Домашнее задание 1 по алгоритмам LaTeX

Попов Николай

13 февраля 2018 г.

1 a)

Верно.
Пусть C = 1 и N = a. Тогда

$$\log_a n \geqslant \frac{1}{C}$$

для $\forall n \geqslant N$, т.к. a $>$ 1. Отсюда

$$n \leqslant C \cdot n \log_a n$$

T.e. n = O($n \log_a n$).

b)

Неверно.
$\exists \varepsilon > 0 : n \log n = \Omega(n^{1+\varepsilon}) \Leftrightarrow \exists N, C > 0 : \forall n \geqslant N \hookrightarrow$
$n \log n \geqslant C n^{1+\varepsilon} \Leftrightarrow logn \geqslant C n^{\varepsilon}$
Но $\lim\limits_{x \to +\infty} \dfrac{\log n}{n^{\varepsilon}} = 0 \Leftrightarrow \forall \delta > 0 \ \exists N \in \mathbb{N} : \forall n \geqslant N \hookrightarrow \left| \dfrac{\log n}{n^{\varepsilon}} \right| < \delta$
Отсюда $\forall \delta > 0 \hookrightarrow -\delta < \dfrac{\log n}{n^{\varepsilon}} < \delta \Rightarrow \log n < \delta n^{\varepsilon}$
А значит $\nexists$ C : $logn \geqslant C n^{\varepsilon}$ и выражение $\exists \varepsilon > 0 : n \log n = \Omega(n^{1+\varepsilon})$ неверно

2

$g(n) = O(n)$ и $g(n) = \Omega(1) \Leftrightarrow 0 < C_1 \leqslant g(n) \leqslant C_2 n \quad \forall n \geqslant max[N_1, N_2]$
(константы соответственно для нижней и верхней оценок)
$f(n) = O(n^2) \Leftrightarrow \exists N_3, C_3 : \forall n \geqslant N_3 \hookrightarrow f(n) \leqslant C_3 n^2$

1.a)

$h(n) = \Theta(n \log n) \Leftrightarrow \exists N, K_1, K_2 : \forall n \geqslant N \hookrightarrow K_1 n \log n \leqslant h(n) \leqslant K_2 n \log n$.

$$h(n) = \frac{f(n)}{g(n)} \leqslant \frac{C_3 n^2}{C_1} = C n^2$$

T.к. $\forall K \hookrightarrow K n \log n \leqslant C n^2$ при $n \to +\infty$, то оценка $h(n) = \Theta(n \log n)$ возможна.

Достижение этой оценки в примере:
$f(n) = n\sqrt{n}\log n$ и $g(n) = \sqrt{n}$. Получаем, что $h(n) = n\log n$ при соблюдении оценок функций.

1.b)

$h(n) = \Theta(n^3) \Leftrightarrow \exists N, K_1, K_2 : \forall n \geqslant N \hookrightarrow K_1 n^3 \leqslant h(n) \leqslant K_2 n^3$
Т.к. $K_1 n^3 \geqslant n^2$ при $n \to +\infty$, то такая оценка не возможна.

2.

$h(n) = O(n^2)$ при $f(n) = n^2$ и $g(n) = 1$

3

а)

Данные: sum - сумма элементов, length - длина последовательности.
Псевдокод:

```
sum = num = 0
for (i = 1; i <= n; i++) do
{
        read(xi)
        sum += xi
        num ++
}
return(sum/num)
```

Если на входе n чисел, то операций порядка n, т.е. асимптотика $\Theta(n)$
Корректность: чтобы найти среднее арифметическое необходимо знать сумму элементов и их число.

б)

Данные: max - предполагаемое значение максимального элемента, number - число повторов этого числа.
Псевдокод:

```
max = x1;
num = 1
for (i = 2; i <= n; i++) do
{
        read(xi)
        if (xi == max) then num++
        if (xi > max) then {max = xi ; num = 1}
}
return (num)
```

Алгоритм линейный, т.к. операций в цикле $n-1$
Корректность: считывая элементы последовательности один за другим, мы навеврняка не знаем какой ее элемент максимален. Поэтому при каждом считывании нового элемента будем проверять, не является ли он максимальным. При этом для каждого предполагаемого максимального элемента будем подсчитывать число его повторов.(Т.к. последовательность конечна, то ее максимум $\exists$).
P.S.
Индуктивные расширения были совокупностью переменных использованных в этих программках.

в)

Данные: last, num, max, где last - последний считанный элемент последовательности, num - текущая число подряд идущих равных элементов хвоста последовательности, max - максимум из этих длин.

Псевдокод:

```
last = x1
num = 1
for (i = 2; i <= n ; i++) do
{
        if (xi == last) then num ++
        else
        {
                if (num > max) then max = num
                num = 1
        }
        last = xi
}
```

return (max)

Число шагов в цикле n-1, поэтому алгоритм линейный.

