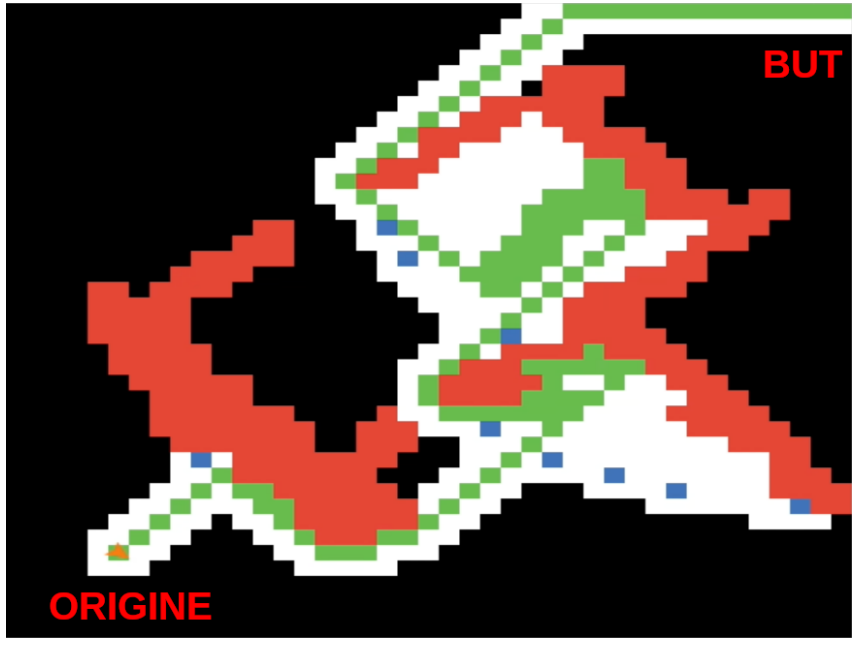
- On ne sait pas où se trouve le but (on sait juste quand on y est).
- Il n'y a plus qu'à tout parcourir...
- Mais comme on a tout parcouru (ou presque) on est sûr d'obtenir le "meilleur" chemin pour arriver.

Rechercher coûteuse mais résultat parfait.



le meilleur d'abord (best first)

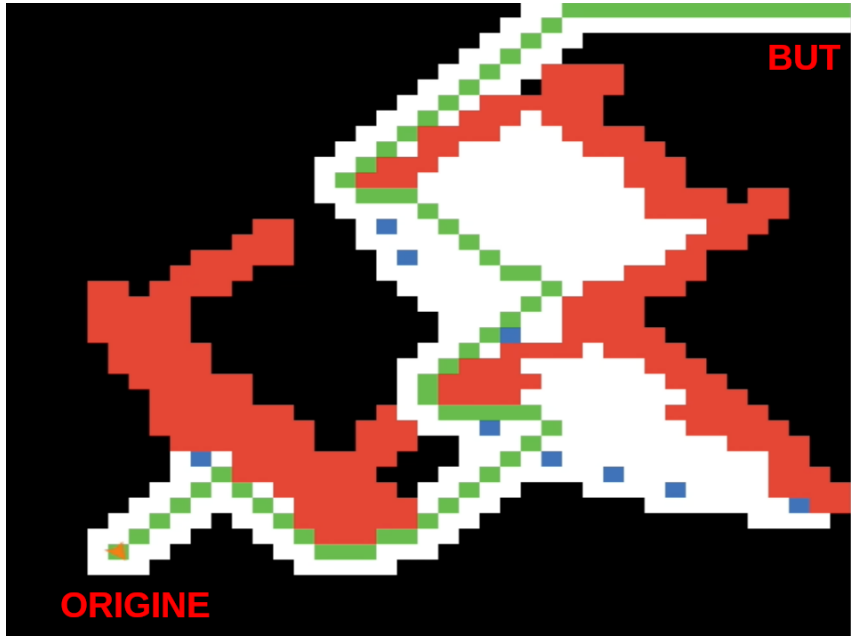
- On utilise l'heuristique qui consiste à **aller globalement dans la bonne direction** (la direction qui diminue la distance vis à vis du but).
  - Risque de blocages.
  - Dans ce cas, on explore autour.
- Au retour, on redonne le chemin que l'on a parcouru.



Recherche rapide mais résultat pas du tout optimal !!!

A*: prendre les bonnes idées de l'un et de l'autre

- On va vers le but en utilisant une **heuristique**, comme avec le meilleur d'abord.
- On **optimise** le chemin parcouru en tenant compte de **la distance à l'origine**, comme avec Dijkstra.



Bon compromis entre temps de recherche et qualité du résultat.
Très utilisé en jeux et I.A.

Comparaison des 3 algorithmes

- C'est le même "moteur" qui permet de faire fonctionner ces différents algorithmes.
- Ne diffèrent que par quelques points

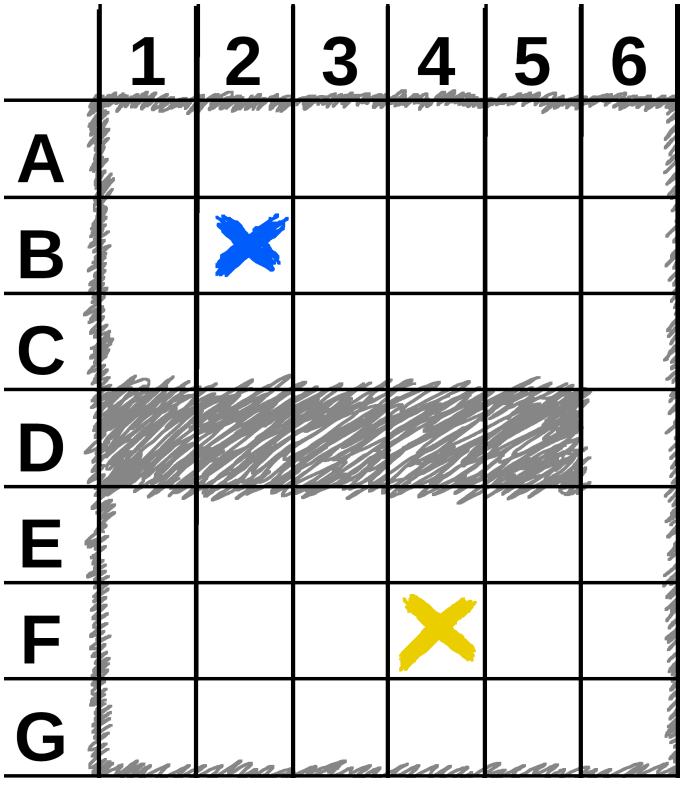
	Best-first	**Dijkstra**	**A***
Choix du noeud à considérer	Plus petit h (heuristique distance au but)	Plus petit g(n) (coût du chemin parcouru)	Plus petit h (heuristique de distance au but)
Tri de la liste	h(n)	g(n)	f(n)=g(n)+h(n)
Calcul du "parent"	Noeud courant	Parent du noeud n identique au courant si $g(n) < g(courant)$	Parent du noeud n identique au courant $sif(n) < f(courant)$
Remarque	Calcul rapide, mais chemin tortueux	Calcul lent mais possibilité d'avoir le "meilleur" chemin	Calcul rapide et obtention d'un "bon" chemin (parfois le meilleur)

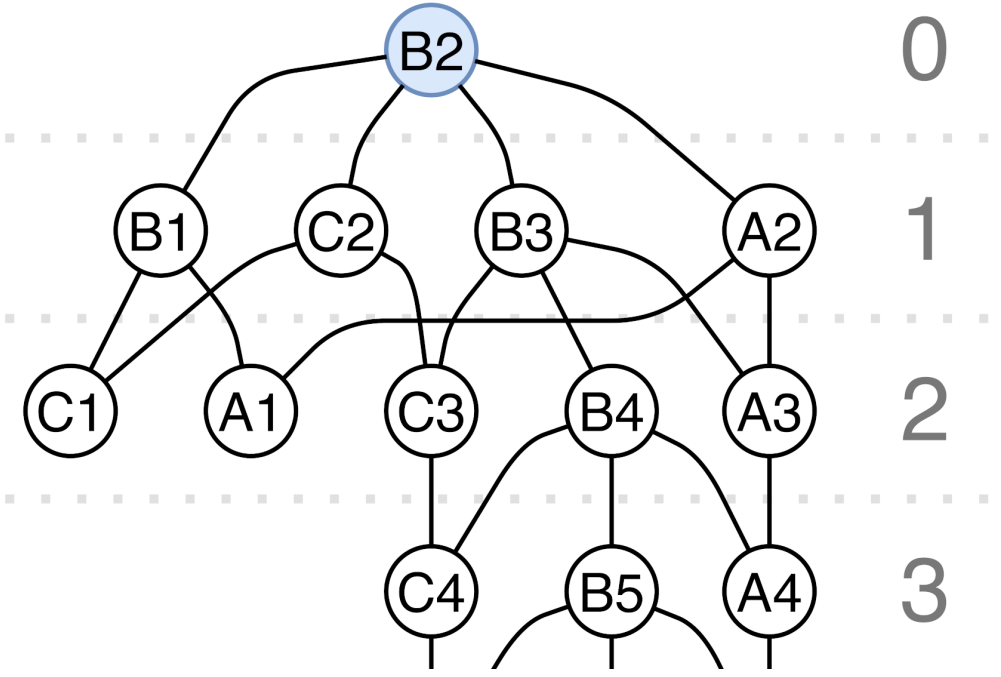
Le moteur de recherche

Principe : on parcours une suite de noeuds dans un graphe

- **Current** : le noeud courant que l'on analyse.
- **In-nodes** : liste des noeuds à analyser, ordonnés suivant un critères (heuristique, coûts ...).
- **Out-nodes** : liste des noeuds déjà traités (pour éviter de boucler).

Etat initial et graphe de recherche





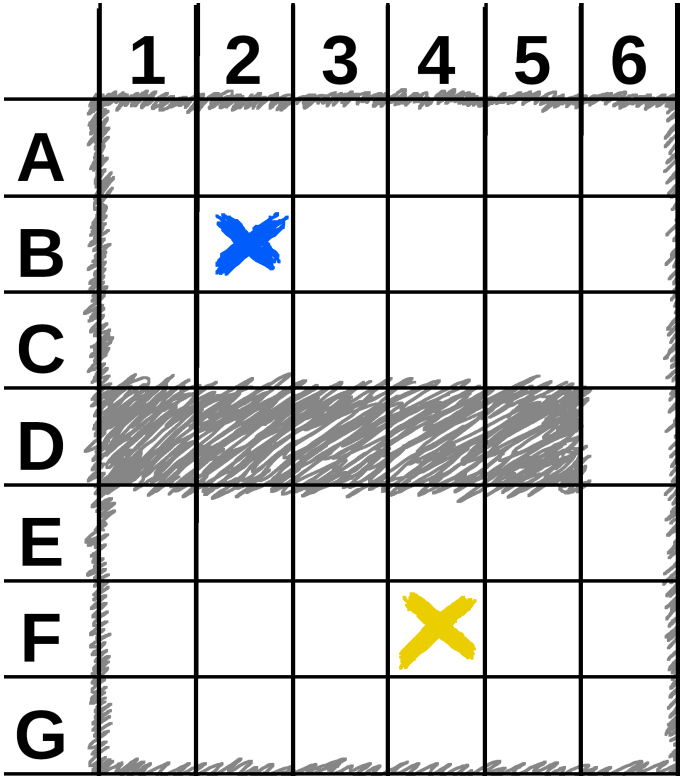
Origine: B2

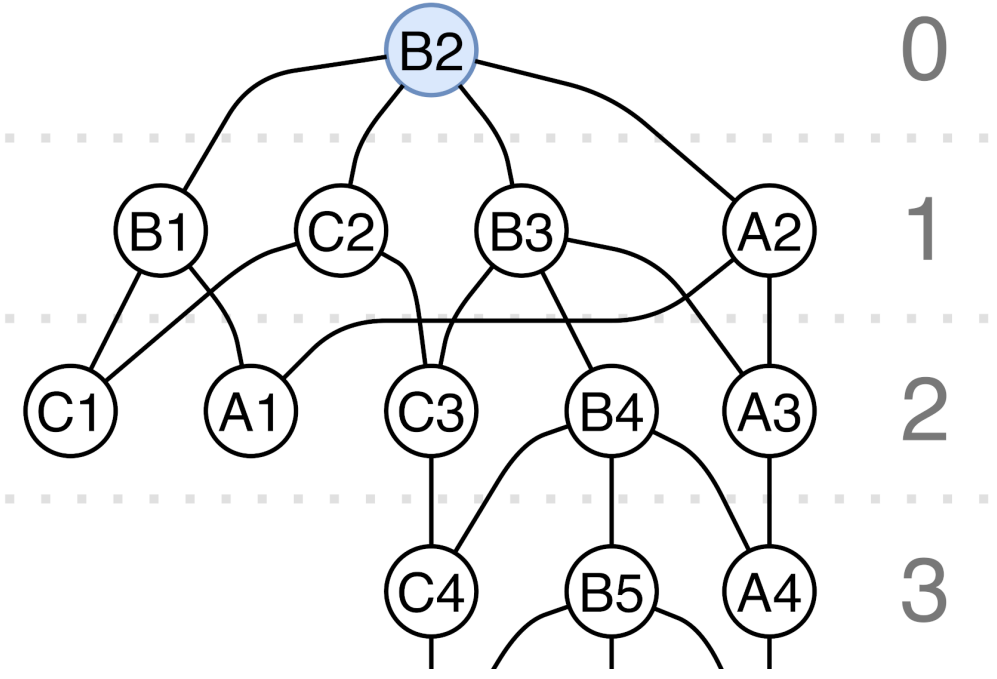
But: F4

Obstacle : en gris

Graphe de recherche : En "1 coup" on peut atteindre les voisins de la case courante.

