Composition d'informatique n°4

Corrigé

⋆ ⋆ ⋆

Ensemble dominant

1 Introduction

Question 1 On remarque que l'ensemble $\{2, 4\}$ est un ensemble dominant, donc $\gamma(G_0) \leqslant 2$. Par ailleurs, aucun sommet de G_0 n'est adjacent à tous les autres, donc $\gamma(G_0) \geqslant 2$. On en déduit que $\gamma(G_0) = 2$.

Question 2 Soit $s \in S$ un sommet quelconque. Alors $|V[s]| \leqslant \Delta(G) + 1$. On en déduit que pour $X \subseteq S$:

$$|V[X]| \leqslant \sum_{s \in X} |V[s]| \leqslant |X|(\Delta(G) + 1)$$

Pour X un ensemble dominant de cardinal minimal, on a bien $|S| \leqslant \gamma(G)(\Delta(G)+1)$, ce qui donne l'inégalité voulue.

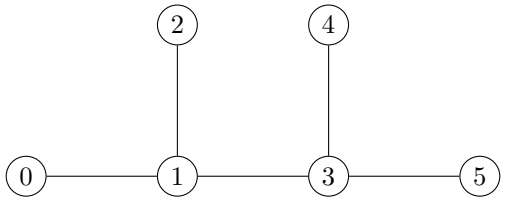
Question 3

1. Dans un graphe complet, tout sommet domine S. On en déduit que $\gamma(K_n) = 1$.
2. Soit $C_n = (s_0, \ldots, s_{n-1})$ un cycle d'ordre n. On note $X = \{s_i \mid i \equiv 0[3]\}$. Alors X est un ensemble dominant C_n. En effet, pour $j \in [\![0, n-1]\!]$:
   - si $j \equiv 0[3]$, alors $s_j \in X$;
   - si $j \equiv 1[3]$, alors s_j est voisin de s_{j-1} qui est dans X ;
   - si $j \equiv 2[3]$, alors s_j est voisin de s_{j+1} (ou s_0) qui est dans X.

   Or $|X| = \left\lceil \frac{n}{3} \right\rceil$. Sachant que $\gamma(C_n)$ est un entier et que $\Delta(C_n) = 2$, on a également $\gamma(C_n) \geqslant \left\lceil \frac{n}{3} \right\rceil$ d'après la question précédente. Finalement $\gamma(C_n) = \left\lceil \frac{n}{3} \right\rceil$.

Question 4 Soit X un stable maximal pour l'inclusion. Supposons que $V[X] \neq S$. Alors il existe $s \in S$ tel que $s \notin V[X]$. On en déduit que $X \cup \{s\}$ est toujours un stable, ce qui contredit l'hypothèse de maximalité. Par l'absurde, on en déduit que X est dominant.

Question 5 On propose le graphe :



Dans ce graphe, le seul ensemble dominant de cardinal minimal est $\{1, 3\}$. En effet, si X est un ensemble dominant de cardinal minimal, alors $1 \in X$, car sinon $\{0, 2\} \subseteq X$, et auquel cas, $X \cup \{1\} \setminus \{0, 2\}$ est toujours dominant et de cardinal strictement inférieur. De même, $3 \in X$.

Question 6 On compte le nombre de `true` dans le tableau.

```
let cardinal x =
    let card = ref 0 in
    Array.iter (fun x -> if x then incr card) x;
    !card
```

Question 7 On crée un tableau de booléens permettant de garder en mémoire l'ensemble des sommets dominés. Pour le modifier, on parcourt tous les sommets, et pour chaque sommet s de X, on met à `true` la valeur pour les sommets de $V[s]$ (et pas seulement $V(s)$).

```
let domines g x =
    let n = Array.length g in
    let dom = Array.make n false in
    for s = 0 to n - 1 do
        if x.(s) then
            List.iter (fun t -> dom.(t) <- true) (s :: g.(s))
    done;
    dom
```

Question 8 Il suffit de vérifier que tous les sommets sont dominés.

```
let est_dominant g x =
    cardinal (domines g x) = Array.length g
```

Dans la fonction `domines`, pour chaque sommet, on parcourt au plus une fois sa liste d'adjacence, et on y fait des opérations en temps constant. La boucle `for` a donc une complexité en $\mathcal{O}(|S| + |A|)$. De plus, la création du tableau et le calcul du cardinal se font en $\mathcal{O}(|S|)$, ce qui donne la complexité voulue.

