https://arena.petlja.org/competition/prolog2020januar

Zadaci koje ne zelite da radite u Prolog-u

takmičenje se završava: 8. 2. 2020 09:00

PERMUTACIJE

Učesnici Državnog 2019: F, G, H, I

Ostali: A, B, C, D, E, G, H

1. Aleksandar je napravio spisak od N različitih reči koje sadrže samo mala slova engleske abecede. Sve reči su na početku bile uređene leksikografski. Ali, Aleksandar je promenio redosled reči u spisku poštujući permutaciju σ veličine N tako da pozicija i-te reči je jednaka $\sigma(i)$. Napisati program koji za datu permutovanu listu reči otkriva i ispisuje permutaciju σ. Vremenska složenost rešenja: $O(N \log N)$

U prvom redu standardnog ulaza dat je ceo broj N. Svaka od narednih N ($1 \leq N \leq 100000$) linija sadrži jednu reč. Na standardni izlaz ispišite N brojeva koji predstavljaju permutaciju σ.
Smatrajte da ukupna dužina spiska reči je manja od 10^5,

Ulaz	Izlaz
3 zaza ana pera Objašnjenje: Prva reč ana se slika u poziciju 2, druga reč pera se slika u poziciju 3, a treća reč zaza se slika u poziciju 1.	2 3 1
6 cloud	2 1 3 4 6 5

`algorithms`

`complexity`

`development`

`python`

`java`

Rešenje: Za svaku reč ulaza možemo odrediti rang (indeks u leksikografski sortiranom spisku). Najpre sortiramo ulazni spisak algoritmom vremenske složenosti $O(N\log N)$. Potom pretrgaom sortiranog spiska za i-tu reč ulaza nalazimo poziciju j u sortiranom spisku i na izlazu ispisujemo vrednost i na poziciji j.

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

int main() {
   int n;
   cin >> n;

   vector<pair<string, int>> spisak;
   for (int i = 1; i <= n; ++i) {
      string s;
      cin >> s;
      spisak.push_back(make_pair(s, i));
   }

