Algorithmique et Base de la programmation

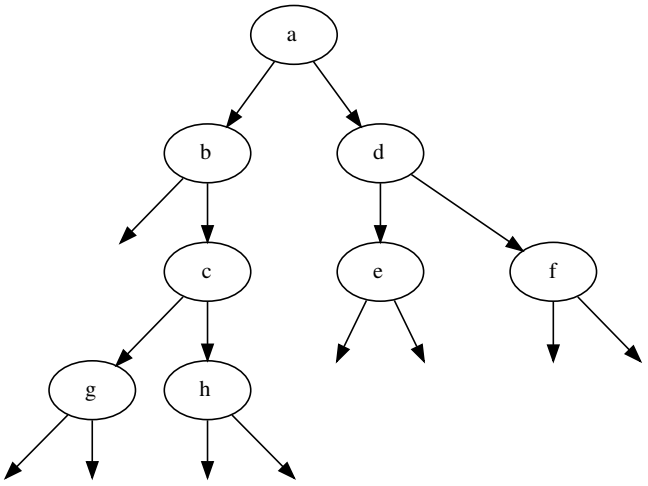
Durée : *3h00*
Documents autorisés : **AUCUN**

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.

1 Arbre binaire et parcours (3 points)

1.1 Parcours d'un arbre binaire

Donnez les parcours RGD, GRD et GDR de l'arbre suivant :



1.2 Construction d'un arbre binaire à partir de deux parcours

Il existe un théorème qui indique qu'un arbre binaire peut être entièrement défini à partir de deux parcours.

Dessinez l'arbre binaire dont :
- le parcours RGD est a, c, d, h, f, g, b, e
- le parcours GRD est h, d, c, f, g, a, b, e

2 Compréhension d'un algorithme sur les graphes (4 points)

L'algorithme de Floyd est un algorithme qui permet de calculer la longueur du plus court chemin entre tous les nœuds d'un graphe orienté valué positivement.

« L'algorithme repose sur la remarque suivante : si $(a_0, \ldots, a_i, \ldots, a_p)$ est un plus court chemin de a_0 à a_p , alors $(a_0, ..., a_i)$ est un plus court chemin de a_0 à a_i , et $(a_i, ..., a_p)$ un plus court chemin

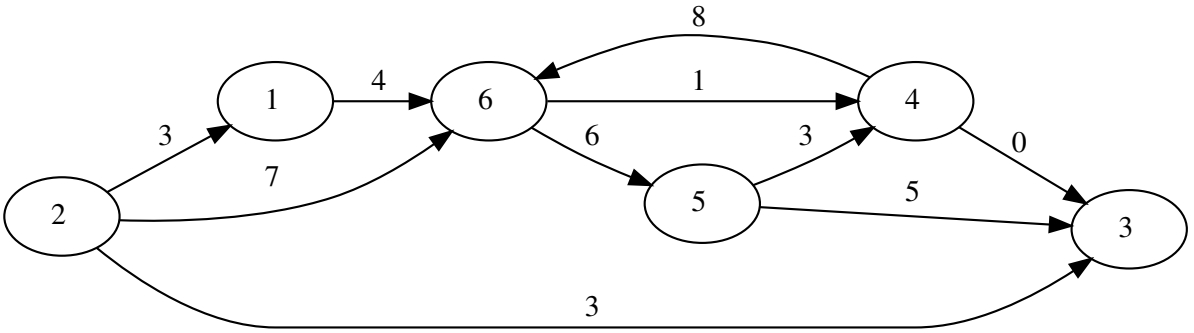


FIGURE 1 – Un graphe valué

de a_i à a_p . De plus, comme les arêtes sont valuées positivement, tout chemin contenant un cycle est nécessairement plus long que le même chemin sans le cycle, si bien qu'on peut se limiter à la recherche de plus courts chemins passant par des sommets deux à deux distincts.

Floyd montre donc qu'il suffit de calculer la suite de matrices définies par :

$$M_{i,j}^k = min(M_{i,j}^{k-1}, M_{i,k}^{k-1} + M_{k,j}^{k-1}). \gg [1]$$

tel que M^0 est la matrice d'adjacence du graphe avec :
- les nœuds qui sont numérotés de 1 à n (et k varie de 1 à n) ;
- $M_{i,i}^0 = 0$;
- $M_{i,j}^0 = +\infty$ s'il n'existe pas d'arc reliant i à j.

Questions

1. Donnez la matrice d'adjacence M^0 du graphe proposé par la figure 1 (pour plus de clarté, vous pouvez ne pas noter les $+\infty$).
2. Donnez les matrices M de Floyd pour k variant de 1 à 6.
3. À partir de la matrice M^6 donnez la longueur du plus court chemin reliant le nœud 2 au nœud 4.

