Zadaci sa rešenjima SIO 2010.

zadatak: Recnik

Dat je niz reči dužine n kojeg ćemo u dalje tekstu zvati rečnik. Reči su sastavljene od malih slova engleskog alfabeta. Za tekst *S* koji je takođe sastavljen od malih slova engleskog alfabeta, kažemo da ima razbijanje (r_1, r_2, ... , r_m) ukoliko:

- r_i pripada rečniku za $i \in [1, m]$
- $r_1 r_2 \ldots r_m = S$ (konkatenacija, nadovezivanje reči)

Za rečnik kažemo da je nedvosmislen ukoliko svaka reč koja se može dobiti nadovezivanjem reči iz rečnika ima jedinstveno razbijanje.
Napisati program koji za dati rečnik ispituje da li je dvosmislen ili ne.
Ulaz.
(Ulazni podaci se učitavaju iz datoteke **recnik.in**.) U prvom redu ulazne datoteke nalazi se prirodni broj $T \leq 10$ koji označava broj primera. Svaki primer je opisan na sledeći način: u prvom redu se nalazi prirodan broj n ($2 \leq n \leq 100$) koji označava broj reči u rečniku. Narednih n linija opisuju rečnik (u svakom redu je data po jedna reč). Između primera neće biti praznih redova. Reči će biti sastavljene od malih slova engleskog alfabeta i njihova dužina neće biti veća od 1000.
Izlaz.
(Izlazne podatke upisati u datoteku **recnik.out**) Za svaki primer u uzlaznoj datoteci, u redosledu kojim su dati na ulazu, šampati u posebnom redu dvosmislen ukoliko je rečnik *dvosmislen*; u suprotnom štampati *nedvosmislen*.
Primer 1.

recnik.in	**recnik.out**
2	dvosmislen
4	nedvosmislen
andre	
and	
jko	
rejko	
3	
a	
ab	
bb	

fajl: recnik.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>

const int MaxN = 110;
const int MaxD = 1010;
const int MaxT = 11;
const int MOD = 10000019;

struct Node {
  int ind, d;
  Node(int _ind, int _d) {
    ind = _ind; d = _d;
  }
  Node() {}
};

int n, T;
char s[MaxN][MaxD];
bool sol[MaxT];
int len[MaxN];
```

```
int preff[MaxN][MaxD], pow26[MaxD];
Node Q[MaxN * MaxD];
bool mark[MaxN][MaxD];


void makeHash() {
  int val;

  pow26[0] = 1;
  for (int i = 1; i < MaxD; i++)
    pow26[i] = (pow26[i - 1] * 26 ) % MOD;
  for (int i = 0; i < n; i++) {
    val = 0;
    for (int j = 0; j < len[i]; j++) {
      val = (26 * val + (s[i][j] - 'a')) % MOD;
      preff[i][j] = val;
    }
  }
}


// hash vrednost s[num] od l-tog do r-tog karaktera
int hashValue(int num, int l, int r) {
  int val;
  if (l == 0) return preff[num][r];
  long long tmp = pow26[r - l + 1];
  val = (preff[num][r] - preff[num][l - 1] * tmp) % MOD;
  //val = (preff[num][r] - preff[num][l - 1] * pow26[r - l + 1]) % MOD;
  if (val < 0) val += MOD;
  return val;
}


// da li postoji put [ind1][d] -> [ind2][nesto] u grafu
bool match(int ind1, int d, int ind2) {
  int l1, r1, l2, r2, i, D;
  bool ok;
  if (ind1 == ind2 && d == len[ind1]) return false;

  D = (len[ind2] <= d ? len[ind2] : d);
  l1 = len[ind1] - d;  r1 = l1 + D - 1;
  l2 = 0;  r2 = D - 1;

  if (hashValue(ind1, l1, r1) != hashValue(ind2, l2, r2))
    return false;
  ok = true; i = 0;
  while (i < D && ok) {
    ok = ok && (s[ind1][l1 + i] == s[ind2][l2 + i]);
    i++;
  }
  return ok;
}


bool BFS() {
  int first, last, ind, d;
  Node u;
  bool found;

  memset(mark, false, sizeof(mark));
  first = 0; last = 0;
  for (int i = 0; i < n; i++) {
    last++;
    Q[last] = Node(i, len[i]);
    mark[i][len[i]] = true;
  }
  found = false;
```

```
  while (first < last && !found) {
    first++;
    u = Q[first];
    for (int i = 0; i < n; i++) {
      if (len[i] <= u.d) {
        ind = u.ind; d = u.d - len[i];
      }
      else {
        ind = i; d = len[i] - u.d;
      }
      if (!mark[ind][d])
        if (match(u.ind, u.d, i)) {
          last++;
          Q[last] = Node(ind, d);
          mark[ind][d] = true;
          if (d == 0) found = true;
        }
    }
  }
  return found;
}


void solve(int testNum) {
  makeHash();
  sol[testNum] = BFS();
}

int main() {

  FILE* inFile = fopen("recnik.in", "r");
  FILE* outFile = fopen("recnik.out", "w");

