Polytech Nice-Sophia
SI 3

Contrôle d'ASD

Durée : 1 heure. Aucun document autorisé.

Année 2017-2018
mercredi 23 mai 2018

Nom :	**Prénom :**	**Groupe :**

Le barême est indicatif et suceptible d'être ajusté. Durant toute l'épreuve, il ne sera répondu à ***aucune*** *question.*

1 Tris (5 points)

1.1 Tri par insertion (2)

Question 1.1.1 Complétez le diagramme suivant en donnant l'état du tableau après chaque étape intermédiaire du tri par insertion.

init	7	3	1	4	8	9	0	6	5	2
	3	7	1	4	8	9	0	6	5	2
	1	3	7	4	8	9	0	6	5	2
	1	3	4	7	8	9	0	6	5	2
	1	3	4	7	8	9	0	6	5	2
	1	3	4	7	8	9	0	6	5	2
	0	1	3	4	7	8	9	6	5	2
	0	1	3	4	6	7	8	9	5	2
	0	1	3	4	5	6	7	8	9	2
end	0	1	2	3	4	5	6	7	8	9

Question 1.1.2 Quelle est la complexité asymptotique du tri par insertion sur un tableau de n éléments dans le meilleur des cas et dans le pire des cas ? Expliquez.
Dans le meilleur de cas, le tableau est déjà trié, et les n étapes sont en $\mathcal{O}(1)$. Donc la complexité est en $\mathcal{O}(n)$.
Dans le pire des cas, le tableau est trié dans l'ordre inverse, il faut donc faire k permutations et comparaisons à l'étape k. Donc la complexité est en $\mathcal{O}(n^2)$.

1.2 Tri par tas (3)

Question 1.2.1 Complétez le diagramme suivant en donnant l'état du tableau après chaque étape intermédiaire du tri par tas.

init	7	3	1	4	8	9	0	6	5	2
heapify	9	8	7	6	3	1	0	4	5	2
	8	7	7	5	3	1	0	4	2	9
	7	6	2	5	3	1	0	4	8	9
	6	5	2	4	3	1	0	7	8	9
	5	4	2	0	3	1	6	7	8	9
	4	3	2	0	1	5	6	7	8	9
	3	1	2	0	4	5	6	7	8	9
	2	1	0	3	4	5	6	7	8	9
	1	0	2	3	4	5	6	7	8	9
end	0	1	2	3	4	5	6	7	8	9

Question 1.2.2 Quelle est la complexité asymptotique du tri par tas sur un tableau de n éléments dans le meilleur des cas et dans le pire des cas ? Expliquez.
Dans tous les cas, il faut faire un heapify qui est en $\mathcal{O}(n)$ et à chaque étape (il y en a $n-1$) on enlève l'extremum du tas, ce qui est en $\mathcal{O}(\log(n))$. Donc la complexité est toujours en $\mathcal{O}(n \cdot \log(n))$.

Question 1.2.3 Quel est l'avantage du tri par tas par rapport au tri fusion ? Expliquez.
Le tri par tas est "en place" : on a pas besoin d'un second tableau auxilliaire pour faire le tri, il consomme donc moins de mémoire

2 Tas (3 points)

Question 2.1 Compléter le diagramme suivant en donnant l'état du tas sous forme de tableau **après** chaque opération indiquée à gauche. On considère ici un tas-min (le minimum est accessible en $\mathcal{O}(1)$) disposant de deux opérations usuelles : `push` pour ajouter une valeur sur le tas et `pop` pour enlever le minimum.

`push 9`	9										
`push 7`	7	9									
`push 5`	5	9	7								
`push 3`	3	5	7	9							
`push 8`	3	5	7	9	8						
`push 2`	2	5	3	9	8	7					
`push 1`	1	5	2	9	8	7	3				
`push 4`	1	4	2	5	8	7	3	9			
`push 6`	1	4	2	5	8	7	3	9	6		
`push 10`	1	4	2	5	8	7	3	9	6	10	
`push 12`	1	4	2	5	8	7	3	9	6	10	12
`pop`	2	4	3	5	8	7	12	9	6	10	
`push 14`	2	4	3	5	8	7	12	9	6	10	14
`pop`	3	4	7	5	8	14	12	9	6	10	
`push 0`	0	3	7	5	4	14	12	9	6	10	8
`pop`	3	4	7	5	8	14	12	9	6	10	
`pop`	4	5	7	6	8	14	12	9	10		
`pop`	5	6	7	9	8	14	12	10			
`pop`	6	8	7	9	10	14	12				
`pop`	7	8	12	9	10	14					

Question 2.2 Quelle est la complexité asymptotique des méthodes `push`, `pop` et `heapify` pour un tas de n éléments ?

`push` est en $\mathcal{O}(\log(n))$ (on fait un parcours de la dernière feuille jusqu'à la racine dans le pire des cas)

`pop` est en $\mathcal{O}(\log(n))$ (on fait un parcours de la racine jusqu'à une feuille dans le pire des cas)

`heapify` est en $\mathcal{O}(n)$ si on utilise le bon algorithme (`percolate_down` en partant de la moitié du tableau et en remontant jusqu'au premier élément).

