Entraînements divers

Ce document est un complément aux exercices de type A CCINP destiné à courvir les différents thèmes au programme dans la filière MPI. Les exercices suivants sont des exercices de révision qui ne sont pas calibrés pour un concours particulier ni en termes de durée ni en termes de contenu.

Logique

Exercice 1 *Syntaxe et sémantique de la logique propositionnelle*

Un voyageur navigue dans un archipel à la recherche de l'île Maya. Les îles de cet archipel sont occupées par d'étranges habitants : les Pires, qui mentent systématiquement ; et les Purs, qui disent toujours la vérité. Sur chaque île, le voyageur rencontre deux habitants, A et B qui acceptent de lui parler.

1. Sur la première île, A et B tiennent les propos suivants :

   A : B est un Pur et nous sommes sur l'île Maya.

   B : A est un Pire et nous sommes sur l'île Maya.

   a) Modéliser la situation à l'aide d'une formule F_1 du calcul propositionnel.

   b) Mettre F_1 sous forme normale conjonctive sans utiliser de table de vérité.

   c) Appliquer l'algorithme de Quine à la FNC obtenue à la question précédente et en déduire si le voyageur est arrivé sur l'île Maya et, si possible, la nature de A et B.

2. Sur la deuxième île, A et B tiennent les propos suivants :

   A : "Nous sommes deux Pires, et nous sommes sur l'île Maya".

   B : "L'un de nous au moins est un Pire, et nous ne sommes pas sur l'île Maya".

   a) Modéliser la situation à l'aide d'une formule F_2 du calcul propositionnel.

   b) Établir la table de vérité de F_2 et en déduire une formule équivalente à F_2 sous forme normale disjonctive.

   c) Déterminer si le voyageur est arrivé sur l'île Maya.

Exercice 2 *Vocabulaire autour de la logique du premier ordre*

Répondre aux questions suivantes en sachant que x, y, z sont des symboles de variables et que l'expression suivante est une formule du premier ordre :

$$F = \forall x \, \exists y \, f(g(x, y), a, z) \wedge \forall z \, f(x, g(x, Q(a)), z)$$

1. Pour chaque symbole dans F, dire si c'est un symbole de fonction ou un symbole de relation et donner son arité.
2. Déterminer l'ensemble des termes et des formules atomiques de F.
3. Pour chaque occurrence de variable dans F, indiquer si elle est libre ou liée ; dans le cas où elle est liée, préciser par quel quantificateur.

Voir aussi :

- L'exercice 11 de la fiche d'entraînement CCINP (déduction naturelle).
- L'exercice 17 de la fiche d'entraînement CCINP (formules propositionnelles croissantes).

Analyse d'algorithmes

Exercice 3 *Variants et invariants*

On considère la fonction suivante :

```
bool mystere(char* s)
{
  int gauche = 0;
  int droite = strlen(s) -1;
  while (gauche < droite)
  {
    if (s[gauche] != s[droite]) {return false; }
    gauche = gauche +1;
    droite = droite -1;
  }
  return true;
}
```

1. Donner la spécification de la fonction `mystere`.
2. Sous l'hypothèse qu'elle termine, justifier de sa correction à l'aide d'invariants bien choisis.
3. Montrer que cette fonction termine effectivement sur toute entrée.
4. Quelle est sa complexité en fonction de la taille de l'entrée ?

Exercice 4 *Graphe de flot de contrôle*

On considère le programme suivant qui prend en entrée un entier, deux tableaux de taille identique et leur taille.

```
void corresponding_value(int v, int* a1, int* a2, int n){
  int r = -1;
  int i = 0;
  while(i < n){
    if (a1[i] == v){
      r = i;
    }
  i = i + 1;
  }
  return a2[r];
}
```

1. Indiquez ce que fait ce programme.
2. Construisez son graphe de flot de contrôle.
3. Indiquez le chemin obtenu avec l'entrée `v = 1, a1 = {3,5,1}, a2 = {5,1,3}, n = 3`.
4. Commentez la couverture des sommets et des arcs que permet ce chemin.
5. Un jeu de test contenant uniquement ce chemin vous semble-t-il suffisant pour tester le programme ?