  fscanf(inFile, "%d", &T);
  int max = 0;
  for (int i = 0; i < T; i++) {
    fscanf(inFile, "%d", &n);
    for (int j = 0; j < n; j++) {
      fscanf(inFile, "%s", s[j]);
      len[j] = strlen(s[j]);
      if (len [j] > max)
        max = len [j];
    }
    makeHash();
    sol[i] = BFS();
  }

  for (int i = 0; i < T; i++) {
    if (sol[i]) fprintf(outFile, "dvosmislen\n");
    else fprintf(outFile, "nedvosmislen\n");
  }

  fclose(inFile);
  fclose(outFile);
  return 0;
}
```

zadatak: Knjige

Mali Đurica je veliki obožavatelj knjiga i čvrsto je rešio da pročita što više knjiga u narednih godinu dana. Prvi, jelte, logičan korak je da ode u biblioteku i tamo potraži štivo za čitanje, što je Đurica i uradio. Radi pojednostavljenja, biblioteku ćemo zamisliti kao niz knjiga. Đurica smatra da će čitanje biti zanimljivije ako čita knjige različitih autora. Shodno tome, prva stvar koju je Đurica zaključio da mu je neophodna jeste da prebroji koliko različitih autora je učestvovalo u pisanju datog podniza uzastopnih knjiga.

Tu stupate vi. Baš vas je zamolio Đurica da mu pomognete i date odgovor na dato pitanje. Jedna olakšavajuća okolnost je to što za svaku knjigu važi da je pisao isključivo jedan autor. Umesto imena autora knjige, Đurica će vam dati jedinstveni broj autora, koji je u stvari broj iz intervala [1, 100 000]. Svaki autor ima tačno jedan jedinstveni broj, i svaki jedinstven broj odgovara tačno jednom autoru.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **knjige.in**.) U prvom redu nalaze se celi brojevi n ($1 \leq n \leq 100{:}000$) i K ($1 \leq K \leq 100{:}000$). U narednom redu se učitava n celih brojeva iz intervala [1, 100 000] koji redom predstavljaju jedinstvene brojeve autora knjiga datog niza. Potom se u K redova učitavaju redom indeksi $idxs_i$ i $idxe_i$ ($idxs_i \leq idxe_i$, $1 \leq i \leq$ K) koji predstavljaju indeks početka i kraj podniza knjiga, redom, za koji treba odgovoriti koliko različitih autora je učestvovalo u pisanju knjiga tog podniza.

Izlaz.

(Izlazne podatke upisati u datoteku **knjige.out**) U K redova ispisati odgovore na postavljena pitanja, u i-tom redu odgovor za podniz čiji su indeksi $idxs_i$ i $idxe_i$ iz ulaza.

Primer 1.

knjige.in	knjige.out
5 6 1 2 3 4 3 1 5 2 2 3 5 1 4 2 5 2 4	4 1 2 4 3 3

Primer 2.

knjige.in	knjige.out
4 3 1 1 12 12 1 4 1 2 3 4	2 1 1

Napomena.

- U 40% test primera jedinstveni brojevi autora će se kretati u intervalu [1, 60].
- U 30% test primera će i broj upita i broj različitih autora biti najviše 1000.
- U 60% test primera će biti ispunjen bar jedan od prethodna dva uslova.

fajl: knjige.cpp

```
/*
  Ovo je zvanicno resenje. Resenje je off-line.
  Slozenost resenja po upitu je O(log broj_autora).
  Nakon ucitavanja intervala, sortiramo sve intervale.
  Shodno tome, slozenost celog programa (gde je I
  broj intervala, A broj autora, a N broj knjiga) je
  O(I * log A + I * log I + N)
*/
#include <algorithm>
#include <iostream>
#include <cstdio>
#include <cmath>
#include <queue>
#include <cstring>
#include <string>
#include <sstream>
#include <vector>
#define ffor(_a,_f,_t) for(int _a=(_f),__t=(_t);_a<__t;_a++)
#define all(_v) (_v).begin() , (_v).end()
#define sz size()
#define pb push_back
#define SET(__set, val) memset(__set, val, sizeof(__set))
#define FOR(__i, __n) ffor (__i, 0, __n)
#define syso system("pause")
```

```
#define mp make_pair

using namespace std;

const int MAXN = 100000;
const int MAXK = 100000;
const int TYPE_BOOK = 0;
const int TYPE_INTERVAL = 1;

struct myt{
  int s, e, val, type, idx;
  myt(){

  }

  myt(int _s, int _e, int _val, int _type, int _idx){
    s = _s;
    e = _e;
    val = _val;
    type = _type;
    idx = _idx;
  }

  friend bool operator <(const myt &a, const myt &b){
    if (a.e < b.e)
      return true;
    if (a.e > b.e)
      return false;
    if (a.type == TYPE_BOOK && b.type == TYPE_INTERVAL)
      return true;
    return false;
  }

};

int a[MAXN];

int bit[MAXN + 1];

void update(int idx, int val){
  while (idx <= MAXN){
    bit[idx] += val;
    idx += (idx & -idx);
  }
}

int read(int idx){
  int ret = 0;
  while (idx){
    ret += bit[idx];
    idx -= (idx & -idx);
  }

  return ret;
}

int lastIdx[MAXN];

myt vals[MAXN + MAXK];
int ret[MAXK];

int main(){
//  clock_t begin, end;
//  begin = clock() * CLK_TCK;

  freopen("knjige.out", "wt", stdout);
  freopen("knjige.in", "r", stdin);
```

```
  int n, k, v, cnt = 0;
  scanf("%d %d", &n, &k);
  FOR (i, n){
    scanf("%d", &v);
    v--;
    vals[cnt++] = myt(i + 1, i + 1, v, TYPE_BOOK, i);
  }