Question 9 On crée les variables nécessaires, puis on écrit une fonction récursive qui prend en argument l'indice i, et on distingue selon que la solution est totale ou non.

```
let dominant_min g =
    let n = Array.length g in
    let x = Array.make n false in
    let xmin = ref [||] and
        cardmin = ref (n + 1) in
    let rec backtrack i =
        if i = n then begin
            if est_dominant g x then begin
                let card = cardinal x in
                if card < cardmin then
                    (xmin := Array.copy x; cardmin := card)
            end
        end else begin
            x.(i) <- false; backtrack (i + 1);
            x.(i) <- true; backtrack (i + 1)
        end
    in
    backtrack 0;
    !xmin
```

Question 10 Il y a $2^{|S|}$ feuilles dans l'arbres des possibilités, et pour chacune on fait un appel à `est_dominant` en $\mathcal{O}(|S| + |A|)$ et à `cardinal` et `Array.copy` en $\mathcal{O}(|S|)$. Les opérations faites dans chaque appel récursif sur un nœud interne étant en $\mathcal{O}(1)$, la complexité totale est donc en $\mathcal{O}((|S| + |A|)2^{|S|})$.

Sur un ordinateur de bureau avec un processeur à 1 GHz (approximation standard), on réalise 10^9 opérations par seconde. Or, $2^{30} \simeq 10^9$. De plus, le nombre d'arêtes d'un graphe à 30 sommets est inférieur à $15 \times 30 = 450$. Comme 450 secondes correspondent à 7 minutes 30, on en déduit qu'on peut utiliser l'algorithme pour des graphes jusqu'à un ordre de 30 environ.

Question 11 Pour l'heuristique d'évaluation, on pourrait se contenter de prendre le nombre de sommets dominés par $\widetilde{X}$, qui garantit d'être admissible. Toutefois, on peut s'inspirer de la question 2, et plutôt considérer :

$$h(\widetilde{X}) = |V[\widetilde{X}]| + \frac{|S \setminus V[\widetilde{X}]|}{\Delta(G)+1}$$

L'idée est qu'il faudra rajouter au moins un sommet par tranche de $(\Delta(G)+1)$ sommets non dominés, ce qui garantit que cette valeur donne une heuristique admissible.

Pour l'heuristique de branchement, on peut par exemple commencer par renuméroter les sommets de graphe pour qu'ils soient rangés par degrés décroissants. Dès lors, on peut choisir de commencer par ajouter le sommet i à la solution partielle, avant de faire l'appel récursif sans i.

2 Ensemble dominant, arbres et forêts

Question 12 Soit T un arbre couvrant de G (existe car G est connexe). Soit $s_0 \in S$ quelconque. On pose $X = \{s \in S \mid d_T(s_0, s) \equiv 0[2]\}$, où $d_T(s_0, s)$ désigne la distance de s_0 à s dans l'arbre T. Alors $S \setminus X = \{s \in S \mid d_T(s_0, s) \equiv 1[2]\}$.

De plus, X et $S \setminus X$ sont des ensembles dominants T, donc dominant G. En effet, tout sommet $s \in S$ est soit dans X, soit à distance 1 d'un sommet de X, et de même pour $S \setminus X$.

Question 13 Dans la question précédente, soit X, soit $S \setminus X$ est de cardinal inférieur à $\frac{|S|}{2}$.

Si G ne possède pas de sommet isolé, alors chaque composante connexe de G est un arbre qui contient au moins deux sommets. On peut donc appliquer la question précédente. En choisissant le plus petit des deux ensembles dominants dans chaque composante connexe, on forme un ensemble dominant G contenant au plus la moitié des sommets.

Question 14 Soit X un ensemble dominant G de cardinal minimal contenant le moins de feuilles de G. Supposons que X contient une feuille s. Soit alors t l'unique voisin de s. Sachant que G est un arbre et que $|S| > 2$, t n'est pas une feuille (sinon $\{s, t\}$ formerait une composante connexe).

Par ailleurs, sachant que $V[s] \subseteq V[t]$, si on pose $Y = X \cup \{t\} \setminus \{s\}$, alors Y est un ensemble dominant et $|Y| \leqslant |X|$. De plus, Y contient une feuille de moins que X, ce qui est absurde par hypothèse sur X.

Par l'absurde, on en déduit le résultat.

Question 15 L'algorithme précédent renvoie bien un ensemble dominant G. En effet, si l'algorithme termine, alors par condition de la boucle TantQue, X est un ensemble dominant.

