Composition d'informatique n°4

(Durée : 4 heures)

L'utilisation de la calculatrice **n'est pas autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Ordonnancement de flux de tâches

Les problèmes d'ordonnancement ont été largement étudiés tant leurs applications sont nombreuses, comme l'ordonnancement des processus au niveau d'un processeur, la planification de transports ferroviaires ou l'agencement de tâches dans une entreprise. Il existe de nombreuses variantes de ces problèmes, dont une grande partie est NP-difficile.

Dans ce sujet, on s'intéresse au cas particulier de l'ordonnancement d'un flux de tâches. Après une définition du problème, la première partie s'intéresse au cas particulier d'un ordonnancement de tâches sur deux machines. La deuxième partie étudie le cas d'une indisponibilité de la première machine, toujours dans le cas à deux machines. La troisième et dernière partie traite le cas général d'un nombre quelconque de machines.

Consignes

Les questions de programmation doivent être traitées en langage C pour les deux premières parties et en OCaml pour la troisième. En C, on supposera que les bibliothèques `stdlib.h`, `stdbool.h` et `string.h` ont été chargées. En OCaml, on autorisera toutes les fonctions des modules `Array` et `List`, ainsi que les fonctions de la bibliothèque standard (celles qui s'écrivent sans nom de module, comme `max`, `incr` ainsi que les opérateurs comme `@`). Sauf précision de l'énoncé, l'utilisation d'autres modules sera interdite.

Présentation du problème

Pour illustrer ce problème, considérons l'atelier du père Noël où trois lutins doivent préparer les cadeaux. Chaque lutin est spécialisé dans une seule opération : Frimousse doit mettre le jouet dans un carton, Pimprenelle doit emballer le carton en un paquet-cadeau et Gribouille doit aller ranger le cadeau sur les étagères en attente de livraison. Selon le jouet, ces opérations peuvent prendre un temps différent ; cependant, elles doivent toujours être faites dans le même ordre : on ne peut pas emballer le carton s'il n'y a pas le jouet dedans, par exemple. De plus, un cadeau ne peut pas en doubler un autre : tous les lutins s'occuperont des différents cadeaux toujours dans le même ordre.

Les lutins cherchent à minimiser le temps total nécessaire pour préparer tous les cadeaux. Par exemple en considérant la répartition des temps de la figure 1, on peut envisager les ordonnancements donnés par les figures 2 et 3. Le premier ordonnancement a une durée totale de 37 minutes, et le deuxième ordonnancement a une durée totale de 33 minutes. Ce dernier est optimal.

		Jouet (indice)			
		Ballon (0)	Puzzle(1)	Peluche (2)	Camion (3)
Lutin	Frimousse	6	3	5	9
	Pimprenelle	8	7	7	6
	Gribouille	2	8	5	3

Figure 1 – Répartition des temps en minutes pour chaque opération

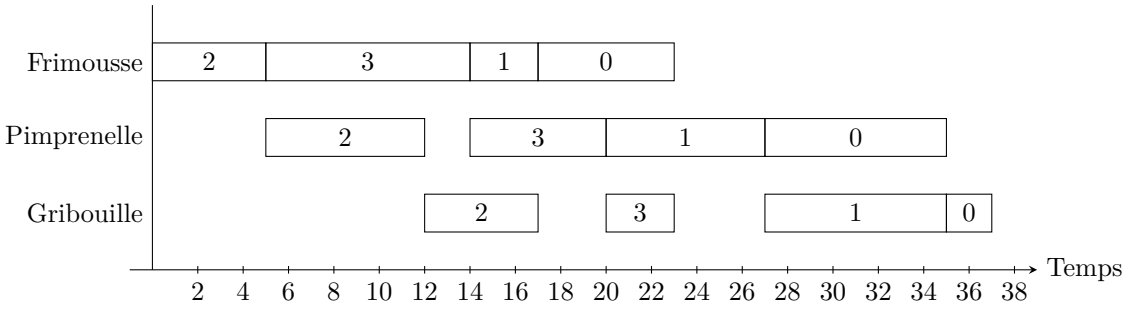

FIGURE 2 – Représentation graphique de l'ordonnancement $\sigma_1 = (2, 3, 1, 0)$, de durée 37 minutes

