Implémentation - *3.1*

parcours "Whatever First Search" Il s'agit du parcours générique d'un graphe, dans lequel le *sac* est une structure de donnée qui déterminera le mode de parcours.

- **Entrée** : un graphe $G = (S, A)$
- **Sortie** : dépendante du but du parcours

```
initialiser un sac contenant S
tant que le sac n'est pas vide :
    v = pop le premier élément du sac
    si v n'est pas marqué :
        marquer v et le traîter
        pour tout sommet w voisin de v :
            ajouter w au sac
```

Implémentation - *3.2*

parcours préfixe Ici, le sac du parcours "Whaterver First Search" est une pile

```
type graphe = int list array

let parcours_pre g s =
let n = Array.length g in (*nb sommets*)
let non_vus = Array.make n true in
let rec visite x voisins =
    if non_vus.(x) then
        (print_int x;
        non_vus.(x) <- false);
    match voisins with
    | [] -> () (*plus de voisins à traiter*)
    | v::q when non_vus.(v) ->
        visite v g.(v);
        visite x q
    | v::q -> visite x q
in visite s g.(s)
```

Implémentation - *3.3*

parcours postfixe Ici, le sac du parcours "Whaterver First Search" est une pile

```
type graphe = int list array
let parcours_post g s =
let n = Array.length g in (*nb sommets*)
let non_vus = Array.make n true in
let rec visite x voisins =
    if non_vus.(x) then
        non_vus.(x) <- false;
    match voisins with
    | [] -> print_int x (*plus de voisins à traiter*)
    | v::q when non_vus.(v) ->
        visite v g.(v);
        visite x q
    | v::q -> visite x q
in visite s g.(s)
```

Définition 3.4 - *fonction heuristique admissible*

Soit $G = (S, A, w)$ un graphe orienté pondéré. Dans le cadre de l'algorithme A* appliqué à G d'un sommet s à un sommet t, une *heuristique $h : S \to \mathbb{R}_+$ est dite admissible* lorsque :

$$\forall v \in S, h(v) \leq \mathrm{d}(v,t) = \inf\left\{\sum_{i=1}^{p-1} w(s_i, s_{i+1}),\ (s_1, \ldots, s_p) \text{ chemin de } v = s_1 \text{ à } t = s_p\right\}$$

Autrement dit losque h sous-estime la distance au sommet cible.

Définition 3.5 - *fonction heuristique monotone*

Soit $G = (S, A, w)$ un graphe orienté pondéré. Dans le cadre de l'algorithme A* appliqué à G d'un sommet s à un sommet t, une *heuristique $h : S \to \mathbb{R}_+$ est dite monotone* lorsque :

$$\begin{cases} \forall (u,v) \in A,\ h(u) \leq w(u,v) + h(v) \\ h(t) = 0 \end{cases}$$

Par récurrence, on a également h admissible.

Implémentation - *algorithme de Kruskal*

On exploite la structue Unir et Trouver pour la relation d'accessibilité, les classes sont alors les composantes connexes.

- **Entrée** : un graphe non orienté pondéré $G = (S, A, w)$
- **Sortie** : un arbre couvrant de poids minimal de G

```
initialiser MST = ∅ //Minimal Spanning Tree
pour tout s ∈ S :
    créer la classe de représentant s

trier la liste A des arêtes par ordre croissant des poids
pour tout {u,v} ∈ A :
    si la classe de u n'est pas la classe de v :
        ajouter {u,v} àMST
        unir les classes de u et v
renvoyer MST
```

Définition 3.20 - *couplage*

Soit $G = (S, A)$ un graphe non orienté. Un *couplage $C \subset A$ de G* est un ensemble d'arêtes tel que tout sommet $s \in S$ est l'extrémité d'au plus une arête de C.

Définition 3.21 - *couplage maximal*

Soit $G = (S, A)$ un graphe non orienté. Un couplage $C \subset A$ de G est dit *maximal* s'il est maximal pour l'inclusion, *i.e.* si on ne peut lui ajouter d'arête sans lui ôter sa nature de couplage :

$$\forall a \in A \setminus C,\ C \cup \{a\} \text{ n'est pas un couplage}$$

Définition 3.22 - *couplage maximum*

Soit $G = (S, A)$ un graphe non orienté. Un couplage $C \subset A$ de G est dit *maximum* s'il est de cardinal maximal parmi les couplages de G :

$$\forall C' \text{ couplage de } G,\ |C'| \leq |C|$$

Définition 3.23 - *couplage parfait*

Soit $G = (S, A)$ un graphe non orienté. Un couplage $C \subset A$ de G est dit *parfait* s'il recouvre tous les sommets de G :

$$\forall s \in S, \exists a \in C, s \in a$$