Olympiades in de Informatica

http://uclouvain.acm-sc.be/olympiades

Voorbeeldvragen voor het hoger onderwijs

Dit document bevat voorbeeldvragen voor de wedstrijd voor leerlingen uit het hoger onderwijs. Het eerste deel bevat voorbeelden van algoritmische vragen en het tweede deel voorbeelden van meerkeuzevragen. Wij raden u aan om het document *"Introductie tot de algoritmiek"* te lezen om de gegeven oplossingen beter te begrijpen. Voor voorbeelden van logicavragen, kunt u het document *"Voorbeeldvragen voor het middelbaar onderwijs"* raadplegen.

1 Algoritmiek

1.1 Zoek een deelreeks

> Schrijf een programma dat 2 reeksen van karakters inleest van standaard input. De eerste reeks W is een woord bestaande uit de letters (`a-zA-Z`), de tweede reeks P is een motiefje bestaande uit de letters (`a-zA-Z`) en het symbool `_`. Het programma moet nagaan of reeks P een deelreeks is van W, (d.w.z. P komt voor in W), waarbij `_` eender welke letter kan zijn.

Met een dubbele lus kan je dit probleem vrij eenvoudig oplossen. De eerste lus overloopt reeks W om na te gaan waar de verschillende locaties zijn waar het motief zich kan bevinden. De tweede lus test dan of alle karakters van het motief daar voorkomen. We moeten rekening houden met het geval dat P langer kan zijn dan W, en dan **false** terugsturen. Dit wordt geregeld in het tweede deel van de conditie in de `while`-lus.

Algorithme 1 : Zoek een deelreeks

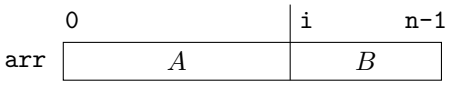
Input : W, een array van karakters uit de verzameling (`a-zA-Z`) met lengte $lenW > 0$, en P, een array karakters uit de verzameling (`a-zA-Z_`) met lengte $lenP$
Output : **true** als P een deelreeks is van W, waarbij `_` elk karakter kan zijn, anders **false**

```
1  // We testen elke positie in W
2  found ← false
3  pos ← 0
4  while not found and pos ≤ lenW − lenP do
5      // Test of P zich bevindt op positie pos
6      match ← true
7      i ← 0
8      while match and i < lenP do
9          if P[i] ≠ '_' and P[i] ≠ W[pos + i] then
10             match ← false
11         i ← i + 1
12     // Stop als P gevonden werd in W
13     if match then
14         found ← true
15     pos ← pos + 1
16 return found
```

1.2 Een array sorteren

Je kan beschikken over twee functies `moveElem` en `minIndex`, waarvan de specificaties hieronder gegeven zijn. Schrijf een algoritme dat toelaat een array van lengte n oplopend te sorteren, **door enkel deze twee functies toe te passen** op de array.
De functie `moveElem (arr, i, j)` laat toe om de elementen op indices `i` en `j` van de array `arr` om te wisselen, waarbij `i` en `j` geldige indices van `arr` zijn. De functie `minIndex (arr, i)` geeft een kleinste element terug van de subarray `arr[i:n-1]`, d.w.z. een index j zodat $\forall k \in [i, n-1] : arr[k] \geq arr[j]$, met `i` een geldige index van `arr`.

Om deze vraag op te lossen moet je eerst goed begrijpen wat de toegelaten functies doen. Eens dat is gebeurd, zal je gemakkelijk een algoritme kunnen schrijven om een array oplopend te sorteren. We vertrekken van het volgende schema:



De array `arr` is verdeeld in twee delen A (indices 0 t/m $i-1$) en B (indices i t/m $n-1$). Alle elementen in A zijn kleiner of gelijk aan alle elementen in B, en A is reeds oplopend gesorteerd:

$$\forall a \in A : \forall b \in B : a \leq b \qquad \forall i \in [0, i-2] : arr[i] \leq arr[i+1]$$

We kunnen dan een algoritme schrijven in drie stappen:

1. Vind de index van een kleinste element in B (d.w.z. een index j zodat $\forall i \in [i, n-1] : arr[j] \leq arr[i]$) door gebruik te maken van `minIndex` ;
2. Wissel het element op index j om met dat op index i gebruikmakend van `moveElem` ;
3. Verhoog de waarde van i met een en ga verder met de lus zolang $i < n-1$.

