Решења задатака - Јуниорска српска информатичка олимпијада

26. мај 2024.

Неодговоран студент

Аутор: Огњен Тешић

Груба сила

Решење грубом силом подразумева да се испробају сви могући низови снимака који трају мање од датог броја минута. Сложеност овог решења је $\mathcal{O}(n^3)$, односно $\mathcal{O}(n^2)$ уз пажљиву имплементацију.

Покретни прозор (два показивача)

Идеја решења је коришћење технике покретног прозора (енгл. *слидинг щиндощ*) да би се одредио најдужи узастопни сегмент снимака који укупно трају највише m минута. Прозор је ограничен са два показивача који се померају кроз низ снимака надесно. Почетак прозора се иницијално поставља на први елемент низа, а крај прозора се помера надесно и нови снимци се додају све док укупна дужина снимака не пређе m. Када се то деси, почетак прозора се помера напред док укупна дужина не постане мања или једнака m. Током овог процеса се прати највећи број узастопних снимака који задовољавају услов, и тај број се чува као резултат. Сложеност овог приступа је $\mathcal{O}(n)$.

Постоје и друга решења задатка, на пример бинарном претрагом, које задатак решава у сложености $\mathcal{O}(n \log n)$, што је такође довољно за максималан број поена.

Најбоља замена

Аутори: Огњен Тешић, Душан Попадић и Немања Мајски

Груба сила

У свим изложеним решењима основна идеја је да се за свако растојање R ($R \leq K$) од тачке A, израчуна који је највећи број који може да се у највише R корака (замена суседних елемената) доведе на позицију A. Јасно је да је тај број максимум интервала $[A-R, A+R]$ (тачније, пресек поменутог интервала и интервала $[0, N-1]$, али у наставку анализе то нећемо посебно истицати).

Када се на довођење неког броја на позицију A потроши R корака, онда преостаје $K-R$ корака за довођење неког другог броја на позицију B.

У случају да се интервали $[A-R, A+R]$ и $[B-(K-R), B+(K-R)]$ не секу (немају заједничких елемената), у преосталих $K-R$ корака просто треба највећи елемент интервала $[B-(K-R), B+(K-R)]$ довести на позицију B. У овом случају, највећа вредност збира за изабрано R је

$$S_R = \max_{A-R \leq i \leq A+R} a_i + \max_{B-(K-R) \leq j \leq B+(K-R)} a_j$$

Укупно решење S се добија када S_R израчунамо за свако R и узмемо највеће S_R:

$$S = \max_{0 \leq R \leq K} S_R = \max_{0 \leq R \leq K} \left(\max_{A-R \leq i \leq A+R} a_i + \max_{B-(K-R) \leq j \leq B+(K-R)} a_j \right)$$

Преостаје случај када се интервали $[A-R, A+R]$ и $[B-(K-R), B+(K-R)]$ секу. Тада треба водити рачуна о томе да не може исти елемент низа да буде доведен и на позицију A и на позицију B. Зато у овом случају претходна идеја са максимумима интервала треба некако да се доради, а то се у решењима која следе ради на различите начине.

Прво решење

Да бисмо избегли узимање истог елемента низа као максимума у оба интервала ($[A-R, A+R]$ и $[B-(K-R), B+(K-R)]$), можемо за сваки интервал да израчунамо по два највећа броја. Уколико се за неко R догоди да интервали имају максимум на истој позицији, бираћемо највећи број из првог интервала и други највећи из другог, или обрнуто, према томе који пар даје већи збир. Приметимо да није довољно да се провери да ли су максимуми два интервала једнаки, него треба проверавати да ли су им исте позиције. Због тога ћемо поред две највеће вредности у интервалу да памтимо и позицију (првог) максимума.

Рачунање две највеће вредности и позиције максимума за свако R треба обавити користећи претходно израчунате вредности за мање R, тј. инкрементално. На тај начин, након учитавања низа и броја K, време потребно за решавање сваког задатка биће сразмерно са K.

Друго решење

Узимање истог елемента низа као максимума у оба интервала $[A-R, A+R]$ и $[B-(K-R), B+(K-R)]$ можемо да избегнемо и другачије. Приметимо прво да ништа нећемо пропустити ако не разматрамо ситуације у којима два броја које доводимо на позиције A и B међусобно размењују места. Заиста, тиме се само троши једна размена, а ништа се не добија. Одавде следи да када се $a[A]$ размењује са неким $a[i]$, онда је довољно да се разматрају размене $a[B]$ само са оним вредностима $a[j]$ за које је $i < j$.

