II - Mines-Telecom Sujets 0

II.1 - Planche 0

Exercice 1

Exercice 1

cf annexe pour un rappel des règles de la déduction naturelle

1.1 Prouver le séquent $A \wedge B \vdash B \wedge A$

1.2 Prouver le séquent $A \wedge (B \vee C) \vdash (A \wedge B) \vee (A \wedge C)$

Soit $\Gamma \vdash C$ un séquent prouvable à l'aide d'un arbre de preuve Π.

1.3 Montrer qu'il existe un arbre de preuve de $\Gamma \vdash C$ tel que cet arbre ne possède pas la succession de la règle élimination de la conjonction puis introduction de la conjonction.

1.4 Que peut-on dire pour les successions de règles de disjonction ?

1.5 Prouver le séquent $\vdash A \vee \neg A$.

Solution de l'Exercice 1

1.

$$\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,(\text{Ax})}{A \wedge B \vdash B}\,(\wedge_e^d) \qquad \dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,(\text{Ax})}{A \wedge B \vdash A}\,(\wedge_e^g)}{A \wedge B \vdash B \wedge A}\,\wedge_i$$

2. Posons $P = (A \wedge B) \vee (A \wedge C)$.

$$\dfrac{\dfrac{\dfrac{}{A \wedge (B \vee C) \vdash A \wedge (B \vee C)}\,(\text{ax})}{A \wedge (B \vee C) \vdash B \vee C}\,\wedge_e^d \qquad \dfrac{}{A \wedge (B \vee C), B \vdash P}\,(\text{lemme}) \qquad \dfrac{}{A \wedge (B \vee C), C \vdash P}\,(\text{lemme})}{A \wedge (B \vee C) \vdash P = (A \wedge B) \vee (A \wedge C)}\,\vee_e$$

Où le lemme est le suivant (symétrique pour B et C, je donne les deux ci-dessous) :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (B \vee C), B \vdash A \wedge (B \vee C)}\,(\text{ax})}{A \wedge (B \vee C), B \vdash A}\,\wedge_e^g \qquad \dfrac{}{A \wedge (B \vee C), B \vdash B}\,(\text{ax})}{A \wedge (B \vee C), B \vdash (A \wedge B)}\,\wedge_i}{A \wedge (B \vee C), B \vdash (A \wedge B) \vee (A \wedge C)}\,\vee_i^g$$

et

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (B \vee C), C \vdash A \wedge (B \vee C)}\,(\text{ax})}{A \wedge (B \vee C), C \vdash A}\,\wedge_e^g \qquad \dfrac{}{A \wedge (B \vee C), C \vdash C}\,(\text{ax})}{A \wedge (B \vee C), C \vdash (A \wedge C)}\,\wedge_i}{A \wedge (B \vee C), C \vdash (A \wedge B) \vee (A \wedge C)}\,\vee_i^d$$

3. Si l'arbre de preuve Π contient la succession de règle donnée, alors Π contient par exemple (on raisonnerait de même avec l'élimination droite) :

$$\dfrac{\dfrac{\dfrac{\overline{\Pi_A}}{\Gamma' \vdash A} \qquad \dfrac{\overline{\Pi_B}}{\Gamma' \vdash B}}{\Gamma' \vdash A \wedge B}\,\wedge_i}{\Gamma' \vdash A}\,\wedge_e^g$$

Dans ce cas, on peut remplacer cette portion de l'arbre par la simple preuve suivante :

$$\dfrac{\overline{\Pi_A}}{\Gamma' \vdash A}$$

4. On peut aussi dire qu'on n'aura pas une succession d'élimination et d'introduction de la conjonction (de bas en haut dans l'arbe), mais le raisonnement est un peu plus complexe :

$$\dfrac{\dfrac{\dfrac{\overline{\Pi_A}}{\Gamma' \vdash A}}{\Gamma' \vdash A \vee B} \vee_i^g \quad \dfrac{\overline{\Pi'}}{\Gamma', A \vdash C} \quad \dfrac{}{\Gamma', B \vdash C}}{\Gamma' \vdash C} \vee_e$$

Et on peut remplacer cet arbre par le suivant :

$$\dfrac{\dfrac{\overline{\Pi'}}{\Gamma', A \vdash C}}{\Gamma' \vdash A \to C} \to_i \quad \dfrac{\overline{\Pi_A}}{\Gamma' \vdash A}}{\Gamma' \vdash C} \to_e$$

Mais cela n'a que peu d'intérêt, on retrouve une succession d'élimination et d'introduction, mais pour l'implication à la place.

5. Dérivons le séquent suivant : $\neg(A \vee \neg A) \vdash \neg A$.

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\,(\text{Ax})}{A \vdash A \vee \neg A}\,(\vee_i) \quad \dfrac{}{\neg(A \vee \neg A) \vdash \neg(A \vee \neg A)}\,(\text{Ax})}{\dfrac{\neg(A \vee \neg A), A \vdash \bot}{\neg(A \vee \neg A) \vdash \neg A}\,(\neg_i)}\,(\neg_e)$$

En utilisant exactement la même méthode, on pourrait obtenir $\neg(A \vee \neg A) \vdash \neg\neg A$. Cependant, en remplaçant la dernière règle par un (RAA), on peut obtenir $\neg(A \vee \neg A) \vdash A$:

$$\dfrac{\dfrac{\dfrac{}{\neg A \vdash \neg A}\,(\text{Ax})}{\neg A \vdash A \vee \neg A}\,(\vee_i) \quad \dfrac{}{\neg(A \vee \neg A) \vdash \neg(A \vee \neg A)}\,(\text{Ax})}{\dfrac{\neg(A \vee \neg A), \neg A \vdash \bot}{\neg(A \vee \neg A) \vdash A}\,(\text{RAA})}\,(\neg_e)$$

On combine ensuite les deux résultats :

$$\dfrac{\dfrac{}{\neg(A \vee \neg A) \vdash \neg A}\,\text{Lemme 1} \quad \dfrac{}{\neg(A \vee \neg A) \vdash A}\,\text{Lemme 2}}{\dfrac{\neg(A \vee \neg A) \vdash \bot}{\vdash A \vee \neg A}\,(\text{RAA})}\,(\neg_e)$$

Exercice 2

Exercice 2

On s'intéresse à un bus touristique pouvant contenir C passagers.
On suppose que l'on a $n > C$ passagers qui attendent leur tour, puis se remettent en attente à l'arrêt du bus dès qu'ils ont fini pour le revoir.
Le bus ne démarre que lorsqu'il est plein.
Pour formaliser ce problème, on utilise des fonctions fictives :

- `board` et `unboard` permettent au passager de monter et de descendre ;
- le bus doit appeler les fonctions `load` lorsqu'il se remplit, `run` lorsqu'il démarre son tour et `unload` lorsqu'il se vide.

Attention, les passagers ne peuvent pas descendre avant que le bus ait ouvert ses portes pour une fin de tour avec `unload` et ne peuvent pas monter avant que le bus ait ouvert ses portes pour un nouveau tour avec `load`.

2.1 Écrire les fonctions en pseudo-code correspondant au bus et aux passagers **sans** prendre en compte les problèmes de synchronisation dans un premier temps.

2.2 Proposer une solution en pseudo-code utilisant deux compteurs protégés par des mutexs et quatre sémaphores.

2.3 Votre solution peut-elle être utilisée dans le cas où l'on a plusieurs bus ? Justifier.

2.4 Proposer une nouvelle solution pour le pseudo-code du bus dans le cas où il y a m bus numérotés en respectant les règles suivantes :

- un seul bus peut ouvrir ses portes aux passagers à la fois ;
- plusieurs bus peuvent faire un tour en même temps ;
- les bus ne peuvent pas se doubler donc ils se chargent et se déchargent toujours dans le même ordre
- un bus doit avoir fini de décharger avant qu'un autre bus vienne décharger.

