STL strikes again

Dodatni resursi
http://www.sgi.com/tech/stl/
http://www.cplusplus.com/reference/stl/

VEKTOR

Vektor je osnovni kontejner iz STL-a i zamenjuje obican niz. Kad mu je potreban dodatni prostor u memoriji, vektor se sam povecava, tako sto se ceo vektor prekopira na dvostruko veci prostor. Buduci da se vektor svaki put povecava za dvostruko, ukupno se povecava log n puta. Time je svako novo kopiranje veće od svih ranije kreiranih memorijskih polja vektora, tako da je ukupno vreme potrebno za sva prethodna kopiranja jednako veličini vektora.
Dakle, ne moramo se uopšte brinuti o veličini vektora, jer se STL sam o tome brine.

Možemo reći da se ubacivanje na kraj vektora odvija konstantno (O(1))

Pristupanje svakom elementu vektora je konstantno, kao i izmena nekog elementa vektora.

Vremenska složenost funkcije size() je O(1), jer ***vector*** pamti svoju trenutnu veličinu.
http://www.cplusplus.com/reference/vector/vector/size/

Ipak, ubacivanje na početak ili sredinu vektora je linearno tj. O(n).

Vektor je vrlo jednostavan kontejner (klasa koja služi za čuvanje podataka). Nalazi se u biblioteci (zaglavlju) **vector**.

Primer 01
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{   vector <int> v;
    v.push_back(3); v.push_back(5);
    v.push_back(7);
    for(int i=10;i<14;i++)
      v.push_back(i);
    v[0]--;
    v[5]=-200;
    for(unsigned i=0;i<v.size();i++)
     cout << v[i] << ' ';
    cout << endl;
    return 0;
}
```
IZLAZ
```
 2 5 7 10 11 -200 13
```

U ovom zadatku smo koristili metod push_back() koji dodaje argument metoda na kraj vektora.
U 11. liniji programskog koda, pristupamo vektoru preko preopterećenog (overload) operatora [], kao da radimo sa nizovima (v[0]--;).
U 15. liniji programskog koda, koristimo metod size() koji vraća veličinu vektora.

Primer 02: vektor može čuvati bilo koji tip podataka

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

int main()
{   vector <string> vs;
    vs.push_back("ja"); vs.push_back("volim");
    vs.push_back("programiranje");

    for(unsigned i=0;i<vs.size();i++)
     cout << vs[i] << ' ';
    cout << endl;

    vector < vector<int> > matrica;
    for(int i=0;i<5;i++)
    {
        vector <int> v;
        matrica.push_back(v);
        for (int j=0;j<=i;j++)
        matrica[i].push_back(j);
    }

    for(unsigned i=0;i<matrica.size();i++)
    {
        for(unsigned j=0;j<matrica[i].size();j++)
         cout << matrica[i][j] << ' ';
        cout << endl;
    }


    return 0;
}
```

U 7.liniji programskog koda
```
   vector <string> vs;
```
kreiran je prazan vektor vs koji u sebi čuva stringove.

U 8. i 9. liniji smo u vektor ubacili 3 stringa koristeći metod push_back. Stringove smo ubacili na kraj vektora i time povećali veličinu vektora.

U 15. liniji definisali smo vektor koji u sebi sadrži vektor celih brojeva.
```
    vector < vector<int> > matrica;
```
Na taj način smo kreirali strukturu sličnu dvodimenzionom nizu celih brojeva.
Pri deklaraciji vektora matrica, morali smo voditi računa da ne izostavimo blanko karakter između karaktera > >, jer bi deklaracija
```
 vector < vector<int>> matrica;
```
izazvala sintaksnu grešku.

U 27. liniji ispisujemo vrednost vektora pristupajući im pomoću operatora []. Moramo voditi računa da operator [] ne koristimo za mesta u vektoru koja nisu zauzeta (zauzeta su ona koja smo ubacili pomoću metoda push_back!!!).

Ako bismo iz našeg programskog koda izbacili 18. i 19. liniju programskog koda
```
 vector <int> v;
 matrica.push_back(v);
```

programski kod bi se uspešno kompajlirao, ali bi se verovatno srušio prilikom izvršavanja. Na takve greške moramo sami pazitii često ih je teško uočiti.

