Друштво математичара Србије
Адреса: Кнез Михаилова 35/4,
11000 Београд
сајт: www.dms.rs
е-маил: pom@dms.rs

Решење проблема Б за Фебруар 2012
Пермутација

У поређењу са претходним проблемом месеца, проблем Пермутација се показао као доста лакши. Већина послатих решења су прихваћена из првог покушаја. Проблем се може урадити на више начина. Интересантно је да су сва примљена решења заснована на реконструкцији пермутације почев од последњег елемента.

Први алгоритам који ћемо изнети обрађује низ d почев од првог елемента. Овај приступ је за нијансу компликованији у поређењу са идејом другог алгоритма (више због имплементације). Одлучили смо се за овај редослед због интуитивнијег тока анализе. У другом алгоритму биће презентована и имплементација у сложености $O(n \log n)$, док је квадратна описана за оба алгоритма (квадратна сложеност задовољава ограничења дата у проблему). Ученик Борис Грубић, гимназија "J.J. Змај" из Новог Сада, је једини послао решење у (редакцији најбржој позназнатој) сложености $O(n \log n)$.

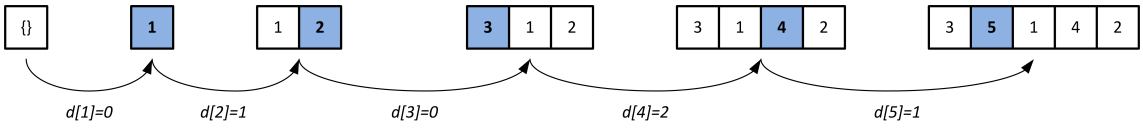
Алгоритам 1: Анализирајмо пример са папира за који је, подсетимо се, $n = 4$ а низ d узима вредности $(0, 1, 0, 1)$. Шта нам говори вредност елемента $d[1]$? Она представља број елемената пре првог елемента, $p[1]$, који су већи од њега. Како је ово први елемент пермутације, испред њега не постоји ни један други елемент, а самим тим ни један већи од њега. Дакле, уколико тражена пермутација постоји, мора да важи $d[1] = 0$. У овом случају услов је задовољен, па прелазимо на наредни елемент.

За други елемент пермутације, $p[2]$, имамо да је $d[2] = 1$. Дакле, елемент $p[1]$, као једини испред другог елемента, је већи од елемента $p[2]$. Да ли је вредност елемента $d[2]$ могла да буде већа од 1? Очигледно не, јер испред постоји само један елемент. Сада можемо извести и генералнији закључак: $d[k] \in [1, k-1]$. Овде се намеће још једно питање: Да ли је ово и довољан услов за постојање пермутације p (управо смо видели да је потребан)? У даљем опису алгоритма видећемо да је одговор на ово питање потврдан.

Када пређемо на трећи елемент, видимо да испред њега нема већих бројева. Другим речима, добијамо да је он већи од $p[1]$ и $p[2]$. Ово можемо записати као: $p[3] > p[1] > p[2]$. На крају, за последњи елемент имамо да је $d[4] = 1$, тј. да је само један елемент са мањим индексом већи од њега. Како знамо поредак за елементе пре њега, управо највећи елемент мора бити

већи од њега. Уколико би то био неки други елемент, тада би $d[4]$ било веће од један. Овим добијамо да је коначни поредак: $p[3] > p[4] > p[1] > p[2]$. На основу овог поредка можемо једноставно реконструисати пермутацију: $p[3] = 4$, $p[4] = 3$, $p[1] = 2$ и $p[2] = 1$.

Изнета дискусија је била везана за конректни пример са папира. Но, врло једноставно је можемо уопштити. Наиме, у сваком кораку одржавамо поредак, строго опадајући, између обрађених елемената премутације. Елементе представљамо индексом. Када смо на позицији са индексом k, елемент $p[k]$, односно индекс k, треба уметнути у овај низ, тако да се испред њега налази тачно $d[k]$ елемената. Другим речима, k постављамо на позицију $d[k] + 1$, а елементе после те позиције транслирамо за једну позицију десно.



