Examen final

Durée : 1 heure 30.

Avant de commencer

Votre copie doit être **anonyme**. Faites donc attention à bien cacheter vos copies en repliant le coin et en le collant, et à ne reporter que votre **numéro d'anonymat** fourni sur la première copie sur les intercallaires. Si vous cachetez votre copie avec de la colle, n'en mettez **que sur le pourtour du coin** et pas sur votre nom ou sur la portion du coin qui se repliera dessus, sinon il nous sera difficile de vous attribuer votre travail après la correction lors de la désanonymisation des copies.

Toutes vos réponses nécessitent une justification. Même lorsque ce n'est pas précisé, ne répondez jamais simplement « oui » ou « non » sans explication, car vous auriez alors très bien pu répondre au hasard. Vous ne recevrez donc pas les points des questions avec une telle réponse. Ce sujet comporte deux problèmes totalement indépendants, et au sein de chaque problème des questions peuvent être indépendantes. Prenez donc le temps de lire le sujet et de choisir les questions que vous abordez pour ne pas rester coincé.

1 Système de contraintes différentielles

Cette partie vise à appliquer les algorithmes de recherche de plus courts chemins à un problème concret de plannification d'un ensemble de tâches. Cet exercice est fortement inspiré de la section 24.4 du livre "Algorithmique" (3ème édition) de Cormen, Leiserson, Rivest et Stein.

1.1 Formalisation

Étant donné un ensemble de variables $\{x_k\}_{k=1}^n$, un système de contraintes différentielles est un ensemble d'inégalités du type $x_j - x_i \leq a_{i,j}$ où $a_{i,j}$ est une constante ne dépendant pas des variables. Par exemple :

$$\begin{cases} x_1 - x_2 \leq 0 & x_1 - x_5 \leq -1 & x_2 - x_5 \leq 1 & x_3 - x_1 \leq 5 \\ x_4 - x_1 \leq 4 & x_4 - x_3 \leq -1 & x_5 - x_3 \leq -3 & x_5 - x_4 \leq -3 \end{cases}$$

La résolution de cet ensemble de contraintes consiste à trouver un ensemble de valeurs, une pour chaque x_k, telles que toutes les inégalités soient respectées en même temps. Notez que la solution n'est pas unique, car il suffit par exemple d'ajouter à chaque valeur d'une solution une constante pour obtenir une autre solution.

1.2 Graphe des contraintes

Une méthode pour résoudre ce type de systèmes consiste à créer un graphe à partir du système, et d'en chercher les plus courts chemins. Nous allons montrer que les distances obtenues par les plus courts chemins fournissent une solution au problème.

Pour chaque variable x_k, un sommet v_k correspondant est créé dans le graphe. Pour chaque contrainte $x_j - x_i \leq a_{i,j}$, une arête orientée de v_i vers v_j (attention à l'ordre) est créée, avec pour poids $a_{i,j}$. En plus de ces sommets et arêtes, un sommet supplémentaire v_0 est ajouté, et pour chaque sommet v_k, une arête orientée de v_0 vers v_k est créée avec le poids 0.

Question 1 – Dessinez le graphe correspondant au système différentiel fourni en exemple en section 1.1.

1.3 Plus courts chemins et solution du système

Notons $w(v_i, v_j)$ le poids de l'arête allant de v_i à v_j dans le graphe, et $\ell(v_i, v_j)$ la longueur du plus court chemin (quand il existe) pour aller de v_i à v_j. Nous allons montrer qu'en associant à chaque variable x_k la longueur $\ell(v_0, v_k)$ du plus court chemin de v_0 à v_k quand il existe, nous obtenons une solution du système de contraintes différentielles associé au graphe.

Question 2 – Soit une arête allant de v_i à v_j, de poids $w(v_i, v_j)$, correspondant à la contrainte différentielle $x_j - x_i \leq w(v_i, v_j)$. En utilisant le fait que $\ell(v_0, v_i)$ et $\ell(v_0, v_j)$ sont des longueurs de **plus courts chemins**, montrez qu'avec $x_i = \ell(v_0, v_i)$ et $x_j = \ell(v_0, v_j)$ la contrainte différentielle est vérifiée.

Question 3 – Dans quelles circonstances est-ce que le plus court chemin de v_0 à v_k pourrait ne pas exister ? Fournissez un petit graphe exemple, et le système de contraintes correspondant.

Question 4 – Montrez que lorsqu'un plus court chemin n'existe pas, le système n'a pas de solution. Cette question est un peu plus difficile, n'est pas utile pour la suite, et n'est donc à traiter que si vous voyez immédiatement la preuve ou avez le temps de la traiter.

1.4 Dijkstra modifié

L'algorithme de Dijkstra basique n'est pas en capacité de gérer des arêtes de poids négatif, car il n'autorise pas un sommet sorti de la file à priorité à y retourner si une arête menant à ce sommet permet de diminuer la longueur du chemin connu pour ce sommet.