Stratégies algorithmiques

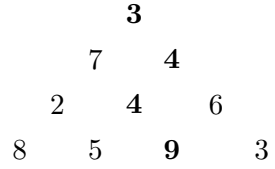
Exercice 5 *Diviser pour régner*

On considère un tableau A de taille n. On pourra dans un premier temps supposer que n est une puissance de deux. Un élément x de A est dit majoritaire si et seulement si A contient strictement plus de $n/2$ occurrences de x.

1. Donner un algorithme naïf permettant de calculer s'il existe un élément majoritaire dans un tableau donné et étudier sa complexité temporelle pire cas et meilleur cas.
2. Écrire un algorithme `occurrences` permettant de calculer le nombre d'occurrences de x dans un tableau A entre les indices i et j. Etudier sa complexité.
3. Supposons qu'on divise A en deux parties égales et qu'on soit en mesure de savoir sur chacune des moitiés si elle comprend un élément majoritaire et si oui quelle est sa valeur. Expliquer comment calculer la valeur majoritaire de A si elle existe à partir de cette connaissance.
4. Écrire un algorithme `majoritaire` prenant en entrée un tableau A et deux indices dans ce tableau i, j et qui renvoit le couple (Vrai, x) si x est majoritaire dans $A[i, j]$ et (Faux, -1) si $A[i, j]$ ne contient pas d'élément majoritaire en utilisant la question précédente.
5. On note $C(n)$ le nombre d'opérations pire cas effectuées sur l'entrée $(A, 1, n)$ par `majoritaire`. Déterminer une relation de récurrence sur $C(n)$ et en déduire la complexité de `majoritaire`.

Exercice 6 *Programmation dynamique*

On considère des entiers naturels disposés sur un triangle équilatéral de hauteur n. La figure suivante donne un exemple pour un tel triangle lorsque $n = 4$:



On appelle chemin une suite de nombres du triangle obtenue en se déplaçant vers les nombres adjacents de la ligne inférieure à partir du sommet et finissant sur un nombre de la base. Un chemin est indiqué en gras sur la figure précédente. On cherche à calculer la valeur maximale d'un chemin reliant le sommet à la base d'un triangle de hauteur n donné. Un triangle sera représenté par une matrice $n \times n$ contenant à la ligne i les coefficients de la i-ème ligne du triangle.

1. Résoudre le problème dans le cas du triangle ci-dessus.
2. Combien y a-t-il de chemins possibles dans le triangle ? En déduire la complexité d'un algorithme naïf (brièvement décrit) résolvant ce problème : est-il envisageable de procéder ainsi ?
3. Proposer un algorithme dynamique permettant de résoudre ce problème. On pourra établir une relation de récurrence sur $f(i, j)$, la valeur maximale d'un chemin passant du sommet à la position (i, j).
4. Etudier la complexité temporelle et spatiale de cet algorithme.
5. Peut on modifier l'algorithme de sorte à calculer un chemin de valeur maximale en plus de ladite valeur sans déteriorer sa complexité ?

Exercice 7 *Algorithmes gloutons et d'approximation*

On considère le problème d'optimisation `SUBSET SUM` suivant :

$$\begin{cases} \textbf{Entrée} : \text{Un tableau } T = [t_1, ..., t_n] \text{ d'entiers naturels et } C \in \mathbb{N}. \\ \textbf{Solution} : \text{Un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \sum_{i \in I} t_i \leq C. \\ \textbf{Optimisation} : \text{Maximiser } \sum_{i \in I} t_i. \end{cases}$$

La version décisionnelle de ce problème est NP-complète. On propose d'approcher une solution à `SUBSET SUM` à l'aide de l'algorithme glouton suivant :

```
S ← 0 et I ← ∅
pour i allant de 1 à n
    si S + t_i ≤ C alors
        S ← S + t_i
        I ← I ∪ {i}
renvoyer I
```

1. Montrer que cet algorithme n'est pas un algorithme d'approximation à facteur constant pour `SUBSET SUM`.

On modifie l'algorithme précédent de sorte à considérer les t_i par ordre décroissant de valeur (plutôt que par ordre croissant d'indice) : cela donne naissance à un algorithme qu'on note A.