La solution doit utiliser deux tableaux de sémaphores en plus des sémaphores utilisés dans la solution précédente, permettant de gérer la coordination entre les bus.

Solution de l'Exercice 2

1. Je ne suis pas sure de comprendre ce que les passagers sont censés faire sans synchronisation. J'imagine que c'est le bus qui appellera les fonctions des passagers ? En tout cas, si on ne fait aucun effort de synchronisation, on propose le code suivant :

Pseudo-code

```
1  PASSAGER() :
2      Tantque true Faire :
3          load()
4          run()
5          unload()
6
7  BUS() :
8      Tantque true Faire :
9          board()
10         unboard()
```

2. On propose une solution avec 4 sémaphores et un compteur. Il n'est pas nécessaire de protéger le compteur par un mutex, car un seul passager peut acquérir un sémaphore loading ou unloading à la fois.

Pseudo-code

```
1  loading = semaphore(0)
2  loaded = semaphore(0)
3  unloading = semaphore(0)
4  unloaded = semaphore(0)
5  cnt = 0
```

On obtient alors pour le bus :

Pseudo-code

```
1  LOAD() :
2      signal(loading)
3      wait(loaded)
4
5  UNLOAD() :
6      signal(unloading)
7      wait(unloaded)
```

Et pour un passager :

Pseudo-code

```
BOARD() :
    wait(loading)
    cnt = cnt + 1
    Si cnt == C Alors :
        signal(loaded)
    Sinon :
        signal(loading)

UNBOARD() :
    wait(unloading)
    cnt = cnt - 1
    Si cnt == 0 Alors :
        signal(unloaded)
    Sinon :
        signal(unloading)
```

3. On ne peut pas directement utiliser cette solution pour plusieurs bus, car on n'aurait aucune maîtrise du bus dans lequel un passager monte, plusieurs bus pourraient charger et décharger en même temps.
4. **Remarque :** Sans la contrainte $m \times C \leqslant n$, il peut y avoir interblocage. En effet, il y a moins de passagers que de places dans les bus. Ainsi, une fois tous les passagers chargés, le dernier bus bloque l'arrêt de bus parce qu'il n'est pas plein, et aucun autre ne peut décharger tant qu'il n'a pas terminé.

   L'idée est que chaque bus doit attendre que le précédent ait terminé une action avant de pouvoir la faire à son tour. On utilise des tableaux de sémaphores au lieu des sémaphores précédents (sauf loading, pour qu'un passager puisse être au courant qu'un bus charge, quel que soit son numéro).

   Par ailleurs, chaque passager dispose d'un numéro de bus, pour savoir dans quel bus il est monté. En effet, un passager peut monter dans n'importe quel bus, mais il ne peut descendre que du bus dans lequel il est monté.

   Enfin, on garde en mémoire un numéro global de bus permettant de savoir quel est le bus courant autorisé à l'arrêt de bus, ainsi qu'un tableau de sémaphores permettant de gérer l'accès des bus à l'arrêt.

Pseudo-code

```
loading = semaphore(0)
loaded[m] = [semaphore(0), ...]
unloading[m] = [semaphore(0), ...]
unloaded[m] = [semaphore(0), ...]
stop[m] = [semaphore(0), ...]
signal(stop[0])
cnt = 0
current_bus = 0

BUS(i):
    wait(stop[i])
    Tantque true Faire :
        load(i)
        current_bus = (i + 1) modulo m
        signal(stop[current_bus])
        run()
        wait(stop[i])
        unload(i)

LOAD(i):
    signal(loading)
    wait(loaded[i])

UNLOAD(i):
    signal(unloading[i])
    wait(unloaded[i])
```

Pour un passager :

Pseudo-code

```
PASSENGER() :
    Tantque true Faire :
        b = board()
        unboard(b)

BOARD() :
    wait(loading)
    b = current_bus
    cnt = cnt + 1
    Si cnt modulo C == 0 Alors :
        signal(loaded[b])
    Sinon :
        signal(loading)
    Renvoyer(b)

UNBOARD(b) :
    wait(unloading[b])
    cnt = cnt - 1
    Si cnt modulo C == 0 Alors :
        signal(unloaded[b])
    Sinon :
        signal(unloading[b])
```

L'utilisation du modulo facilite l'écriture, car les chargements et déchargements se font par vagues de C passagers de toute façon.

Annexe 1 - Sujet 0

Rappel des règles de la déduction naturelle

Les arbres de preuves doivent être effectués à partir de l'ensemble de règles fourni ci-dessous.

$$\frac{}{\Gamma, A \vdash A}\,\text{ax} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\,\text{aff} \qquad \frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\,\text{RAA}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,{\to_i} \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\,{\to_e}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,{\wedge_i} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,{\wedge_e^g} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,{\wedge_e^d}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,{\vee_i^g} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,{\vee_i^d} \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,{\vee_e}$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\,{\neg_i} \qquad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \bot}\,{\neg_e}$$

II.2 - Planche 1

Exercice 3

Exercice 1

Soit G un graphe non-orienté à $n \geq 1$ sommets et p arêtes.

1.1 Montrer que si G est connexe alors $p \geq n-1$.

1.2 Montrer que si G est acyclique alors il possède un sommet de degré au plus 1.

1.3 Montrer que si G est acyclique alors $p \leq n-1$.

1.4 Montrer que les trois propriétés suivantes sont équivalentes :

(i) G est connexe et acyclique.

(ii) G est connexe et $p = n-1$.

(iii) G est acyclique et $p = n-1$.

Solution de l'Exercice 3

Toutes ces questions sont des preuves du cours de 1ère année.

1. Par récurrence sur l'ordre n du graphe.

   - Initialisation : Pour $n = 1$, le graphe est réduit à un sommet. Il a donc bien au moins $1-1 = 0$ arêtes.
   - Hérédité : Supposons la propriété vraie jusqu'au rang n, montrons-la vraie au rang $n+1$. On distingue deux cas :

     – S'il existe un sommet de degré 1 : nommons-le u et nommons v son unique voisin. Notons $S' = S \backslash \{u\}$ et $A' = A \backslash \{uv\}$. Le graphe G' induit par S' est connexe.
     [FAIRE UN SCHÉMA.]
     ((Preuve : Soient x et y dans G' (donc distincts de u). Par connexité de G, il existe un chemin élémentaire de x à y dans G. Ce chemin étant élémentaire, il ne peut pas passer par u : en effet, u ayant un seul voisin v, si le chemin passait par u il serait de la forme $x, ..., v, u, v, ..., y$ et ne serait pas élémentaire. Mais donc ce chemin élémentaire n'emprunte jamais une arête dont u est une extrémité, c'est à dire qu'il n'utilise que des arêtes de G'. Donc il existe un chemin de x à y dans G', donc G' est connexe.))
     Remarque : Dans un oral, on peut sans doute se contenter d'affirmer que G' est connexe à partir du dessin.

     Enfin, $|S'| = n < n+1$ donc par hypothèse de récurrence $|A'| \geq n-1$. Or $|A| = |A'|+1$, donc $|A| \geq n = (n+1)-1$. D'où l'hérédité dans ce premier cas.

     – Sinon il n'y a pas de sommets de degré 1. Puisque le graphe est connexe il n'y a pas de sommets de degré 0 : donc le degré minimal est 2.
     Or, on sait que dans un graphe non-orienté :

     $$\sum_{s \in S} deg(s) = 2|A|$$

     Donc :

     $$|A| = \frac{1}{2} \sum_{s \in S} deg(s) \geq \frac{1}{2} \sum_{s \in S} 2 \geq |S|$$

     On a bien au moins $n-1$ arêtes. D'où l'hérédité dans ce second cas.

