On désignera par $V(G)$ et $E(G)$ respectivement l'ensemble des sommets et l'ensemble des arêtes d'un graphe G. Les variables n et m désigneront respectivement le nombre de sommets et d'arêtes.

Recherche d'un chemin hamiltonien dans un tournoi

Un *tournoi* est un graphe simple complet orienté. Autrement dit, si T est un tournoi et u et v deux sommets de T, alors $(u, v) \in E(T) \Leftrightarrow (v, u) \notin E(T)$.

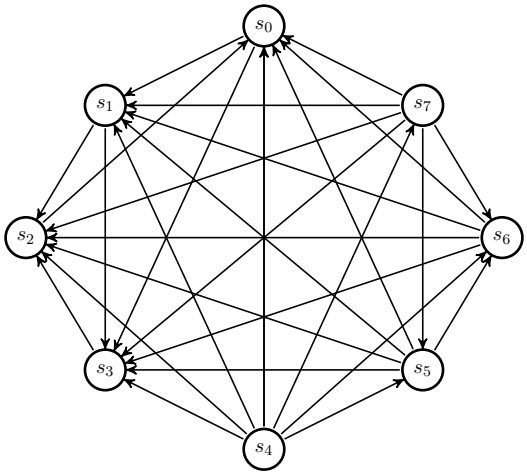

FIGURE 1 – Exemple de tournoi à 8 sommets

Un *chemin hamiltonien* est un chemin qui passe par tous les sommets du graphe une et une seule fois. Par exemple, pour le tournoi de la Figure 1 : $s_4, s_7, s_5, s_6, s_0, s_1, s_3, s_2$ est un chemin hamiltonien.

0.1) Montrer que tout tournoi possède un chemin hamiltonien (*indication :* utiliser un raisonnement par l'absurde).

Soit T un tournoi et $P = x_1, ..., x_k$ un chemin de T de longueur maximum. Supposons qu'il existe un sommet u de T qui n'appartient pas à P.
a) L'arc entre u et x_1 est forcément x_1u, sinon on pourrait augmenter la longueur de P en rajoutant u devant x_1 (contradiction).
b) Si l'arc entre un sommet x_i, $1 \leq i \leq k-1$ et u est x_iu, alors l'arc entre x_{i+1} et u est forcément $x_{i+1}u$, sinon on pourrait insérer u dans P entre x_i et x_{i+1}
a) et b) $\Rightarrow$ l'arc entre x_k et u est forcément x_ku. Contradiction car on peut rajouter u derrière x_k pour rallonger P.
Conclusion : tout tournoi possède un chemin contenant tous les sommets, donc hamiltonien.

0.2) Appliquer l'algorithme de parcours en profondeur rappelé à la fin du document (Algorithmes 3 et 4) au tournoi T de la Figure 1. On conviendra que dans les listes d'adjacence, les

sommets sont rangés dans l'ordre lexicographique. Pour chaque sommet, donner les valeurs `d`, `f`, `pere` retournées par l'algorithme en respectant l'ordre de visite des sommets.

L'arbre obtenu en partant de s_0 et en considérant les voisins dans l'ordre lexicographique est donné par la Figure 2.

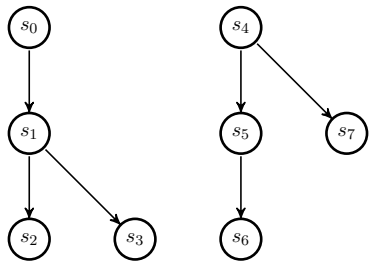

FIGURE 2 – Parcours en profondeur du tournoi de la Figure 1

Les valeurs des tableaux d, f, et père sont données ci-dessous :

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
$pere$	NIL	s_0	s_1	s_1	NIL	s_4	s_5	s_4
d	1	2	3	5	9	10	11	14
f	8	7	4	6	16	13	12	15

Ordonner les sommets de T dans l'ordre inverse de leur fin de visite. Que remarquez-vous ?

L'ordre des sommets dans l'ordre inverse de leur fin de visite est :
$s_4, s_7, s_5, s_6, s_0, s_1, s_3, s_2$, ce qui correspond au chemin hamiltonien donné en exemple.

0.3) Lorsque l'on range les sommets dans l'ordre inverse de leur fin de visite, quels sont les types d'arcs (parmi *liaison, retour, avant, transverse*) reliant deux sommets consécutifs ? Montrer que si les sommets sont rangés dans l'ordre inverse de leur fin de visite, alors si le sommet v suit immédiatement le sommet u dans cet ordre, l'arc entre ces deux sommets est orienté de u vers v.

Si u et v sont deux sommets consécutifs dans l'ordre inverse de l'heure de fin de visite, s'il existe un arc entre u et v, ce qui sera toujours le cas dans un tournoi, cet arc sera soit un arc de liaison, soit un arc transverse.
En effet, si uv est un arc retour, alors $f[u] < f[v]$. Si uv est un arc avant, il y aura toujours des sommets intermédiaires dans l'arbre de parcours entre leur origine et leur destination. Donc leurs heures de fin de visite ne seront pas consécutives.
De plus, les arcs de liaisons et transverses sont toujours orientés du sommet ayant la plus grande heure de fin de visite vers celui ayant la plus petite. Donc si v suit u dans l'ordre inverse de fin de visite, l'arc entre u et v sera l'arc uv.

0.4) Modifier l'algorithme de parcours en profondeur pour afficher les sommets dans un ordre induisant un chemin hamiltonien. Quelle est la complexité de l'algorithme obtenu ?

Dans la partie $PP(G)$, on crée une pile vide P, par exemple après l'initialisation de la

variable *temps*, puis on affiche cette pile après avoir terminé le parcours.
Dans $VisiterPP(v)$, on empile v dans P quand on met à jour son heure de fin de visite
Cela donne les algorithmes suivants PP' et $VisiterPP'$:

Algorithme 1 PP'(G)

```
1: pour tout v ∈ V(G) faire
2:    couleur(v) ← BLANC
3:    pere(v) ← NIL
4: fin pour
5: temps ← 0
6: P ← Pile_Vide()
7: pour tout v ∈ V(G) faire
8:    si couleur(v) = BLANC alors
9:       VisiterPP(v)
10:   fin si
11: fin pour
12: Afficher(P)
```

Algorithme 2 VisiterPP'(v)

```
1: d(v) ← temps ← temps + 1
2: couleur(v) ← GRIS
3: pour tout w ∈ Adj(v) faire
4:    si couleur(w) = BLANC alors
5:       pere(w) ← v
6:       VisiterPP(w)
7:    fin si
8: fin pour
9: couleur(v) ← NOIR
10: f(v) ← temps ← temps + 1
11: Empiler(P, v)
```

La complexité du parcours en profondeur n'est pas modifiée et l'algorithme obtenu est donc en $O(n + m)$.

Algorithmes

Algorithme 3 Parcours en profondeur PP(G)

```
1: pour tout v ∈ V(G) faire
2:    couleur(v) ← BLANC
3:    pere(v) ← NIL
4: fin pour
5: temps ← 0
6: pour tout v ∈ V(G) faire
7:    si couleur(v) = BLANC alors
8:       VisiterPP(v)
9:    fin si
10: fin pour
```

Algorithme 4 VisiterPP(v)

```
1: d(v) ← temps ← temps + 1
2: couleur(v) ← GRIS
3: pour tout w ∈ Adj(v) faire
4:    si couleur(w) = BLANC alors
5:       pere(w) ← v
6:       VisiterPP(w)
7:    fin si
8: fin pour
9: couleur(v) ← NOIR
10: f(v) ← temps ← temps + 1
```