3 Tas optimisé pour la suppression (4 points)

Le problème des tas est que la suppression d'un élément autre que l'extremum n'est pas possible de manière efficace. On propose ici d'ajouter à un tas, la possibilité de supprimer un élément quelconque du tas rapidement. Pour cela, on utilise un dictionnaire `indexes` qui à chaque valeur dans le tas associe sa position dans le tableau qui représente le tas.

```
class BinaryHeap:
  def __init__(self):
    self.tab=[]
    self.indexes={}

  def swap(self,i,j):
   self.tab[i],self.tab[j]=self.tab[j],self.tab[i]
   pass

  def percolateUp(self,i): ...
  def percolateDown(self,i): ...
  def push(self,v):
    self.tab.append(v)
    self.indexes[v] = len(self.tab) - 1
    self.percolateUp(len(self.tab) - 1)

  def pop(self): ...
```

Question 3.1 La méthode `swap` donnée dans l'énoncé n'est pas satisfaisante : elle ne met pas à jour `indexes`. Écrire ici la ou les lignes nécessaires (à la place du `pass`) pour que la méthode `swap` fonctionne correctement.

```
  def swap(self,i,j):
     self.tab[i],self.tab[j]=self.tab[j],self.tab[i]


     self.indexes[self.tab[i]]=i
     self.indexes[self.tab[j]]=j
```

Question 3.2 Écrire la méthode `delete` qui efface une valeur dans le tas de manière efficace tout en conservant les bonnes propriétés de la structure de données (pour effacer une entrée associée à une valeur dans un dictionnaire on peut utiliser `del dictionnaire[valeur]`)

```
  def delete(self,v):


    i=self.indexes[v]
    self.swap(i,len(self.tab)-1)
    self.tab.pop()
    del self.indexes[v]
    if i<len(self.tab):
      self.percolateUp(i)
      self.percolateDown(i)
```

4 Connexité (4 points)

On considère une classe `Graph` représentant des graphes non dirigés, et on souhaite avoir une méthode `connected` qui renvoie un booléen nous indiquant si le graphe est connecté (une seule composante connexe) ou non. La classe `Graph` dispose déjà d'une méthode `vertex_count` (et `edge_count`) qui renvoie le nombre de sommets (respectivement arêtes) dans le graphe et d'une méthode `random_vertex` qui renvoie un sommet du graphe.

Question 4.1 Complétez la fonction `aux` et la fonction `connected`. La complexité asymptotique de votre solution doit être optimale. Voici le code provisoire de la fonction :

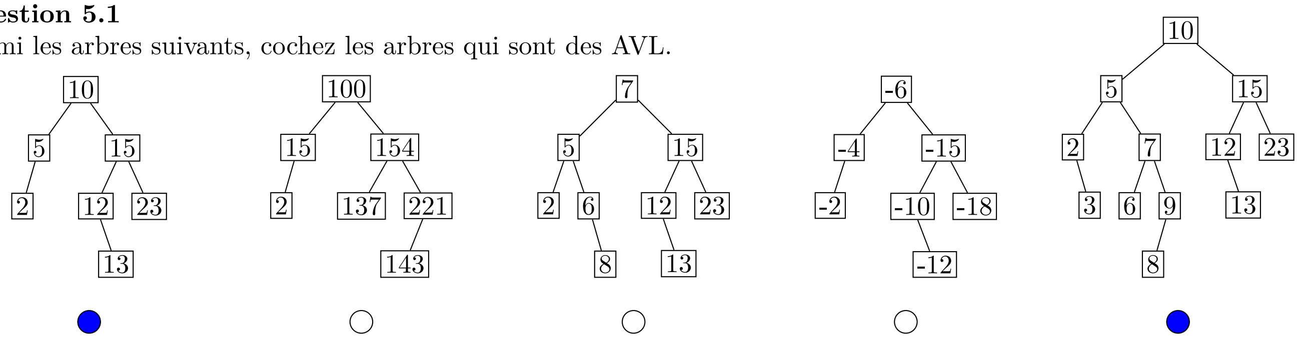
```
def connected(self):
   marked=set()
   def aux(v) :
       marked.add(v)
       for n in self.adjacents(v) :


         if n not in marked: aux(n)

   aux(self.random_vertex())
   return self.vertex_count() == len(marked)
```

5 AVL (4 points)

Question 5.1
Parmi les arbres suivants, cochez les arbres qui sont des AVL.



Question 5.2 *Deux arbres à dessiner*
On considére un AVL vide auquel on ajoute successivement les valeurs 10, 20, 30, 40, 50, 60. Dessiner l'abre obtenu après ces 6 opérations. À partir de ce résultat, on ajoute ensuite successivement 35, 33, 36, 37, 38. Dessiner l'arbre obtenu après ces 11 opérations.