  // kroz sve upite
  int idxs, idxe;
  FOR (i, k){
    scanf("%d %d", &idxs, &idxe);
    vals[cnt++] = myt(idxs, idxe, -1, TYPE_INTERVAL, i);
  }

  sort(vals, vals + n + k);
  SET(lastIdx, 255);
  FOR (i, cnt)
    if (vals[i].type == TYPE_BOOK){
      v = vals[i].val;
      if (lastIdx[v] != -1)
        update(lastIdx[v], -1);
      lastIdx[v] = vals[i].s;
      update(lastIdx[v], 1);
    }
    else
      ret[vals[i].idx] = read(vals[i].e) - read(vals[i].s - 1);

  FOR (i, k)
    printf("%d\n", ret[i]);

//  end = clock() * CLK_TCK;
//  cout << (end - begin) / 1000.0 << endl;

  return 0;
}
```

zadatak: NishTel

Kompanija *NishT* el trenutno ugrađuje širom zemlje svoj najnoviji hit - brze jednosmerne međugradske telefnoske linije. Do sada su napravili *m* jednosmernih linija između nekih od *n* gradova. Svaka linija povezuje neka dva grada i ukoliko postoji linija od grada *A* do grada *B*, tada je preko nje moguće iz grada *A* zvati grad *B* ali ne i obratno (to je jedna od mana sistema ali linije su mnogo jeftinije od starih, a to je ono što je bitno). Svaku jednosmernu telefonsku liniju karakteriše jedan broj t_i - vreme (u mikrosekundama) potrebno da se uspostavi veza između odgovarajućih gradova.
Grad *A* može da zove grad *B* i indirektno, ako postoji neki niz gradova, gde je *A* prvi, a *B* poslednji, i između svaka dva uzastopna grada postoji telefonska linija u odgovarajućem smeru. Tada je vreme za uspostavljanje veze između *A* i *B* jednako zbiru vremena za uspostavlja nje svih međuveza na putu. Ukoliko postoji više načina da se uspostavi veza između neka dva grada, operateri uvek biraju onaj kod koga je vreme uspostavljanja minimalno.
Poznato je da *NishTel* ima centre u dva grada (*NishB* i *ABNish*) kao i da je iz svakog od ta dva grada moguće zvati (direktno ili indirektno) bilo koji drugi grad. Takođe je poznato da je *NishTel* zapao u dugove i da mora da sruši neke od *m* linija koje su postavili. Oni žele da sruše **što više** linija ali tako da **minimalna vremena** potrebna za uspostavljanje veza od gradova *NishB* i *ABNish* do svakog od ostalih gradova **ostanu ista** (uključujući i vremena između dva centra). Pomozite *NishTel*-u da ispuni svoj plan i spase se stečaja.
Ulaz.
(Ulazni podaci se učitavaju iz datoteke **nishtel.in**.) U prvom redu ulazne datoteke nalaze se 4 prirodna broja *n*, *m*, *A* i *B* koji predstavljaju, redom, broj gradova, broj telefonskih linija i redne brojeve gradova *NishB* i *ABNish* ($n \leq 10^4$, $m \leq 10^5$). Gradovi su numerisani brojevima od 1 do *n*. U narednih *m* redova nalaze se po tri prirodna broja a_i, b_i i t_i koji označavaju da postoji linija od grada a_i do grada b_i i da je potrebno t_i mikrosekundi za uspostavljanje veze ($a_i \neq b_i$, $t_i \leq 10^5$). Između dva grada mogu postojati više telefonskih linija.
Izlaz.
(Izlazne podatke upisati u datoteku **nishtel.out**) U prvom i jedinom redu izlazne datoteke ispisati najveći broj telefonskih linija koje je moguće srušiti.

Primer 1.

nishtel.in	nishtel.out
4 6 1 2 1 2 1 2 1 5 2 4 1 1 4 2 1 3 3 4 3 2 2 1 4	2

Objašnjenje.
Rušenjem druge i četvrte telefonske linije, minimalna rastojanja od centara do ostalih gradova ostaju ista, dok je to nemoguće ako srušimo tri ili više linija.
Napomena.
U 50% test primera biće $n \leq 10^3$.

fajl: nishtel.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <vector>

using namespace std;

const int inf = 2000000000;
const int MaxN = 10010;
const int MaxM = 100010;

struct edge {
  int v, w;
  edge(int _v, int _w) {
    v = _v; w = _w;
  }
};

int n, m, A, B, u, v, w, sol;
int dA[MaxN], dB[MaxN], d[MaxN];
int adj[MaxM], weight[MaxM], a[MaxM], b[MaxM], w1[MaxM];
int p[MaxN], deg[MaxN];
bool mark[MaxN];

struct Heap {
  int heap_size;
  int h[MaxN];
  int posInHeap[MaxN];

  void init(int n) {
    heap_size = n;
    for (int i = 1; i <= n; i++) {
      h[i] = i;
      posInHeap[i] = i;
    }
  }

  bool empty() { return (heap_size == 0); }

  void update(int u, int newVal) {
    int f, s;
    d[u] = newVal;
    s = posInHeap[u];
    f = s / 2;

    while (f != 0 && d[ h[f] ] > d[u]) {
```

```
      h[s] = h[f];
      posInHeap[ h[f] ] = s;
      s = f; f = s / 2;
    }
    h[s] = u;
    posInHeap[u] = s;
  }

  int extractMin() {
    int f, s, u, min;
    min = h[1];
    h[1] = u = h[heap_size--];
    f = 1;
    s = 2 * f;
    if (heap_size > s && d[ h[s + 1] ] < d[ h[s] ]) s++;

    while (s <= heap_size && d[u] > d[ h[s] ]) {
      h[f] = h[s];
      posInHeap[ h[s] ] = f;
      f = s; s = 2 * f;
      if (heap_size > s && d[ h[s + 1] ] < d[ h[s] ]) s++;
    }
    h[f] = u;
    posInHeap[u] = f;

    return min;
  }
};

Heap H;

void Dijkstra(int START) {
  int u, v, w, min;
  for (int i = 1; i <= n; i++) d[i] = inf;
  H.init(n);
  H.update(START, 0);

  while (!H.empty()) {
    u = H.extractMin();
    for (int i = p[u]; i < p[u] + deg[u]; i++) {
      v = adj[i];
      w = weight[i];
      if (d[u] + w < d[v])
        H.update(v, d[u] + w);
    }
  }
}

int main() {

  FILE* inFile = fopen("nishtel.in", "r");
  FILE* outFile = fopen("nishtel.out", "w");

  for (int i = 1; i <= n; i++) deg[i] = 0;
  fscanf(inFile, "%d%d%d%d", &n, &m, &A, &B);
  for (int i = 1; i <= m; i++) {
    fscanf(inFile, "%d%d%d", &a[i], &b[i], &w1[i]);
    deg[a[i]]++;
  }

  p[0] = 0;
  for (int i = 1; i <= n; i++)
    p[i] = p[i - 1] + deg[i - 1];

  for (int i = 1; i <= m; i++) {
    adj[p[a[i]]] = b[i];
    weight[p[a[i]]] = w1[i];
```

```
      p[a[i]] = p[a[i]] + 1;
  }
    for (int i = 1; i <= n; i++)
    p[i] = p[i - 1] + deg[i - 1];

  Dijkstra(A);
  for (int i = 1; i <= n; i++) dA[i] = d[i];
  Dijkstra(B);
  for (int i = 1; i <= n; i++) dB[i] = d[i];

  sol = m - 2 * n + 2;
  for (int i = 1; i <= n; i++) mark[i] = false;

  for (u = 1; u <= n; u++)
    for (int i = p[u]; i < p[u] + deg[u]; i++) {
      v = adj[i];
      w = weight[i];
      if (!mark[v] && dA[v] - dA[u] == w && dB[v] - dB[u] == w) {
        mark[v] = true;
        sol++;
      }
    }
  fprintf(outFile, "%d\n", sol);

  fclose(inFile);
  fclose(outFile);
  return 0;
}
```