Fonctionnement

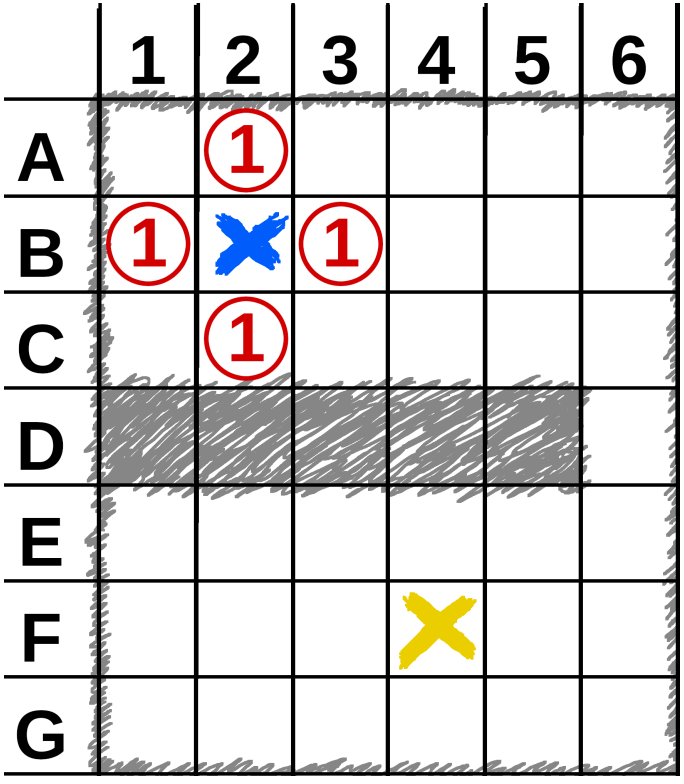


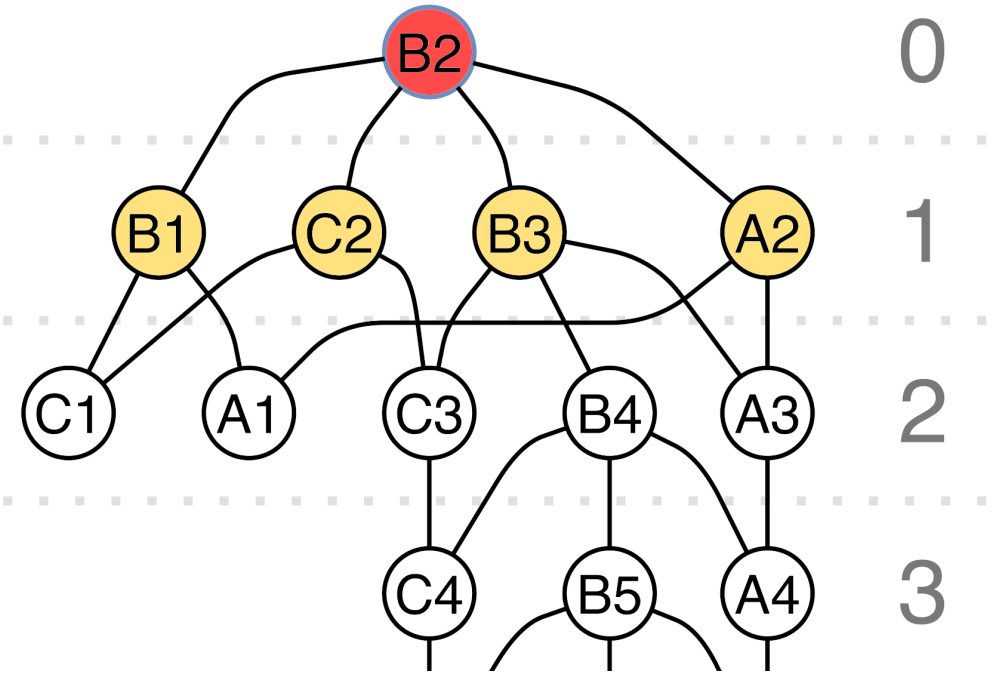


IN :[B2]

OUT:[]

Fonctionnement

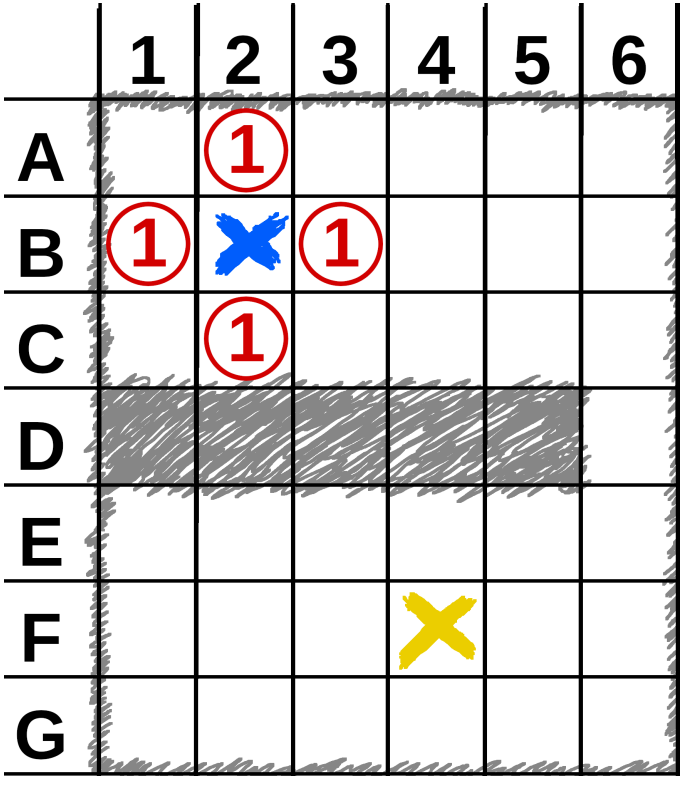


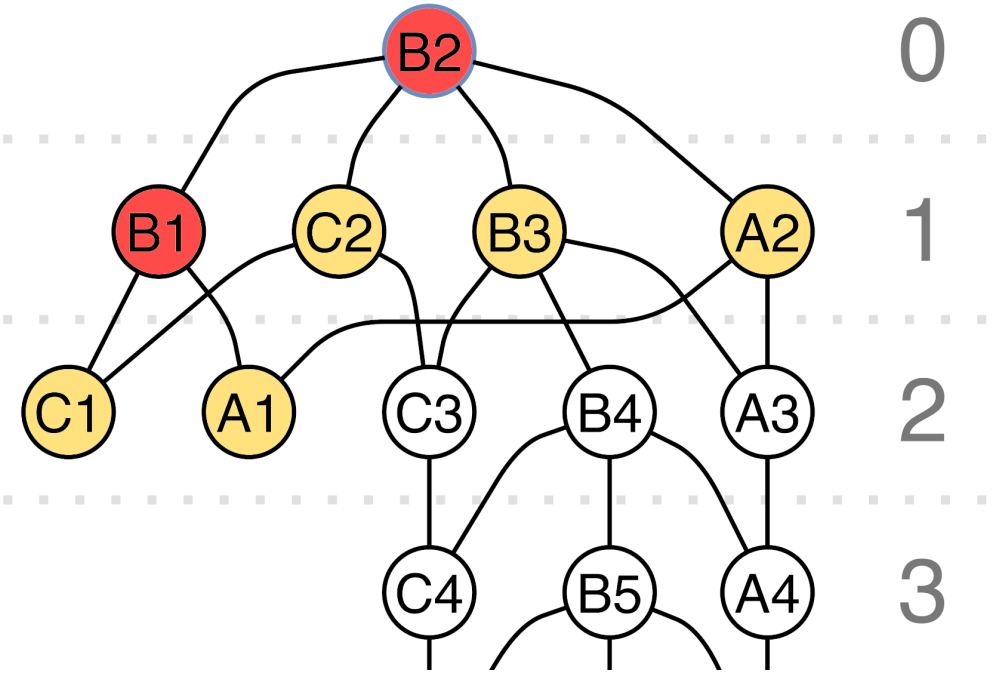


IN :[] ← B1 C2 A2 B3

OUT:[B2]

Fonctionnement

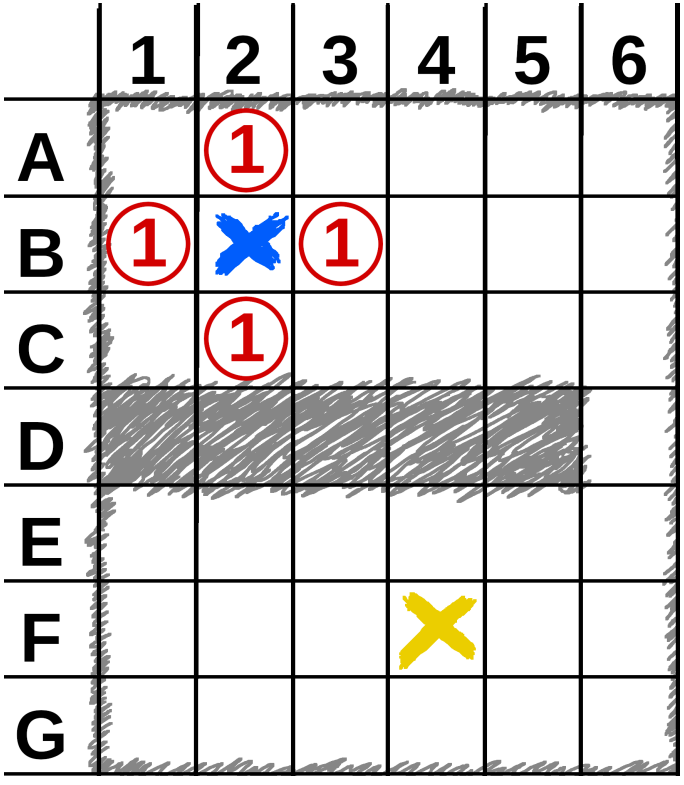


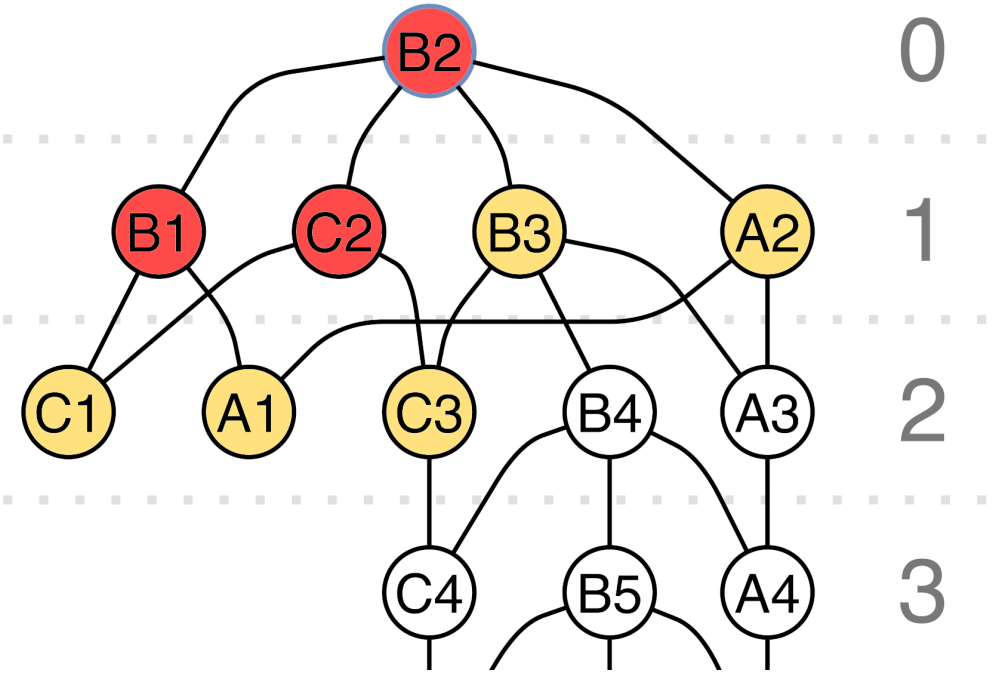


IN :[C2 A2 B3] ← C1 A1

OUT:[B2 B1]

Fonctionnement

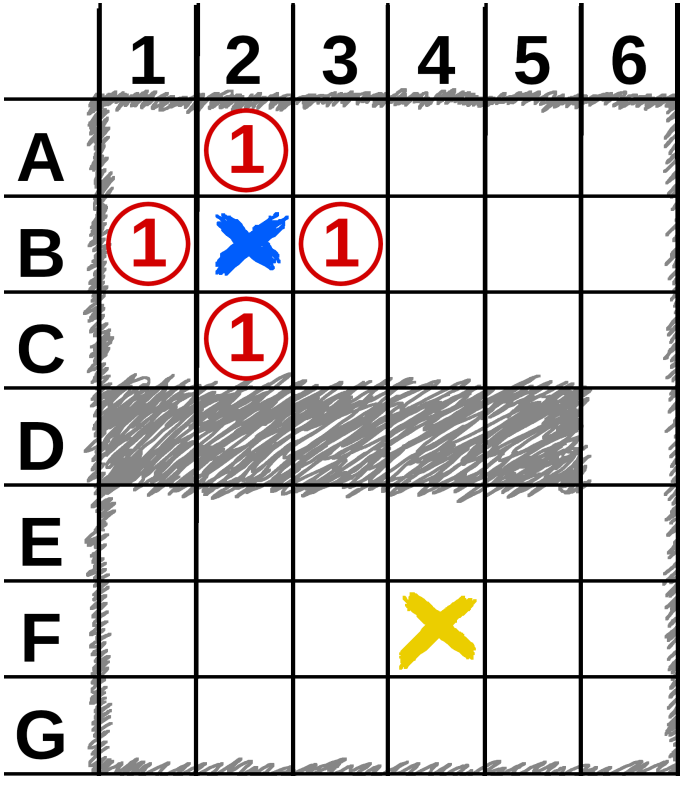


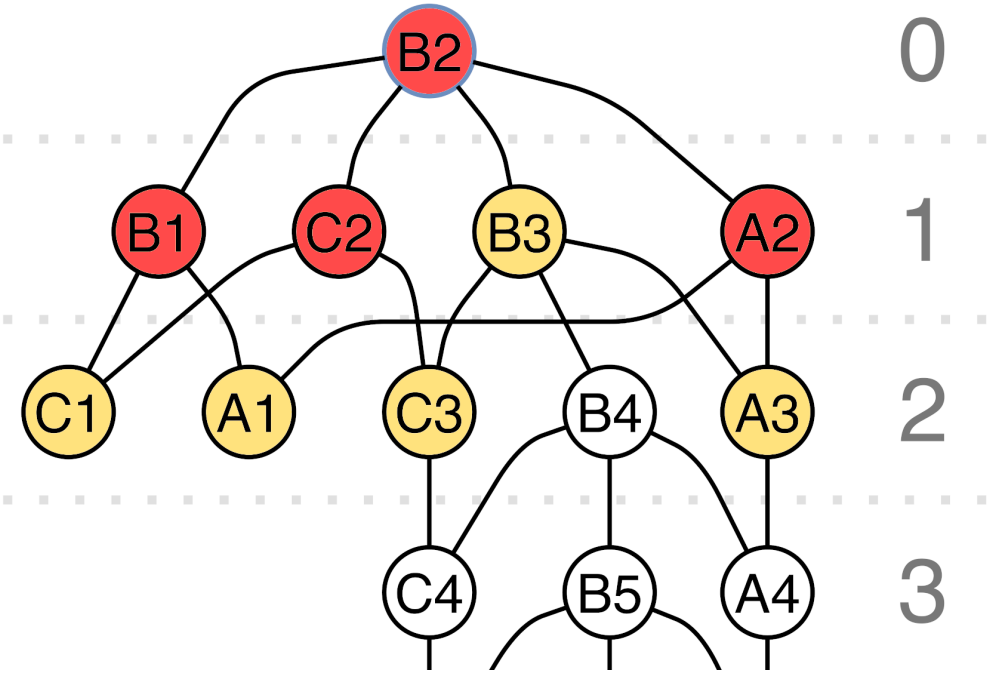


IN :[A2 B3 C1 A1]← C1 C3

OUT:[B2 B1 C2]

