NAT: Nostalgic Alien Trespassers — TCR NWERC 2013

Filip Strömbäck, Magnus Selin, Carl Einarson

November 24, 2013

Contents

1 Environment

1.1 Template

```cpp
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <string>
#include <bitset>
#include <algorithm>
#include <cstring>

using namespace std;

#define rep(i, a, b) for(int i = (a); i < int(b); ++i)
#define trav(it, v) for(typeof((v).begin()) it = (v).begin(); it != (v).end(); ++it)

typedef double fl;
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;


bool solve(){

  return true;
}

int main(){
  int tc=1; //scanf("%d", &tc);
  rep(i, 0, tc) solve();

  return 0;
}
```

2 Data Structures

2.1 Union Find

```cpp
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;

int find(int * root, int x){
  if (root[x] == x) return x;
  root[x] = find(root, root[x]);
  return root[x];
}

void uni(int * root, int * deep, int x, int y){
  int a = find(root, x);
  int b = find(root, y);
  root[a] = b;
}

bool issame(int * root, int a, int b){
  return(find(root, a) == find(root, b));
}

int main(){
  int n, no; scanf("%d%d", &n, &no);
  int root[n];
  for(int i = 0; i < n; i++){
    root[i] = i;
  }

  for(int i = 0; i < no; i++){
    char op; int a, b;
    scanf("%*[ \n\t]%c", &op);
    scanf("%d%d", &a, &b);
    if(op == '?'){
      if(issame(root, a, b)) printf("yes\n");
      else           printf("no\n");
    }
    if(op == '=')
      uni(root, deep, a, b);
  }
}
```

2.2 Fenwick Tree

```cpp
#include <iostream>
#include <stdio.h>
#include <vector>

using namespace std;


typedef long long int lli;
typedef vector<lli> vi;


#define last_dig(x) (x & (-x))

void fenwick_create(vi &t, lli n){
  t.assign(n + 1, 0);
}
lli fenwick_read(const vi &t, lli b){
  lli sum = 0;
  while(b > 0){
    sum += t[b];
    b -= last_dig(b);
  }
  return sum;
}

void fenwick_update(vi &t, lli k, lli v){
  while(k <= (lli)t.size()){
    t[k] += v;
    k += last_dig(k);
  }
}

int main(){
  lli N, Q; scanf("%lld%lld", &N, &Q);
  vi ft; fenwick_create(ft, N);

  char op; lli a, b;
  for(lli i = 0; i < Q; i++){
    scanf("%*[ \n\t]%c", &op);
    switch (op){
      case '+':
      scanf("%lld%lld", &a, &b);
      fenwick_update(ft, a+1, b);
      break;

      case '?':
      scanf("%lld", &a);
      printf("%lld\n", fenwick_read(ft, a));
      break;
    }
  }

  return 0;
}
```

3 Numerical

3.1 General Utils

```
// Externa funktioner:
// OutIt copy(InIt first, InIt last, OutIt x);
// Returvrde: x + N, utiteratorn efter sista elementet.
// void fill(FwdIt first, FwdIt last, const T& x);
// bool next_permutation(BidIt first, BidIt last, Pred pr); // O(n)
// Funktion: Permuterar mngden till nsta variant enligt lexikal ordning.
// Kommentar: Brja med en sorterad mngd. Tar ej med dubbletter.
// void nth_element(RanIt fi,RanIt nth, RanIt la [,Pred pr]);
// Funktion: Delar upp elementen s att *nth r strre
// eller lika alla element i [first, nth [
// och *nth r mindre eller lika alla element i ]nth, last[.
// Komplexitet: O(n) i medeltal
// BidIt partition(BidIt first, BidIt last, Pred pr); // O(n)
// Returvrde: first + k, iteratorn f r frsta elementet i andra intervallet.
// Funktion: Delar upp elementen s att pr() r sant resp. falskt f r alla
// element i intervallen [0, k[ respektive [k, n[.
// FwdIt stable_partition(FwdIt first, FwdIt last, Pred pr);
// Kommentar: Samma som ovan men bevarar inbrdes ordning.
// void sort(RanIt first, RanIt last [, Pred pr]); // O(n*log(n))
// Kommentar: F r list<> anvnd den interna funktionen l.sort().
// void stable_sort(RanIt first, RanIt last [, Pred pr]);
// Kommentar: Samma som ovan men bevarar inbrdes ordning.
// FwdIt unique(FwdIt first, FwdIt last [, Pred pr]); // O(n)
// Returvrde: first + k, iteratorn efter sista elementet i mngden.
// Funktion: Delar upp elementen s att inga p varandra fljande
// element i [0, k) r lika.
// Elementen i [k, last[ r odefinierade.
// Kommentar: F r list<> anvnd den interna funktionen l.unique().
//
// Skning i sorterade mngder
// Fljande funktioner har tidskomplexiteten O(log(n)) med undantaget O(n)
// f r list. De tre sista samt funktion find() finns internt i map
// och set. Returnerar c.end() om inget passande element hittas.
// bool binary_search(FwdIt first, FwdIt last, T& x [, Pred pr]);
// Returvrde: true om x finns, annars false.
// FwdIt lower_bound(FwdIt first, FwdIt last, T& x [, Pred pr]);
// Returvrde: first + k, frsta positionen som x kan sttas
// in p s att sorteringen, dvs. varje element i [0, k[ r mindre n x.
// FwdIt upper_bound(FwdIt first, FwdIt last, T& x [, Pred pr]);
// Returvrde: first + k, sista positionen som x kan sttas
// in p s att sorteringen bibehlls, dvs. varje element i
// ]k, n[ r strre n x.
// pair<It, It> equal_range(It first, It last, T& x [,Pred pr]);
// Returvrde: pair(lower_bound(fi, la, x),upper_bound(fi, la, x))

// Binary search (from Wikipedia)
// The indices are _inclusive_.
int binary_search(T *a, int key, int min, int max) {
 while (min < max) {
  int mid = (min + max) / 2; // midpoint (min, max)

  // assert(mid < max)

  // The condition can be replaced by some other function
  // depending on mid, eg worksFor(mid + 1) to search for
  // the last index "worksFor" returns true for.
  if (a[mid] < key) {
   min = mid + 1;
  } else {
   max = mid;
  }
 }

 // Equality test, can be skipped when looking for a specific value
 if ((max == min) && (a[min] == key))
  return min;
 else
  return NOT_FOUND;
}

// Fenwick tree:
```

