Cours Info - 11

Algorithmes de recherche

D.Malka

MPSI 2016-2017

Sommaire

Sommaire

La structure de Tableau

Définition

Tableau

Un *tableau* est une suite de valeurs stockées dans des cases mémoires contigües.

En Python, la structure de données correspondant aux tableaux est le type *list*. Python autorise les listes stockant des variables de types différents.

Nous nous restreindrons à des tableaux dont les éléments sont tous de même type.

La structure de Tableau

Exemples

Tableau d'entiers

indice	0	1	2	3	4	5	6
elt	4	75	6	93	101	3	12

Tableau de caractères

indice	0	1	2	3	4	5
elt	'a'	'e	'i'	'o'	'u	'y'

La structure de Tableau

Création, manipulation

Quelques rappels en Python

- **Création `T=[1,2,3,....]`**
- **Lecture de l'élément d'indice i : `T[i]`**
- **Modification de l'élément d'indice i : `T[i]=3`**
- **Insertion en queue : `T.append(elt)`**
- **Insertion : `T.insert(indice,elt)`**
- **Suppression : `del(T[indice])` ou `T.remove(elt)`**

La structure de Tableau

Création, manipulation

Quelques rappels en Python

- **Slicing : `T[i:j]` renvoie un sous-tableau contenant les éléments d'indice i(inclus) à j(exclu) de T**
- **Concaténation : [1,2,3]+[3,2,1] $\rightarrow$ [1,2,3,3,2,1]**
- **Revoir le chapitre 5 ...**
- **et la documentation Python**

La structure de Tableau

Parcours des éléments d'un tableau

On considère le tableau (ou la liste) T. Comment le parcourir en Python ?

Méthode 1 :

```
1 for i in range(len(T)):
2     T[i]
```

Méthode 2 :

```
1 for elt in T:
2     elt
```

Méthode 3 :

```
1 for indice, elt in enumerate(T):
2     elt,indice
```

Sommaire

Recherche séquentielle d'un nombre dans une liste

Principe

Recherche séquentielle : principe

- **Parcourir le tableau**
- **Comparer chaque élément parcouru à l'élément recherché**
- **Si l'élément est trouvé, quitter la boucle et renvoyer l'indice de l'élément (plus précisément de la 1ère occurrence)**
- **Arrivé au dernier élément du tableau, quitter la boucle et renvoyer NULL**

Recherche séquentielle d'un nombre dans une liste

Algorithme

Algorithme 1 : Recherche séquentielle d'une valeur

```
1 fonction Recherche_sequentielle(T,x)
2 pour i de 0 à longueur(T)-1 faire
3 |   si T[i]==x alors
4 |   |   retourner i
5 retourner NULL
```

Complexité : $O(n)$ où n est la longueur du tableau T.

Recherche séquentielle d'un nombre dans une liste

Algorithme

Implémentation en Python :

```
1  def recherche_seq(T,x):
2      n=len(T)
3      for i in range(n)
4          if T[i]==x:
5              return i
6      return None
```

Recherche dichotomique

Principe

Existe-t-il un algorithme plus efficace que la recherche naïve ?

OUI si la liste est triée.