Слика 1. Симулације алгоритма на примеру улаза $d = \{0, 1, 5, 2, 4\}$, за који се добија решење $p = \{3, 1, 5, 2, 4\}$.

Након овог дела добијамо низ, означимо га са q, који представља сортиране елементе, преко индекса, пермутације p. Зато добијамо да је елемент на позицији $q[1]$ највећи. Другим речима, $p[q[1]] = n$, $p[q[2]] = n - 1$ итд. Формалније ово можемо записати као

$$p[q[k]] = n + 1 - k$$

Коначин алгоритам можемо описати на следећи начин:

- Уколико за неки елемент важи да је $d[k] \geq k$, одмах вратити $wrong$ и прекинути извршавање алгоритма.
- За свако k почев од 1 до n:
  - Елементе низа q са позиција $d[k] + 1, \ldots, k - 1$ померити за по једну позицију удесно. Односно за свако j почев од $k - 1$ до $d[k] + 1$ (дакле j опада) елемент $q[j + 1]$ поставимо на вредност елемента $q[j]$.
  - Поставити $q[d[k] + 1] = k$.
- Реконструисати пермутацију p преко везе $p[q[k]] = n + 1 - k$.

Сложеност овог алгоритма је квадратна. Заиста, у сваком кораку вршимо транслирање елемената за једно место удесно. У најгорем случају ћемо за сваки корак транслирати цео низ (ово је пример за који је $d[k] = k - 1$). Реконструкција премутације као и почетна провера да ли решење постоји су линеарне сложености.

Напомена: Овај алгоритам се може имплементирати и преко листи. Наиме, описани низ q може бити заправо листа а не низ. Тада нам транслирање није потребно, односно њега вршимо индиректно тако што елемент k убацујемо на позицију $d[k] + 1$ променом показивача. Ово је свакако једноставнији приступ имплементацији, поготову за такмичаре који раде у C++-у. Структура података $vector$ има већ имплементирану функцију $insert$ која

као параметар приме позицију уметања нове вредности (управо оно што је нама потребно). Читаоцу препоручујемо да имплементира и ову верзију алгоритма.

Алгоритам 2: Као што смо напоменули, проблему се може приступити и почев од последњег елемента. Анализирајмо поново пример са папира. За елемент на позицији 4 важи да је $d[4] = 1$. Ово значи да је тачно један елемент већи од елемента $p[4]$, па добијамо да $p[4]$ мора узети вредност 3. Овим смо фиксирали последњи елемент. Сада прелазимо на елемент $p[3]$. Преостале вредности које могу узети елементи су $\{1, 2, 4\}$. Како је $d[3] = 0$ то значи да је $p[3]$ већи од свих елемената пре њега. Другим речиима, од преосталих могућих вредности он узима највећу вредност тј. $p[3] = 4$. Ово разматрање настављамо до првог елемента.

Кроз низ се крећемо почев од последњег елемента. У сваком кораку чувамо скуп могућих вредности, који је на почетку једнак $\{1, 2, \ldots, n\}$. Када обрађујемо елемент на позицији k, он мора узети $(k - d[k])$-ту вредност по величини из скупа могућих вредности. Овим се одражава тражени услов да је он мањи од $d[k]$ елемената пре њега (а њих тек треба обрадити). Довољан и потребан услов за постојање решења, који смо добили и у првом алгоритму, је да у сваком кораку постоји $(k - d[k])$-та вредност у скупу могућих вредности, односно да је $k \geq d[k] + 1$ за свако k.

