UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

t.p.

Théorie des Graphes

informatique / licence 3

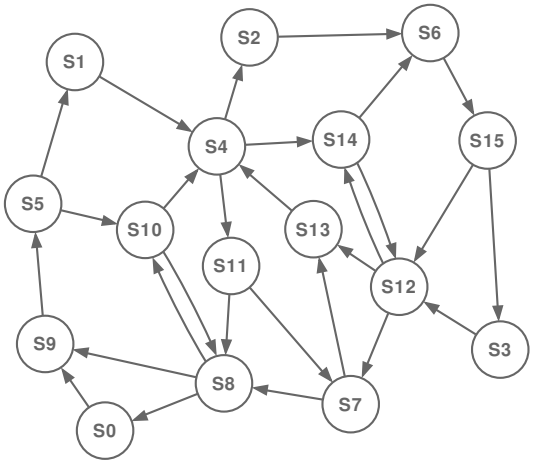
On continue d'explorer les graphes, en revenant aux graphes orientés, mais en implémentant désormais un parcours en largeur (en implémentant une file).

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

Le fichier `TP3.c`, fourni pour cette séance, contient une fonction appelée `random_graph`, de signature

```
graphe_t random_graph(int rlevel)
```

qui construit, de façon aléatoire, des graphes planaires, orientés et fortement connexes. La figure ci-contre a été créée avec un niveau de récursivité égal à 2 (ce qui produit des graphes contenant en moyenne une vingtaine de sommets).



On programmera des fonctions calculant la distance entre deux sommets, recherchant le sommet le plus éloigné, traçant un itinéraire entre deux sommets, et calculant le diamètre d'un graphe.

Parcours en largeur d'un graphe

On utilise la structure `chainon_t` des TP précédents pour implémenter une file de sommets. La file est donc assimilée à une liste simplement chaînée : les éléments (entiers représentant des numéros de sommets) sont ajoutés à droite (fin de la liste) et extraits à gauche (début de la liste).

Le type utilisé pour ces manipulations est le suivant :

```
typedef struct {
    chainon_t * debut;
    chainon_t * fin;
} file_t;
```

1. Programmer les deux opérations élémentaires suivantes :

```
chainon_t *inserer_file(file_t *f, int data);
int extraire_file(file_t *f);
```

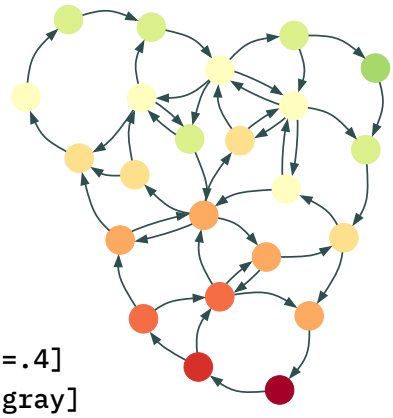
La seconde fonction doit extraire un chaînon, libérer la mémoire nécessaire, etc., *mais aussi renvoyer la valeur qui a été extraite.*

2. Écrire une fonction `distances_sommet` de signature

```
int * distances_sommet(graphe_t graphe, int sommet);
```

calculant toutes les plus courtes distances du sommet spécifié à tous les autres sommets du graphe et renvoyant le tableau de ces distances. Le calcul se fait à l'aide d'un parcours en largeur du graphe.

3. Écrire une fonction `coloration_dist` recourant au langage *dot* (du logiciel *Graphviz* déjà utilisé auparavant), afin d'obtenir une figure similaire à celle ci-contre ; le dégradé fait ressortir la distance de chaque sommet par rapport à l'un d'entre eux.



Le code en langage *dot* commence par les lignes suivantes :

```
digraph G {
    layout=neato
    splines=curved
    node [colorscheme=rdylgn11, shape=point, width=.4]
    edge [width=.4, penwidth=1.75, color=darkslategray]
```

Chaque sommet est ensuite déclaré d'une certaine couleur (on utilisera la distance comme indice de couleur *en démarrant l'indiçage des couleurs à 1 et non à 0*) :

```
    S0 [ color = 1 ]
    S10 [ color = 2 ]
    S5 [ color = 3 ]
    etc.
```

Les arcs (orientés) dont ensuite déclarés comme suit :

```
    S15 -> S13
    S8 -> S14
    S6 -> S20
    etc.
```

4. Écrire une fonction `max_distance_sommet` de signature

```
int max_distance_sommet(graphe_t graphe, int sommet);
```

qui renvoie la distance maximale entre un sommet spécifié du graphe et un autre.

5. Écrire une fonction `diametre_graphe`

```
int diametre_graphe(graphe_t graphe);
```

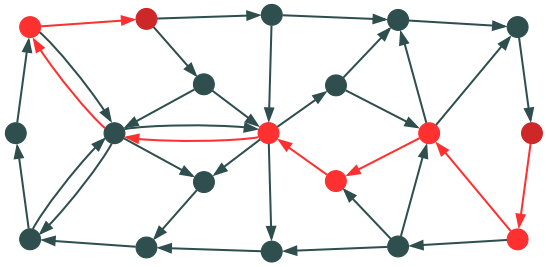
qui renvoie la plus grande des distances (minimales) entre deux sommets du graphe.

6. Calculer le diamètre moyen des graphes obtenus avec la fonction `random_graph` pour plusieurs niveaux de récursivité différents.

Calcul d'un plus court chemin

Le type `liste_t` utilisé jusqu'à présent pour représenter les listes d'adjacence peut également servir à représenter n'importe quelle suite de sommets. On s'en servira dans cette section pour représenter un chemin entre deux sommets.

Dans la figure ci-dessous, un chemin a été coloré en rouge. Il s'agit d'un chemin minimal (non unique) vers un autre sommet situé à une distance 7.



On cherche à construire la liste (chaînée) et correctement ordonnée des sommets constituant ce chemin minimal. Cette liste doit se terminer par le sommet « cible » ; en revanche

elle ne doit pas contenir le sommet de départ. Cette liste aura donc une longueur égale à la distance entre les deux sommets.

Le principe général de la construction de la liste est le suivant :

- lors de l'exploration en largeur, à chaque découverte d'un nouveau sommet, on calcule sa distance et on met à jour un tableau des distances ;
- lors de l'exploration en largeur, à chaque découverte d'un nouveau sommet, on stocke son prédécesseur (le sommet qui a permis de le découvrir) dans un tableau ;
- après l'exploration du graphe, on part du sommet « cible » et on le met dans une liste chaînée, puis on passe à son prédécesseur et on l'ajoute en tête de la liste, et ainsi de suite pour chaque prédécesseur jusqu'à retrouver le sommet de départ.

7. Écrire une fonction `plus_court_chemin` de signature

```
liste_t plus_court_chemin(graphe_t graphe, int sommet1, int sommet2);
```

qui renvoie une liste contenant le plus court chemin entre le premier sommet et le second.