Algorithmique et Structures de données 1

L2 2021-2022

Travaux Pratiques 10

Site du cours : `https://defelice.up8.site/algo-struct.html`
Les exercices marqués de (@) sont à faire dans un second temps.
Un fichier écrit en langage C se termine conventionnellement par `.c`.
Une commande de compilation est `gcc fichier_source1.c fichier_source2.c fichier_source3.c`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall -Wextra` pour demander au compilateur d'afficher plus de Warnings
- `-std=c11` pour compiler selon la norme C11
- `-g -fsanitize=address` pour compiler avec information de débogage et en interdisant la plupart des accès a une zone mémoire non reservée.

Exemple : `gcc -Wall fichier1.c -o monprogramme`

Bien qu'elle soit très gourmande en espace, on utilisera la définition de type suivante pour représenter un graphe étiqueté. Le champs `liste` est un tableau de tableau utilisé pour stocker le graphe par liste d'adjacence.

Chaque élément de la liste est un élément de type `couple_t` destiné à contenir un sommet qui sera le sommet adjacent et une valeur qui sera l'étiquette de l'arc arrivant à ce sommet.

La fin d'une liste sera codée par un élement ayant la valeur -1 pour étiquette de sommet.

```
typedef struct
{
   int s; // sommet
   int e; // etiquette
} couple_t ;

# define MAX_SOMMET 40  // Le nombre maximum de sommets d'un graphe
typedef struct gra
{
   couple_t liste[MAX_SOMMET][MAX_SOMMET];
   int n; // nombre de sommets
} grapheE_t;  // graphe étiqueté
```

Exemple :

Le tableau de tableau suivant :

$(1:3)$	$(3:2)$	$(4:1)$	$(-1:*)$			
$(2:1)$	$(-1:*)$					
$(-1:*)$						
$(0:2)$	$(1:1)$	$(-1:*)$				
$(4:3)$	$(-1:*)$					

représente le graphe suivant :

- $0 \longrightarrow 1:3$
- $0 \longrightarrow 3:2$
- $0 \longrightarrow 4:1$
- $1 \longrightarrow 2:1$
- $3 \longrightarrow 0:2$
- $3 \longrightarrow 1:1$
- $4 \longrightarrow 4:3$

Exercice 1. *Création de graphe*

Écrire la fonction `void creerGraphe(grapheE_t* g,FILE* grDesc)` qui construit un graphe à partir du flux `grDesc`.

Exemple de contenu du fichier `grDesc`.

```
7
0->6:3
3->3:2
6->2:1
1->0:3
2->0:6
```

le graphe a 7 sommets 0 1 2 3 4 5 6

Pour la lecture on peut utiliser `fscanf(grDesc,"%d->%d:%d",&d,&a,&e)`.

Exercice 2. *Dijikstra*

Créer la fonction `void dijikstra(grapheE_t* g,int depart,couple_t* t)` qui remplit le tableau `t`. Après appel,...

- ...le champs `e` de la case `t[i]` doit contenir la distance du sommet `depart` au sommet `i`. (si le sommet n'est pas acessible alors on conviendra que le champs `e` doit contenir `-1` après appel).
- ...le champs `s` de la case `t[i]` doit contenir l'avant dernier sommet du plus court chemin allant du sommet `depart` vers le sommet `i`. (Seulement si ce plus court chemin existe)

Exercice 3. *Affiche plus court chemin*

Créer la fonction `void afficheChemin(int s,couple_t* t)` qui affiche le plus court chemin du sommet de départ vers le sommet `s`. `t` est un tableau préalablement remplit par la fonction `dijikstra` et le sommet de départ est justement l'argument `depart` de `dijikstra`.