Examen d'algorithmique de graphes Licence MIAGE 2005-2006

Claire Hanen. Le 31 janvier 2006, durée 2h. Les documents ne sont pas autorisés.

Exercice 1 : tris

On considère le tableau suivant

12	4	25	8	2	16	42	30

Montrer les étapes successives du tri du tableau en employant :

Q1 : le tri fusion (préciser les appels récursifs, et l'état des sous-tableaux fusionnés)

Q2 : le tri rapide (indiquer les états du tableau après chaque séparation)

Q3 : le tri par tas (construire graphiquement les tas correspondants en précisant les permutations de clés effectuées)

Rappeler les complexités dans le pire des cas de chacun de ces tris.

Exercice 2 : Files

Une file d'entiers est un ensemble sur lequel deux opérations peuvent être effectuées : ajouter un entier x (on dit enfiler x), ou retirer un entier (on dit défiler). L'entier retiré est toujours celui le plus ancien de la liste – c'est-à-dire le premier ajouté. On peut également tester si la file est vide.

Vous avez vu en TD une implémentation de file à l'aide de tableaux. On se propose ici d'en réaliser une autre en C à l'aide d'une liste dite circulaire : il s'agit d'une liste simplement chaînée dont le dernier élément pointe sur le premier. La liste est accessible à l'aide d'un pointeur sur l'un des maillons que nous appellerons ici F.

Pour pouvoir implémenter une file avec cette structure, on suppose que ce pointeur F pointe toujours sur le dernier élément ajouté dans la file (le plus récent), sauf naturellement quand la file est vide. La figure suivante indique les opérations qui doivent être réalisées en enfilant et défilant.

Q1 : Ecrire les déclarations de type nécessaires en C pour implémenter cette structure, et écrire quatre fonctions :

Int Estvide(File F) // indique si la file est vide

File maillon (x) // crée un nouveau maillon contenant l'entier x

File enfiler (File F, int x)// renvoie le pointeur sur la file après avoir ajouté l'élément x

File défiler (File F, int *px) // défile, renvoie le pointeur sur la file et également l'entier retiré de la file (pointé par px).

Le parcours en largeur dans un graphe utilise une file pour gérer les sommets ouverts. Vous trouverez en annexe le code d'un parcours en largeur qui implémente la file avec un tableau et numérote les sommets (le tableau marque donne une nouvelle numérotation):

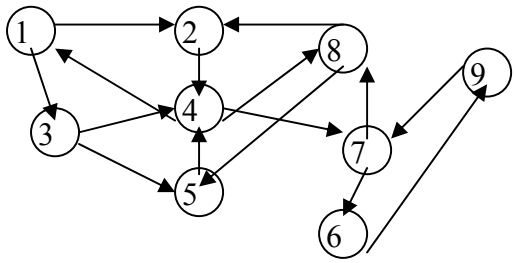
Q2 : Modifier cette fonction en utilisant l'implémentation liste circulaire de la file (et donc les fonctions ci-dessus).

Exercice 3 : 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)



Q2 : Le graphe de la figure 2 est-il fortement connexe (justifier)?

On considère la version récursive du parcours en profondeur où l'on numérote les sommets donnée en annexe.

Q3 : Que faut-il ajouter pour que ce parcours calcule également un tableau pere, dans lequel est stocké pour chaque sommet son père dans l'arborescence du parcours ?

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 marque[100]
listesom succ[100]
void parcours_prof(int i)
{ listesom p;
  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
}
```