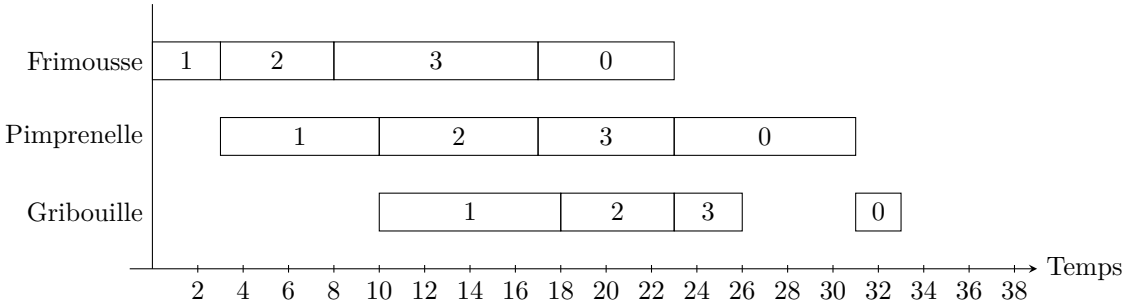

FIGURE 3 – Représentation graphique de l'ordonnancement $\sigma_2 = (1, 2, 3, 0)$, de durée 33 minutes

Pour définir plus formellement le problème, on souhaite traiter des tâches sur des machines, sous les hypothèses suivantes :

- il y a n **tâches** (les cadeaux dans l'exemple, avec $n = 4$), d'indices compris entre 0 et $n - 1$;
- il y a m **machines** (les lutins dans l'exemple, avec $m = 3$), d'indices compris entre 0 et $m - 1$;
- une tâche $i \in \{0, \ldots, n-1\}$ est décomposée en m opérations ; pour traiter la tâche i, il faut traiter les m opérations, dans l'ordre (l'opération 0, puis l'opération 1, ...), en respectant la contrainte que l'opération j doit être traitée sur la machine j ;
- chaque opération a une durée définie à l'avance ; on notera $t_{i,j}$ la durée de l'opération $j \in \{0, \ldots, m-1\}$ de la tâche $i \in \{0, \ldots, n-1\}$; on supposera que les $t_{i,j}$ sont des entiers positifs ou nuls.
- une machine ne peut traiter qu'une seule opération à la fois. Lorsqu'une opération est commencée sur une machine, la machine doit terminer complètement cette opération avant de commencer à traiter une nouvelle opération ;
- une tâche ne peut pas en doubler une autre : cela signifie que sur chaque machine, les numéros de tâches des opérations traitées seront toujours dans le même ordre.

Une instance du problème est donc une matrice $t = (t_{i,j})_{0 \leqslant i < n, 0 \leqslant j < m}$.

Étant donnée une instance t du problème, un **ordonnancement** est une permutation σ de $\{0, \ldots, n-1\}$. Un ordonnancement $(\sigma(0), \sigma(1), \ldots, \sigma(n-1))$ s'interprète par le fait que la première tâche dont on exécutera l'opération sur chaque machine est celle d'indice $\sigma(0)$, la deuxième celle d'indice $\sigma(1)$, etc. Pour un ordonnancement σ, on définit les temps de **début** et de **fin** de l'opération j de la tâche i, notés respectivement $d_{i,j}$ et $f_{i,j}$, par :

- le temps de fin d'une opération est égal à son temps de début plus sa durée :

$$\text{pour } i \geqslant 0 \text{ et } j \geqslant 0, f_{i,j} = d_{i,j} + t_{i,j}$$

- pour commencer une opération d'une certaine tâche, il faut que l'opération précédente de la même tâche soit terminée, et que l'opération sur la même machine de la tâche précédente soit terminée :

$$\text{pour } i \geqslant 0 \text{ et } j \geqslant 0, d_{\sigma(i),j} = \max(f_{\sigma(i-1),j}, f_{\sigma(i),j-1})$$

On pose, par convention, $\sigma(-1) = -1$ et $f_{-1,j} = f_{i,-1} = 0$, pour que la formule précédente reste correcte pour $i = 0$ ou $j = 0$.

