ДАА Семинар 8

Алгоритми върху масиви

① Семейството на Живко / Vito's family - // Steven Skiena - the programming contest training manual

Всичките роднини на Живко живеят в София на улица X.
Той е решил да им отиде на гости и тъй като уе посещава всеки от тях често иска да гостува в този апартамент, който уе минимизира общото разстояние до всички негови роднини
Да се изчисли алгоритъм, който по зададени n адреса - s_i - на всеки роднина, където $s_i \in \mathbb{N}^+ \ \forall i \in \{1,\dots,n\}$ (n-несетно), намира оптимален адрес за Живко и минималното разстояние от него до всичките му роднини.
// Считаме, че разстоянието между два апартамента s_i, s_j е $d_{ij} = |s_i - s_j|$
Оптималният адрес $\overset{x}{\Sigma}$ е такъв, че $\sum_{i=1}^{n} |x - s_i|$ е минимална $(x \in \{s_1,\dots,s_n\})$

Решение ↳ // Пример $[s_1, s_2] = [2,4]$ - opt: 2 със сума 2
$[s_1, s_2, s_3] = [2,4,6]$ - opt: 4 със сума 4

1н.) Brute Force

Намираме общото разстояние до всеки роднина от текущия кандидат за апартамент. Апартаментът с най-малкото такова разстояние е оптимален. Връщаме номера му и самото разстояние

```
BruteForce(A[1..n], A[i]=s_i)
1. optAp ← 0
2. min ← +∞
3. current ← 0
4. for i ← 1 to n
5.     current ← 0
6.     for j ← 1 to n
7.         current ← current + |A[i] - A[j]|
8.     if current < min
9.         min ← current
10.        optAp ← i
11. return (optAp, i)
```

→ $T(n) \leq n^2$
$M(n) \leq 1$

2н) Better

Може да разглеждаме адресите s_i като точки по оста Ox. Търсим такава точка от тях, че сумата от разстоянията от нея до всички останали е минимална.

Търсим $x \in \{s_1,\dots,s_n\}$: $\sum_{i=1}^{n} |x - s_i|$ е минимална (★)

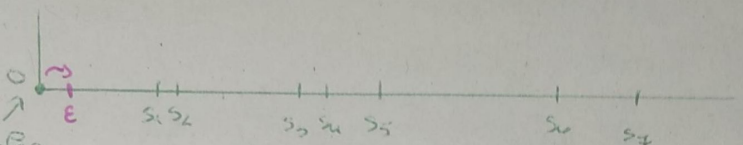
Горната сума (*) е минимална когато X е медианата

↳ Интуиция:

Разполагаме точките $S_1, .., S_n$ по оста Ox в сортиран ред.

Нека за простота $k = 7$. и $S_1 < S_2 < .. < S_7$

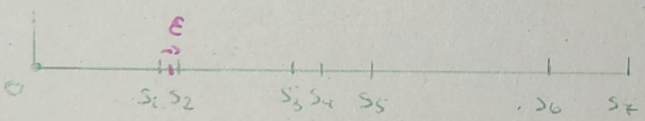
Започваме да се придвижваме от стартова позиция - точката O със стъпка - $\varepsilon > 0$ надясно. При първата стъпка сме отишли с разстояние ε по близо до всяка от $S_1, .., S_n$. » Общото разстояние $\overset{X}{\vee}$ се е намалело със 7ε



стартова позиция ↳ от нея започваме да търсим общо минимално разстояние X до всяка точка S_i

Продължаваме постепенно да се придвижваме надясно със стъпка ε, като след всяка такава стъпка общото разстояние се намалява със 7ε

Това продължава докато не достигнем точката S_1.
Ако сега се придвижим със стъпка ε надясно, разстоянието до точката S_1 се увеличава с ε, а разстоянието до всяка от останалите $S_2, .., S_7$ се намалява с ε. След тази стъпка ε надясно от S_1 X се намалява със 6ε и се увеличава с ε. Общо се намалява с 5ε



Това продължава докато не достигнем S_2. Тоест, след всяка ε стъпка надясно ~~от~~ $-S_1$ преди S_2 намаляване X с 5ε.

Достигаме S_2. Ако се придвижим със стъпка ε надясно, разстоянието до всяка от точките S_1 и S_2 се увеличава с ε, а разстоянието до всяка от точките $S_3, .., S_7$ се намаля с ε. Общо X се намаля с $5\varepsilon - 2\varepsilon = 3\varepsilon$.

...

Достигаме S_3. При всяко ε-придвижване надясно от S_3 разстоянието до всяка от точките S_1, S_2, S_3 се увеличава с ε, а разстоянието до всяка от точките $S_4, .., S_7$ се намалява с ε. Общо X се намалява с $-3\varepsilon + 4\varepsilon = \varepsilon$

...