   sort(spisak.begin(), spisak.end());
   for (auto& word : spisak) {
      cout << word.second << " ";
   }
}
```

2. Permutacije

vreme	memorija	ulaz	izlaz
1 s	64 Mb	standardni ulaz	standardni izlaz

Napiši program koji generiše i ispisuje sve permutacije skupa $\{1,2,\ldots,n\}$.
Ulaz Sa standardnog ulaza se učitava broj n ($1 \le n \le 8$).
Izlaz Na standardni izlaz ispisati tražene permutacije. Svaku permutaciju ispisati u posebnom redu, a elemente razdvojiti po jednim razmakom. Redosled permutacija može biti proizvoljan.
Primer

Ulaz
3
Izlaz
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Rešenje
Rekurzivno generisanje permutacija
Rekurzivno generisanje permutacija u leksikografskom redosledu je veoma komplikovano, tako da ćemo se odreći uslova da permutacije moraju biti poređane leksikografski.
U tom slučaju možemo postupiti na sledeći način. Na prvu poziciju u nizu u kojem čuvamo tekuću permutaciju treba da postavljamo jedan po jedan element skupa, a zatim da rekurzivno određujemo sve permutacije preostalih elemenata. Fiksirane elemente i elemente koje treba permutovati možemo čuvati u istom nizu. Neka na pozicijama [0,k) čuvamo elemente koje treba permutovati, a na pozicijama [k,n) čuvamo fiksirane elemente.
Razmatramo poziciju k−1. Ako je k=1, tada postoji samo jedna permutacija jednočlanog niza na poziciji 0, nju pridružujemo fiksiranim elementima (pošto je ona već na mestu 0 nema potrebe ništa dodatno raditi) i ispisujemo permutaciju. Ako nijje k jednako 1,tada je situacija komplikovanija.
Jedan po jedan element dela niza sa pozicija [0,k) treba da dovodimo na mesto k-1 i da rekurzivno pozivamo permutovanje dela niza na pozicijama [0, k-1).
Ideja koja se prirodno javlja je da vršimo razmenu elementa na poziciji k-1 redom sa svim elementima iz intervala [0, k) i da nakon svake razmene vršimo rekurzivne pozive.
Na primer,
Ako je niz na početku 123,onda menjamo element,3 sa elementom 1, dobijamo 321 i pozivamo rekurzivno generisanje permutacija niza 32 sa fiksiranim elementom 1 na kraju.
Zatim u početnom nizu menjamo element na kraju.

Međutim, sa tim pristupom može biti problema. Naime, da bismo bili sigurni da će na poslednju poziciju stizati svi elementi niza, razmene moramo da vršimo u odnosu na *početno* stanje niza. Jedan način je da se pre svakog rekurzivnog poziva pravi kopija niza, ali postoji i efikasnije rešenje. Naime, možemo kao invarijantu funkcije nametnuti da je nakon svakog rekurzivnog poziva raspored elemenata u nizu isti kao pre poziva funkcije. Ujedno to treba da bude i invarijanta petlje u kojoj se vrše razmene. Na ulasku u petlju raspored elemenata u nizu biće isti kao na ulasku u funkciju. Vršimo prvu razmenu, rekurzivno pozivamo funkciju i na osnovu invarijante rekurzivne funkcije znamo da će raspored nakon rekurzivnog poziva biti isti kao pre njega. Da bismo održali invarijantu petlje, potrebno je niz vratiti u početno stanje. Međutim, znamo da je niz promenjen samo jednom razmenom, tako da je dovoljno uraditi istu tu razmenu i niz će biti vraćen u početno stanje. Time je invarijanta petlje očuvana i može se preći na sledeću poziciju. Kada se petlja završi, na osnovu invarijante petlje znaćemo da je niz isti kao na ulazu u funkciju. Na osnovu toga znamo i da

će invarijanta funkcije biti održana i nije potrebno uraditi ništa dodatno nakon petlje.

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

using namespace std;

// ispisuje permutaciju na standarndi izlaz
void obradi(const vector<int>& permutacija) {
  for (int x : permutacija)
    cout << x << " ";
  cout << endl;
}

void obradiSvePermutacije(vector<int>& permutacija, int k) {
  if (k == 1)
    obradi(permutacija);
  else {
    for (int i = 0; i < k; i++) {
      swap(permutacija[i], permutacija[k-1]);
      obradiSvePermutacije(permutacija, k-1);
      swap(permutacija[i], permutacija[k-1]);
    }
  }
}

void obradiSvePermutacije(int n) {
  vector<int> permutacija(n);
  for (int i = 1; i <= n; i++)
    permutacija[i-1] = i;
  obradiSvePermutacije(permutacija, n);
}

int main() {
  int n;
  cin >> n;
  obradiSvePermutacije(n);
  return 0;
}
```

Određivanje sledeće permutacije

Sve permutacije u leksikografskom redosledu se mogu dobiti tako što se krene od početne permutacije 1,2,…,n i u svakom koraku se ispisuje tekuća permutacija i menja se sa narednom permutacijom u leksikografskom poretku.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool sledecaPermutacija(vector<int>& a){
  int n = a.size();

  // linearnom pretragom pronalazimo prvu poziciju i takvu da
  // je a[i] > a[i+1]
  int i = n - 2;
```