3.2 Rational Numbers Class

```
#include <stdio.h>

using namespace std;

class Q{
private:
  long long int p, q;
  long long int gcd(long long int a, long long int b) {
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    if (0 == b) return a;
    else return gcd(b, a % b);
  }
public:
  Q(){}
  Q(long long int a, long long int b){
    p = a; q = b;
    if(q < 0){p = -p; q = -q;}
    if (p == 0) q = 1;
    if (q == 0){
      printf("ERR: den = 0!\n");
      q = 1;
    }
    long long int g = gcd(p, q);
    p /= g; q /= g;
  }
```

```
27
28    Q operator + (Q a){
29      Q b = * this;
30      Q res = Q((a.p * b.q + b.p * a.q), (
        a.q * b.q));
31      return res;
32    }
33
34    Q operator - (Q a){
35      Q b = * this;
36      Q res;
37      if(a==b) res = Q(0,0);
38      else res = Q((b.p * a.q - a.p * b.q)
        , (a.q * b.q));
39      return res;
40    }
41
42    Q operator * (Q a){
43      Q b = * this;
44      Q res = Q(a.p * b.p, a.q * b.q);
45      return res;
46    }
47
48    Q operator / (Q a){
49      Q b = * this;
50      Q res = Q(b.p * a.q, b.q * a.p);
51      return res;
52    }
53
54    bool operator == (Q a){
55      Q f = * this;
56      Q s = Q(a.p, a.q);
57      return (f.p == s.p and f.q == s.q);
58    }
59
60    void operator = (Q a){
61      this->p = a.p;
62      this->q = a.q;
63    }
64
65    void print(){
66      printf("%lld / %lld\n", p, q);
67    }
68  };
69
70  int main(){
71    int n; scanf("%d", &n);
72    for(int i = 0; i < n; i++){
73      int tp, tn;
74      scanf("%d%d", &tp, &tn); Q a = Q(tp,
          tn);
75
76      char t=' '; while (t == ' ') scanf("
        %c", &t);
77
78      scanf("%d%d", &tp, &tn); Q b = Q(tp,
          tn);
79
80      switch(t){
81        case '+': (a+b).print(); break;
82        case '-': (a-b).print(); break;
83        case '*': (a*b).print(); break;
84        case '/': (a/b).print(); break;
85      }
86    }
87
88    return 0;
89  }
```

3.3 Binary Search

```
1  // Example usage of the bsearch
2  #include <cstdlib>
3  #include <cstdio>
4
5  int check(const void *key, const void *
       elem) {
6   int k = (int)key;
7   int e = (int)elem;
8   printf("Comparing %d with %d\n", k, e);
9
10  if (k == e) return 0;
11  if (k < e) return -1;
12  return 1;
13 }
14
15 int main() {
16  int found = (int)bsearch((const void *)
       10, 0, 100, 1, &check);
17
18  printf("I found: %d\n", found);
19
20  return 0;
21 }
```

3.4 De Brujin

```
1
2  #include <iostream>
3  #include <vector>
4  #include <cmath>
5
6  using namespace std;
7  vector<bool> seq;
8  vector<bool> a;
9  int n, k;
10
11 void db(int t, int p){
12   if (t > n){
13     if (n % p == 0)
14       for (int j = 1; j < p + 1; j++)
15         seq.push_back(a[j]);
16   }
17   else{
18     a[t] = a[t - p];
19     db(t + 1, p);
20     for (int j = a[t - p] + 1; j < 2; j
       ++){
21       a[t] = j;
22       db(t + 1, t);
23     }
24   }
25 }
26
27 int de_bruijn(){
28   for(int i = 0; i < n; i++)
29     a.push_back(0);
30   db(1, 1);
31
32   int sum = 0;
33   for(int i = 0; i < n; i++){
34     sum += seq[(k+i) % (int)pow((double)
       2, n)] * pow((double)2, n-i-1);
35   }
36   cout << sum << '\n';
37 }
38
39 int main(){
40   int tc;
41   cin >> tc;
42   for(int we = 0; we < tc; we++){
43     cin >> n >> k;
44     a.clear(); seq.clear();
45     de_bruijn();
46   }
47 }
```

3.5 Prime Generator

```
1  #include <cstdio>
2
```

```
int prime[664579];
int numprimes;

void calcprimes(int maxn){
  prime[0] = 2; numprimes = 1; prime[
    numprimes] = 46340; // 0xb504*0xb504
    = 0x7FFEA810
  for(int n = 3; n < maxn; n += 2) {
    for(int i = 1; prime[i]*prime[i] <=
    n; ++i) {
      if(n % prime[i] == 0) goto
    not_prime;
    }
    prime[numprimes++] = n; prime[
    numprimes] = 46340; // 0xb504*0xb504
    = 0x7FFEA810
not_prime:
    ;
  }
}

int main(){
  calcprimes(10000000);
  for(int i = 0; i < 664579; i++) printf
    ("%d\n", prime[i]);
}
```

3.6 Factorisation

```
int factor[1000000];
int numf[1000000];
int numfactors;

void calcfactors(int n){
  numfactors = 0;
  for(int i = 0; n > 1; ++i){
    if(n % prime[i] == 0){
      factor[numfactors] = prime[i];
      numf[numfactors] = 0;
      do {
        numf[numfactors]++;
        n /= prime[i];
      } while(n % prime[i] == 0);
    numfactors++;
    }
  }
}
```