3 Liste ordonnée (5 points)

Pour rappel, le TAD `ListeOrdonnee` est défini de la façon suivante :

Nom:	ListeOrdonnee	
Paramètre:	Element (possédant une relation d'ordre total, donc l'opération $<$)	
Utilise:	**Booleen**, **Naturel**	
Opérations:	listeOrdonnee:	$\rightarrow$ ListeOrdonnee
	estVide:	ListeOrdonnee $\rightarrow$ **Booleen**
	inserer:	ListeOrdonnee $\times$ Element $\rightarrow$ ListeOrdonnee
	supprimer:	ListeOrdonnee $\times$ **Naturel** $\nrightarrow$ ListeOrdonnee
	obtenirElement:	ListeOrdonnee $\times$ **Naturel** $\nrightarrow$ Element
	longueur:	ListeOrdonnee $\rightarrow$ **Naturel**
Préconditions:	supprimer(l,i):	$0 < i \leq longueur(l)$
	obtenirElement(l,i):	$0 < i \leq longueur(l)$

1. `http://www.nimbustier.net/publications/djikstra/floyd.html`

3.1 Conception préliminaire

Donnez les signatures des fonctions et procédures correspondant à ces opérations.

3.2 Utilisation

Donnez le corps de la fonction suivante qui permet de fusionner deux listes ordonnées :
– **fonction** fusionner (l1,l2 : ListeOrdonnee) **:** ListeOrdonnee

3.3 Conception détaillée

On se propose de concevoir ce TAD de la façon suivante :
Type ListeOrdonnee = **Structure**
elements : ListeChainee
nbElements : **Naturel**
finstructure

Après avoir rappelé les signatures des fonctions et procédures permettant de manipuler des variables de type `ListeChainee` vu en cours, donnez le corps des fonctions ou procédures correspondant aux opérations `listeOrdonnee` et `inserer`.

4 Comment retrouver son chemin ? (8 points)

L'objectif de cet exercice est d'étudier le problème du labyrinthe. Comme l'indique la figure 2, l'objectif est de trouver un algorithme permettant de trouver le chemin qui mène de l'entrée à la sortie.

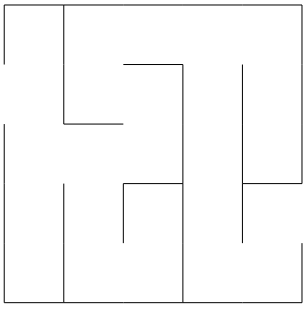


FIGURE 2 – Un labytinthe

4.1 Partie publique

En fait, un labyrinthe est composé de cases. On accède à une case à partir d'une case et d'une direction. Les directions possibles sont `Nord` (vers le haut), `Sud` (vers le bas), `Est` (vers la droite) et `Ouest` (vers la gauche).

Par exemple, comme le montre la figure 3 le labyrinthe précédent peut être considéré comme étant composé de 25 cases. La case numéro 6 est la case d'entrée. La case 20 est la case de sortie. La case 8 est accessible depuis la case 13 avec la direction `Nord`.

4.1.1 Le TAD labyrinthe

Les opérations disponibles sur un labyrinthe sont les suivantes :
– créer un labyrinthe,
– obtenir la case d'entrée,

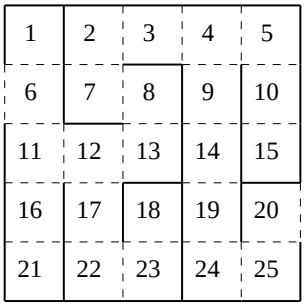


FIGURE 3 – Un labytinthe composé de cases

- savoir si une case est la case de sortie,
- obtenir une liste de directions possibles depuis une case donnée,
- obtenir la case accessible depuis une case avec une direction.

1. Donnez le type `Direction`
2. Donnez les TAD `Labyrinthe` et `Case` (TAD liés donc définis en même temps comme nous l'avons vu en TD)

4.1.2 Algorithme du petit-poucet

Une solution pour trouver la sortie est d'utiliser le principe du petit poucet, c'est-à-dire mettre un caillou sur les cases rencontrées.

Pour ne pas modifier le TAD `Labyrinthe`, plutôt que de marquer une case avec un caillou on peut ajouter une case à un ensemble. Pour vérifier si on a déja rencontré une case, il suffit alors de vérifier si la case est présente dans l'ensemble.

Après avoir rappelé les fonctions et procédures permettant de manipuler des variables de type `Ensemble` vu en cours, proposez le corps de la procédure suivante qui permet de trouver le chemin de sortie (s'il existe) à partir d'une case donnée [2] :
procédure calculerCheminDeSortie **(E** l : Labyrinthe, caseCourante : Case, **E/S** casesVisitees : Ensemble<Case>, **S** permetDAllerJusquALaSortie : **Booleen**, lesDirectionsASuivre : ListeChainee <Direction>**)**

4.2 Partie privée

On peut représenter un labyrinthe à l'aide d'un graphe étiqueté (label associé à chaque nœud) et valué (label associé à chaque arc). On considère dans ce cas que les étiquettes des nœuds du graphe sont les cases du labyrinthe et les arcs valués par les directions.

Dessinez le graphe associé à l'exemple de la figure 4.

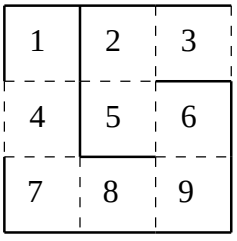


FIGURE 4 – Un labytinthe composé de 9 cases

2. Vous pouvez vous inspirer de la procédure *remplir* vu dans le dernier cours