Montrons que l'algorithme termine. Pour cela, montrons que $|S| - |X|$ est un variant de boucle.

- sachant que $X \subseteq S$, $|S| - |X| \geqslant 0$;
- à chaque itération, pour s le sommet choisi, $p_X(s) > 0$ (car s est non dominé). Si $p_X(s) = 1$, alors le cardinal de X augmente de 1, donc $|S| - |X|$ diminue de 1. Sinon, alors $p_X(s) > 1$, donc s possède un voisin non dominé, donc le sommet t est bien défini. À nouveau, le cardinal de X augmente de 1.

En revanche, l'ensemble X n'est pas nécessairement de cardinal $\gamma(G)$. En effet, on remarque que par construction, X est un stable. D'après la question 5, X n'est pas nécessairement minimal.

Question 16 Si T_s est une feuille, alors $\gamma(s) = \gamma^+(s) = 1$ et $\gamma^?(s) = 0$.

Question 17

1. un ensemble X contenant s et dominant T_s est l'union de $\{s\}$ et d'ensemble dominant les T_{s_i}, sauf éventuellement les s_i (car s les domine déjà).
2. un ensemble dominant T_s sauf éventuellement s peut soit contenir s, soit être l'union d'ensembles dominant les T_{s_i}.
3. un ensemble dominant T_s peut soit contenir s, soit être l'union d'ensembles dominant les T_{s_i}, dont au moins l'un d'entre eux contient s_i.

Question 18 C'est un simple parcours de graphe.

```
let enracine g =
    let n = Array.length g in
    let tree = Array.make n [] in
    let vus = Array.make n false in
    let rec dfs s =
        vus.(s) <- true;
        let traiter t =
            if not vus.(t) then begin
                tree.(s) <- t :: tree.(s);
                dfs t
            end
        in
        List.iter traiter g.(s)
    in
    dfs 0;
    tree
```

Question 19 On garde en mémoire trois tableaux permettant de calculer les valeurs suggérées par l'énoncé. On fait les calculs récursivement avec mémoïsation.

```
let gamma_arbre g =
    let tree = enracine g in
    let n = Array.length g in
    let gamma = Array.make n 1 and
        gamma_plus = Array.make n 1 and
        gamma_int = Array.make n 0 in
    let rec remplir s =
        if tree.(s) <> [] then begin
            let somme = ref 0 and somme_int = ref 0 in
            let traiter t =
                remplir t;
                somme := !somme + gamma.(t);
                somme_int := !somme_int + gamma_int.(t)
            in
            List.iter traiter g.(s);
            gamma_plus.(s) <- 1 + !somme_int;
            gamma_int.(s) <- min gamma_plus.(s) !somme;
            gamma.(s) <- List.fold_left
                (fun m t -> min m (gamma_plus.(t) + !somme - gamma.(t)))
                gamma_plus.(s) g.(s)
        end
    in
    remplir 0;
    gamma.(0)
```

La complexité linéaire est garantie, car l'appel à `remplir` n'aura lieu qu'une seule fois par sommet (car G

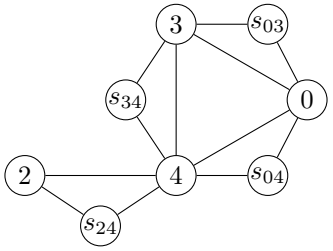
n'a pas de cycle). Pour chaque sommet, on parcourt deux fois sa liste d'adjacence, en y faisant des opérations en temps constant.

3 Un problème difficile

Question 20 Le problème a lieu dans la réciproque : comme X est dominant, soit s, soit un voisin de s est dans X, mais ce voisin n'est pas nécessairement t. X n'est donc pas nécessairement une couverture des arêtes.

Par exemple, dans le graphe complet K_n, il existe un ensemble dominant K_n de cardinal 1, mais la plus petite couverture des arêtes par les sommets contient $n-1$ sommets.

Question 21 On obtient le graphe :



Question 22 Soit $G = (S, A)$ un graphe, $k \in \mathbb{N}$, et G' le graphe donné par la construction précédente. Notons S_0 l'ensemble des sommets isolés de S.