Question 5 – Pouvez-vous fournir un exemple de graphe avec des poids négatifs dans lequel les plus courts chemins existent, mais Dijkstra ne fournit pas le bon résultat ?

Question 6 – En autorisant les sommets à retourner dans la file, utilisez l'algorithme de Dijkstra pour déterminer les plus courts chemins dans le graphe fourni en question 1. Il n'est pas demandé ici uniquement les plus courts chemins, mais surtout de détailler les étapes de l'algorithme pour les obtenir, par exemple via un tableau comme vu en cours.

Question 7 – Vérifiez que ces plus courts chemins vous fournissent bien une solution au système différentiel initial.

1.5 Application à un problème pratique

Une personne souhaite rénover une vieille maison. Elle doit niveler le sol par une chappe en béton, refaire la peinture et l'électricité, et poser un escalier. La remise à niveau du sol prend une journée. La chappe doit durcir au moins 10 jours avant de pouvoir circuler dessus pour pouvoir faire l'électricité ou la peinture, et au moins 20 jours avant de pouvoir poser l'escalier. La peinture prend trois jours, ne peut être faite qu'après l'électricité, et nécessite un échafaudage. L'électricité prend trois jours et nécessite un échafaudage ainsi que l'expertise d'une belle mère bricoleuse. La pose de l'escalier prend deux jours et nécessite aussi l'expertise de la belle mère bricoleuse. Une association de quartier peut prêter un échafaudage, pour une durée d'une semaine maximum. La belle mère bricoleuse peut poser ses congés quand elle le souhaite, mais pas plus d'une semaine.

Dans la suite vous noterez
- t_{ec} la date d'emprunt de l'échafaudage ;
- t_{bm} la date du début des congés de la belle mère bricoleuse ;
- t_{el} la date du début de l'électricité ;
- t_p la date du début de la peinture ;
- t_{es} la date du début de la pose de l'escalier ;
- t_n la date du nivellement du sol par une chappe.

Question 8 – Exprimez sous la forme d'équations différentielles du type $\ldots - \ldots \leq \ldots$ sur t_a et t_b les contraintes du type « la tâche A dépend de la tâche B qui prend ... jours ».

Question 9 – Exprimez sous la même forme les contraintes du type « la tâche A dépend de la ressource B qui n'est disponible que pendant ... jours à partir de la date t_b ».

2 Test d'intersection d'un ensemble de segments

Cette partie étudie un algorithme permettant de déterminer à partir d'un ensemble de segments si deux d'entre eux se coupent. Cet algorithme ne liste pas toutes les intersections, mais se contente de répondre vrai ou faux selon qu'une intersection existe ou non. Votre rôle consistera à comprendre l'algorithme proposé puis à utiliser les structures de données que vous connaissez pour en proposer une adaptée à ce problème, et à déterminer la complexité de l'algorithme utilisant la structure choisie. Vous n'avez pas à prouver la correction de l'algorithme. Si cette preuve vous intéresse elle est disponible dans la section 33.2 du livre "Algorithmique" (3ème édition) de Cormen, Leiserson, Rivest et Stein.

2.1 Spécifications

Vous travaillez en deux dimensions : abscisse (x, horizontal) et ordonnée (y, vertical). Vous supposerez que deux extrémités de segments (même différents) n'ont jamais la même abscisse, et qu'une extrémité de segment n'est jamais positionnée exactement sur un autre segment. Il est ainsi possible de parler de l'extrémité gauche et de l'extrémité droite d'un segment. Étant donné un segment s, vous pouvez utiliser $\mathsf{s}.\mathit{gauche}$ et $\mathsf{s}.\mathit{droite}$ pour obtenir ces extrémités. Il est également possible de trier les extrémités de tous les segments de la plus à gauche à la plus à droite.

Vous disposez d'une fonction $\mathsf{intersection}(\mathsf{s}_1, \mathsf{s}_2)$ pour déterminer si deux segments s_1 et s_2 se coupent. Cette fonction est de complexité $\Theta(1)$ en temps et en espace mémoire.

2.2 Algorithme naïf

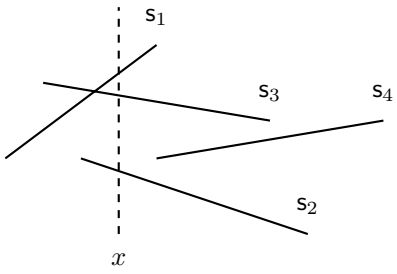
Question 10 – Écrivez un algorithme naïf prenant en paramètre un ensemble de n segments dans un tableau t, et indiquant si deux d'entre eux (au moins) se croisent. L'idée n'est pas ici de chercher un algorithme performant, mais simplement d'avoir un algorithme qui donne la bonne réponse. Détaillez la complexité de cet algorithme dans le meilleur et le pire cas. Votre complexité devra être cohérente avec l'algorithme que vous proposerez.

