Ispravak kontrolnog – stack i trie

Da se podsetimo: Vremenska složenost algoritma

Algoritmi čija je složenost odozgo ograničena polinomijalnim funkcijama smatraju se efikasnim.

$n/f(n)$	$\log n$	$\sqrt{n}$	n	$n \log n$	n^2	n^3
10	0,003 μs	0,003 μs	0,01 μs	0,033 μs	0,1 μs	1 μs
100	0,007 μs	0,010 μs	0,1 μs	0,644 μs	10 μs	1 ms
1,000	0,010 μs	0,032 μs	1,0 μs	9,966 μs	1 ms	1 s
10,000	0,013 μs	0,1 μs	10 μs	130 μs	100 ms	16,7 min
100,000	0,017 μs	0,316 μs	100 μs	1,67 μs	10 s	11,57 dan
1,000,000	0,020 μs	1 μs	1 ms	19,93 μs	16,7 min	31,7 god
10,000,000	0,023 μs	3,16 μs	0,01 s	0,23 s	1,16 dan	3×10^5 god
100,000,000	0,027 μs	10 μs	0,1 s	2,66 s	115,7 dan	
1,000,000,000	0,030 μs	31,62 μs	1 s	29,9 s	31,7 god	

Algoritmi čija je složenost ograničena odozdo eksponencijalnom ili faktorijelskom funkcijom se smatraju neefikasnim.

$n/f(n)$	2^n	$n!$
10	1 μs	3,63 ms
20	1 ms	77,1 god
30	1 s	$8,4 \times 10^{15}$ god
40	18,3 min	
50	13 dan	
100	4×10^{13} god	

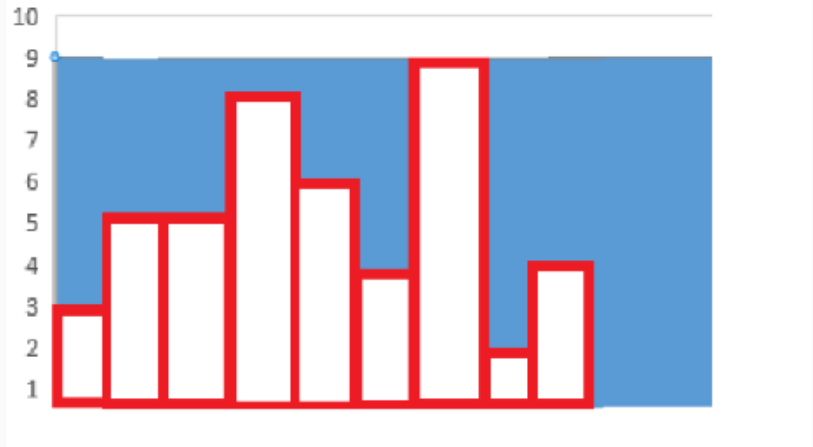
Domaci zadatak tokom raspusta: Histogram (Kvadrat u histogramu, 3D stampac histogram)

1.

Max PravougaonikUpisan U Histogram

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Niz brojeva predstavlja visine stubića u histogramu (svaki stubić je jedinične sirine). Odredi površinu najvećeg pravougaonika u tom histogramu.



Ulaz: Prvi red standardnog ulaza sadrži ceo broj N ($N \leq 1000000$). U sledećem redu je dato N celih brojeva $H_1, H_2, \ldots, H_N$, gde H_I je visina I-tog stubića, $0 < H_I \leq 15000$.

Izlaz: Program treba da ispiše na standardni izlaz nađenu maksimalnu površinu.

ULAZ

9

3 5 5 8 6 4 9 2 4

IZLAZ

24

Za svaki stubić u histogramu potrebno je da odredimo
1. poziciju prvog stubića levo od njega koji je strogo manji od njega (ili poziciju -1 neposredno ispred početka, ako takav stubić ne postoji) i

2. poziciju prvog stubića desno od njega koji je strogo manji od njega (ili poziciju n, neposredno iza kraja niza, ako takav stubić ne postoji).

