TD XXXVII

GRAPHES EULÉRIENS

Définition XXXVII.1 – Chemin eulérien, circuit eulérien

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.

► **Question 1** Montrer que le graphe suivant possède un chemin eulérien.

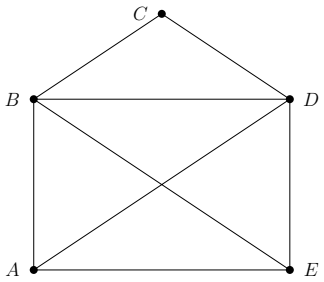


FIGURE XXXVII.1 – Le graphe G_1.

► **Question 2** Montrer que le graphe suivant possède un circuit eulérien.

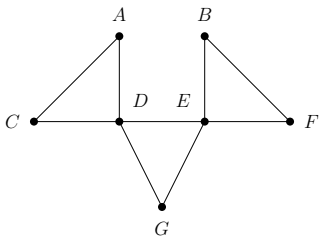


FIGURE XXXVII.2 – Le graphe G_2.

1 Condition nécessaire

► **Question 3** Soit G un graphe eulérien. Montrer que tous les sommets de G sont de degré pair.

► **Question 4** Énoncer une condition nécessaire similaire à celle de la question précédente pour qu'un graphe G possède un chemin eulérien.

2 Condition suffisante

► **Question 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 le processus se termine nécessairement au sommet x.

► **Question 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.

► **Question 7** En déduire qu'un graphe connexe est eulérien si et seulement si tous ses sommets sont de degré pair.

► **Question 8** Énoncer et démontrer une propriété similaire pour l'existence d'un chemin eulérien.

► **Question 9** Le problème historique qui a donné naissance à la notion de graphe eulérien est celui des *ponts de Königsberg*. La ville de Königsberg [1] possédait à l'époque d'Euler sept ponts, disposés comme suit :

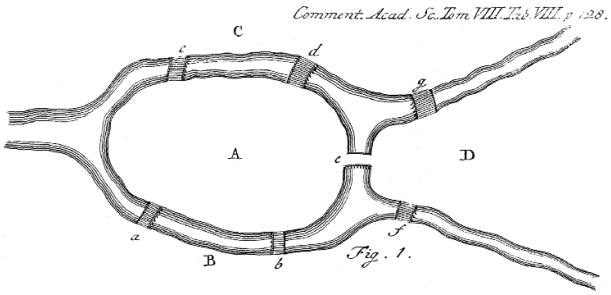


Figure XXXVII.3 – Les sept ponts de Königsberg.

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.

3 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.

1. Aujourd'hui Kaliningrad, en Russie.

▶ **Question 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.

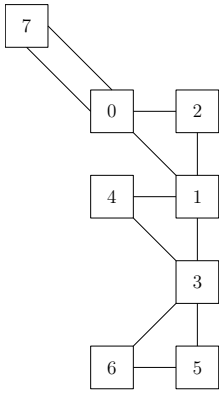


Figure XXXVII.4 – Le graphe G_3.

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 [2] :

- 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 $O(n + p)$;
- `get_edge`, `has_available_edge` et `delete_edge` sont en temps constant.

▶ **Question 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);
```

▶ **Question 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);
```

▶ **Question 13** Créer un fichier correspondant au graphe G_3 et vérifier que l'on obtient bien ce qui était prévu.

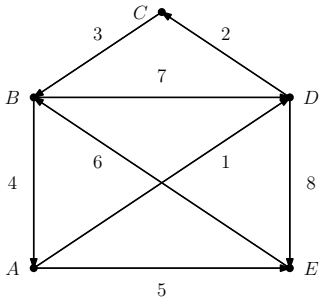
▶ **Question 14** Déterminer la complexité totale de l'algorithme (lecture des données, construction du graphe, construction du circuit eulérien).

▶ **Question 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.

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

Solutions

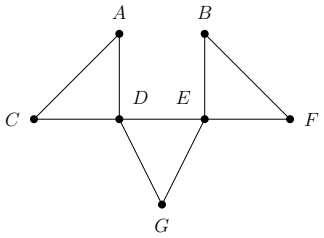
► **Question 1**



A-D-C-B-A-E-B-D-E

FIGURE XXXVII.5 – Un chemin eulérien dans G_1.

► **Question 2**



E-F-B-E-D-A-C-D-G-E

FIGURE XXXVII.6 – Un circuit eulérien dans G_2.

► **Question 3** Considérons un cycle eulérien $x_0, x_1, \ldots, x_p = x_0$ (avec p le nombre d'arêtes du graphe). Pour chaque occurrence d'un sommet x dans ce cycle, on a deux arêtes incidentes à x : une pour y entrer et une pour en sortir. De plus, le graphe n'ayant pas de boucle, on ne compte jamais une arête deux fois (il n'y a pas de $x - x$ dans le cycle). Le nombre d'arêtes incidentes à x présentes dans le cycle est donc pair, et comme toutes les arêtes sont dans le cycle une et une seule fois, le degré de x est pair. D'autre part, un sommet n'apparaissant pas dans le cycle est nécessairement de degré nul.
Dans un graphe eulérien, tous les sommets sont de degré pair.

► **Question 4** Le raisonnement précédent reste valable pour un chemin eulérien, sauf pour les deux extrémités. Si un graphe possède un chemin eulérien, il a au plus deux sommets de degré impair.

► **Question 5** Supposons qu'on finisse bloqué en un sommet $y \neq x$. Alors on est « entré » dans y une fois de plus qu'on en est « sorti », et l'on a ce faisant utilisé toutes les arêtes incidentes à y. Donc y est de degré impair, ce qui est absurde. On termine donc nécessairement en x.

