TP 37

Chemins euleriens

Définition. Soit $G = (V, E)$ un graphe non orienté.

- Un *chemin eulérien* de G est un chemin simple (c'est-à-dire constitué d'arêtes distinctes) reliant deux sommets de G et utilisant toutes les arêtes. Autrement dit, c'est un chemin simple de longueur $|E|$.
- Un *circuit eulérien* est un chemin eulérien dont les deux extrémités sont les mêmes. Autrement dit, c'est un cycle de longueur $|E|$.
- Un *graphe eulérien* est un graphe possédant un circuit eulérien.

1. Montrer que le graphe suivant possède un chemin eulérien.

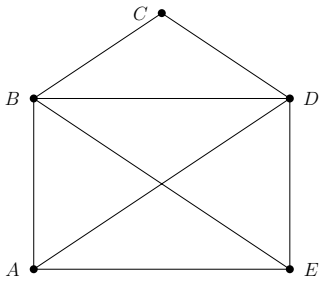

FIGURE 37.1 – Le graphe G_1.

2. Montrer que le graphe suivant possède un circuit eulérien.

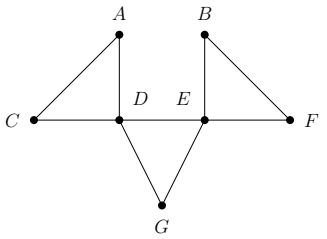

FIGURE 37.2 – Le graphe G_2.

1 Condition nécessaire

3. Soit G un graphe eulérien. Montrer que tous les sommets de G sont de degré pair.
4. Énoncer une condition nécessaire similaire à celle de la question précédente pour qu'un graphe G possède un chemin eulérien.

II Condition suffisante

5. Soit G un graphe dont tous les sommets sont de degré pair. On considère le processus suivant :
   - on part d'un sommet x quelconque ;
   - on suit des arêtes à partir de x, sans les réutiliser, jusqu'à se retrouver bloqué. Ici, *bloqué* signifie que l'on est sur un sommet depuis lequel il n'y a plus d'arête disponible.

   Montrer que le processus se termine nécessairement au sommet x.
6. On suppose avoir construit un cycle en suivant le processus de la question précédent. Montrer que, si le graphe est connexe, on est forcément dans l'un des deux cas suivants :
   - soit le cycle construit est un circuit eulérien ;
   - soit il existe un sommet du cycle qui dispose encore d'une arête disponible.
7. En déduire qu'un graphe connexe est eulérien si, et seulement si, tous ses sommets sont de degré pair.
8. Énoncer et démontrer une propriété similaire pour l'existence d'un chemin eulérien.
9. Le problème historique qui a donné naissance à la notion de graphe eulérien est celui des *ponts de Köningsberg*. La ville de Köningsberg possédait à l'époque d'Euler sept ponts, disposés comme suit :

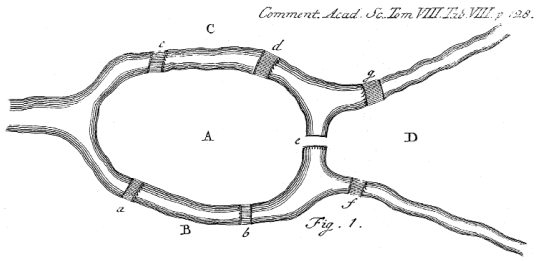

FIGURE 37.3 – Les sept ponts de Köningsberg.

   Le problème est de savoir s'il est possible de traverser successivement tous ces ponts sans repasser deux fois par le même. Proposer une extension des résultats précédents aux *multigraphes* (graphes dans lesquels on peut avoir plusieurs arêtes entre deux sommets donnés) permettant de répondre à cette question.

III Construction de chemins eulériens

On s'intéresse dans cette partie à la construction effective d'un circuit eulérien dans un graphe. L'algorithme utilisé, dit *algorithme de Hierholzer*, suit essentiellement la démonstration faite plus haut pour la condition suffisante.
On maintient deux piles, une nommée *actuel* correspondant au chemin en cours d'exploration, et l'autre *euler* correspondant au circuit eulérien que l'on construit.

- On commence à un sommet x_0 quelconque.
- On suit des arêtes pour former un chemin, en supprimant au fur et à mesure les arêtes du graphe et en empilant les sommets visités sur *actuel*.
- Si à un moment on arrive sur un sommet ne disposant plus d'arête disponible, on dépile des sommets de *actuel* jusqu'à retomber sur un sommet qui dispose encore d'une arête. Ces sommets sont empilés sur *euler* au fur et à mesure.
- L'algorithme se termine quand *actuel* est vide : *euler* contient alors un circuit eulérien si le graphe de départ était eulérien.

10. Simuler à la main l'exécution de l'algorithme sur le multigraphe suivant, en supposant que, quand il y a plusieurs arêtes x, y disponibles depuis un certain sommet x, on traite d'abord celle pour laquelle y est minimal. On le fera une fois en commençant par le sommet 0 et une fois en commençant par le sommet 4.

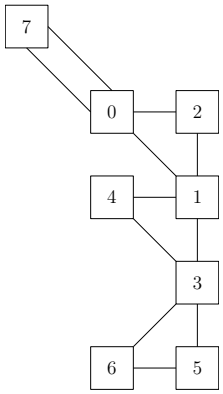

FIGURE 37.4 – Le graphe G_3.

Le fichier `stack.h` contient l'interface d'une structure de pile mutable, et le fichier `graph.h` l'interface d'une structure de graphe. On donne les garanties suivantes sur la complexité des fonctions [1] :

- toutes les opérations élémentaires sur les piles sont en temps constant, l'initialisation d'une pile en temps proportionnel à sa capacité ;
- `build_graph`, avec `nb_vertex` $= n$ et `nb_edges` $= p$, est en $\mathcal{O}(n + p)$;
- `get_edge`, `has_available_edge` et `delete_edge` sont en temps constant.

11. Écrire une fonction lisant des données sur l'entrée standard au format suivant :
    - la première ligne contient deux entiers n et p séparés par une espace : n sera le nombre de sommets, p le nombre d'arêtes ;
    - les p lignes suivantes contiennent chacune deux entiers de $[0 \dots n-1]$ séparées par une espace : les deux sommets incidents à une arête.

    La fonction renverra un tableau `edges` de longueur $2p$ tel que `edges[2 * i]` et `edges[2 * i + 1]` soient les deux sommets constituant l'arête i. Elle modifiera également les valeurs pointées par les arguments de sortie `nb_vertex` et `nb_edges`.

```
int *read_data(int *nb_vertex, int *nb_edges);
```

12. Écrire une fonction `euler_tour` qui renvoie un circuit eulérien du graphe `g` (en supposant qu'un tel circuit existe), sous forme d'un pointeur vers une pile de sommets.

```
stack *euler_tour(graph g);
```

13. Créer un fichier correspondant au graphe G_3 et vérifier que l'on obtient bien ce qui était prévu.
14. Déterminer la complexité totale de l'algorithme (lecture des données, construction du graphe, construction du circuit eulérien).
15. Si le graphe possède un chemin eulérien mais pas de circuit eulérien, est-il garanti que cet algorithme le trouve ? Si ce n'est pas le cas, indiquer la modification qu'il faudrait apporter pour traiter correctement ce cas.

1. Pour certaines fonctions, la réalité est un peu plus complexe mais cela n'affecte pas le résultat final.