Correction TD d'algorithmique n°4

Cours d'algorithmique avancée

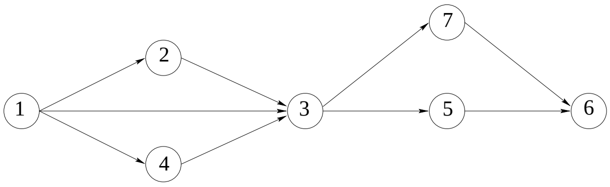
— IMAC première année —

Les graphes - Parcours en largeur

Lors de cette scéance, nous abordons le parcours en largeur d'un graphe.

▸ **Exercice 1.** ***Parcours en largeur d'un graphe - Matrice des distances***

Pour cet exercice nous allons travailler exclusivement avec la représentation `FSAPS` *des graphes sans valeur sur les arcs.*



- *Donner la matrice des distances du graphe ci-dessus (l'entrée (i, j) contient la longueur (en nombre d'arêtes) du plus court chemin allant du sommet i au sommet j) ;*
- *On désire réaliser un parcours en largeur. A quel algorithme déja vu cela vous fait il penser ? Quels sont les problèmes qui se posent ?*
- *Donner un algorithme formel de parcours en largeur d'un graphe G à partir d'un sommet S ;*
- *Étant donné un graphe et un sommet, écrire une fonction qui effectue un tel parcours et renvoie le tableau contenant les distances du plus court chemin allant du sommet d'appel à tous les autres sommets ;*
- *Écrire une fonction* `MatDistances` *qui construit la matrice des distances d'un graphe.*

✂ -

-

$$M_d = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 & 3 & 2 \\ \infty & 0 & 1 & \infty & 2 & 3 & 2 \\ \infty & \infty & 0 & \infty & 1 & 2 & 1 \\ \infty & \infty & 1 & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

- procedure ParcoursLargeur(Graphe : G, Sommet : S)
  var ensemble : X,Y;

  debut
    X <- {S};   /* sommets en cours (niveau) */
    Y <- {};   /* sommets deja visites   */

    tant que X <> {} faire
         Traitement de tous les sommets de X ;
         Y <- Y U X ;   /* mise a jour des sommets visites   */
         X <- Successeurs(X) \ Y ;   /* sommets a traiter a l'etape suivante */
    fait
  fin

- pour un graphe codé par **FS** et **APS** :

```
int * ParcoursLargeurFSAPS(int *fs, int t_fs, int *aps, int t_aps, int s)
{
  File X, Tmp;
  int * Y;                /* tableau des sommets :  0=non visite, 1=visite */
  int * dist;             /* tableau des distances                         */
  int i,j,d;

  InitFile(&X);
  Y = (int*)calloc(t_aps,sizeof(int));

  Enfile(&X,s);
  Y[s-1] = 1;

  dist=(int*)malloc(t_aps*sizeof(int));
  for(i=0;i<t_aps;i++)
     dist[i] = -1;  /* infinie */
  d=0;

  while ( !EstVide(X) )
      {
         /* Traitement : mise a jour du tableau des distances */
         Recopier(&Tmp,X);
         while ( !EstVide(Tmp) )
               {
                 i=Defiler(Tmp);
                 dist[i-1] = d;
               }
         d++;

         Recopier(&Tmp,X);        /* X recoit les successeurs des sommets */
         InitFile(&X);            /* de X non deja marques                */
         while ( !EstVide(Tmp) )
            {
              i=Defiler(Tmp);
              j=aps[i-1];
              while (fs[j] != 0)
```

```
                {
                  if ( Y[fs[j]-1] == 0 )
                     {
                       Enfiler(&X,fs[j]);
                       Y[fs[j]-1] = 1; /* Mise a jour des sommets visites */
                     }
                  j++;
                }
            }
       }
  return dist;
}
/* Avec ce type de traitement, le tableau Y n'est pas indispensable : on peut
   utiliser a la place le tableau des distances (dist[i]==-1 si non marque) */
```

- Matrice des distances :

```
int ** MatDistancesMA(int **m, int n)
{
  int **md;
  int s;

  md = (int**)malloc(n*sizeof(int*));
  for(s=1;s<=n;s++)
     md[s-1] = ParcoursLargeurMA(m,n,s);

  return md;
}

int ** MatDistancesFSAPS(int *fs, int t_fs, int *aps, int t_aps)
{
  int **md;
  int s;

  md = (int**)malloc(t_aps*sizeof(int*));
  for(s=1;s<=t_aps;s++)
     md[s-1] = ParcoursLargeurFSAPS(fs,t_fs,aps,t_aps,s);

  return md;
}
```