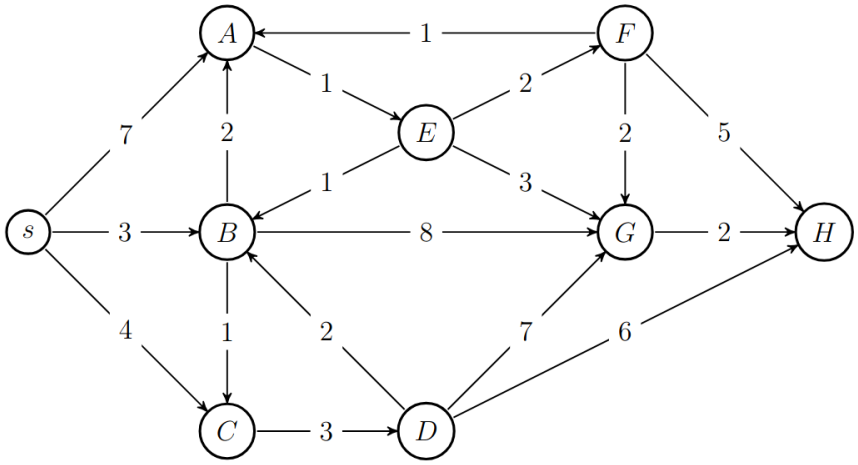
TD4. Parcours de graphe

Exercice 1. Applications

On considère le graphe orienté et pondéré suivant :



1. Dérouler à la main le parcours en profondeur. On ne tiendra pas compte des pondérations.
2. Dérouler à la main l'algorithme de Dijkstra depuis le sommet s.
3. Dérouler à la main l'algorithme A* pour trouver un plus court chemin de s à H, en utilisant l'heuristique suivante :
$s \to 10$; $A \to 6$; $B \to 8$; $C \to 7$; $D \to 6$; $E \to 4$; $F \to 3$; $G \to 2$; $H \to 0$

Exercice 2. Correction de Dijkstra

Dans cet exercice, on veut montrer que l'algorithme de Dijkstra est correct. Plus précisément, on va montrer les deux propriétés suivantes :

1. Si un sommet `u` est accessible depuis `s`, alors il est inséré dans la file.
2. Lorsqu'un sommet `u` est extrait de la file, `dist.(u)` est la longueur d'un plus court chemin de `s` à `u`.

1. Montrer le premier point.
On va montrer le second point en deux temps.
2. Montrer qu'à tout moment, si `dist.(u)` ne vaut pas `max_int`, alors `dist.(u)` est la longueur d'un chemin de `s` à `u`.
3. Grâce à un raisonnement par l'absurde, montrer qu'au moment où `u` est extrait de la file, `dist.(u)` est la longueur d'un plus court chemin de `s` à `u`.

Exercice 3. Recherche de cycle

Écire une fonction OCaml `contient_cycle : graphe -> int -> bool` qui détermine s'il existe un cycle dans un graphe **non orienté** représenté par liste d'adjacence à partir d'un sommet donné.
Pour cela, on utilisera un parcours en profondeur. Au moment de regarder les voisin du sommet `v`, si on tombe sur un sommet déjà visité qui n'est pas le prédécesseur de `v`, c'est qu'on a découvert un cycle.

Exercice 4. Graphe biparti
Un graphe est dit biparti si son ensemble de sommets peut être divisé en deux sous-ensembles disjoints S et V tels que chaque arête ait une extrémité dans U et l'autre dans V.
En utilisant un parcours en profondeur, écrire une fonction `est_biparti : graphe -> bool` qui détermine si un graphe est biparti.
Indication : Marquer les sommets visités avec une couleur 0 ou 1.

Exercice 5. Variations d'heuristique pour A*
1. À l'aide d'un exemple, montrer que si l'heuristique n'est pas admissible, l'algorithme A* peut renvoyer un résultat incorrect.
2. Quel est le comportement de l'algorithme A* lorsque l'heuristique est la fonction nulle ?
3. Quel est le comportement de A* lorsque l'heuristique est parfaite, c'est-à-dire lorsque $h(u)$ est exactement la distance qui sépare u de la destination dans le graphe ? On pourra supposer qu'il y a un unique plus court chemin de la source à la destination.

Exercice 6. Tri topologique
Soit $G = (S, A)$ un graphe orienté.
Un tri topologique sur G est une liste L de ses sommets tels que si $(i, j) \in A$, alors le sommet i apparaît avant le sommet j dans L.
1. Montrer qu'un tri topologique n'existe pas toujours.
2. Montrer qu'un graphe admet un tri topologique ssi il n'admet pas de cycle.
Indication : Pour le sens indirect, on pourra raisonner par récurrence sur le nombre de sommets du graphe.
3. Combien de prédécesseurs possède le premier sommet d'un tri topologique ?
4. En déduire un algorithme en pseudo-code qui permet de calculer un tri topologique s'il existe, et renvoie une erreur s'il n'existe pas.
5. En utilisant la représentation qui permet la complexité la plus intéressante, implémenter l'algorithme proposé.
Indication : On peut implémenter l'algorithme avec une complexité en $O(|S| + |A|)$.