Fonctionnement

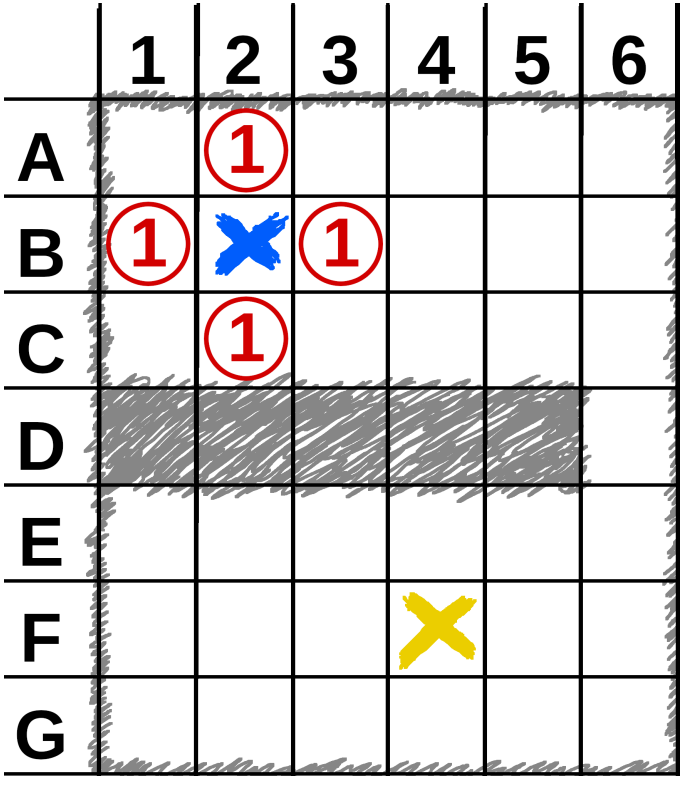


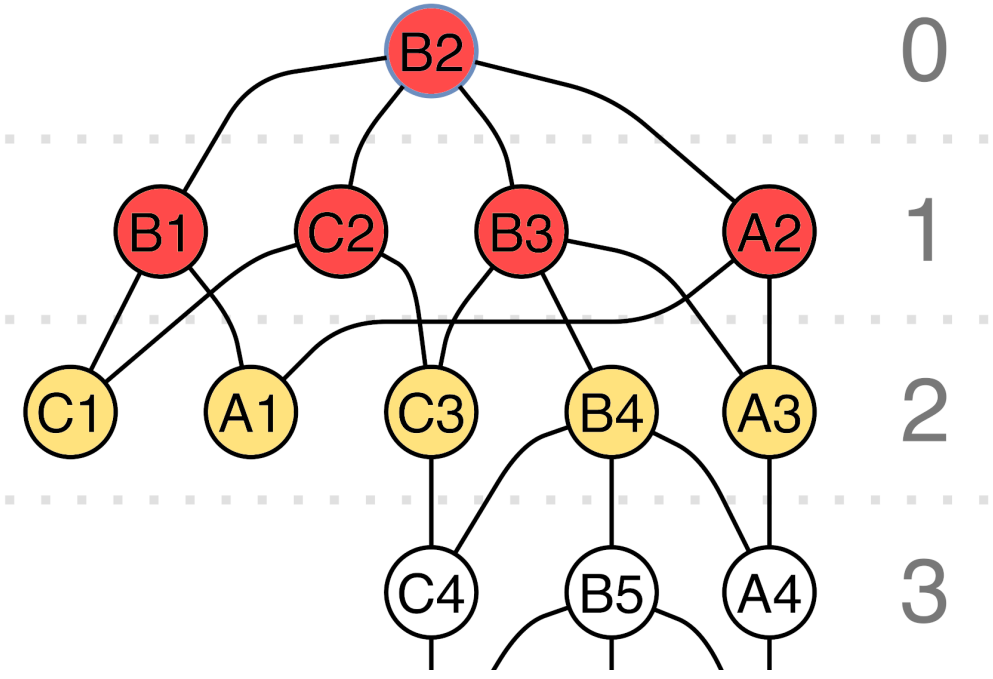


IN :[B3 C1 A1 C3]← A1 A3

OUT:[B2 B1 C2 A2]

Fonctionnement

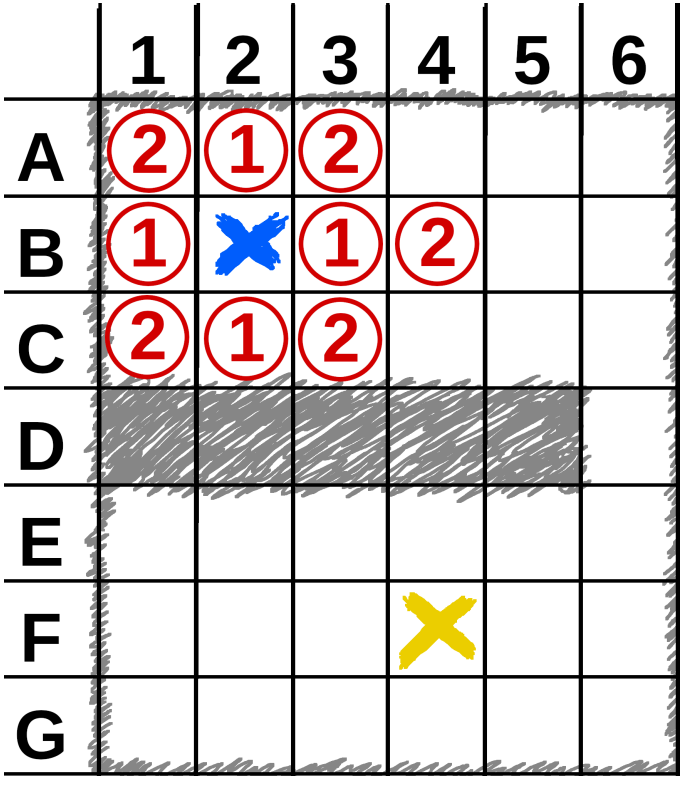


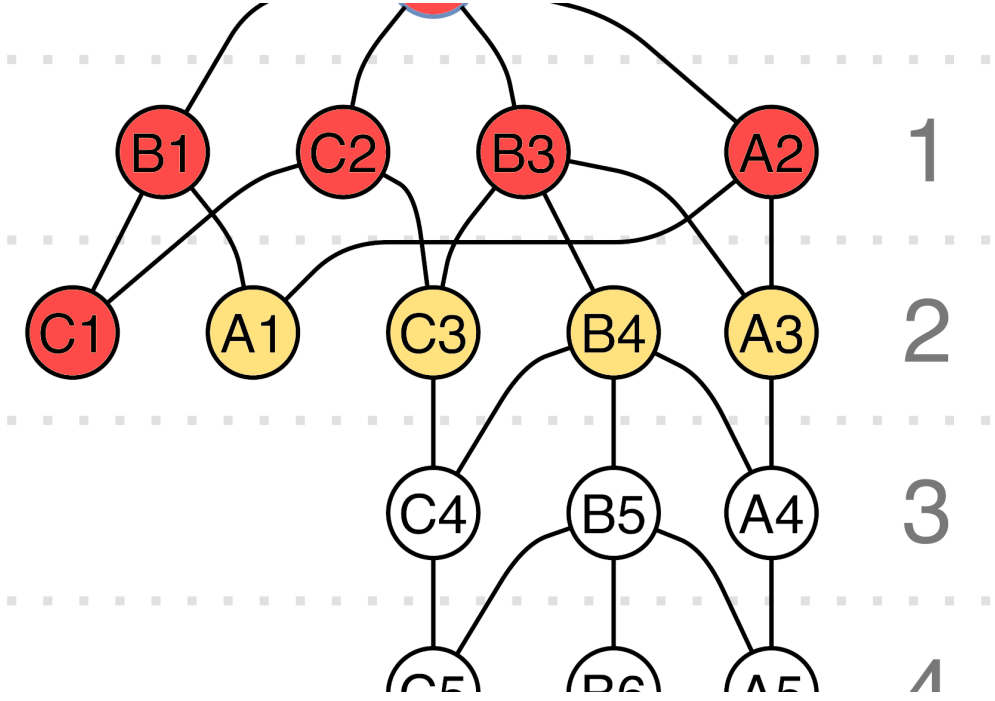


IN :[C1 A1 C3 A3]← C3 A3 B4

OUT:[B2 B1 C2 A2 B3]

Fonctionnement

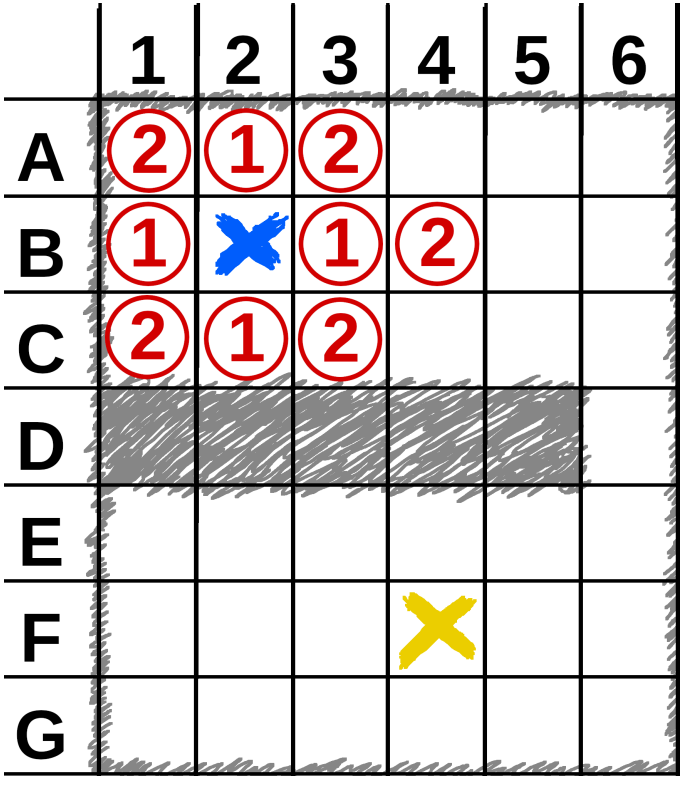


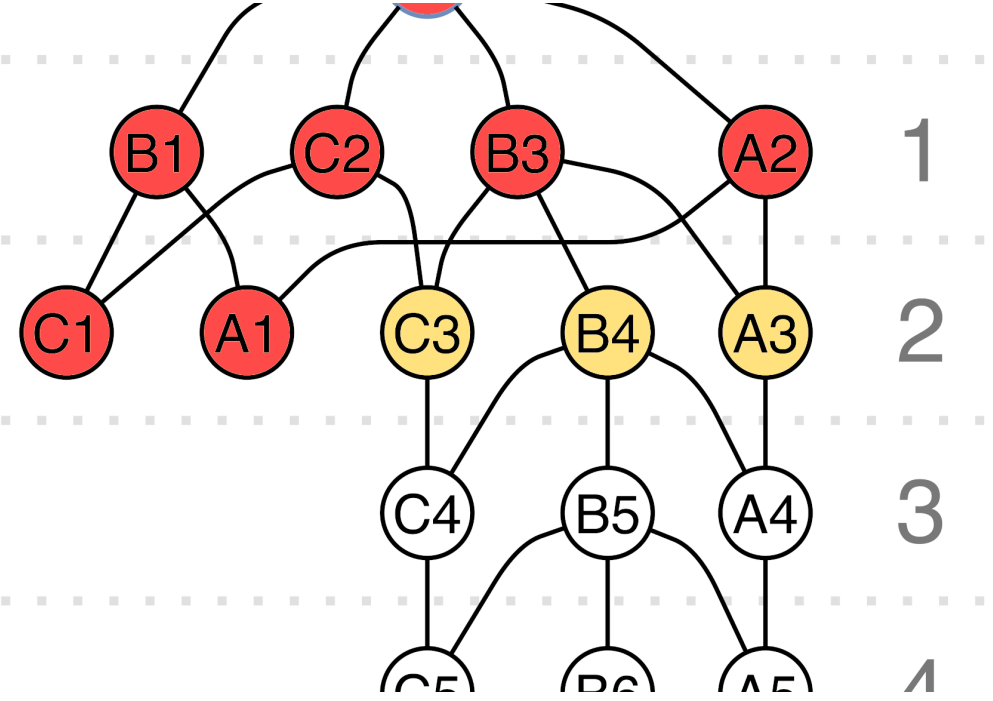


IN :[A1 C3 A3 B4] $\leftarrow$ B1 C2

OUT:[B2 B1 C2 A2 B3 C1]