Si G possède une couverture par sommets X de cardinal $\leqslant k$, alors $Y = X \setminus S_0$ est toujours une couverture par sommets de G (les sommets isolés n'apportant rien), et Y est un ensemble dominant de G'. En effet :

- si $s \in S \setminus S_0$ est un sommet de G', alors s est incident à au moins une arête $a = \{s, t\}$. Comme X est une couverture par sommets, soit s, soit t est dans Y. On en déduit que dans les deux cas, s est dominé par Y ;
- si $a = \{s, t\} \in A$, alors à nouveau, soit s, soit t est dans Y. On en déduit que dans les deux cas, s_a est dominé par Y.

De plus, Y est de cardinal $\leqslant k$.

Réciproquement, si G' possède un ensemble dominant X de cardinal $\leqslant k$, alors il existe un ensemble Y dominant G' de cardinal $\leqslant k$ tel que $Y \subseteq S$. En effet, si X contient un sommet s_a, avec $a = \{s, t\} \in A$, alors $X \cup \{s\} \setminus \{s_a\}$ est toujours dominant et ne contient pas plus de sommets que X. En répétant ce processus pour chaque sommet s_a, on obtient l'ensemble Y voulu. Dès lors, Y est une couverture par les sommets de G. En effet, soit $a = \{s, t\} \in A$. Comme s_a est dominé par Y et que $s_a \notin Y$, alors $s \in Y$ ou $t \in Y$ (car s et t sont les seuls voisins de s_a).

Enfin, la construction de G' peut bien se faire en temps polynomial en $|S| + |A|$, ce qui donne la réduction voulue.

Question 23 La réduction précédente montre que `DOMINANT` est NP-difficile. Montrons que `DOMINANT` $\in$ NP.

Un certificat valide est une partie $X \subseteq S$. C'est bien de taille polynomiale en $|S| + |A|$. On peut vérifier en temps polynomial que c'est un ensemble dominant et que $|X| \leqslant k$ (cf questions 6 et 7).

On en déduit que `DOMINANT` $\in$ NP, puis que `DOMINANT` est NP-complet.

4 Des algorithmes efficaces non optimaux

Question 24 Par linéarité de l'espérance, $\mathbb{E}(|Y|) = \sum_{s \in S} \mathbb{P}(s \in Y)$. Or $s \in Y$ si et seulement si aucun sommet de $V[s]$ n'est ajouté à X. Cet événement se produit avec probabilité $(1-p)^{\deg(s)+1}$ (car $|V[s]| = \deg(s)+1$).

On en déduit que $\mathbb{E}(|Y|) = \sum_{s \in S} (1-p)^{\deg(s)+1} \leqslant |S|(1-p)^{\delta(G)+1}$.

Question 25 On remarque que $X \cup Y$ est un ensemble dominant G, par construction (car $S = V[X] \cup Y \subseteq V[X \cup Y]$).

De plus, $\mathbb{E}(|X \cup Y|) = \mathbb{E}(|X| + |Y|) = \mathbb{E}(|X|) + \mathbb{E}(|Y|) \leqslant |S|p + |S|(1-p)^{\delta(G)+1}$.

Comme l'espérance de $|X \cup Y|$ est majorée par cette valeur, il existe une exécution telle que $|X \cup Y|$ est inférieur à cette valeur, ce qui conclut.

Question 26 Comme pour `est_dominant`, on garde en mémoire un tableau des sommets dominés (`VX`). On parcourt alors tous les sommets de S, et dès qu'on tombe sur un sommet non dominé, on l'ajoute à X et on marque comme dominé ce sommet et tous ses voisins.

```
bool* glouton_naif(graphe G){
    bool* X = malloc(G.n * sizeof(bool));
    bool* VX = malloc(G.n * sizeof(bool));
    for (int s=0; s<G.n; s++){
        X[s] = false; VX[s] = false;
    }
    for (int s=0; s<G.n; s++){
        if (!VX[s]){
            X[s] = true; VX[s] = true;
            for (int i=0; i<G.deg[s]; i++){
                VX[G.adj[s][i]] = true;
            }
        }
    }
    free(VX);
    return X;
}
```

Question 27 On considère un graphe en étoile : $G = (S, A)$ avec :

- $S = [\![0, n-1]\!]$;
- $A = \{\{i, n-1\} \mid i \in [\![0, n-2]\!]\}$.

Alors $\{n-1\}$ est un ensemble dominant G de cardinal minimal, mais l'algorithme glouton renverrait $\{0, 1, \ldots, n-2\}$, qui est de cardinal $n-1 = (n-1)\gamma(G)$.