IZLAZ
ja volim programiranje
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4

Primer 03: vektor i iteratori

U STL-u postoje iteratori kako bi se pomoću njih moglo iterirati kroz neke strukture podataka u kojima se ne može definisati koji je element po redu, nego samo koji je element pre ili nakon nekog elementa. U vektorima se može direktno pristupiti nekom elementu što nije slučaj sa svim strukturama podataka koje ćemo kasnije obraditi.

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

int main()
{   vector <int> v;
   for(unsigned i=0;i<15;i++)
    v.push_back(i);
   vector<int>::iterator it;
   for(it=v.begin(); it!=v.end();it++)
    cout << *it << ' ';
   cout << endl;
   return 0;
}
```

IZLAZ
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

U 10. liniji programskog koda kroz vektor v umesto brojačem, iteriramo iteratorom it. Deklaracija iteratora it je vector<int>::iterator it;
gde iterartor je naziv tipa, a koristimo notaciju :: da odredimo kojoj klasi pripada.
U 10. liniji inicijalizujemo iterator na početak vektora, tj. dodeljujuemo it=v.begin();

Metod begin() vraća iterator na 1. element vektora, a metod end() vraća iterator na element iza poslednjeg.
Pomoću operatora ++ menjamo iterator tako da pokazuje na sledeći element.
Pomoću operatora != gledamo da li je it različit od nekog iteratora.

U 11. liniji programskog koda ispisujemo elemente vektora pomoću iteratora koristeći preopterećen operator * kako bismo pristupili elementu na koji iterator pokazuje.

Primer 04: još neki metodi klase vector

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

int main()
{   vector <int> v(10); //v.size() je 10
```

```
    v[0]=-1; v[9]=99;
    v[4]=v[5]=100;
    v.push_back(2345);
    for(unsigned i=0;i<v.size();i++)
    cout << v[i] << ' ';
    v.pop_back();
    cout << v.front() << ' ' << v.back() << endl;
    v.clear(); //sve brise i size postaju 0
   if (v.empty()) cout << "v je prazan\n";
    return 0;
}
```

U 6. liniji programskog koda stvaramo vector<int> koji ima 10 elemenata postavljenih na 0. Time v.size() vraca 10, a v.push_back(nesto); ubacuje na lokaciji v[10].
U 12. liniji metoda pop_back izbacuje poslednji element (tj. v[v.size()-1]) iz vektora u vremenskoj slozenosti O(1) i smanjuje v.size() za 1.
U 13. liniji programskog koda v.front() vraca prvi element vektora, tj. v[0], dok v.back() vraca poslednje element vektora tj. v[v.size()-1].
U 14.liniji pozivamo metod v.clear(); da bi se obrisali svi elementi vektora v i onda ce v.size() postati 0.
U 15. liniji pozivamo metod v.empty() koji vrati true ako je v.size()==0

Primer 05: još neki metodi klase vector

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

inline void ispisi(const vector<int>& x)
{  for(int i=0;i<x.size();i++) cout << x[i] << ' ';
   cout << endl;
}

