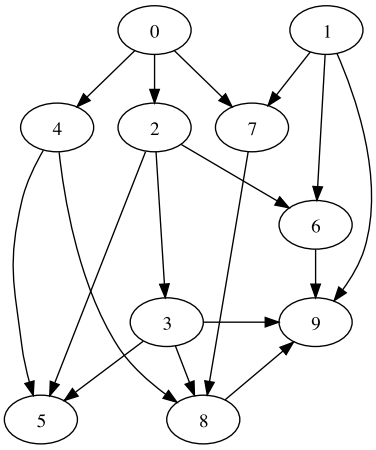
Algorithmique : Contrôle de TP

Université de Lille-1
Licence d'Informatique
charles.bouillaguet@lifl.fr

1 décembre 2012

Le but du TP est de programmer une fonction qui compte le nombre de chemins entre deux sommets d'un graphe orienté acyclique (ce qui est abrégé en "DAG" dans la suite).



Par exemple, dans ce graphe, il y a **5** chemins entre "0" et "9" (l'un d'entre eux est : 0–2–6–9).

Avant de commencer à programmer

Ne paniquez pas.

Avant de commencer à répondre aux questions, veuillez lire complétement le fichier `path_count.c`. Cela vous permettra de comprendre les questions qui vous sont demandées.

Dans toute cette partie, vous ne devez pas changer les entêtes des fonctions, ni le contenu de la fonction `main` (à part décommenter des tests). Complétez le fichier `CR.txt` au fur et à mesure.

Question 1. Trouvez (à la main) les 4 autres chemins entre 0 et 9 dans le graphe ci-dessus.

Tri topologique

Les deux fonctions qui effectuent le tri topologique sont `DFS` et `tri_topologique`. La fonction `DFS` accompli un parcours en profondeur (Depth-First Search en anglais).

`tri_topologique(graphe,s,pile)` initialise un tableau de couleurs, marque tous les sommets `BLEU`, puis appelle `DFS(graphe,s,couleurs,pile)`.

`DFS(graphe, s, couleurs, pile)` est une fonction *récursive* dont le pseudo-code est le suivant :
- **Marquer** le sommet `s` en `VERT`
- **Pour chaque successeur** `u` de `s` :
  - **Si** `u` est `BLEU` **alors** appeler `DFS(graphe,u,couleurs,pile)`
- **Marquer** le sommet `s` en `ROUGE`
- **Empiler** le sommet `s`

Ceci accomplit un parcours en profondeur (en effet, quand on découvre un noeud et qu'on le marque `VERT`, on s'occupe de ses enfants avant de s'occuper de ses frères). A la fin, la pile contient l'ensemble des sommets marqués `ROUGE`, avec celui qui a été marqué en dernier sur le dessus de la pile.

Si on dépile les sommets les uns après les autres, on va donc les trouver dans l'ordre topologique. Notez qu'on ne trouve que les sommets accessibles à partir de `s`, mais ce n'est pas gênant.

Question 2. Complétez la fonction `tri_topologique`.

Question 3. Complétez la fonction `DFS`.

Question 4. Compilez le programme avec la commande `gcc -Wall -std=c99 path_count.c graphe.c -o path_count`. Vous utiliserez toujours cette commande de compilation. Ignorez les avertissements portant sur les variables inutilisées de la fonction `main`, et sur les `control reaches end of non-void function`.

Question 5. Décommentez dans la fonction `main` le test correspondant au tri topologique et vérifiez que votre programme fonctionne bien en le testant sur `exemple_tp.grp` (qui décrit le graphe de l'introduction), et sur `petersen.grp`.

Test d'acyclicité

Comme compter les chemins n'a de sens que pour les DAGs, on va d'abord programmer une fonction qui teste si on ne peut pas atteindre un cycle depuis le sommet de départ s (en effet, il y aurait alors une infinité de chemins, en empruntant autant de fois la boucle qu'on veut).

Pour ça, la procédure est simple :

- On suppose que le graphe est bien un DAG, et on en fait un **tri topologique** en partant de s.
- **Marquer** tous les sommets en `BLEU`
- **Pour chaque sommet** u (dans l'ordre topologique) :
  - **Marquer** u en `VERT`
  - **Si** un successeur de u est déjà `VERT` **alors** le graphe contient un cycle.
- **Si** on n'a pas trouvé de cycle, **alors** le graphe est bien un DAG

Question 6. Complétez la fonction `is_DAG`.

Question 7. Décommentez dans la fonction `main` le test correspondant. Vérifiez que votre programme fonctionne bien. `cycle.grp` n'est pas un DAG, mais tous les autres en sont.

Comptage de chemins

Pour compter les chemins, on utilise une variante de l'algorithme de plus courts chemins dans un DAG. L'idée principale est que le nombre de chemins de s vers t est la somme des nombres de chemins de s vers chacun des prédécesseurs de t. Pour compter les chemins de s vers chacun des autres sommets, on peut donc utiliser la procédure suivante :

- On effectue un **tri topologique** du graphe, partant de s
- On **initialise** un tableau N (à la fin, $N[u]$ contiendra le nombre de chemins de s vers u).
- **Pour chaque sommet** u, on pose $N[u] \leftarrow 0$.
- On fixe $N[s] \leftarrow 1$.
- **Pour chaque sommet** u (dans l'ordre topologique) :
  - **Pour chaque successeur** v de u :
    - $N[v] \leftarrow N[v] + N[u]$.

Question 8. Complétez la fonction `path_count`.

Question 9. Décommentez dans la fonction `main` le test correspondant. Lancez votre programme sur tous les graphes d'exemple, en partant du sommet "0".

Enumération de chemins

Question 10. Programmez une fonction qui prend en argument deux sommets s et t, et qui affiche l'ensemble des chemins de s vers t (on peut le faire en 10–15 lignes). Ajoutez un test à la fonction `main`, et testez votre fonction.