Сложност на алгоритмични задачи.

времева   по памет

Прочетете худава глава 7 от учебника на зад. Маркова!

<u>def/ Долна и горна граница на сложност на задача</u>

Ако P е произволна изч. зад., то:

- долна граница за P е всяка ф-я $f(n)$, т.е <u>всеки</u> алгоритъм $T(n)$, който решава P работи $f(n) \preceq T(n)$.
- горна граница за P е всяка ф-я $f(n)$, т.ч. <u>поне един</u> алгоритъм $T(n)$, решаващ P, работи $T(n) \preceq f(n)$

Интересуваме се от възможно най-голямата долна граница и възможно най-ниската горна.

Начини за доказване на долни граници

(1) Елементарни д-ва:

- по размера на входа
  при: за неструктурирани входни данни като несортиран масив (трив. $\Omega(n)$)
- по размера на изхода

(2) Д-ва използващи дървета за вземане на решения

- за всеки SORT: $\Omega(n \lg n)$
- Element uniqueness: $\Omega(n \lg n)$
- Balance Puzzle: 2
- Twelve-coin puzzle: 3

(3) Д-ва използващи редукции

- Сега ще разгледаме

(4) Игра срещу противник

- търсене в сортиран масив:
  - $O(\log n)$ – binary Search
  - $\Omega(\log n)$ – игра с/у противник

  може и с дърво на вземане на решения.

Редукция

Идеята е следната. Имаме известен набор от задачи, чиито долни граници знаем като ЕЦ, SORT и други. Даваме ни се ново неизвестна задача P. Ще се опитаме да редуцираме известна задача през М към P и пишем $M \propto P$. Така ще установим неговото, че знаем долната граница на М.

★ Ако P се решава бързо, то и М се решава бързо!

Т.е. имаме следните изисквания към редукцията:

① Посока на редукцията
- решаваме стара задача през новата

② Коректност
- решава напълно старата задача

③ Бързина
- редукцията трябва да изисква сравнително-малко време от дадената

долна граница на старата задача.

Да приложим тези дефиниции в действие.

Зад.1) Даден е числов масив $A[1..n]$. Търси се $\min\limits_{i \neq j} |A[i]-A[j]|$

а) Измислете алгоритъм с времева сложност $O(n\lg n)$

b) Докажете, че долната граница на тази задача е $\Omega(n\lg n)$.

а) Ако имаме примерно масива 8 100 5 6 9, то можем да го сортираме 5 6 8 9 100 и да търсим $\min\limits_{2 \le i \le n} (A[i]-A[i-1])$.

Псевдокод на другата страница.

```
minDist1D(A[1..n]: array of nums): num
    sort(A) // O(n lg n)
    min ← A[2] − A[1] // O(1)
    for i ← 2 to n−1 do
        current ← A[i+1] − A[i]
        if current < min
            min ← current
    return min // O(1)
```

O(n)

Т.е. $T(n) \asymp n \lg n + n \asymp n \lg n$ и

$M(n) \asymp 1$ (заради броячи и доп. променливи)

Опитайте се да докажете коректност на този алгоритъм.

Инварианта:

$min = \min\{A[2]-A[1], \ldots, A[k+1]-A[k]\}$

за $k \in \{2, \ldots, n-1\}$.

b) Дали не можем ElementUniqueness да решим?

Ако допуснем, че можем да решаваме MinDist1D бързо от EU т.е. за $o(n \lg n)$, то:

```
EU2(A[1..n]: array of nums): bool
    if MinDist1D(A[1..n]) = 0
        return false
    else
        return true
```

① $EU \propto MinDist1D$

② Коректност на редукцията:

няма повторения в $A[1..n] \Leftrightarrow \min\limits_{i \neq j} |A[i] - A[j]| > 0$

Нека MinDist1D връща резултат ans. Ако то е 0, то значи в масива съществуват две числа на различни позиции, които са равни ($ans = \min\limits_{i \neq j} |A[i] - A[j]|$).

