Lycée Saint François-Xavier

Spécialité NSI - Bac Blanc 2

Ce sujet comporte 8 pages hormis cette page de titre.
Il faut rendre l'énoncé à l'issue de l'épreuve.

Nom : _____

Classe : _____

- 27 septembre 2024 -

Exercice 1 :

Bertrand est directeur d'une salle de théâtre. Il a reçu l'autorisation d'ouvrir sa salle au public mais en raison des contraintes sanitaires il doit s'assurer que deux spectateurs sont toujours séparés par au moins un siège vide. Cherchant à remplir au maximum chaque rang de fauteuils, Bertrand compte installer les spectateurs une place sur deux.

Amateur de casse-tête, il se pose toutefois une question : dans une rangée de N fauteuils, combien de *dispositions* différentes respectent la condition d'espacement des spectateurs ?

Partie A : Approche naïve

Ayant suivi des cours d'informatique, il repense à la représentation binaire des nombres. Positionner des spectateurs sur un rang de N fauteuils revient à compter combien de nombres de N bits n'ont aucun bits `1` consécutifs.

Par exemple si $N = 3$, les dispositions valides sont représentées par les nombres `000`, `001`, `010`, `100`, `101`. Il y en a 5.

1. Quelles sont les dispositions valides dans le cas où $N = 4$? Combien y en a-t-il ?

Afin de trouver le nombre de configurations satisfaisantes, Bertrand envisage dans un premier temps de lister tous les nombres de N bits et de ne retenir que ceux satisfaisants la condition.

Afin de vérifier qu'un nombre vérifie bien la condition, il faut s'assurer qu'il ne comporte pas deux bits consécutifs égaux à `1`. Après quelques recherches, il établit la règle suivante :

«*Dans un nombre entier positif ou nul n, le test* `n & 2**k == 2**k` *renvoie* `True` *si et seulement si le* `k`*-ième bits de n vaut* `1`»

Remarque : on numérote les bits de droite à gauche en débutant avec l'indice 0. Ainsi, le bit de rang 1 de 5 est `0` et celui de rang 2 est `1` car $5_{10} = 101_2$.

Par exemple `5 & 2**1 == 2**1` renvoie `False` alors que `5 & 2**2 == 2**2` renvoie `True`.

2. a. Que renvoie l'appel `25 & 2**3 == 2**3` ?
   b. Que renvoie l'appel `25 & 2**5 == 2**5` ?

3. Écrire en `python` une fonction `k_ieme_bit` qui prend en argument deux nombres entiers positifs ou nuls `n` et `k` et renvoie le `k`-ième bit de `n`.

4. Écrire en `python` une fonction `disposition_valide` qui prend en argument deux nombres entiers positifs ou nuls `n` et `maxi` et renvoie `False` si deux bits consécutifs de `n` sont égaux à `1`, `True` dans le cas contraire. On pourra utiliser la fonction `k_ieme_bit` définie à la question précédente et tester l'égalité des bits de rangs `0` et `1`, `1` et `2`, ..., `maxi-1` et `maxi`.

5. a. Le théâtre de Bertrand propose des rangs de 50 sièges. Combien de nombres faut-il tester pour répondre au problème ?
   b. Dans l'hypothèse où pour chacun de ces nombres on teste chacun des 50 bits, combien de tests faut-il effectuer au total sur l'ensemble de tous les nombres ?

Partie B : Programmation dynamique

Cette approche naïve est donc trop coûteuse. Bertrand décide d'opter pour la *programmation dynamique*. Son observation initiale est simple : une rangée de N fauteuils n'est rien d'autre qu'une rangée de $N-1$ fauteuils à laquelle on rajoute une place.

Dès lors on peut rencontrer deux cas de figures :

- le dernier fauteuil est **occupé** par un spectateur : dans ce cas l'avant-dernier fauteuil était obligatoirement **vide**
- le dernier fauteuil n'est **pas occupé** par un spectateur : dans ce cas l'avant-dernier fauteuil était **vide ou plein**

Il est dès lors possible de compter le nombre de dispositions satisfaisantes en complétant un tableau comme le tableau 1.