Question 28

- première itération : le sommet de portée maximale est 1, de portée 5 ;
- deuxième itération : le sommet de portée maximale est 4, de portée 4 ;
- troisième itération : deux sommets sont de portée maximale : 2 et 5. On choisit le sommet 2.

L'ensemble renvoyé est $\{1, 2, 4\}$. Il est effectivement de cardinal minimal. En effet, 7 étant une feuille, un ensemble dominant G_2 contient 4 ou 7. Comme $V[7] \subseteq V[4]$, on peut sans perte de généralité supposer qu'il contient 4. Comme il n'existe aucun sommet qui domine $S \setminus V[4]$, on en déduit que $\gamma(G_2) > 2$.

Question 29 Pas de difficulté ici. On parcourt les voisins pour incrémenter éventuellement un compteur.

```
int portee(graphe G, bool* VX, int s){
    int pxs = 0;
    if (!VX[s]) pxs++;
    for (int i=0; i<G.deg[s]; i++){
        if (!VX[G.adj[s][i]]) pxs++;
    }
    return pxs;
}
```

Question 30 On initialise `X` et `VX` comme dans l'algorithme glouton naïf, puis on applique le principe de l'algorithme glouton amélioré, sans optimisation particulière. On garde en mémoire un compteur pour savoir si l'ensemble X est dominant.

```
bool* glouton_mieux(graphe G){
    bool* X = malloc(G.n * sizeof(bool));
    bool* VX = malloc(G.n * sizeof(bool));
    for (int s=0; s<G.n; s++){
        X[s] = false; VX[s] = false;
    }
    int nb_dom = 0;
    while (nb_dom < G.n){
        int s = 0;
        int pxs = portee(G, VX, 0);
        for (int t=0; t<G.n; t++){
            int pxt = portee(G, VX, t);
            if (pxt > pxs){
                s = t; pxs = pxt;
            }
        }
        X[s] = true; VX[s] = true;
        nb_dom++;
        for (int i=0; i<G.deg[s]; i++){
            VX[G.adj[s][i]] = true;
            nb_dom++;
        }
    }
}
```

Question 31 Le calcul de portée a une complexité en $\mathcal{O}(\deg(s))$. À chaque itération on calcule la portée de chaque sommet, soit une complexité en $\mathcal{O}(|S|+|A|)$. Il y a au plus $|S|$ itérations, soit une complexité totale en $\mathcal{O}(|S|(|S|+|A|))$.

Question 32 Lorsqu'un sommet s est choisi par l'algorithme pour être ajouté à X, le coût d'exactement $p_X(s)$ sommets est fixé à $\frac{1}{p_X(s)}$. On en déduit qu'à chaque itération, la somme des coûts est augmentée d'exactement 1, comme le cardinal de X.

Question 33 Au moment où le premier sommet s_1 est dominé, notons t le sommet choisi par l'algorithme. Par principe de l'algorithme, à cette étape, $p_X(t) \geqslant p_X(s^*)$. Or, à cette étape, $p_X(s^*) = \deg(s^*)+1$ (car aucun sommet de $V[s^*]$ n'est dominé). On en déduit que $c(s_1) = \frac{1}{p_X(t)} \leqslant \frac{1}{p_X(s^*)} \leqslant \frac{1}{\deg(s^*)+1}$.

Question 34 Lorsque le sommet s_i est dominé, les sommets $s_i, s_{i+1}, \ldots, s_k$ ne sont pas encore dominés. Par un raisonnement similaire à celui de la question précédente, on en déduit que $c(s_i) \leqslant \dfrac{1}{k-i+1} =$

$\dfrac{1}{\deg(s^*) - i + 2}$, car $k = |V[s^*]| = \deg(s^*) + 1$.

Dès lors, $\sum_{i=1}^{k} c(s_i) \leqslant \sum_{i=1}^{\deg(s^*)+1} \dfrac{1}{\deg(s^*) - i + 2} = H(\deg(s^*) + 1) \leqslant H(\Delta(G) + 1)$.

Question 35 On a les inégalités suivantes :

$$|X| = \sum_{s \in S} c(s) \leqslant \sum_{s^* \in X^*} \sum_{s \in V[s^*]} c(s) \leqslant \sum_{s^* \in X^*} H(\Delta(G) + 1) = \gamma(G) H(\Delta(G) + 1)$$

On conclut en remarquant que pour tout n, $H(n + 1) \leqslant \ln n + 2$.

$\star \star \star$