► **Question 6** S'il n'y a plus d'arête disponible dans les sommets du cycle, cela signifie que ces sommets forment une composante connexe. Le graphe étant connexe, cela signifie que tous les sommets sont dans le cycle, et donc qu'on a épuisé les arêtes du graphe. Ainsi, le cycle est un circuit eulérien.

▶ **Question 7** Soit G un graphe connexe dont tous les sommets sont de degré pair. On considère le processus suivant :

- on part d'un sommet v quelconque et l'on construit un cycle C comme décrit plus haut
- tant que C n'est pas un circuit eulérien :
  - on trouve un sommet x de C possédant encore une arête disponible
  - on construit un cycle C' autour de x (par le même processus, le graphe ayant toujours ses sommets de degré pair en tenant compte des arêtes retirées)
  - on remplace C par sa fusion avec C' : en écrivant $C : x = x_0, x_1, \dots, x_{k-1} = x$ et $C' : x = y_0, y_1, \dots, y_{l-1} = x$, on pose $C : x = x_0, x_1, \dots, x_{k-1}, y_1, \dots, y_{l-1} = x$.

À chaque étape C est un cycle simple, et le nombre d'arêtes disponibles décroît strictement. Le processus se termine donc avec un cycle simple C, qui est un circuit eulérien d'après la question précédente. On en déduit que la condition nécessaire trouvée en 3 est en fait suffisante :

un graphe connexe possède un circuit eulérien si et seulement si tous ses sommets sont de degré pair.

▶ **Question 8** Si le graphe ne possède pas de sommet de degré impair, il a un circuit (et donc un chemin) eulérien d'après la question précédente. S'il possède deux sommets de degré impair, on commence le processus précédent à l'un de ces deux sommets. On termine forcément à l'autre, puis on greffe des cycles supplémentaires sur ce chemin comme précédemment, jusqu'à épuisement des arêtes. Notons qu'il ne peut y avoir un unique sommet de degré impair. Combiné à la question 4, on peut conclure que

G possède un chemin eulérien si et seulement si il a au plus deux sommets de degré impair.

▶ **Question 9** Les résultats précédents s'étendent tels quels aux multigraphes (en définissant bien sûr le degré d'un nœud comme le nombre d'arêtes incidentes à ce nœud) : il n'y a rien à changer dans la démonstration. Le problème se réduit à la recherche d'un chemin eulérien dans le graphe ci-dessous :

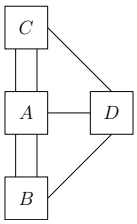


Figure XXXVII.7 – Le graphe des ponts de Königsberg.

Il y a quatre sommets de degré impair, un tel chemin n'existe donc pas.

▶ **Question 10**

En partant de 0

- On commence par empiler 0, 1, 2, 0, 7, 0 dans `actuel`.
- On dépile 0, 7, 0, 2 pour arriver en 1 qui a encore des arêtes disponibles.
- On a alors (sommet à droite) $\texttt{courant} = 0, 1$ et $\texttt{euler} = 0, 7, 0, 2$.
- On empile dans `courant` jusqu'à obtenir $\texttt{courant} = 0, 1, 3, 4, 1$.
- On dépile pour obtenir $\texttt{courant} = 0, 1, 3$ et $\texttt{euler} = 0, 7, 0, 2, 1, 4$.
- On empile jusqu'à $\texttt{courant} = 0, 1, 3, 5, 6, 3$.
- On dépile jusqu'à $\texttt{courant} = \emptyset$ et $\texttt{euler} = 0, 7, 0, 2, 1, 4, 3, 6, 5, 3, 1, 0$

▶ **Question 11**

```c
int *read_data(int *nb_vertex, int *nb_edges){
    scanf("%d %d", nb_vertex, nb_edges);
    int *data = malloc(2 * *nb_edges * sizeof(int));
    for (int i = 0; i < *nb_edges; i++) {
        int x, y;
        scanf("\n%d %d", &x, &y);
        data[2 * i] = x;
        data[2 * i + 1] = y;
    }
    return data;
}
```

▶ **Question 12** C'est une traduction directe de l'algorithme, les structures de données étant fournies.

```c
stack *euler_tour(graph g){
    int p = g.nb_edges;
    stack *euler = stack_new(2 * p);
    stack *current = stack_new(2 * p);
    stack_push(current, 0);
    while (!stack_is_empty(current)) {
        int v = stack_peek(current);
        if (has_available_edge(g, v)) {
            edge e = get_edge(g, v);
            stack_push(current, e.to);
            delete_edge(g, e);
        } else {
            stack_pop(current);
            stack_push(euler, v);
        }
    }
    stack_free(current);
    return euler;
}
```

▶ **Question 13** On peut écrire la fonction `main` suivante :

```c
int main(void){
    int nb_vertex;
    int nb_edges;
    int *data = read_data(&nb_vertex, &nb_edges);

    graph g = build_graph(data, nb_vertex, nb_edges);

    stack *tour = euler_tour(g);

    while (!stack_is_empty(tour)) {
        int v = stack_pop(tour);
        printf("%d ", v);
    }
    printf("\n");


    stack_free(tour);
    graph_free(g);

    return 0;
}
```

Comme on affiche le circuit en le dépilant, on l'obtient « à l'envers » (ce qui ne serait gênant que si le graphe était orienté). On obtient bien :

```
$ ./euler < g3.txt
0 1 3 5 6 3 4 1 2 0 7 0
```