fajl: nishtel.pas

```
const
   MaxN = 10010;
   MaxM = 100010;
   inf = 2000000000;


var
   inFile, outFile : Text;
   n, m, A, B, u, v, w, sol, i, heap_size : longint;
   a1, b1, w1, adj, weight : array[0..MaxM] of longint;
   d, dA, dB, p, deg : array[0..MaxN] of longint;
   h, posInHeap : array[0..MaxN] of longint;
   mark : array[0..MaxN] of boolean;


procedure Input;
var
   i, u, v, w: longint;

begin

   assign( inFile, 'nishtel.in' );
   reset( inFile );
   fillchar( deg, sizeof( deg ), 0 );

   readln( inFile, n, m, A, B );
   for i := 1 to m do begin
      readln( inFile, u, v, w );
      deg[ u ] := deg[ u ] + 1;
      a1[ i ] := u; b1[ i ] := v; w1[ i ] := w;
   end;

   close( inFile );
```

```
end;


procedure Output;
begin

   assign( outFile, 'nishtel.out');
   rewrite( outFile );
   writeln( outFile, sol);
   close( OutFile );

end;


(* simulacija liste preko counting sorta *)
procedure ListSimulation;
var
   i: longint;

begin

   p[0] := 0;
   for i := 1 to n do
      p[ i ] := p[ i - 1 ] + deg[ i - 1 ];

   for i := 1 to m do begin
      adj[ p[ a1[ i ] ] ] := b1[ i ];
      weight[ p[ a1[ i ] ] ] := w1[ i ];
      p[ a1[ i ] ] := p[ a1[ i ] ] + 1;
   end;

   for i := 1 to n do
      p[ i ] := p[ i - 1 ] + deg[ i - 1 ];

end;


procedure updateHeap( u, newVal : longint );
var
   f, s : longint;

begin

   d[ u ] := newVal;
   s := posInHeap[ u ];
   f := s div 2;

   while ((f <> 0) and (d[ h[ f ] ] > d[ u ])) do begin
      h[ s ] := h[ f ];
      posInHeap[ h[ f ] ] := s;
      s := f; f := s div 2;
   end;

   h[ s ] := u;
   posInHeap[ u ] := s;

end;


function extractMin: longint;
var
   f, s, u, min : longint;

begin

   min := h[ 1 ];
   u := h[ heap_size ];
```

```
   h[ 1 ] := u;
   heap_size := heap_size - 1;
   f := 1;
   s := 2 * f;
   if ((heap_size > s) and (d[ h[ s + 1 ] ] < d[ h[ s ] ])) then
      s := s + 1;

   while ((s <= heap_size) and (d[ u ] > d[ h[ s ] ])) do begin
      h[ f ] := h[ s ];
      posInHeap[ h[ s ] ] := f;
      f := s; s := 2 * f;
      if ((heap_size > s) and (d[ h[ s + 1 ] ] < d[ h[ s ] ])) then
         s := s + 1;
   end;

   h[ f ] := u;
   posInHeap[ u ] := f;
   extractMin := min;

end;


procedure Dijkstra( START : longint );
var
  u, v, w : longint;

begin

   for i := 1 to n do begin
      d[ i ] := inf;
      h[ i ] := i;
      posInHeap[ i ] := i;
   end;
   heap_size := n;
   updateHeap( START, 0 );

   while (heap_size > 0) do begin
      u := extractMin;
      for i := p[ u ] to p[ u ] + deg[ u ] - 1 do begin
         v := adj[ i ]; w := weight[ i ];
         if (d[ u ] + w < d[ v ]) then
            updateHeap( v, d[ u ] + w );
      end;
   end;

end;


begin

   Input;
   ListSimulation;

   Dijkstra( A );
   for i := 1 to n do dA[ i ] := d[ i ];
   Dijkstra( B );
   for i := 1 to n do dB[ i ] := d[ i ];

   sol := m - 2 * n + 2;
   for i := 1 to n do mark[ i ] := false;

   for u := 1 to n do
      for i := p[ u ] to p[ u ] + deg[ u ] - 1 do begin
         v := adj[ i ]; w := weight[ i ];
         if ((not mark[ v ]) and (dA[ v ] - dA[ u ] = w) and (dB[ v ] - dB[ u ] = w))
         then begin
            mark[ v ] := true;
```

```
            sol := sol + 1;
          end;
       end;

   Output;

end.
```