Тогава EU2 връща false, което значи, че има поне един елемент на масива, който не е уникален. ✓

Ако ans > 0, то значи не съществуват еднакви елементи в масива и тогава ЕИ2 връща true, което е коректно.

③ Бързина редукцията

Нека е $T_{red}(n)$ отделни работата на ЕИ2 без работата на MinDist1D в нея.

$T_{red}(n) \asymp 1$ (заради if-else, return).

$\underline{1 \prec n \lg n}$

значи редукцията е достатъчно бърза $T_{red}(n) \prec n \lg n$.

Но тогава, ако и MinDist1D се решава с $O(n \lg n)$ времева сложност, то излиза, че $T_{ЕИ2}(n) \in O(n \lg n)$, но винаги се решава за време

$T_{ЕИ2}(n) \in \underline{\underline{\Omega(n \lg n)}}$

Значи и MinDist1D се решава за $\Omega(n \lg n)$.

зад(2) Задача за търсене на мода на даден числов масив A[1..n]. Докажете, че:

a) горна граница е $O(n\lg n)$

b) долна граница е $\Omega(n\lg n)$

a) Трябва да съставим алгоритъм с такава сложност:

```
mode(A[1,2,..,n] : arrayofnumbers) : number
  sort(A[1,..,n])
  mode ← A[1]
  maxCount ← 1
  for i ← 2 to n do:
      if A[i-1] = A[i]
          if A[i-1] = mode
              maxCount ++
          else
              lengthMostRightGroup ++
              if lengthMostRightGroup > maxCount
                  mode ← A[i]
                  maxCount ← lengthMostRightGroup
      else lengthMostRightGroup ← 1
```

Инварианти:

① в mode се съдържа модата на A[1..k-1]

2) maxCount съдържа броя появи на mode в A[1...k-1]

3) Ако A[k-1] не е mode, то lengthMostRightGroup съдържа дължината на най-дясната група в A[1,..,k-1]

за $k \in \{2,3,...,n+1\}$

$$T(n) \asymp \underbrace{n \lg n}_{sort} + \underbrace{n}_{\text{цикъла}} \asymp n \lg n.$$

b) Избираме отново ЕИ.

```
ЕИ2(A[1..n]: array of nums): bool
1. mode ← Mode(A[1..n])
2. count ← 0
3. for i ← 1 to n do
4.       if A[k] = mode
5.             count ← count + 1
6. if count = 1
7.       return true
8. else
9.       return false
```

1. Входа на редукцията:
   $EU \propto Mode$

2. Коректност на редукцията
   - няма повторения $\iff$ $count = 1$

3. Бързина на редукцията
   $T_{red}(n) \asymp n$ (заради for)
   Допуснали сме, че $T_{Mode}(n) \propto n \lg n$.
   $T_{red}(n) \propto n.\lg n$
   достатъчно бърза е
   Излиза, че $T_{EU_2}(n) \propto n \lg n$ 2

Сега MinDist2D и MinDist3D

① Ще направим MinDist1D $\propto$ MinDist2D:

MinDist1D(A[1..n]: array of nums): num

1. B[1..n] : array of tuples <x,y>
2. for i ← 1 to n do
3. B[i].x ← A[i]
4. B[i].y ← 0
5. return MinDist2D(B[1..n])

② Коректност на редукцията:

$$\underset{\substack{\text{изх}}}{\text{от 2D}}\left(\min_{i\neq j}\sqrt{(B[i].x-B[j].x)^2+(\underbrace{B[i].y}_{0}-\underbrace{B[j].y}_{0})^2}=\right.$$

$$=\min_{i\neq j}\sqrt{(B[i].x-B[j].x)^2}=\min_{i\neq j}|B[i].x-B[j].x|=$$

$$\left.=\min_{i\neq j}|A[i]-A[j]|\right)\underset{\text{изх.}}{\text{от 1D}}$$

③ Бързина на редукцията

$$\left\{\begin{array}{l}\left\{\begin{array}{l}T_{red}(n)\asymp n \propto n\lg n\\ T_{MinDist2D}(n)\propto n\lg n\end{array}\right.\\ T_{MinDist1D}(n)\propto n\lg n\end{array}\right.$$

Аналогично е за MinDist3D.

