UNIVERSITÉ DE BOURGOGNE

RAPPORT:

Algorithmique avancée

-Fatima Zahra Talibi
-Imane Choukri

>>> SOMMAIRE:

Partie1:

Comment modéliser informatiquement une telle grille ?

-Comme marqué sur le sujet , nous avons pris en compte la dimension du rectangle (grille) , ceci nous a permis de conclure que la modélisation de cette grille se fera à partir d'une matrice de M lignes et N colonnes (ces valeurs seront 3 et par défaut). Cependant , comment allons nous visualiser les cellules de cette grille , ainsi que l'épaisseur de chaque mur l'encadrant ?

-Nous avons choisi de donner à chaque épaisseur une valeur , ainsi, plus la valeur augmentera, plus le mur sera épais . Ensuite nous avons constater que : pour définir une cellule ,il faut tout d'abord définir les murs l'encadrant (droit , gauche , haut et bas) .
Finalement , chaque élément de la matrice (cellule) sera définit par une liste de 4 éléments aléatoires (entre 1 et 5) qui définiront respectivement à leur tour l'épaisseur du mur haut, droit, bas puis gauche de la cellule en question .
Pour conclure , une modélisation d'une telle grille se fera en créant une matrice de listes à 4 éléments (chaque élément de la matrice est une case et chaque élément de cette case(liste) est une épaisseur d'un mur adjacent).
La case située en haut à gauche correspond à l'entrée de la grille, et celle située en bas à droite est la sortie.

Le code est expliqué à travers les commentaires.

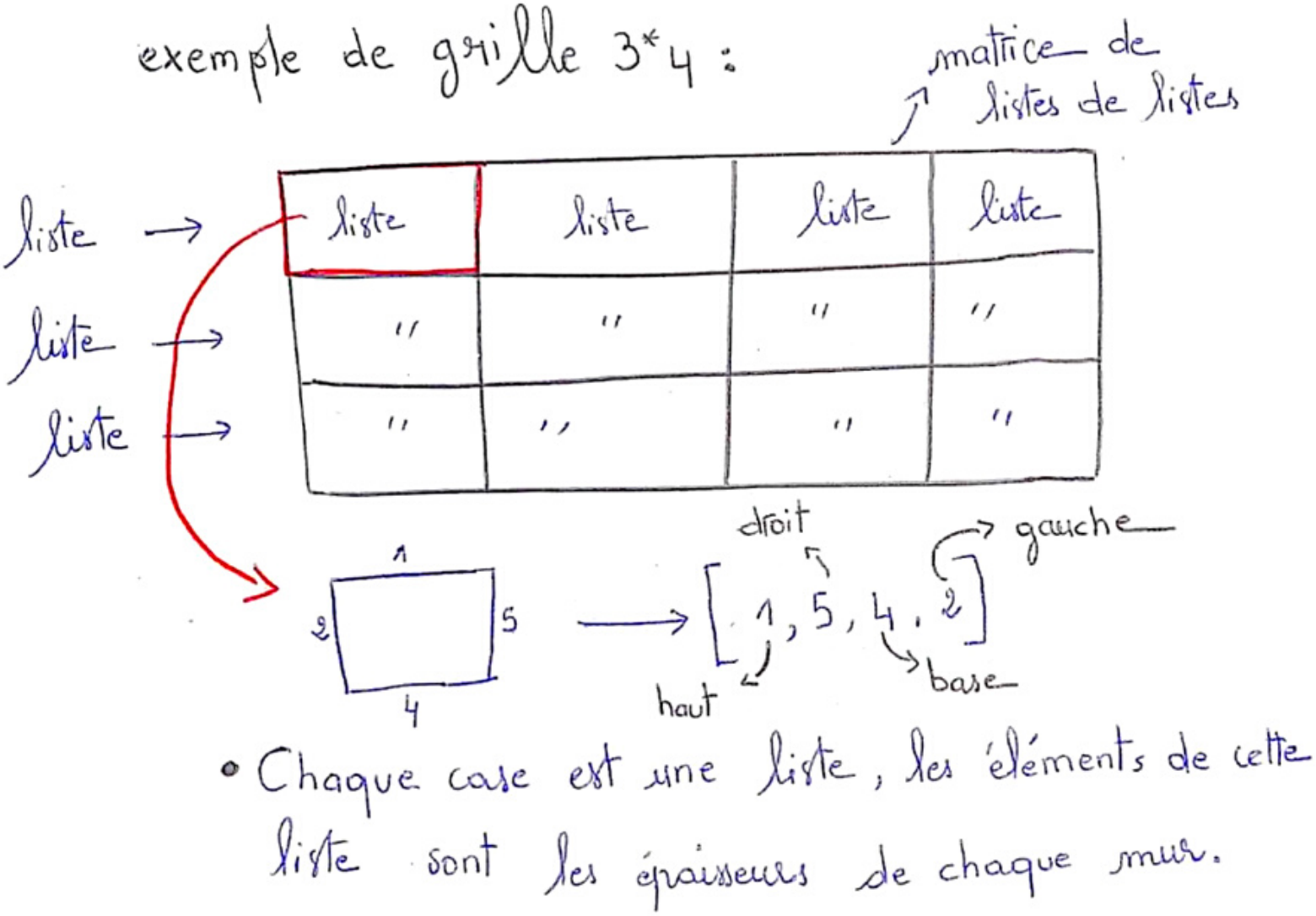
Comment afficher la grille?