int main()
{    vector <int> v,q;
    v.insert(v.end(),7,1);
    v.insert(v.begin()+2,3);
    ispisi(v);
    q.insert(q.begin(),10,8);
    q.insert(q.end(),9);
    v.insert(v.begin()+4, q.end()-3, q.end());
    ispisi(v);ispisi(q);
    v.erase(v.begin()+2);
    v.erase(v.begin()+3,v.begin()+6);
    ispisi(v);
    vector <int> ve(3,4);
    ispisi(ve);
    v.swap(ve);
    ispisi(v);ispisi(ve);
    return 0;
}
```

IZLAZ
```
1 1 3 1 1 1 1 1
1 1 3 1 8 8 9 1 1 1 1
8 8 8 8 8 8 8 8 8 9
```

1 1 1 1 1 1 1
4 4 4
4 4 4
1 1 1 1 1 1 1

Napomene:
U 5. liniji programskog koda implementirana je funkcija koja ispisuje sadrzaj vektora. Kako bismo ubrzali rad funkcije, opredelili smo se da kao argument funkcije primami referencu na vektor i zbog toga ga ne kopiramo. Kako bismo se zastitili od slucajne izmene vektora, argument je konstantna referenca.

U 12. liniji programskog koda
v.insert(v.end(),7,1);
dodajemo sedam 1ca ispred iteratora v.end(), gde v.end() pokazuje ne element iza poslednjeg.

U 13.liniji programskog koda dodajemo broj 3
v.insert(v.begin()+2,3);
dodajemo broj 3 ispred iteratora v.begin()+2. To je iterator na 3. element vektora. Metod insert i erase imaju slozenost O(n).

Iteratori nisu jednaki za sve kontejnere iz STL-a, tako da iterator na vektor moze imati vecu funkcionalnost nego iterator na neki drugi kontejner.
Iterator na vektor moze se opteretiti + ili – ili pomeriti nekoliko elemenata unapred ili unazad (jer vektor interno podatke cuva u obliku niza), dok neki drugi iteratori to nece moci.

U 15. liniji
q.insert(q.begin(),10,8);
dodajemo 10 puta broj 8 na pocetak vektora q.

16. linija
q.insert(q.end(),9);
moze da se napise i kao
q.push_back(9)

17. linija
v.insert(v.begin()+4, q.end()-3, q.end());
sadrzi metod insert koji prima tri iteratora.
insert(it, begin, end) dodaje sve elemente između iteratora begin, end i to ispred iterarora it.

U 19. liniji
v.erase(v.begin()+2);
brišemo 3. element vektora v.

U 20. liniji
v.erase(v.begin()+3,v.begin()+6);
brišemo 4., 5., 6. element vektora v.

U 22. liniji
vector <int> ve(3,4);
kreiramo vektor ve koji sadrži tri četvorke.

Vektori se u lineranoj složenosti mogu pridruživati jedni drugima.
Na primer.
```
vector <int> s;
s=ve;
ispisi(s);
```

Vektori se u lineranoj složenosti mogu upoređivati operatorima == != <= < >= >.

U 24. liniji
```
  v.swap(ve);
```
zamenjuju se vektori v i ve u vremenskoj slozenosti O(1).

Nakon trampe vektora metodom swap, svi iteratori koji pokazu na bilo koji od dva zamenjena vektora postaju nevazeci (moraju se ponovo izracunati).

LISTE

Klasa list iz STL-a definisana je u biblioteci (zaglavlju) list i predstavlja impementaciju dvostruko povezane liste. Svaki element u takvoj listi pokazuje na element ispred i iza sebe i na taj nacin omogucuje ubacivanje i izbacivanje elemenata u vremenskoj slozenosti O(1).
Dohvatanje n-tog elementa u takvoj listi ima vremensku slozenost O(n).
POREDJENJE SA kontejnerom vector: Vremenska slozenost izbacivanja u vector je O(n), ali dohvatanje n-tog elementa slozenosti O(1).

Primer 06: uvod u rad sa listom
```
#include <iostream>
#include <list>
using namespace std;