2.3 Algorithme élaboré

Voici un algorithme plus élaboré permettant de répondre au problème. Cet algorithme utilise une structure de données A pour organiser les segments en cours de traitement. C'est cette structure de données sur laquelle vous allez travailler.

L'algorithme fonctionne par balayage de gauche à droite il traite une par une toutes les extrémités des segments dans ce sens. Lorsqu'une extrémité gauche est rencontrée, le segment est inséré dans la structure de données. Lorsqu'une extrémité droite est rencontrée, le segment est retiré de la structure de données. En plus de l'insertion et du retrait, la structure de données permet de répondre aux requêtes dessus et $\mathsf{dessous}$.

En toute abscisse x, il est possible de tracer une droite verticale, et de comparer les uns avec les autres tous les segments qui intersectent cette droite : un segment s_1 est *au dessus* d'un autre segment s_2 si son intersection avec la droite verticale d'abscisse x est au dessus de celle de s_2. Nous noterons cette comparaison $<_x$, et par extension $<_\mathsf{e}$ lorsqu'elle est utilisée sur une droite verticale passant par une extrémité e. Sur le schéma ci-contre, au niveau de la droite verticale d'abscisse x, nous avons $\mathsf{s}_3 <_x \mathsf{s}_1$, $\mathsf{s}_2 <_x \mathsf{s}_1$ et $\mathsf{s}_2 <_x \mathsf{s}_3$. À cette abscisse, le segment s_4 n'est pas comparable avec les autres.



La requête dessus sur A renvoie le segment dans A qui est immédiatement au dessus du segment passé en paramètre selon la comparaison fournie en paramètre. De même la requête $\mathsf{dessous}$ fournit le segment immédiatement en dessous. Sur la figure précédente, $\mathsf{dessus}(\mathsf{A}, <_x, \mathsf{s}_2)$ donnera s_3, $\mathsf{dessous}(\mathsf{A}, <_x, \mathsf{s}_2)$ donnera $\emptyset$ et $\mathsf{dessous}(\mathsf{A}, <_x, \mathsf{s}_1)$ donnera s_3.

Les requêtes sur la structure de données A utilisent la comparaison $<_\mathsf{e}$. Même si au cours de l'algorithme cette comparaison change avec e, vous pouvez faire comme si elle ne changeait pas : si nous avons $\mathsf{s}_1 <_{\mathsf{e}_1} \mathsf{s}_2$ et $\mathsf{s}_2 <_{\mathsf{e}_2} \mathsf{s}_1$ à la fois pour deux segments s_1 et s_2 et deux extrémités e_1 et e_2 différentes, c'est que les deux segments s'intersectent, et l'algorithme s'arrête dès qu'une telle intersection est détectée. Vous pouvez donc considérer

que cette comparaison reste cohérente tout au long de la durée de l'algorithme et l'utiliser pour les structures de données que vous connaissez qui nécessitent des comparaisons.

Données : S un ensemble de segments
Sorties : vrai si deux segments de S se coupent, faux sinon
Algorithme

```
A ← ∅ // votre structure de données adaptée
E ← tableau de toutes les extrémités des segments de S
trier E par abscisse croissante (de gauche à droite)
pour chaque extrémité e de E de gauche à droite faire
    si e est l'extrémité gauche d'un segment s alors
        insérer(A, <_e, s)
        s_+ ← dessus(A, <_e, s)
        s_− ← dessous(A, <_e, s)
        si (s_+ ≠ ∅ et intersection(s, s_+)) ou (s_− ≠ ∅ et intersection(s, s_−)) alors
            retourner vrai
    si e est l'extrémité droite d'un segment s alors
        s_+ ← dessus(A, <_e, s)
        s_− ← dessous(A, <_e, s)
        si s_+ ≠ ∅ et s_− ≠ ∅ et intersection(s_+, s_−) alors
            retourner vrai
        retirer(A, <_e, s)
retourner faux
```

Les questions qui suivent sont interdépendantes et doivent être réfléchies dans leur ensemble. Il n'y a pas nécessairement une seule bonne réponse pour ces questions. Il est donc impératif que les complexités que vous détaillerez soient cohérentes avec la structure que vous proposez, et que vos réponses soient cohérentes entre elles. Il y aura des points de complexité pour une structure mal adaptée, mais dont l'étude de complexité est correcte, mais pas de points de complexité pour une étude fausse d'une structure, même bonne.

Question 11 – Proposez une structure de données adaptée pour prendre le rôle de A dans l'algorithme. La pertinence de la structure de données sera évaluée en fonction des complexité des différentes requêtes demandées et de celle de l'algorithme final dans les questions suivantes.

Question 12 – Proposez des algorithmes pour réaliser les requêtes dessus et dessous sur votre structure de données. Explicitez ensuite les complexités de vos algorithmes dans le meilleur et le pire cas.

Question 13 – Étudiez la complexité de l'algorithme ci-dessus utilisant votre structure de données.