Par exemple, pour l'ordonnancement σ_1 de la figure 2, on aurait $d_{1,0} = 14$, $f_{1,0} = 17$, $d_{1,1} = 20$, $f_{1,1} = 27$, $d_{1,2} = 27$ et $f_{1,2} = 35$.

On cherche à minimiser le temps total des opérations, c'est-à-dire minimiser le temps de fin de la dernière opération sur la dernière machine : $f_{\sigma(n-1),m-1}$. On notera $\Phi(t, \sigma) = f_{\sigma(n-1),m-1}$ et $\Phi(t) = \min_{\sigma \in \mathfrak{S}_n} \Phi(t, \sigma)$.

1 Cas à deux machines

Cette partie est à traiter en langage C.

Dans cette partie, on suppose qu'il n'y a que deux machines, c'est-à-dire que $m = 2$. Pour reprendre l'exemple précédent, la journée du 24 décembre, les cadeaux doivent être mis directement dans le sac du père Noël, donc seules Frimousse et Pimprenelle s'occupent des cadeaux. Avec les temps donnés par la figure 4, l'ordonnancement $\sigma_1 = (0, 1, 2, 3)$ aurait une durée totale de 24 minutes (figure 5) et l'ordonnancement $\sigma_2 = (1, 3, 2, 0)$ une durée optimale de 22 minutes (figure 6).

		Jouet (indice)			
		Ballon (0)	Puzzle(1)	Peluche (2)	Camion (3)
Lutin	Frimousse	6	5	5	4
	Pimprenelle	2	4	3	4

FIGURE 4 – Répartition des temps dans un cas à deux machines

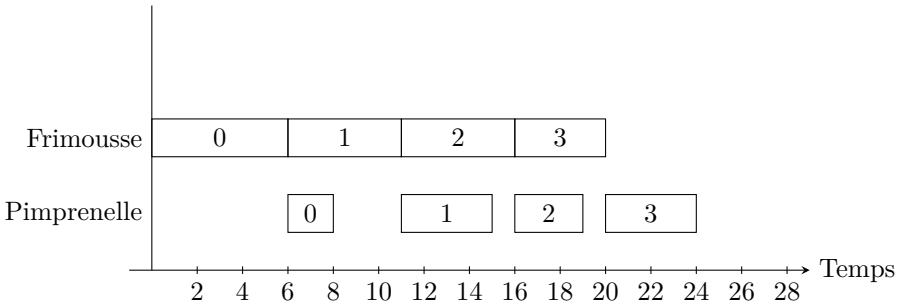

FIGURE 5 – Représentation graphique de l'ordonnancement $(0, 1, 2, 3)$, de durée 24 minutes

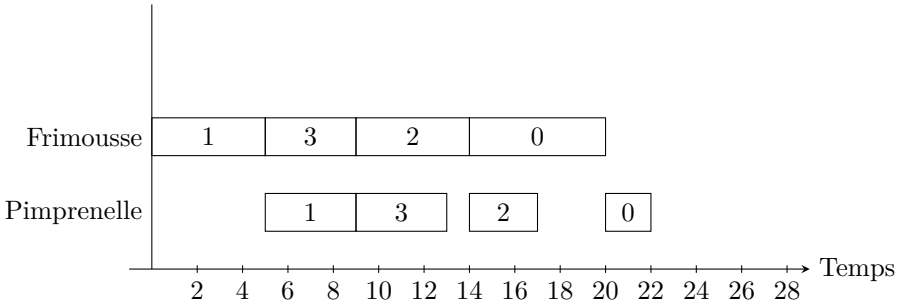

FIGURE 6 – Représentation graphique de l'ordonnancement $(1, 3, 2, 0)$, de durée 22 minutes

1.1 Préliminaires

Question 1 On considère une instance pour $n = 4$ du problème, définie par :

- $(t_{0,0}, t_{0,1}, t_{0,2}, t_{0,3}) = (3, 2, 1, 4)$;
- $(t_{1,0}, t_{1,1}, t_{1,2}, t_{1,3}) = (1, 3, 4, 2)$.

