

# ALGORITHMIQUE ET STRUCTURES DE DONNÉES

## TP 3. Graphes

28 Janvier 2026

### 1 Parcours en largeur d'un graphe

Implémenter un parcours en largeur dans le fichier `parcours_largeur.cpp`. La structure de graphe est définie dans `graphe.h`. Le membre `A` de la structure de graphe, est un vecteur de listes d'adjacences. `A[u]` est un vecteur contenant tous les voisins de `u`.

Votre but sera de renvoyer un vecteur contenant les distances entre la source et tous les autres sommets.

Vous pourrez utiliser la structure de file de la librairie standard de C++.

```
#include <queue>
```

```
std::queue<int> Q; // crée une file Q
Q.push(x); // Ajoute un élément à Q
Q.front(); // Retourne l'élément le plus ancien de Q
Q.pop(); // Supprime l'élément le plus ancien de Q
```

### 2 Plus court chemin dans un labyrinthe [Exercice Noté]

*Cet exercice est à rendre par mail à l'adresse courriel [yoann.coudert-osmont@ens-lyon.fr](mailto:yoann.coudert-osmont@ens-lyon.fr) avant le **Lundi 2 Février à 23h59**.*

Dans l'archive du TP vous trouverez des fichiers `labyrinthe1.txt`, `labyrinthe2.txt`, `labyrinthe3.txt` et `labyrinthe4.txt`. Ouvrez-en un et observez le fichier. Ces fichiers commencent pas une ligne contenant deux nombres :

- $n$  : Le nombre de lignes du labyrinthe
- $m$  : Le nombre de colonnes du labyrinthe

S'en suit alors  $n$  lignes composées de  $m$  caractères chacune.

Les caractères sont soit un chiffre entre '1' et '9' (inclus), soit le caractère '#'. Ces derniers indiquent un mur. On ne peut se déplacer sur ces cases.

Il est possible de se déplacer sur une case contenant un chiffre depuis une case voisine. Si le chiffre de la case est  $x$  alors cette action nécessite  $x$  secondes.

**Votre objectif :** Vous déplacer de la première case en haut à gauche du labyrinthe vers la dernière case en bas à droite en prenant le moins de temps possible.

**Exemple :**

```

1-2-1 #
# 4 1 #
# 4 3 #
# 2 2-5

```

Le chemin représenté ici nécessite un temps :

$$2 + 1 + 1 + 3 + 2 + 5 = 14$$

pour être traversé. Attention, le 1 sur la case de départ n'est pas pris en compte. En effet on commence de cette case, mais on ne se déplace pas sur cette case. Le temps pris par un déplacement est le temps écrit sur la case d'arrivée.

**Implémentation** Vous trouverez un fichier `labyrinthe.cpp` à compléter.

- Commencez par implémenter la fonction `construire_graphe` qui construit un graphe **pondéré** à partir des lignes du labyrinthe.
- Implémentez ensuite la fonction `dijkstra` qui prend en entrée un graphe pondéré et un sommet source. Pour ce faire, vous utiliserez les files de priorité de C++ `std::priority_queue`.

```
#include <queue>
```

```
// crée une file de priorité Q dont les éléments sont des paires d'entiers
std::priority_queue<std::pair<int, int>> Q;
```

```
Q.emplace(x, y); // Ajoute un élément {x, y} à Q
Q.top(); // Retourne l'élément de plus grande valeur dans Q
Q.pop(); // Supprime l'élément le plus grande valeur dans Q
```

**Attention :** `std::priority_queue` est un tas max. Si vous voulez que `Q` vous retourne la plus petite distance, vous pouvez par exemple insérer l'opposée de la distance, c'est à dire utiliser `Q.emplace(-d, u)` pour insérer un sommet `u` situé à une distance `d` de la source. Une autre solution est de changer la définition de `Q` par :

```
std::priority_queue<
    std::pair<int, int>,
    std::vector<std::pair<int, int>>,
    std::greater<std::pair<int, int>>
> Q;
```

- **BONUS :** Vous trouverez en haut du fichier une structure `Tas` à implémenter. Vous pouvez compléter les méthodes manquantes et ensuite utiliser ce `Tas` pour remplacer `std::priority_queue` dans votre fonction `dijkstra`.

**Résultats** Si vous obtenez les résultats suivants, tout est bon :

Fichier	Distance
labyrinthe1.txt	44
labyrinthe2.txt	146
labyrinthe3.txt	963
labyrinthe4.txt	2015

### 3 Calcul d'un arbre couvrant de poids minimal

Implémentez l'algorithme de calcul d'arbre couvrant de poids minimal de votre choix. Cette fois-ci vous n'aurez pas de fichiers à compléter. Vous pouvez réutiliser `graphe.h`. Vous aurez à construire vous-même un graphe sur lequel tester votre algorithme.