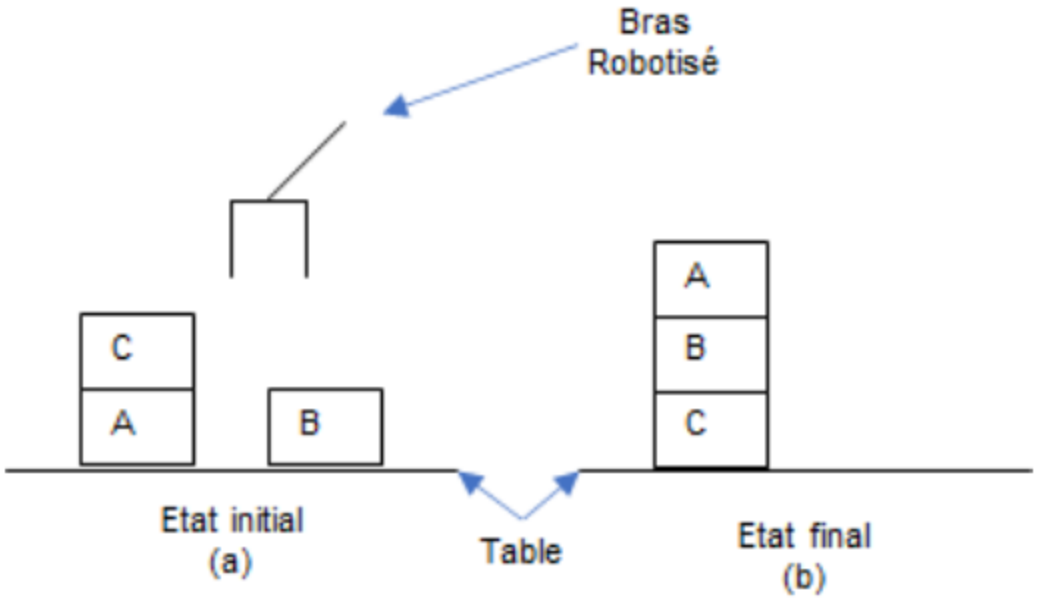
Rapport TP2

Nathan DAVID

Groupe: TP3



📕 Algorithme A*

L'algorithme de recherche A* se base sur deux fonctions :

- Une fonction d'heuristique pour évaluer la distance entre l'état actuel et l'état objectif
- Et un coût pour chaque avancement dans l'arbre

A chaque étape on évalue le noeud qui a le plus petit coût, et on ajoute tous ses fils à la pile.
Cet algorithme s'arrête si il n'y a plus de noeuds à évaluer ou si le noeud but est atteint.

Pour avoir le plus court chemin, si un état identique est rencontré à deux endroits dans l'arbre, on ne conserve que celui qui propose la plus petite évaluation.

❓ Comment représenter un état

Dans notre exercice un état est composé de l'emplacement des cubes et du status du bras.

—

Dans le code, cela peut se représenter sous la forme d'une liste de pile et d'un variable contenant une lettre ou non pour le status du bras.

Avec l'état initial on a :

```
piles = [
  ['C', 'A'],
  ['B']
]
bras = None
```

Les prédicats du systèmes sont ainsi :

Libre(X): X est libre si bras ≠ X et l'indice de X dans la pile est 0

Sur(X,Y): X est sur Y si l'indice de Y est égal à l'indice de X + 1

SurTable(X): X n'X est sur la table si son indice est égal à la taille de la pile (le dernier élément)

BrasVide: Le bras est vide si sa valeur est à None

🎲 Quelle fonction d'heuristique

Remarque : Lorsqu'une condition est vraie, elle retourne 1, sinon -1

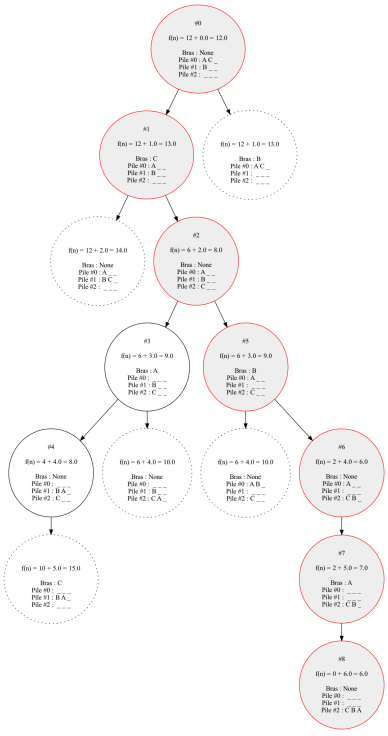
Les tests réalisés utilisent deux fonctions différentes (Avec des variantes selon l'état final et le nombre de blocs):

- **6 – (SUR (A,B) * 1 + SUR (B,C) * 2 + SURTABLE(C) * 3).**
- **3 – (SUR (A,B) + SUR (B,C) + SURTABLE(C)).**

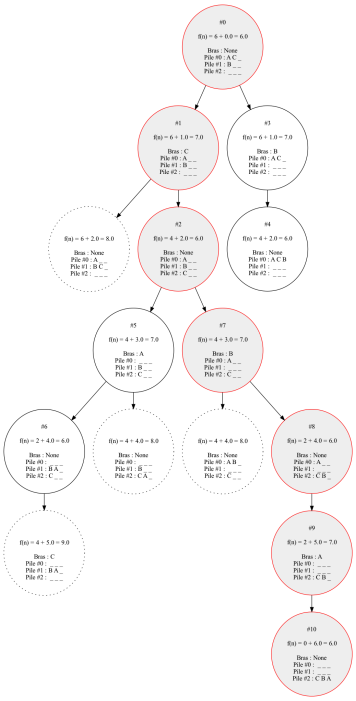
Les arbres générés avec ces deux fonctions montrent que le choix de la fonction est crutial au fonctionnement rapide de l’algorithme.
Comme cet algorithme conserve tous les noeuds visités en mémoire, il est important de choisir et tester sa fonction.