3.7 Chinese remainder

```
#pragma once
#include "number.h"
#include "modular.h"

/*
 * Solve the problem:
 * x = a (mod m)
 * x = b (mod n)
 * ...
 * Where a, b, m and n are valid modular
     numbers. A NoSolution is thrown if
 * there is no solution.
 */
Modular<int64> remainder(const vector<
    Modular<int64> > &eq) {
  typedef Modular<int64> M;

  M last = eq[0];

  for (nat i = 1; i < eq.size(); i++) {
    M curr = eq[i];

    /* Given the two currently first
    equations: */
    /* r = a (mod m) = b (mod n) */
    /* r = a + x*m = b + y*n */
    /* a - b = y*n - x*m */
    Dioph<int64> sol = diophantine(-last
    .modulo, curr.modulo, last.value -
    curr.value);
    int64 mod = lcm(last.modulo, curr.
    modulo);

    last = M(last.value, mod) + M(last.
    modulo, mod) * M(sol.x, mod);
  }

  return last;
}
```

3.8 Diophantine equations

```
/*
 * Find the greatest common divider of
    two positive numbers. T is an
    integral type.
 */
template <class T>
inline T gcd(T a, T b) {
  if (a < b) swap(a, b);
  while (b != 0) {
    T r = a % b;

    a = b;
    b = r;
  }

  return a;
}


/*
 * Find the least common multiple of two
    positive numbers. T is an integral
    type.
 */
template <class T>
inline T lcm(T a, T b) {
  return a * (b / gcd(a, b));
}

/*
 * Do the euclid's algorithm both
    forward and reverse, to find
    solutions to
 * diophantine equations. T shall be
    signed, as this algorithm will use
    negative
 * numbers internally.
 * Generates one integer solution to the
    equation a*x + b*y = sum, throws
    NoSolution
 * if none exists. "a" and "b" are
    assumed to be positive.
 */
template <class T>
struct Dioph {
  T x, y;

  Dioph(const T &x = T(), const T &y = T
    ()) : x(x), y(y) {}
};

template <class T>
ostream &operator <<(ostream &to, const
    Dioph<T> &f) {
  return to << "{ x=" << f.x << ", y="
    << f.y << "}";
}
```

```
template <class T>
Dioph<T> diophantine(T a, T b, T sum) {

  /* If b == 0, it is easy: */
  if (b == T()) {
    /* x = sum / a, iff sum % a != 0 */
    if (sum % a == 0)
      return Dioph<T>(sum / a, 0);
    else
      throw NoSolution();
  } else if (a == T()) {
    if (sum % b == 0)
      return Dioph<T>(sum / b, 0);
    else
      throw NoSolution();
  }

  /* Compute the form: a = z * b + r */
  /* We can also say: r = a - z*b */
  T z = a / b;
  T r = a % b;

  /* Solve the new equation, b*x + r*y =
     sum */
  Dioph<T> sol = diophantine(b, r, sum);

  /* We know: r = a - z*b */
  /* Substitution gives: b*x + (a - z*b)
    *y = sum */
  /* Which equals: b*x + a*y - z*b*y =
    sum <=> a*y + b*(x - z*y) */
  return Dioph<T>(sol.y, sol.x - sol.y*z
    );
}
```

4 Graphs

4.1 Single Source Shortest Path

Dijkstra's algorithm
Time Complexity $O(E + V \log V)$

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

#define INF 100000000

using namespace std;

typedef pair<int, int> ii;

template<class T>

class comp{
public:
  int operator()(const pair<int, T> & a,
     const pair<int, T> & b){return (a.
    second > b.second);}
};

template<class T>
vector<T> dijkstras(vector<pair<int, T>
    > G[], int n, int e, int s){
  priority_queue<pair<int, T> , vector<
    pair<int, T> >, comp> Q;

  vector<T> c; for(int i = 0; i < n; i
    ++) c.push_back(INF); c[s] = 0;
  vector<int> p; for(int i = 0; i < n; i
    ++) p.push_back(-1);

  Q.push(pair<int, T>(s, c[s]));
  int u, sz, v; T w;
  while(!Q.empty()){

    u = Q.top().first; Q.pop();
    sz = G[u].size();
    for(int i = 0; i < sz; i++){
      v = G[u][i].first;
      w = G[u][i].second;
      if( c[v] > c[u] + w ){
        c[v] = c[u] + w;
        p[v] = u;
        Q.push(pair<int, T>(v, c[v]));
      }
    }
  }

  //printf("Path to follow: ");
  //for(int i = 0; i < n; i++) printf("%
    d ", p[i]);
  //printf("\n");

  return c;
}

int main(){
  int n, e, q, s;
  scanf("%d%d%d%d", &n, &e, &q, &s);
  while(n!=0 or e!=0 or q!=0 or s!=0){
    vector<ii> G[n];
    for(int i = 0; i < e; i++){
      int f, t, w;
      scanf("%d%d%d", &f, &t, &w);
      G[f].push_back(ii(t, w));
    }
    vector<int> c = dijkstras(G, n, e, s
    );

    for(int i = 0; i < q; i++) {
      int d; scanf("%d", &d);
      if(c[d] == INF)  printf("
    Impossible\n");
      else        printf("%d\n", c[d]);
    }
    printf("\n");

    scanf("%d%d%d%d", &n, &e, &q, &s);
  }

  return 0;
}
```

4.2 Single Source Shortest Path Time Table

Single Source Shortest Path Time Table (Dijkstra)
Time Complexity $O(E + V \log V)$

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

#define INF 100000000

using namespace std;

struct A{
  A(int a, int b, int c){t0=a; tn = b; w
    = c;}
  int t0, tn, w;
};

typedef pair<int, int> ii;
typedef pair<int, A> iA;

class comp{
public:
  int operator()(const ii& a, const ii&
    b){return (a.second > b.second);}
};