In pseudo-code neemt het algoritme dus de volgende vorm aan :

Algorithme 2 : Sorteren van een array.

Input : arr, een array van lengte $n > 0$
Output : Een array met dezelfde elementen als arr, maar oplopend gesorteerd

```
1 i ← 0
2 while i < n − 1 do
3 |   j ← minIndex(arr, i)
4 |   moveElem(arr, i, j)
5 |   i ← i + 1
6 return arr
```

We zijn niet verplicht om de waarde van arr expliciet terug te geven. Het algoritme kan zich beperken tot het aanpassen van de waarden in de array die eraan meegegeven wordt als input.

1.3 Vergelijk het eerste en het laatste element van een lijst

Context : Je kan een lijst beschouwen als bestaande uit twee delen: de kop (`H`) en de staart (`T`). Je kan een lijst dus voorstellen als `L = H|T`. De enige operaties die toegelaten zijn op een lijst `L` zijn `head(L)` die je de kop van de lijst teruggeeft, en `tail(L)` die je de staart geeft. De lege lijst wordt voorgesteld als `nil`.

Voorbeeld : Gegeven de lijst `L = 1|2|3|4|5`. De kop van de lijst is `head(L) = 1` en de staart is `tail(L) = 2|3|4|5`. De kop van de staart is `head(tail(L)) = 2` en de staart van de staart is `tail(tail(L)) = 3|4|5`, etc.

Gevraagd : Schrijf een algoritme dat kan testen of het eerste en het laatste element van een niet lege lijst `L` van lengte n gelijk zijn aan elkaar. Je mag bovendien de operaties `head` en `tail` maximaal n keer gebruiken.

Hier moeten we het eerste en laatste element van een lijst opvragen om ze te kunnen vergelijken met elkaar. De enige toegelaten operaties zijn `head` die je het eerste element alvast geeft, en `tail` die de rest van de lijst teruggeeft. We weten ook dat een lege lijst wordt weergegeven door `nil`. Tot slot mogen we `head` en `queue` maximaal n keer oproepen, waarbij n de lengte is van de lijst.

Algorithme 3 : Vergelijking van de eerste en laatste elementen van een lijst

Input : L, een niet-lege lijst
Output : **true** als het eerste en laatste element van L gelijk zijn aan elkaar, anders **false**

```
1 first ← head(L)
2 last ← first
3 tmp ← tail(L)
4 while tmp ≠ nil do
5 |   last ← head(tmp)
6 |   tmp ← tail(tmp)
7 return first = last
```

1.4 Doolhof

Een doolhof wordt voorgesteld door een verzameling van $M \times N$ vakjes. Je kan je in het doolhof verplaatsen van het ene vak naar het andere als ze een zijde delen (diagonale verplaatsingen zijn niet toegestaan). Je krijgt de waarden M en N, en een matrix van grootte $M \times N$ waar de waarde van een vakje $mat[i][j]$ gelijk is aan 0 als het vak leeg is en we kunnen er doorheen, 1 als het geblokkeerd is, 2 als het de startpositie is en 3 als het de eindpositie is. Schrijf een algoritme dat 0 teruggeeft als er geen pad bestaat tussen vertrek en aankomst, en 1 als er wel een pad bestaat.

(*Hulplijn :* een queue kan nuttig zijn)

1.5 Pattern search

Gegeven twee lijsten van gehele getallen `List` en `Sublist`, niet leeg. Schrijf een algoritme dat de kleinste index i van `List` teruggeeft, zodat de deellijst `List[i, i + len (Sublist)[` gelijk is aan `Sublist`. Als zo'n index niet bestaat, geef dan -1 terug.

1.6 De binaire optelling

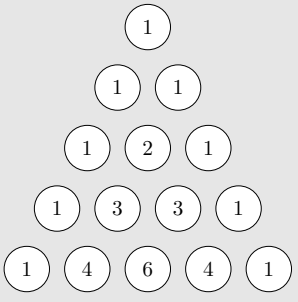
Gegeven twee lijsten `List1` en `List2` van gehele getallen 0 en 1. Schrijf een algoritme dat de som berekent van de twee binaire getallen die voorgesteld worden door deze lijsten. De minst significante bit bevindt zich op index 0.

