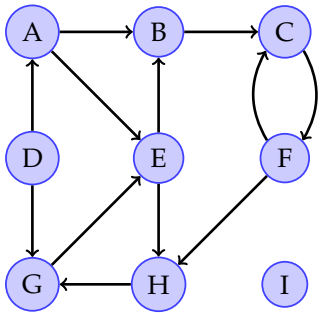
TD4 : Graphes

1 Kosaraju

On considère le graphe suivant :



Dérouler l'algorithme de KOSARAJU-SHARIR sur le graphe ci-dessus, pour en trouver ses composantes fortement connexes. Dessiner et annoter le graphe miroir ainsi que le graphe initial avec les temps de visite des parcours en profondeur utilisés. Lorsque plusieurs choix seront possibles on utilisera toujours l'ordre alphabétique. Représenter sous la forme d'une pile l'ordre dans lesquels les sommets vont être considérés pour le second parcours.

2 Composantes connexes par Unir et Trouver

On considère un graphe $G = (S, A)$ avec $S = \{0, 1, \ldots, n-1\}$. On représente un graphe non orienté par la liste de ses arêtes (chaque arête est représentée exactement une fois).

```
type graph = {nb_sommets : int; aretes : (int * int) list};;
```

On considère le type suivant pour représenter une partition avec l'algorithme unir et trouver :

```
type partition = {pere : int array; rang : int array};;
```

où `rang.(i)` représente la hauteur de l'arbre enraciné en `i` et cette valeur n'est effectivement garantie à jour que lorsque `i` est bien son propre père.

1. Écrire la fonction `creer_singletons : int -> partition` qui initialise une partition de $\{0, 1, \ldots, n-1\}$ avec n singletons.
2. Écrire la fonction `trouver : partition -> int -> int` qui permet de trouver le représentant d'un élément, en appliquant l'heurisitique de compression de chemin.
   On ne demande pas d'écrire la fonction `unir : partition -> int -> int -> unit` qui réalise l'union par rang. On admet qu'en utilisant ces deux heuristiques, la complexité d'une séquence de n opérations `creer_singletons`, trouver et unir est en $O(n\alpha(n))$ où α est une fonction à croissance extrêmement lente ($\alpha(n) \leq 4$ pour tous les cas pratiques ; α est une sorte de réciproque de la fonction d'Ackermann qui elle croît inimaginablement vite).
3. Écrire une fonction `composantes_connexes : graph -> partition` qui calcule la partition représentant les composantes connexes. Quelle est la complexité de cette approche ?
4. Quelle autre approche pourrait-on utiliser pour calculer les composantes connexes et quelle en serait la complexité ? Une ou deux phrases.
5. On suppose que l'on ajoute une arête au graphe (en déclarant le champ mutable ou en renvoyant un nouveau graphe avec une arête de plus). Que faurait-il faire pour mettre à jour les composantes connexes du graphe pour chacune des deux approches ? Discuter des avantages et inconvénients de ces deux approches. Deux ou trois phrases.

3 Diamètre d'un graphe

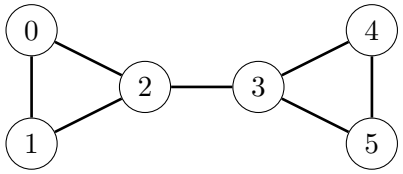
Dans cet exercice, on considère des graphes non orientés connexes. Les sommets d'un graphe à n sommets ($n \in \mathbb{N}^*$) sont numérotés de 0 à $n-1$. On suppose qu'aucune arête ne boucle sur un même sommet.
Un chemin de longueur $p \in \mathbb{N}$ d'un sommet a vers un sommet b dans un graphe est la donnée de $p+1$ sommets s_0 , s_1,..., s_p tels que $s_0 = a$, $s_p = b$ et, pour tout $1 \leqslant k \leqslant p$, les sommets s_{k-1} et s_k sont reliés par une arête.
Un plus court chemin d'un sommet a vers un sommet b dans un graphe G est un chemin de longueur minimale parmi tous les chemins de a vers b. Sa longueur est notée $d_G(a, b)$.
Le diamètre d'un graphe G, noté $diam(G)$, vaut le maximum des longueurs des plus courts chemins entre deux sommets du graphe G. Autrement dit,

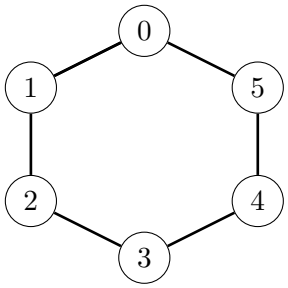
$$diam(G) = \max_{a,b \text{ sommets de } G} d_G(a, b).$$

Un chemin maximal d'un graphe G est un plus court chemin de G de longueur $diam(G)$.

3.1 Exemples de graphes

1. Donner sans justification le diamètre et les chemins maximaux pour chacun des deux graphes G_1 et G_2 ci-dessous.





En OCaml, les graphes sont représentés par liste d'adjacence et implémentés par le type
`type graphe = int list array;;`

2. Graphes de diamètre maximal.
   (a) Dessiner sans justification un graphe à 5 sommets ayant un diamètre le plus grand possible.
   (b) Écrire en OCaml une fonction `diam_max` de type `int -> graphe` qui prend en argument un entier naturel n non nul et qui renvoie un graphe à n sommets de diamètre maximal.
3. Graphes de diamètre minimal.
   (a) Dessiner sans justification un graphe à 5 sommets ayant un diamètre le plus petit possible.
   (b) Écrire en OCaml une fonction `diam_min` de type `int -> graphe` qui prend en argument un entier naturel n non nul et qui renvoie un graphe à n sommets de diamètre minimal.

3.2 Algorithme de calcul du diamètre

Dans cette partie, on suppose que les graphes sont représentés par listes d'adjacence.

1. Donner l'entrée et la sortie de l'algorithme de Dijkstra. Comment cet algorithme permet-il de calculer le diamètre d'un graphe ?
2. Quel parcours de graphe peut être utilisé pour le calcul du diamètre ?
3. Laquelle des deux méthodes précédentes est la mieux adaptée pour calculer le diamètre d'un graphe ?