```

```
vector<int> dijkstras(vector<iA> G[], int n, int e, int s){
  priority_queue<ii, vector<ii>, comp> Q;

  vector<int> c; for(int i = 0; i < n; i++) c.push_back(INF); c[s] = 0;
  vector<int> p; for(int i = 0; i < n; i++) p.push_back(-1);

  Q.push(ii(s, c[s]));
  int u, sz, v, t0, tn, w, wt;
  while(!Q.empty()){

    u = Q.top().first; Q.pop();
    sz = G[u].size();
    for(int i = 0; i < sz; i++){
      v = G[u][i].first;
      tn = G[u][i].second.tn;
      t0 = G[u][i].second.t0;
      w = G[u][i].second.w;

      wt = t0 - c[u];
      if (wt < 0 and tn == 0) continue;
      while(wt < 0) wt+=tn;

      if( c[v] > c[u] + w + wt){
        c[v] = c[u] + w + wt;
        p[v] = u;
        Q.push(ii(v, c[v]));
      }
    }
  }

  //printf("Path to follow: ");
  //for(int i = 0; i < n; i++) printf("%d ", p[i]);
  //printf("\n");

  return c;
}

int main(){
  int n, e, q, s;
  scanf("%d%d%d%d", &n, &e, &q, &s);
  while(n!=0 or e!=0 or q!=0 or s!=0){
    vector<iA> G[n];
    for(int i = 0; i < e; i++){
      int f, t, t0, tn, w;
      scanf("%d%d%d%d%d", &f, &t, &t0, &tn, &w);
      G[f].push_back(iA(t, A(t0, tn, w)));
    }
    vector<int> c = dijkstras(G, n, e, s);

    for(int i = 0; i < q; i++) {
      int d; scanf("%d", &d);
      if(c[d] == INF)  printf("Impossible\n");
      else        printf("%d\n", c[d]);
    }
    printf("\n");

    scanf("%d%d%d%d", &n, &e, &q, &s);
  }

  return 0;
}
```

4.3 All Pairs Shortest Path

Floyd Warshall's algorithm. Assign nodes which are part of a negative cycle to minus infinity.
Time Complexity $O(V^3)$

```
// All pairs shortest path (Floyd Warshall). Assign nodes which are part of a
// negative cycle to minus infinity.

#include <stdio.h>
#include <iostream>
#include <vector>
#include <algorithm>

#define INF 1000000000
using namespace std;

template<class T>
vector< vector <T> > floyd_warshall(vector< vector<T> > d){
  int n = d.size();
  for(int i = 0; i < n; i++) d[i][i] = 0;

  for (int k = 0; k < n; k++)
   for (int i = 0; i < n; i++)
     for (int j = 0; j < n; j++)
       if (d[i][k] != INF and d[k][j] != INF)
         d[i][j] = min(d[i][j], d[i][k]+d[k][j]);

  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      for(int k = 0; d[i][j] != -INF && k < n; k++)
        if(d[i][k] != INF && d[k][j] != INF && d[k][k] < 0)
          d[i][j] = -INF;

  return d;
}

int main(){
  int n, m, q; scanf("%d%d%d", &n, &m, &q);
  while(n!=0 or m!=0 or q!=0){
    vector< vector<int> > d;
    d.resize(n);
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        d[i].push_back(INF);

    for(int i = 0; i < m; i++){
      int f, t, w; scanf("%d%d%d", &f, &t, &w);
      d[f][t] = min(w, d[f][t]);
    }

    d = floyd_warshall(d, n);
    for(int i = 0; i < q; i++){
      int f, t; scanf("%d%d", &f, &t);
      if(d[f][t] == INF)      printf("Impossible\n");
      else if(d[f][t] == -INF)  printf("-Infinity\n");
      else              printf("%d\n", d[f][t]);
    }
    printf("\n");
    scanf("%d%d%d", &n, &m, &q);
  }
  return 0;
}
```

4.4 Minimum Spanning Tree

Time Complexity $O(E + V \log V)$

```cpp
#include <stdio.h>
#include <algorithm>
#include <vector>

using namespace std;

struct AnsEdge{
  int f, t;
  bool operator<(const AnsEdge& oth)
    const{
    if(f == oth.f)
      return(t < oth.t);
    return(f < oth.f);
  }

  AnsEdge(){};
  AnsEdge(int a, int b){f = a; t = b;};
};
struct Tree{
  int w;
  bool complete;
  std::vector<AnsEdge> e;
  Tree(){
    w = 0;
    complete = true;
  }
};

struct Vertex{
  Vertex *p;
  Vertex *root(){
    if(p->p != p)
      p = p->root();
    return p;
  }
};
struct Edge{
  int f, t, w;

  bool operator<(const Edge& oth) const{
    if (w == oth.w)
      return(t < oth.t);
    return(w < oth.w);
  }
};


Tree kruskal(Vertex * v, Edge * e, int
    numv, int nume){
  Tree ans;
  int sum = 0;

  for(int i = 0; i < numv; ++i){
    v[i].p = &v[i];
  }

  sort(&e[0], &e[nume]);

  for(int i = 0; i < nume; ++i){
    if(v[e[i].f].root() != v[e[i].t].
    root()){
      v[e[i].t].root()->p = v[e[i].f].
    root();
      ans.w += e[i].w;

      if(e[i].t < e[i].f) ans.e.
    push_back(AnsEdge(e[i].t, e[i].f));
      else          ans.e.push_back(
    AnsEdge(e[i].f, e[i].t));
    }
  }

  Vertex * p = v[0].root();
  for(int i = 0; i < numv; ++i)
    if(p != v[i].root()){
      ans.complete = false;
      break;
    }

  sort(ans.e.begin(), ans.e.end());

  return ans;
}

int main(){
  int n, m; scanf("%d%d", &n, &m);
  while(n or m){
    Vertex v[n];
    Edge e[m];

    for(int i = 0; i < m; i++){
      int f, t;
      scanf("%d%d%d", &f, &t, &e[i].w);
      e[i].f = f;
      e[i].t = t;
    }

    Tree ans = mst(v, e, n, m);

    if(ans.complete){
      printf("%d\n", ans.w);
      for(int i = 0; i < ans.e.size(); i
    ++){
        printf("%d %d\n", ans.e[i].f,
    ans.e[i].t);
      }
    }
    else printf("Impossible\n");

    scanf("%d%d", &n, &m);
  }

  return 0;
}
```