Fonctionnement

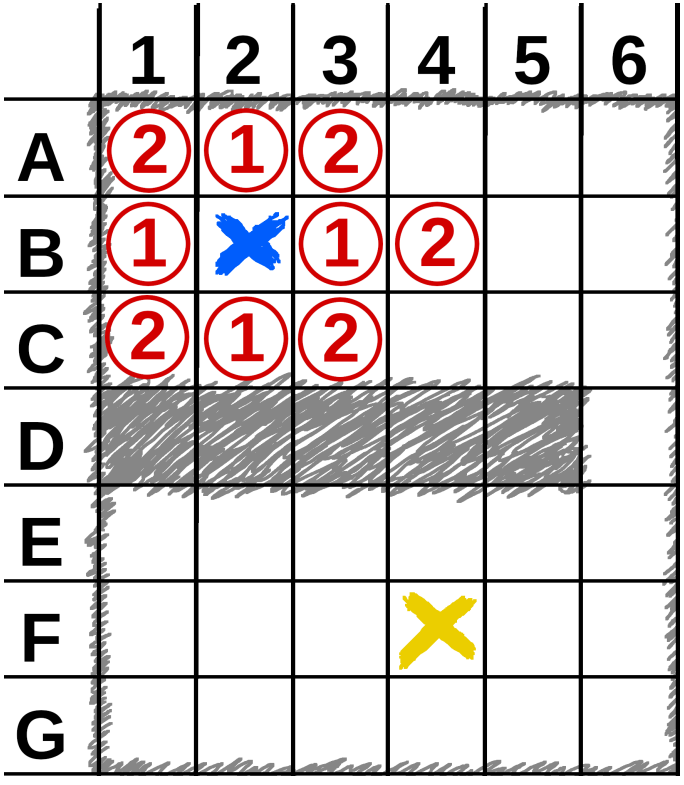


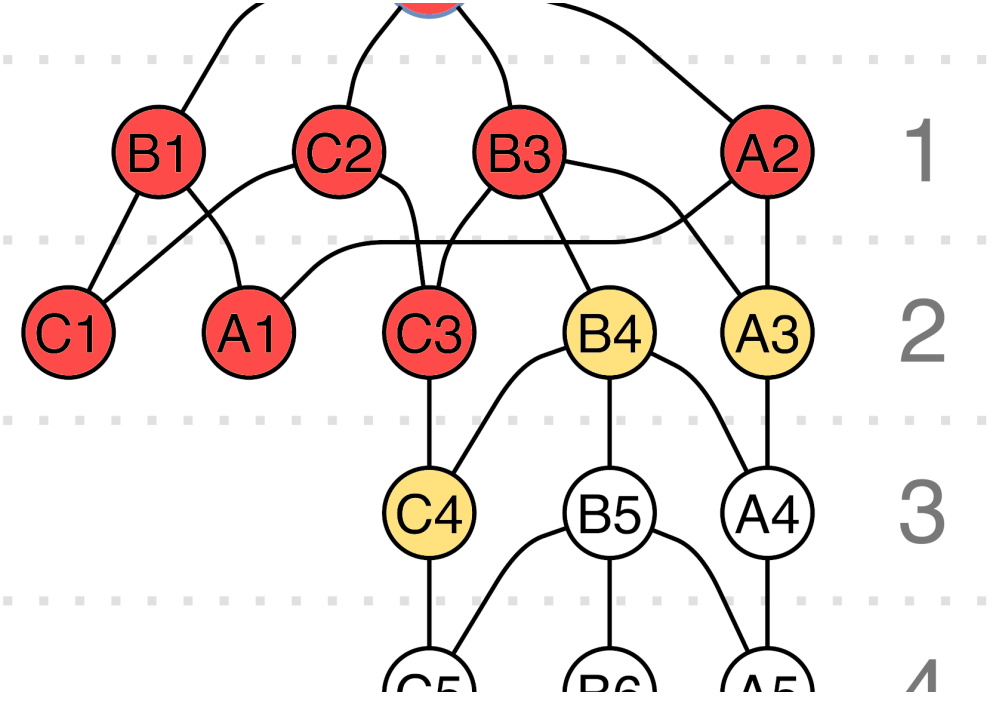


IN :[C3 A3 B4] ← B1 A2

OUT:[B2 B1 C2 A2 B3 C1 A1]

Fonctionnement

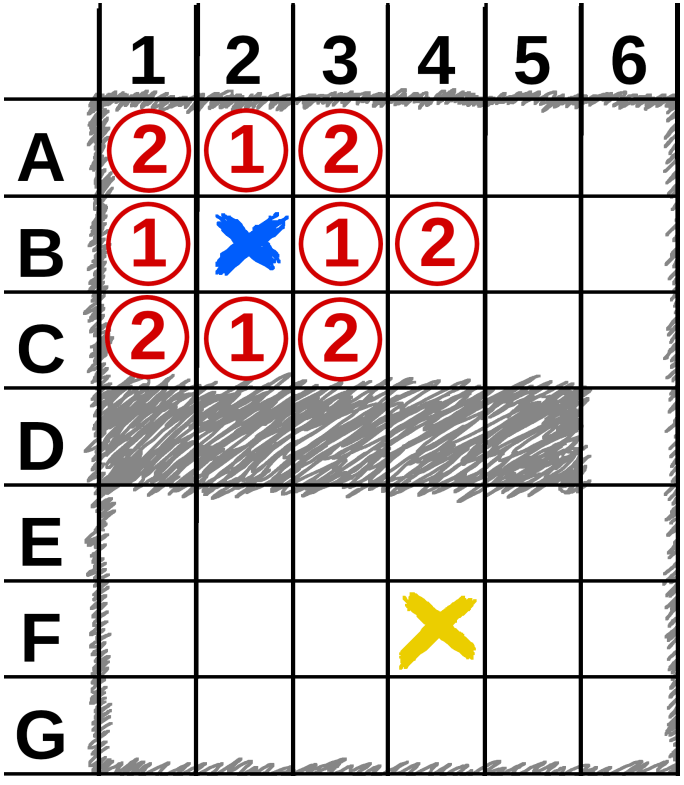


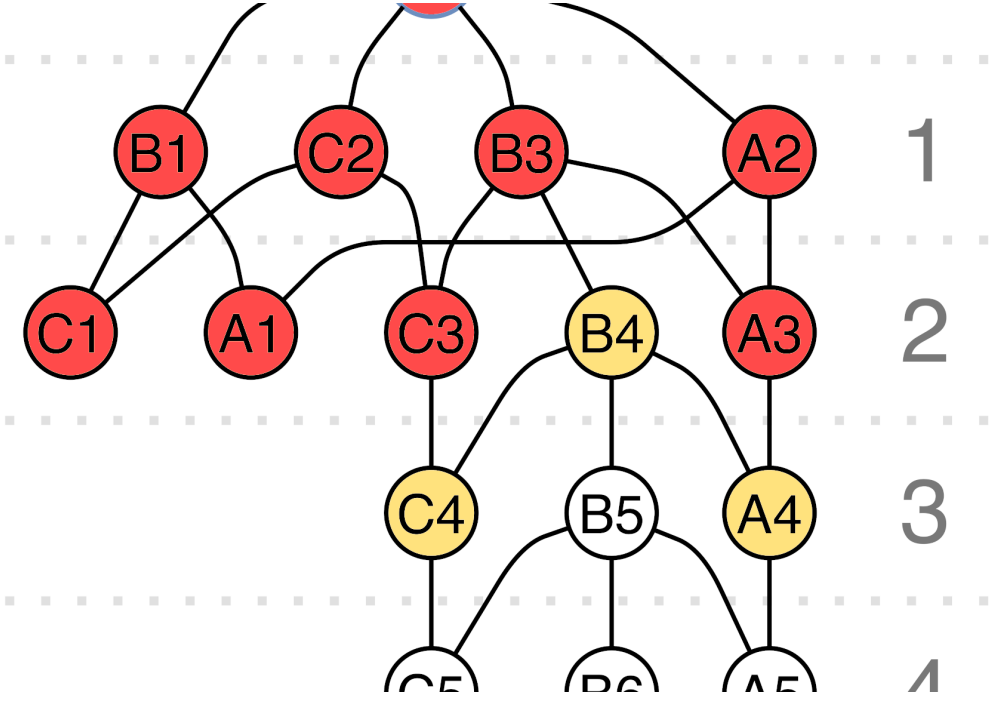


IN :[A3 B4] ← C2 B3 C4

OUT:[B2 B1 C2 A2 B3 C1 A1 C3]

Fonctionnement

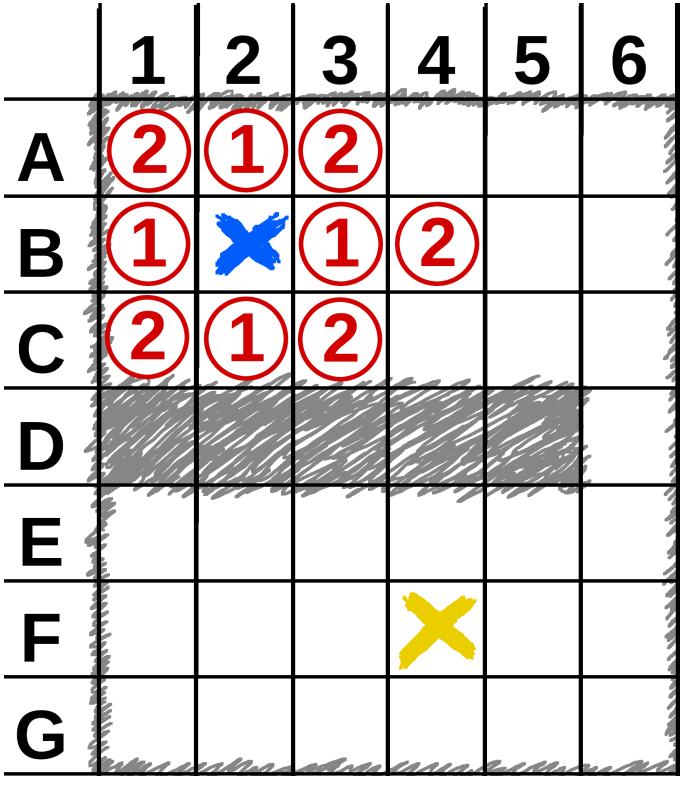


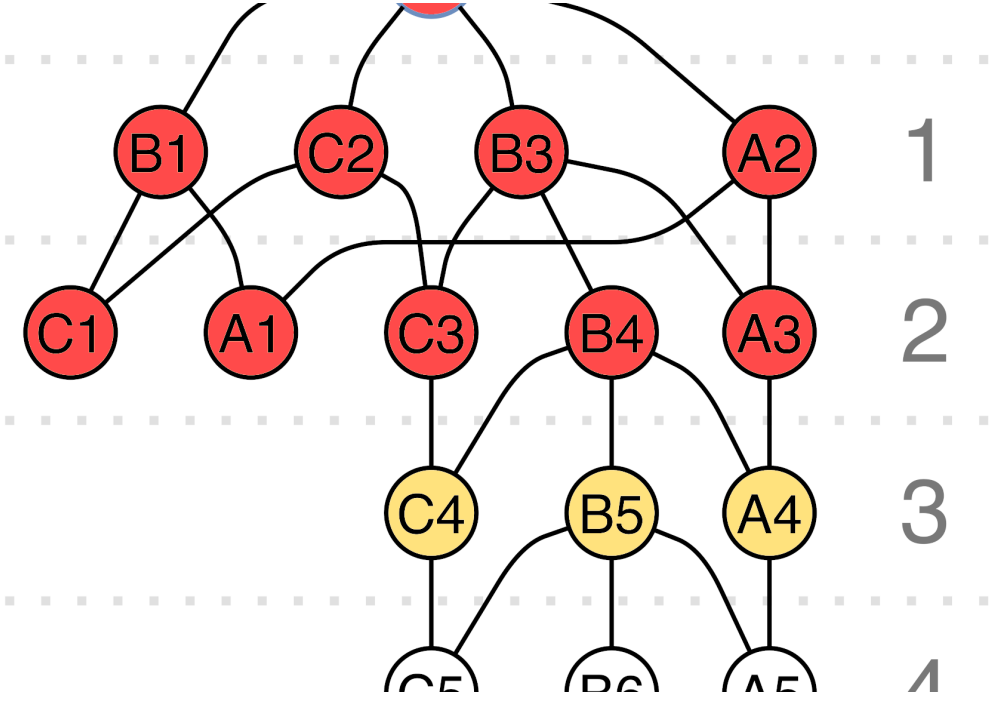


IN :[B4 C4] ← A2 B3 A4

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3]

Fonctionnement

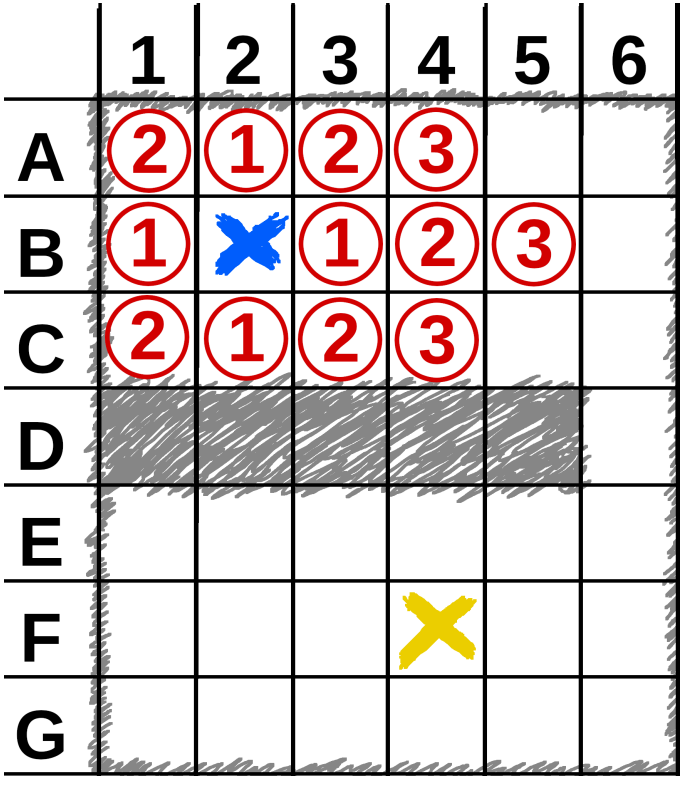


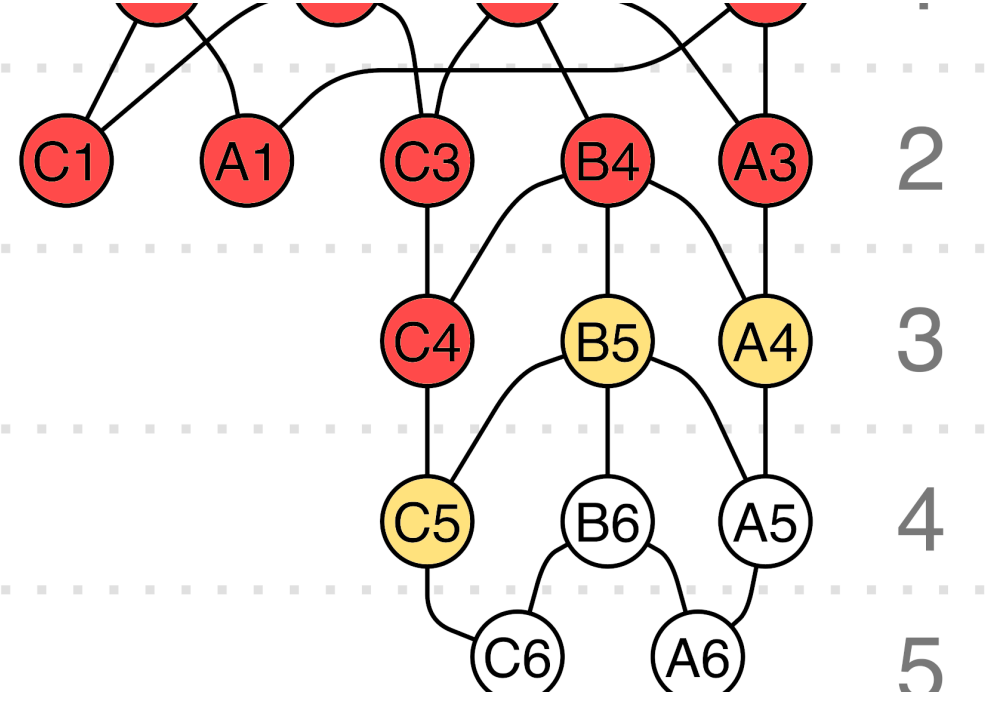


IN :[C4 A4] ← A4 C4 B5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4]