Корректность: чтобы найти максимальную длину отрезков из одинаковых элементов, необходимо подсчитать длины каждой из них. При этом если любые два соседних элемента в отрезке равны, то в нем все они равны. При считывании нового элемента, искомый отрезок прекращается, а значит мы знаем егоо длину и можем сравнить ее с максимальной.Длина нового искомого отрезка равна 1.

4

1) Данные : три массива A,B,C,три элемента из каждого из них a,b,c, длины массивов len.a,len.b,len.c, три индекса pos.a,pos.b,pos.c, для обращения к элементу соответствующего массива, num - для хранения ответа, m - вспомогательная.

. **Псевдокод**:

проходим по каждому из массивов, подсчитывая в соответствующих переменных len их длины
$pos.a = pos.b = pos.c = 1$ (индексы с 1)
$num = 0$
do {
if (pos.a == len.a+1)then a = +infinity else a = A[pos.a]
if (pos.b == len.b+1)then b = +infinity else b = B[pos.b]
if (pos.c == len.c+1)then c = +infinity else c = C[pos.c]
m = min(a,b,c)(находим минимальное число из 3-х)
if (a = m) then pos.a++
if (b = m) then pos.b++
if (c = m) then pos.c++
num++
} while not[(pos.a == len.a+1)and(pos.b == len.b+1)and(pos.c == len.c+1)]
return(num)

2)Время работы -в худшем случае(когда во всех трех массивах нет совпадающих элементов) равно $C \cdot (len.a + len.b + len.c)$, т.е. алгоритмм линейный.

3)Док-во корректности: Предположим, что какой-то произволный элемент множества, включающего все элементы обьединения массивов, посчитан дважды. Это, значит, что он на каком-то шагом цикла, находясь в рассматриваемой тройке элементов, был равен минимальному в ней, но его число-позиция pos не была увеличена,что невозможно, либо он находился еще и правее какого-то элемента из тройки и был посчитан позже второй раз. Но это также невозможно, потому что если элемент посчитан, будучи в тройке, значет он минимальный в ней, но если он стоит правее какого-либо из элементов данной тройки, то нарушается условие строго возрастания элементов внутри массивов.
Быть не посчитанным ни разу элемент не может,т.к. даже если больше всех элементов во всех трех массивах, то он будет посчитан в самом конце, будучи меньше, чем +infinity.

5 a)

1)1. Данные:
Массивы : a, pred, dist, каждый длиной n. a[i]- элемент последовательности, pred[i]- индекс элемента перед элементом x[i] в самой длинной строго возрастающей подпоследовательности, dist[i]-макситмальная длина строго возрастающей подпоследовательности с концом в x[i]. Вспомогательные переменные: max, imax,i,j.

2. **Псевдокод**:

```
for i=1 to n do
begin
      pred[i]=i
      dist[i]=1
end
for i = 2 to n do
begin
      max = 0
      for j= 1 to i-1 do
            if (x[j] < x[i]) and (dist[j] > max)then
            begin
                  max = dist[j]
                  imax = j
            end
      dist[i] = max+1
      pred[i] = imax
```

```
end
max = 1
for i =1 to n do
        if (dist[i] > max) then max = dist[i]
```

В переменной max хранится максимальная длина строго возрастающей подпоследовательности, которую можно восстановить и сохранить ее в массив rev[n] в обратном порядке:

```
for i = n to 1 do
        if dist[i] == max then
        begin
                imax = i
        end
pos = 1
do begin
rev[pos]= x[imax]
if (imax != pred[imax]) then
begin
        imax = pred[imax]
        pos++
end
while (imax != pred[imax])
```

Реверсом массива rev получим самую длинную строго возрастающую подпоследовательность.

2)Асимптотика:

Число повторов в первом цикле равно : $1+2+3+...+n-1 = \frac{1}{2}n(n-1) = \Theta(n^2)$ т.к. $\forall n \geqslant 2 \hookrightarrow \frac{1}{10}n^2 \leqslant \frac{1}{2}n^2 - \frac{1}{2}n \leqslant n^2$

3)Корректность:

Воспользуемся методом мат. индукции. Если последовательность состоит из 1 элемента, то максимальная длина строго возрастающей подпоследовательности равна 1 и она завершается этим элементом. Предположим мы нашли для каждого элемента последовательности из k элементов максимальную длину строго возрастающей подпоследовательности, которая завершается этим элементом. Теперь добавим следующий (k+1) элемент. Для того чтобы найти максимальную длину строго возрастающей подпоследовательности, которая завершается этим элементом, необходимо найти элемент, расположенный левее и меньше этого, для которого значение длины необходимой подпоследовательности максимально, т.к. только в этом случае у нас получится строго возрастающая подпоследовательность с концом в (k+1) элементе.