   - Conclusion : On a prouvé par récurrence sur n qu'un graphe connexe à n sommets a au moins $n-1$ arêtes.

2. On raisonne par contraposée. Montrons que si tous les sommets de G sont de degré au moins 2, alors il existe un cycle dans G.
   Notons $G = (S, A)$, on suppose donc que $\forall s \in S, deg(s) \geqslant 2$. On définit une suite $(s_i)_i \in S^{\mathbb{N}}$ de sommets du graphe de la manière suivante :

- $s_0 \in S$ (existe car $n \geqslant 1$). s_1 est un voisin de s_0 (quelconque).
- $forall i \geqslant 2$, on pose s_{i+1} un voisin de s_i différent de s_{i-1} (possible car $\deg(s_i) \geqslant 2$).

Comme S est fini, il existe des sommets qui se répètent dans la suite. Soit s_j le premier sommet apparaissant une deuxième fois dans la suite, et $j' < j$ son premier indice d'apparition (faire un schéma). Alors $s_{j'}, s_{j'+1}, ..., s_j = s_{j'}$ est un cycle. Montrons en effet que ce chemin est un chemin simple. Notons que $s_{j'}, s_{j'+1}, ..., s_{j-1}$ est un chemin élémentaire, donc simple (par minimalité de j). Donc si une arête est parcourue deux fois dans ce chemin, ça ne peut être que l'arête $s_{j-1} - s_j$. Or par minimalité de j, s_{j-1} n'apparaît qu'une seule fois, donc le chemin considéré est exactement $s_{j'} - s_{j'+1} - s_{j'+2}$, ce qui contredit la définition de $s_{j'+2}$.

3. Par récurrence sur l'ordre $n \geqslant 1$ du graphe.

- **Initialisation (n=1) :** 1 sommet et 0 arêtes : ok.
- **Hérédité :** Par la question précédente, il existe $s_0 \in S$ tel que $\deg(s_0) \leqslant 1$. Le sous-graphe de G induit par $S \backslash \{s_0\}$ est acyclique d'ordre $n-1$ (un cycle dans ce sous-graphe serait un cycle dans G), donc par HR il possède au plus $n-2$ arêtes.
  Comme $\deg(s_0) \leqslant 1$, on a supprimé au plus une arête en passant au sous-graphe induit. Donc G possède au plus $n-1$ arêtes.

4. On montre les implications suivantes :

- **(i) => (ii)** et **(i) => (iii)** : Si G est connexe acyclique, alors par les questions précédentes, il vérifie à la fois $p \leqslant n-1$ et $p \geqslant n-1$, donc $p = n-1$. D'où (ii) et (iii).
- **(ii) => (i)** : Supposons G connexe à $p = n-1$ arêtes. Supposons par l'absurde que G possède un cycle $s_0, s_1, ..., s_k = s_0$. Alors supprimer l'arête $s_0 s_1$ ne déconnecte pas le graphe, car on peut la remplacer par $s_0 = s_k, s_{k-1}, ..., s_1$ dans un chemin.
  Donc $G' = (S, A \backslash s_0 s_1)$ est un graphe connexe d'ordre n possédant $p = n-2 < n-1$ arêtes : absurde d'après les questions précédentes. Donc G est acyclique.
- **(iii) => (i)** : Supposons G acyclique à $n-1$ arêtes. Notons r le nombre de composantes connexes de G, et $n_1, ..., n_r$ leurs nombres de sommets respectifs. On a $n = \sum_{i=1}^{r} n_i$ car les composantes connexes partitionnent les sommets du graphe.
  Chaque composante connexe est acyclique (car G est acyclique) donc chacune possède $n_i - 1$ arêtes d'après le premier point de cette preuve. Ainsi G possède $\sum_{i=1}^{r}(n_i - 1) = \sum_{i=1}^{r} n_i - r = n - r$ arêtes. Or G possède $n-1$ arêtes donc G possède exactement 1 composante connexe, i.e. G est connexe.

Exercice 4

Exercice 2

On considère le problème suivant :

Définition 1

SUBSET-SUM :

Entrée : Un tableau $T = [t_1, \dots, t_n]$ de n entiers positifs et un entier c.

Sortie : La valeur maximum de $\sum_{i \in I} t_i$ où $I \subseteq \{1, \dots, n\}$ et $\sum_{i \in I} t_i \leq c$.

2.1 Décrire un algorithme naïf qui résout ce problème et préciser sa complexité.

2.2 On note $s_{i,j}$ la plus grande somme inférieure à j que l'on peut obtenir avec des éléments $t_1, \dots, t_i$.
Donner une équation de récurrence pour $s_{i,j}$ et en déduire un algorithme pour SUBSET-SUM. Comparer sa complexité avec l'algorithme précédent.

On considère l'algorithme glouton suivant :

```
1 Trier les éléments de T par ordre décroissant.
2 S ← 0
3 pour i de 1 à n faire
4 |  si S + t_i ≤ c alors
5 |  |  S ← S + t_i
6 renvoyer S
```

2.3 Donner la complexité de cet algorithme.

2.4 Montrer que l'algorithme glouton donne une $\frac{1}{2}$-approximation de la solution optimale.

2.5 Soit $\alpha \in]\frac{1}{2}, 1]$. Donner une instance de SUBSET-SUM telle que la somme S renvoyée par l'algorithme glouton vérifie $S \leq \alpha S^*$ où S^* est la solution optimale.

Solution de l'Exercice 4

1. Pour un algorithme complètement naïf, on propose :

Pseudo-code

```
SUBSET-SUM(T, c) :
    maxi ← 0 //ensemble vide
    Pour chaque sous-ensemble I ⊆ {1,...,n} Faire :
        Si Σ_{i∈I} t_i < maxi alors :
            maxi ← Σ_{i∈I} t_i
    Renvoyer maxi
```

où on peut implémenter le parcours sur les sous-ensemble par un compteur binaire, par exemple.

On effectue 2^n itérations de la boucle Pour, et pour chaque itération on calcule une somme de n élément en temps $O(n)$. La mise à jour de l'ensemble I peut se faire en temps constant amorti par un compteur binaire (et ne dépassera pas O(n)). On obtient donc une complexité temporelle totale en $O(n.2^n)$.

2. On suppose qu'on a bien $i, j \geqslant 0$. On a :

$$s_{i,j} = \begin{cases} 0 & \text{si i = 0 (et } j \geqslant 0) \\ \max(s_{i-1,j-t_i} + t_i, s_{i-1,j}) & \text{si } t_i \leqslant j \\ s_{i-1,j} & \text{sinon} \end{cases}$$

On cherche alors à calculer $s_{n,c}$.
Si on se contente d'un algorithme récursif naïf, on a toujours une complexité exponentielle, en $O(2^n)$, même si on gagne un facteur n. Cependant, on peut mémoïser les résultats (programmation dynamique) pour améliorer ce coût.
Avec l'approche ascendante, par exemple, on remplit un tableau de dimensions $n \times c$, et chaque case se calcule en temps constant $O(1)$, donc on obtient une complexité $O(nc)$.
Remarque : SUBSET-SUM est NP-complet, et cette complexité ne vient pas contredire ce résultat. En effet, la taille de l'entier c (qui fait partie des entrées) est $\log(c)$, la complexité ci-dessus est donc toujours une complexité exponentielle.