Разрадимо још мало ову идеју. Нека је i неки индекс из интервала $[A-K, A+K]$. Тада је за размену $a[i]$ са $a[A]$ потребно $R = |A-i|$ корака. Остаје нам још $K-R$ корака за размену $a[B]$ са неким $a[j]$ за које је $j > i$, а то $a[j]$ треба изабрати као максимум интервала $a[i+1, B+(K-R)]$. Овде је незгодно то што у случају када је $i \geq B$, максимум интервала $a[i+1, B+(K-R)]$ за разне i више није могуће рачунати инкрементално, почевши од позиције B. Зато ћемо описано скраћивање другог интервала са $a[B-(K-R), B+(K-R)]$ на $a[i+1, B+(K-R)]$ примењивати само у случају када је $i < B$. Случајеве када се $a[A]$ размењује са неким $a[i]$ за $i > B$ можемо да обрадимо тако што позиције A и B замене улоге. Другим речима, за $i > B$ подразумевамо да се $a[i]$ размењује са $a[B]$, док се $a[A]$ размењује са неким $a[x]$ за које је $x < i$. То $a[x]$ бирамо као максимум интервала $[A-(K-(i-B)), i-1]$. Овакви максимуми за разне i могу да се израчунају инкрементално, почевши од позиције A.

Случај када се $a[A]$ размењује са $a[B]$, а $a[B]$ се размењује са неким $a[x]$ за $x > B$ је специјалан. У том случају, $a[x]$ одређујемо као максимум интервала $a[i+1, B+(K-R)]$.

Решење у коме се претпоставља да k није веће од 1000

Пошто је k довољно мало, у овом решењу максимуми појединих интервала не морају да се рачунају инкрементално. Тиме се добија једноставнији алгоритам, али је његова сложеност $O(k^2)$ за сваки задатак.

Решење у коме се претпоставља $k < B - A$

Пошто се интервали $[A - R, A + R]$ и $[B - (K - R), B + (K - R)]$ у овом случају не секу, није потребно водити рачуна о заједничком максимуму.

Погађање непознатог броја

Аутор: Немања Мајски

Аутори решења: Немања Мајски, Филип Марић

Иако је из примера могуће помислити да је оптимална стратегија постављања питања увек бинарна претрага, тј. половљење интервала, због ограниченог броја одговора `не`, то није случај. Задатак се, дакле, не може решити грамзивим приступом који би био заснован на неком облику симулације бинарне претраге приликом постављања питања.

Покушајмо прво да дамо рекурзивну формулацију решења. Обележимо са $f(n, k)$ најмањи број питања које Огњен мора да постави да би погодио број мањи или једнак од n ако сме да добије највише k одговора `не`.

Ако је $n = 1$, тада Огњен зна да је у питању број 1 и без постављања питања (јер је једини позитиван природан број мањи или једнак 1 број 1). Зато је $f(1, k) = 0$, за било коју вредност $k \geq 1$.

Ако је $k = 1$, тада Огњен сме само једном да добије одговор `не`. У тој ситуацији једина стратегија која му је на располагању је линеарна претрага тј. једино може редом да пита да ли је број мањи или једнак $n - 1$, да ли је број мањи или једнак $n - 2$ итд. Први пут када добије одговор `не` на питање да ли је број мањи или једнак x, знаће да је у питању број $x + 1$. Пошто замишљени број може бити највише n, он ће бити погођен тек када се постави питање да ли је број мањи или једнак 1, што значи да је у најгорем случају потребно поставити $n - 1$ питања. Зато је $f(n, 1) = n - 1$, за било коју вредност $n \geq 1$.

У осталим случајевима је потребно размотрити различите могућности за прво питање које Огњен може да постави. У принципу, он може да постави било које питање облика да ли је број мањи или једнак x за вредности x од 1 до $n - 1$.

- Ако на питање да ли је број мањи или једнак x добије одговор `da`, проблем се своди на проблем у коме и даље смемо да имамо `k` нетачних одговора, али

сада знамо да је број мањи или једнак од `x` (уместо од `n`). Дакле, у овом случају је $f(n, k) = 1 + f(x, k)$ (једно питање смо већ поставили).