Псеудо код Алгоритма 2

```
Input: низ d дужине n
Output: тражена пермутација p или "wrong" уколико она не постоји
for k ← 1 to n do
    if d[k] ≥ k then
        return "wrong";
    end
end
possibleValues = {1, 2, ..., n};
for k ← n to 1 do
    c = (k − d[k]) − ta vrednost iz skupa possibleValues;
    p[k] = c;
    izbaci c iz skupa possibleValues;
end
return p;
```

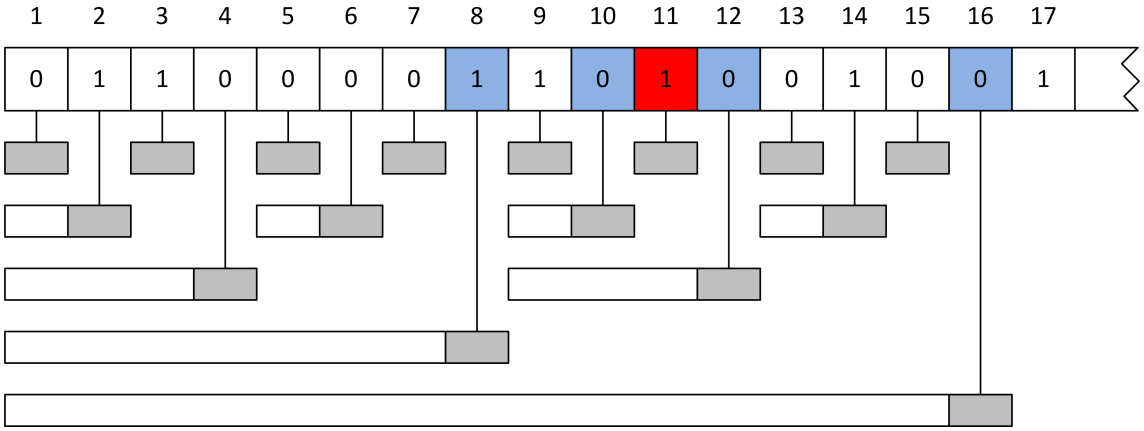
Имплементација приказана у датом псеудо коду има сложеност $O(n^2)$, јер у сваком кораку обилазимо неки део скупа $possibleValues$ (у најгорем случају цео скуп). Овај скуп је најлакше имплементирати преко низа логичких вредности $mark$ дужине n, које ће нам говорити да ли је нека вредност већ убачена или не. Претрага се врши тако што тражимо $(k - d[k])$-ти немаркирани елемент.

Оптимална имплементација: На почетку смо напоменули да се Алгоритам 2 може имплементирати у сложености $O(n \log n)$. У помоћ нам прискаче структура података: **кумулативна табела** (енг. *Binary Indexed Trees*). Овде нећемо изнети објашњење ове структуре. Читаоцу предлажемо да се са њом упозна из других материјала. Препоручујемо материјал који можете пронаћи на сајту *topcoder.com*.

Дефинишимо низ a дужине n. Он ће нам служити за посматрање скупа могућих вредности (слично низу $mark$). Тачније, $a[v] = 1$ уколико је вредност v дозвољена, нула иначе. У k-том кораку потребно је наћи $c = (k - d[k])$-ти дозвољени елемент, односно c-ти елемент у низу a који има вредност 1. Уколико ово посматрамо преко кумулативних сума, добијамо да нам је потребна најмања вредност v за коју је $kSum[v] = c$, где смо са $kSum[i]$ означили суму првих i елемента низа a.

Како је низ $kSum$ сортиран и неопадајући поредак, једна идеја је да се тражена вредност v налази бинарно у низу $kSum$. Међутим, ми имамо само низ кумулативних сума sum. Другим речима, низ $kSum$ је индиректно дат преко низа sum. Подсетимо се да кумулативна табела памти суме одређених поднизова, али нам омогућава да вредност $kSum[i]$ добијемо у логаритамском времену. Сложеност овог приступа је $O(nlog^2n)$ је у сваком кораку бинарне претраге, којих има $\log n$, рачунамо кумулативну суму, која има сложеност $O(\log n)$.