```
  while (i >= 0 && a[i] > a[i+1])
    i--;
  // ako takve pozicije nema, permutacija a je leksikografski maksimalna
  if (i < 0) return false;
  // linearnom pretragom pronalazimo prvu poziciju j takvu da
  // je a[j] > a[i]
  int j = n - 1;
  while (a[j] < a[i])
    j--;
  // razmenjujemo elemente na pozicijama i i j
  swap(a[i], a[j]);
  // obrcemo deo niza od pozicije i+1 do kraja
  for (j = n - 1, i++; i < j; i++, j--)
    swap(a[i], a[j]);
  return true;
}

void obradiSvePermutacije(int n) {
  vector<int> permutacija(n);
  for (int i = 0; i < n; i++)
    permutacija[i] = i + 1;
  do {
    obradi(permutacija);
  } while (sledecaPermutacija(permutacija));
}

int main() {
  int n;
  cin >> n;
  obradiSvePermutacije(n);
}
```

Bibliotečka funkcija za sledeću permutaciju

U jeziku C++ funkcija next_permutation deklarisana u zaglavlju određuje narednu permutaciju u odnosu na datu. Funkciji se prosleđuju dva iteratora koji ograničavaju raspon elemenata u kojima se nalazi permutacija.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> permutacija(n);
  iota(begin(permutacija), end(permutacija), 1);
  do {
    for (int x : permutacija)
      cout << x << " ";
    cout << endl;
  } while (next_permutation(begin(permutacija), end(permutacija)));
}
```

Funkcija iota

```
#include <algorithm>
#include <iostream>
#include <list>
#include <numeric>
#include <random>
#include <vector>

int main()
{
    std::list<int> l(10);
    std::iota(l.begin(), l.end(), -4);

    std::vector<std::list<int>::iterator> v(l.size());
    std::iota(v.begin(), v.end(), l.begin());

    std::shuffle(v.begin(), v.end(), std::mt19937{std::random_device{}()});

    std::cout << "Contents of the list: ";
    for(auto n: l) std::cout << n << ' ';
    std::cout << '\n';

    std::cout << "Contents of the list, shuffled: ";
    for(auto i: v) std::cout << *i << ' ';
    std::cout << '\n';
}
```

Possible output:

```
Contents of the list: -4 -3 -2 -1 0 1 2 3 4 5
Contents of the list, shuffled: 0 -1 3 4 -4 1 -2 -3 2 5
```

Heap-ov algoritam

Ovaj algoritam generiše sve moguće permutacije niza od n članova. Algoritam je kreirao B. R. Heap, 1963. godine. Strategija je da se minimizuje pomeranje elemenata u odnosu na naivni algoritam koji smo prvi izložili i koji ima dosta razmena. Osnovna ideja je da se generiše svaka permutacija na osnovu prethodne preko razmene jednog para elemenata; preostalih $n-2$ elemenata se ne premeštaju.

Pri naivnoj implementaciji, morali smo da obavimo dve trampe kako bismo pokupili naredni element koji uklanjamo.
Alo šta ako osmislimo mudriji način da beležimo koje elemente smo već uklonili? U tom slučaju, možemo samo trampiti neuklanjane elemente i nemamo potrebu da obavimo drugu razemenu po redu kako bismo rekonstruisali poredak elemenata!
Ova akcija je ključni korak Heap-ovog algoritma tj. Razvijen je metod za izbor elementa koji učestvuje tramp i taj izbor zavisi od parnosti broja elemenata kako bismo obezbedili različitost izbora. Dakle, izbor se vrši ovako:

- ako je ukupan broj elemenata neparan, onda se uvek bira prvi element
- ako je ukupan broj elemenata paran, onda se pri izboru gledaju uzastopni elementi

Nizovi permutacija za *n* elemenata koji su generisani Heap-ovim algoritmom su početak niza permutacija za *n+1* elemenata. Inače, postoji jedan beskonačni niz permutacija generisanih Heap-ovim algoritmom (niz A280318 na online enciklopediji OEIS).