zadatak: Meda

Kao i sve male devojčice, Katarina obožava plišane mede. Zato je na vreme počela da se priprema za Dan plišanih meda koji se održava u njenom gradiću. Na taj dan u svakoj prodavnici igračaka svaki posetilac dobija na poklon po jednog plišanog medu. Postoji n vrsta plišanih meda. Katarina je obilazila prodavnice i zaključila sledeće: Neka su prodavnice numerisane brojevima od 1 do m. Za i-tu vrstu meda postoje brojevi a_i i b_i koji govore da se taj meda prodaje u prodavnicama numerisanim brojevima iz intervala $[a_i, b_i]$. Pre ili kasnije, Katarina će želeti da ima po jednog medu od svake vrste. Zato ona želi da suma cena meda koje ne dobije tokom Dana plišanih meda bude što manja. Vaš zadatak je da odredite minimalnu vrednost te sume. Pretpostavlja se da Katarina svaku prodavnicu može posetiti najviše jedanput i da u prodavnici može izabrati bilo kog od meda koji se u njoj prodaju.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **mede.in**.) U prvom redu ulazne datoteke nalazi se broj n ($n \leq 20.000$), ukupan broj vrsta plišanih meda. Zatim se u narednih n redova nalaze po tri broja - u i-tom redu brojevi a_i, b_i i c_i ($1 \leq a_i \leq b_i \leq 2.000$, $c_i \leq 10.000.000$) - a_i i b_i su granice intervala brojeva prodavnica u kojima se i-ta vrsta meda prodaje, a c_i je cena te vrste. Sve cene meda su različite.

Izlaz.

(Izlazni podaci se ispisuju u datoteku **mede.out**.) U izlaznoj datoteci treba da se nalazi samo jedan broj - minimalna moguća suma cena meda koje Katarina ne dobije tokom Dana plišanih meda.

Primer 1.

mede.in	**mede.out**
6 1 4 1 3 4 2 1 2 3 2 2 4 1 1 5 1 1 6	8

Objašnjenje.

Prvi meda se može pronaći u prodavnicama 1, 2, 3 i 4, i njegova cena je 1. Drugi meda se može pronaći u prodavnicama 3 i 4, i njegova cena je 2, itd. Jedno od rešenja je da u prodavnici 1 dobije medu 6, u prodavnici 2 medu 4, u prodavnici 3 medu 1 i u prodavnici 4 medu 2. Tada joj preostaju mede 3 i 5, njihov zbir cena je 8.

Primer 2.

mede.in	**mede.out**
2 3 10 15 15 20 25	0

Objašnjenje.