Fauteuils dans la rangée	Dispositions se terminant par un siège **vide**	Dispositions se terminant par un siège **occupé**
0	0	0
1	1	1
2	2	1
3	3	2
4		
5		

Tableau 1 – Nombre de dispositions selon la taille de la rangée

1. a. Compléter sur cet énoncé les deux dernières lignes du tableau 1.
   b. Combien de dispositions valides existent pour une rangée de 5 fauteuils ?

Afin d'écrire un programme permettant de répondre au problème, Bertrand décide de coder le tableau sous forme d'une liste de liste. Afin de créer un tableau `T` de `n+1` lignes et deux colonnes initialement remplies de `0`, il utilise l'instruction `T = [[0,0] for k in range(n+1)]`.

Dans ce tableau la case de la n-ième ligne et de la colonne 0 (resp. 1) donne le nombre de dispositions valides se terminant par un siège vide (resp. occupé) pour une rangée de n fauteuils.

Ainsi en reprenant l'exemple du tableau 1 on a `T[3][0] = 3` car il existe 3 dispositions valides de 3 sièges se terminant par un siège vide.

De même, puisqu'il existe deux dispositions se terminant par un siège occupé dans une rangée de 3 fauteuils, on a `T[3][1] = 2`.

Après avoir initialisé le tableau, Bertrand modifie deux valeurs. En effet il est évident qu'il n'existe qu'une disposition valide se terminant par un fauteuil vide dans une rangée d'un seul fauteuil. Il en est de même pour la disposition se terminant par un fauteuil occupé.

2. Soit n un nombre entier supérieur ou égal à 1.
   a. Justifier que l'on a `T[n][0] = T[n-1][0] + T[n-1][1]`.
   b. Justifier que l'on a `T[n][1] = T[n-1][0]`.

3. Écrire en `python` la fonction `disposition` qui prend en argument le nombre `n` de fauteuils d'une rangée et qui renvoie le nombre de dispositions valides existant pour cette taille de

rangée. Cette fonction pourra suivre la démarche suivante :

- Création d'un tableau `T` de dimensions adaptées initialement rempli de `0`
- Mise à jour des valeurs de `T` pour la rangée de un fauteuil
- Parcours de tous les entiers `i` entre `2` et `n` (inclus) et calcul des valeurs de `T[i][0]` et `T[i][1]`
- Retour du nombre de dispositions cherché

Partie C : Affichage des dispositions valides

Après avoir réussi à compter les dispositions valides, Bertrand souhaite désormais les afficher.

Pour ce faire il souhaiter écrire en `python` une fonction `afficher`. Cette fonction récursive fonctionnera de la manière suivante :

- Elle prend en argument un entier `n` correspondant au nombre de sièges de la rangée et une chaîne de caractères `dispo` initialement vide
- Si la valeur de `n` est égale à `0`, elle affiche le contenu de `dispo`
- Sinon, elle teste la valeur de `dispo` :
  - Si cette chaîne est vide ou si son dernier caractère est égal à `0` elle s'appelle deux fois :
    - ◇ une première fois avec les arguments `n-1` et `dispo+"0"`
    - ◇ une seconde avec les arguments `n-1` et `dispo+"1"`
  - Sinon elle s'appelle avec les arguments `n-1` et `dispo+"1"`

1. Écrire le code de la fonction `afficher`.

Exercice 2 :

Partie A : Observation des processus

Un utilisateur d'un système *linux* utilise la commande `ps -el`. On rappelle que les options `-el` indiquent que l'on souhaite lister tous les processus (`e` pour *every*) et afficher beaucoup d'informations (`l` pour *long*).

