TP 9 - Graphe d'un terrain et diagramme de Voronoï

Graphe structuré en grille rectangulaire

On considère un graphe non orienté dont les sommets sont plongés dans une grille rectangulaire et sont dotés d'une altitude. Le sommet (i, j) positionné sur la $i$ème ligne et la $j$ième colonne de la grille est muni d'une altitude $h(i, j)$. Les seuls voisins possibles d'un sommet (i, j) sont les sommets à sa gauche $(i, j-1)$, à sa droite $(i, j+1)$, au dessus $(i-1, j)$ et en dessous de lui $(i+1, j)$ (on les désignera sous l'appellation voisins Ouest, Est, Nord et Sud), mais cela n'a pas besoin d'être stocké de manière explicite. Cela signifie qu'on ne va pas utiliser une représentation par matrice ou par liste d'adjacence pour stocker le graphe. Les sommets du bord de la grille ont bien entendu moins de voisins. La valuation d'une arête correspond à la distance Euclidienne entre ses deux extrémités 3D (par exemple la valuation de l'arête joignant le sommet (i, j) à son voisin Nord $(i-1, j)$ est $sqrt(1 + sqr(h(i, j) - h(i-1, j)))$. On désigne par L et C le nombre de lignes et de colonnes de la grille.

On se propose de mettre en œuvre un tel module graphe représentant un Terrain, en utilisant uniquement un tableau 1D de taille $L \times C$ contenant des informations de hauteur. Le sommet (i, j) $(0 \leq i \leq L-1,\ 0 \leq j \leq C-1)$ est représenté par l'indice global $i * C + j$ de la case du tableau qui contient la valeur de son altitude $h(i, j)$. Il est inutile de créer une structure de donnée arête, puisqu'on connaît la position et l'altitude des voisins d'un sommet (i, j).

Parmi les procédures d'initialisation d'un Terrain, en prévoir une pour charger un graphe sauvegardé dans un fichier.

Format de stockage d'un graphe dans un fichier :

```
15 24    // Dimensions du graphe L (largeur) et H (hauteur)
12 34 5 .... //Altitudes des L*H sommets listés ligne par ligne
             // (du haut vers le bas et de gauche vers la droite)
```

En plus des opérations classiques d'initialisation, d'affectation et de testament de graphe, prévoir des fonctions d'accès à l'indice global d'un sommet en fonction de ses indices de ligne ou de colonne, d'accès à l'altitude d'un sommet (en fonction de son indice global), d'accès à l'indice global du voisin Nord (resp. Sud, Est et Ouest) d'un sommet (avec pour précondition que le voisin existe) ainsi que des procédures de modification de l'altitude d'un sommet et une procédure d'affichage de la grille de hauteur.

Faire un petit programme qui illustre que votre module met correctement en œuvre toutes les fonctionnalités demandées pour les graphes de terrain.

1 Connaître la librairie la plus proche : diagramme de Voronoï

L'algorithme de Dijkstra vu en cours permet de trouver les plus courts chemins entre UN unique nœud `n` et tous les nœuds accessibles d'un graphe orienté valué, mais il peut également être mis en oeuvre sur un graphe non orienté. Cela signifie seulement que toutes les arêtes sont orientées dans les deux sens.

Vous allez à présent implanter et modifier l'algorithme de Dijkstra de telle sorte que l'origine des plus courts chemins ne sera plus un unique site de départ, mais UN ENSEMBLE de N sites n_i, $0 \leq i \leq N-1$ correspondant à des librairies dans notre graphe de terrain. A la fin de l'exécution

de l'algorithme de Dijkstra, pour tous les nœuds de la grille, vous obtiendrez ainsi le chemin le plus court depuis la librairie la plus proche. Après avoir exécuté votre version multi-source de l'algorithme de Dijkstra, comment faites-vous pour connaître la librairie la plus proche d'une position donnée ?