Како можемо ово убрзати? Подсетимо се да је низ кумулативних сума sum дефинисан као: $sum[k]$ је сума елемената са индексима из сегмента $[k - 2^r + 1, k]$, где је r позиција јединице најмање тежине. Тражена вредност v има највише $\log n+1$ битова и њу ћемо реконструисати преко бинарног записа и то почев од цифре највеће тежине. Означимо са $bitPos$ тренутну бит позицију коју испитујемо. На поцехтку она узима вредност $bitPos = \log n + 1$. Бинарну цифру на позицији k у број v означићемо са v_k.



Слика 2. Пример кумулативне табеле и претраге за вредност $c = 5$. Плавом бојом су обележени елементи за које су кумулативне суме су упоређиване. Сиви елементи представљају кумулативну табелу, а "прикачени" поднизови делове низа чију суму садрже.

Уколико је $sum[2^{bitPos}] > v$ тада знамо да је $v_{bitPos} = 0$, јер је $sum[2^{bitPos}]$ сума свих елеманата из сегмента $[1, 2^{bitPos}]$. У случају да је $sum[2^{bitPos}] \leq v$ тада имамо да је ова бит позиција у број v постављена на један. Преостаје да иницијализујемо битове на позицијама $[1, bitPos-1]$. Заправо њиховом иницијализацијом испитујемо кумулативне суме из сегмента $[2^{bitPos}, 2^{bitPos+1} - 1]$. Међутим, лепа чињеница да испитивањем кумулативних сума у низу sum за ове вредности индекса, не "силазимо" испод индекса 2^{bitPos} јер знамо да је тај бит није јединица најмање тежине па је нећемо брисати. Другим речима, поднизови кумулативних сума су управо у овом сегменту. Зато добијамо нови проблем: тражимо (кумулативну) суму $v - sum[2^{bitPos}]$ у делу низа са индексима $[2^{bitPos}, 2^{bitPos+1} - 1]$ кумулативних сума.

Уводимо још једну промењиву $startIndex$ која показује на почетак сегмента који тренутно испитујемо. На почетку она је постављена на 1. Када добијемо да је нека бит позиција

постављена на један, $startIndex$ се повећава за 2^{bitPos}. У исто време, тражену вредност смањујемо за $sum[startIndex + 2^{bitPos}]$. Крајња позиција v је управо $startIndex$.

Псеудо код алгоритма претраге за оптималну имплементацију

Input: бинарни низ a дужине n, низ кумулативних сума sum, тражена вредност c
Output: најмање позиција чија је кумулативна сума једнака v

```
startIndex = 0;
bitPos = log n + 1;
while (startIndex <= n) and (bitPos > 0) do
    currentSum = sum[startIndex + 2^bitPos];
    if currentSum ≤ c then
        startIndex = startIndex + 2^bitPos;
        c = c − currentSum;
    end
    bitPos = bitPos − 1;
end
return startIndex;
```

Сложеност ове претраге је $O(\log n)$ јер у сваком кораку иницијализујемо по једну бинарну цифру, при чему имамо само дикретни позив низ sum а не и стварних кумулативних сума. Укупна сложеност алгоритма је $O(n \log n)$.

Напомена: Задатак се може слично урадити и преко сегментног стабла. Међутим са меморијске стране, решење преко кумулативне табеле је оптималније (захтева $O(n)$ меморије, а сегментно $O(n \log n)$). Такође, алгоритам кумулативне табеле, иако то на први поглед не делује тако, је доста лак за имплементацију. Из ових разлога, уколико обе структуре покривају потребе проблема, препоручујемо да користите кумулативну табелу. Наравно, треба имати на уму да је сегметно стабло доста "јача" структура података.

Решење задатка припремио:
Андреја Илић,
Природно математички факултет, Ниш