Devoir maison n°2

À rendre le lundi 02/10

⋆ ⋆ ⋆

Vous indiquerez en début de copie le numéro de la partie dont vous souhaitez la correction.

Les deux parties sont indépendantes.

1 Castors affairés

Dans cette partie, on suppose que l'exécution d'un programme OCaml se fait sur un ordinateur à mémoire infinie. En particulier, on suppose que la représentation des entiers est non bornée.

On dit qu'une fonction $f : \mathbb{N} \to \mathbb{N}$ est **calculable** s'il existe une fonction OCaml `f : int -> int`, dont l'exécution termine toujours, qui calcule les images par f.

Un castor affairé (*busy beaver* en anglais) est un programme dont l'exécution termine toujours et qui calcule une valeur la plus grande possible parmi tous les programmes de même taille. Formellement, on s'intéresse au problème d'optimisation suivant : `Castor affairé` :

- **Instance** : un entier naturel n.
- **Solution** : la valeur k renvoyée par un programme OCaml contenant n caractères (parmi les 128 caractères possibles de l'encodage ASCII), dont l'exécution termine toujours et renvoie un entier.
- **Optimisation** : maximiser k.

Par exemple, le programme suivant :

```
let k=99 in k*k*k
```

est un programme de taille 17, qui termine toujours et renvoie 970299. Ce n'est pas un castor affairé, en effet le programme suivant (qui n'est toujours pas un castor affairé) renvoie une valeur plus grande :

```
99999999999999999
```

Pour un entier n donné, on notera $C(n)$ la valeur maximale renvoyée par un programme OCaml de taille n qui termine et renvoie un entier. On pose $C(0) = 0$ par convention.

On cherche à montrer que le problème du castor affairé n'est pas calculable, c'est-à-dire que la fonction C n'est pas calculable.

Question 1 Combien existe-t-il de programme OCaml à n caractères, en supposant qu'on dispose de 128 caractères différents possibles ? Expliquer pourquoi le fait qu'il y en ait un nombre fini ne permet pas de conclure que le problème est calculable.

Question 2 Montrer que la fonction C est correctement définie.

Question 3 Montrer que C est une fonction croissante, puis que pour $n \in \mathbb{N}$, $C(n+2) > C(n)$.

Question 4 Que vaut $C(1)$? Le fait qu'on puisse déterminer cette valeur contredit-il la non-calculabilité du problème ?

On considère $f : \mathbb{N} \to \mathbb{N}$ une fonction calculable.

Question 5 Montrer qu'il existe un entier k tel que pour tout $n \in \mathbb{N}$, $f(n) \leqslant C(\lfloor \log_{10} n \rfloor + k)$.

Question 6 Montrer qu'il existe un entier $n_0 \in \mathbb{N}$ tel que $f(n_0) < C(n_0)$.

Question 7 Conclure.

On considère le problème `Arrêt` :

* **Instance** : le code source d'un programme OCaml.
* **Question** : est-ce que l'exécution de ce programme termine ?

Question 8 Montrer, en utilisant la non-calculabilité du problème du castor affairé, que le problème `Arrêt` est indécidable.

2 Graphes d'intervalles

On s'intéresse à l'ordonnancement de tâches sur des machines. Chaque tâche est donnée sous la forme d'un intervalle de temps durant lequel une machine doit être dédiée à l'exécution de cette tâche. Étant donné un ensemble de tâches $t_0, ..., t_{n-1}$, l'objectif est de minimiser le nombre k de machines $M_0, ..., M_{k-1}$ utilisées pour toutes les exécuter.

L'exécution d'une tâche représentée par un intervalle $t_i = [d_i, f_i]$ sur une machine M_j occupe M_j durant la totalité de l'intervalle de temps t_i. Chaque machine ne peut donc exécuter qu'un ensemble de tâches dont les intervalles de temps sont disjoints.