1.7 Fibonacci

De reeks van Fibonacci is een bekende reeks gehele getallen : $0, 1, 1, 2, 3, 5, 8, \cdots$ De eerste twee elementen van de reeks zijn dus 0 et 1. Elk volgende element is gelijk aan de som van de twee voorgaande elementen. Als we het n-de element voorstellen door Fib_n, dan is $Fib_1 = 0$, $Fib_2 = 1$ en $Fib_{i+2} = Fib_{i+1} + Fib_i$ voor $i > 0$. Schrijf een algoritme dat de n-de term van de reeks van Fibonacci kan berekenen voor alle $n > 0$.

1.8 De driehoek van Pascal

De driehoek van Pascal is een driehoek van getallen die als volgt wordt geconstrueerd: de twee neergaande zijden van de driehoek worden gevuld met 1. Vervolgens is elk overig element van de driehoek gelijk aan de som van de twee elementen erboven. Het begin van deze driehoek ziet er zo uit:



Schrijf een algoritme dat twee parameters L en C aanvaardt, en de waarde berekent van het C-de element op de L-de lijn. Als er geen element is op die plaats, geef dan -1 terug.

1.9 De langste deelreeks

Gegeven twee reeksen A en B. Schrijf een algoritme dat de langste deelreeks vindt van B die gelijk is aan een deelreeks van A. Het algoritme moet 3 waarden teruggeven: i is de index van de deelreeks in A, j is de index van de startpositie van die deelreeks in B, en ℓ is de lengte van de deelreeks.

Bijvoorbeeld, voor de reeksen $A = entrapment$ en $B = traploper$, geeft het algoritme $\langle 2, 0, 4\rangle$ terug, wat overeenkomt met de deelreeks *trap*. Voor reeksen $A = ta$ en $B = ratata$ is het resultaat $\langle 0, 2, 2\rangle$. En dus, als de reeks B geen enkele deelreeks bevat die eveneens deelreeks is van A, geeft het algoritme $\langle x, 0, 0\rangle$ terug, waar x eender welke waarde kan zijn tussen 0 en $n - 1$ (met n is de lengte van reeks A).

2 Multiple Choice

2.1 Machtsverheffing

Gegeven de volgende twee algoritmes `pow_1` en `pow_2` :

Algorithme 4 : `pow_1`

Input : X een reëel getal en N een natuurlijk getal
Output : De waarde van X^N

```
1 if N = 0 then
2 |  result ← 1
3 else
4 |  result ← X * pow_1(X, N − 1)
5 return result
```

Algorithme 5 : `pow_2`

Input : X een reëel getal en N een natuurlijk getal
Output : De waarde van X^N

```
1 if N = 0 then
2 |  result ← 1
3 else if N mod 2 = 1 then
4 |  result ← X * pow_2(X, N − 1)
5 else
6 |  Y ← pow_2(X, N/2)
7 |  result ← Y * Y
8 return result
```

1. Voor dezelfde startgegevens (X etn$N$) die groot genoeg zijn, welk algoritme zal de minste bewerkingen nodig hebben (het minste aantal recursies) ?
2. Als de waarde van X verdubbeld wordt, hoeveel extra recursiestappen worden dan uitgevoerd in algoritme `pow_1` ?
3. Als de waarde van N verdubbeld wordt, hoeveel extra recursiestappen worden dan uitgevoerd in algoritme `pow_2` ?

2.2 Een lijst van delers

We willen een functie schrijven die een array *arr* van strikt positieve gehele waarden aanvaardt, en een array van 5 gehele getallen teruggeeft. $retour[i]$ bevat het kleinste getal van *arr* dat deelbaar is door i, en $retour[0]$ bevat 0. Dit is een skelet van de functie in Java:

```
int[] sortByDivisibility (int[] arr)
{
   int retour[] = new int[5];
   // Vervolledig hier
}
```

Hoe moeten we deze functie vervolledigen?

(a)
```
for (int i = 1; i < 5; i++) {
   if (arr[i] % i == 0 && retour[i] > arr[i]) {
      retour[i] = arr[i];
   }
}
```

(b)
```
for (int x : arr) {
   for (int i = 1; i < 5; i++) {
      if (x % i == 0 && (retour[i] > x || retour[i] == 0)) {
         retour[i] = x;
      }
   }
}
```

(c)
```
for (int x : arr) {
   for (int i = 1; i < 5; i++) {
      if (retour[i] % i && x > retour[i]) {
         retour[i] = arr[i];
      }
   }
}
```