int main()
{ list<int> l;
  l.push_back(3);
  l.push_front(1);
  l.insert(++l.begin(),2);
  list<int>::iterator it;
  for(it=l.begin();it!=l.end();it++)
   cout << * it << ' ';
    return 0;
}
```

IZLAZ
```
1 2 3
```

U 6. liniji
```
  list<int> l;
```
kreiramo praznu listu celih brojeva koju nazovemo l.

Mogli smo kreirati i listu sa 10 praznih elemenata na sledeci nacin (slicno kao kod vektora)
```
list<int> l(10);
```

Mogli smo kreirati i listu sa 5 elemenata jednakih 7 na sledeci nacin (slicno kao kod vektora)
```
list<int> l(5,7);
```

U 7. liniji
```
 l.push_back(3);
```
dodaje se 3 na kraj liste. Vremenska slozenost ove operacije je O(1).

U 8. liniji
```
 l.push_front(1);
```
dodaje se 1 na pocetak liste. Vremenska slozenost ove operacije je O(1).

Takodje, kao i za vektor postoje metodi pop_front(), pop_back(). Postoje i metodi front() i back() koje u vremenskoj slozenosti O(1) vracaju prvi i poslednji element liste.

Pomocu navedenih metoda se lista moze koristiti kao queue ili stack sa neogranicenim kapacitetom. O kapacitetu se brine lista.

U 9. liniji
```
l.insert(++l.begin(),2);
```
je ubacen broj 2 ispred 2. elementa.

Naredbu `l.insert(++l.begin(),2);`
nismo smeli napisati kao
```
l.insert(l.begin()+1,2);
```
jer iteratori liste nisu isti kao i iteratori vektora (slabiji su iteratori liste) i nemaju preopterecen operator sabiranja i oduzimanja, nego samo operatore ++ i --. To je i logicno, jer bi se operatori + i – morali izvoditi u slozenosti O(n).

Metod ***insert(iterator, vrednost)*** se izvrsava u vremenskoj slozenosti O(1), za razliku od vektora koji tu operaciju obavlja u vremenskoj slozenosti O(n). Lista takodje ima metod ***insert*** i ***erase*** kao i vektor, ali su kod liste brži.

Liste se mogu, kao i vektori, uporedjivati u slozenosti O(n) pomocu operatora == != <= < > >=, te se takodje mogu pridruzivati u O(n).

Liste, za razliku od vektora, nemaju preopterecen operator [] i nijma **se ne moze pristupati kao** da se radi o nizu.

Zato 10. liniju programskog koda NISMO mogli ispisati u obliku
```
for (int i=0;i<l.size();i++)   cout << l[i] << ' ';
```

Ispis liste smo obavili u 10. i 11. liniji programskog koda u formi
```
for(it=l.begin();it!=l.end();it++)
 cout << * it << ' ';
```
Koristili smo iteratore koji su inicijalizovani kao it=l.begin().
U 11. liniji pristupamo pojedinacnom elementu sa *it.

Liste (kao i vektori) mogu sadrzavati bilo sta, tako da je moguce definisati:
```
list<vector<list<string> > > listaVektoraSaListomStringova;
```

Metod erase moze izazvati problem ako se ne koristi pazljivo. Ako napisemo naredbu
```
l.erase(it);
```
nakon toga ne smemo dalje koristiti iterator **it**, jer postaje nevazeci (tj. mora se iznova izracunati). To, takodje, vazi i za vektore. Razlog je što brišući element, gubimo element, te ne možemo iz njega pročitati koji element je sledeći. Želimo li nakon brisanja elementa imati i dalje važeći iterator, onda to možemo učiniti ovako:
```
it=l.erase(it);
```

Metod *erase* vraca iterator na element nakon obrisanog.
Takodje, mozemo napisati
```
 l.erase(it++);
```
To je moguce zbog nacina na koji je operator ++ preopterecen.

Ako napisemo
```
*it=2;
l.insert(it,1);
```
onda se element 1 ubacuje pre elementa 2, a iterator *it* ce i dalje pokazivati na element s vrednoscu 2.

Lista takodje ima metodu *swap* kao i vektor, koja je O(1), kao i metodi *clear* i *empty*.

Lista ima dodatni metod *reverse* koji okrene listu u O(n) i metod *sort* koji sortira listu u slozenosti O(n log n).

Primer 07: metodi za rad sa listom
```
#include <iostream>
#include <list>
```

```
using namespace std;

void ispis (list<int> &li)
{ for(list<int>::iterator x=li.begin();x!=li.end();x++)
    cout << *x << ' ';
    cout << endl;
}

int main()
{  list<int> l;
   l.push_back(3); l.push_back(2);
   l.push_back(4); l.push_back(1);
   ispis(l);
   l.reverse();
   ispis(l);
   l.sort();
   ispis(l);
   l.reverse();
   ispis(l);
    return 0;
}
```

U 5. liniji ne mozemo pisati
const list<int> &li
umesto
list<int> &li

Zbog toga je definisan poseban iterator zvan const_iterator pomocu kog se moze iterirati (znaci nije nepromenjiv), ali

PAIR

Klasa *pair* (tj. par) nalazi se u biblioteci *utility,* ali je često ne moramo uključiti (include-ovati), jer je uključe razne druge biblioteke. Klasa *pair* je veoma jednostavna i koristi se kada želimo da čuvamo dva različita tipa podataka pod istim nazivom.
Par ne podržava iteriranje, te nije kontejner. Često se dešava da funkcije koje moraju vratiti dve vrednosti, vrate par vrednosti.

Primer 01

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

int main()
{   pair<int,double> a;
    pair<int,int> b(3,4);
    pair<int,int> c(6,2);
a=make_pair(2,3.14);
cout << a.first << ' ' << a.second << endl;
cout << (b<c) << ' ' << (b==c) << endl;
b=c;
cout << (b==c)  << endl;

pair<pair<int,int>,int> trojka;
trojka=make_pair(make_pair(1,2),3);
cout << trojka.first.first << ' ';
cout << trojka.first.second << ' ';
cout << trojka.second << endl;
```

```
    return 0;
}
```