Katarina može dobiti obe vrste meda, pa kasnije neće morati da kupi nijednog medu.

fajl: mede.cpp

```
#include <stdio.h>
#include <stdlib.h>

using namespace std;

const int maxN = 20000, maxP = 2001;

struct meda { int start, end, cena; };
```

```
int n;
long long minCena;
meda mede[maxN];
int kupili[maxP], pom[maxP];

int komparatorPoCeni(const void *a, const void *b) {
    meda *m1 = (meda*)a;
    meda *m2 = (meda*)b;
    return (int)(m2 -> cena - m1 -> cena);
}

int main() {
    freopen("mede.in", "r", stdin);
    freopen("mede.out", "w", stdout);

    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d%d%d", &(mede[i].start), &(mede[i].end), &(mede[i].cena));

    // sortiramo mede po ceni, od najskupljeg do najjeftinijeg
    qsort(mede, n, sizeof(meda), komparatorPoCeni);

    // na pocetku su nam sve prodavnice slobodne
    for (int i = 0; i < maxP; i++)
        kupili[i] = -1;
    minCena = 0;

    for (int i = 0; i < n; i++) {
        // pokusavamo da dobijemo i-tog medu
        int tekMeda = i;
        int tekP = mede[i].start;

        // prekidamo ako smo naisli na praznu prodavnicu ili ako nemamo vise prodavnica u
kojima se
        // tekuci meda prodaje
        while ((tekMeda >= 0) && (tekP <= mede[tekMeda].end))
            if (kupili[tekP] == -1) {
                // ukoliko imamo slobodnu prodavnicu dodelicemo medu njoj i zavrsiti
                pom[tekP] = tekMeda;
                tekMeda = -1;
            } else {
                if (mede[tekMeda].end < mede[kupili[tekP]].end) {
                    // ako se meda kupljen u prodavnici koju trenutno posmatramo prodaje u
vise prodavnica
                    // nego meda kojeg pokusavamo da dodamo, onda cemo njega uzeti kao
tekuceg medu, a naseg
                    // medu kupiti u ovoj prodavnici
                    pom[tekP] = tekMeda;
                    tekMeda = kupili[tekP];
                } else pom[tekP] = kupili[tekP];
                tekP++; // prelazimo na sledecu prodavnicu
            }

        if (tekMeda < 0)
            // ako smo imali slobodnu prodavnicu za medu, treba samo promene da sacuvamo
            for (int j = mede[i].start; j <= tekP; j++)
                kupili[j] = pom[j];
        else
            // nismo uspeli da dodamo i-tog medu, treba da povecamo cenu meda koje ne
mozemo dobiti
            minCena += mede[i].cena;
            // stanje prodavnica ostaje kakvo je bilo
    }

    printf("%lld\n", minCena);
    return 0;
}
```

zadatak: Podela

Data je povezana mapa n gradova sa m dvosmernih puteva. Samo dva od n gradova A i B imaju fabriku čokolade. Posle kraćih dogovora, ljudi su odlučili da naprave dve disjunktne povezane mreže koristeći postojećih m puteva, tako da grad A distribuira čokoladu u svom delu, a grad B u svom delu. Svaki grad mora da pripada tačno jednoj mreži. Ako sa W_A i W_B označimo ukupnu dužinu puteva u mrežama koje sadrže A i B, potrebno je minimizirati veću od ukupnih dužina mreža, odnosno minimizirati izraz $\max(W_A, W_B)$.
Za ovaj zadatak potrebno je da predate izlazne datoteke za 10 ulaznih datoteka **podela.01.in**, **podela.02.in**, ..., **podela.10.in**, koji se nalaze u arhivi na računaru. Izlazi se pamte u datoteke **podela.01.out**, **podela.02.out**, ..., **podela.10.out**, pri čemu broj u nazivu izlazne datoteke odgovara broju u nazivu ulazne datoteke.

Ulaz.
U prvom redu ulaza se nalaze dva prirodna broja n i m. U drugom redu se nalaze indeksi gradova A i B, dok se u sledećih m redova nalaze opisi puteva: u svakom redu se nalaze tri broja x, y i z, koji označavaju da postoji put između gradova x i y dužine z.

Izlaz.
U prvom redu izlaza se nalate dva realna broja, koji predstavljaju težine mreža koje sadrže grad A i grad B, redom. Zatim slede kompletni opisi mreža: u sledećem redu štampati broj gradova i broj puteva koji pripadaju mreži koja sadrži grad A, a zatim i sve puteve iz mreže; u sledećem redu štampati broj gradova i broj puteva koji pripadaju mreži koja sadrži grad B, a zatim i sve puteve iz mreže.

Ograničenja.