-Pour afficher la grille en faisant apparaitre l'épaisseur de chaque mur nous avons choisi d'utiliser le module graphique "turtle" qui permet de piloter un «crayon» afin de tracer dynamiquement des figures géométriques. Nous avons donc parcouru la matrice afin de traiter chaque cellule de la grille et chaque mur de la cellule.

-L'idée est de dessiner un carré et changer l'épaisseur du crayon avant de passer au côté suivant . Cette épaisseur n'est rien d'autre que les éléments de M[i][j][k] avec k entra 0 et 3, les 4 éléments désignent respectivement les quatre épaisseurs des murs de la cellule . Nous avons pris en compte le chemin du crayon de la tortue , à chaque changement d'épaisseur (pensize) la tortue avance de 30 pixels et tourne à droite et ce , jusqu'à revenir à la case de départ cela veut dire que le premier mur dessiné de chaque cellule est celui du haut , le prochain est celui d'à droite puis d'en bas et enfin le mur à gauche .

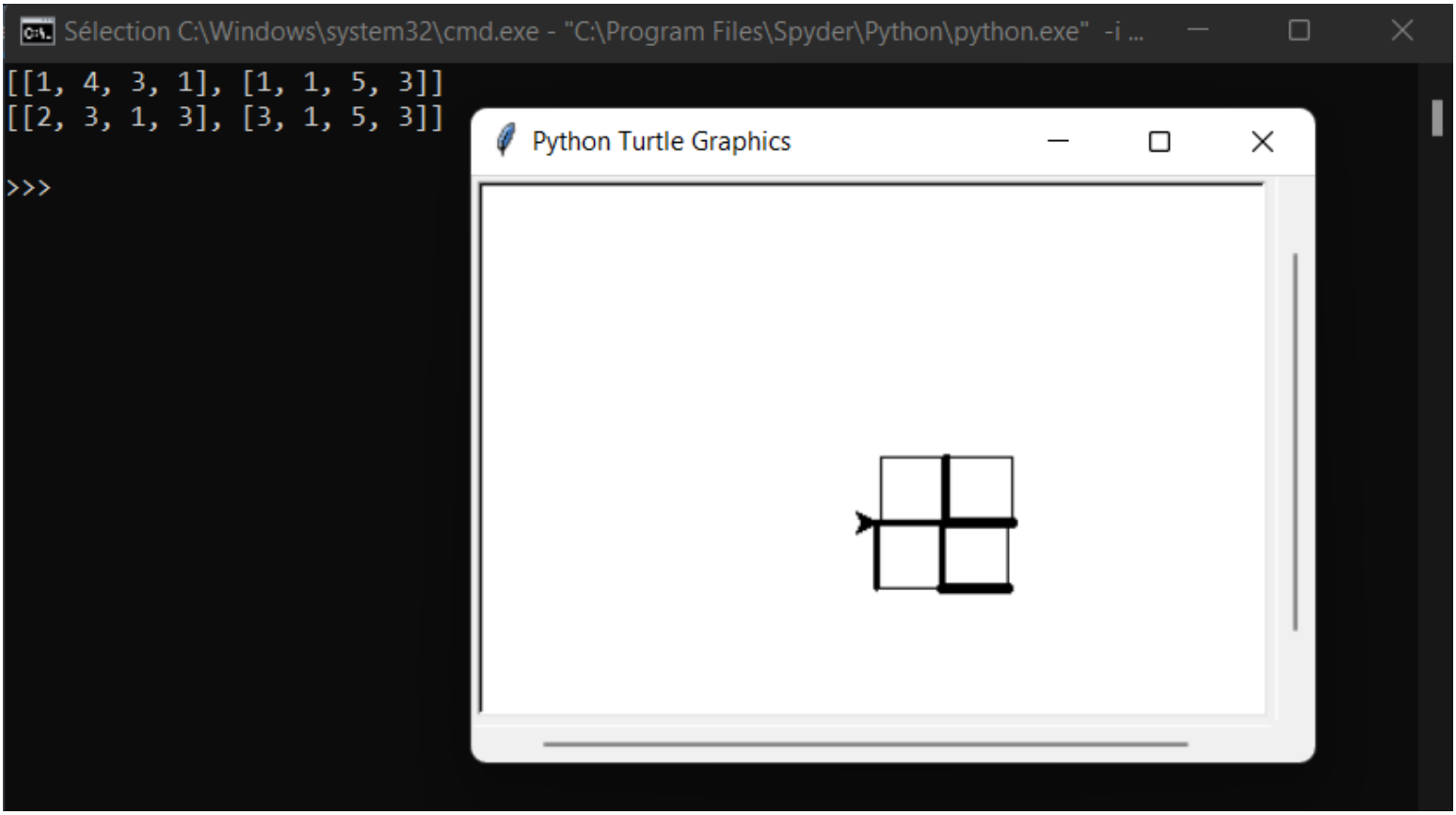
- Pour passer d'un mur à l'autre , la tortue fait une rotation de 90 degrés afin de créer un angle droit qui désigne le coté d'un carré.Et pour passer d'une épaisseur à l'autre , on traite un à un les éléments k entre 0 et 3 stockés dans la liste qui désigne la case en cours de dessin . A la fin de chaque cellule dessinée on passe à la liste suivante de la première ligne de la matrice , une fois la dernière liste de chaque liste (la dernière colonne de la ligne en question) dessinée nous passons automatiquement à la ligne suivante jusqu'à atteindre la dernière ligne .Nous avons choisi de ne pas laisser d'espace entre les cases car c'est une grille , donc les murs sont adjacents .

