Малко контролно II

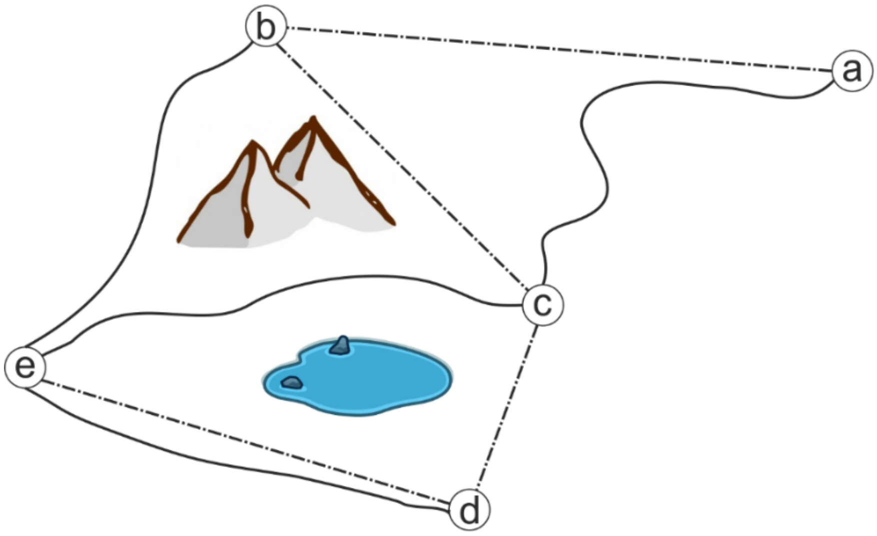
Име: ФН: Курс: Група:

Задача 1: (*6 точки*)

В транспортната мрежа на една страна има два вида транспорт - железопътен и автомобилен. Знае се разстоянието между всеки два съседни града и по двата транспорта.

Предложете алгоритъм, които намира най-късия път между два града s и t с комбиниран транспорт.

Пример:



Най-късият път между a *и* d *е от* a *до* c *по шосе и от* c *до* d *по жп линия.*

Задача 2: (*7 точки*)

Даден е ориентиран граф $G(V, E)$ без цикли. Предложете алгоритъм с време $O(|V| + |E|)$, който намира броя пътища между два върха s и t.

Задача 3: (*7 точки*)

Даден е масив от n положителни цели числа. Предложете алгоритъм с време $O(n)$, който намира най-голямата сума, деляща се на 3 сред сумите от последователни елементи на масива.

Задача 4: (*6 точки*)

За редицата $a_1, \ldots, a_n$ от различни положителни цели числа е дадена операция Δ, която премества елемент от нея в началото ѝ. Например Δ приложена върху $1, 16, 5, 7, 12$ и 5 преобразува редицата в $5, 1, 16, 7, 12$.

Предложете алгоритъм с време $O(n \log n)$, който намира с колко най-малко операции може да се сортира редицата.

Решения:

Задача 1.

Градовете в задачата дефинират граф с два типа ребра - по шосе и по жп линия. Цената на всяко ребро е разстоянието със съответния транспорт.

Изпълняваме алгоритъма на Дийкстра от връх s и търсим $dist[t]$, като за всеки връх проверяваме съседите и по шосето и по жп линията.

Освен това, ако за някои връх намалим $dist$-стойността, то към информацията за предшественика му трябва да добавим и вида на реброто - по шосе или по жп линия.

Задача 2.

Сортираме топологично G и въвеждаме масива $dist$ с $|V|$ елемента, като $dist[u]$ ще казва колко са пътищата от връх u до t.

Обхождаме върховете на графа обратно на топологичната наредба, като:

- за всеки елемент u след t в наредбата $dist[u] = 0$.
- $dist[t] = 1$
- за всеки елемент u преди t в наредбата:

$$dist[u] = \sum_{v \in Adj(u)} dist[v]$$

Тъй като няма цикли, броят пътища от връх u до t е сумата на броя пътищата от всеки съсед на u до t.

Сложността на този алгоритъм е $O(|V| + |E|)$, а търсеният резултат е $dist[s]$.

Задача 3.

Разглеждаме следния алгоритъм:

```
Alg (int a[n])
{
      int b[3] = [0, -1, -1];
      int m = 0, s = 0;

      for (int i = 0; i < n; i++)
      {
            s += a[i];
            int u = s % 3;
            if (b[u] == -1) b[u] = s;
            else if (s - b[u] > m) m = s - b[u];
      }

      return m;
}
```

Търсим две парциални суми, които дават еднакъв остатък при делене на 3 и чиято разлика е най-голяма.

Ясно е, че ако S_i, S_j и S_k за $i < j < k$ са три парциални суми, даващи един и същи остатък, то подмасивът с най-голяма сума, получен от тях е този между $a[i]$ и $a[k]$.

Затова в $b[u]$ пазим първата парциална сума, която дава остатък u при делене на 3. Ако такава сума все още не е намерена, то $b[u] = -1$.

Останалата част от инвариантата на цикъла е: На всяка стъпка s пази парциалната сума до i, а m пази текущата максимална сума на подмасив.

Задача 4.

Нека $b_1, \dots, b_n$ е сортираната редица.

Разглеждаме b_{n-1}.

Ако той се намира след b_n в началната редица, то на някоя стъпка трябва да го сложим най-отпред и после да прехвърлим всички по-малки от него елементи.

Оптимално е да направим това още в началото - преместваме b_{n-1} и един по един преместваме всички по-малки от него, започвайки от b_{n-2}.

Ако b_{n-1} се намира преди b_n, то няма нужда да го местим и гледаме позицията на b_{n-2} спрямо позицията на b_{n-1}.

Продължаваме по същия начин за останалите елементи.

Алгоритъмът е следния:

Сортираме наредените двойки (a_i, i) за $O(n \log n)$, след което с цикъл от $n-1$ до 1 търсим най-голямото k, за което (b_k, p_k) и (b_{k+1}, p_{k+1}) са такива, че $p_k > p_{k+1}$.

Ако няма такова k, то резултатът е 0, тоест редицата вече е сортирана.

Можем да решим задачата и за линейно време:

```
minDelta (int a[n])
{
     var target = −∞, max = −∞;

     for (int i = 0; i < n; i++)
          if (max < a[i]) max = a[i];
          else if (target < a[i]) target = a[i];

     int steps = 0;
     for (int i = 0; i < n; i++)
          if (a[i] <= target) steps++;

     return steps;
}
```

Търсим най-големия елемент, за който има по-голям преди него в масива.

След това търсим колко елемента са по-малки от него (в случая сравняваме с $\leq$ за да осигурим и стъпката за самия елемент).