2. Déterminer la complexité temporelle de A.
3. Montrer que A est une 1/2-approximation de `SUBSET SUM`.
4. On vient de montrer que le facteur d'approximation de A est au moins égal à 1/2. Est-il meilleur ?

Exercice 8 *Algorithmes probabilistes*

On considère l'opération suivante : étant donnés n éléments distincts et comparables, les insérer dans un arbre binaire de recherche (ABR).

1. Quelle est la complexité pire cas d'une recherche dans le pire ABR résultant de cette opération ? Et dans le meilleur ABR ? Donner un ordre d'insertion lorsque les éléments sont ceux de $[\![1, 7]\!]$ pour lequel un des pires ABR est produit puis un ordre pour lequel un des meilleurs est produit.

Afin d'éviter le pire cas, on insère les n éléments en les choisissant dans un ordre aléatoire : on considère que les $n!$ permutations possibles des éléments sont des ordres d'insertion équiprobables. On note H_n la variable aléatoire correspondant à la hauteur de l'arbre obtenu, $X_n = 2^{H_n}$, R_n le rang de la racine de l'arbre parmi les n éléments (c'est-à-dire sa position si les n éléments étaient triés) et $Z_{n,i}$ la variable aléatoire valant 1 si $R_n = i$ et 0 sinon.

2. Montrer que, si $R_n = i$ alors $X_n = 2\max(X_{i-1}, X_{n-i})$.
3. Montrer que $X_n = \sum_{i=1}^{n} Z_{n,i} \times 2\max(X_{i-1}, X_{n-i})$.
4. En observant que $Z_{n,i}$ est indépendante de X_{i-1} et X_{n-i}, montrer que $\mathbb{E}[X_n] \leq \frac{4}{n} \sum_{k=0}^{n-1} \mathbb{E}[X_k]$.

On admet qu'une telle relation de récurrence implique que $\mathbb{E}[X_n] \leq \frac{1}{4}\binom{n+3}{n}$.

5. L'inégalité de Jensen garantit que si f est une fonction convexe sur un intervalle I et X est une variable aléatoire sur I alors $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$. Déduire de ce qui précède que $\mathbb{E}[H_n] \leq O(\log n)$ et conclure.

Algorithmique

Exercice 9 *Algorithmes de compression*

1. Compresser la chaîne "charabiabaragouin" à l'aide de l'algorithme de Huffman.
2. Décompresser la chaîne

   1101111100101110111010010110110011100101110100111001101010001101110111100101000

   sachant qu'elle a été compressée à l'aide de l'algorithme de Huffman et que la clé de codage est la suivante :

Lettre	a	d	e	i	n	o	s	t	u
Code	0010	11	0100	011	101	100	000	0011	0101

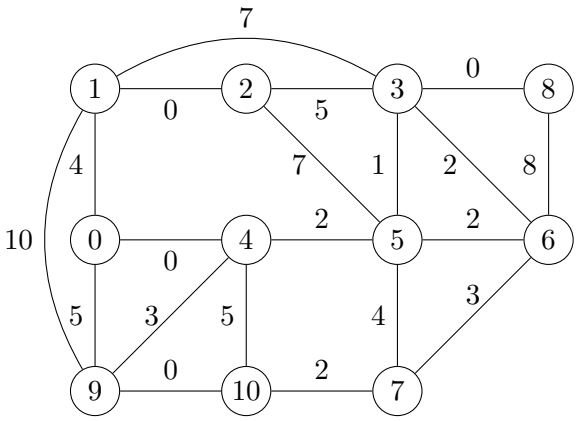
3. Compresser la chaîne AABAACBAAAABBCBAC à l'aide de l'algorithme LZW en prenant comme dictionnaire de base celui qui suit :

Chaine	A	B	C
Code	1	2	3

4. Sachant que le dictionnaire initial est constitué des lettres latines minuscules codées par leur position dans l'alphabet (ainsi, la lettre a est associée au code 1 et la lettre z au code 26), décompresser via l'algorithme LZW la chaîne suivante :