Fonctionnement

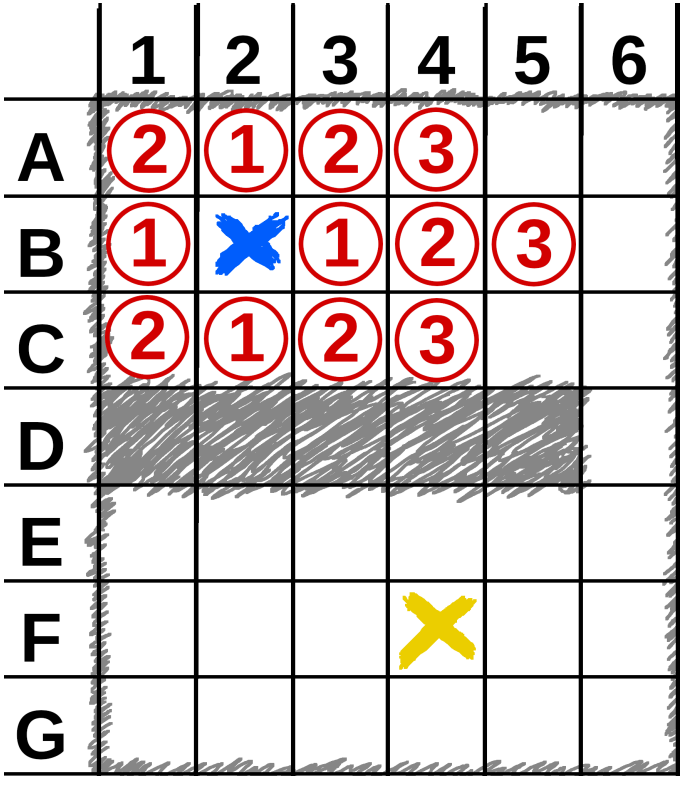


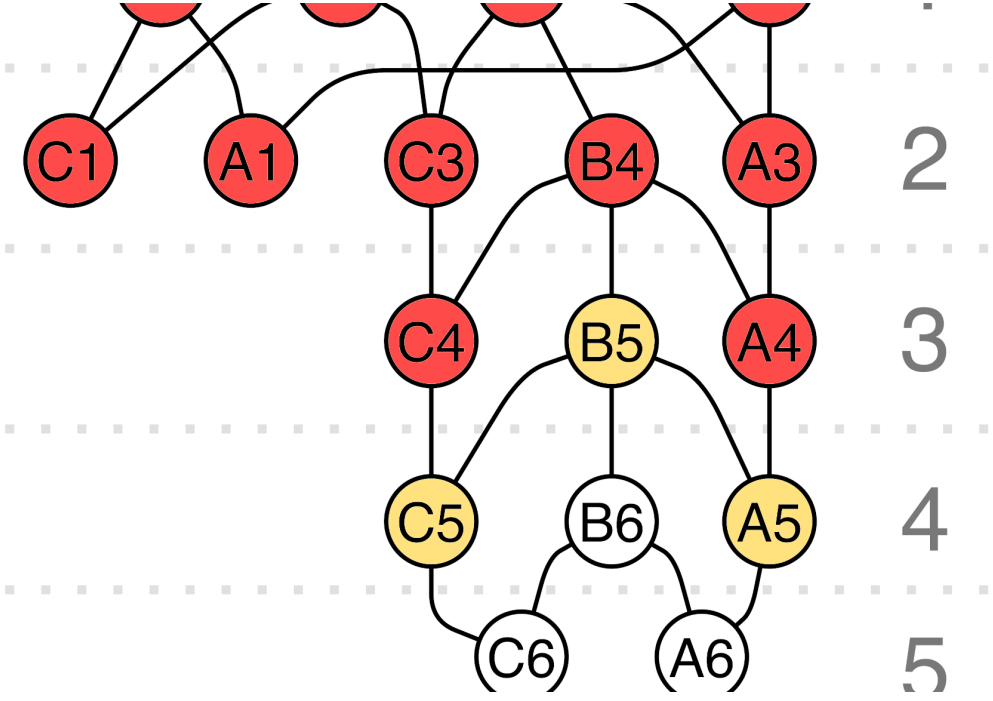


IN :[A4 B5] ← C3 B4 C5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4]

Fonctionnement

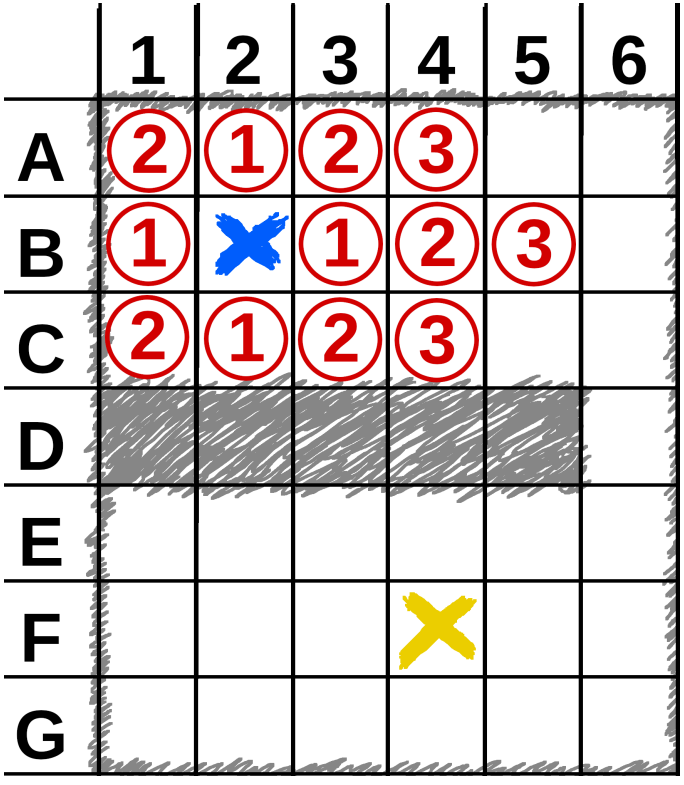


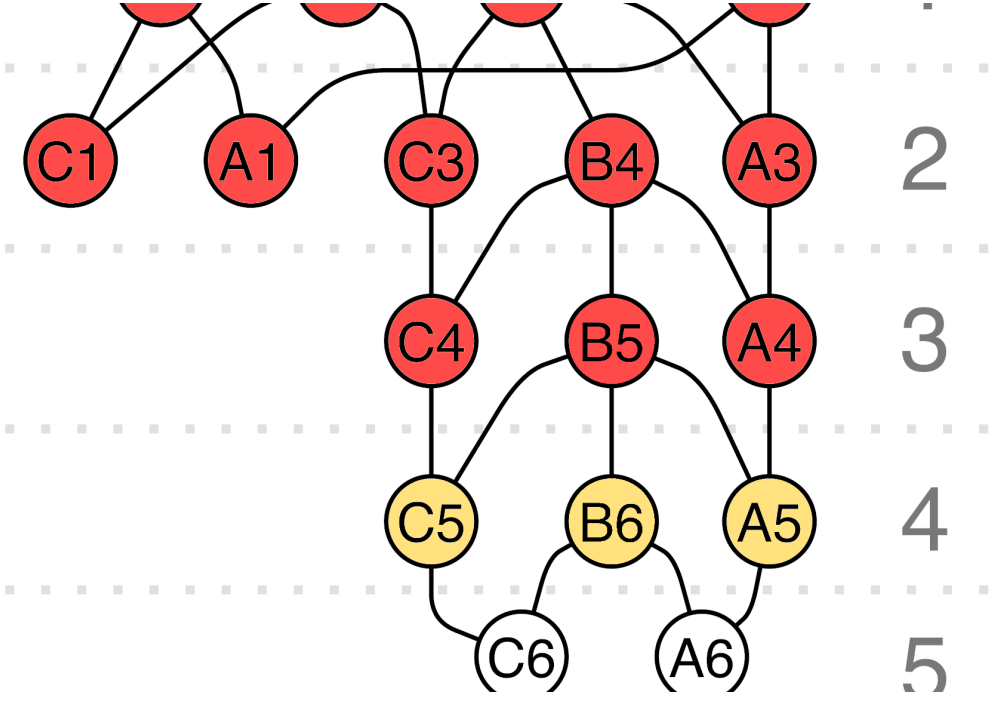


IN :[B5 C5] ← A3 B4 A5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4]

Fonctionnement

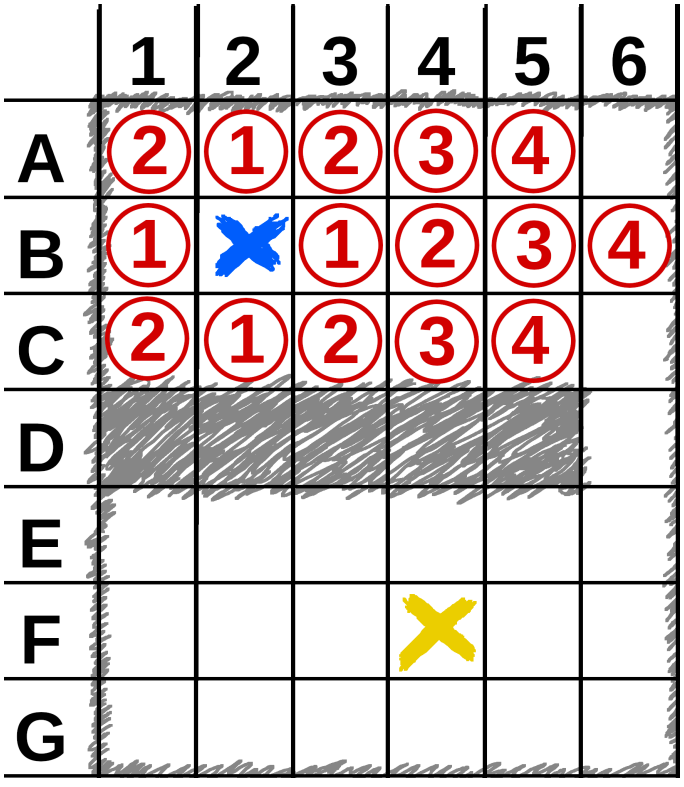


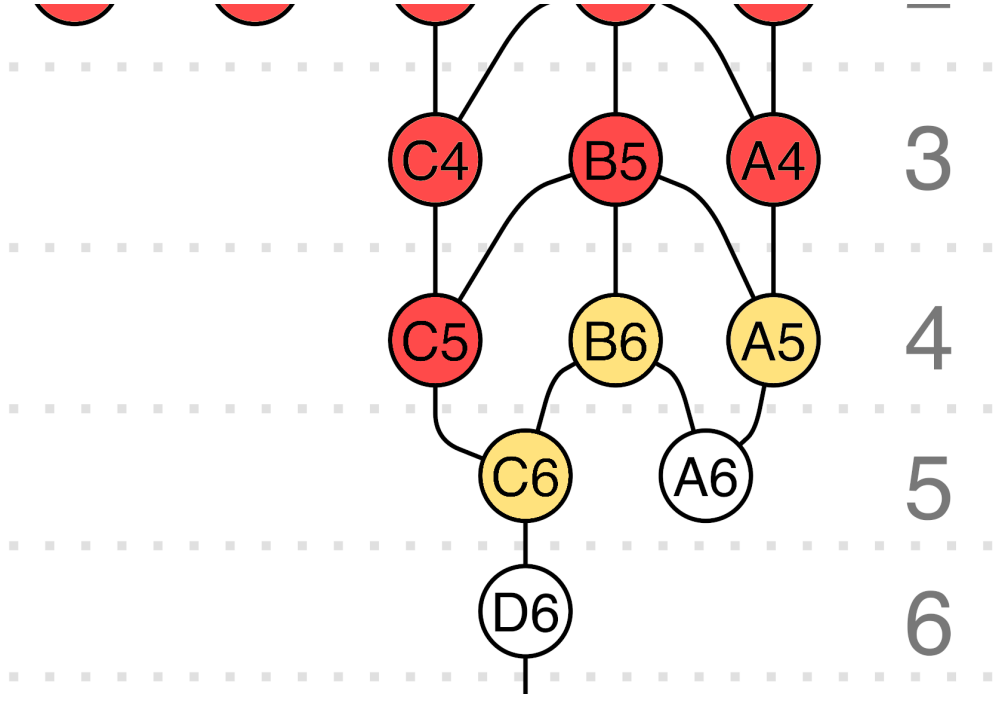


IN :[C5 A5] ← A5 B6 C5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5]