Зад. 3 Нека $A[1...n]$ е масив от цели числа. Искаме да го сортираме и знаем, че допълнително:

a) числата в $A[1..n]$ са положителни

b) за $(\forall i \in \{1,..,n\})[2n \leq A[i] \leq 5n]$

Кой/кои от тези случаи може/могат да се реши/решат за $o(n \lg n)$?

a) Този не може. Ще го покажем с редукция към Sort.

1. Посока на редукцията

$Sort \propto SortPositive$

```
Sort(A[1..n]: array of nums)
  min ← A[1]
  for i ← 2 to n do
      if min > A[i]
          min ← A[i]
  min ← min − 1
  for i ← 1 to n do
      A[i] ← A[i] − min
  SortPositive(A[1..n])
  for i ← 1 to n do
      A[i] ← A[i] + min
```

$O(n)$ за min = min A[1..n]

Прави A[i] > 0 за вс. i $O(n)$

$O(n \lg n)$

$O(n)$ за възст. на ориг. ст-ти.

② <u>Коректност на редукцията</u>

1) Изваждането на (min − 1) прави числата положителни

2) Прибавянето на (min − 1) връща оригиналните ст-ти на числата.

③ Бързина на редукцията

$T_{red}(n) \asymp n + n + n \asymp n \propto n \lg n$

$T_{SortPositive}(n) \propto n \lg n$

Значи редукцията е достатъчно бърза и показва, че в общия случай по сортиране можем да решим задачата за $O(n \lg n)$ с $\Omega(n \lg n)$ долна граница.

b) Да можем с Counting sort в range $2n$ до $5n$.

1 до $3n$.

$T(n) \simeq 3n \simeq n < n \lg n$.

Зад 1) Имаме масив от числа A[1..n] и искаме да покажем, че намирането на броя на различните стойности в масива е $\Theta(n \lg n)$ по време.

По def. значи:

a) измислете алгоритъм основан на сравнение със сложност $O(n \lg n)$ по време.

b) докажете, че долната граница за алгоритми, решаващи задачата, е $\Omega(n \lg n)$.

---

a)
```
alg(A[1..n]: array of nums)
1. sort(A[1..n])  // Θ(n lg n)
2. cnt ← 0
3. for i ← 2 to n do
4.     if A[i-1] ≠ A[i]
5.         cnt ← cnt + 1
6. return cnt
```

Има и друг вариант →

```
alg2(A[1..n]: array of nums)
sort(A[1..n]) // Θ(n lg n)
cnt ← 0
i ← 1
while i <= n
    cnt ← cnt + 1
    j ← i + 1
    while j <= n ∧ A[i] = A[j]
        j ← j + 1
    i ← j
return cnt
```

Сложността и на двата алгоритъма е $\Theta(n \lg n)$.

b) Можем да редуцираме EU до нея.

① Посока на редукцията
EU $\propto$ CntDiffNums

② Коректност на редукцията
Всички елементи са уникални $\Leftrightarrow$
брой различни елементи е $\underline{n}$
размерът на масива

Eu (A[1..n] : array of nums)

1. cnt ← CntDiffNums(A[1..n])
2. if n = cnt
3.   return true
4. else
5.   return false

③ Бързина на редукцията

$T_{red}(n) \asymp 1 \propto n \lg n$

$T_{CntDiffNums}(n) \propto n \lg n$

↑ по допускане

Значи $T_{Eu}(n) \propto n \lg n$

↑ Ч