4.5 Maximum Flow

Edmonds Karp's Maximum Flow Algorithm
Input: Adjacency Matrix (res)
Output: Maximum Flow
Time Complexity: $O(VE^2)$

```cpp
int res[MAX_V][MAX_V], mf, f, s, t;
vi p;

void augment(int v, int minEdge) {
  if(v == s){f = minEdge; return;}
  else if(p[v]  != -1){augment(p[v], min
    (minEdge, res[v][p[v]]));
            res[p[v]][v] -= f; res[v][p[
    v]] += f; }
}

int solve(){
  mf = 0; // Max Flow

  while(1){
    f = 0;
    vi dist(MAX_V, INF); dist[s] = 0;
    queue<int> q; q.push(s);
    p.assign(MAX_V, -1);
    while(!q.empty()){
      int u = q.front(); q.pop();
      if(u == t) break;
      for(int v = 0; v < MAX_V; v++)
        if (res[u][v] > 0 && dist[v] ==
    INF)
          dist[v] = dist[u] + 1, q.push(
    v), p[v] = u;
```

```
    }
    augument(t, INF);
    if(f == 0) break;
    mf += f;
  }

  printf("%d\n", mf);
}
```

4.6 Euler Tour

Time Complexity $O(E+V)$

```
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <list>

typedef vector<int> vi;

using namespace std;

list<int> cyc;

void euler_tour(list<int>::iterator i,
    int u) {
  for(int j = 0; j < (int)AdjList[u].
    size(); j++){
    ii v = AdjList[u][j];
    if (v.second){
      v.second = 0;
      for(int k = 0; k < (int)AdjList[u
    ].size(); k++){
        ii uu = AdjList[v.first][k];
        if(uu.first == u && uu.second) {
    uu.second = 0; break;}
      }
      euler_tour(cyc.insert(i, u), v.
    first)
    }
  }
}

int main(){
  cyc.clear();
  euler_tour(cyc.begin(), A);
  for(list<int>::iterator it = cyc.begin
    (); it != cyc.end(); it++;)
    printf("%d\n", *it);
}
```

4.7 Bipartite Matching

```
/* Name: Bipartite DFS
 * Description: Simple bipartite
    matching.
 * Slower than HopcroftKarp but shorter.
 * Graph g should be a list of
    neighbours
 * of the left partition.
 * n is the size of the left partition
 * and m is the size of the right
    partition.
 * Ifyou want to get the matched pairs,
 * \lstinline|match[i]| contains match
    for vertex i on
 * the right side or -1 if it's not
    matched.
 * Time: \(\mathcal{O}(EV)\)
 * Usage example:
 * \begin{lstlisting}[frame=none,
    aboveskip=-0.6cm, ]
 * Graph left(n);
 * trav(it, edges){
 *  l[it->left].push_back(it->right);
 * }
 * dfs_matching(left, size_left,
    size_right);
 * \end{lstlisting}
 * Source: KACTL */

typedef vector<vector<int> > Graph;

vector<int> match;
vector<bool> visited;
template<class G>
bool find(int j, G &g) {
 if (match[j] == -1) return true;
 visited[j] = true; int di = match[j];
 trav(e, g[di])
 if (!visited[*e] && find(*e, g)) {
  match[*e] = di;
  match[j] = -1;
  return true;
 }
 return false;
}
int dfs_matching(Graph &g, int n, int m)
    {
 match.assign(m, -1);
 rep(i,0,n) {
  visited.assign(m, false);
  trav(j,g[i])
  if (find(*j, g)) {
   match[*j] = i;
   break;
  }
 }
 return m - count(match.begin(), match.
    end(), -1);
}
```

4.8 Strongly Connected Components

```
/* Name: Strongly Connected Components -
    Double DFS
 * Description: Untested SCC algorithm.
    Calculates a new graph where all
    strongly connected components are
    merged. Does not require the graph
    to be connected.
 * Source: Fredrik Svensson - 2009 */
struct vertex
{
        vector<vertex*> from, to;
        bool visited;
};
vector<vertex> v;
vector<vector<vertex*> > res;

vector<vertex*> sorted;
vector<vertex*>::reverse_iterator
    visitIt;
vector<vertex*>* curRes;

void dfs(vertex* p)
{
        if(p->visited) return;
        p->visited = true;
        if(curRes) curRes->push_back(p);
        for(vector<vertex*>::iterator it
            = p->to.begin();
                it != p->to.end(); ++it)
                dfs(*it);
        *(visitIt++) = p;
}

void run()
{
        sorted.resize(v.size());
        visitIt = sorted.rbegin();
```

```
        for(vector<vertex>::it it = v.
            begin();
    it != v.end(); ++it)
                it->visited = false;
        for(vector<vertex>::it it = v.
            begin();
    it != v.end(); ++it)
                dfs(&(*it));
        for(vector<vertex>::it it = v.
            begin();
    it != v.end(); ++it)
        {
                it->visited = false;
                it->from.swap(it->to);
        }
        for(vector<vertex>::iterator it
            = sorted.begin();
                it != sorted.end(); ++it
                    )
                if(!(*it)->visited)
                {
                        curRes = &(*res.
                            insert(res.
                            end()));
                        dfs(&(*it));
                }
}
```

5 String processing

5.1 STL

```
#include <string>

std::size_t found = str.find(str2);
if (found!=std::string::npos)
  std::cout << "first found at: " <<
    found << '\n';

str.replace(str.find(str2),str2.length()
    ,"new word");
```