Ako te pozicije obeležimo sa l i d, onda možemo zaključiti da je najveća površina pravougaonika koji sadrži tekući stubić h[i]$(d-l-1)\cdot h[i]$
(taj pravougaonik je visine $h[i]$, a pod pretpostavkom da je svaki stubić jedinične širine, ukupna širina mu je $d-l-1$).

Pošto su svi stubići od pozicije $l+1$ do $d-1$ visine veće ili jednake od h[i], takav pravougaonik je moguće upisati histogram.
Jasno je i da ne može da postoji pravougaonik koji bi bio viši od ovoga (jer je i-ti stubić visine $h[i]$ i viši pravougaonik bi njega prevazišao) niti da postoji pravougaonik koji bi bio širi od ovoga
(jer čak i da postoje stubići na pozicijama l i d, oni su niži od h[i] i pravougaonik visine h[i] ne bi mogao da se proširi i upiše u njih).

Naivan način da se za svako *i* odrede *l* i *d* je da za svaki stubić *i* iznova puštamo petlje koje nalaze odgovarajuće ivične stubiće.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {

 int n;
 cin >> n;

 vector<int> h(n);

 for (int i = 0; i < n; i++)
  cin >> h[i];

 int max = 0;
 for (int i = 0; i < n; i++) {
  int l = i;
  while (l >= 0 && h[l] >= h[i])
   l--;

  int d = i;
  while (d < n && h[d] >= h[i])
   d++;

  int P = (d - l - 1) * h[i];

  if (P > max)
   max = P;
 }

 cout << max << endl;
 return 0;

}
```

Ovaj algoritam je prilično neefikasan (složenost najgoreg slučaja mu je očigledno $O(n^2)$ i ona nastupa, na primer, kada su svi stubići jednake visine).

Već smo razmatrali probleme određivanja najbližeg manjeg prethodnika i najbližeg manjeg sledbenika za svaki element niza i videli smo da oba problema možemo rešiti u vremenu $O(n)$.
Jedno moguće rešenje je da u jednom prolazu odredimo pozicije najbližih manjih prethodnika svakog elementa, u drugom pozicije najbližih većih sledbenika za svaki element niza i u trećem da na osnovu tih pozicija izračunamo maksimalne površine pravougaonika za svaki stubić.
Međutim, pokazaćemo da se zahvaljujući sličnosti algoritama za određivanje najbližeg manjeg prethodnika i najbližeg manjeg sledbenika sve može uraditi samo u jednom prolazu.

Na stek ćemo postavljati one stubiće za koje još ne znamo poziciju najbližeg manjeg sledbenika (i za koje još nismo odredili površinu maksimalnog pravougaonika koji određuju).
Stubiće obilazimo s leva na desno i za svaki stubić h_d na koji naiđemo sa vrha steka redom skidamo i obrađujemo jedan po jedan stubić h koji je strogo veći od njega. Svakom od tih stubića h stubić h_d je najbliži manji sledbenik.

Ako ispod h na steku postoji neki stubić on je najbliži manji ili jednak prethodnik stubiću h.
Da bismo našli najbližeg strogo manjeg prethodnika potrebno je da nastavljamo da sa steak skidamo sve stubiće čija je visina jednaka h. Stubić h_l koji se nalazi na vrhu steka nakon toga je najbliži strogo manji prethodnik stubiću h. Ako takvog stubića nema, stubić h nema manjih prethodnika i histogram koji mu odgovara se može širiti skroz do levog kraja histograma (tada je $l=-1$).

Površina maksimalnog pravougaonika koji uključuje i stubić h je $h\cdot(d-l-1)$.

Primetimo da se za stubiće jedanke visine stubiću h koji su skinuti sa steka ne računa površina pravougaonika - za tim nema potrebe jer je ona jednaka površini pravougaonika određenog stubićem h.

Recimo i da smo umesto potrage za najmanjim strogo manjim prethodnikom mogli da površine računamo na osnovu najbližih manje-jednakih prethodnika. U tom slučaju ne bismo imali petlju u kojoj se sa steka skidaju svi stubići čija je visina jednaka h. Tada bi se za svaki stubić računala površina, ali ne bi površine pravougaonika svih bile maksimalne. Algoritam bi i u tom slučaju bio korektan, jer bi se maksimalna površina pravougaonika određenog susednim stubićima iste visine korektno izračunala prilikom skidanja poslednjeg od njih sa steka. Kada se stubići sa vrha steak obrade (kada se stek isprazni ili kada se na njegovom vrhu nađe stubić h_l koji je manji ili jednak od h_d), na vrh steka se stavlja
h_d.

Naglasimo i da je po dolasku do kraja niza, potrebno još obraditi sve stubiće koji su ostali na vrhu steka (to će biti stubići za koje ne postoji strogo manji sledbenik). Njihova obrada teče po istom prinicipu kao i ranije, osim što se za vrednost pozicije d uzima n. Zato je u implementaciji poželjno objediniti tu završnu obradu sa prvom fazom algoritma.

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;


int main() {

  int n;

  cin >> n;
```

```
 vector<int> a(n);
 for (int i = 0; i < n; i++)
  cin >> a[i];

 int max = 0;
 stack<int> s;

 for (int d = 0; d < n || !s.empty(); d++) {
  while (!s.empty() && (d == n || a[s.top()] > a[d])) {
   int h = a[s.top()];
   s.pop();
   while (!s.empty() && a[s.top()] == h)
      s.pop();
   int l = s.empty() ? -1 : s.top();
   int P = (d - l - 1) * h;
   if (P > max)
  max = P;
   }
   if (d < n)
    s.push(d);
 }
 cout << max << endl;
 return 0;
}
```