U liniji
```
  pair<int,double> a;
```
kreiran je par celog i realnog broja

U naredne dve linije kreirani su parovi <int,int> kojima su pridružene vrednosti.

Naredbom a=make_pair(2,3.14); pridružene su vrednosti paru a.

Paru se može pristupiti pomoću članova *first* i *second*.
Parovi se mogu i porediti i to tako što se najpre porede prvi elementi iz para, a ao su oni jednaki porede se i drugi elementi. Zato je par b(3,4) < c(6,2)

Par se cesto koristi za kreiranje ovakvih struktura:
```
vector  <pair<int,int> > v1;
vector  <pair< vector<int>, list<int> > > v2;
```

Primer 02

1. Data je pravougaona mrežu ulica predgrađa Beograda poznatog kao Bisergrad. Postoji 50000 vertikalnih ulica u smeru sever-jug (označenih x-koordinatama od 1 do 50000) i 50000 horizontalnih ulica u smeru istok-zapad (označenih y-koordinatama od 1 do 50000). Sve ulice su dvosmerne. Presek horiznotalne i vertiklane ulice naziva se biserraskršće. Stanovnici Bisergrada mogi biti vrlo neodgovorni i nesaavesni vozači, te je iz zbog bezbednosnih potreba, gradonačelnik Bisergrada postavio semafore na **N** biserraskršća. **Put** između dva biserraskršće je **opasan** ako na njemu negde postoji **skretanje bez semafora**, a inače je bezopasan. Nije moguće osigurati da svi putevi budu bezopasni, ali gradonačelniku je dovoljno da **između svaka dva semafora** bar **jedan od najkraćih puteva** bude **bezopasan**. Na žalost, trenutni raspored semafora to ne osigurava. Vaš je zadak postaviti **još neke semafore** (manje od 700 000) tako da skup semafora (koji sadrži i stare i nove semafore!) zadovoljava traženi zahtev. Razmislite i pomozite Bisergradu!

ULAZ

U prvoj liniji standardnog ulaza nalazi se prirodan broj **N** ($2 \leq \mathbf{N} \leq 50000$), broj postavljenih semafora. U svakoj od sledećih **N** linija nalazi se lokacija jednog semafora, predstavljena prirodnim brojevima **X** i **Y** ($1 \leq \mathbf{X}, \mathbf{Y} \leq 50000$), koordinatama vertikalne i horizontalne ulice koje se seku u tom biserraskršću. Svi semafori biće jedinstveni.

IZLAZ

Ispišite lokacije novih semafora, svaku u posebnoj liniji. Dozvoljeno je postavljanje više semafora na istu lokaciju. Broj novih semafora mora biti **manji od 700 000**.

PRIMERI TEST PODATAKA

ULAZ	**IZLAZ**
2 1 1 3 3	1 3
3 2 5 5 2 3 3	3 5 5 3
5 1 3 2 5 3 4 4 1 5 2	1 5 3 3 3 5 4 2 4 3

Resenje:

Date semafore sortirajmo po x-koordinati. Neka je **x'** srednja (median) x koordinata
u tom nizu. Neka je **A** skup datih semafora levo od **x'**, a **B** skup datih semafora desno od **x'**.

Povežimo bezopasnim putem svaki par semafora **a**, **b** takvih da je **a** iz skupa **A**, te **b** iz skupa **B**. Kako? Tako da dodamo nove semafore na lokacije (**x'**, **y**) za sve y-koordinate **y** iz skupova **A** i **B**. Sada za semafore **a** i **b** imamo bezopasan put (x_a, y_a) → (**x'**, y_a) → (**x'**, y_b) → (x_b, y_b).