   16 1 16 15 21 19 27 27 19 28 30 24 33 1 19

Exercice 10 *Algorithmique des graphes*



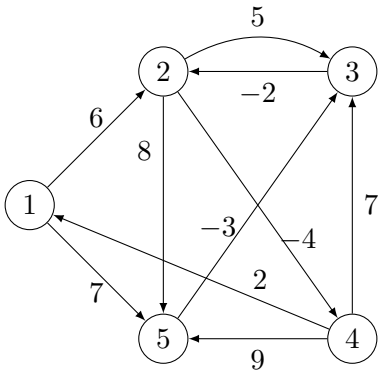
1. Effectuer un parcours en profondeur du graphe depuis 0 en parcourant d'abord les petits sommets.
2. En utilisant l'algorithme de Dijkstra, déterminer un plus court chemin de 1 à 7.
3. En utilisant l'algorithme de Kruskal, déterminer un arbre couvrant minimal pour ce graphe.
4. Orienter les arêtes de poids pair vers le plus petit des deux sommets et les arêtes de poids impair vers le plus grand. Déterminer les composantes fortement connexes du graphe résultant via l'algorithme de Kosaraju et en déduire le graphe de CFC.

Exercice 11 *Algorithme de Bellman-Ford*

On considère l'algorithme suivant, du à Bellman et Ford :

```
   Entrée : Un graphe G = (S, A) pondéré par une fonction w ; un sommet s ∈ S.
   Sortie : Un tableau D contenant à la case i le poids d'un plus court chemin de s
            à i si G ne contient pas de cycle de poids strictement négatif accessible
            depuis s et la phrase "il y a un cycle de poids < 0" sinon.
1  D ← tableau de |S| cases contenant +∞ partout sauf D[s], qui contient 0
2  pour k allant de 1 à |S| − 1
3  |  pour tout arc (u, v) ∈ A
4  |  |  D[v] ← min(D[v], D[u] + w(u, v))
5  pour tout arc (u, v) ∈ A
6  |  si D[u] + w(u, v) < D[v] alors
7  |  |  renvoyer "il y a un cycle de poids < 0"
8  renvoyer D
```

1. Appliquer cet algorithme au graphe suivant avec $s = 1$:



   Que ce serait-il passé si l'arc $(4, 5)$ était de poids 8 au lieu de 9 ?

Pour tout sommet $v \in S$ et tout $k \in [\![0, |S| - 1]\!]$, on note $d(v, k)$ le contenu de la case $D[v]$ à la fin de la k-ème itération de la boucle en ligne 2. Ainsi on a en particulier que $d(v, 0)$ est le contenu de $D[v]$ avant de rentrer pour la première fois dans cette boucle.

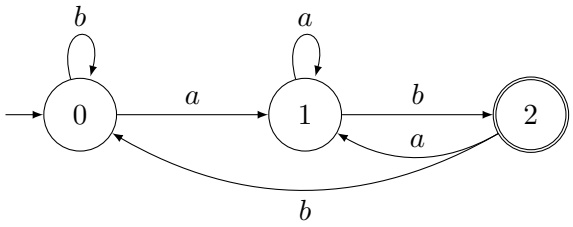
2. En raisonnant par l'absurde, montrer que s'il existe un cycle de poids strictement négatif dans G alors l'algorithme de Bellman-Ford le détecte.
3. Montrer que lorsque G ne contient aucun cycle de poids strictement négatif, pour tout $v \in S$ et tout $k \in [\![0, |S| - 1]\!]$, $d(v, k)$ est le poids d'un plus court chemin de s à v empruntant au plus k arcs.
4. Déduire de ce qui précède que l'algorithme de Bellman-Ford est correct vis-à-vis de sa spécification.
5. Quelle est la complexité de cet algorithme ?
6. Expliquer comment on aurait pu reconstruire un plus court chemin de s à v pour chaque sommet v plutôt que de simplement connaître son poids.
7. Discuter des atouts et inconvénients de cet algorithme par rapport à l'algorithme de Dijkstra.

Exercice 12 *Algorithmes sur les automates*

1. a) En appliquant l'algorithme de Berry-Sethi, construire un automate reconnaissant le langage L dénoté par $(a + c)^\star abb + (a + c)^\star$.

   b) En déduire un automate reconnaissant L^c.