-Lors du passage d'une cellule à l'autre , la tortue doit changer de position sinon la grille se dessinera sous forme de carrés superposés . Pour cela, à la fin de chaque cellule de la même ligne , on récupère l'abscisse de la position de la tortue et on y rajoute la longueur d'un mur , ainsi , la tortue pourra passer au dessin de la cellule suivante à partir de l'extrémité droite du mur en haut de la cellule précédente. Cependant , dans le cas où on est arrivé au bout de la ligne , pour passer à la ligne suivante il faut modifier l'abscisse et l'ordonné de la tortue. Pour cela , on descend d'un mur (30 pixels) au niveau de l'axe des ordonnées , et on recule d'une ligne (la variable 'c' qui dépendra du nombre de colonnes saisies par l'utilisateur lors de la création de la grille) au niveau de l'axe des abscisses.



- Chaque case est une liste , les éléments de cette liste sont les épaisseurs de chaque mur.

Test sur une grille 2*2 :



Partie2:

-La structure de données associée à l'ensemble des chemins passant par toutes les cellules d'une telle grille et ce , en tenant compte de l'épaisseur de chaque mur est : les graphes orientés pondérés . Pourquoi? Car ceci est un problème associé à des parcours ou plutôt des successions d'actions. Et pour cela on fait souvent appel à la notion de chemin. L'épaisseur de murs sera considérée comme une arête ou un arc qui sera lui associé à un poids , dans ce cas : la valeur de l'épaisseur du mur .

-Afin de présenter informatiquement cette structure , on utilisera les listes d'adjacence de ses sommets, avec les poids des arêtes (murs) . Ceci nous permettra de stocker pour chaque sommet la liste de ses voisins , et ainsi , on pourra visualiser le graphe . C'est donc comme vu au cours , une représentation économique en mémoire . Les représentations de graphes par liste d'adjacence privilégient une approche plus centrée sur les sommets.