- Ако на то питање добије одговор `не`, проблем се своди на проблем у коме сада смамо да имамо `k-1` нетачан одговор, али знамо да је у питању број из интервала $[x + 1, n]$. Можемо приметити да је погађање броја из интервала $[x + 1, n]$ са датим бројем допуштених нетачних одговора потпуно еквивалентно погађању броја из интервала $[1, n - x]$ са датим бројем нетачних одговора. Дакле, у овом случају је $f(n, k) = 1 + f(n - x, k - 1)$ (једно питање смо већ поставили).

Пошто Огњен не може да зна да ли ће добити одговор `да` или `не`, а жели да гарантовано погоди број, ако му је прво питање да ли је број мањи или једнак x, морамо узети неповољнију варијанту (онај одговор који доводи до већег броја даљих питања). Са друге стране, Огњен има право да бира x за које ће поставити питање, тако да може одабрати најповљнију вредност x (ону која доводи до најмањег броја питања). Тако долазимо до рекурзивне везе:

$$f(n, k) = \min_{1 \leq x < n} \left\{ 1 + \max\{ f(x, k), f(n - x, k - 1) \} \right\}.$$

Пошто у овом решењу долази до преклапања (понављања идентичних) рекурзивних позива, оно је веома неефикасно (сложеност му је експонецијална) и оно ради само за веома мале вредности n (тек мало више од 15 у датом временском ограничењу).

Претходно решење је неопходно оптимизовати применом неке технике динамичког програмирања. Могуће је, на пример, употребити мемоизацију. Меморијска сложеност овог решења је $O(n^2)$, а временска $O(n^3)$. То је довољно да би се решили проблеми до димензије од око $n = 1000$, али не и више од тога.

Могуће је употребити и динамичко програмирање навише. Меморијска сложеност овог решења остаје $O(N^2)$, а временска $O(N^3)$.

Анализом решења за мале вредности n може се приметити да се оптимални број питања врло брзо стабилизује на некој вредности.

```
k:     1 2 3 4 5 6 7 8 9 10 11 12   ceil(log2(n))
n=1    0                           0
n=2    1 1                         1
n=3    2 2 2                       2
```

```
n=4     3  2  2  2                          2
n=5     4  3  3  3  3                       3
n=6     5  3  3  3  3  3                    3
n=7     6  3  3  3  3  3  3                 3
n=8     7  4  3  3  3  3  3  3              3
n=9     8  4  4  4  4  4  4  4  4           4
n=10    9  4  4  4  4  4  4  4  4   4       4
n=11   10  4  4  4  4  4  4  4  4   4   4    4
n=12   11  5  4  4  4  4  4  4  4   4   4   4   4
...
```

Размислимо зашто је то тако? Када је број допушених одговора `не` довољно велики (већи или једнак $\lceil \log_2 n \rceil$), тада је до оптималног одговора могуће стићи помоћу $\lceil \log_2 n \rceil$ питања, коришћењем алгоритма бинарне претраге (половљења). До тачног одговора није увек могуће доћи у мањем броју питања од тога. Пошто се за те вредности k и n одговор унапред зна, може се уштедети и простор и време тако што се одговарајући део матрице динамичког програмирања уопште не попуњава. На наредној слици су тачкицама обележени елементи које нема потребе израчунавати, јер им се унапред зна вредност.

```
k:      1  2  3  4  5  6  7  8  9  10 11 12   ceil(log2(n))
n=1     .                                     0
n=2     .  .                                  1
n=3     2  .  .                               2
n=4     3  .  .  .                            2
n=5     4  3  .  .  .                         3
n=6     5  3  .  .  .  .                      3
n=7     6  3  .  .  .  .  .                   3
n=8     7  4  .  .  .  .  .  .                3
n=9     8  4  4  .  .  .  .  .  .             4
n=10    9  4  4  .  .  .  .  .  .   .         4
n=11   10  4  4  .  .  .  .  .  .   .   .     4
n=12   11  5  4  .  .  .  .  .  .   .   .   .  4
...
```

Потребно је обратити пажњу на то да некада вредност `dp[x][k]` или `dp[n-x][k-1]` може бити ван попуњеног дела матрице и тада је уместо читања елемента матрице потребно употребити њихове познате вредности ($\lceil \log 2(x) \rceil$ тј. $\lceil \log_2(n-x) \rceil$).