3. Trier les éléments se fait en temps $O(n \log n)$ par un tri par comparaison (par exemple, tri par tas).
Le reste est en $O(n)$, on a n itérations qui se font chacune en temps constant. La complexité totale de cet algorithme est donc $O(n)$.

4. On suppose que tous les éléments de T sont plus petits que c, sinon on les retire (ils ne participent ni à la somme du glouton, ni à l'optimal). On suppose de plus qu'il reste au moins deux éléments dans T sinon l'optimal et glouton coïncident, soit sur l'ensemble vide, soit sur l'unique élément restant.

Soit $T_{opt} \subset T$ réalisant la somme optimale $S_{opt} = \sum_{t \in T_{opt}} t \leqslant c$. Montrons que l'algorithme glouton renvoie une somme $S \geqslant \frac{1}{2} S_{opt}$.

- Si $\sum_{t \ni T} t \leqslant c$ alors l'algorithme glouton ajoute successivement tous les éléments (aucun ne fait dépasser), et on renvoie l'optimal, à savoir $\sum_{t \in T} t$.

— Sinon, montrons que la somme S renvoyée par le glouton vérifie $S \geqslant \frac{c}{2}$, et donc $S \geqslant \frac{S_{opt}}{2}$ (car $S_{opt} \leqslant c$).
Notons i_0 l'indice du premier élément qui nous fait dépasser c, en les sommant dans l'ordre décroissant.
Un tel élément existe car $\sum_{t \in T} t > c$. Plus formellement, $i_0 = \min\{i \mid \sum_{k=0}^{i} t_k > c\}$.
En particulier, le glouton prend tous les t_i dans sa somme S jusqu'à atteindre i_0 et il en prend au moins 1 ($i_0 > 1$ car $t_1 \leqslant c$, hypothèse de départ). L'idée est alors que le t_{i_0} qui nous fait dépasser c est moins grand que ce qu'on a déjà pris, et donc on a déjà pris au moins $\frac{c}{2}$.
Formalisons ça :
$S \geqslant \sum_{i=1}^{i_0-1} t_i$
d'où $2S \geqslant \sum_{i=1}^{i_0-1} t_i + t_1 \geqslant \sum_{i=1}^{i_0-1} t_i + t_{i_0}$ par décroissance des t_i $= \sum_{i=0}^{i_0} > c$.
Donc $2S > c$ d'où $S > \frac{c}{2}$.

5. Soit $c > 2$. On prend $T = \{\frac{c}{2} + 1, \frac{c}{2}, \frac{c}{2}\}$. L'optimal est vérifié en prenant les deux derniers éléments et $S^* = c$. L'algorithme glouton prend uniquement le premier, et on a $S = \frac{c}{2} + 1$.
Donc

$$\frac{S}{S^*} = \frac{\frac{c}{2} + 1}{c} = \frac{1}{2} + \frac{1}{c} \underset{c \to +\infty}{\longrightarrow} \frac{1}{2}$$

Donc, à partir d'une certaine valeur de c, on a bien $\frac{S}{S^*} \leqslant \alpha$, i.e. $S \leqslant \alpha S^*$, car $\alpha > \frac{1}{2}$ (on pourrait même calculer cette valeur, mais c'est inutile).

II.3 - Planche 2

Exercice 5

Exercice 1

Une entreprise de livraison de nourriture dispose d'une base de donnée pour représenter ses clients et les commandes passées. On présente le schéma relationnel correspondant.

Clients(id : `int`, id_addresse : `int`, nom : `text`, prenom : `text`)

Adresses(id : `int`, ville : `text`, nom_rue : `text`, numero : `int`)

Commandes(id_client : `int`, plat : `text`, prix : `int`, date : `date`)

L'entreprise souhaite ajouter une fonctionnalité pour vérifier qu'un client n'est pas allergique à un plat qu'il commande. Pour cela, il est nécessaire de pouvoir enregistrer les allergènes présents dans chaque plat ainsi que les allergies de chaque client.

1.1 Proposer des modifications à notre schéma de relation pour pouvoir ajouter ces informations. Justifier pourquoi votre choix convient.

Dans la suite, on se base dans le modèle tel que vous l'avez modifié.

1.2 Écrire une requête pour trouver les noms et prénoms des clients étant allergiques à au moins un ingrédient d'une pizza.

1.3 Écrire une requête pour trouver les trois villes où le plus d'argent est dépensé, ainsi que cette somme totale.

1.4 Écrire une requête pour effectuer la moyenne d'argent dépensé en fonction des prénoms des clients.

Solution de l'Exercice 5

1. On propose de rajouter deux tables représentant respectivement les allergies des clients et les allergènes des plats :

   Allergies(id_client : `int`, aliment : `text`)

   Allergenes(plat : `text`, aliment : `text`)

   On pourrait plutôt vouloir ajouter des attributs aux tables existantes, mais il faudrait une nouvelle ligne par allergie différente de chaque client, ce qui dupliquerait les adresses, noms et prénoms.

2. On propose :

Pseudo-code
```
SELECT DISTINCT noms, prenoms FROM Clients AS C
    JOIN Allergie AS A1 ON C.id = A1.id_client
    JOIN Allergene AS A2 ON A1.aliment = A2.aliment
WHERE plat = 'pizza'
```

3. On propose :

Pseudo-code
```
SELECT ville, SUM(prix) FROM Adresse AS Ad
    JOIN Clients As Cl ON Cl.id_addresse = Ad.id
    JOIN Commandes AS C ON Cl.id = C.id_client
GROUP BY ville
ORDER BY SUM(prix) DESC LIMIT 3
```

4. On propose :

Pseudo-code
```
SELECT prenom, AVG(S) FROM
    (SELECT id_client, prenom, SUM(prix) AS S FROM Adresse AS Ad
        JOIN Clients As Cl ON Cl.id_addresse = Ad.id
        JOIN Commandes AS C ON Cl.id = C.id_client
    GROUP BY id_client)
GROUP BY prenom
```

   Attention à ne pas faire directement la moyenne sur les prénoms, où vous autre un montant moyen de commande par prénom, et non un montant moyen de **somme** dépensée, par prénom.

Exercice 6

Exercice 2

Dans cet exercice, on ne considère que des graphes orientés. Soit $G = (S, A)$ un graphe, on appelle clôture transitive d'un graphe, qu'on note $G^* = (S, A^*)$, le graphe ayant les mêmes sommets que S et tel que (x, y) soit une arête de G^* si et seulement si il existe un chemin de x à y dans G. On note $|S| = n$.

2.1 Justifier que la relation $\mathcal{R}$ définie par $x\mathcal{R}y$ si et seulement si (x, y) appartient à A^* est transitive.

2.2 On suppose que A est donné sous la forme d'une matrice d'adjacence. Donner un algorithme pour calculer A^* en $O(n^3)$ opérations.

Une réduction transitive d'un graphe $G = (S, A)$ est un sous-graphe $G_R = (S, A_R)$ avec un nombre minimum d'arêtes tel que $G^* = G_R^*$.

2.3 Montrer que dans le cas général, une réduction transitive n'est pas unique.

Dans toute la suite, on ne considérera plus que des graphes acyclique.