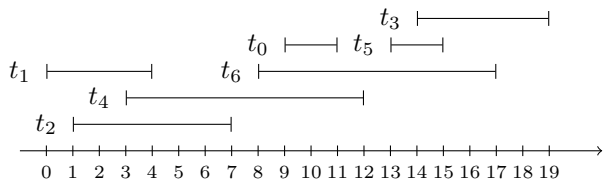
Un **ordonnancement** de l'ensemble de tâches $t_0, \ldots, t_{n-1}$ est une fonction $ordo : [\![0, n-1]\!] \to [\![0, k-1]\!]$. Si $ordo(i) = j$, cela signifie que la tâche t_i est exécutée sur la machine M_j.

On dit qu'un ordonnancement est **cohérent** si pour tout couple (i, i') tel que $i \neq i'$ et $ordo(i) = ordo(i')$, on a $t_i \cap t_{i'} = \emptyset$; c'est-à-dire que deux tâches exécutées sur la même machine correspondent toujours à des intervalles disjoints.

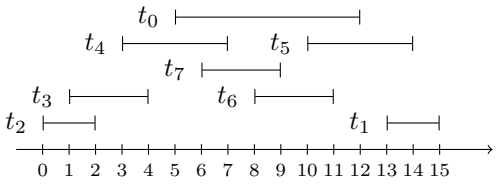
Le nombre de machines utilisées par un ordonnancement *ordo* est le nombre d'images distinctes par *ordo*. Un ordonnancement cohérent est **optimal** s'il n'existe pas d'ordonnancement cohérent utilisant strictement moins de machines.

2.1 Représentations des ensembles de tâches

Pour faciliter les raisonnements sur les ensembles de tâches, on les représente graphiquement comme suit :



Ensemble de tâches $\mathcal{T}_1$



Ensemble de tâches $\mathcal{T}_2$

L'ensemble de tâches $\mathcal{T}_1$ représenté ci-dessus est donc constitué de l'ensemble :

$$\mathcal{T}_1 = \{[9, 11], [0, 4], [1, 7], [14, 19], [3, 12], [13, 15], [8, 17]\}$$

Ici, l'axe horizontal représente le temps, il est gradué sur les entiers à partir de 0. Les différences de hauteur des intervalles dans les représentations graphiques n'ont pas d'importance, elles ne sont présentes que pour la lisibilité.

Question 9 Donner, sous forme de tableau, un ordonnancement optimal de l'ensemble $\mathcal{T}_1$, puis de l'ensemble $\mathcal{T}_2$. Justifier que l'on ne pourrait pas utiliser moins de machines.

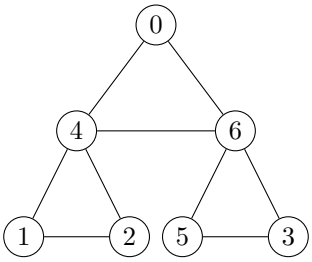
2.2 Lien avec les graphes

Dans cette partie, on introduit comment représenter un ensemble de tâches sous forme d'un graphe.

Soit $\mathcal{T} = \{t_0, ..., t_{n-1}\}$ un ensemble de tâches. On définit le graphe non orienté $G_\mathcal{T} = (S_\mathcal{T}, A_\mathcal{T})$ associé à $\mathcal{T}$ comme suit :

- $S_\mathcal{T} = \{0, ..., n-1\}$
- $A_\mathcal{T} = \{\{s, s'\} \in \mathcal{P}_2(S) | t_s \cap t_{s'} \neq \emptyset\}$.

Par exemple, le graphe associé à $\mathcal{T}_1$ est le suivant :

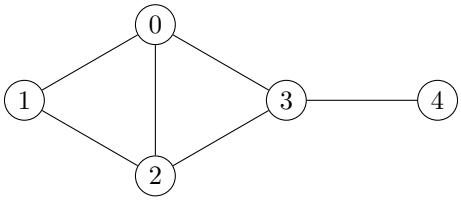


Un graphe construit de cette façon est un **graphe d'intervalles** et l'ensemble de tâches utilisé est une **réalisation** de ce graphe.