Меморијска сложеност решења са овом оптимизацијом постаје $O(n \log n)$, док временска сложеност постаје $O(n^2 \log n)$.

Имплементација може бити мало једноставнија, а асимптотска сложеност иста, ако попунимо почетни део таблице за све вредности k од један па до $\lceil \log_2 n \rceil$, за унету вредност броја n. На пример, за $n = 12$, довољно је попунити наредни део таблице (вредности обележене са `.` се никада не користе, а вредности обележене са `x` су једнаке $\lceil \log_2 n \rceil = 4$).

```
k:     1 2 3 4 5 6 7 8 9 10 11 12   ceil(log2(n))
n=1    0 0 0                        0
n=2    1 1 1                        1
n=3    2 2 2                        2
n=4    3 2 2 .                      2
n=5    4 3 3 . .                    3
n=6    5 3 3 . . .                  3
n=7    6 3 3 . . . .                3
n=8    7 4 4 . . . . .              3
n=9    8 4 4 . . . . . .            4
n=10   9 4 4 . . . . . .  .         4
n=11  10 4 4 . . . . . .  .  .      4
n=12  11 5 4 x x x x x x  x  x  x   4
```

За свако фиксирано k са порастом величине n број питања расте. То значи да се приликом израчунавања вредности $f(n, k)$ дешава да са порастом вредности x (која одређује прво питање) вредности $f(x, k)$ расту, док вредности $f(n - x, k - 1)$ опадају. У почетку ће вредности $f(n - x, k - 1)$ бити веће, а онда ће у једном тренутку вредност $f(x, k)$ постати већа. Минимум који тражимо (вредност $f(n, k)$) ће бити једнак последњој вредности $f(n - x, k - 1)$ која је већа или једнака $f(x, k)$ тј. првој вредности $f(x, k)$ која је строго већа од $f(n - x, k - 1)$. Можемо да приметимо да се ова вредност може наћи бинарном претрагом, чиме се избегава испробавање свих n различитих вредности x.

Пошто се за свако n бинарна претрага интервала $[1, n - 1]$ врши у $O(\log n)$ корака, временска сложеност овог решења постаје $O(n \log^2 n)$.

На веома сличан начин се ова бинарна претрага може имплементирати и у склопу технике динамичког програмирања навише (у овом случају попуњавамо цео правоугаони појас матрице за свако k од 1 до $\lceil \log_2 n \rceil$).

Ако за фиксирано k посматрамо вредности x које налазимо бинарном претрагом примећујемо да оне расту са порастом броја n. Зато не морамо сваку нову вредност броја x одређивати из почетка, већ претрагу можемо вршити линеарно, кренувши од претходне вредности x (која је иницијално једнака 1). Пошто тражимо најмање x за које је $f(x, k) \geq f(n - x, k - 1)$, у сваком кораку увећавамо x све док важи $f(x, k) < f(n - x, k - 1)$. Током попуњавања целе колоне за фиксирано k, променљива x се увећава редом од 1, па највише до n, па је сложеност попуњавања једне колоне линеарна, а меморијска и временска сложеност целокупног решења су $O(n \log n)$.

Нагласимо и да, као што је то често случај са динамичким програмирањем, алокација матрице у облику вектора који садржи векторе одузима значајно време (јер је матрица велика). Брже решење се може постићи ако се уместо вектора користе обични, статички низови. Пошто је вредност n по условима задатка мања или једнака од 10^6, важи да је $\lceil \log_2 n \rceil$ мање или једнако 20, чиме је одређена и максимална могућа димензија матрице која нам је потребна.

Ово решење је било довољно да би се освојило максималних 100 поена. Наредно решење је за ученике који желе да знају више.

Другачији приступ решењу

Мотивација оваквог приступа је што уочавамо „блокове” у нашој дп матици. Уместо да рачунамо колики је најмањи број питања потребан да би се гарантовано погодио број за дате вредности n и k, можемо наћи начин да дате вредности n и k проверимо да ли је довољно q питања. Коришћењем те провере, можемо за свако $k < \lceil \log_2 n \rceil$ наћи најмањи q бинарном претрагом интервала могућих вредности за q (за веће вредности k већ унапред знамо да је резултат $\lceil \log_2 n \rceil$).