Considérons pour tout x, y dans S $A_{x,y}$ définie par :

$$A_{x,y} = \{e \in A \mid e \text{ appartient à un chemin de longueur maximal entre } x \text{ et } y\}$$

De plus on considère

$$A' = \bigcup_{x,y \in S} A_{x,y}$$

2.4 Montrer l'égalité suivante :

$$A' = \{(x, y) \in A \mid \text{ la longueur du plus long chemin entre } x \text{ et } y \text{ est } 1\}$$

2.5 En déduire que $G' = (S, A')$ est l'unique réduction transitive de G.

2.6 Donner un algorithme pour calculer l'unique réduction transitive d'un graphe acyclique.

Solution de l'Exercice 6

1. Soient $x, y, z \in S$ tels que $S\mathcal{R}y$ et y. Alors par définition (comme $(x, y) \in A^*$), il existe un chemin de x à y dans G. Notons le $x = s_0, ..., s_k = y$. De même, il existe un chemin de y à z dans G. Notons le $y = s'_0, ..., s'_l = z$.
   Donc il existe un chemin de x à z dans G, obtenu en concaténant ces deux chemins : $x = s_0, ..., s_k, s'_1, ..., s'_l = z$ (si $s'_0 = s'_l = z$, on s'arrête à $s_k = y = s'_0 = z$).
2. On peut simplement adapter l'algorithme de Floyd-Warshall. Au lieu de chercher des distances minimales, on cherche simplement à vérifier l'accessibilité. La matrice k représente par un coef 1 les paires de sommets accessibles par un chemin utilisant les sommets intermédiaires 1 à k (et 0 sinon). Donc on calcule les matrices successives (M_k) définies par :
   $$\begin{cases} M_0 = A \text{ matrice d'adjacence} \\ M_{k+1} \text{ est définie par ses coefficients } m_{i,j}^{k+1} = \begin{cases} 1 \text{ si } m_{i,j} = 1 \text{ ou } m_{i,k} = m_{k,j} = 1 \\ 0 \text{ sinon} \end{cases} \end{cases}$$
   Chaque matrice M_i se calcule en temps $O(n^2)$ (temps constant pour chaque coefficient), et on cherche à renvoyer la matrice $M_n = A^*$ (sous forme de matrice d'adjacence). Le temps total est donc en $O(n^3)$ comme demandé.

   Remarque : on peut aussi lancer un parcours depuis chaque sommet, chaque parcours est en $O(n^2)$ par matrice d'adjacence et on en fait n, on met à jour les voisins de s au fur et à mesure : sommet visités lors du parcours depuis s. Mais le Floyd-Warshall me semble plus simple à coder.
3. Il suffit de donner un exemple. Prenons par exemple le graphe constitué de 3 sommets x_1, x_2, x_3 et possédant toutes les arêtes possibles (dans les deux sens possibles). On peut prendre comme réduction transitive les cycles $x_1 \to x_2 \to x_3 \to x_1$ ou $x_1 \to x_3 \to x_2 \to x_1$ qui contiennent chacune trois arêtes différentes de A.

Ce nombre d'arêtes est bien minimal car le graphe doit rester fortement connexe, sinon certains sommets ne seront plus accessibles et on n'aura plus G^* le graphe contenant toutes les arêtes.

4. Raisonnons par double inclusion :

$\subseteq$: Soit $e = (x, y) \in A'$, alors il existe $z, t \in S$ tels que $e \in A_{z,t}$. Donc $e \in A$ et e appartient à un chemin de longueur maximale entre z et t, notons c un tel chemin. Montrons alors que la longueur du plus long chemin entre x et y est 1. En effet, si ce n'était pas le cas, comme on sait qu'il existe un chemin de longueur 1 (donné par e), cela signifie que la longueur d'un plus long chemin entre x et y est > 1 (notons c' un tel chemin). Mais alors en remplaçant e par ce chemin c' dans c, on obtient un chemin strictement plus long entre z et t, ce qui contredit la maximalité du chemin c.

$\supseteq$: Réciproquement, soit $e = (x, y) \in A$ tel que le chemin de longueur maximal entre x et y est e (de longueur 1). Alors on a $e \in A_{x,y}$ donc $e \in A$'.

5. Soit $G_R = (S, A_R)$ une réduction transitive (quelconque) de G. Montrons que $G_R = G'$, i.e. que $A_R = A'$. Par double inclusion à nouveau :

$A_R \subseteq A'$: Soit $(x, y) \in A$ une arête de G_R (sous-graphe de G). Alors il existe un chemin de longueur 1 entre x et y, donné par (x, y). S'il existait un chemin de longueur > 1 entre x et y, alors on pourrait enlever l'arête (x, y) dans A_R et obtenir une nouvelle réduction transitive (on ne casse aucune accessibilité, on peut remplacer cette arête par l'autre chemin), ce qui contredit la minimalité de A_R. En effet, cet autre chemin ne contient pas l'arête (x, y), sinon on aurait un cycle contenant x ou y. Donc par la question précédente, $(x, y) \in A'$.

$A' \subseteq A_R$: Soit $(x, y) \in A'$. Alors le seul chemin entre x et y est celui passant par (x, y), sinon on obtient un chemin de longueur au moins 2 entre x et y. Or il existe un chemin entre x et y dans G et $G^* = G_R^*$ donc il existe un chemin entre x et y dans G_R. Donc G_R contient l'arête $(x, y) \in A_R$.

6. On donne un algorithme pour calculer G'.
On peut par exemple de manière naïve, pour chaque arête (x, y) on l'enlève et on teste l'accessibilité de y depuis x, ceci se fait en temps $O(n^4)$ (on parcourt les arêtes en temps $O(n^2)$ et chacune d'entre elle nécessite un parcours en temps $O(n^2)$ via matrice d'adjacence).

On peut proposer un meilleur algorithme en $O(n^3)$ grâce aux résultats précédents de l'exercice. On calcule A^* la matrice dont le coefficient (i, j) vaut 1 si j est accessible depuis i, en temps $O(n^3)$. Ensuite, la multiplication de matrice $M = A.A^*$, calculée en temps $O(n^3)$, vérifie la propriété suivante :

$$\begin{cases} m_{i,j} > 0 \text{ s'il existe un chemin entre i et j de longueur au moins 2} \\ 0 \text{ sinon (il n'existe qu'une arête ou aucun chemin entre i et j)} \end{cases}$$

En effet, $m_{i,j} > 0$ si et seulement si il existe k tel que $A_{i,k} = 1$ et $A^*_{k,j} = 1$, i.e. il existe une arête de i à k et un chemin de k à j dans G, c'est à dire qu'il existe un chemin passant par un sommet tiers k, donc de longueur au moins 2 entre i et j.

On calcule alors A' en vérifiant, pour chaque paire (i, j) de sommets, s'il existe une arête entre i et j ($A_{i,j} = 1$) et il n'existe pas de chemin entre i et j de longueur au moins 2 ($M_{i,j} = 0$).

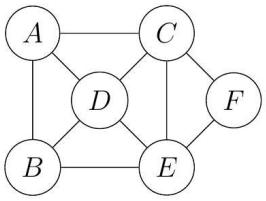
II.4 - Planche 3

Exercice 7

Exercice 1

1.1 Rappeler la définition de couplage dans un graphe non orienté. Quand peut-on qualifier un couplage de maximal ? maximum ? parfait ?

1.2 Donner un couplage maximum dans le graphe suivant. Donner un couplage maximal qui n'est pas maximum.



1.3 Qu'est ce qu'un chemin augmentant alternant pour un graphe $G = (S, A)$ et un couplage $C \subset A$ sur ce graphe ?

1.4 Soit $G = (S, A)$ un graphe. Montrer qu'un couplage est maximum si et seulement s'il n'existe pas de chemin augmentant alternant pour ce couplage dans le graphe G.

Solution de l'Exercice 7

Cet exercice est constitué d'une portion du cours sur les couplages (cf cours !!!).

1. Soit $G = (S, A)$ un graphe non orienté. Un couplage de G est un ensemble d'arêtes $C \subseteq A$ tel que tout sommet de G est incident à au plus une arête de C ($\forall e = (x, y), e' = (x', y') \in C, \{x, y, x, y'\}$ sont deux à deux disjoints).