Donner, en justifiant, un ordonnancement optimal sur deux machines de cette instance. Quel est le temps total d'exécution de cet ordonnancement ?

On représente une instance du problème par la donnée de l'entier `n` et de deux tableaux de taille `n` :

- `temps0` correspondant aux valeurs de $t_{i,0}$, pour $i \in \{0, \ldots, n-1\}$;
- `temps1` correspondant aux valeurs de $t_{i,1}$, pour $i \in \{0, \ldots, n-1\}$.

Un ordonnancement σ est donné par un tableau `sigma` de taille `n` contenant les valeurs $\sigma(0), \sigma(1), \ldots$

Question 2 Écrire une fonction `int temps_total(int n, int* temps0, int* temps1, int* sigma)` qui détermine le temps total des opérations selon l'ordonnancement `sigma`, c'est-à-dire $\Phi(t, \sigma)$.

1.2 Algorithme de Johnson

S'il existe une tâche dont l'opération est très courte sur la machine 0 et très longue sur la machine 1, il peut être intéressant de la programmer en premier dans l'ordonnancement : le temps court sur la machine 0 permettra de rendre active la machine 1 le plus vite possible, le temps long sur la machine 1 évitera des temps morts sur cette dernière lors de l'attente de la fin d'une opération sur la machine 0.

C'est cette idée qui est mise en œuvre dans l'algorithme de Johnson.

On dit qu'un ordonnancement σ satisfait la **condition de Johnson** si et seulement si pour tout couple (i, j) vérifiant $0 \leqslant i < j < n$, on a :

$$\min(t_{\sigma(i),0}, t_{\sigma(j),1}) \leqslant \min(t_{\sigma(i),1}, t_{\sigma(j),0})$$

Question 3 L'ordonnancement $\sigma_1 = (0, 1, 2, 3)$ vérifie-t-il la condition de Johnson pour les temps donnés par la figure 4 ? Même question pour l'ordonnancement $\sigma_2 = (1, 3, 2, 0)$.

On pourra remarquer que la condition de Johnson peut se simplifier sur cette instance particulière.

Pour une instance du problème et $i \in [\![0, n-1]\!]$, on définit la tâche i par $\tau_i = (t_{i,0}, t_{i,1})$. On pose alors $T_0 = \{\tau_i \mid t_{i,0} < t_{i,1}\}$ et $T_1 = \{\tau_i \mid t_{i,0} \geqslant t_{i,1}\}$.

On définit la relation $\prec$ sur les tâches par $\tau_i \prec \tau_j$ si et seulement si :

- $\tau_i \in I_0$ et $\tau_j \in I_1$;
- ou $\tau_i, \tau_j \in I_0$ et $t_{i,0} < t_{j,0}$;
- ou $\tau_i, \tau_j \in I_1$ et $t_{i,1} > t_{j,1}$.

On note $\tau_i \preccurlyeq \tau_j$ si et seulement si $\tau_i \prec \tau_j$ ou ($\tau_j \not\prec \tau_i$ et $i \leqslant j$). On admet que $\preccurlyeq$ est une relation d'ordre total.

Question 4 Soit σ un ordonnancement. Montrer que si $\tau_{\sigma(0)} \preccurlyeq \tau_{\sigma(1)} \preccurlyeq \ldots \preccurlyeq \tau_{\sigma(n-1)}$, alors σ vérifie la condition de Johnson.

On représente une tâche par le type de données :

```c
struct Tache{
    int tm0;
    int tm1;
    int ind;
};
typedef struct Tache tache;
```

de telle sorte que si `tau` est un objet de type `tache` représentant la tâche τ_i d'indice i, alors `tau.tm0` vaut $t_{i,0}$, `tau.tm1` vaut $t_{i,1}$ et `tau.ind` vaut i.

Question 5 Écrire une fonction `bool cmp_johnson(tache taui, tache tauj)` qui prend en argument deux tâches τ_i et τ_j et renvoie `true` si et seulement si $\tau_i \preccurlyeq \tau_j$.