Još je preostalo povezati međusobno semafore unutar skupa **A**, kao i semafore unutar skupa **B**. To činimo tako da rekurzivno ponovimo opisani postupak posebno za skup **A** i posebno za skup **B**. Ovaj način razmišljanja naziva se *podeli pa vladaj* (divide and conquer).

Koliki je broj dodatih semafora? Budući da delimo skup na dva dela, najveća je dubina rekurzije O(log **N**). Posmatrajmo neki početni semafor na lokaciji (**x**, **y**). U najgorem slučaju, na svakoj dubini rekurzije on će biti uključen u jedan skup i tamo će generisati jedan novi semafor (**x'**, **y**). Dakle, jedan početni semafor generise O(log **N**) novih semafora, što daje ukupno O(**N** log **N**) novih semafora.
Tačan broj uz pažljivu implementaciju ispada manji od 700000.

```
#include <algorithm>
#include <cstdio>
using namespace std;

const int NN = 100005;
pair<int, int> t[NN];

void divcon(int lo, int hi) {
  if (lo + 2 == hi) {
    printf("%d %d\n", t[lo].first, t[lo + 1].second);
    return;
  }
  if (lo + 2 > hi) return;
  int mid = (lo + hi) / 2;
  int x = t[mid].first;
  for (int i = lo; i < hi; ++i)
    if (i != mid)
      printf("%d %d\n", x, t[i].second);
  divcon(lo, mid);
  divcon(mid + 1, hi);
}

int main () {
  int n;
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) {
    scanf("%d%d", &t[i].first, &t[i].second);
  }
  sort(t, t + n);
  divcon(0, n);
}
```

SKUP (klasa set)

Klasa set definisana je u biblioteci set.
SET je kontejner koji implementira veoma korisnu strukturu podataka, eng. red-black tree koja u sebi održava elemente poređane po veličini (ili po nekom našem poretku).

SET je sličan matematičkom pojmu skup. Svaki element je jedinstven. Ubacivanje i uklanjanje elementa sprovodi se za vreme O(log n). Svim elementima može se pristupati (za razliku od priority_queue) i može se iterirati kroz njih pomoću iteratora koji podržavaju ++ i --.
Primer 01

```
#include <iostream>
#include <set>
using namespace std;
inline void ispis(const set<int> &s)
{  set<int>::const_iterator it;
     for(it=s.begin();it!=s.end();it++)
       cout << *it << ' ';
   cout << endl;
}

int main()
{ set <int> s;
  s.insert(3); s.insert(5); s.insert(2);
  s.insert(6); s.insert(1); s.insert(-3);
  ispis(s);
  for(int i=4;i<10;i++) s.insert(i);
  ispis(s);
  s.erase(3);
  s.erase(s.find(7));
  ispis(s);
  s.erase(s.lower_bound(-2));
  ispis(s);
  s.erase(s.upper_bound(5));
  ispis(s);
  cout << "Ima ih: " << s.size() << endl;
  s.clear();
  if (s.empty()) cout << "prazan" << endl;
   return 0;
}
```

MAPA

Klasa map je struktura set<pair<kljuc, vrednost> >

Struktura map koju pretrazujemo pomocu kljuca kojim pristupamo polju vrednost. Za svaki kljuc postoji jedinstvena vrednost. Mapa ima preopterecen operator [] pomocu kog preko kljuca pristupamo polju vrednost. Mapa se nalazi u biblioteci map.

Primer01 – mapa

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{   map<string,int> m;
   m["dvadeset"]=20;
   m[string("pet")]=5;
   m.insert(pair<string,int>("dva",2));
   m.insert(make_pair(string("sto"),100));
```