En plus des hauteurs fournies en entrée, l'utilisateur fixera (ou pourra récupérer dans un fichier) les coordonnées (indices de ligne et de colonne) d'un ensemble de sites choisis parmi les sommets de la grille. La grille modélise le site géographique et ces coordonnées matérialisent les positions des librairies. Suivant où vous vous trouvez, vous êtes dans le secteur d'une librairie différente et c'est ce que vous allez matérialiser. La cellule de Voronoï du i^{eme} site est l'ensemble des sommets de la grille qui sont plus proches de ce site que des autres sites (en intégrant bien l'information de hauteur dans les calculs de longueur de chemin, comme évoqué plus haut pour connaître le coût d'un arc).

Faites une procédure Voronoï dans laquelle vous enregistrerez et vous visualiserez pour chaque point de la grille l'indice du site dont il est le plus proche à l'issu de l'algorithme de plus court chemin (ainsi que la distance). Ainsi votre affichage final permettra de faire apparaître le diagramme de Voronoï de l'ensemble des sites de librairies. Cet affichage peut-être réalisé sur la sortie standard ou dans un fichier.

Faîtes attention à ce que votre programme de test puisse prendre un fichier en entrée, de manière à ce qu'il soit facile à l'utilisateur de modifier la valeur de hauteur de certains points, ainsi que le nombre et les coordonnées des sites.

Pistes de travail : On peut stocker les infos relatives au parcours du graphe (couleur, prédécesseur, longueur de chemin) dans des tableaux de taille $L \times C$, indexés dans le même ordre que les $L \times C$ sommets du graphe.

2 Choisir la meilleure librairie pour la livraison

On se propose à présent d'établir la cartographie de la meilleure librairie pour le service de livraison. Il s'agit là encore de produire un diagramme de Voronoï mais pour une nouvelle fonction de coût. En effet, chaque librairie est caractérisée par un tarif différent du coût de livraison par km. Le taux kilométrique pratiqué par chaque librairie intervient donc dans le coût de livraison, par simple multiplication par la distance parcourue.

Faites une procédure VoronoïLivraison qui permet de visualiser pour chaque point de la grille, la meilleure librairie à choisir pour la livraison.

3 Conditions de rendu

Vous déposerez une archive `Nom1_Nom2.tgz` (`Nom1` et `Nom2` étant les noms des 2 étudiants composant le binôme) de votre travail sur Tomuss avant le **mardi 13 décembre** à 23h (le dépôt sera ensuite fermé). **Les deux membres du binôme doivent appartenir au même groupe de TP et il est interdit de se greffer au travail d'un étudiant dans un autre groupe**.

Votre archive doit contenir un répertoire `Nom1_Nom2` avec les sources de votre application. Les sources sont bien évidemment tous les fichiers `.cpp` et `.hpp`, mais aussi le fichier `Makefile`, ainsi que tout autre fichier (`README`, etc.) utile à la compréhension de votre programme, ainsi qu'à sa compilation et à son exécution.

Le fichier `Makefile` doit modéliser clairement les dépendances entre les fichiers et permettre une recompilation partielle, en cas de modification partielle.

Attention !

— Votre archive NE DOIT PAS contenir de fichiers objets (.o) ni d'exécutable.
— Le travail rendu doit être issu de la collaboration entre 2 personnes et toute récupération flagrante du code d'autrui sera sanctionnée.
— Votre programme doit pouvoir être compilé sans erreur ni avertissement avec `g++` sans nécessiter l'installation de bibliothèques supplémentaires sur une des machines de la salle de TP.
— une attention particulière devra être consacrée à la mise en œuvre d'une programmation modulaire débouchant sur la proposition de véritables types abstraits.
— l'interface de vos modules doit offrir toutes les informations utiles à l'utilisation des types et des fonctionnalités offertes.
— soignez bien votre programme principal (`main.cpp`) qui doit illustrer que les opérations à coder ont été faites correctement.

Le **mercredi 14 décembre**, vous serez amenés à répondre à une petite interrogation sur votre travail puis à faire une petite présentation au cours de laquelle des questions seront posées à chacun des 2 membres du binôme.

Profitez du mercredi 7 décembre après-midi pour avancer sur votre TP avec votre binôme.