Question 6 Écrire une fonction

```
void tri_taches(int n, tache* tab, bool (*compare)(tache, tache))
```

qui prend en argument un entier n, un tableau `tab` de n tâches et une fonction de comparaison sur les tâches (de même type que `cmp_johnson`) et trie le tableau `tab` en place, de telle sorte qu'après le tri, pour $0 \leqslant i < j < n$, un appel à `compare(tab[i], tab[j])` renvoie toujours `true`.

On attend une complexité en $\mathcal{O}(n^2)$ en supposant que la fonction de comparaison s'effectue en temps constant.

Question 7 Écrire une fonction `int* taches_vers_ordo(int n, tache* tab)` qui prend en argument un tableau de tâches d'indices supposés distincts et renvoie un tableau correspondant à l'ordonnancement des tâches dans l'ordre où elles apparaissent dans `tab`.

On admet qu'un ordonnancement σ est optimal si et seulement s'il vérifie la condition de Johnson.

Question 8 En déduire une fonction `int* johnson(int n, int* temps1, int* temps2)` qui renvoie un ordonnancement optimal.

Question 9 Déterminer la complexité temporelle de la fonction `johnson` en fonction de n.

2 Maintenance sur l'une des deux machines

Cette partie est à traiter en langage C.

Dans cette partie seulement, on s'intéresse à un cas particulier où la première des deux machines peut être mise en maintenance pendant les opérations (Frimousse prend une pause).

Dans la donnée du problème, on suppose connues deux valeurs entières x et y correspondant au début et à la fin de la maintenance respectivement. Aucune opération ne peut être continuée pendant l'intervalle de temps $[x, y]$. Cependant, l'opération qui était en cours de traitement au temps x peut continuer sans pénalité au temps y. Toutes les opérations qui devaient terminer à un temps $> x$ sur la machine 0 termineront alors à un temps décalé de $y - x$.

Par exemple, considérons la répartition des temps de la figure 7, avec une pause pour Frimousse entre les temps $x = 10$ et $y = 18$. Pour l'ordonnancement $(0, 1, 3, 2)$ de la figure 8, on aurait une durée totale de 37 minutes. Pour l'ordonnancement l'ordonnancement $(3, 0, 2, 1)$ de la figure 9, on aurait une durée totale de 29 minutes, qui est optimale.

		Jouet (indice)			
		Ballon (0)	Puzzle(1)	Peluche (2)	Camion (3)
Lutin	Frimousse	6	7	2	3
	Pimprenelle	7	3	7	6

FIGURE 7 – Répartition des temps dans un cas à deux machines

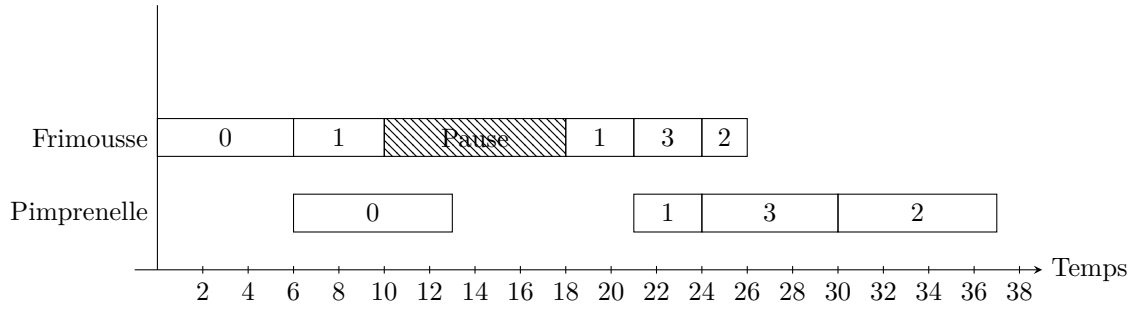

FIGURE 8 – Représentation graphique de l'ordonnancement $(0, 1, 3, 2)$, de durée 37 minutes