Интервал у ком се налази резултат за свако k сигурно припада интервалу $[1, n]$, међутим, он се веома брзо сужава јер резултујућа вредност опада (већ само са два допуштена нетачна одговора сваки број од 1 до милион можемо погодити тек са нешто више од 1500 питања). Зато нећемо интервал бинарне претраге постављати на $[1, n]$, већ ћемо му десну границу одређивати тако што ћемо кренути од границе $d = 1$, коју ћемо множити са 2 све док не добијемо прву довољну вредност броја питања d^* (вредност такву да је са d^* питања и k нетачних одговора могуће погодити

било који природан број из интервала $[1, n]$). Након тога ћемо бинарном претрагом резултат тражити у интервалу $[d^*/2, d^*]$.

Посебно ћемо обрадити случај $k = 1$ тј. случај $f(n, 1) = n - 1$, јер ту унапред већ знамо резултат (а интервал који би требало претражити је веома широк и не може се ефикасно претражити).

Замислимо све могуће секвенце одговора дужине не веће од q које можемо да добијемо, тако да највише добијемо k одговора `не`. Ради једноставности представићемо их словима, где `н` представља одговор не, а `д` одговор да. На пример, за $q = 4$ и $k = 2$ то су следеће секвенце:

```
дддд
дддн
ддндд
дндд
ндддд
ддднн
дндн
ндднн
днн
ндн
нн
```

Јасно је да свака од ових секвенци мора да одговара тачно једном замиљеном броју, као и да сваком замишљњеном броју из интервала од $[1, n]$ треба да одговара тачно једна од ових секвенци. Дакле, ако постоји n секвенци дужине највише q са највише k одговора `не`, можемо сигурно погодити било окји број из интервала $[1, n]$. Постоји увек начин да питања организујемо тако да секвенциама једнозначно додеимо бројеве од 1 до n. Ако сортирамо секвенце лексикографски, онда ће n-та секвенца одговарати томе да је замишљени број n. На пример, за $q = 4$ и $k = 2$:

```
дддд --- 1
ддддн --- 2
ддндд --- 3
ддднн --- 4
дндд --- 5
дндн --- 6
днн  --- 7
ндддд --- 8
```

```
нддн --- 9
ндн  --- 10
нн   --- 11
```

Прво је питање да ли је број мањи или једнак 7. Ако је одговор `да`, наредно питање је да ли је број мањи или једнак 4, а ако је одговор `не`, онда је наредно питање да ли је наредни број мањи или једнак 10. На сличан начин се могу одредити и остала питања. Дакле, увек можемо успоставити бијекцију између скупа ових секвенци и бројева интервала $[1, n]$.

Дакле, да бисмо одредили највећи број n такав да се било који број из интервала $[1, n]$ може погодити са највише q питања и највише k одговора `ne`, потребно је одредити број оваквих секвенци. Приметимо да ако на њихов крај додамо слова `д` тако да све секвенце буду дужине q, оне представљају све речи од слова `д` и `н` које садрже највише k слова `н`. Да нађемо њихов број можемо користити суму биномних коефицијената. Биномни коефицијент $\binom{N}{K}$ представља број начина да од N различитих елементата изаберемо њих K. У нашем случају то ће бити да од q слова изаберемо k оних који ће бити `н`.

Формула за број секвенци је:

$$\binom{q}{0} + \binom{q}{1} + \binom{q}{2} \ldots + \binom{q}{k} = \sum_{i=0}^{k} \binom{q}{i}$$

Услов за то да ли је q питања довољно када замишљен број може да буде до n и имамо k одговора не је:

$$\sum_{i=0}^{k} \binom{q}{i} \geq n$$

Пошто се неки биномни коефицијенти могу користити више пута, израчунате биномне коефицијенте ћемо памтити у матрици (користићемо мемоизацију). Највећи биномни коефицијенти који се израчунавају су облика $\binom{d^*}{k}$. Пошто биномни коефицијенти веома брзо расту осим за мале вредности q довољно је израчунати тек неколико почетних вредности коефицијанта $\binom{q}{k}$, па можемо претпоставити да је време за израчунавање свих потребних биномних коефицијаната $O(d^*)$. Врши се $O(\log n)$ бинарних претрага, и у свакој се извршава $O(\log d^*)$ корака (што је $O(\log n)$). У сваком кораку се сабирају већ израчунати биномни коефицијенти, што

ће најчешће бити веома брзо (у најгорем случају $O(d^*)$). За исписивање резултата потребно је време $O(n)$. Дакле, сложеност, овог приступа је $O(\log n \log d^* d^* + n)$. На пример, за $n = 10^6$, експериментално се утврђује да је $d^* = 2048$, па је време израчунавања мање од времена исписивања резултата.

