INF5130 — Devoir 2

Samuele Giraudo

Session 2025-01

Déclaration. *J'affirme, pour ce devoir, n'avoir reçu aucune aide d'aucune forme que ce soit, incluant les aides par des personnes et par les outils génératifs d'intelligence artificielle. Ceci vaut pour toute la période du devoir, depuis sa date de publication initiale à sa date de rendu finale.*

Je comprends que si j'ai bénéficié de telles aides et que je remplis cette déclaration, elle devient fallacieuse. Cela peut entraîner des sanctions prévues en conformité avec ce qui sera décidé par le comité des infractions académiques.

Répondre « J'affirme que tout ceci est exact. » et signer par prénom, nom et code permanent. Si ceci manque, le devoir ne pourra pas être évalué.

Réponse —

Conventions générales. Tout tableau t de longueur n est indicé de 0 à $n-1$. La notation $t[i]$ désigne la case mémoire à l'indice i de t. La fonction Longueur renvoie la longueur du tableau en argument en temps constant. La fonction CréerTableau prend en entrée un entier naturel n et une valeur x, et renvoie un tableau de longueur n dont toutes les cases contiennent la valeur x. La complexité en temps de cette fonction dans le pire et meilleur cas est linéaire en n.

Nous adoptons toutes les conventions vues en cours, notamment pour ce qui concerne les graphes.

1 Algorithmes gloutons

1.1 Problème du voyageur de commerce

Question 1.1. *(4 points)* — Considérons le *problème du voyageur de commerce* selon la formulation suivante (attention, il diffère légèrement de sa présentation vue en cours). Nous disposons de $n \geq 1$ villes numérotées de 0 à $n-1$ et d'une matrice C de coûts telle que $C(i,j)$ est le coût nécessaire pour se rendre, depuis la ville i, à la ville j, pour tous $0 \leq i \leq n-1$ et $0 \leq j \leq n-1$. Chaque coût $C(i,j)$ est un réel (qui peut donc même être négatif). Un *parcours* est un n-uplet $P = (i_0, \ldots, i_{n-1})$ tel que $\{i_0, \ldots, i_{n-1}\} = \{0, \ldots, n-1\}$. Cela signifie que chaque entier de 0 à $n-1$ apparaît exactement une fois dans P. Remarquons que dans cette formulation du problème, il n'y a pas d'exigence à commencer la visite avec la ville 0 ni à revenir à la ville de départ à la fin du parcours. Le *coût* $|P|$ du parcours P est défini par

$$|P| = \sum_{j=0}^{n-2} C(i_j, i_{j+1}) = C(i_0, i_1) + C(i_1, i_2) + \cdots + C(i_{n-2}, i_{n-1}). \tag{1}$$

Le problème accepte en entrée une taille n et une matrice C, et produit en sortie un parcours P de coût minimal.

Décrire de manière informelle (donc, sans pseudo-code mais avec des explications générales mais tout de même précises) un algorithme adoptant une approche gloutonne pour résoudre le problème du voyageur de commerce. Il n'est pas nécessaire que l'algorithme produise une solution optimale, mais doit tout de même s'en rapprocher raisonnablement. La stratégie proposée doit être appuyée d'un exemple concret et exhaustif avec $n = 5$.

Réponse —

Question 1.2. *(8 points)* — Donner le pseudo-code de l'algorithme VoyageurGlouton décrit informellement dans la question 1.1.

Réponse —

Question 1.3. *(4 points)* — En considérant l'algorithme VoyageurGlouton proposé dans la question 1.2, expliquer s'il donne ou non, pour chaque entrée possible, une solution optimale. Que la réponse soit positive ou négative, le démontrer.

Réponse —

Question 1.4. *(4 points)* — Exprimer, en fonction du nombre de villes n, la complexité en temps dans le pire cas de l'algorithme VoyageurGlouton proposé dans la question 1.2. Cette complexité doit être justifiée correctement, mais il n'est pas nécessaire de le faire de manière aussi rigoureuse qu'avec la méthode des instructions barométriques. Une explication générale mais précise est suffisante.

Réponse —

1.2 Problème de l'ensemble intersectant

Question 1.5. *(4 points)* — Considérons le *problème de l'ensemble intersectant* selon la formulation suivante. Nous disposons de deux entiers $n \geq 1$ et $m \geq 1$ ainsi que d'un ensemble d'ensembles $H = \{H_1, \ldots, H_m\}$ tel que pour tout $1 \leq i \leq m$, $H_i \neq \emptyset$ et $H_i \subseteq \{1, \ldots, n\}$. Un ensemble $X \subseteq \{1, \ldots, n\}$ est *H-intersectant* si pour tout $1 \leq i \leq m$, $X \cap H_i \neq \emptyset$. Autrement dit, tout ensemble de H possède au moins un élément qui est dans X. Le problème accepte en entrée les entiers n et m ainsi que H, et produit en sortie un ensemble X de cardinal minimal.