Pour le coût d’avancement dans l’arbre (fonction g(n)). De grandes différences sont aussi visibles, un coût élevé va favoriser l’exploration en largeur alors qu’un coût faible va favoriser l’exploration en profondeur.

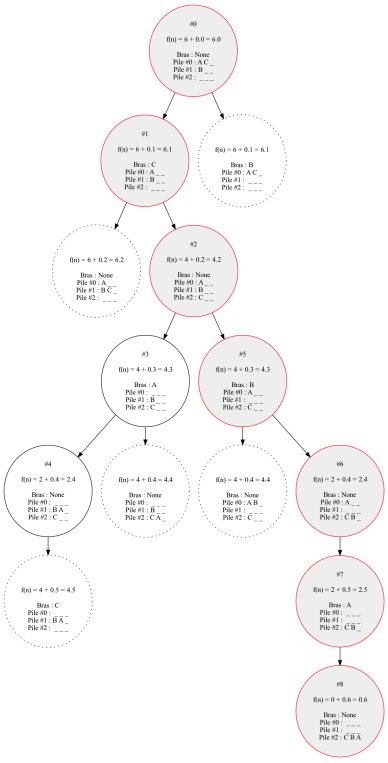
🔍 Observations



Heuristique 1 et g = 1



Heuristique 2 et g = 1



Heuristique 2 et g = 0,1

On remarque que chaque chaque changement à un en effet un impact :

- Sur la première image le résultat est trouvé après avoir analysé 9 noeuds et visités 21 en tout.
- Sur la deuxième, 11 noeuds analysés, et 21 visités
- Sur la troisième, 9 noeuds analysés et 21 visités

Mais dans tous les cas le chemin est identique.

⏳ Extensibilité du code

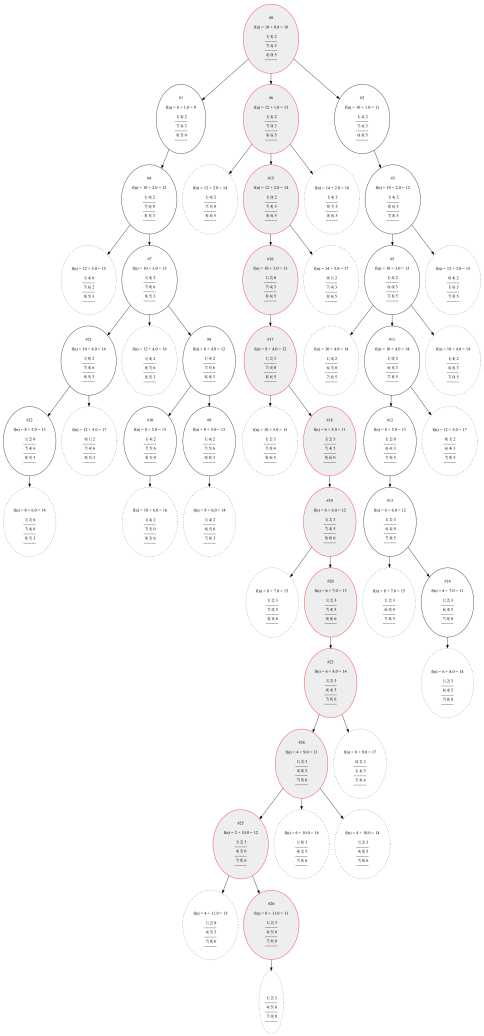
Pour utiliser l'algorithme, il ne faut créer qu'une classe représentant l'état d'un noeud, l'extends de AStarNode et d'implementer les méthodes. Ici seule la méthode `children` est nécessaire.

Une fois la classe d'état créé, il est aussi possible de customiser tous les paramètres de recherche.

En effet, la fonction de recherche prend en paramètre la fonction d'heuristique et le coût

```
def a_star_search(from_state: AStarNode,
                  to_state: AStarNode,
                  h: Callable[[any, any], float],
                  cost: float = 1) -> AStarResult or None:
```

Pour illustrer que ce code est extensible, le taquin a aussi été testé.



Ici la fonction h utilisée est une distance de manhattan, et le coût g est de 1.

Puis, il ne faut plus que changer la fonction h, l’état initial et d’arrivée.

Exemple d’utilisation avec la machine:

```
from_state = MachineState(None, [['C', 'A'], ['B']], 3)
to_state = MachineState(None, [['A', 'B', 'C']], 3)

path = a_star_search(from_state, to_state, heuristic_1, cost)
```

Exemple d’utilisation avec le taquin:

```
from_state = TaquinState([
        [12, 1, 3, 4],
        [2, 13, 14, 5],
        [11, 10, 8, 6],
        [9, 15, 7, 0]
])

to_state = TaquinState([
        [2, 12, 3, 4],
        [1, 13, 0, 5],
        [11, 14, 7, 8],
        [10, 9, 15, 6]
])

path = a_star_search(from_state, to_state, manhattan, cost)
```

L’algorithme de recherche est indépendant de toutes librairies externes.

anytree est utilisé pour la création d’images

J’ai testé avec cet environnement :

Python	3.9.10
anytree	2.8.0