Le graphe d'intervalles est une représentation moins précise qu'une réalisation car on ne garde que l'information sur les intersections entre les intervalles des tâches. Deux ensembles de tâches distincts peuvent être des réalisations du même graphe d'intervalle.

Question 10 Représenter graphiquement le graphe d'intervalles associé à l'ensemble $\mathcal{T}_2$.

Question 11 Déterminer une réalisation du graphe d'intervalles suivant :



Représenter graphiquement l'ensemble des tâches pour plus de lisibilité.

2.3 Coloration de graphes

Soit $G = (S, A)$ un graphe non orienté. On appelle **coloration** de G une fonction $c : S \to \mathbb{N}$ telle que pour tout $(s, t) \in S^2$, $\{s, t\} \in A \Rightarrow c(s) \neq c(t)$, c'est-à-dire que deux sommets voisins n'ont jamais la même couleur (c'est-à-dire la même image par c). On dit que c est une k-coloration si $|c(S)| \leqslant k$. On impose généralement aux couleurs d'être des entiers consécutifs commençant à 0.

La figure 1 donne une 4-coloration d'un graphe G_0 :

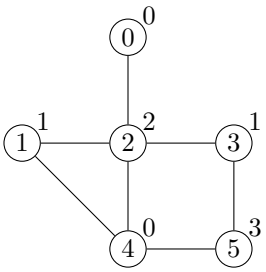


FIGURE 1 – Une 4-coloration de G_0. Les couleurs sont indiquées à l'extérieur des sommets.

G est dit *k-colorable* s'il possède une k-coloration. On définit également le *nombre chromatique* de G, noté $\chi(G)$, comme étant le plus petit entier k tel que G est k-colorable.

Question 12 Déterminer le nombre chromatique du graphe G_0 représenté figure 1. On exhibera une coloration optimale, et on justifiera rigoureusement qu'il est impossible d'utiliser moins de couleurs.

Le calcul du nombre chromatique d'un graphe est un problème difficile. On se propose donc d'étudier dans cette partie un algorithme glouton efficace permettant de déterminer des k-colorations d'un graphe G, sans garantir que $k = \chi(G)$.

Début algorithme
  Entrée : graphe $G = (S, A)$
  Pour chaque sommet $s \in S$ non coloré **Faire**
    Colorer s en utilisant la plus petite couleur possible

La ligne « Colorer s en utilisant la plus petite couleur possible » consiste à parcourir tous les sommets colorés adjacents à s et à numéroter s en utilisant le plus petit numéro n'apparaissant pas parmi les couleurs des voisins de s. On attribue toujours la couleur 0 au premier sommet traité.

Question 13 Déterminer la coloration obtenue en appliquant l'algorithme glouton au graphe G_0 de la figure 1, en supposant que les sommets sont parcourus par indices croissants.

Question 14 Déterminer une indexation des sommets telle que pour le graphe G_1 de la figure 2, l'algorithme glouton ne renvoie pas une coloration optimale (c'est-à-dire utilisant strictement plus de $\chi(G_2)$ couleurs), en supposant que les sommets sont parcourus par indices croissants.

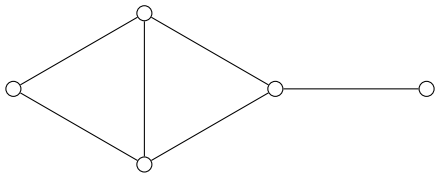


FIGURE 2 – Le graphe G_1.

2.4 Coloration de graphes d'intervalles

Dans la partie précédente, on a montré les limites de l'approche gloutonne pour la coloration d'un graphe quelconque. Dans cette partie, on fait le lien entre ordonnancement et coloration.

Question 15 Soit $G = (S, A)$ un graphe d'intervalles et r une réalisation de G. Prouver que *ordo* est un ordonnancement cohérent de r si et seulement si *ordo* est une coloration de G.