   Un couplage C est dit :
   - **maximal** si il n'est pas strictement inclus dans un autre couplage ($\forall C'$ couplage, $C \subseteq C' \Rightarrow C = C'$).
   - **maximum** ou **de cardinalité maximale** si sa cardinalité (son nombre d'arêtes) est maximale parmi tous les couplages de G ($\forall C'$ couplage, $|C'| \leqslant |C|$).
   - **parfait** si tout sommet de G est incident à une arête de G ($\forall s \in S, \exists e \in C, \exists y \in S, e = (x, y)$ ou encore $|C| = |S|/2$)
2. $C = \{(A, B), (C, D), (E, F)\}$ est un couplage parfait, donc maximum. Le couplage $C' = \{(A, C), (B, E)\}$ est maximal (on ne peut lui ajouter aucune arête), mais il n'est pas maximum car C contient strictement plus d'arêtes.
3. Un chemin alternant pour C dans G est un chemin élémentaire dont les arêtes sont alternativement dans C et dans $A \backslash C$. Un chemin augmentant est un chemin alternant dont les deux extrémités sont libres (i.e. ne sont incidentes à aucune arête du couplage C).
4. Il s'agit du lemme de Berge du cours. Redémontrons-le ici.

   $\Rightarrow$: Par contraposée. Si C possède un chemin augmentant c, alors $C \Delta c$ (la différence symétrique) est un couplage de cardinalité $|C| + 1 > |C|$, donc $|C|$ n'est pas maximum. (cf schéma du cours de $C \Delta c$.)

   $\Leftarrow$: Par contraposée. Supposons que C n'est pas maximal et soit C' un couplage tel que $|C'| > |C|$. Montrons que C possède alors un chemin augmentant. On considère $D = (S, C \Delta C')$. Chaque sommet de G est incident à au plus une arête de C et une arête de C', donc le degré maximal de D est au plus 2. Donc chaque composante connexe de D est :
   - soit réduite à un sommet
   - soit un chemin élémentaire dont les arêtes sont alternativement dans C et dans C'
   - soit un cycle élémentaire dont les arêtes sont alternativement dans C et dans C'. Notons que cela implique que le cycle est de longueur paire et contient autant d'arêtes de C que de C'.

   (proposition précédente du cours, qui se prouve en considérant un chemin élémentaire de longueur maximale dans G et en regardant s'il est de longueur 0 et s'il est "fermable".)

Chaque composante cycle (et sommet) possède autant d'arête de C que de C', or $|C'| > |C|$ donc $D = C\Delta C'$ possède au moins une arête de plus dans C' que dans C. Donc il existe au moins une composante chemin avec une arête de plus dans C' que dans C. Cette composante fournit un chemin augmentant pour C.

Exercice 8

Exercice 2

Soit Σ un alphabet.
Pour deux mots u, v dans Σ^*, on appelle entrelacement de u et v, un mot w qui utilise exactement les lettres de u et de v dans leur ordre dans chaque mot.
Par exemple $babaa$ est un entrelacement de bb et aaa ou encore $abcabc$ est un entrelacement de aba et cbc. Mais $bacb$ n'est pas un entrelacement de ab et cb car l'ordre n'est pas respecté.
On peut voir le lien avec les entrelacements des tâches de plusieurs fils d'exécution.

On note $\mathcal{E}(u, v)$ l'ensemble des entrelacements de u et v pour deux mots u, v.

2.1 Donner les entrelacements de ab et cb.

2.2 Soit u, v deux mots.

1. Majorer grossièrement le cardinal de $\mathcal{E}(u, v)$.
2. Montrer que $\mathcal{E}(u, v)$ est un langage régulier.

2.3 Soit L_1, L_2 deux langages réguliers. Montrer que $\mathcal{E}(L_1, L_2) = \bigcup_{u \in L_1, v \in L_2} \mathcal{E}(u, v)$ est un langage régulier.

2.4 Soit u, v, w trois mots dans Σ^*, proposer un algorithme de programmation dynamique permettant de savoir si $w \in \mathcal{E}(u, v)$. Préciser la complexité.

Solution de l'Exercice 8

Remarque : cet exercice est un des exercices du TD, que j'ai eu l'occasion de donner (un peu) en colles.

1. $\mathcal{E}(u, v) = \{abcb, acbb, cbab, cabb\}$.
2. **a.** On aura le plus grand nombre de mots possible dans le cas où les lettres sont deux à deux distinctes (il n'y a aucun doublon). Dans ce cas, toutes les combinaisons sont différentes. On compte le nombre de façon d'insérer les lettres du mot v dans le mot u.
   Cela revient à choisir les $|v|$ positions des lettres de v dans le mot mélange, les autres positions étant celles de u. Une fois ce choix effectué, on doit mettre les lettres de u et de v dans l'ordre et on n'a plus de choix à faire. Il y a donc $\binom{|u|+|v|}{|v|}$ éléments dans $u \star v = \mathcal{E}(u, v)$.

   $$\boxed{\mathcal{E}(u, v) \leqslant \binom{|u| + |v|}{|v|}}$$

   b. On s'inspire du produit de deux automates. Sauf qu'au lieu d'avancer dans les deux automates en parallèle en suivant une transition (en lisant une lettre), on ne va avancer que dans un automate à la fois (deux transitions possibles, de manière non déterministe). Formalisons cette idée :

   (2.3) Soient $A = (\Sigma, Q, q_I, F, \delta)$ et $A = (\Sigma', Q', q'_I, F', \delta')$ deux automates finis déterministes (AFD) acceptant respectivement les langages L et L'. On construit l'automate (non déterministe) sur $\Sigma \cup \Sigma'$ dont l'ensemble d'états est $Q \times Q'$, les ensembles d'états initiaux et finaux sont (q_I, q'_I) et $F \times F'$ et dont l'ensemble des transitions est :

   $$\{(p, p') \xrightarrow{a} (q, p') \mid \delta(p, a) = q\} \bigcup \{(p, p') \xrightarrow{a} (q, p') \mid \delta'(p', a) = q'\}.$$

   On vérifie sans difficulté que cet automate accepte $L \star L' = (L, L')$. Ceci répond à la question pour $L = \{u\}$ et $L' = \{v\}$.

***

(2.2 2.) Pour une preuve plus simple, utilisant directement les mots (u, v), on peut donner l'automate ci-dessus directement. Notons $u = u_1...u_n$ et $v = v_1...v_m$. On construit l'automate contenant $(n+1) \times (m+1)$ états notés $(i,j)_{i,j \in [\![0,n]\!] \times [\![0,m]\!]}$ où l'état (i,j) représente le fait qu'on a lu les i premières lettres du mot u et les j premières lettres du mot v.

