

Grand Devoir 2

Structures de données et algorithmes

Graphes et Arbres

Mentions générales

- Pour ce projet vous allez travailler **par équipes de 2 personnes (ou seuls ☺)**.
 - Le projet est à rendre **sur la plateforme Moodle** (curs.upb.ro). S'il y a des problèmes lors du téléchargement vers ou depuis la plateforme, contactez par Teams ou par mail votre assistant des travaux pratiques: Iulia Stanica (iulia.stanica@gmail.com) ou Mara Chirascu (marachirascu@gmail.com)
 - Le projet est à rendre au plus tard le **18.05.2023, à 08:00**. Aucun retard ne sera accepté.
 - Vous recevrez des questions concernant votre solution durant la séance suivante du TP. Des devoirs qui **ne seront pas présentés au TP suivant ne seront pas notés!**
 - La rendue finale du projet comportera une archive nommée Etudiant1Nom_Etudiant1Prenom_Etudiant2Nom_Etudiant2Prenom_HW1 avec:
 - o **les fichiers du code source (.cpp et .h)** et non pas des fichiers objets (i.e. *.o), des fichiers exécutables (i.e. *.exe) ou des fichiers codeblocks (i.e. *.cbp); **SVP mettez chaque exercice dans un dossier séparé !**
 - o **un fichier de type README** où vous allez spécifier (dans quelques mots) votre approche, avec des instructions pour exécuter les exercices. Au contraire, s'il y a des exigences qui ne sont pas fonctionnelles, vous pouvez proposer des idées pour une solution possible que vous allez expliquer en classe aussi (et peut-être obtenir des points supplémentaires).
 - Pour toute question concernant le sujet du projet ou les exigences, envoyer un mail/message sur Teams à vos assistants ou **utiliser le canal de SDA de Teams**. N'oubliez pas de mentionner le nom de votre prof @NomProf ou de notifier tout le monde @General.
 - Attention : notre équipe utilise des logiciels détectant le plagiat (Moss du Stanford). Dans le cas malheureux de **plagiat, le projet sera noté par un 0 (zéro)**.
- ! Remarque : Vous pouvez utiliser les graphes et les arbres qu'on a fait ensemble en classe (seront représentés par des fichiers en-tête (headers), en utilisant des chablon (<template>)). Comme alternative, vous pouvez utiliser les implémentations standard du C++ POUR pile ou file d'attente, mais PAS d'autres implémentations prises de l'Internet.

1. Graphes (4p)

Sydney Fox (professeur d'Histoires Anciennes) et son assistant, Nigel Bailey, ont trouvé un livre écrit dans une langue inconnue, mais qui utilise les mêmes lettres que l'alphabet anglais – pas de majuscules, seulement minuscules (e.g. a, b, c....). Le livre contient un bref index, mais l'ordre des mots d'index est différent de l'alphabet anglais. Les chasseurs au trésor ont ensuite tenté d'utiliser cet index pour déterminer l'ordre des caractères dans la langue inconnue, mais ont échoué.

Écrivez un programme en C++ pour aider les chasseurs au trésor afin de déterminer l'ordre des caractères dans la langue inconnue. Vous devez utiliser le tri topologique et les **graphes**. Affichez le graphe construit utilisé pour résoudre le problème.

Les données d'entrée: index.in

Dans le fichier «index.in» il y a au moins 1 et au plus 50 mots suivie d'une ligne contenant le caractère "." (point). Chaque mot a au plus 10 caractères. Les mots d'un index contiennent uniquement les caractères minuscules de l'alphabet anglais et apparaissent dans l'ordre croissant de l'alphabet inconnu. Pas nécessairement toutes les minuscules apparaissent dans l'index, mais un index se traduira par une séquence unique de caractères qui apparaissent.

Les données de sortie: index.out

Dans le fichier «index.out» on écrit la séquence unique de caractères qui apparaissent dans «index.in», dans l'ordre alphabétique correct de la langue inconnue.

Exemple:

index.in	graphe	index.out
ion ana adonia doina doinn ddan ddao .	<pre> graph TD i((i)) --> a((a)) a --> d((d)) a --> n((n)) d --> o((o)) n --> o </pre>	ianod
b .	<pre> graph TD b((b)) </pre>	b
xwy zx zxy zxw ywwx .	<pre> graph LR x((x)) --> z((z)) z --> y((y)) y --> w((w)) </pre>	xzyw

Obs: vous pouvez utiliser l'entrée et la sortie classiques (cin, cout) a la place des fichiers.