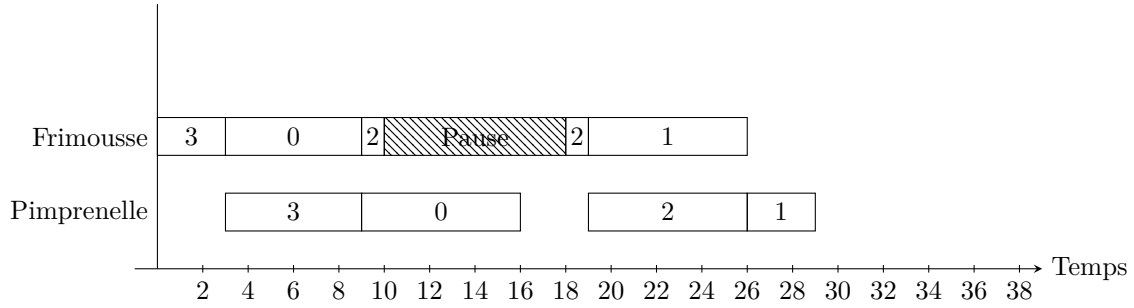

FIGURE 9 – Représentation graphique de l'ordonnancement $(3, 0, 2, 1)$, de durée 29 minutes

2.1 Un problème difficile

Là où le cas à deux machines peut être traité en temps polynomial, on peut montrer que ce n'est pas le cas en cas de maintenance, sauf si $\mathsf{P} = \mathsf{NP}$.

Pour prouver cela, on étudie une variante décisionnelle du problème, qu'on nommera `OFT` (Ordonnancement de Flux de Tâches)

* **Instance** : des temps $t_0 = (t_{0,0}, \ldots, t_{0,n-1})$ et $t_1 = (t_{1,0}, \ldots, t_{1,n-1})$, deux entiers $x < y$ et une borne B.
* **Question** : existe-t-il un ordonnancement des tâches sur deux machines avec maintenance sur la machine 0 entre les temps x et y, avec un temps total inférieur ou égal à B ?

Question 10 Montrer que le problème `OFT` est dans NP.

On souhaite effectuer une réduction depuis le problème `SOMME PARTIELLE` :

* **Instance** : des entiers naturels $x_0, \ldots, x_{k-1}$ et une cible $C \in \mathbb{N}$.
* **Question** : existe-t-il un sous-ensemble $I \subseteq [\![0, k-1]\!]$ tel que $\sum_{i \in I} x_i = C$?

qu'on admet comme étant NP-complet.

Dans un premier temps, on se ramène au problème `PARTITION` :

* **Instance** : des entiers naturels $x_0, \ldots, x_{k-1}$.
* **Question** : existe-t-il un sous-ensemble $I \subseteq [\![0, k-1]\!]$ tel que $\sum_{i \in I} x_i = \sum_{i \notin I} x_i$.

Question 11 Soit $(x_0, \ldots, x_{k-1}, C)$ une instance de `SOMME PARTIELLE` telle que $\sum_{i=0}^{k-1} x_i \geqslant C > 0$. Montrer que $(x_0, \ldots, x_{k-1}, C)$ est une instance positive de `SOMME PARTIELLE` si et seulement si $(x_0, \ldots, x_{k-1}, x_k, x_{k+1})$ est une instance positive de `PARTITION`, où $x_k = S + C$ et $x_{k+1} = 2S - C$, avec $S = \sum_{i=0}^{k-1} x_i$.

Question 12 En déduire que SOMME PARTIELLE $\leqslant_{\mathsf{m}}^{\mathsf{p}}$ PARTITION.

Dans un second temps, on veut réduire PARTITION au problème OFT. On considère $(x_0, \ldots, x_{k-1})$ une instance de PARTITION, de somme totale $\sum_{i=0}^{k-1} x_i = 2S$ et telle que $0 < x_i \leqslant S$ pour tout i. On pose :

- $n = k + 1$;
- pour $i \in [\![0, k-1]\!]$, $t_{i,0} = x_i$;
- $t_{k,0} = 0$;
- pour $i \in [\![0, k-1]\!]$, $t_{i,1} = x_i(S+1)$;
- $t_{k,1} = x_{\max} = \max_{i=0}^{k-1} x_i$;
- $x = S$ et $y = S^2 + S$;
- $B = 2S^2 + 2S + x_{\max}$.