Prikažimo rad algoritma na jednom primeru.
3 5 5 8 6 4 9 2 4

Stek: 0 3
Stek: 0 1 3 5
Stek: 0 1 2 3 5 5
Stek: 0 1 2 3 3 5 5 8
6 na poziciji 4 je najbliži strogo manji sledbenik za 8
Stek: 0 1 2 3 5 5
5 na poziciji 2 je najbliži strogo manji prethodnik za 8
P = 8
Stek: 0 1 2 4 3 5 5 6
4 na poziciji 5 je najbliži strogo manji sledbenik za 6
Stek: 0 1 2 3 5 5
5 na poziciji 2 je najbliži strogo manji prethodnik za 6

```
P = 12
4 na poziciji 5 je najbliži strogo manji sledbenik za 5
Stek: 0 1          3 5
Stek: 0            3
3 na poziciji 0 je najbliži strogo manji pretodnik za 5
P = 20
Stek: 0 5          3 4
Stek: 0 5 6        3 4 9
2 na poziciji 7 je najbliži strogo manji sledbenik za 9
Stek: 0 5          3 4
4 na poziciji 5 je najbliži strogo manji prethodnik za 9
P = 9
2 na poziciji 7 je najbliži strogo manji prethodnik za 4
Stek: 0            3
3 na poziciji 0 je najbliži strogo manji prethodnik za 4
P = 24
2 na poziciji 7 je najbliži strogo manji sledbenik za 3
Stek:
3 nema strogo manjeg prethodnika
P = 21
Stek: 7            2
Stek: 7 8          2 4
4 nema najbližeg strogo manjeg sledbenika
Stek: 7            2
2 na poziciji 7 je najbliži strogo manji prethodnik za 4
P = 4
2 nema najbližeg strogo manjeg sledbenika
Stek:
2 nema najbližeg strogo manjeg prethodnika
P = 18

maxP = 24
```

2.

Прва дама такмичарског програмирања ММ увежбава алгоритме текста. ММ креира речник у ком поставља речи (у лексикографски растућем поретку). Напишите програм који за дату реч утврђује колико речи у речнику је мање (у смислу лексикографског поређења).

У првом реду стандардног улаза дат је природан број N ($1 \leq N \leq 100000$) који представља број команди који Ваш програм обавља над речником. У наредних N редова, налази се нека од команди

INSERT реч

LESS реч

Речи су образоване од малих и великих слова енглеске абецеде, али се сматра да су речи "MagDalina" и "magDaLINa" исте. Можете претпоставити да ниједна реч речника није дужа од 100 карактера, као и да креирани речник неће имати више од 1800000 карактера.