Il obtient le retour ci-dessous :

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  1  80   0 - 40059 -      ?        00:00:03 systemd
1 S     0     2     0  0  80   0 -     0 -      ?        00:00:00 kthreadd
1 I     0     3     2  0  60 -20 -     0 -      ?        00:00:00 rcu_gp
1 I     0     4     2  0  60 -20 -     0 -      ?        00:00:00 rcu_par_gp
1 I     0     5     2  0  80   0 -     0 -      ?        00:00:00 kworker/0:0-ev
1 I     0     6     2  0  60 -20 -     0 -      ?        00:00:00 kworker/0:0H-k
1 I     0     7     2  0  80   0 -     0 -      ?        00:00:00 kworker/0:1-ev
```

FIGURE 1 – Sortie de l'appel `ps -el`

1. Que signifie l'acronyme `PID` ?

2. Que signifie l'acronyme `PPID` ?

3. Combien de processus sont actuellement en exécution sur cette machine ?

On rappelle les commandes du terminal suivantes :

- `ps -C <commande>` : permet de récupérer le `PID` du processus associé à la `<commande>`
- `ps -ppid <PPID>` : permet de lister les processus dont le `PPID` est égal à `<PPID>`

L'utilisateur effectue donc les commandes suivantes (on fournit aussi les résultats) :