Достигаме S_4. При всяко ε-придвижване надясно от S_4 разстоянието до всяка от точките S_1, S_2, S_3, S_4 се увеличава с ε, а разстоянието до всяка от точките S_5, S_6, S_7 се намалява с ε. Общо X се намалява с $-4\varepsilon + 3\varepsilon = -\varepsilon$ - Тоест, след всяко ε-придвижване от точката S_4 X се увеличава с ε. Минималната сума до всяка една от точките се постига от т. S_4 - медианата.

Тоест, достатъчно е да намерим медианата, за да решим задачата.

```
Better(A[1..n], A[i]=s_i)
1. median ← 0
2. sum ← 0
3. HeapSort(A)
4 median ← A[⌈n/2⌉]
5. for i←1 to n
6.      sum ← sum + |A[median] - A[i]|
7. return (median, sum)
```

→ $T(n) \asymp n\log n$

$M(n) \asymp 1$

3н.) PICK/SELECT

↳ С линейно време намира (k+1)вия по големина елемент на несортиран масив

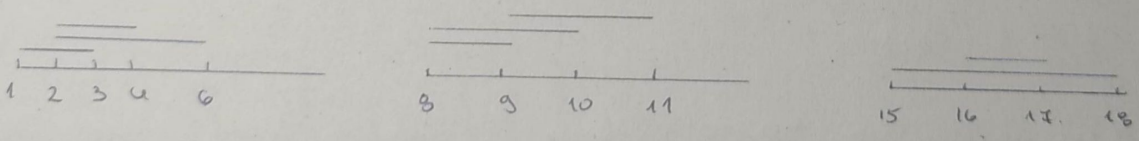
```
Opt(A[1..n], A[i]=s_i)
1. median ← Select(A, ⌊n/2⌋)
2. sum ← 0
3. for i←1 to n
4.      sum ← sum + |A[median]-A[i]|
5. return (median, sum)
```

→ $T(n) \asymp n$

② Даден е масив $A[1..n]$ от интервали (a,b), $a \le b$. Да се върне масив, състоящ се от два по два непресичащи се интервала, получени от сливането на пресичащи се интервали от входния масив A.

// Пример: $A[1..n] = A[(1,3), (2,6), (3,9), (8,10), (9,11), (2,4), (15,18), (16,17)]$

Изход $B := [(1,6), (8,11), (15,18)]$



1н) BruteForce

За всеки един елемент обхождаме масива и го сливаме с тези, които го застъпват. Ако текущият елемент е бил част от предишно сливане го пропускаме.

```
BruteForce(A[1..n], A[i]=(a_i, b_i), a_i≤b_i)
1. B - dynArr
2. for i←1 to n ⟨ curr←A[i]
3.     if B.empty || !isAlreadyUsed(B, A[i])
4.         for j←i+1 to n
5.             if overlap(curr, A[j])
6.                 curr ← (min(curr.first, A[j].first), max(curr.second, A[j].second))
7.         B.push_back(curr)
8 return curr
```

→ $T(n) \asymp n^2$

$M(n) \asymp O(n)$

24) Better

Сортиране A по a. Линейно минаване през така сортирания масив и сливане съседни интервали, ако се налага.

Better($A[1...n]$, $A[i] = (a_i, b_i)$, $a_i \le b_i$)

→ $T(n) \approx n \log n$

```
1. B – dynArr
2. Sort A by first coord.
3.
3. for i ← 1 to n
4.     curr ← A[i]
5.     if B.empty || (!B.empty && B.back.second < curr.first )
6.         B.push-back(curr)
7.     else
8.         B.back.second ← max(B.back.second, curr.second)
9. return B
```

↳ [(1,3), (2,4), (2,6), (8,9), (8,10), (9,11), (15,19), (16,17)]

1ст. B = [(1,3)]

2ст. B = [(1,4)]

3ст. B = [(1,6)]

4ст. B = [(1,6), (8,9)]

5ст. B = [(1,6), (8,10)]

6ст. B = [(1,6), (8,11)]

7ст. B = [(1,6), (8,10), (15,19)]

8ст. B = [(1,6), (8,10), (15,19)]

③ Да се изчисли алгоритъм, който приема график и връща максимален интервал (a,b), $a \le b$, през който няма ангажимент. Графикът е представен като масив от елементи (a,b), $a,b \in \{1,\dots,24\}$, $a \le b$, където a е началният час на ангажимента, а b - крайният. Ангажиментите могат да се припокриват.

Решение:

Използване алгоритъма от предната задача ②, след което линейно минаване и търсим максималното време между два съседни интервала

AlgX($A[1..n]$, $A[i] = (a_i, b_i)$, $1 \le a_i \le b_i \le 24$)

1. B ← mergeIntervals(A)
2. Намери такива два съседни интервала в B,
3. такива че времето между тях е
4. максимално