Fonctionnement

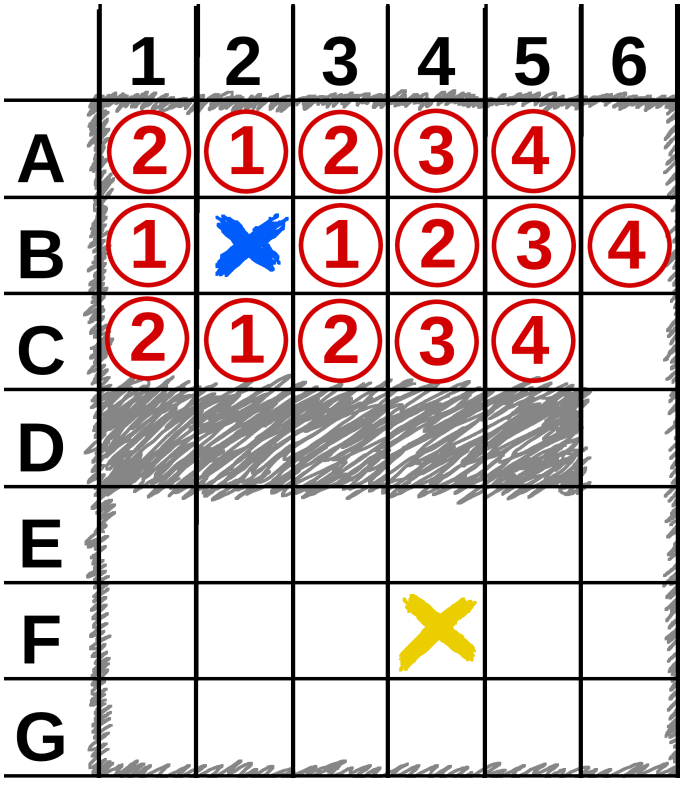


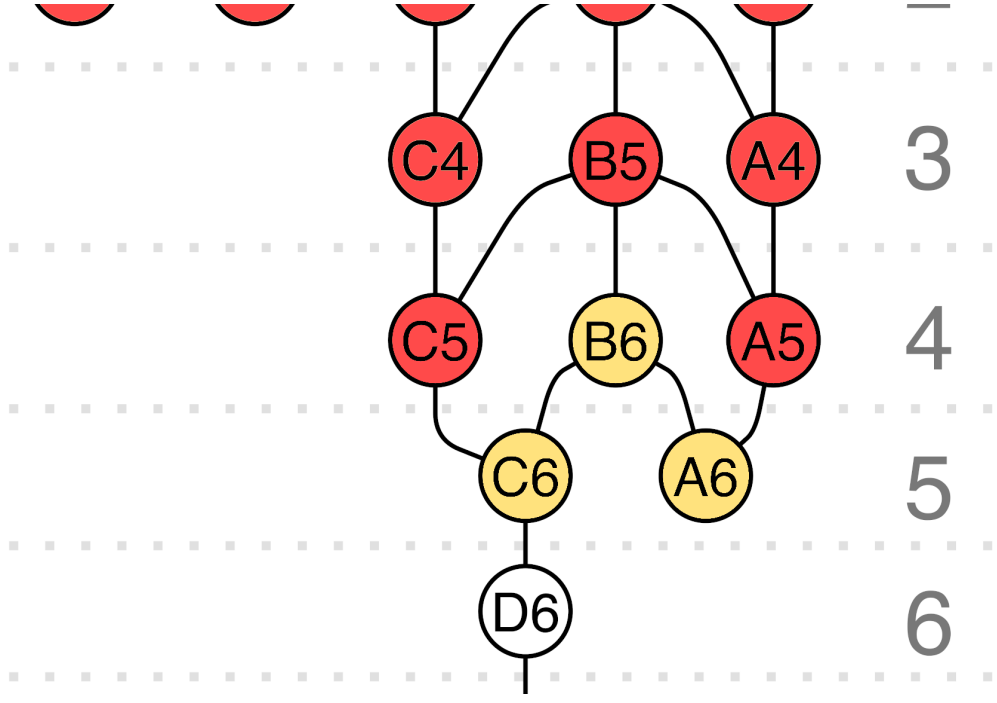


IN :[A5 B6] ← C4 B5 C6

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5]

Fonctionnement

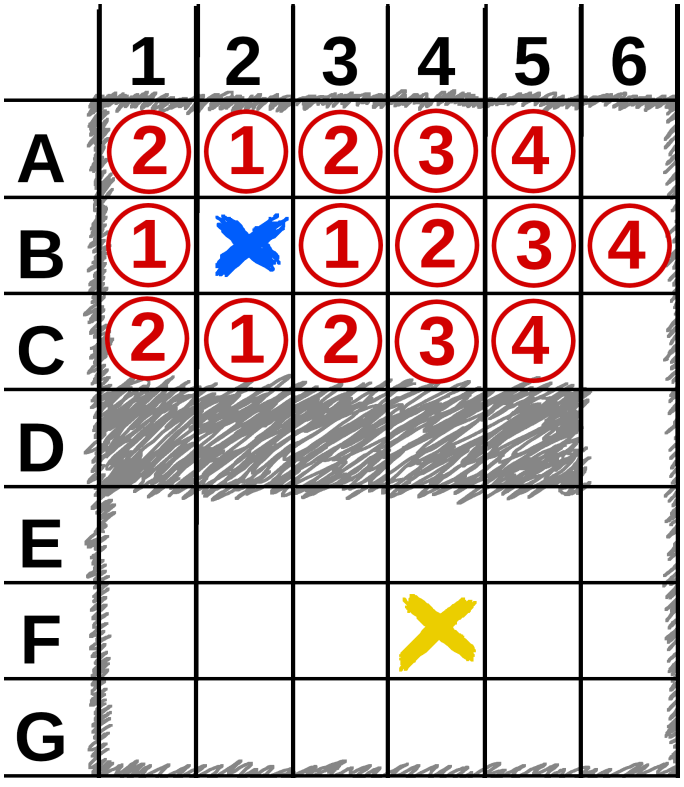


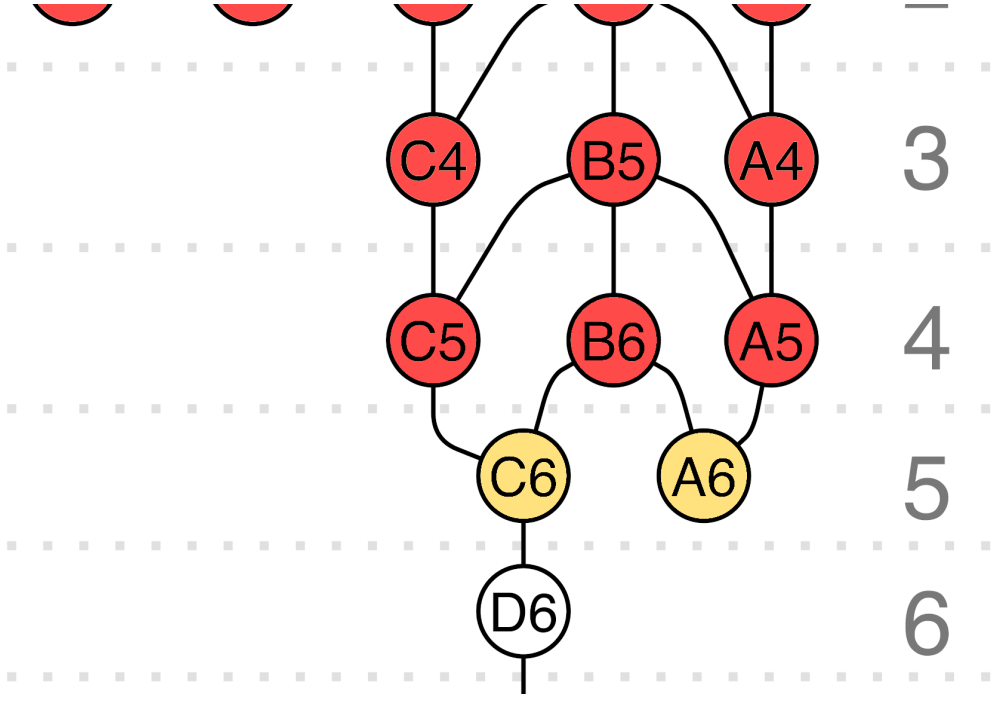


IN :[B6 C6] ← A4 B5 A6

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5]

Fonctionnement

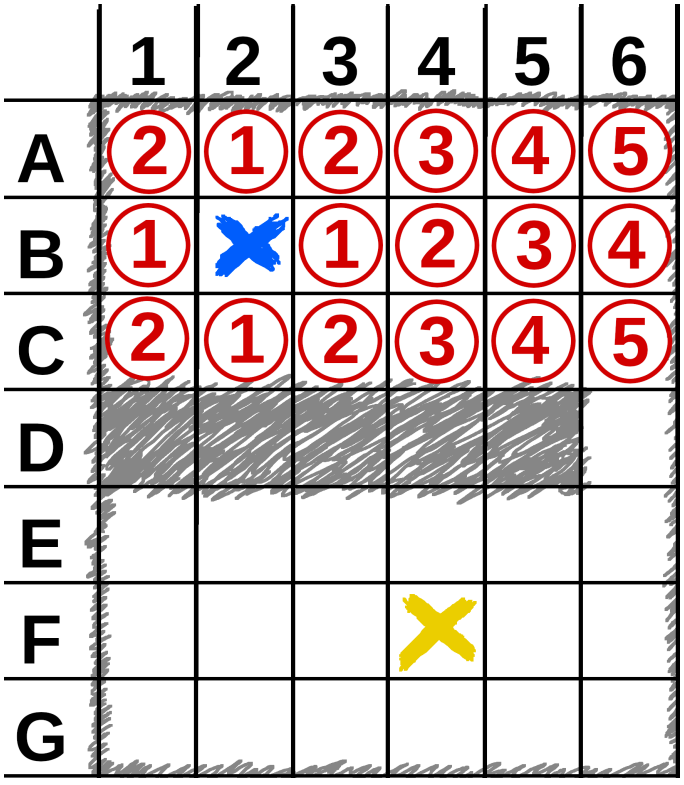


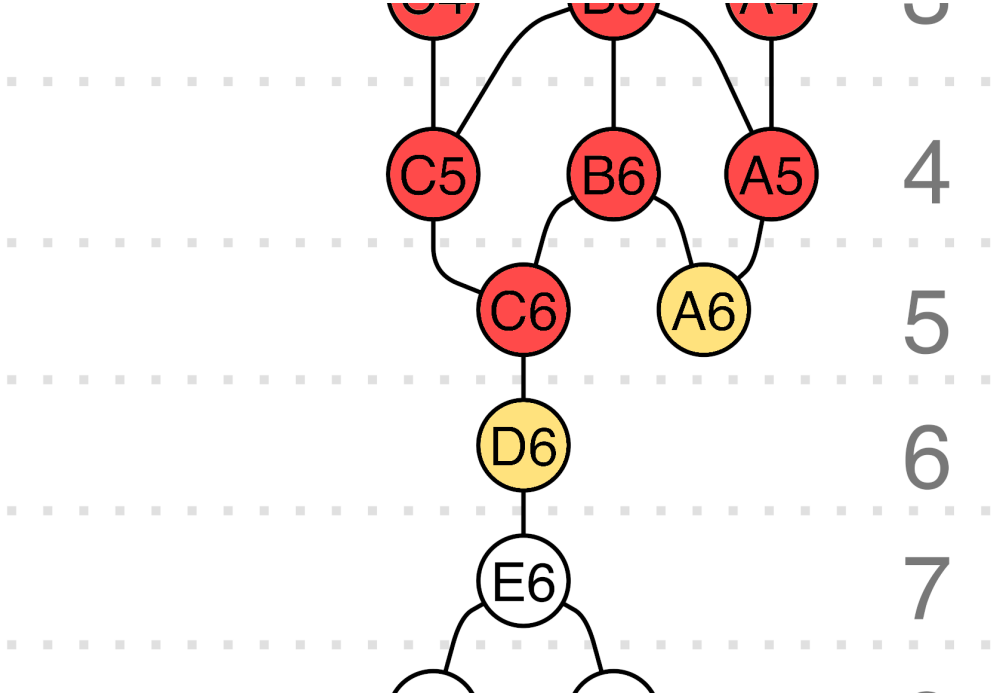


IN :[C6 A6] ← C6 A6 B5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6]

Fonctionnement



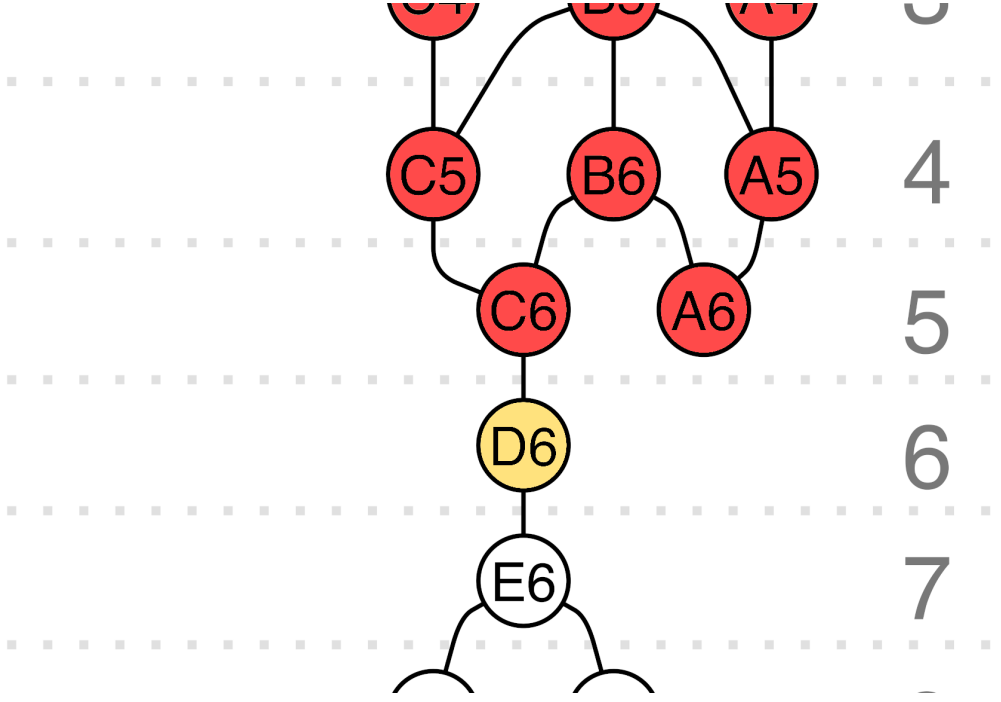


IN :[A6] ← C5 D6 B6

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6]

Fonctionnement

	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						
E						
F				X		
G						

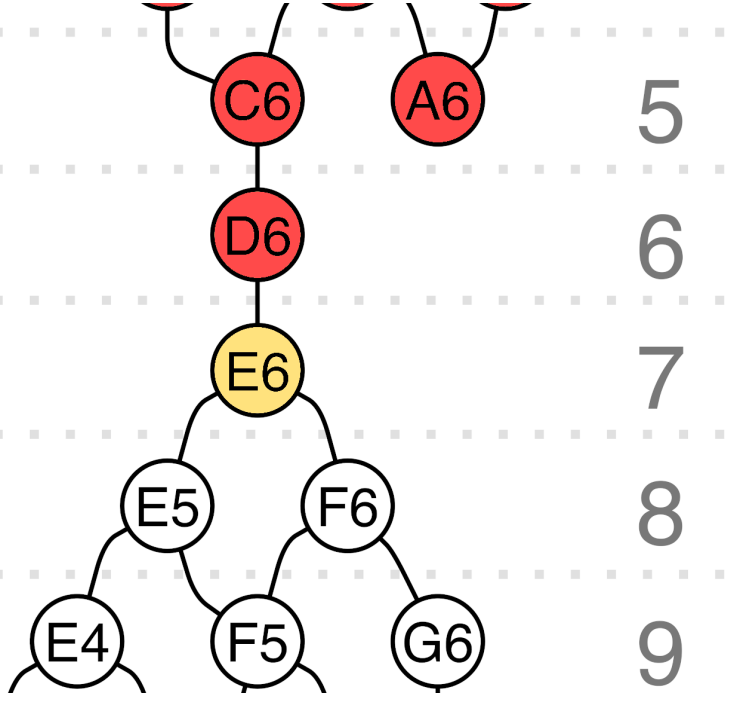


IN :[D6] ← A5 B6

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6]