```
$ ps -C firefox
2978
$ ps --ppid 2978
PID    TTY    TIME       COMMAND
3379   tty2   00:00:05   Pivileged Cont
3417   tty2   00:00:05   WebExtensions
3455   tty2   00:00:05   Web Content
3501   tty2   00:00:05   Web Content
```

4. Combien de sous-processus ont pour parent le processus associé à `firefox` ?

La commande `kill -15 <PID>` permet de terminer un processus désigné par son `<PID>`. D'autre part, la documentation de `firefox` indique que la commande `Web Content` permet d'ouvrir et de gérer un onglet dans le navigateur.

5. Quelle commande doit-on saisir dans le terminal afin de fermer le dernier onglet ouvert dans `firefox` ?

Partie B : Exécution des processus

Trois processus P_1, P_2 et P_3 se partagent trois ressources R_1, R_2 et R_3. Dans l'état initial :

- le processus P_1 a obtenu la ressource R_1 et demande la ressource R_2
- le processus P_2 a obtenu les ressources R_2 et R_3
- le processus P_3 demande les ressources R_1 et R_3

Cette situation peut être modélisée par le graphe orienté suivant :

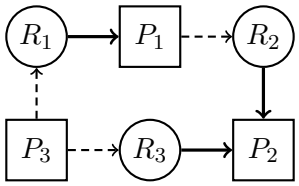

FIGURE 2 – Exemple de relations processus/ressources

Dans ce graphe, une arête pointant d'un processus vers une ressource exprime le fait que ce processus demande cette resource. *A contrario*, une arête pointant d'une ressource vers un processus indique que le processus a obtenu le contrôle de cette ressource pour s'exécuter.

1. Proposer, si possible, un ordre d'exécution des processus permettant à la machine de ne pas être bloquée. Justifier.

On considère désormais la situation suivante :

- Un processus P_1 a obtenu la ressource R_1 et demande les ressources R_2 et R_3
- le processus P_2 a obtenu la ressources R_2 et demande les ressources R_3 et R_1

2. a. Représenter cette situation à l'aide d'un graphe respectant la nomenclature décrite ci-dessus.

b. Proposer, si possible, un ordre d'exécution des processus permettant à la machine de ne pas être bloquée. Justifier.

Partie C : Gestion des priorités

On s'intéresse à la gestion des priorités des processus.

Une des méthodes possibles est de donner une valeur de *priorité* strictement positive à chaque processus. Le système d'exploitation exécutera le processus ayant la priorité la plus importante. On suppose que les processus ont tous des valeurs de priorité différentes les unes des autres.

Il faut donc déterminer facilement quel processus a la priorité la plus grande. On peut utiliser la structure de *tas-max*. Cette structure de données est définie ainsi :

- un *tas-max* est un arbre binaire
- tous les niveaux d'un *tas-max* sont complètement remplis sauf éventuellement le dernier qui est impérativement rempli de gauche à droite
- chaque nœud a une valeur supérieure à celle de tous ses enfants

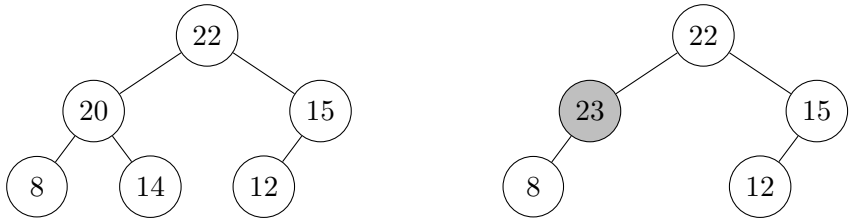

FIGURE 3 – Un *tas-max* à gauche et un contre-exemple à droite

Ainsi dans le contre-exemple de la figure 3, on observe qu'un des nœuds ne respecte pas la condition d'ordre et que le dernier rang n'est pas bien construit.

Une telle structure de données peut être implémentée à l'aide d'un tableau dont on numérote les cellules en à partir de l'indice 1. Dans ce cadre :

- On place la valeur `0` dans la cellule d'indice 0. Cette valeur sera ignorée dans les différents traitements
- la racine du *tas-max* est dans la cellule d'indice 1
- si l'on considère un nœud dont la valeur est stockée dans la cellule n :
  - la valeur de son fils gauche est stockée dans la cellule $2n$
  - la valeur de son fils droit est stockée dans la cellule $2n + 1$

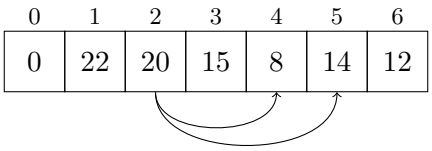

FIGURE 4 – Représentation du *tas-max* de la figure 3 dans un tableau

1. Écrire le tableau représentant le *tas-max* de la figure 5.

2. Dessiner le *tas-max* représenté par le tableau de la figure 6.

3. On considère un nœud dont la valeur est stockée dans la cellule d'indice 54 d'un tableau.
   a. Ce nœud est-il le fils gauche ou droit de son père ?
   b. Dans quelle cellule se trouve son père ?

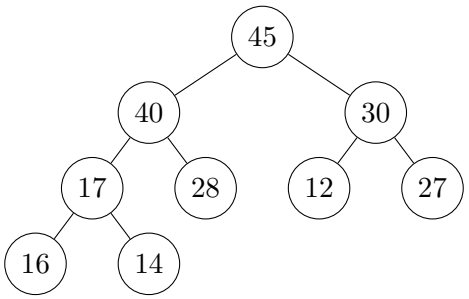

FIGURE 5

0	1	2	3	4	5	6	7	8
0	35	30	12	20	23	10	5	2

FIGURE 6

c. Dans quelle cellule se trouve son grand-père (le père de son père) ?

Lors de la construction d'un *tas-max*, on commence par ajouter la nouvelle valeur à la fin du tableau. Cette opération peut rompre la règle d'ordre si cette valeur est supérieure à celle d'au moins un de se ses ascendants. Il faut dans ce cas *tasser* l'arbre afin de placer la nouvelle valeur à sa place. Pour ce faire on la fait remonter de proche en proche jusqu'à ce que son parent lui soit supérieur.

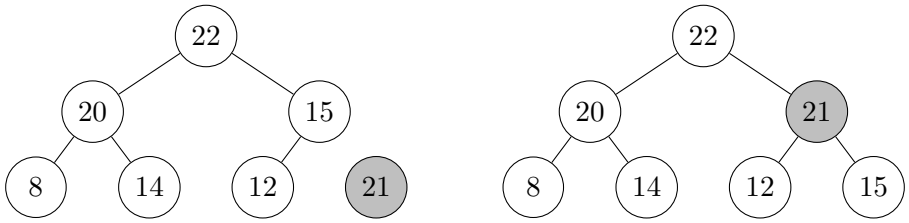

FIGURE 7 – Insertion de la valeur 21 dans le *tas-max*

4. Écrire le tableau représentant le *tas-max* de la figure 6 après insertion de la valeur 18.

Le pseudo-code de la fonction d'insertion est donné ci-dessous. Son fonctionnement correspond à celui expliqué plus haut.

