TP N°03 : Algorithmes de graphes

Objective TP: Implémentation d'algorithmes pour résoudre quelques problèmes dans les graphes.

Remarque : Ecrivez des commentaires dans vos programmes.

Exercice 3.1 (Parcours de graphes)

- 1) Implémenter l'algorithme de parcours en largeur (Breadth First Search = BFS) du cours `void BFS(GRAPH G, int S0)` qui affiche l'ordre de parcours en largeur des nœuds du graphe G (le sommet de départ du parcours est la racine S_0).
- 2) Implémenter l'algorithme de parcours en profondeur (Depth First Search = DFS) du cours `void DFS(GRAPH G, int S0)` qui affiche l'ordre de parcours en profondeur des nœuds du graphe G (le sommet de départ du parcours est la racine S_0).

Exercice 3.2 (Problème du plus court chemin)

- 1) Ecrire une fonction `void Dijkstra(GRAPH G, int S0, int pere[])` sur le modèle de l'algorithme de *Dijkstra* vu en cours. La racine de l'arbre des plus courts chemins est le sommet S_0 . En sortie, le tableau `pere` qui représente l'arbre des plus courts chemins.
- 2) Ecrire une fonction `void BellmanFord(GRAPH G, int S0, int pere [])` sur le modèle de l'algorithme de *Bellman-Ford* vu en cours. La racine de l'arbre des plus courts chemins est le sommet S_0 . En sortie, le tableau `pere` qui représente l'arbre des plus courts chemins.

Exercice 3.3 (Arbres couvrants minimaux)

- 1) Ecrire une fonction `void Kruskal(GRAPH G, int K[])` qui applique l'algorithme de *Kruskal* au tableau d'arêtes et construit le tableau arbre `K` qui contient les $n - 1$ arêtes de l'arbre de distance minimum.
- 2) Ecrire une fonction `void Prim(GRAPH G, int K[])` qui applique l'algorithme de *Prim* au tableau d'arêtes et construit le tableau arbre `K` qui contient les $n - 1$ arêtes de l'arbre de distance minimum.