Fonctionnement

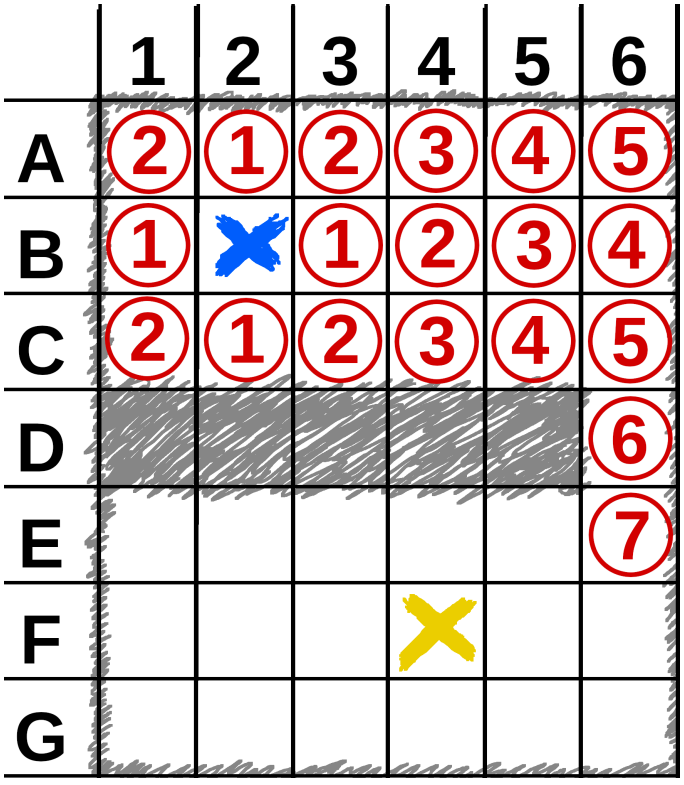
	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						6
E						
F				X		
G						

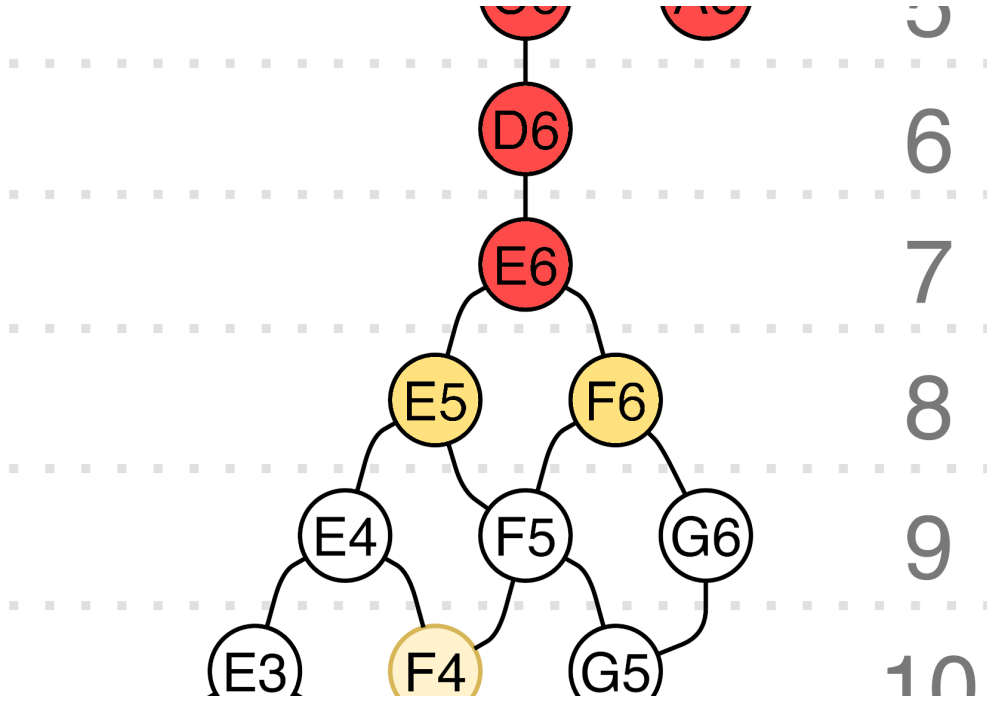


IN :[] ← E6 C6

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6]

Fonctionnement



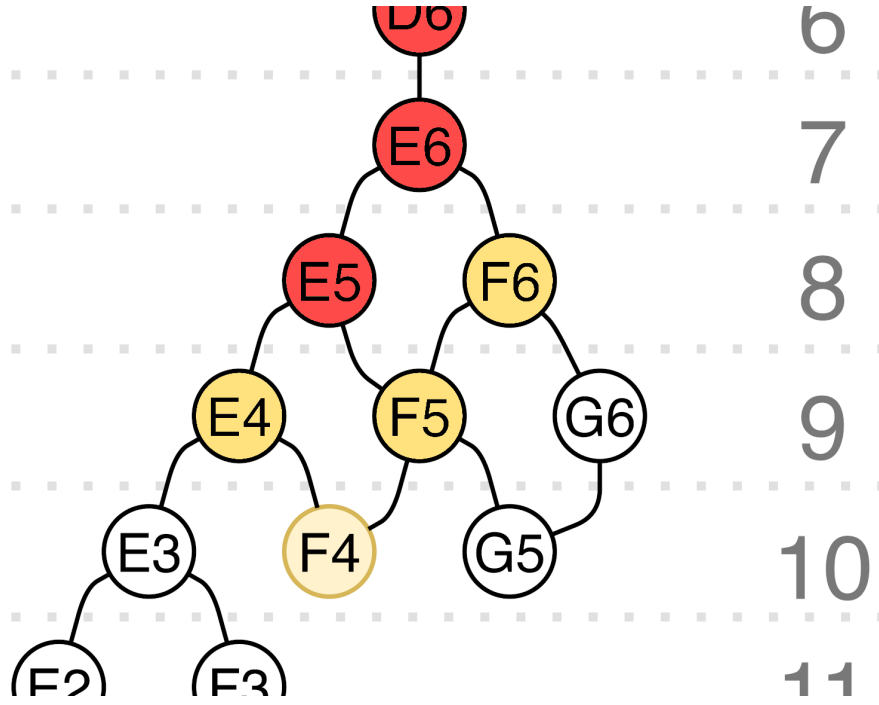


IN :[] ← E5 F6 D6

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6]

Fonctionnement

	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						6
E					8	7
F				X		8
G						

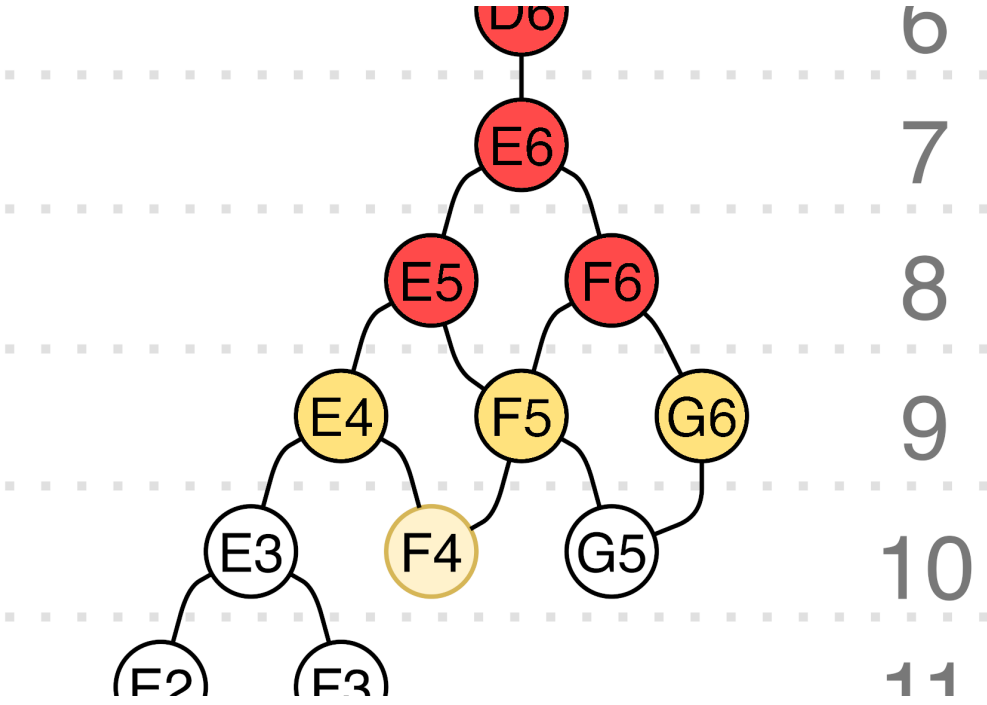


IN :[F6] ← E6 F5 E4

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6 E5]

Fonctionnement

	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						6
E					8	7
F				X		8
G						

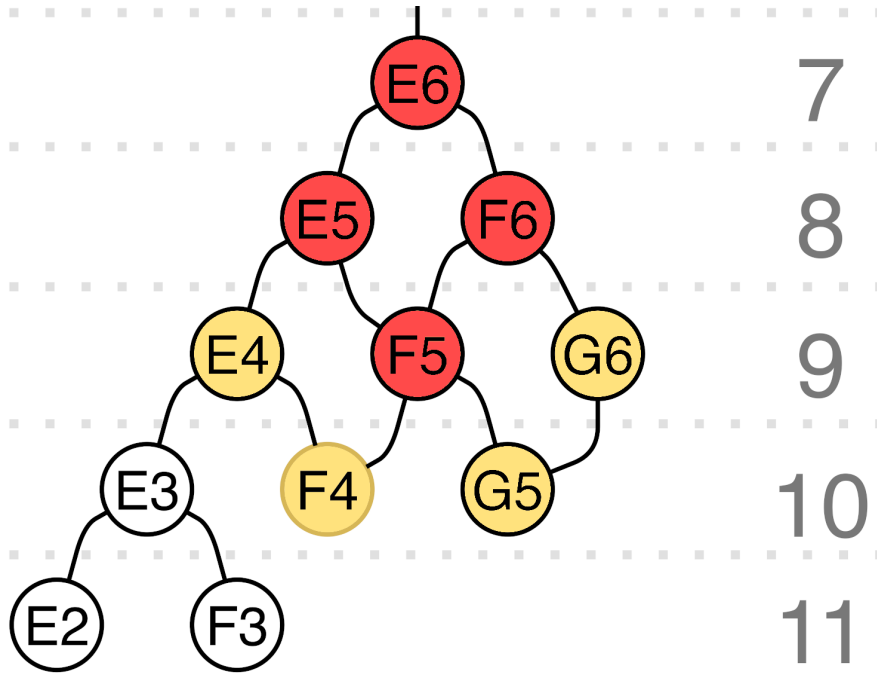


IN :[F5 E4] ← E6 F5 G6

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6 E5 F6]

Fonctionnement

	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						6
E				9	8	7
F				X	9	8
G						9

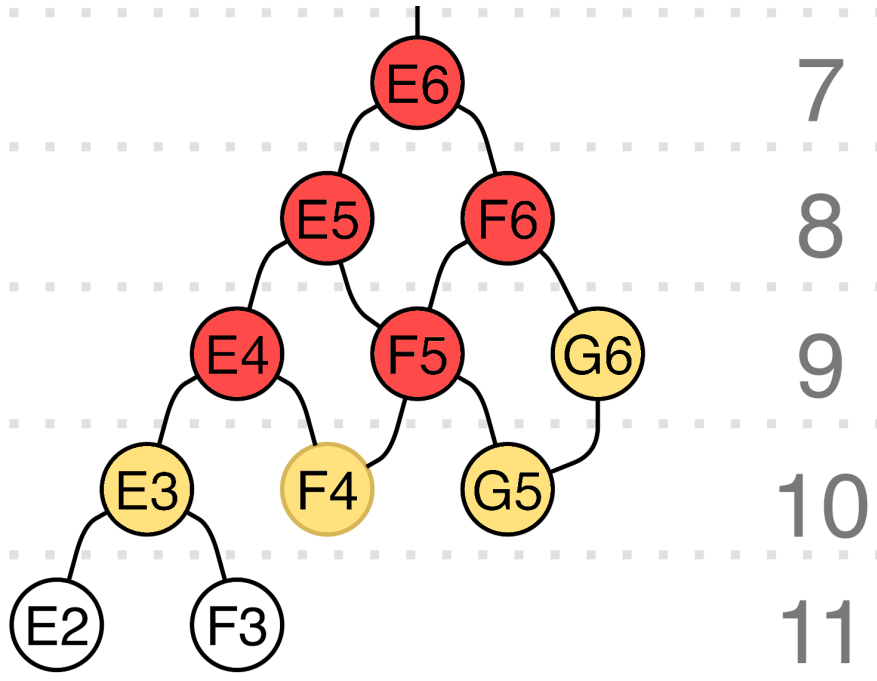


IN :[E4 G6] ← F4 E5 F6 G5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6 E5 F6 F5]

Fonctionnement

	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						6
E				9	8	7
F				X	9	8
G						9