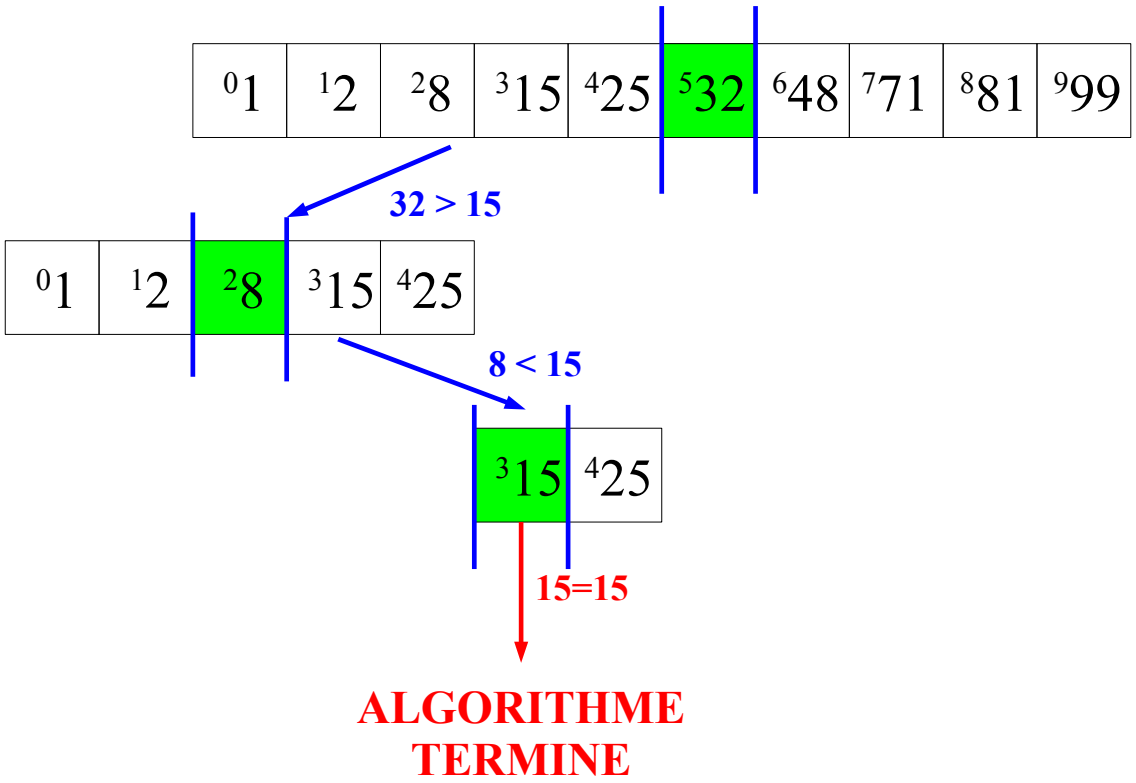
C'est la *recherche dichotomique*.

Recherche dichotomique

Principe

Paradigme : « diviser pour régner »

Recherchons 15 dans le tableau d'entier croissant suivant :



Recherche dichotomique d'un nombre dans une liste triée

Algorithme

Algorithme 2 : Recherche dichotomique d'une valeur

Entrées : Entier x, tableau trié par ordre croissant T

Sorties : Indice de x dans T

```
1  fonction Recherche_dichotomique(T,x)
2  g=0
3  d=longueur(T)-1
4  tant que g<=d faire
5      m=(g+d)/2
6      si T[m]<x alors
7          g=m+1
8      sinon si T[m]>x alors
9          d=m-1
10     sinon
           /*T[m]==x*/
11         retourner m
12 retourner NULL
```

Recherche dichotomique d'un nombre dans une liste triée

Implémentation en Python

```
1  def rech_dicho(x,T):
2      g=0
3      d=len(T)-1
4      while g<=d:
5          m=(d+g)//2
6          if x>T[m]:
7              g=m+1
8          elif x<T[m]:
9              d=m-1
10         else :#x==T[m]:
11             return m
12
13     return None
```

Recherche dichotomique d'un nombre dans une liste triée

Terminaison

Terminaison

Montrons que l'algorithme de recherche dichotomique termine.

On considère la suite $\{u_k = d_k - g_k\}$.

Dans la boucle, $\{u_k\}$ est une suite :

- **Entière**
- **Positive car** $g_k \leq m_k \leq d_k$
- **Strictement décroissante (**$u_{k+1} < u_k$**) car** $\forall\, k$**,** $d_{k+1} < d_k$ **ou** $g_{k+1} > g_k$

Donc l'algorithme termine.