Question 16 Soit G un graphe d'intervalles et r une réalisation de G. Déterminer un ordre de traitement des sommets permettant à l'algorithme glouton de renvoyer une coloration optimale de G.

Indication : on pourra s'intéresser à un tri bien choisi des intervalles de la réalisation.

Le problème de la coloration de graphes d'intervalles peut ainsi être résolu efficacement si l'on dispose d'une réalisation de ce graphe. La partie suivante montre que même sans connaître de réalisation, le problème de la coloration admet toujours une résolution efficace pour les graphes d'intervalles. Pour cela, on étudie le parcours en largeur lexicographique et ses propriétés.

Le parcours en largeur lexicographique est un parcours en largeur où l'on contraint l'ordre de parcours des voisins d'un même sommet. Dans le parcours en largeur classique, on parcourt les voisins de chaque sommet dans un ordre arbitraire (ici on considère que le parcours se fait par indices de sommets croissants). Pour le parcours en largeur lexicographique, on associe à chaque sommet une étiquette que l'on construit au fur et à mesure et le parcours des voisins d'un sommet se fait en sélectionnant toujours le sommet dont l'étiquette est maximale pour l'ordre lexicographique. **Si deux sommets ont la même étiquette, on commence par celui d'indice minimum.**

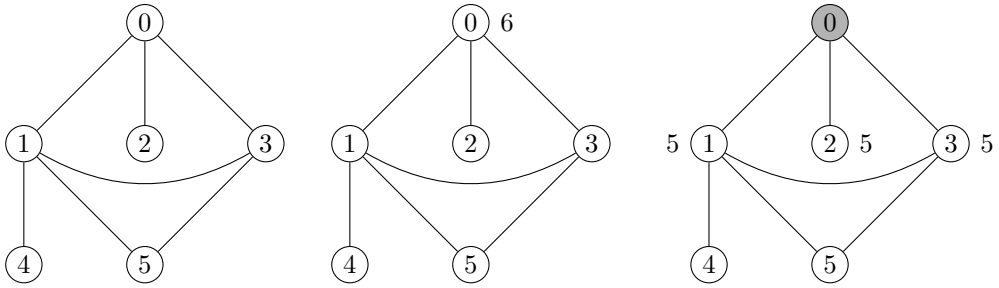
On appelle LexBFS l'algorithme de parcours en largeur lexicographique, cet algorithme est résumé ci-après. On attribue pour cela une étiquette à chaque sommet. Lorsque $S = \{0, 1, \ldots, n-1\}$, une *étiquette* est un mot, c'est-à-dire une suite de symboles, de $\{1, 2, \ldots, n\}$. La concaténation entre deux mots u et v est dénotée par

$u \cdot v$. Le mot vide est dénoté ε.

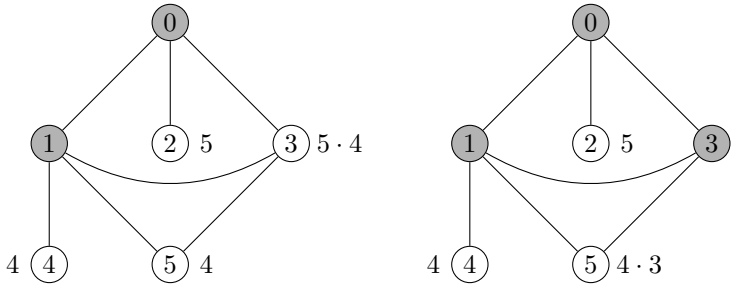
Début algorithme
  Entrée : graphe $G = (S, A)$ connexe d'ordre n
  Sorties : ordre de parcours σ
  Pour chaque sommet $s \in S$ **Faire**
    étiquette$(s) := \varepsilon$
  étiquette$(0) := n$
  Pour i de $n-1$ à 1 **Faire**
    Choisir s un sommet non traité d'étiquette maximale
    $\sigma(n-i-1) := s$ (s est maintenant traité)
    Pour chaque sommet t non traité, voisin de s **Faire**
      étiquette(t) := étiquette$(t) \cdot i$
  Renvoyer σ

Voici le déroulement de cet algorithme pas à pas sur un graphe simple. Pour une meilleure lisibilité, les étiquettes vides et celles des sommets déjà traités ne sont pas représentées.

