

Théorie des Graphes

informatique / licence 3

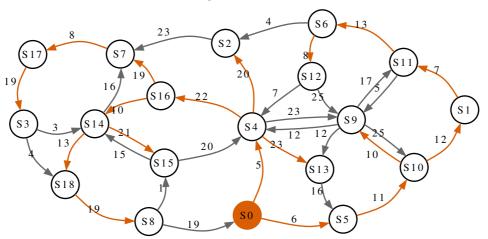
On continue d'explorer les graphes orientés, en ajoutant cette fois une fonction de pondération. La fonction poids donne le poids entre un sommet de départ et un sommet d'arrivée (à condition que l'arête existe).

On poursuit le travail sur les types implémentés lors de la séance précédente. Les fonctions déjà codées seront indispensables pour les questions qui suivent.

On implémente l'algorithme le Dijkstra sur un graphe aléatoire (créé avec la même fonction que dans le TP 3). On cherche à produire un graphique faisant apparaître les sommets et les arêtes, les poids des arêtes — et mettant en évidence à l'aide d'une couleur :

- un sommet de départ;
- les plus courts chemins depuis le sommet de départ.

Le graphique attendu ressemblera à la figure suivante :



La fonction main créera un graphe aléatoire à l'aide de la commande :

Il sera possible d'afficher le graphe, pour en prendre connaissance, à l'aide de write_graphviz_poids(stdout, graphe);

1. Implémenter l'algorithme de Dijkstra dans une fonction de signature

void dijkstra(graphe_t *g, int sommet, double *dist, int *pred);

où g est un pointeur vers un graphe, et où sommet est un numéro de sommet arbitraire. Les deux autres variables sont des pointeurs vers des tableaux numériques *créés par l'utilisateur et non initialisés*. La fonction dijkstra remplira ces fonctions lors de son exécution et l'utilisateur disposera des résultats dans ces tableaux.

La fonction pourra de son côté utiliser un troisième tableau, de booléens, indiquant si un sommet a déjà été extrait de l'ensemble initial ou non. Ce tableau ne sera pas renvoyé.

La procédure d'extraction du sommet minimal ne s'appuiera pas sur une structure spécifique. On se contentera de parcourir les trois tableaux à la recherche d'un sommet encore disponible et dont la distance au sommet de départ est minimale. 2. Écrire une fonction print_paths de signature

```
void print_paths(graphe_t *g, int sommet, int *pred);
affichant les trajets minimaux entre le sommet de départ et tous les sommets du graphe
sous la forme suivante :
```

```
S0 -> S0 : S0

S0 -> S1 : S0 S5 S1

S0 -> S2 : S0 S5 S1 S6 S11 S2

S0 -> S3 : S0 S5 S1 S6 S11 S2 S12 S7 S17 S3

S0 -> S4 : S0 S5 S1 S4

S0 -> S5 : S0 S5

etc.
```

Une fonction auxiliaire (récursive) pourra être utilisée.

3. Écrire une fonction write graphviz2 de signature

```
void write_graphviz2(FILE *f, graphe_t graphe, double *dist, int *pred);
produisant un graphique similaire à celui de la page précédente.
```

Le début du fichier (en langage dot) sera :

```
digraph G {
 layout=sfdp
 splines=curved
 node [
  colorscheme=dark28
  penwidth=1.75
  shape=circle
  fixedsize=true
]
 edge [
  colorscheme=dark28
  penwidth=1.75
  arrowhead=normal
]
```

Le sommet de départ sera déclaré de la façon suivante :

```
S0 [ color=2, style=filled, fillcolor=2 ]
```

(ce sommet se reconnaît car il est le seul à avoir une valeur égale à -1 dans le tableau des prédécesseurs pred) — les autres sommets ne seront pas déclarés.

Les arêtes seront tracées avec leur poids et leur couleur de la façon suivante :

```
S1 -> S11 [label=7, color=2]
S2 -> S7 [label=23, color=8]
```

où la couleur 8 est utilisée pour les arêtes inutilisées et la couleur 2 pour les arêtes utilisées dans les plus courts chemins.