Recherche dichotomique d'un nombre dans une liste triée

Exactitude

Montrons que si x est dans T alors l'algorithme renvoie i l'indice de x dans le tableau.

Pour cela, montrons que $\mathcal{P}_k : x \in [T[g_k], T[d_k]]$ ou $x = T[m_k]$ est un invariant de boucle.

Initialisation

Pour $k = 0$:

- $d_0 = n - 1$, $g_0 = 0$
- **Par hypothèse** $x \in T$ **donc** $x \in [T[g_0], T[d_0]]$
- $\mathcal{P}_0$ **est vraie**

Recherche dichotomique d'un nombre dans une liste triée

Exactitude

Montrons que $\mathcal{P}_k \Rightarrow \mathcal{P}_{k+1}$

Hérédité – Cas 1 : $x = T[m_{k+1}]$

Cas trivial : $\mathcal{P}_{k+1}$ est vraie.

Recherche dichotomique d'un nombre dans une liste triée

Exactitude

Montrons que $\mathcal{P}_k \Rightarrow \mathcal{P}_{k+1}$.

Hérédité – Cas 2 : $x < T[m_{k+1}]$

Si $x < T[m_{k+1}]$ alors :

- $m_{k+1} = \dfrac{d_k + g_k}{2}$
- $g_{k+1} = g_k$ **or par hypothèse** $\mathcal{P}_k$ **est vrai donc** $x \geq T[g_k]$ **donc** $x \geq T[g_{k+1}]$
- $d_{k+1} = m_{k+1} - 1$ **or** $x < T[m_{k+1}]$ **donc** $x \leq T[m_{k+1} - 1]$ **soit** $x \leq T[d_{k+1}]$
- **Les deux dernières assertions sont équivalentes à** $x \in [T[g_{k+1}], T[d_{k+1}]]$ **soit à** $\mathcal{P}_{k+1}$
- **Donc** $\mathcal{P}_k \Rightarrow \mathcal{P}_{k+1}$

Recherche dichotomique d'un nombre dans une liste triée

Exactitude

Montrons que $\mathcal{P}_k \Rightarrow \mathcal{P}_{k+1}$.

Hérédité – Cas 3 : $x > T[m_{k+1}]$

Si $x > T[m_{k+1}]$ alors :

- $m_{k+1} = \dfrac{d_k + g_k}{2}$
- $d_{k+1} = d_k$ **or par hypothèse** $\mathcal{P}_k$ **est vrai donc** $x \leq T[d_k]$ **donc** $x \leq T[d_{k+1}]$
- $g_{k+1} = m_{k+1} + 1$ **or** $x > T[m_{k+1}]$ **donc** $x \geq T[m_{k+1} + 1]$ **soit** $x \geq T[g_{k+1}]$
- **Les deux dernières assertions sont équivalentes à** $x \in [T[g_{k+1}], T[d_{k+1}]]$ **soit à** $\mathcal{P}_{k+1}$
- **Donc** $\mathcal{P}_k \Rightarrow \mathcal{P}_{k+1}$

Recherche dichotomique d'un nombre dans une liste triée

Exactitude

Finalement :

- **Soit x a été trouvé avant la dernière itération.**
- **Soit à la dernière itération $d = g = m$ et donc d'après l'invariant de boucle $x = T[m]$ donc x est trouvé.**
- **La démonstration est complète si l'algorithme retourne NULL si x n'est pas dans le tableau ce qui assuré par la négation de la condition $x == T[m]$ et par la terminaison de l'algorithme.**

Recherche dichotomique d'un nombre dans une liste triée

Complexité

La complexité de la recherche dichotomique est $O(\log_2(n))$ où n est la taille du tableau.

Démonstration par récurrence sur la diapo suivante.

Recherche dichotomique d'un nombre dans une liste triée

Complexité

Preuve

Montrons par récurrence qu'à l'itération k, $l_k = d_k - g_k \leq \frac{n}{2^k}$ où n est la longueur du tableau.