```
Fonction insertion(tas, valeur) :
  """
  Insère la valeur dans le tableau tas représentant un tas-max
  Le tableau respecte la condition d'ordre avant l'appel de la
     fonction
  Renvoie un tableau respectant la condition  l'ordre avec la
     valeur inserée
  """

  # On ajoute valeur à la fin du tableau
  Ajouter valeur à la fin de tas

  # L'indice étudié
  i = longueur(tas) - 1

  Tant que i != 1 et valeur > tas[i//2] :
```

```
    # i//2 est la division entière
    Echanger les valeurs de tas[i] et tas[i//2]
    i = i // 2

  Renvoyer tas
```

5. Écrire le code `python` de la fonction `insertion` décrite ci-dessus.

On possède désormais une structure de *tas-max* satisfaisante. L'élément maximal est donc toujours à l'indice 0. Lorsqu'il doit déterminer quel processus exécuter en priorité, le système d'exploitation n'a qu'à piocher ce premier élément. Plutôt que d'ôter cette valeur du tableau (opération coûteuse car il faudrait "décaler" toutes les autres valeurs d'un cran), on échange cette valeur avec la dernière que l'on supprime. Cette opération risque toutefois de rompre la structure d'ordre comme indiqué dans la figure 8.

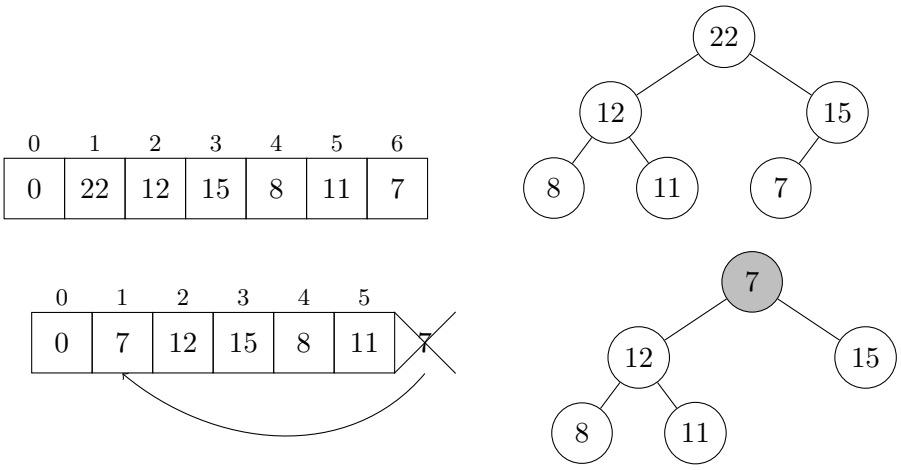

FIGURE 8 – Un *tas-max* avant et après avoir ôté la valeur maximale

6. Compléter sur cet énoncé la fonction `retablir` ci-dessous écrite en `python`. Cette fonction prend en argument un tableau codant un *tas-max* dans lequel il est possible que la première valeur rompe l'ordre. Il faut alors faire "descendre" cette valeur. La fonction renvoie le tableau `tas` modifié.

```
def retablir(tas) :
  # La valeur a descendre
  v = tas[1]
  # l'indice en cours
  i = 1

  # Tant qu'il y a un fils gauche
  while 2*i < ..............  :
    # l'indice du fils-gauche
    i_g =  ..............
    # l'indice du fils-droit
    i_d =  ..............
    # Selection du fils ayant la priorite maximale
    i_max = i_g
    if i_d < len(tas) and tas[i_g] .... tas[i_d] :
      i_max = ..............
    # Si la racine est superieure au fils maximal, on arrete
    if v > tas[i_max] :
```

```
            break
        else :
            # on place la valeur d'indice i_max en i
            tas[..............] = tas[..............]
            # On etudie desormais i_max
            i = ..............
    # On place la valeur a descendre en i
    t[i] = v

    # on retourne le tas
    return ..............
```