- $1 < n \leq 500$
- mapa gradova je uvek povezana
- dužine puteva su realni brojevi strogo veći od nule, a manji od 10.000

Primer 1.

podela.in	**podela.out**
6 8	5.0 2.0
1 6	3 2
1 2 3.0	1 2
1 3 3.0	2 5
2 4 2.0	3 2
2 5 2.0	3 4
3 4 1.0	4 6
4 5 3.0	
4 6 1.0	
5 6 4.0	

Ocenjivanje.
Broj bodova za svaki test primer se određuje na sledeći način. Neka je maksimalna dužina puteva u mrežama u vašem rešenju $VAL = \max(W_A, W_B)$, a OPT je minimalna vrednost među svim takmičarskim rešenjima, uključujući i komisijsko rešenje. Broj bodova koje dobijete se računa po formuli (zaokrugljivanje se vrši na najmanji ceo broj koji nije manji):

$$10^{11-10 \cdot VAL/OPT}.$$

Drugim rečima, ako je vaše rešenje identično sa najboljim dobijate 10 poena. U slučaju da je rešenje 10 ili više puta veće (lošije) od najboljeg rešenja - dobićete 1 poen. Ako je dužina puteva koju navedete u izlaznom fajlu različita od stvarne dužine puteva za izlazne mreže - dobijate 0 poena za taj test primer.

zadatak: Gramzivi farmeri

Poslednjih nekoliko meseci, naš stari farmer Branko, videvši da se dosta farmera preselilo u grad, odlučio je da podeli svoju zemlju ostalim farmerima koji će vredno da rade na njoj, i time spreči da njegovo mestašce nestane.
Farmer Gojko, koji je živeo nedaleko od Brankovog poseda, je upravo saznao za ovu akciju. Požurio je i on da dobije neko parče zemlje, ali već je bilo kasno. Branko je već podelio skoro celu zemlju. Ostalo je samo jedno malo parče zemlje.
Iznenađen ovakvim razvojem situacije, farmer Branko je izmislio igru, kojom će podeliti ostatak svoje zemlje, tako da što više ljudi dobije svoje parče. Branko je postavio n ($n \leq 2000$) stubića u zemlju. Onaj ko

prvi dođe ima pravo da uzme bilo koje četvorougaono parče zemlje, čija su temena tamo gde se nalaze stubići. Pravila igre za one koji stignu kasnije su veoma komplikovana. Na vašu sreću, farmer Gojko je stigao prvi, i sada treba da izabere četiri stubića. Pomozite mu da izabere najbolja 4 stubića, tako da dobije zemlju najveće površine.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **farmeri.in**.) U prvoj redu ulaza nalazi se broj n ($4 \leq n \leq 2.000$). U sledećih n redova nalaze se po dva mešovita (realna) broja - x i y koordinata jednog stubića (-200.000,0 $\leq x, y \leq$ 200.000,0). U 50% test primera, $n \leq 200$.

Izlaz.

(Izlazni podaci se ispisuju u datoteku **farmeri.out**.) U izlaznom fajlu trebate ispisati najveću površinu koju farmer Gojko može dobiti, zaokruženu na tri decimale.

Napomena. Nisu svi stubići kolinearni, tj rešenje će uvek biti veće od 0.

Primer 1.

farmeri.in	**farmeri.out**
6 0 0 0 1 1 0 1 1 0.5 0.5 1.5 1.5	1.500

Rešenje

Primetimo prvo da svaki četvorougao (i konveksni i ne konveksni) sadrži bar jednu svoju dijagonalu. Za neki par tačaka A i B, izračunaćemo tačke C i D, tako da je C sa leve, a D sa desne strane prave AB, i da trouglovi ABC i ABD imaju najveću moguću površinu. Tako da znamo, da od svih četvorouglova koji sadrže dijagonalu AB u sebi, najveća moguća površina je baš $P(ACBD)$. Ako izračunamo $P(ACBD)$ za svaki par tačaka A i B, najveći od tih brojeva je traženi rezultat.

Ako, za fiksirano A i B, prođemo kroz svaku tačku sa leve strane i tako izračunamo C, i slično za D, složenost ovog rešenja je $O(n^3)$, što donosi 50 poena.

Ako primetimo da tačka C je najudaljenija od segmenta AB, znamo da ona mora biti jedna od tačaka sa konveksnog omotača. Neka su E_1, E_2 ... E_k tačke na konveksnom omotaču sa leve strane segmenta AB, i označimo sa H_i rastojanje tačke E_i do prave AB. Znamo da je $P(ABE_i) = |AB| * Hi / 2$. Takođe znamo da je $H_1 < H_2 < ... < H_t > ... > H_k$, i $C = H_t$. Tako da možemo binarnom pretragom naći tačku H_t. Složenost ovog rešenja je $O(n^2 * log\ n)$, što donosi oko 80 poena.

Lako se vidi da samo jedna tačka najvećeg četvorougla ne mora biti na konveksnom omotaču (i to ako i samo ako se nalaze tačno tri tačke na konveksnom omotaču). Tako da možemo reći da to mora biti tačka A. Za svaku tačku A (i sa konvekson i one koje nisu), izračunaćemo najbolje tri tačke B, C i D na konveksnom omotaču. Neka su E_1, E_2 ... E_k tačke sa konveksnog omotača. Ako smo izračunali $C = E_v$ i $D = E_w$, za neku tačku $B = E_u$, gde se mogu nalaziti tačke C i D, za $B = E_{u+1}$? Ako povećamo v dok je $H_v < H_{v+1}$, dobijamo baš novu tačku $C = E_v$, i na slični način dobijemo i D. Koja je složenost ovog algoritma. Primetimo da za jednu tačku A, i v i w mogu svaki broj proći tačno jednom, tako da je složenost $O(n^2)$. Ovo rešenje donosi 100 poena.

Očigledno rešenje u $O(n^4)$ donosi 10-20 poena.