5.2 String Matching

```
// Knuth Morris Prat : Search for a
    string in another one
// Alternative STL algorithms : strstr
    in <ctring> find in <string>
// Time complexity : O(n)

#include <cstdio>
#include <cstring>

#define MAX_N 100010

char T[MAX_N], P[MAX_N];  // T = text, P
    = pattern
int b[MAX_N], n, m;      // b = back
    table, n = length of T, m = length of
    P

void kmpPreprocess() {
  int i = 0, j = -1; b[0] = -1;
  while (i < m){
    while(j >= 0 && P[i] != P[j]) j = b[
    j];
    i++; j++;
    b[i] = j;
  }
}

void kmpSearch() {
  int i = 0, j = 0;
  while(i < n){
    while(j >= 0 && T[i] != P[j]) j = b[
    j];
    i++; j++;
    if(j==m){
      printf("P is found at index %d in
    T\n", i - j);
      j = b[j];
    }
  }
}

int main(){
  strcpy(T, "asdhasdhejasdasdhejasdasd")
    ;
  strcpy(P, "hej");

  n = 25; m = 3;

  kmpPreprocess();
  kmpSearch();

  return 0;
}
```

5.3 String Multimatching

6 Geometry

6.1 Points Class

```
#include <cmath>

template<class T>
class Vector{
public:

  T x, y;
  Vector(){};
  Vector(T a, T b){x = a; y = b};

  T abs(){return sqrt(x*x+y*y);}
  Vector operator* (T oth){ return
    Vector(x*oth, y*oth); }
  Vector operator/ (T oth){ return
    Vector(x/oth, y/oth); }

  Vector operator+ (Vector oth){ return
    Vector(x+oth.x, y+oth.y); }
  Vector operator- (Vector oth){ return
    Vector(x+oth.x, y+oth.y); }
  T operator* (Vector oth){ return x*oth
    .x + y*oth.y; }
  Vector operator/ (Vector oth){ return
    Vector(x*oth.y-oth.x*y)}
};
```

6.2 Lines

```
/*
 * Simple line segment representation.
 */
template <class T>
class Line2 {
public:
 typedef T Type;
 typedef Point2<T> Point;

 Point from, to;

 Line2() {}
 Line2(const Point &from, const Point &
    to) : from(from), to(to) {}

 /*
  * The line's direction, measured from
    "from".
```

```
   */
  Point dir() const { return to - from; }

  /*
   * The line's angle. This angle is the
     angle of "dir", relative the vector
     (1, 0).
   */
  double angle() const {
   Point pt = dir();
   return atan2(pt.y, pt.x);
  }

  /*
   * Orthogonal projection of a point
     onto this line. Returns where on the
     line the point
   * is located, 0 = from, 1 = to. May be
      outside this range.
   */
  double project(Point pt) const {
   return double((pt - from) | dir()) /
     lengthSq(dir());
  }

  /*
   * Orthogonal projection of the two
     points of another line onto this line
     .
   */
  pair<double, double> project(Line2<T>
     line) const {
   return make_pair(project(line.from),
     project(line.to));
  }
 };

 /*
  * Point - line-segment intersection.
  */
 template <class T>
 bool intersects(Point2<T> point, Line2<T
    > line) {
  if (line.from == line.to) {
   return line.from == point;
  } else {
   Point2<T> dir = line.to - line.from;
   Point2<T> pt = point - line.from;
   Point2<T> pt2 = point - line.to;

   return (dir % pt) == 0 && (dir | pt)
     >= 0 && (-dir | pt2) >= 0;
  }
 }

 /*
  * Distance between a point and a line
     segment.
  */
 template <class T>
 double distance(Point2<T> point, Line2<T
    > line) {
  if (line.from == line.to)
   return distance(line.from, point);

  /* Project pt unto the line. */
  double projection = line.project(point)
     ;

  if (projection <= 0)
   return distance(line.from, point);
  else if (projection >= 1)
   return distance(line.to, point);

  Point2<double> proj = Point2<double>(
     line.dir()) * projection;
  return length(proj - Point2<double>(
     point - line.from));
 }


 /*
  * line-segment - line-segment
     intersection. Reports wether the
     lines intersect
  * or not.
  */
 template <class T>
 bool intersects(Line2<T> a, Line2<T> b)
    {
  typedef Point2<T> Pt;

  Pt aDir = a.dir();
  Pt bDir = b.dir();
  Pt delta = a.from - b.from;

  if (aDir == Pt() && bDir == Pt()) {
   return a.from == b.from;
  } else if (aDir == Pt()) {
   /* We can manage if bDir == 0 */
   return intersects(b, a);
  }

  T cross = aDir % bDir;
  if (cross == 0) {
   /* Parallel lines, project the two
     points in B onto A and see where they
      are located. */

   /* Check if the lines are on the same
     line. */
   if (delta % aDir != 0)
    return false;

   double from = a.project(b.from);
   double to = a.project(b.to);

   double low = min(from, to);
   double high = max(from, to);

   return low <= 1 && high >= 0;
  } else {
   /* The lines intersect, where? */

   /* Distance along bDir * cross */
   T u = aDir % delta;
   /* Distance along aDir * cross */
   T v = bDir % delta;

   return between(T(0), u, cross) &&
     between(T(0), v, cross);
  }
 }

 /*
  * Helper to intersection() below. Adds
     two points, only if they are
     distinct.
  */
 template <class T>
 inline void addPoints(vector<Point2<
    double> > &result, const Point2<T> &a
    , const Point2<T> &b) {
  result << Point2<double>(a);
  if (a != b) result << Point2<double>(b)
    ;
 }

 /*
```

```
 * Compute the intersection between two line segments.
 * Returns zero, one or two points depending on if there is an intersection
 * and if that intersections is a point or a line.
 * If there are two points, this always returns the intersection points
 * sorted along the direction of line a.
 */
template <class T>
vector<Point2<double> > intersection(Line2<T> a, Line2<T> b) {
 typedef Point2<T> Pt;
 typedef Point2<double> PtD;

 vector<PtD> result;

 Pt aDir = a.dir(), bDir = b.dir();
 Pt delta = a.from - b.from;

 if (aDir == Pt() && bDir == Pt()) {
  if (a.from == b.from)
   result << PtD(a.from);
  return result;
 } else if (aDir == Pt()) {
  return intersection(b, a);
 }

 T cross = aDir % bDir;
 if (cross == 0) {
  /* The lines are parallel. */

  /* Are they in the same line? */
  /* See if vectors from a.from to b.from is parallel with a.from to a.to */
  if (delta % aDir == 0) {
   double low = a.project(b.from);
   double high = a.project(b.to);

   Pt lowPt(b.from);
   Pt highPt(b.to);

   if (low > high) {
        swap(low, high);
        swap(lowPt, highPt);
   }

   if (b.from == b.to) {
        if (low >= 0 && low <= 1)
         result << PtD(b.from);
   } else {
        if (low < 0 && high >= 0 && high <= 1)
         addPoints(result, a.from, highPt);
        else if (low < 0 && high >= 1)
         addPoints(result, a.from, a.to);
        else if (low >= 0 && high <= 1)
         addPoints(result, lowPt, highPt);
        else if (low >= 0 && low <= 1)
         addPoints(result, lowPt, a.to);
   }
  }
 } else {
  /* Distance along bDir * cross */
  T u = aDir % delta;
  /* Distance along aDir * cross */
  T v = bDir % delta;

  if (between(T(0), u, cross) && between(T(0), v, cross)) {
   result << PtD(a.from) + PtD(aDir) * v / double(cross);
  }
 }

 return result;
}

/*
 * Distance between two line segments.
 */
template <class T>
double distance(Line2<T> a, Line2<T> b) {
 if (intersects(a, b))
  return 0.0;

 /*
  * Now that we know that the lines are not intersecting, we can easily
  * compute the shortest distance between all endpoints on the lines
  * to the other line.
  */
 double shortest =
  min(min(distance(a.from, b), distance(a.to, b)),
      min(distance(b.from, a), distance(b.to, a)));

 return shortest;
}
```