On note alors $t_0 = (t_{0,0}, \ldots, t_{k,0})$ et $t_1 = (t_{1,1}, \ldots, t_{k,1})$.

Question 13 Expliquer pourquoi on peut se limiter au cas où la somme totale $\sum_{i=0}^{k-1} x_i$ est un entier pair et $0 < x_i \leqslant S$ pour tout i.

Question 14 Montrer que si $(x_0, \ldots, x_{k-1})$ est une instance positive de PARTITION, alors (t_0, t_1, x, y, B) est une instance positive de OFT.

Question 15 Réciproquement, montrer que si (t_0, t_1, x, y, B) est une instance positive de OFT, alors $(x_0, \ldots, x_{k-1})$ est une instance positive de PARTITION.

Question 16 En déduire que OFT est NP-complet.

2.2 Approximation

Question 17 Adapter la fonction `temps_total` de la question 2 pour prendre en compte le temps de maintenance de la machine 0. La fonction attendue sera de la forme :

```
int temps_maintenance(int n, int* temps1, int* temps2, int x, int y, int* sigma)
```

et renverra le temps total de l'ordonnancement σ avec maintenance sur la machine 0 entre les temps x et y.

Question 18 Montrer que l'algorithme de Johnson étudié dans la partie précédente fournit une 2-approximation du problème.

On se propose maintenant de construire une $\frac{3}{2}$-approximation du problème, en renvoyant l'ordonnancement ayant un temps minimal parmi les deux suivants :

- σ_1 un ordonnancement quelconque commençant par la tâche maximisant le temps de l'opération sur la deuxième machine ;
- σ_2 un ordonnancement correspondant aux tâches triées par ordre décroissant de $\dfrac{t_{i,1}}{t_{i,0}}$.

On souhaite montrer qu'il s'agit effectivement d'une $\frac{3}{2}$-approximation. Pour ce faire, on considère σ^* un ordonnancement optimal. On note F_1, F_2 et F^* les temps totaux d'exécution des ordonnancements σ_1, σ_2 et σ^* respectivement.

On dit que l'indice i est critique pour un ordonnancement σ si i est le plus petit indice tel que les tâches à partir de la tâche $\sigma(i)$ s'effectue sans discontinuer sur la machine 1, c'est-à-dire formellement si :

$$f_{\sigma(n-1),1} = d_{\sigma(i),1} + \sum_{j=i}^{n-1} t_{\sigma(j),1}$$

Question 19 Montrer que si i est critique pour σ, alors $f_{\sigma(i),0} = d_{\sigma(i),1}$.

Question 20 Montrer que $F_1 - \sum_{i=1}^{n-1} t_{\sigma(i),1} \leqslant F^*$.

Indication : on pourra distinguer selon que l'indice 0 *est critique pour* σ_1 *ou non.*

Question 21 Soit i_c l'indice critique pour σ_2. On pose I l'ensemble des indices des tâches qui terminent après la fin de la tâche $\sigma_2(i_c)$ sur la machine 0, c'est-à-dire après $f_{\sigma_2(i_c),0}$, dans un ordonnancement optimal.

1. Montrer que $\sum_{i\in I} t_{i,0} \geqslant \sum_{i=i_c+1}^{n-1} t_{\sigma_2(i),0}$.
2. En déduire que $F_2 - t_{\sigma_2(i),1} \leqslant F^*$.

Question 22 En déduire que dans tous les cas, l'algorithme décrit précédemment est bien une $\frac{3}{2}$-approximation du problème.

Indication : on pourra distinguer selon qu'il existe une tâche τ_i *telle que* $t_{i,1} > \dfrac{F^*}{2}$ *ou non.*

Question 23 Écrire une fonction

```
int* approximation(int n, int* temps0, int* temps1, int x, int y)
```

qui calcule une $\frac{3}{2}$-approximation du problème à deux machines avec maintenance.