Neko odvojeno može uraditi slučaj ako na konveksnom omotaču imaju tačno tri tačke, ali samo jedan primer ima slučaj da su tačno tri tačke na omotaču. Ali je sa ovim zapažanjem rešenje malo kraće i jednostavnije. Ako postoje više od tri tačke na konveksnom omotaču, možemo posmatrati samo tačke sa omotača. Test primeri 5., 13. i 15. imaju malo tačaka na omotaču, pa ovo zapažanje donosi poene na ovim primerima.

Test primeri:

nom - broj na konveksnom omotaču

rbr.	n	nom
1.	4	3
2.	30	7
3.	50	50
4.	70	70
5.	100	14

rbr.	n	nom
11.	300	257
12.	500	428
13.	700	232
14.	700	694
15.	2000	22

6.	100	100
7.	130	130
8.	150	149
9.	200	166
10.	200	200

16.	2000	572
17.	1000	993
18.	1500	1480
19.	2000	816
20.	2000	1966

fajl: farmeri.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

const int maxn=2000;
const double eps=1e-8;
const double infinite=1e30;
const double pi=3.141592653589;

typedef struct
{ double x,y,angle;
} point;

int n,noh,hull[maxn];
point p[maxn];



inline int different(int i,int j,int k=-1,int l=-2)
{
    return ( (i!=j) && (i!=k) && (i!=l) && (j!=k) && (j!=l) && (k!=l) );
}



inline double area(int i,int j,int k,int l)
{
    double a=0;
    a+=p[i].x*p[j].y-p[j].x*p[i].y;
    a+=p[j].x*p[k].y-p[k].x*p[j].y;
    a+=p[k].x*p[l].y-p[l].x*p[k].y;
    a+=p[l].x*p[i].y-p[i].x*p[l].y;
    if (a<0) a=-a;
    return a;
}

inline double area(int i,int j,int k)
{
    double a=0;
    a+=p[i].x*p[j].y-p[j].x*p[i].y;
    a+=p[j].x*p[k].y-p[k].x*p[j].y;
    a+=p[k].x*p[i].y-p[i].x*p[k].y;
    return a;
}
int compare(const void *pa,const void *pb)
{
   point *a=(point*)pa,*b=(point*)pb;

  if ((a->angle)>(b->angle)) return 1;
  if ((a->angle)==(b->angle)) return 0;
  return -1;
}
```

```
int left(int a,int b,int j)
{
  double t=(p[b].x-p[a].x)*(p[j].y-p[a].y)-(p[b].y-p[a].y)*(p[j].x-p[a].x);
  if (t<-eps) return 1;
  if (t>eps) return -1;
  return 0;
}

int main()
{
    int ts=clock();

    freopen("farmeri.in","r",stdin);
    freopen("farmeri.out","w",stdout);

    int i,j,k,l,nextl;
    double result,inter,bestleft,bestright;
    point pom;

    scanf("%d",&n);
    for(i=0;i<n;i++)
       scanf("%lf%lf",&(p[i].x),&(p[i].y));

 //------------------------
    p[0].angle=0;
    k=0;

    for(i=1;i<n;i++)
    {
      p[i].angle=0;
      if ( (p[k].y>p[i].y) || (((p[i].y-p[k].y)<eps)&&(p[k].x<p[i].x)) )
       k=i;
    }

    pom=p[0];p[0]=p[k];p[k]=pom;

    for(i=1;i<n;i++)
    {
       p[i].x-=p[0].x;
       p[i].y-=p[0].y;
       p[i].angle=atan2(p[i].y,p[i].x);
       if (p[i].angle<0) p[i].angle+=2*pi;
    }

    p[0].x=0;p[0].y=0;p[0].angle=0;

    qsort(p,n,sizeof(point),compare);

    hull[0]=0;hull[1]=1;
    k=1;


    for(i=2;i<n;i++)
    {
       while ((k>=1) && (left(hull[k-1],hull[k],i)==1)) k--;
       k++;
       hull[k]=i;
    }

    while ((k>=1) && (left(hull[k-1],hull[k],hull[0])==1)) k--;

    k++;

    noh=k;

    result=-infinite;
    if (noh>3)
```

```
      for(i=0;i<noh;i++)
      {
         k=i+1;l=(i+3)%noh;
         for(j=i+2;j<noh;j++)
         {
            while (area(hull[i],hull[k],hull[j])<area(hull[i],hull[k+1],hull[j])) k++;
            nextl=(l+1)%noh;
            while (area(hull[i],hull[j],hull[l])<area(hull[i],hull[j],hull[nextl]))
               {l=nextl;nextl=(l+1)%noh;}
            inter=area(hull[i],hull[k],hull[j])+area(hull[i],hull[j],hull[l]);
            if (inter>result) result=inter;
         }
      }
   else
      for(i=0;i<n;i++)
         if (different(hull[0],hull[1],hull[2],i))
         {
            inter=area(hull[0],hull[1],hull[2],i);
            if (inter>result) result=inter;
            inter=area(hull[0],hull[1],i,hull[2]);
            if (inter>result) result=inter;
            inter=area(hull[0],i,hull[1],hull[2]);
            if (inter>result) result=inter;
         }

   printf("%.3lf\n",result / 2);

   return 0;
}
```