L'état initial est alors $(0,0)$, l'état final (unique) est (n,m) et on a les transitions suivantes pour tout i, j :

$$\begin{cases} (i,j) \overset{u_{i+1}}{\to} (i+1,j) & \text{si } i < n \\ (i,j) \overset{v_{j+1}}{\to} (i,j+1) & \text{si } j < m \end{cases}$$

3. C'est la première preuve donnée dans la réponse précédente (la plus générale des deux).

4. Posons $w = w_1 w_2 ... w_{n+m}$ en gardant les notations précédentes. Si la longueur de w n'est pas égale à $n+m$, on peut directement conclure que $w \notin \mathcal{E}(u,v)$. On note $e_{i,j} = \begin{cases} 1 & \text{si } w_1...w_i \in \mathcal{E}(u_1...u_j, v_1...v_{i-j}) \\ 0 & \text{sinon} \end{cases}$

On cherche à renvoyer $e_{n+m,n}$, et on peut calculer récursivement les $e_{i,j}$ à l'aide des formules suivantes (notons qu'on a toujours $i, j \geqslant 0$ et $j - i \geqslant 0$) :

$$e_{i,j} = \begin{cases} 1 & \text{si } i = j = 0 \\ \mathbb{1}_{(w_i = v_{i-j} \text{ et } e_{i-1,j}=1)} & \text{sinon si j = 0 (plus de lettre dans } u) \\ \mathbb{1}_{(w_i = u_j \text{ et } e_{i-1,j-1}=1)} & \text{sinon si j = i (plus de lettre dans } v) \\ \mathbb{1}_{(w_i = u_j \text{ et } e_{i-1,j-1}=1) \text{ ou } (w_i = v_{i-j} \text{ et } e_{i-1,j}=1)} & \text{sinon.} \end{cases}$$

On calcule $e_{n+m,n}$ par programmation dynamique, par exemple en version récursive mémoïsée dans un tableau de taille $(n+1) \times (n+m+1)$ en temps $O(n.(n+m))$ car chaque case se calcule en temps constant (en fonction d'au plus deux autres cases, et moins quand les lettres sont distinctes). De manière itérative, on peut remplir le tableau de haut en bas et de gauche à droite (pour (i,j) croissant lexicographiquement), puisque la case (i,j) dépend au plus de la case d'au dessus et de la case de gauche.

II.5 - Planche 4

Exercice 9

Exercice 1

Soit $F = ((a \wedge b) \vee \neg c) \wedge (a \vee \neg b)$

1.1 F est-elle satisfiable ? $\neg F$ est-elle satisfiable ?

1.2 Mettre F en forme normale conjonctive.

1.3 Mettre F en forme normale disjonctive.

1.4 Rappeler le théorème de Cook-Levin.

Solution de l'Exercice 9

1. F est satisfiable, par exemple une valuation $v : \begin{cases} a \mapsto V \\ b \mapsto V \end{cases}$ la satisfait (peu importe la valeur de c).

   $\neg F$ est satisfiable, par exemple la valuation $v : \begin{cases} a \mapsto F \\ b \mapsto V \\ c \mapsto V \end{cases}$ évalue F à Faux, donc $\neg F$ à Vrai.

2. Il y a deux façons de faire, la table de vérité a l'avantage de donner des formes canoniques et de répondre aux deux questions suivantes rapidement une fois qu'elle est remplie, mais la résolution par distributivités me semble plus rapide au total que le remplissage d'une table, même s'il faut la faire deux fois. Donnons d'abord cette deuxième solution.
   On identifie $\wedge$ à $+$ et $\vee$ à $\times$ pour obtenir en distribuant une conjonction. On calcule alors aisément :
   $F = ((a+b).\neg c) + (a.\neg b) = a.\neg c + b.\neg c + a.\neg b$, d'où
   $\boxed{F \equiv (a \vee \neg c) \wedge (a \vee \neg b) \wedge (a \vee \neg b)}$
3. De même, cette fois on identifie $\vee$ à $+$ et $\wedge$ à $\times$. On distribue :
   $F = ((a.b) + \neg c).(a + \neg b) = a.b.a + a.b.\neg b + \neg c.a + \neg c.\neg b$, d'où
   $F \equiv (a \wedge b \wedge a) \vee (a \wedge b \wedge \neg b) \vee (\neg c \wedge a) \vee (\neg c \wedge \neg b) \equiv \boxed{(a \wedge b) \vee (\neg c \wedge a) \vee (\neg c \wedge \neg b)}$.

   ***

   Version avec table de vérité :

a	b	c	$a \wedge b$	$(a \wedge b) \vee \neg c$	$(a \vee \neg b)$	F
0	0	0	0	1	1	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	0
1	0	0	0	1	1	1
1	0	1	0	0	1	0
1	1	0	1	1	1	1
1	1	1	1	1	1	1

   Remarque : on peut se permettre de ne pas remplir tous les colonnes et conclure directement pour F dans certains cas, pour gagner du temps.
   On lit alors la table pour trouver les CNF et DNF canoniques (let "et" des non-lignes à 0, le "ou" des lignes à 1) :
   (CNF :) $F \equiv (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c)$
   (DNF :) $F \equiv (\neg a \wedge \neg b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge b \wedge c) \vee (a \wedge b \wedge c)$
4. SAT est NP-complet (admis).

Exercice 10

Exercice 2

On considère un ensemble de clés $\mathcal{K} \subseteq \mathbb{N}$ fini de taille m. On note $\mathcal{A}_\mathcal{K}$ l'ensemble des arbres binaires dont les noeuds sont étiquetés par des éléments *distincts* de $\mathcal{K}$. Pour $A \in \mathcal{A}_\mathcal{K}$, on notera $\kappa(A)$ l'ensemble des étiquettes des nœuds de A. Par convention, on dira que la racine de A est de profondeur 0 .

2.1 Rappeler la définition inductive d'un *arbre binaire de recherche* (ABR).

2.2 Rappeler la complexité dans le pire des cas de la recherche dans un ABR ayant n nœuds.

Soit $A \in \mathcal{A}_\mathcal{K}$ tel que $\kappa(A) = \mathcal{K}$. Soit $\mathbb{P}$ une loi de probabilité quelconque, sur les clés de $\mathcal{K}$. On définit H_A la variable aléatoire qui, à une clé de $\mathcal{K}$, associe sa profondeur dans A. Ainsi $H_A(k)$ est la profondeur de k pour une clé k.

2.3 Soit un ABR fixé $A \in \mathcal{A}_\mathcal{K}$. Rappeler l'expression de la complexité en moyenne de la recherche d'une clé dans A.

2.4 On suppose ici que $\mathbb{P}$ est la loi uniforme sur les clés de $\mathcal{K}$. Quelles sont les particularités des ABR $A \in \mathcal{A}_\mathcal{K}$ pour lesquels $\mathbb{E}(H_A)$ est minimale. Montrer que $\mathbb{E}(H_A) = \mathrm{O}(\log(m))$.

On dit que $A \in \mathcal{A}_\mathcal{K}$ est optimal pour $\mathcal{K}$ et $\mathbb{P}$ si et seulement si $\kappa(A) = \mathcal{K}$ et il minimise $\mathbb{E}(H_A)$.

On cherche un algorithme calculant un ABR optimal étant donné un ensemble de clés fini et une loi de probabilité sur ces clés.

2.5 De quel type de problème algorithmique s'agit-il ?

2.6 Pour cette question, on suppose $\mathcal{K} = \{1, 2, 3\}, \mathbb{P}(1) = \frac{2}{3}, \mathbb{P}(2) = \mathbb{P}(3) = \frac{1}{6}$. Dessiner un ABR optimal pour $\mathcal{K}$ et $\mathbb{P}$

2.7 Soit $A \in \mathcal{A}_\mathcal{K}$ un ABR optimal non vide. A possède donc un sous-arbre gauche A_g. On se place dans le cas où A_g est non vide. On note $S_g = \sum_{k \in \kappa(A_g)} \mathbb{P}(k)$ et $\mathbb{P}_g = \frac{1}{S_g}\mathbb{P}$. Montrer que A_g est optimal pour $\kappa(A_g)$ et $\mathbb{P}_g$. Montrer le résultat analogue pour le sous-arbre droit.