On pourra réutiliser la fonction `tri_taches` *avec des fonctions de comparaison adaptées.*

3 Cas général

Cette partie est à traiter en langage OCaml

On admet que pour $m > 2$, le problème est NP-difficile.

On représente une instance du problème par la donnée d'une matrice `temps` de taille $n \times m$ telle que pour $i \in [\![0, n-1]\!]$ et $j \in [\![0, m-1]\!]$, `temps.(i).(j)` correspond à $t_{i,j}$. Comme dans les parties précédentes, un ordonnancement est représenté par un tableau de taille n.

3.1 Une approche gloutonne

On souhaite mettre en place une approche gloutonne pour résoudre le problème, en ajoutant les tâches petit à petit dans l'ordonnancement. Pour ce faire, on considère des ordonnancements partiels. Étant donné une instance du problème, on appelle **ordonnancement partiel** tout préfixe d'un ordonnancement. Par exemple,si $n = 5$, $\tilde{\sigma} = (2, 0, 3)$ est un ordonnancement partiel (car $(2, 0, 3, 1, 4)$ est un ordonnancement), dont l'interprétation est « on a programmé les tâches 2, 0 et 3 dans cet ordre, et pas encore programmé les tâches 1 et 4 ».

Question 24 Écrire une fonction

```
temps_partiel (temps : int array array) (sigma : int array) (i : int) : int
```

qui prend en argument une matrice de temps $n \times m$, un tableau σ et un entier i et calcule le temps d'exécution en supposant que seules les tâches d'indice $\sigma(0), \ldots, \sigma(i-1)$ sont programmées. On supposera que la taille de `sigma` est supérieure ou égale à i et que ces valeurs sont distinctes ; on n'impose aucune contrainte sur le reste des valeurs du tableau `sigma`.

L'approche gloutonne est la suivante : on part d'un ordonnancement partiel $\tilde{\sigma}$ initialement vide, puis, pour i de 0 à $n-1$, on insère la tâche i dans $\tilde{\sigma}$, en envisageant toutes les positions d'insertion possible et en conservant celle qui minimise le temps partiel (tel que calculé par la fonction précédente).

Question 25 Donner, sans justifier, l'ordonnancement obtenu en appliquant l'algorithme glouton à l'instance donnée en figure 10. Comparer le temps total de cet ordonnancement avec le temps total de l'ordonnancement $(0, 1, 2)$. Commenter.

		Jouet (indice)		
		Ballon (0)	Puzzle(1)	Peluche (2)
Lutin	Frimousse	1	2	3
	Pimprenelle	4	5	5
	Gribouille	2	4	1

FIGURE 10 – Instance de temps pour trois tâches et trois machines

Question 26 Écrire une fonction `glouton (temps : int array array) : int array` qui renvoie une solution du problème donnée selon l'algorithme glouton.

Question 27 Déterminer la complexité temporelle de la fonction `glouton`.

3.2 Algorithme *Branch and Bound*

On se propose de décrire une approche par séparation et évaluation (*Branch and Bound*) pour résoudre le problème. Pour cela, on considère à nouveau des des solutions partielles. Une solution partielle sera complétée en rajoutant les tâches manquantes **après** celles qui sont présentes.

Question 28 Expliquer comment résoudre le problème par retour sur trace (*backtracking*).

Pour améliorer l'exploration par retour sur trace, on se donne une heuristique d'évaluation calculée pour une solution partielle $\tilde{\sigma}$:

- déterminer le temps de fin de l'ordonnancement partiel $\tilde{\sigma}$;
- ajouter la somme des temps des opérations **sur la dernière machine** des tâches n'apparaissant pas dans $\tilde{\sigma}$;
- poser $h(\tilde{\sigma})$ cette valeur ;

Question 29 Justifier que la fonction h est une heuristique d'évaluation admissible.

Question 30 Proposer une heuristique de branchement utilisant l'heuristique h.

Bonus

Question 31 Montrer qu'un ordonnancement vérifiant la condition de Johnson est optimal pour le problème à deux machines sans maintenance.

⋆ ⋆ ⋆