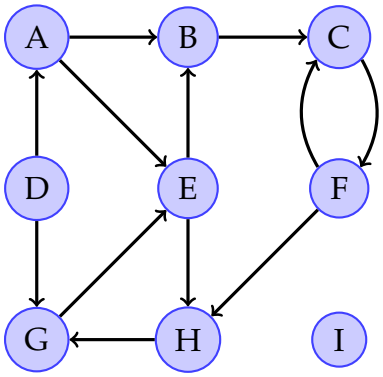
Devoir surveillé n° 03 — 30 min

EXERCICE 1 *Trace de l'algorithme de Kosaraju-Sharir*

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. Pour ce second parcours, donc du graphe initial, indiquer le nombre d'arc de parcours, d'arcs arrières, d'arcs avants et d'arcs transverses.

EXERCICE 2

On représente un graphe $G = (S, A)$ avec $S = \{0, 1, \ldots, n-1\}$ par un type `graph` et on suppose que les deux fonctions suivantes sont données :

OCAML
```
(* size g renvoie le nombre de sommets de g *)
val size : graph -> int
(* neighbors g s renvoie la liste des voisins de s *)
val neighbors : graph -> int -> int list
```

Écrire une fonction `dfs : graph -> int * int array` telle que l'appel `dfs g` pour un graphe orienté $G = (S, A)$ représenté par `g` effectue un parcours en profondeur de G et renvoie un couple formé :

- du nombre d'arcs arrière rencontrés lors de ce parcours ;
- de la forêt de parcours représentée par un tableau de lien de parenté : à la case d'indice s se trouve l'unique parent de s dans la forêt de parcours. On convient qu'un sommet qui est racine de son arborescence est son propre parent.

EXERCICE 3 *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).

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

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

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

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 à 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.*

EXERCICE 4 *Cliques*

Soit $G = (S, A)$ un graphe non orienté. Une *clique* de G est un ensemble de sommets connectés deux à deux dans G. Une *anti-clique* est un ensemble de sommets deux à deux non connectés dans G.

1. Soit S_1 une clique et S_2 une anti-clique de G. Montrer que $|S_1 \cap S_2| \leq 1$.
2. Montrer que si l'ensemble des sommets de G se partitionne en une clique et une anti-clique, alors il en est de même pour $G^c = (S, A^c)$, le graphe complémentaire de G, dans lequel $A^c = \mathcal{P}_2(S) \setminus A$ où $\mathcal{P}_2(S)$ est l'ensemble des paires d'éléments de S.
3. On suppose que $|S| = 6$. Montrer que G admet une clique de taille 3 ou une anti-clique de taille 3.