Граф од клика

Аутор: Немања Мајски

Решење за $N \leq 15$ и сви скупови су величине 2

Услов да су скупови величине 2 нам значи да сваки скуп представља грану у графу. Овај подзадатак је замишљен да решења сложености $\mathcal{O}(2^N)$ добију поене или неоптималне имплементације ДФС алгоритма.

Решење за $N \leq 100$ и сви скупови су величине 2

У овом подзадатку може двоструком петљом да се прође кроз све парове чворова и за сваки да се пусти ДФС или БФС како би се нашла њихова удаљеност. Да, може и ДФС, но, доказ за то ће бити дат у решењу када су сви скупови величине 2.

Сложеност је $\mathcal{O}(N^3)$.

Решење за $N \leq 2000$ и сви скупови су величине 2

Нама ће БФС (и ДФС) дати удаљеност од једног чвора до свих осталих чворова. Тако да из сваког чвора пустимо БФС/ДФС и саберемо удаљености до чворова са већим индексом.

Сложеност је $\mathcal{O}(N^2)$.

Решење када су сви скупови величине 2

Проблем суме збирова најкраћих путева за све парове чворова за генерални граф није решен довољно брзо да прође овај подзадатак. Али нама није задат генерални граф, пошто постоји услов:

За сваки пар скупова (A, B), постоји неки чвор v такав да сви путеви који почињу у било ком чвору a из скупа A и завршавају у било ком чвору b из скупа B морају проћи кроз v.

Тај услов имплицира да су сви циклуси у којима нема понављања чворова унутар једног скупа. Пошто након што је пут изашао из скупа, он мора опет да прође кроз чвор где је изашао да се врати у скуп.

Како је величина скупова 2, то значи да је наш граф ацикличан. По услову задатка граф је повезан, а пошто је и ацикличан значи да је стабло. Зато може у прва три подзадатка да се ради ДФС.

За свака два чвора постоји један пут који их спаја. Да бисмо израчунали наш резултат, за сваку грану ћемо израчунати колико парова чворова има пут који пролази кроз њу. Када склонимо грану, ми ћемо поделити стабло на две повезане компоненте. Приметимо да парови чији пут пролази кроз њу су они који су у различитим компонентама. Тако да можемо да додамо производ величина компоненти на резултат. За налажење величина тих компоненти можемо да користимо ДФС.

Сложеност је $\mathcal{O}(N)$.

Решење за $N \leq 1000$

У овом подзадатку можемо додати све гране у граф и радити решење из подзадатка 3 са БФС-ом. Такође може да се примени Флојд-Варшалов алгоритам.

Сложеност је $\mathcal{O}(N^3)$.

Решење за $N \leq 5000$

Нама је највећи проблем у задатку што скуп може да има јако пуно грана. Ако бисмо могли да имамо сличну структуру графа са $\mathcal{O}(N)$ грана, могли бисмо применити БФС решење из претходног подзадатка.

Заправо, постоји начин да повежемо све чворове у скупу величике K са K грана. Уместо да повезујемо сваки чвор са сваким, направићемо лажни чвор и повезати све чворове из скупа са њим користећи грану дужине 0.5. Сада су удаљености свих парова оригиналних чворова исте као у оригиналном графу, али имамо $\mathcal{O}(N)$ грана.

Значи да можемо применити БФС решење из претходног подзадатка. Такође, да избегнемо рад са децималним бројевима, уместо да дужине грана буду 0.5, оне ће бити 1, а ми ћемо резултат на крају поделити са 2.

Сложеност је $\mathcal{O}(N^2)$.

Главно решење

Приметимо да када применимо конструкцију из претходног подзадатка на скуп, успећемо да се ослободимо свих циклуса тог скупа. Тако да ће наш граф заправо бити стабло. Онда на њега примењујемо решење из подзадатка 4, али бројимо само праве чворове.

Сложеност је $\mathcal{O}(N)$.