Java implementacija

```
public void heaps_algorithm(int[] a, int n) {
    if(n == 1) {
        // (nova permutacija)
        System.out.println(Arrays.toString(a));
        return;
    }
    for(int i = 0;i > (n - 1);i++) {
        heaps_algorithm(a, n-1);
        // razmena u zavisnosti
        // parnosti elemenata
        if(n % 2 == 0) swap(a, n-1, i);
        else swap(a, n-1, 0);
    }
    heaps_algorithm(a, n-1);
}
```

Za dalje istraživanje:
Steinhaus–Johnson–Trotter algorithm

Fisher–Yates shuffle

3. **Ilija organizuje šahovski turnir. Na turniru će učestvovati ukupno N ekipa tako da svaka ekipa se sastoji od neparnog broja članova K. Za svakog igrača na turniru poznata je njegova snaga predstavljena prirodnim brojem pri čemu veći broj označava veću snagu, tj. jačeg igrača. Dva igrača na turniru ne smeju imati istu snagu. Ukupno će se održati M mečeva, od kojih u svakom učestvuju dve ekipe. Meč se sastoji od K partija: najjači igrač iz prve ekipe igra protiv najjačeg igrača iz druge ekipe, drugi najjači igrač iz prve ekipe igra protiv drugog najjačeg igrača iz druge ekipe, itd. U susretu dvaju igrača pobeđuje onaj s većom snagom. Od ukupno K odigranih partija, ekipa koja je ostvarila više pobeda smatra se pobednikom meča. Ilija je zamislio ishod jednog mogućeg turnira. Tačnije, odredio je broj ekipa N, kao i M uređenih parova (A, B) koji označavaju da se održao meč između ekipa A i B, te je ekipa A pobedila. No, Ilija nije uspeo pridružiti snage igračima na način da zadati susreti dobiju tražene ishode. Zato traži vašu pomoć: odredite neparan broj igrača K manji od 200, kao i N · K jedinstvenih prirodnih brojeva između 1 i 10^9 koji predstavljaju snage igrača na turniru. Između svakog para ekipa održaće se najviše jedan meč. Garantuje se je da, uz date uslove, uvek postoji rešenje koje zadovoljava tražena ograničenja.**

ULAZ

U prvom redu standardnog ulaza dati su prirodni brojevi N (1 ≤ N ≤ 100), broj ekipa, i M (1 ≤ M ≤ N(N-1)/2), broj mečeva.
U narednih M redova su data po dva različita prirodna broja Ai i Bi (1 ≤ Ai, Bi ≤ N) sa značenjem da ekipa Ai pobeđuje ekipu Bi. Parovi se ne ponavljaju, tj. između dve ekipa igra se najviše jedan meč.

IZLAZ

U prvi red standardnog izlaza ispišite K, broj igrača u svakoj ekipi. K mora biti neparan prirodan broj manji od 200.
U sledećih N redova ispišite po K prirodnih brojeva između 1 i 10^9, odvojenih razmakom, koje označavaju snage igrača u timu koji opisuje taj red. Timovi moraju biti ispisani redom od tima 1 do tima N, dok igrači unutar pojedinog tima mogu biti zadati u bilo kom poretku. Svih N · K brojeva moraju biti međusobno različiti.

Primer 1
ulaz
3 3
1 2
2 3
3 1
izlaz

3

30 20 10

25 16 15

24 22 12

Primer 2

ulaz

3 3

3 2

3 1

2 1

izlaz

3

7 4 1

8 5 2

9 6 3
Hint:

Zadatak je ekvivalentan formulaciji: potrebno je odabrati neparan broj K i K permutacija brojeva od 1 do N tako da za M zadanih uređenih parova (A, B) vazi da se A pojavljuje nakon B u više od pola permutacija. Pritom k-ta permutacija predstavlja relativan poredak snaga k-tog najjačeg igrača iz timova.

Za dati niz permutacija, neka f(A, B) označava razliku broja permutacija u kojima se A pojavljuje pre B i broja permutacija u kojima se B pojavljuje pre A.

Ideja je sledeća:
želimo pronaći mali broj permutacija kojima ćemo promeniti vrednost f(A, B) za neke parove (A1, B1), (A2, B2), ... (As, Bs) na željeni način, a da se pritom ne promeni vrednost funkcije f za ostale parove.