6.3 Matrix Class

```
/* Description: Untested matrix implementation
 * Source: Benjamin Ingberg */
template<typename T>
struct Matrix {
 typedef Matrix<T> const & In;
 typedef Matrix<T> M;

 int r, c; // rows columns
 vector<T> data;
 Matrix(int r_, int c_, T v = T()) : r(r_), c(c_), data(r_*c_, v) { }
 explicit Matrix(Pt3<T> in)
  : r(3), c(1), data(3*1) {
  rep(i, 0, 3)
   data[i] = in[i];
 }
 explicit Matrix(Pt2<T> in)
  : r(2), c(1), data(2*1) {
  rep(i, 0, 2)
   data[i] = in[i];
 }
 // copy constructor, assignment
 // and destructor compiler defined
 T & operator()(int row, int col) {
  return data[col+row*c];
 }
 T const & operator()(int row, int col) const {
  return data[col+row*c];
 }
 // implement as needed
 bool operator==(In rhs) const {
  return data == rhs.data;
 }
 M operator+(In rhs) const {
  assert(rhs.r == r && rhs.c == c);
  Matrix ret(r, c);
```

```
    rep(i, 0, c*r)
     ret.data[i] = data[i]*rhs.data[i];
    return ret;
   }
   M operator-(In rhs) const {
    assert(rhs.r == r && rhs.c == c);
    Matrix ret(r, c);
    rep(i, 0, c*r)
     ret.data[i] = data[i]-rhs.data[i];
    return ret;
   }
   M operator*(In rhs) const { // matrix
      mult
    assert(rhs.r == c);
    Matrix ret(r, rhs.c);
    rep(i, 0, r)
     rep(j, 0, rhs.c)
              rep(k, 0, c)
               ret(i,j) += operator()(i, k)
               *rhs(k,j);
    return ret;
   }
   M operator*(T rhs) const { // scalar
      mult
    Matrix ret(*this);
    trav(it, ret.data)
     it = it*rhs;
    return ret;
   }
  };

  template<typename T> // create identity
      matrix
  Matrix<T> id(int r, int c) {
   Matrix<T> m(r,c);
   rep(i, 0, r)
    m(i,i) = T(1);
  }
```