2.8 Proposer un algorithme de programmation dynamique pour trouver un ABR optimal.

2.9 Quel est sa complexité ?

Solution de l'Exercice 10

1. *(Comme un air de déjà-vu... CCINP a posé la même question en début d'un de ses exercices de Type A...).* Un arbre binaire de recherche est un arbre binaire dont les clés sont à valeurs dans un ensemble totalement ordonné. De plus, pour chaque noeud de cet arbre, son étiquette x est strictement supérieure aux étiquettes dans son fils gauche et strictement inférieure aux étiquettes dans son fils droit (des variantes autorisent une inégalité large d'un des deux côtés).
2. La complexité de la recherche dans un ABR de hauteur h est $O(h)$, dans le pire cas cette hauteur vaut n ou $\Theta(n)$ (par exemple si on a un graphe ligne ou un peigne). Donc la complexité pire cas est $O(n)$.
3. Pour une clé k fixée, la recherche de k dans A se fait en temps $O(H_A(k))$. La clé k a une probabilité d'apparition donnée par $\mathbb{P}$. Donc la complexité en moyenne (espérance) sur les entrées de la recherche d'une clé dans A est :
$$O(\mathbb{E}(H_A))$$
4. L'espérance est alors la moyenne (classique) des hauteurs des noeuds de l'arbre. Les ABR pour lesquels $\mathbb{E}(H_A)$ est minimale sont les arbres les plus équilibrés possibles (ils minimisent la somme des hauteurs de leurs noeuds, car on remplit d'abord entièrement les étages de plus petite profondeur), c'est à dire les arbres complets (à remplissage du dernier étage près, qui est rempli comme on veut).
   Pour montrer que $\mathbb{E}(H_A) = O(\log(m))$, il suffit de montrer que la profondeur maximale d'un noeud de l'arbre (i.e. sa hauteur) est un $O(\log(m))$, car l'espérance est plus petite que la valeur maximale avec une probabilité uniforme (c'est la moyenne).
   On a montré dans le cours de 1ère année que pour un arbre complet à m noeuds, on a $h = \lfloor \log m \rfloor$. Redémontrons ce résultat. On montre d'abord que $2^h \leqslant m < 2^{h+1}$.

Preuve : Notons m_k le nombre de noeuds à la profondeur k dans l'arbre. Comme l'arbre est complet, on a $m_k = 2^k$ pour tout $k \in [\![0, h[\![$ et on a $1 \leqslant m_h \leqslant 2^h$.

Donc comme $m = \sum_{k=0}^{h} m_k$, on a :

$$1 + \sum_{k=0}^{h-1} 2^k \leqslant m \leqslant \sum_{k=0}^{h} 2^k$$

D'où $2^h \leqslant m \leqslant 2^{h+1} - 1$. On conclut :

$h \leqslant \log m < h + 1$, c'est à dire par définition de la partie entière $\boxed{h = \lfloor \log m \rfloor = O(\log m)}$.

5. Il s'agit d'un problème d'optimisation. *Je ne suis pas sure de ce que la question attendait de plus spécifique ici ?*

6.

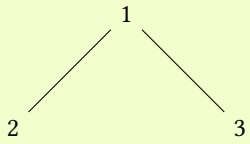


FIGURE IV.1 – Arbre pour les probabilités $\mathbb{P}(1) = 2/3, \mathbb{P}(2) = 1/6, \mathbb{P}(3) = 1/6$.

7. L'idée est toute simple : si ce n'était pas le cas, il existerait un meilleur arbre A'_g pour les noeuds à gauche, et il suffirait de remplacer A_g par A'_g dans A pour obtenir un arbre strictement meilleur que A, ce qui contredirait l'optimalité de A. Rédigeons cela rigoureusement.

On sait que A est optimal, i.e. qu'il minimise $\mathbb{E}(H_A)$. Or :

$$\begin{aligned}\mathbb{E}(H_A) &= \sum_{k \in \mathcal{K}} \mathbb{P}(k) H_A(k) \\ &= \sum_{k \in \kappa(A_g)} \mathbb{P}(k) H_A(k) + \sum_{k \in \mathcal{K} \setminus \kappa(A_g)} \mathbb{P}(k) H_A(k)\end{aligned}$$

Or $\sum_{k \in \kappa(A_g)} \mathbb{P}(k) H_A(k) = \sum_{k \in \kappa(A_g)} S_g \mathbb{P}_g(k)(H_{A_g}(k) + 1) = S_g \sum_{k \in \kappa(A_g)} \mathbb{P}_g(k) H_{A_g}(k) + \sum_{k \in \kappa(A_g)} S_g \mathbb{P}_g(k)$.

Résumons : $\boxed{\mathbb{E}(H_A) = \boxed{S_g \sum_{k \in \kappa(A_g)} \mathbb{P}_g(k) H_{A_g}(k)} + \sum_{k \in \kappa(A_g)} S_g \mathbb{P}_g(k) + \sum_{k \in \mathcal{K} \setminus \kappa(A_g)} \mathbb{P}(k) H_A(k)}$

Supposons maintenant par l'absurde que A_g n'est pas optimal pour $\kappa(A_g)$ et $\mathbb{P}_g$. Alors il ne minimise pas $\mathbb{E}(H_{A_g})$, donc il existe un arbre A'_g de même noeuds ($\kappa(A_g) = \kappa(A'_g)$) tel que $\mathbb{E}(H_{A'_g}) < \mathbb{E}(H_{A_g})$ (espérance pour $\mathbb{P}_g$), i.e. :

$$\sum_{k \in \kappa(A_g)} \mathbb{P}_g(k) H_{A'_g}(k) < \sum_{k \in \kappa(A_g)} \mathbb{P}_g(k) H_{A_g}(k)$$

En remplaçant A_g par A'_g dans A, on a les mêmes ensembles de clés et on obtient donc la nouvelle espérance consistant à échanger ces deux termes dans l'expression précédente (tous les autres termes restant égaux). Cette espérance est strictement plus faible, ce qui contredit l'optimalité de A, absurde.

De même pour le sous-arbre droit.

8. On remarque que dans un ABR, une fois qu'on a choisi une racine $r \in \mathcal{K}$, cela fixe les clés des sous-arbres gauche et droit. Notons $=\{k_1, ..., k_m\}$ les clés énumérées dans l'ordre croissant. Si k_i est la racine de l'arbre, alors le sous-arbre gauche contient les clés $k_1, ..., k_{i-1}$ et le sous-arbre droit contient les clés $k_{i+1}, ..., k_m$.

Notons alors $E_{i,j}$ la meilleure espérance qu'on puisse obtenir pour un ABR optimal sur les clés $k_i, ..., k_j$ pour la probabilité $\mathbb{P}_{i,j} = \frac{1}{S_{i,j}} \mathbb{P}$, avec $S_{i,j} = \sum_{l=i}^{j} \mathbb{P}(k_l)$. On peut calculer les $E_{i,j}$ via les formules de récurrence :

$$E_{i,j} = \begin{cases} 0 & \text{si } i \geqslant j \\ \min\limits_{r \in [\![i,j]\!]} \left(S_{i,r-1}(1 + E_{i,r-1}) + S_{r+1,j}(1 + E_{r+1,j}) \right) & \text{sinon} (i < j) \end{cases}$$

On calcule alors $E_{1,m}$, le résultat recherché, par programmation dynamique, par exemple de manière récursive en mémoïsant les $E_{i,j}$ calculés dans un tableau de taille $m \times m$ ou par une table de hachage, par exemple.

9. On calcule $O(n^2)$ valeurs $E_{i,j}$ au plus, toutes celles pour $i < j$, et chacune d'entre elle peut se calculer en temps $O(n)$ en utilisant les autres valeurs. En effet, on peut commencer par pré-calculer toutes les sommes partielles $S_{i,r-1}$ et $S_{r+1,j}$ en temps $O(n)$ (par simples passes dans deux tableaux pour les stocker). Puis, on calcule le min sur r en $O(n)$.

On obtient donc une complexité totale en $\boxed{O(n^3)}$.