Examen d'algorithmique de graphes   Licence MIAGE 2005-2006

Claire Hanen. Le 6 septembre 2006, durée 2h. Les documents ne sont pas autorisés.

Exercice 1 : tris

Le tri par sélection d'un tableau de n entiers consiste à trouver l'élément maximum du tableau, l'échanger avec le dernier élément du tableau, et recommencer avec le tableau des n-1 premiers éléments.

Q1 : Ecrire un programme C permettant de réaliser ce tri.
Q2 : Dérouler son exécution sur le tableau suivant :

13	5	45	10	2	16	42	25

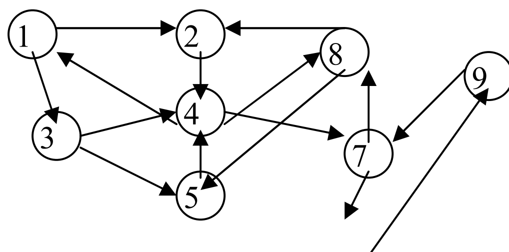
Q3 : Analyser sa complexité dans le pire des cas.
Q4 : En supposant que le tableau est au départ déjà trié par ordre croissant, est-ce que le tri est alors plus rapide (que dans le pire des cas ?).
Q5 : Supposons maintenant que le tableau est implémenté sous forme d'une liste simplement chaînée d'entiers. Implémentez un algorithme de tri par sélection qui retire à chaque étape le plus grand élément de la liste pour constituer la liste résultat (triée par ordre croissant).
Q6 : Cela change-t-il la complexité du tri ?
Q7 : réaliser un tri par tas du tableau de la question 2. (construire graphiquement les tas correspondants en précisant les permutations de clés effectuées)
Q8 : Rappeler la complexité dans le pire des cas du tri par tas. En supposant que le tableau est au départ déjà trié par ordre croissant, est-ce que le tri est alors plus rapide (que dans le pire des cas ?).

Exercice 2 : parcours

Rappel : tout parcours d'un graphe orienté induit une partition des arcs du graphe en :

- l'ensemble des arcs de l'arborescence du parcours : si un sommet y est marqué lors de l'examen des successeurs d'un sommet x, l'arc (x,y) est un arc de l'arborescence. On dit alors que x est le père de y dans l'arborescence. Chaque sommet a donc un unique père.
- L'ensemble des arcs avant ; (x,y) est avant si y est un descendant de x dans l'arborescence du parcours
- L'ensemble des arcs arrière : (x,y) est arrière si x est un descendant de y dans l'arborescence du parcours
- Le reste des arcs est l'ensembles des arcs transverse.

Q1 : Effectuer un parcours en profondeur et un parcours en largeur du graphe suivant à partir du sommet s=1 (en supposant les successeurs rangés dans l'ordre de leur numérotation), dessiner pour chaque parcours son arborescence, et préciser le statut des autres arcs (avant, arrière, transverse)



On considère la version récursive du parcours en profondeur donnée en annexe. Cette version construit deux tableaux : un tableau marque indique pour chaque sommet son numéro d'ordre dans le parcours, et un tableau quelnum qui indique pour chaque sommet un autre numéro.
Q2 : Calculer ces deux numérotations pour l'exemple ci-dessus en considérant un appel à parcours_prof(1). On supposera qu'initialement les tableaux quelnum et marque sont initialisés à 0.

Q3 : Supposons que lors du parcours en profondeur à partir d'un sommet i, il existe un successeur j de i tel que u

Q4 : Même question pour le parcours en largeur.

Q5 : Ces ajouts modifient-ils l'ordre de grandeur de la complexité du parcours que vous rappellerez ?

Q6 : On définit la distance du sommet s à chaque sommet i comme la longueur (en nombre d'arcs) du chemin de s à i dans l'arborescence du parcours. Calculer cette distance pour les deux parcours sur le graphe de la figure 2 (avec s=1)

Q7 : On suppose donné le tableau pere d'un parcours. Ecrire une fonction permettant de calculer pour un sommet, sa distance au sommet s.

Q8 : Utiliser cette fonction pour remplir un tableau distance, indiquant pour chaque sommet sa distance au sommet s. Quelle est la complexité de cette fonction?

Q9 : Comment peut-on intéger le calcul du tableau distance aux fonctions de parcours en largeur et en profondeur, de sorte que la complexité soit plus faible?

Annexe

Parcours en largeur:

```
typedef struct chainon *listesom;
struct chainon{int s; listesom suite;};
void parcours_ite_larg(int s, listesom succ[], int n, int marque[n])   //  itératif en largeur
{ listesom p;
  int l[n], dl,fl,i,j,cpt=1;
  marque[s]=cpt++;       // on note que s a été visité
  l[0]=s, dl=0, fl=1;// l=(s)
  while(dl<fl)       // tant l n'est pas vide
  { i=l[dl++];       // on enlève le premier élément de l et on le met dans i
    for(p=succ[i];p;p=p->suite)  // pour chaque successeur j (=p->s) de i faire
     if(!marque[j=p->s])         //     si j n'a pas encore été visité alors
    { marque[j]=cpt++;              //         on note que j a été visité
      l[fl++]=j;                //         on ajoute j à la fin de la liste l
    }                           //     finsi fait
  }                  // fait
}
```

Parcours en profondeur:

```
typedef struct chainon *listesom;
struct chainon{int s; listesom suite;};
int cpt=1; int autre=1 ;
int marque[100],quelnum[100] ;
listesom succ[100];
void parcours_prof(int i)
{ listesom p; int j ;
  marque[i]=cpt++;
       for(p=succ[i];p;p=p->suite) // pour chaque successeur j de i
              if(!marque[j=p->s]) parcours_prof(j);// appel récursif si j non marqué
```

```
quelnum[i]=autre++}
```