2. a) En appliquant la méthode d'élimination des états et en supprimant les états par ordre croissant de numéro, donner une expression rationnelle pour le langage reconnu par l'automate suivant :



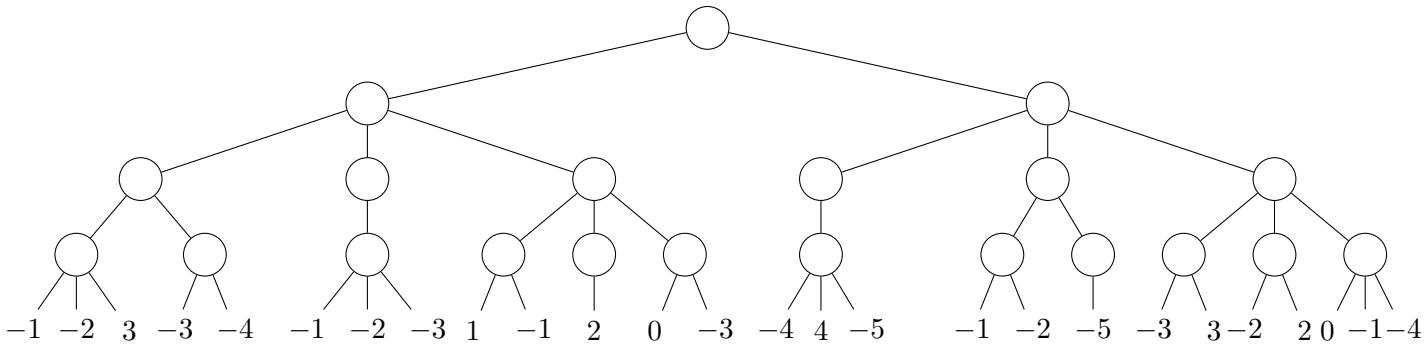
b) Donner une description simple des mots reconnus par cet automate et en déduire une autre expression rationnelle pour le langage qu'il reconnaît.

Exercice 13 *Min-max et élagage*

Déterminer le score de la racine étant données les valeurs de l'heuristique aux feuilles de l'arbre suivant :

1. En supposant que la racine est un état du joueur qui veut maximiser le score de la racine.
2. En supposant que la racine est un état du joueur qui veut le minimiser.

Identifier les branches qui n'auraient pas été explorées avec un élagage alpha-beta dans le premier cas.



Structures de données

Exercice 14 *Structures de données linéaires*

On s'intéresse à une structure appelée *deque* (double ended queue) qui se comporte comme une file mais dans laquelle on peut ajouter et retirer des éléments aux deux extrémités.

1. Représenter l'évolution du contenu de la structure lors de la séquence d'opérations suivantes à partir d'une deque d'entiers vide : ajout de 5 à gauche, ajout de 3 à droite puis de 7 à droite puis de 1 à gauche, suppression à droite, ajout de 8 à droite, trois suppressions à gauche.
2. Indiquer les opérations qu'on voudrait pouvoir effectuer sur une telle structure.
3. On propose d'implémenter une deque d'entiers en C à l'aide d'un tableau. Définir une struct adaptée (on réfléchira à quels champs introduire pour pouvoir manipuler la deque efficacement). *Indication : comment auriez-vous implémenté une file à l'aide d'un tableau en C ?*
4. Avec votre proposition à la question 3, indiquer comment on implémenterait les opérations de la question 2 et quelle serait la complexité de chacune.

Exercice 15 *Structure de tas*

Un flux de données est une séquence infinie de données. On peut se représenter un tel objet comme étant des données produites en temps réel. Étant donné le caractère infini de cette séquence, on ne peut pas attendre qu'elle soit entièrement émise pour pouvoir traiter les données : il faut le faire "online", c'est-à-dire au fur et à mesure. Ici, on s'intéresse au problème consistant à connaître à tout moment la valeur du k-ème plus grand élément parmi tous les éléments qui ont déjà émis sur le flux.

1. Dans le cas où $k = 1$, proposer une méthode simple pour stocker ce k-ème plus grand élément.
2. Comment peut-on adapter cette méthode pour $k = 2$? Et pour k quelconque ?
3. Quelles sont les complexités spatiale et temporelle de l'approche précédente ?
4. Expliquer comment on pourrait utiliser une structure de tas pour stocker le k-ème plus grand élément d'un flux et justifier que cette stratégie améliore la complexité temporelle par rapport à la méthode naïve. Améliore-t-elle la complexité spatiale ?

