Algorithmique - LI0436
Contrôle continu - Devoir sur table
Documents non autorisés

Durée 2h

05 Mars 2009

1 Tas Ternaire

Pour réduire la hauteur du tas dans le cadre d'un tri par tas on propose d'utiliser un *tas ternaire*. Dans ce type de tas chaque noeud possède trois fils. La valeur de chaque noeud est inférieure à celle de ses trois fils.

1. Représentez le tas ternaire construit à partir des éléments $[3, 6, 1, 13, 17, 18, 2, 7, 42, 15, 9]$
2. Expliquez comment stocker un tas ternaire dans un tableau.
3. Écrivez une fonction qui supprime le plus petit élément d'un tas ternaire à p éléments et réorganise le reste des éléments en un tas contenant $p - 1$ éléments.
4. Donnez l'algorithme de tri par tas basé sur l'utilisation d'un tas ternaire. Si vous le désirez, vous pouvez utiliser la fonction définie à la question précédente.
5. Est-ce que l'utilisation d'un tas ternaire améliore la complexité en terme de **comparaisons** (pire cas/meilleur cas) ?

2 Recherche d'un mot

On s'intéresse au problème de la recherche d'un mot dans un texte. Pour ce faire on considère un texte comme un tableau de caractères au sein duquel on cherche une suite spécifique de caractères (qu'on appellera *facteur* plutôt que mot).

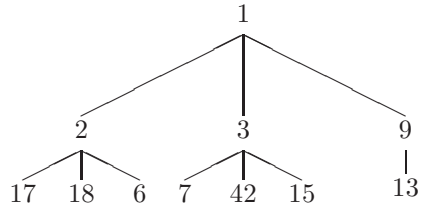
On considère que le texte est de longueur N et le facteur de longueur M.

1. Écrivez une fonction qui répond `VRAI` si le facteur recherché se trouve à la position i dans le texte, et `FAUX` sinon.
2. Donnez un algorithme pour une fonction à deux arguments (le texte et le facteur à trouver) qui répond `VRAI` si le facteur est présent au moins une fois dans le texte et `FAUX` sinon. Si vous le désirez, vous pouvez vous servir de la fonction définie à la question précédente.
3. Évaluez la complexité de votre algorithme.

A Correction

A.1 Tas ternaire

1.



2. On procède de façon similaire au stockage d'un tableau binaire dans un tableau. Chaque emplacement du tableau correspond à un noeud. Les emplacements de $(n+2)//3$ à n sont les feuilles de l'arbre. À un noeud interne à la position i on associe trois fils situés à $3i+1$, $3i+2$ et $3i+3$. Le tas précédent a comme représentation : $[1, 2, 3, 9, 17, 18, 6, 7, 42, 15, 13]$

3. La correction s'inspire de la méthode trouvée durant le TD avant l'examen.

   On suppose qu'on possède une fonction `switch(tab,i,j)` qui échange les valeurs d'un tableau `tab` situées aux index i et j. La fonction `makeTas` prend trois arguments :
   - La liste contenant les éléments du tas à considérer
   - La longueur du tas à considérer

```
int[] makeTas(int[] tas, int longueur){
  int[longueur] newTas;
  for(int i=0;i<newTas.length-1;i++)
    newTas[i] = tas[i];
  pos=0;
  while((pos<(liste.length+2)//3) && (tas[pos]>tas[3*pos+1])
       ||(tas[pos]>tas[3*pos+2])||(tas[pos]>tas[3*pos+3])){
   if(tas[pos]>tas[3*pos+1]){
     switch(tas,pos,3*pos+1);
     if(tas[pos]>tas[3*pos+2]){
       switch(tas,pos,3*pos+2)
       if(tas[pos]>tas[3*pos+3])
         switch(tas,pos,3*pos+3)
     }
     else if(tas[pos]>tas[3*pos+3]){
       switch(tas,pos,3*pos+3)
     }
     pos=3*pos+1;
   }
   else if(tas[pos]>tas[3*pos+2]){
     switch(pos,3*pos+2);
     if(tas[pos]>tas[3*pos+3]){
       switch(tas,pos,3*pos+3)
     }
     pos=3*pos+2;
   }
   else{
     switch(pos,3*pos+3);
```

```
      pos=3*pos+3;
    }
   }
  }
```

4. Ici // désigne la division entière.

```
int[] triTasTern(int[] liste){
  int[liste.length] result;
  int[liste.length] tas;
  //Construction du premier tas
  //Initialisation
  for(i=0;i<(liste.length+2)//3;i++)
    tas[i]=-1
  for(i=(liste.length+2)//3;i<liste.length;i++)
    tas[i]=liste[i]

  //Construction
  for(i=0;i<(liste.length+2)//3;i++){
    tas[0]=liste[i];
    tas = makeTas(tas,tas.length)
  }

  //Dépilage successifs de l'élément du dessus
  for(int i=0;i<liste.length;i++){
   result[i]=tas[0];
   tas[0]=tas[tas.length-1];
   tas = popTas(tas,tas.length-1);
  }
  return result;
}
```

5. Le nombre de comparaisons reste identique à celui d'un tri par tas binaire. Lorsqu'on réorganise les tas il faut le même nombre de comparaisons pour trouver le nouvel emplacement d'un élément.

A.2 Recherche d'un mot

1.
```
boolean findFacteurPos(char[] facteur, char[] txt, int pos){
  if(pos+facteur.length+1>txt.length){
    return false;
  }
  for(int i=pos;i<pos+facteur.length;i++){
    if(facteur[i-pos]!=txt[i]){
      return false;
    }
  }
  return true;
}
```

2.
```
boolean findFacteur(char[] facteur, char[] txt){
```

```
  for(int i=0;i<txt.length-facteur.length;i++){
    if(findFacteurPos(facteur,txt,i)){
      return true;
    }
  }
  return false;
}
```

3. Dans le pire des cas chacune des boucles se termine :
   - Grande boucle de `findFacteur` : $O(N - M)$
   - Petite boucle de `findFacteurPos` : $O(M)$
   - TOTAL : $O((N - M) \times M)$

   Dans le meilleur des cas le facteur est trouvé à la première position, il faut M comparaisons pour s'en assurer.