Donner une instance de ce problème avec $n = 8$ et $m = 4$. Pour l'ensemble H proposé, donner un ensemble H-intersectant et un ensemble qui n'est pas H-intersectant.

Réponse —

Question 1.6. *(6 points)* — Décrire de manière informelle (donc, sans pseudo-code mais avec des explications générales mais tout de même précises) un algorithme adoptant une approche gloutonne pour résoudre le problème de l'ensemble intersectant. Il n'est pas nécessaire que l'algorithme produise une solution optimale, mais doit tout de même s'en rapprocher raisonnablement. La stratégie proposée doit être appuyée d'un exemple concret et exhaustif.

Indice : dans une approche gloutonne, la solution est souvent construite en faisant grandir une solution petit à petit. Cela n'est pas obligatoire et il peut être bon de penser de manière inversée.

Réponse —

Question 1.7. *(8 points)* — Donner le pseudo-code de l'algorithme EnsembleIntersectant décrit informellement dans la question 1.6. Il est important d'expliquer comment les ensembles sont représentés et de donner les fonctions de manipulation intermédiaires des ensembles.

Réponse —

Question 1.8. *(4 points)* — En considérant l'algorithme EnsembleIntersectant proposé dans la question 1.7, expliquer s'il donne ou non, pour chaque entrée possible, une solution optimale. Que la réponse soit positive ou négative, le démontrer.

Réponse —

Question 1.9. *(4 points)* — Exprimer, en fonction des valeurs n et m, la complexité en temps dans le pire cas de l'algorithme EnsembleIntersectant proposé dans la question 1.7. Cette complexité doit être justifiée correctement, mais il n'est pas nécessaire de le faire de manière aussi rigoureuse qu'avec la méthode des instructions barométriques. Une explication générale mais précise est suffisante.

Réponse —

2 Algorithmes sur les graphes

2.1 Raccourcis dans les graphes

Question 2.1. *(4 points)* — Soit $G = (S, A)$ un graphe orienté. Un *chemin simple* dans G entre deux de ses sommets u_0 et u_k est une suite $(u_0, u_1, \ldots, u_{k-1}, u_k)$ de sommets tous différents de G telle que pour tout $0 \leq i \leq k-1$, (u_i, u_{i+1}) est un arc de G. La valeur $k \geq 0$ est la *longueur* de ce chemin simple. Un *raccourci* dans G est un arc $(u_0, u_k) \in A$ tel qu'il existe un chemin simple de longueur $k \geq 2$ entre u_0 et u_k.

Décrire de manière informelle (donc, sans pseudo-code mais avec des explications générales mais tout de même précises) un algorithme prenant un graphe orienté en entrée et qui teste s'il possède un raccourci.

Réponse —

Question 2.2. *(8 points)* — Donner le pseudo-code de l'algorithme ContientRaccourci décrit informellement dans la question 2.1.

Réponse —

Question 2.3. *(4 points)* — Exprimer, en fonction de $n = |S|$ et $m = |A|$, la complexité en temps dans le pire de l'algorithme ContientRaccourci proposé dans la question 2.1. Cette complexité doit être justifiée correctement, mais il n'est pas nécessaire de le faire de manière aussi rigoureuse qu'avec la méthode des instructions barométriques. Une explication générale mais précise est suffisante.

Réponse —

2.2 Test mystérieux

Question 2.4. *(8 points)* — Soit l'algorithme Test décrit par le pseudo-code suivant, utilisant l'algorithme TestAux :

```
1.  Fonction Test(g)
2.      Pour u ∈ g.sommets Faire
3.          u.statut ← BLANC
4.      Fin Pour
5.      Pour u ∈ g.sommets Faire
6.          Si ¬TestAux(g, u) Alors
7.              Renvoyer Faux
8.          Fin Si
9.      Fin Pour
10.     Renvoyer Vrai
11. Fin Fonction
```

```
1.  Fonction TestAux(g, u)
2.      Si u.statut = NOIR Alors
3.          Renvoyer Vrai
4.      Sinon Si u.statut = GRIS Alors
5.          Renvoyer Faux
6.      Sinon
7.          u.statut ← GRIS
8.          Pour v ∈ g.successeurs(u) Faire
9.              Si ¬TestAux(g, v) Alors
10.                 Renvoyer Faux
11.             Fin Si
12.         Fin Pour
13.         u.statut ← NOIR
14.         Renvoyer Vrai
15.     Fin Si
16. Fin Fonction
```

Il prend en entrée un graphe orienté g pourvu des attributs présentés en cours. Chaque sommet u de g possède aussi un attribut supplémentaire $u.statut$ qui peut prendre BLANC, GRIS ou NOIR comme valeur.