- Initialisation : pour $k = 0$, $l_0 = n - 1 - 0 + 1 = n$ cqfd
- Hérédité :
  - $\mathcal{P}_k : l_k < \frac{n}{2^k}$ vraie
  - montrons $\mathcal{P}_{k+1} : l_{k+1} < \frac{n}{2^{k+1}}$
  - à la $(k+1)^{ieme}$ itération :
  - $m_{k+1} = \frac{d_k + g_k}{2}$ et :
    - soit $d_{k+1} = m_{k+1} - 1$ et $g_{k+1} = g_k$,
    - soit ou $g_{k+1} = m_{k+1} + 1$ et $d_{k+1} = d_k$.
  - cas où $g_{k+1} = m_{k+1} + 1$ et $d_{k+1} = d_k$: $l_{k+1} = d_{k+1} - (m_{k+1} + 1) + 1 = d_k - \left(\frac{d_k + g_k}{2}\right) + 1 = \frac{d_k - g_k}{2}$ or $\frac{l_k}{2} = \frac{(d_k - g_k) + 1}{2}$ donc $l_{k+1} = \frac{l_k}{2} \leq \frac{n}{2^{k+1}}$: cqfd.
  - cas où $d_{k+1} = m_{k+1} - 1$ et $g_k = g_{k+1}$: même raisonnement.
- donc $\forall k$, $l_k < \frac{n}{2^k}$.

A la sortie de la boucle $k = N$, et au pire, $l = 1$ tableau à un élément. Donc $1 \leq \frac{n}{2^N} \Leftrightarrow N \leq \log_2(n)$. La complexité au pire de l'algorithme est donc $O(\log_2(n))$.

Sommaire

Recherche d'un mot dans une chaîne de caractères

Le problème

Recherche d'un motif dans un texte

On souhaite rechercher toutes les occurrences d'une chaine de caractère `motif` de longueur m dans une chaine de caractères `texte` de longueur n.

On supposera $m \leq n$.

Recherche d'un mot dans une chaîne de caractères

Algorithme naïf

Algorithme 3 : Recherche d'une chaîne de caractère

Entrées : String motif, string texte

Sorties : Liste des positions du motif (premier caractère)

```
1  fonction Recherche_motif(motif,texte)
2  position_motif=[]
3  n=longueur(texte)
4  m=longueur(motif)
5  pour i allant de 0 à n-m-1 faire
6      j=0
7      tant que j<m et texte[i+j]==motif[j] faire
8          j=j+1
9      si j==m alors
           /*Alors texte[i+j]==motif[j] a toujours été vraie*/
0          Ajouter i à position_motif
1  retourner position_motif
```

Recherche d'un mot dans une chaîne de caractères

Algorithme naïf – Implémentation en Python

```
1  def rech_motif(mot,t):
2
3      liste_pos=[]
4      n=len(t)
5      m=len(mot)
6      for i in range(n-m):
7          j=0
8          while(j<m and t[i+j]==mot[j]):
9              j+=1
10         if j==m:
11             liste_pos.append(i)
12     return liste_pos
```

Recherche d'un mot dans une chaîne de caractères

Algorithme naïf – Complexité

Complexité

- **Unité de mesure : les comparaisons**
- **Boucle for : une boucle `while` $n-m$ fois et $n-m$ comparaisons (embranchement)**
- **Boucle `while` : au pire on trouve à chaque fois le motif $\Rightarrow 2m$ comparaisons**
- **Boucle `while` : au mieux on ne trouve jamais le motif $\Rightarrow 2$ comparaisons**
- **Complexité au pire : $O(m(n-m))$.**
- **Complexité au mieux : $O(n-m)$.**

Sommaire

Recherche du zéro d'une fonction continue monotone

VOIR PARTIE CALCULS NUMERIQUES