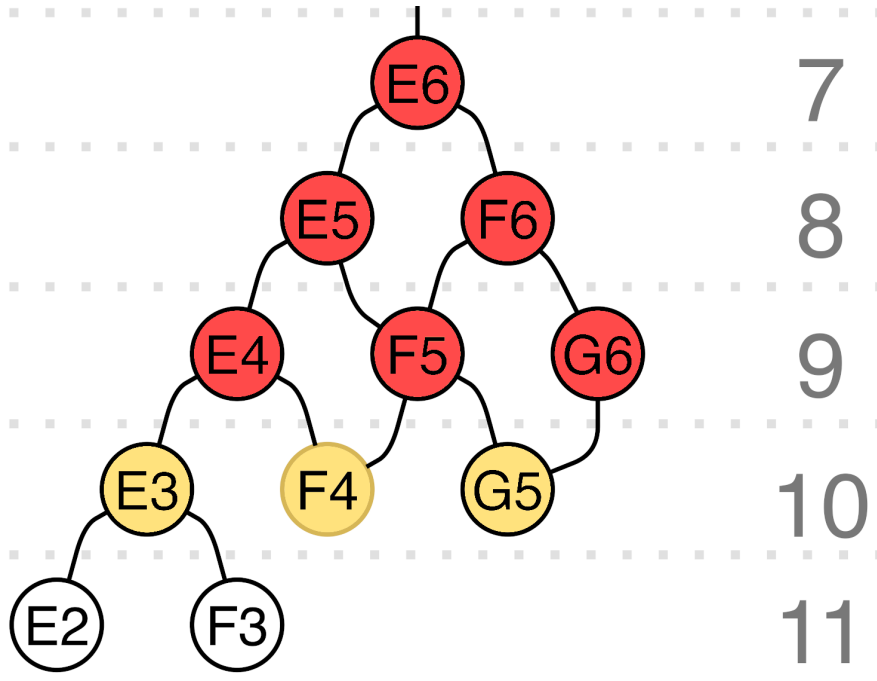


IN :[G6 F4 G5] ← E3 F4 E5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6 E5 F6 F5 E4]

Fonctionnement

	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						6
E				9	8	7
F				X	9	8
G						9

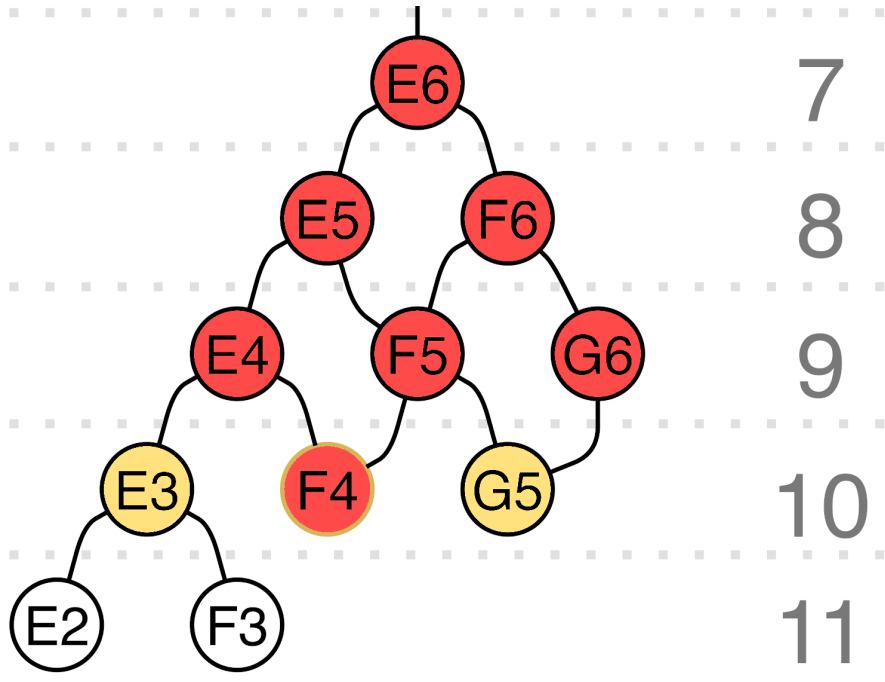


IN :[F4 G5 E3] ← F6 G5

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6 E5 F6 F5 E4 G6]

Fonctionnement

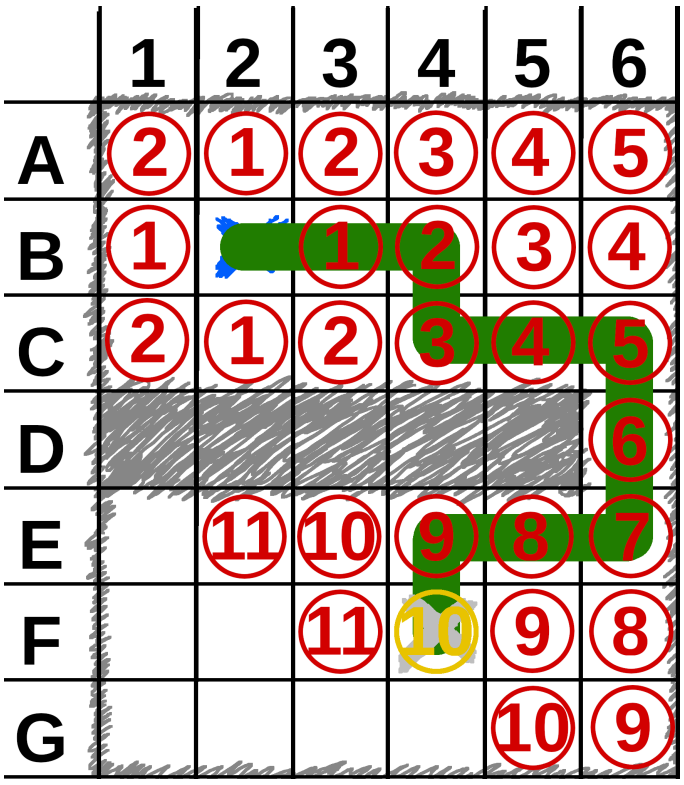
	1	2	3	4	5	6
A	2	1	2	3	4	5
B	1	X	1	2	3	4
C	2	1	2	3	4	5
D						6
E			10	9	8	7
F				10	9	8
G					10	9

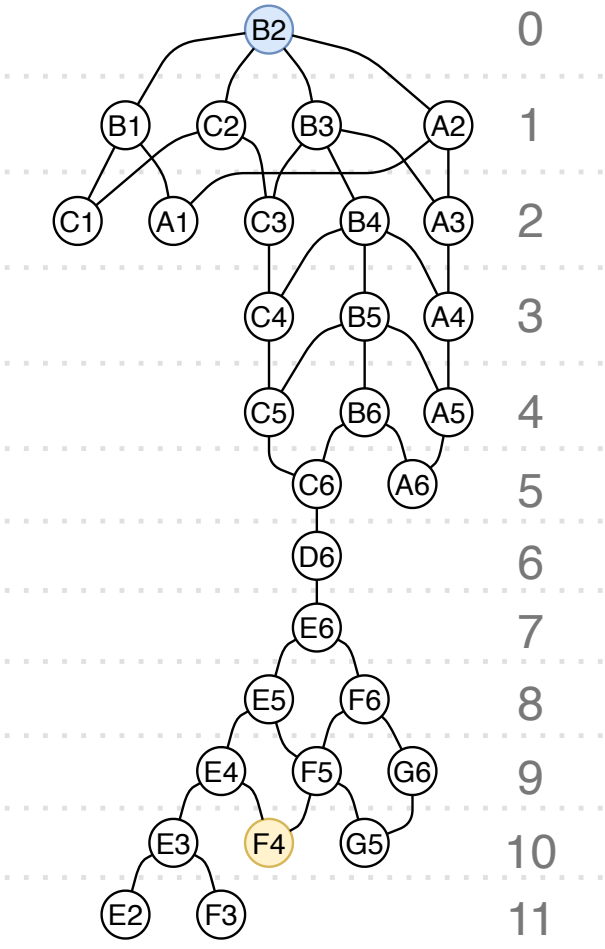


IN :[F4 G5 E3 F6]

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6 E5 F6 F5 E4 G6]

Fonctionnement





IN :[F4 G5 E3 F6]

OUT:[B2 B1 C2 A2 B3 C1 A1 C3 A3 B4 C4 A4 B5 C5 A5 B6 C6 A6 D6 E6 E5 F6 F5 E4 G6]

Le moteur de recherche générique

```
1  to search
2     current <- init-curent()
3     tant que pas trouve {
4        si current = goal alors succes ;; et recuperer le chemin.
5        in-nodes <- merge-nodes (generate (current), in-nodes )
6        si in-nodes est vide, alors echec ;; pas trouve le chemin.
7        tmp <- first in-nodes ;; et le supprime de in-nodes.
8        si tmp est dans out-nodes, continuer ;; pour ne pas boucler.
9        current <- tmp
10       out-nodes <- add(out-nodes, current) ;; pour ne pas boucler.
11    }
12 end
```

Ce qui fait la différence entre les algorithmes:

- La structure des noeuds et les fonctions d'évaluation.
- La gestion du noeud parent.
- Le tri des noeuds dans merge-node.

La structure d'un noeud

Un noeud comporte plusieurs informations:

- **p** : Le contenu, qui est le lieu même. Dans une carte routière, cela sera la ville. Dans notre cas, c'est la case (le patch).
- **h** : La valeur de ce noeud pour l'heuristique h(n) de choix de chemin à trouver à la "descente" (ou "l'aller" quand on va vers le but).
- **c** : Le coût ou valeur de ce nœud comme "meilleur" noeud pour reconstituer le chemin à la. "remontée" (ou au "retour" après avoir trouvé le but). Fonction f(n).
- **parent** : Le noeud parent.

Note: pour Best-first et Dijsktra, les valeurs h et f sont confondues. Ce n'est que pour A* que l'on fait la différence.

make-node (p, h, c, parent)

Le noeud parent

- Le noeud parent est initialement le noeud qui a généré le noeud.
- Correspond au lieu précédent par lequel on est passé pour arriver à cet endroit.
- Vrai pour Best-First.
- Mais pour Dijkstra et A* on reconstruit le noeud parent pour reprendre, au retour.

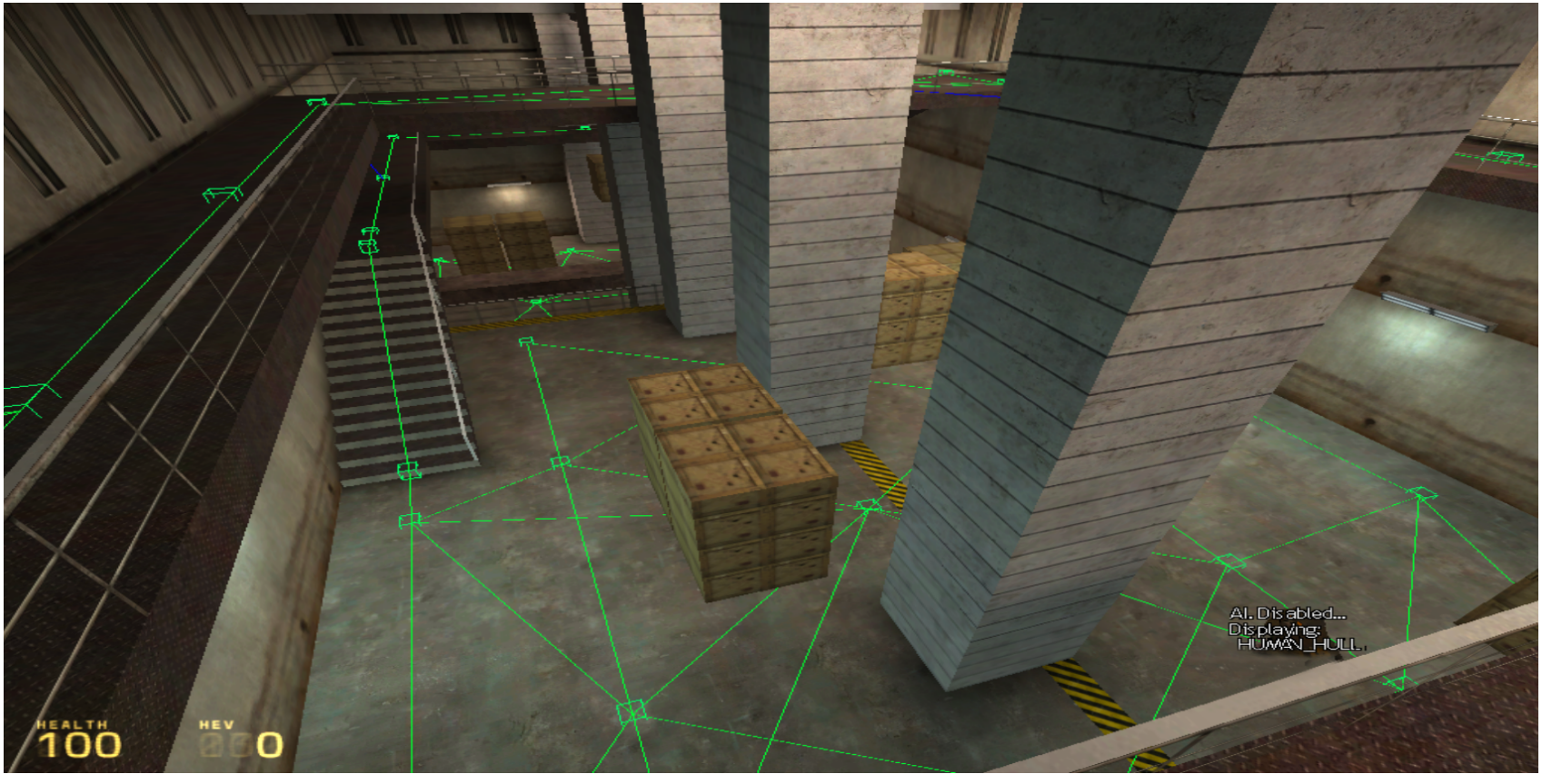
Gestion du noeud parent (merge-nodes)

- Principe: pour tout n des noeud générés par le noeud courant.
- S'il existe x un noeud de in-node de même nom (même case), tel que:
  le cout de n $<$ cout de x alors
  val(x) $<-$ val(n),
  cout (x) $<-$ val(n),
  parent(x) $<-$ parent(n)
  s'il n'existe pas de noeud de in-node avec le même nom (même case), alors on ajouter n à in-nodes...

Movement dans les jeux : waypoints

- Waypoint
  - Position sur une carte qui est utilisée pour la navigation.
  - Généralement placée par un concepteur de jeu.
- Link
  - Connection entre deux waypoints.
  - Souvent annoté avec le type de navigation (sauter, nager, grimper).
  - Pour un personnage donné, 2 waypoints sont reliés si:
    - Le personnage a suffisamment de place pour aller d'un noeud à l'autre sans entrer dans le décors.
    - Le personnage possède les capacité de navigation requises pour passer d'un point à l'autre.
- Graphe de waypoint
  - Structure de donnée comprenant tous les waypoints ainsi que les liens.
  - Généré manuellement par un concepteur au créé par programme et annoté par un concepteur.

Applications



Conclusion

- Ces trois algorithmes utilisent le même moteur d'exploration.
- Ne diffèrent que par quelques points.
- Utiliser Dijkstra quand la qualité du résultat est importante. Très utilisé pour le calcul d'itinéraire routier (économie carburant).
- Utiliser A* quand on a besoin d'avoir un bon résultat rapidement. Très utilisé en robotique ou dans les jeux videos (décision rapide).