Points:

0.5p entree

1p construction correcte graphe + son affichage

2p tri topologique

0.5p sortie correcte + validations

References utiles:

fclose: <http://www.cplusplus.com/reference/cstdio/fclose/>

fopen: <http://www.cplusplus.com/reference/cstdio/fopen/>

fprintf: <http://www.cplusplus.com/reference/cstdio/fprintf/>

fscanf: <http://www.cplusplus.com/reference/cstdio/fscanf/>

<http://www.cplusplus.com/doc/tutorial/files/>

<http://www.cs.washington.edu/education/courses/cse373/02au/lectures/lecture19l.pdf>

2. Arbres (6p)

Un arbre Quad est fréquemment utilisé dans le traitement d'image. Chaque image peut être divisée en quatre quadrants. Chaque quadrant peut également être divisé en quatre autres quadrants, etc. Dans un arbre Quad, l'image est représentée comme un nœud parent, tandis que les quadrants sont représentés comme quatre nœuds enfants. Si l'image entière n'a qu'une seule couleur, alors l'arbre quadratique ne contient qu'un seul nœud. Un quadrant est divisé seulement s'il contient des pixels de différentes couleurs. Cela signifie que l'arbre peut ne pas avoir une forme équilibrée.

Un designer graphique travaille avec des images de $32 * 32$ unités, cela signifie 1024 pixels/image. Une des opérations qu'il doit effectuer est de superposer deux images, pour obtenir une nouvelle image en noir et blanc. Dans la nouvelle image, un pixel est noir s'il est noir dans au moins une des deux images initiales, sinon il est blanc. Il veut savoir combien de pixels noirs il aura dans la nouvelle image, avant de faire l'opération de superposition, car les images avec trop de pixels noirs sont coûteuses à imprimer. Vous devez l'aider en écrivant un programme qui calcule combien de pixels noirs l'image finale aura. Malheureusement, il ne fait confiance à personne, donc il ne veut pas vous donner les images initiales, seulement les représentations Quad de celles-ci.

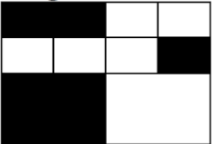

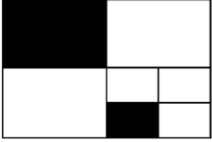
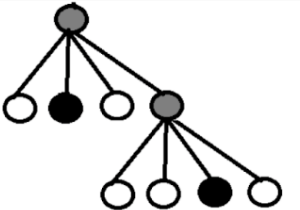
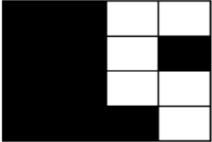
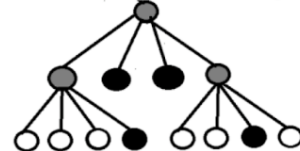
Entrée pour votre programme : une paire de chaînes de caractères représentant les deux images. Ainsi, une image est représentée par le parcours en préordre de son arbre Quad : la lettre "p" signifie un nœud parent, la lettre "b" signifie un nœud enfant pour un quadrant noir (black), la lettre "w" signifie un nœud enfant pour un quadrant blanc (white). La paire de chaînes peut être lue en utilisant la console ou d'un fichier.

Sortie pour votre programme : parcours en préordre de l'arbre Quad de l'image finale et du nombre de pixels noirs qu'elle contient, en fonction des pixels noirs contenus par chaque image initiale. Faites attention aux pixels noirs qui se chevauchent des images initiales.

Explications :

Ordre de représentation des quadrants :

2	1
3	4

Image 1: 	Quad tree for Image 1: 	Preorder representation of image 1: ppwwwbpbwbwbw	Nr of pixels in image 1: 448
Image 2: 	Quad tree for Image 2: 	image 2: pwbwpwbw	320
Final image: 	Quad tree for final image: 	Preorder representation of final image: ppwwwbbbpwbw	Nr of pixels in final image: 640

Tâches :

- (1p) Trouvez une représentation appropriée pour un arbre Quad.
- (1p) Écrivez une fonction qui transforme le parcours en préordre que vous donnez comme entrée dans la représentation que vous avez choisie pour les arbres Quad.
- (1.5p) Écrire une fonction qui reçoit comme arguments deux arbres Quad et les transforme dans un autre arbre Quad, pour l'image finale, basé sur l'algorithme décrit dans le problème.
- (1p) Écrire une fonction qui calcule les pixels pour un arbre Quad donné représentant une image.
- (1p) Testez votre programme avec différentes entrées. Affichez la représentation finale de l'arbre Quad et le nombre de pixels.
- (0.5) Créer un menu d'interaction qui permet à l'utilisateur de tester les fonctionnalités ci-dessus (a-e). Le programme doit permettre de tester plusieurs fonctionnalités dans la même exécution.