-Pour ceci on a commencé par définir une fonction ‘cases_voisines’ qui vas renvoyer les couples d’indices des cases voisines .Deux cases sont dites voisines si une des cases est située immédiatement au dessus, en dessous, à gauche, ou à droite de l’autre . De plus , chaque case de la grille est associée à un sommet et il y un arc entre deux sommet (le mur) si ces cases sont voisines . Ensuite , nous avons définit une fonction ‘ matrice_en_graphe’ qui va tout simplement convertir notre matrice en un dictionnaire qui représentera le graphe . Cette fonction prend en entrée la matrice en question puis renvoie la liste d’adjacence du graphe pondéré associé sous forme de dictionnaire .

-L’algorithme qui calculera le parcours de cette structure permettant , à partir d'une grille, de représenter cette structure est l’algorithme de Dijkstra qui fournit tous les chemins de coût minimal partant d’un sommet donné s et rejoignant les différents sommets du graphe (accessibles depuis s qui est dans ce cas le point qui relie les deux murs haut et gauche de la case en première ligne et première colonne) .

-Ceci est un algorithme glouton qui , à chaque étape, sélectionne parmi les sommets non encore traités celui qui “coûte le moins cher” ou alors le mur le plus fin . On utilisera donc une liste des distances minimales pour atteindre chaque sommet depuis s . On aura aussi la possibilité de mémoriser pour tout sommet son prédécesseur qui permettra d’obtenir le chemin de cout minimal voulu .

-Pour cela , on définit dans un premier temps la fonction minimum calculant le minimum des valeurs de ce dictionnaire, et renvoyant la clé correspondante puis la fonction “dijkstra_pred” qui parcourt les voisins de k le sommet de distance minimale (grâce à la fonction minimum) puis vérifie si v n’a pas déjà été traité et si c’est le plus court chemin afin de le stocker dans la liste des prédécesseurs . Cela nous renvoi une liste de prédécesseurs qu’on va utiliser dans la fonction finale ‘ chemin_min ‘ :

Comment ça marche ?

1. On choisit le sommet accessible de distance minimale comme sommet à explorer.
2. A partir de ce sommet, on explore ses voisins et on met à jour les distances pour chacun. On ne met à jour la distance que si elle est inférieure à celle que l’on avait auparavant
3. On répète jusqu’à ce qu’on arrive au point d’arrivée ou jusqu’à ce que tous les sommets aient été explorés.

-Pour conclure cette fonction nous permet à partir d'une matrice , de calculer et afficher son poids minimal , ainsi que le chemin correspondant reliant la case en haut à gauche à la case en bas à droite sous forme de liste de couples comportant les coordonnées des cellules traversées , dans l'ordre .

Partie 3 :

-Finalement, pour enrichir notre modélisation de la grille des cellules et pour pouvoir percer des murs afin d'obtenir le chemin voulu , nous allons combiner les deux parties précédentes. En fonction de la liste de couples renvoyée par la fonction ' chemin_min' nous allons détecter les murs percés et les grilles traversée et ainsi changer leur couleur au fur et à mesure du dessin pour visualiser la grille initiale (avec l'épaisseur des murs), la grille "percée", le chemin associé, et son "coût".
Pour cela , nous allons vérifier au fur et à mesure si le couple dans la liste du chemin générée par la fonction 'chemin_min' coïncide avec le couple d'indice des extrémités d'un mur , si c'est le cas , on dessine le mur comme sur la fonction ' afficher-dessin' mais cette fois avec une couleur .

-On passe on deuxième couple de la liste et ainsi on aura un chemin au sain de la grille de base colorié et qui nous montrera dans ce cas le chemin le plus court et donc le moins lourd en ce qui concerne l'épaisseur des murs (plus la valeur de l'épaisseur est grande plus le poids en passant par ce mur est lourd) .

-De plus , comme demandé au sujet , nous allons retourner la grille de base grâce à la fonction ‘afficher_dessin_perce’ puis le chemin et son cout grace à la fonction ‘chemin_min’ .

BONUS :

Pour optimiser notre algorithme nous avons opté pour la fonction djikstra_pred qui stocke simultanément les prédécesseurs puis les retourne sous forme de liste à la fin . Nous pouvons aussi essayer d’ignorer tous les poids depuis la source qui ne nous intéressent pas et plutôt se focaliser surtout sur le chemin de poids minimal entre deux sommets du graphe . Finalement , pour optimiser l’algorithme de Dijkstra, on peut s’arrêter dès que le sommet t est traité car on sait que d[t] n’évoluera plus .

merci