À présent on souhaiterait connaître non seulement le k-ème plus grand élément du flux mais aussi le $(k+1)$-ème, le $(k+2)$-ème... jusqu'au $(k+q-1)$-ème plus grand élément.

5. Proposer une méthode pour stocker ces éléments et discuter de sa complexité temporelle et spatiale. On pourra continuer à utiliser des tas.
6. On aimerait, en plus de stocker les plus grands éléments entre le k-ème et le $(k+q-1)$-ème, pouvoir à tout moment les afficher en temps linéaire en q. Si votre méthode précédente ne le permet pas, indiquer comment la raffiner pour répondre à cette contrainte.

Voir aussi :

- L'exercice 12 CCINP (minima locaux dans un arbre binaire).
- L'exercice 19 CCINP (arbres binaires de recherche).

Langages formels

Exercice 16 *Langages quelconques*

Dans ce qui suit, A, B, C sont des langages sur un même alphabet.

1. Déterminer en justifiant quelles sont les affirmations correctes parmi :

   (1) $(A^\star)^\star = A^\star$.
   (2) $A(B + C) = AB + AC$.
   (3) $A(B \cap C) = AB \cap AC$.
   (4) $(A + B)^\star = (A^\star B^\star)^\star$.

On définit à présent la racine carrée d'un langage L sur un alphabet Σ par

$$\sqrt{L} = \{u \in \Sigma^\star \mid u^2 \in L\}$$

1. Déterminer quelles sont les inclusions vraies parmi les suivantes :

   (1) $L \subset \sqrt{L^2}$.
   (2) $\sqrt{L^2} \subset L$.
   (3) $L \subset (\sqrt{L})^2$.
   (4) $(\sqrt{L})^2 \subset L$.
   (5) $(\sqrt{L})^2 \subset \sqrt{L^2}$
   (6) $\sqrt{L^2} \subset (\sqrt{L})^2$.

Voir aussi :

- L'exercice 14 CCINP (langages rationnels et automates finis).
- L'exercice 16 CCINP (grammaires algébriques).

Jeux et IA

Exercice 17 *Jeux et couplages*

Le jeu de Slither est un jeu à deux joueurs qui se joue sur un graphe connexe. Tour à tour, chaque joueur doit choisir un sommet de G qui n'a jamais été choisi par aucun des joueurs et qui est voisin de celui choisi précédemment (autrement, dit, la suite de sommets choisis doit former un chemin élémentaire dans G). Le premier joueur qui ne peut plus jouer a perdu.

Montrer que si G admet un couplage parfait alors le second joueur a une stratégie gagnante.

Exercice 18 *Modélisation d'un jeu à deux joueurs*

On considère un jeu dont le support est une rangée de n bâtonnets. Tour à tour, chaque joueur enlève de la rangée 1, 2 ou 3 bâtonnets. Le perdant est celui qui prend le dernier bâtonnet.

1. Dessiner le graphe des configurations de ce jeu lorsque $n = 7$.
2. Donner une stratégie gagnante pour le premier joueur dans ce cas en justifiant.
3. Montrer que les configurations où le nombre n de bâtonnets est congru à 1 modulo 4 sont perdantes. Si $n = 20$, vaut-il mieux jouer en premier ou en deuxième ? Et si $n = 21$?

Voir aussi l'exercice 20 CCINP (apprentissage avec ID3).

Autres

Exercice 19 *Représentation de flottants*

1. On représente des réels sur 8 bits avec 3 bits d'exposant et 4 de mantisse. Le flottant 01101101 est-il normalisé ? Que vaut-il ? Même question pour le flottant 00001000.
2. Donner la valeur réelle des expressions suivantes et indiquer si un ordinateur fournirait les mêmes réponses : $\sum_{n=1}^{+\infty} \frac{1}{n}$ et $(10^{50} + 1) - 10^{50}$. Expliquer.

Voir aussi :

- L'exercice 13 CCINP (problèmes NP-complets).
- L'exercice 15 CCINP (concurrence).
- L'exercice 18 CCINP (SQL).