б)

1. Данные:
k- максимальная длина возрастающей подпоследовательности ,
массив u [1],..., u [к] , где u[i] — минимальный из последних членов возрастающих подпоследовательностей длины i.

Псевдокод:

```
nl := 1; k := 1; u[l] := x[l] ;
while n1 != n do
begin
        nl : = nl + 1;
        i:=0; j:=k+l;
        while (j - i) != 1 do
        begin
                s := i + (j-i) div 2;
                if x[n1] <= u[s] then
                begin
                        j := s;
                end else
                        begin
                                i := s;
                        end;
        end;
        if i = k then
        begin
                k := k + 1;
                u[k+l] := x[nl];
        end else
        begin
                u[i+l] := x[nl];
        end;
end;
```

2.Корректность:
Для того, чтобы найти максимальную строго возрастающую подпоследовательность необходимо хранить информацию о боллее коротких подпоследовательностях, которые мы могли бы достроить. Для этого надо хранить минимальные последние элементы этих подпоследовательностей (минимальные чтобы они получились как можно длиннее). При этом мы не знаем насколько большим или маленьким будет очередной элемент и для этого храним "информацию min последний элемент подпоследова-

тельностей всех длин от 1 до k (k - макс длина искомой подпоследовательности), чтобы знать куда разместить этот новый элемент. Размещаем элемент либо в самом начале либо в самом конце, когда он соответственно меньше всех элементов ранее занесенных в "концы либо больше их всех. Иначе один из сохраненных концов заменим этим новым элементом, а для этого найдем место из соображений, что соседи нового элемента должны быть максимально близки к нему(по значениям). При этом в массиве u и будет сохранена искомая подпоследовательность. 3. Внешний цикл выполняется n-1 раз. Бинарный поиск, вследствие того, что на каждом шаге число изучаемых элементов уменьшается вдвое, содержит $\log_2 n$ шагов.В итоге получаем временную асимптотику $O(n \log_2 n)$.

6

1) Данные:
times - для числа повторов предполагаемого элемента, now - предполагаемое значение искомого элемента (инициализируется значением dont.know), i - вспомогательная переменная.
. **Псевдокод**

```
times = 0
now = dont.know
for i = 1 to n do
begin
        read(xi)
```

считываем очередной элемент последовательности

```
        if (now == dont.know) then
        begin
                now = xi
                times ++
        end
        else
        begin
                if (xi == now) then
                        times ++
                else
                begin
                        if (times > 1) then
                                times --
```

```
                                    else
                                    begin
                                            times--
                                            now = dont.know
                                    end
                            end
                    end
            end
```

2)Корректность:
Расмотрим два случая - когда число элементов в последовательности четно и когда оно не четно. В первом случае в последовательности из 2n элементов по крайней мере n+1 раз встречается искомый элемент, а остальных элементов не более n-1 штук.

. T.e. если с каждым искомым элементом вычеркнуть из последовательности один не равный ему элемент, то в итоге останется хотя бы два искомых элемента. Аналогично, если элементов изначально 2n+1, то в ней хотя бы n+1 искомый элемент и максимум n не равных ему элементов. При аналогичном вычеркивании остается хотя бы один искомый элемент. Если же вычеркивать просто любые два разных элемента (что и делает алгоритм), то тем более в итоге останется большее 1 число искомых элементов.

. Таким образом, в переменной now обязательно будет значение элемента, повторяющего строго больше чем n/2 раз , а это значение единственно по условию, т.е. это искомое значение.

При этом используем O(1) битов памяти.

3)Асимптотика:
Если сравнения и присваивания выполняются за константное время, то ввиду того, что в цикле выполяняется всегда n итераций, временная асимптотика равна $\Theta(n)$.

№5(с семинара)

1. Данные: два массива x и y, их длины n и k соот-но, pos - индекс элемента y-последовательности, i - ан-но для x.

2. **Псевдокод**

```
if (n < k) then
        return(NO)
pos = 1
for i =1 to n do
begin
```

```
            if (pos == k+1) then
                    return(YES)
            if (x[i] == y[pos]) then
                    pos++
    end
    if (pos == k+1) then
            return(YES)
    else
    return(NO)
```

(на return прекращаем выполнение программы с возвратом результата)

2)Корректность:
Чтобы вторая последовательностость была подпоследовательностью первой, необходимо чтобы ее элементы находились в том же порядке в первой, с добавлением каких-то еще элементов. Это достигается, если индекс pos, указывающий на положение элемента второй последовательности, для которого ищем элемент первой последовательности, ему равный, превосходит длину 2-й последовательности на 1, т.е. каждый из элементов 2-й последовательности мы нашли в первой.

3)Асимптотика: Т.к. очевидно, что длина 1-й последовательности должна быть не меньше длины 2-й последовательности, то достаточно пройтись по всем элеметам 1-й последовательности. В итоге, операций в цикле n и асимптотика $\Theta(n)$.