Donner la description du problème auquel cet algorithme répond. La réponse doit être justifiée. Elle peut s'appuyer sur des exemples bien choisis qui illustrent le comportement de l'exécution de cet algorithme.

Réponse —

3 Analyse amortie

3.1 Deux piles pour une file

Question 3.1. *(4 points)* — Nous représentons une file d'attente f par la combinaison de deux piles classiques $f.entrants$ et $f.sortants$. L'idée est que les éléments sortent de la file via $f.sortants$ et rentrent dans la file via $f.entrants$. Lorsque $f.sortants$ est vide que souhaitons tout de même défiler la file, tous les éléments de $f.entrants$ sont au préalable dépilés et empilés dans $f.sortants$. Voici ces deux algorithmes, avec aussi l'algorithme de construction de la file vide :

```
1. Fonction FileVide() :
2.     Soit f
3.     f.entrants ← PileVide()
4.     f.sortants ← PileVide()
5.     Renvoyer f
6. Fin Fonction
```

```
1. Procédure Enfiler(f, x)
2.     Empiler(f.entrants, x)
3. Fin Procédure
```

```
1. Fonction Défiler(f)
2.     Si ¬ EstVide(f.sortants) Alors
3.         x ← Dépiler(f.sortants)
4.         Renvoyer x
5.     Sinon
6.         Tant Que ¬ EstVide(f.entrants) Faire
7.             x ← Dépiler(f.entrants)
8.             Empiler(f.sortants, x)
9.         Fin Tant Que
10.        Si ¬ EstVide(f.sortants) Alors
11.            x ← Dépiler(f.sortants)
12.            Renvoyer x
13.        Sinon
14.            Renvoyer Erreur
15.        Fin Si
16.    Fin Si
17. Fin Fonction
```

La fonction PileVide renvoie une pile vide et les deux fonctions Empiler et Dépiler permettent respectivement d'empiler un élément dans une pile et de dépiler une pile non vide et de renvoyer l'élément obtenu. Elles s'exécutent toutes en temps constant.

Donner l'était de la file f (donc, le contenu de ses deux piles) ainsi que les valeurs des éléments défilés le cas échéant, après chaque opération de la suite d'opérations

1. $f \leftarrow$ FileVide() ;
2. Enfiler(f, 5) ;
3. Enfiler(f, 2) ;
4. Enfiler(f, 4) ;
5. Enfiler(f, 6) ;
6. Défiler(f) ;
7. Défiler(f) ;
8. Enfiler(f, 1) ;
9. Enfiler(f, 8) ;
10. Défiler(f) ;
11. Défiler(f) ;

12. DÉFILER(f) ;
13. DÉFILER(f).

Réponse —

Question 3.2. *(4 points)* — Déterminer la complexité amortie des opérations FileVide, Enfiler et Défiler introduites dans la question 3.1 en utilisant la méthode de l'agrégat.

Réponse —

Question 3.3. *(4 points)* — Déterminer la complexité amortie des opérations FileVide, Enfiler et Défiler introduites dans la question 3.1 en utilisant la méthode du comptable.

Réponse —

Question 3.4. *(4 points)* — Déterminer la complexité amortie des opérations FileVide, Enfiler et Défiler introduites dans la question 3.1 en utilisant la méthode du potentiel.

Réponse —

4 Algorithmes de retour arrière

4.1 Problème de l'ensemble intersectant

Question 4.1. *(6 points)* — Considérons le problème de l'ensemble intersectant proposé dans la question 1.5.

Décrire de manière informelle (donc, sans pseudo-code mais avec des explications générales mais tout de même précises) un algorithme adoptant une approche de retour arrière pour résoudre le problème de l'ensemble intersectant. Il est important que l'algorithme renvoie une solution de cardinal minimal.

Réponse —

Question 4.2. *(8 points)* — Donner le pseudo-code de l'algorithme EnsembleIntersectantRetourArriere décrit informellement dans la question 4.1.

Réponse —