TP 9 - Graphe d'une image contenant une forme 2D

Les algorithmes sur les graphes sont appliqués avec succès pour résoudre des problèmes de vision par ordinateur et de traitement d'images. Etant donné une image contenant **une forme noire sur fond blanc**, nous allons **calculer la distance à cette forme de tous les pixels** blancs (les pixels de la forme sont à distance nulle). On obtient ainsi une description très utilisée en informatique graphique pour modéliser et combiner des formes.

Dans ce travail vous aller mettre en place une application qui prendra en entrée une image d'une forme noire sur fond blanc, et qui renverra une autre image dite "image de distance" où la forme d'origine sera entourée par un dégradé d'intensité lumineuse qui augmentera avec la distance à la forme. Votre application devra également être capable de fournir l'opération inverse, à savoir restituer l'image d'une forme noire sur fond blanc à partir d'une image de distance. Etant donné deux images de distance correspondant à des formes différentes, votre application permettra enfin de construire l'image de distance correspondant à l'**union des 2 formes**, avec une **complexité linéaire** par rapport à la taille (commune) des deux images.

Vous fournirez par ailleurs un algorithme capable de **projeter un pixel de l'image de départ sur la forme**, c'est à dire sur le pixel de la forme le plus proche. Un préalable à l'utilisation de cet algorithme sera le calcul de l'image de distance ainsi qu'un tableau permettant de construire le chemin utile à la projection.

Dans votre application finale, les pixels seront indiqués par leurs coordonnées (ligne, colonne) et l'intensité de niveau de gris d'un pixel sera inférieure ou égale à 255. Les images que l'on considèrera auront une largeur et une hauteur inférieure à 120.

D'une image à un graphe

On peut considérer les pixels d'une image comme les nœuds d'un graphe orienté où chaque pixel est relié par un arc sortant à ses 8 pixels voisins (voisins dans les directions horizontales, verticales mais aussi diagonales). Comme les pixels sont organisés en une grille rectangulaire, il est possible de voir le graphe de l'image comme un tableau 2D où l'on sait où trouver les voisins de chaque nœud sans avoir à coder cette relation de voisinage. Ainsi, on sait que le pixel $(3, 7)$ est voisin des pixels $(3, 6)$, $(4, 6)$, $(4, 7)$, $(4, 8)$, $(3, 8)$, $(2, 8)$, $(2, 7)$, $(2, 6)$, par simple calcul sur les coordonnées du pixel. Dans le tableau modélisant l'image, nous proposons simplement que chaque pixel stocke son information d'intensité lumineuse.

Travail à faire : Pour mettre en œuvre le module graphe d'une image composée de L lignes et C colonnes, on peut **utiliser uniquement un tableau 1D** de taille $L \times C$. Le pixel $p = (i, j)$ $(0 \leq i \leq L-1,\ 0 \leq j \leq C-1)$ est situé dans la case $i * C + j$ du tableau, son intensité lumineuse notée $I(p)$ ou $I(i, j)$ est une valeur entière comprises entre 0 et 255. Les voisins d'un nœud (i, j) sont les nœuds $(i, j-1)$, $(i+1, j-1)$, $(i+1, j)$, $(i+1, j+1)$, $(i, j+1)$,$(i-1, j+1)$, $(i-1, j)$ et $(i-1, j-1)$ (on les désignera sous l'appellation voisins Ouest, SudOuest, Sud, SudEst, Est, NordEst, Nord et NordOuest). Pour l'utilisateur un pixel sera désigné par ses coordonnées, mais dans l'implémentation de vos algorithmes il sera plutôt désigné par son indice global dans le tableau 1D.

Parmi les procédures d'initialisation d'une Image, en prévoir une pour charger une image sauvegardée dans un fichier pgm (supprimer les commentaires pour une lecture de fichier plus rapide, et

regarder les exemples fournis sur la page web du cours pour des rappels sur les lectures de fichiers).