Koliko smo uspešni da s malim brojem permutacija “rešimo” velik broj parova? Kakav god pristup primenimo, uočimo da algoritmi mogu napraviti 200 permutacija, tj. 200 igrača, ali da će f(A, B) biti barem 2 za tražene parove. Tada ćemo izbacivanjem proizvoljne permutacije dobiti traženo rešenje.

4. **Organizatori jednog takmičenja odlučili su sve učesnike počastiti prigodnim nagradama. Na takmičenju je učestvovalo N učenika označenih brojevima od 1 do N redom kojim su se pojavili na konačnoj rang listi (učenik 1 osvojio je prvo mesto, učenik 2 osvojio je drugo mesto, itd.), ali redosled dodele nagrada nije nužno odgovarao tom poretku. Svečana ceremonija već je bila u toku kad su organizatori usred podele nagrada primetili nekolicinu učenika kako se svađaju. Naime, organizatori nisu vodili računa o vrednostima nagrada, te su neki visoko rangirani učenici dobili manje vredne nagrade od nekih niže rangiranih učenika. U stvari, utvrđeno je da će se dvoje učenika posvađati ako više rangirani učenik dobije nagradu strogo manje vrednosti od niže rangiranog učenika. Kako bi ispravili pogrešku, organizatori su preostalim učenicima (onima koji još nisu dobili nagrade) odlučili dodeliti nagrade tako da ukupan broj parova posvađanih učenika nakon dodele svih nagrada bude minimalan. Odredite vrednosti nagrada koje će dobiti preostali učenici.**

ULAZ: U prvom redu standardnog ulaza dat je prirodan broj N (1 ≤ N ≤ 100 000), broj učenika. U sledećem redu je dato N celih brojeva. i-ti broj Xi predstavlja vrednost nagrade koju je dobio i-ti učenik na rang listi do trenutka kada je uočen problem. Ako učenik do tog trenutka još nije dobio nagradu, biće Xi = -1. Inače, važi 1 ≤ Xi ≤ 100 000.

IZLAZ: U jedini red standardnog izlaza ispišite niz X nakon što se svim učenicima dodele nagrade, čije vrednosti moraju biti prirodni brojevi između 1 i 100 000. Ako postoji više mogućih rešenja, ispišite bilo koje od njih.

PRIMER 1
ulaz

5
2 5 -1 2 5
izlaz
2 5 2 2 5
Pojašnjenje: U ovom rešenju se nalaze četiri para posvađanih učenika, i to s rednim brojevima 1 i 2, 1 i 5, 3 i 5, 4 i 5. Alternativno, učeniku br. 3 mogli smo dodeliti i nagradu vrednosti 5. Tada bi opet bila četiri para posvađanih učenika. U drugim slučajevima bilo bi ih više.

PRIMER 2
ulaz
5
4 6 -1 1 -1
izlaz
4 6 2 1 1

PRIMER 3
ulaz
3
-1 -1 -1
izlaz
3 2 1

Rešenje:

Uočimo da su u optimalnom rešenju vrednosti nagrada koje je potrebno odrediti ***neopadajuće***, tj. sve preostale nagrade možemo odrediti tako da se preostali učenici neće međusobno svađati.

Zadatak možemo rešiti tako da svakog učenika kom je potrebno dodeliti neku vrednost razrešavamo nezavisno od drugih takmičara koji imaju isti problem omalovažavanja.

Za učenika na poziciji x i neku vrednost nagrade v možemo unapred izračunati (izgradnjom početnog niza):
• f(x, v) - broj učenika nakon x koji su dobili nagradu strogo veće vrednosti od v
• g(x, v) - broj učenika pre x koji su dobili nagradu strogo manje vrednosti od v.
Za svakog od preostalih učenika x sada možemo pronaći optimalnu vrednost nagrade v tako da minimizujemo izraz f(x, v) + g(x, v).