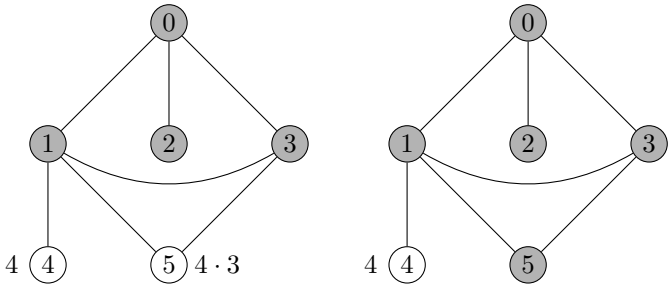
Au départ, les étiquettes sont toutes vides sauf le sommet 0 qui prend l'étiquette 6. Le parcours commence donc au sommet d'indice 0. Lors du traitement de 0, on concatène la valeur 5 aux étiquettes de tous ses voisins.



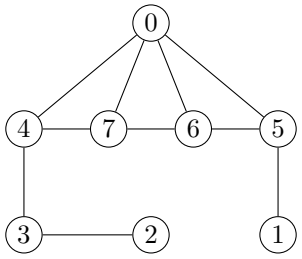
Le parcours continue à partir du sommet d'indice 1 car c'est le plus petit indice des sommets de d'étiquettes maximales. Lors du traitement du sommet 1, on concatène la valeur 4 aux étiquettes de tous ses voisins qui n'ont pas encore été traités. Le parcours continue alors sur le sommet d'indice 3 dont l'étiquette $5 \cdot 4$ est maximale pour l'ordre lexicographique. La valeur 3 est concaténée à l'étiquette du voisin non encore traité.



Le parcours continue sur le sommet 2 dont l'étiquette est à présent maximale, mais qui n'a pas de voisin. Puis le parcours se termine par les sommets 5 puis 4 qui n'ont pas de voisins non-traités. L'ordre renvoyé est donc $\sigma = (0, 1, 3, 2, 5, 4)$.



Question 17 Appliquer l'algorithme LexBFS sur le graphe ci-dessous.



On ne demande que l'ordre σ obtenu et pas une explication détaillée.

Question 18 Appliquer l'algorithme glouton de coloration en suivant l'ordre obtenu avec LexBFS. Comparer avec l'ordre obtenu avec l'algorithme de parcours en largeur classique où les voisins sont parcourus par indices croissants.

On veut montrer que l'ordre de parcours obtenu avec LexBFS pour un graphe d'intervalles permet toujours à l'algorithme glouton de renvoyer une coloration optimale de ce graphe.

On dit qu'un ordre de parcours σ est **simplicial** si pour chaque sommet, ses voisins qui le précèdent (dans σ) forment un sous-graphe complet.

Dans un graphe G, si $(s_0, s_1, ..., s_k)$ avec $s_k = s_0$ est un cycle, on dit qu'il admet une **corde** s'il existe deux sommets non-consécutifs du cycle qui sont adjacents.

On admet qu'un graphe d'intervalle ne contient aucun cycle de taille $\geqslant 4$ sans corde.

Question 19 Prouver que l'ordre renvoyé par LexBFS lorsqu'il est appliqué à un graphe d'intervalles est simplicial.

Indication : on pourra noter $s \prec t$ si s apparaît dans σ avant t, et montrer qu'on peut construire une suite infiniment décroissante pour $\prec$ s'il existe des sommets dont les voisins antérieurs ne forment pas une clique.

Question 20 En déduire que l'algorithme glouton de coloration appliqué avec l'ordre renvoyé par LexBFS est optimal sur les graphes d'intervalles.

$\star \star \star$