Ваш програм мора (за сваку команду која почиње са LESS) да испише у посебном реду ненегативан цео број који представља број речи лексикографских мањих од речи која следи иза команде LESS. Ако дата реч не постоји у речнику, исписати "NA".

Улаз

14

INSERT potop

INSERT PotoP

INSERT Ognjen

INSERT Rajkov

LESS rajkov

INSERT Daniil

LESS uros

INSERT Milica

INSERT Masa

LESS OgnjeN

INSERT Irina

Излаз

2

NA

0

5

Решење: Све речи које се додају у речник, убацивaћемо у префиксно стабло. Када наиђе

упит less за реч word, прво је потребно испитати да ли се реч налази у речнику. Ово радимо једноставним кретањем од корена, а пратећи карактере речи word. Уколико се реч налази у стаблу, завршавамо у маркираном чвору.

3.

Да се подсетимо: Префиксно стабло (Trie) је структура података која представља све префиксе речи из речника на следећи начин:

1. Свака грана префиксног стабла означена је неким словом енглеске абецеде.

2. Корен префиксног стабла је представљен празним префиксом.

3. Сви други чворови у префиксном стаблу представљају непразан префикс,

тако да сваки чвор је представљен оним префиксом који се добија спајањем слова који су

уписани на гранама које воде од корена до тог чвора (поштујући редослед)

4. Из једног чвора никад неће излазити две гране означене истим словом.

На тај начин се смањује број чворова који су потребни за приказ свих префикса.

![trieslika](https://sr.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D0%B5#/media/File:Trie_example.svg)

Млада програмерка Ирина је направила речник који има N речи које се састоје од малих слова енглеске абецеде и које су смештене у префискно стабло. Написати програм који ће израчунати најмањи број чворова које префиксно стабло може имати након пермутовања слова унутар сваке речи на произвољан начин.

Улаз: У првом реду стандардног улаза налази се природан број N ($1 \leq N \leq 16$). У следећих N редова налази се по једна реч која се састоји од малих слова енглеске абецеде. Укупна дужина свих речи није већа од 999999.

Излаз: У једини ред стандардног излаза испишите један број који представља најмањи број чворова префиксног стабла.

Пример 1

Улаз

```
3
a
ab
abc
```

Излаз
4

Појашњење: Све речи можемо представити преко речи “abц”, те ће префиксно стабло имати 4 чвора (3+1 за корен стабла – празан префикс).

Пример 2
Улаз
3
a
ab
c

Излаз
4

Пример 3
Улаз
4
baab
abab
aabb
bbaa

Излаз

5

Решење: ако градимо префиксно стабло за неки скуп речи сигурно је оптимално започети префиксно стабло тако да сва слова која су заједничка свим речима искористимо као почетни ланац тог подстабла.
Дакле, ако имамо три речи aaab, baab, cab оптимално је искористити слова abc за стварање заједничког префикса свих речи, након тога од речи ће нам остати: aa, ab, c.
Сада смо у стању коналног аутомата када више немамо ниједно слово које се налази у све три речи, потребно је некако наставити градити стабло.

С обзиром да знамо да остали суфикси три ријечи неће моћи бити у истом подстаблу јер немају ниједно слово које деле; бирамо два подскупа речи за које ћемо рећи да настављају да се граде у различитим подстаблима, у нашем случају оптимално је одабрати (aa, ab) у један подскуп и (c) у други. Поделу за неки подскуп радимо на 2^n начина, уколико је у скупу n речи. Свака реч се може налазити или у једном или у другом скупу у који делимо те речи.

Даље, уочимо да иако су неке две речи у истом скупу не значи да ће се градити даље у истом подстаблу барем један корак, већ се одмах може догодити да буду још даље подељене, рекурзивно.

Овакво размишљање нас наводи на решавање задатка динамичким програмирањем, где нам је стање описано скупом речи (битмаском), таквих стања има 2^n. Динамика ће рачунати величину најмањег стабла створеног од тих речи. С обзиром да у сваком стању динамике пролазимо по свим подскупима стања укупна сложеност је $O(3^n)$.