Format de stockage d'une Image dans un fichier (format pgm ascii) :

```
P2       # PGM ASCII
15 24    # Dimensions du graphe C (largeur) et L (hauteur)
255     # Intensité maximale  (couleur blanche)
12 34 5 .... //Intensités des C*L pixels listés ligne par ligne
               // (de gauche à droite et du haut vers le bas)
```

En plus des opérations classiques d'initialisation, d'affectation et de testament du graphe, prévoir des fonctions d'accès à l'indice global d'un nœud positionné sur la ligne i et la colonne j, d'accès à l'indice global du voisin Ouest (resp SudOuest, Sud, SudEst, Est, NordEst, Nord et NordOuest) d'un nœud désigné par son indice global (avec pour précondition que ce voisin existe), d'accès à son intensité lumineuse ainsi que des procédures de modification de cette intensité. Prévoir également une procédure d'affichage de la grille ou de sauvegarde dans un fichier. Un nœud peut être désigné par son indice global dans le tableau ou par ses coordonnées de ligne et de colonne (i, j). Un arc est désigné par le nœud dont il est issu ainsi que par une direction.

1 Calcul des plus courts chemins à la forme et de leur longueurs

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 sites correspondant à l'ensemble des pixels noirs dans l'image de départ**. A la fin de l'exécution de l'algorithme de Dijkstra, pour tous les nœuds blancs de la grille de départ, vous obtiendrez ainsi le chemin le plus court depuis le pixel de la forme la plus proche. Plus précisément, l'algorithme de Dijkstra va remplir deux tableaux indicés par l'indice global des pixels, de la même manière que le tableau modélisant l'image de départ. Dans le premier tableau, l'algorithme de Dijkstra construira progressivement la fonction de distance, et dans le second tableau, l'algorithme consignera progressivement l'indice du prédécesseur dans le plus court chemin menant de la forme au pixel. **La valuation des arcs du graphe correspond ici à la distance entre les 2 pixels voisins, elle est arbitrairement fixée à 2 pour des voisins horizontaux ou verticaux et à 3 pour des voisins diagonaux**. Si le pixel est sur le bord de l'image, il a moins de voisins.

Après avoir exécuté votre version multi-source de l'algorithme de Dijkstra, comment faites-vous pour connaître le pixel de la forme le plus proche d'une position donnée ?

Pistes de travail : On peut stocker les infos relatives au parcours du graphe (tags de visites, 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 Application à la modélisation de formes

Faites un module qui offre les fonctionnalités suivantes :
— **Chargement d'une forme** noire sur fond blanc stockée dans un fichier au format pgm.

— **Sauvegarde d'une forme** dans un fichier pgm.
— **Construction de l'image de distance** d'une forme.
— **Sauvegarde d'une image de distance** dans un fichier pgm.
— Vous pouvez également proposer un format de fichier pour sauvegarder les informations de prédécesseurs dans les plus courts chemins.
— **Projection** d'un pixel d'une image **sur une forme** (renvoie les coordonnées du pixel le plus proche sur la forme).
— Construction de l'image de distance de l'**union de deux formes** à partir de leurs images de distance respectives.
— Etant données 2 formes pour lesquelles on a préalablement calculé les images de distances et les tableaux de prédécesseurs, projection d'un point sur l'union des deux formes, sans recourir une nouvelle fois à l'algorithme de Dijkstra.

Vous veillerez à offrir à l'utilisateur un **exécutable avec un menu** lui permettant de tester toutes les fonctionnalités de votre bibliothèque sur les formes de son choix. Libre à lui de fabriquer des images de formes dans des fichiers pgm.

3 Conditions de rendu

Vous placerez dans Tomuss une archive de votre travail $Nom1\_Nom2.tgz$ ($Nom1$ et $Nom2$ étant les noms des 2 étudiants composant le binôme) avant le **mardi 3 décembre 2024** à 22h.

Cette archive doit contenir un répertoire $Nom1\_Nom2$ avec les sources de votre application. Les sources sont bien évidemment tous les fichiers *.cpp* et *.h*, 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 $g++$ **dans une des salles de TP Linux de l'Université**, et sans nécessiter l'installation de bibliothèques supplémentaires.
— 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.
— l'exécution de votre main doit illustrer le bon fonctionnement de l'ensemble des éléments du projet.

Le **mercredi 4 décembre 2024**, vous serez amenés à répondre à quelques questions sur papier concernant votre travail, puis à faire une petite présentation au cours de laquelle des questions seront posées à chacun des 2 membres du binôme.