```
    m.insert(make_pair("pedeset",50));
    map<string,int>::iterator it;
    for(it=m.begin(); it!=m.end();it++)
     cout << it->first << ' ' << it->second << endl;
     return 0;
}
```

IZLAZ
dva 2
dvadeset 20
pedeset 50
pet 5
sto 100

Primer 02
Pekar Joca prodaje burek u **tri oblike** – **četvrtina, polovina**, **tri četvrtine.** Za potrebe proslave skole koja se nalazi u blizini pekare, dobio je porudzbine. Skolska deca su mala niko od njih ne može pojesti ceo burek ali svako je prijavio svojoj uciteljici koliko parce Jocinog bureka može pojesti. Uciteljice instiraju da pekar Joca postuje njihov spisak tako da svako dete dobije **tačnu kolicinu zeljenog bureka.** Napišite program koji će pomoći Joci da odredi koliki je **minimalni** broj bureka koje mora napraviti kako bi svako dete dobilo tačno onoliki komad koliki želi.
Ulaz
U prvom redu standardnog ulaza se nalazi ceo broj N, $1 \leq N \leq 10{,}000$, broj dece koja ce jesti burek.
U svakom od sledećih redova standardnog ulaza nalazi se veličina bureka koju svako od dece želi pojesti tj. razlomak **1/4**, **1/2** ili **3/4**.
Izlaz
U prvi i jedini red standardnog ulaza ispisati traženi minimalni broj bureka koje Joca treba da napravi.

PRIMERI
ulaz
3
1/2
3/4
3/4
izlaz
3

ulaz
5
1/2
1/4
3/4
1/4
1/2
izlaz
3

ulaz
6
3/4
1/2
3/4
1/2
1/4
1/2
izlaz
4

Resenje:

```
#include <cstdio>
#include <map>
#include <string>

using namespace std;

map<string, int> broj;

int main( void ) {
  int n;
  scanf( "%d", &n );
  for( int i = 0; i < n; ++i ) {
    char linija[10];
    scanf("%s", linija );
    broj[linija]++;
  }

  int burek_broj = broj["3/4"] + broj["1/2"] / 2;

  broj["1/4"]  = max( broj["1/4"] - broj["3/4"], 0 );
  broj["1/4"] += 2 * (broj["1/2"] % 2);

  burek_broj += (broj["1/4"] + 3) / 4;

  printf( "%d\n", burek_broj );

  return 0;
}
```

Multimap

Map ima iste metode kao i set. Slicno kao sto postiji multiset u biblioteci set, tako postoji i multimap u biblioteci map. Ali, multimap nema operator [], jer za vise istih kljuceva moze imati razlicite vrednosti.

Multimap kontejner može da se koristi umesto hash strukture. Svaki element je sastavljen od dva dela: ključ i vrednost. Struktura dozvoljava ponavljanje ključeva po kom se vrši pretraga .
Ključevi po kojima se pretražuju moraju biti uporedljivi, tj. mora da može da se nad njima definiše relacija <. Za standardne tipove podataka, uključujući i biblioteku string, već imaju definisne ove operatore za poređenje.

Da bi mogla da se koristi ova struktura, potrebno je na početku programa da se uključi sledeća direktiva

```
#include <map>
```

Primer

```
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main()
{
// Definisanje multimap-a
multimap<string,int> mymm;
// Definisanje iteratora za ovakav tip multimape
multimap<string,int>::iterator it;
// Unos elemenata u multimap-u
mymm.insert(pair<string,int>("marko",10));
mymm.insert(pair<string,int>("ana",20));
```

```
mymm.insert(pair<string,int>("ana",30));
mymm.insert(pair<string,int>("mica",40));
mymm.insert(pair<string,int>("maca",50));
mymm.insert(pair<string,int>("kuca",60));
mymm.insert(pair<string,int>("ana",60));

// Stampanje elemenata multimap-e
for (it = mymm.begin();it != mymm.end();++it)
cout << " [" << (*it).first << ", " << (*it).second << "]" << endl;

// Stampanje svih vrednosti pri svim pojavljivanjima kljuca ana
pair<multimap<string,int>::iterator,multimap<string,int>::iterator> ret;
ret = mymm.equal_range("bliksa");
for (it=ret.first; it!=ret.second; ++it)
cout << " " << (*it).second;
cout << endl;
// Broj elemenata sa kljucem ana
cout<<endl<<"Broj elemenata sa kljucem 'ana' : "<<mymm.count("ana");
// Dodavanje elementa na pocetak multiset-a
it=mymm.begin();
mymm.insert(it,pair<string,int>("bla",123));
// Stampamo elemente multimap-e
for (it = mymm.begin();it != mymm.end();++it)
cout << " [" << (*it).first << ", " << (*it).second << "]" << endl;
return 0;
}
```