6.4 Matrix3d Class

```
/* 3 dimensional matrix class
 * with Gauss Elimination and
     Eigenvectors
 * Source: Magnus Selin
 */

#include <cmath>

class Matrix3d{
  friend std::ostream& operator<< ( std
    ::ostream& os, Matrix3d fb );
private:
  double a[3][3];
public:
  Matrix3d(){
    for(int i = 0; i < 3; i++){
      for(int j = 0; j < 3; j++){
        if(j >= 0 && j < 3 && i >= 0 &&
    i < 3) a[i][j] = 0;
      }
    }
  }

  double get(int i, int j)      { if(j
   >= 0 && j < 3 && i >= 0 && i < 3)
    return a[i][j];  else std::cerr << "
    Out of bounds! \n"; }
  void set(int i, int j, int v)  { if(j
   >= 0 && j < 3 && i >= 0 && i < 3) a[i
    ][j] = v;    else std::cerr << "Out
    of bounds! \n"; };

  void chg_row(int x, int y);
  void mult_row(int x, double c);
  void add_row(int x, int y, double c);

  Matrix3d gauss();
  Matrix3d get_inverse();

  double get_det();
  void get_eigenvectors();

  Matrix3d operator= (Matrix3d);
  Matrix3d operator+ (Matrix3d);
  Matrix3d operator- (Matrix3d);
  Matrix3d operator* (Matrix3d);
  Matrix3d operator* (double);
};

void Matrix3d::chg_row(int x, int y){
  int temp[3] = {a[x][0], a[x][1], a[x
    ][2]};

  for(int i = 0; i < 3; i++){
    a[x][i] = a[y][i];
    a[y][i] = temp[i];
  }

}

void Matrix3d::mult_row(int x, double c)
    {
  for(int i = 0; i < 3; i++){
    a[x][i] *= c;
  }
}
void Matrix3d::add_row(int x, int y,
    double c){
  for(int i = 0; i < 3; i++){
    a[x][i] += c * a[y][i];
  }
}

void Matrix3d::get_eigenvectors(){
  double eig[3];
  double p = a[0][1] * a[0][1] + a[0][2]
     * a[0][2] + a[1][2] * a[1][2];
  double q, r, phi;
  Matrix3d B; Matrix3d I;
  for (int i = 0; i < 3; i++) I.set(i, i
    , 1);

  if ( p == 0 ){
    eig[0] = a[0][0];
    eig[1] = a[1][1];
    eig[2] = a[2][2];
  }
  else {
    q = (a[0][0] + a[1][1] + a[2][2]) /
    3;
    p = (a[0][0] - q) * (a[0][0] - q) +
      (a[1][1] - q) * (a[1][1] - q) +
      (a[2][2] - q) * (a[2][2] - q) + 2
    * q;
    p = sqrt( p / 6 );

    B = ((*this) - I * q);
    B = B * (1 / p);
    r = B.get_det();

    if (r <= -1)
      phi = M_PI / 3;
    else if (r >= 1)
      phi = 0;
    else
      phi = acos(r) / 3;

    eig[0] = q + 2 * p * cos(phi);
    eig[2] = q + 2 * p * cos(phi + M_PI
    * (2/3));
```

```
93      eig[1] = 3 * q - eig[0] - eig[2];
94    }
95
96    std::cout << eig[0] << ' ' << eig[1]
       << ' ' << eig[2] << ' ';
97
98    for (int i = 0; i < 3; i++) {
99      Matrix3d temp = (*this);
100
101     temp.set(0, 0, temp.get(0, 0) - eig[
        i]);
102     temp.set(1, 1, temp.get(1, 1) - eig[
        i]);
103     temp.set(2, 2, temp.get(2, 2) - eig[
        i]);
104     temp = temp.gauss();
105
106     std::cout << "Temp " << i << ":\n"
       << temp << "\n";
107   }
108 }
109
110 double Matrix3d::get_det(){
111   return a[0][0] * a[1][1] * a[2][2] + a
       [0][1] * a[1][2] * a[2][0] + a[0][2]
       * a[1][0] * a[2][1] -
112       a[0][2] * a[1][1] * a[2][0] - a
       [0][1] * a[1][0] * a[2][2] - a[0][0]
       * a[1][2] * a[2][1];
113 }
114
115
116 Matrix3d Matrix3d::gauss(){
117   Matrix3d * temp = new Matrix3d;
118   temp = this;
119
120   for (int i = 0; i < 3; i++){
121     if(temp->get(i, i) == 0){
122       for (int j = i; j < 3; j++){
123         if(temp->get(j, i) != 0){
124           temp->chg_row(i, j);
125           break;
126         }
127       }
128
129       if(temp->get(i, i) == 0){
130         std::cout << "Parameter solotion
       !!\n";
131         break;
132       }
133     }
134
135
136     double mult_val = temp->get(i, i);
137     temp->mult_row(i, 1 / mult_val );
138
139     for (int j = 0; j < 3; j++){
140       if(i != j){
141         double mult_val = -temp->get(j,
       i);
142         temp->add_row(j, i, mult_val);
143       }
144     }
145   }
146
147   std::cout << "Temp " << ":\n" << *temp
       << "\n";
148   std::cout << "This " << ":\n" << *this
       << "\n";
149
150   return *temp;
151 }
152
153 Matrix3d Matrix3d::get_inverse(){
154   Matrix3d temp = (*this) , inverse;
155   for (int i = 0; i < 3; i++) inverse.
       set(i, i, 1);
156
157   for (int i = 0; i < 3; i++){
158     if(temp.get(i, i) == 0){
159       for (int j = i; j < 3; j++){
160         if(temp.get(j, i) != 0){
161           temp.chg_row(i, j);
162           inverse.chg_row(i, j);
163
164           std::cout << "Change row " <<
       i << " and " << j << ".\n";
165           std::cout << temp << '\n';
166           std::cout << inverse << '\n';
167
168           break;
169         }
170       }
171
172       if(temp.get(i, i) == 0){
173         std::cout << "Singularity!\n";
174         break;
175       }
176     }
177
178
179     double mult_val = temp.get(i, i);
180     temp.mult_row(i, 1 / mult_val );
181     inverse.mult_row(i, 1 / mult_val );
182
183     std::cout << "Divide row " << i << "
       by " << mult_val << ".\n";
184     std::cout << temp << '\n';
185     std::cout << inverse << '\n';
186
187     for (int j = 0; j < 3; j++){
188       if(i != j){
189
190         double mult_val = -temp.get(j, i
       );
191
192         temp.add_row(j, i, mult_val);
193         inverse.add_row(j, i, mult_val);
194
195         std::cout << "Multiply row " <<
       i << " by " << mult_val << " and
       adding it to " << j << ".\n";
196         std::cout << temp << '\n';
197         std::cout << inverse << '\n';
198       }
199     }
200   }
201
202   return inverse;
203 }
204
205 Matrix3d Matrix3d::operator= (Matrix3d
       param){
206   Matrix3d temp;
207   for(int i = 0; i < 3; i++){
208     for(int j = 0; j < 3; j++){
209       temp.set(i, j, param.get(i, j));
210     }
211   }
212
213   return temp;
214 }
215 Matrix3d Matrix3d::operator+ (Matrix3d
       param){
216   Matrix3d temp;
217   for(int i = 0; i < 3; i++){
218     for(int j = 0; j < 3; j++){
219       temp.set(i, j, a[i][j] + param.get
       (i, j));
220     }
```

```
    }

    return temp;
}
Matrix3d Matrix3d::operator- (Matrix3d param){
    Matrix3d temp;
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            temp.set(i, j, a[i][j] - param.get(i, j));
        }
    }

    return temp;
}
Matrix3d Matrix3d::operator* (double param){
    Matrix3d temp;
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            temp.set(i, j, a[i][j] * param);
        }
    }

    return temp;
}
Matrix3d Matrix3d::operator* (Matrix3d param){
    Matrix3d temp;

    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            temp.set(i, j, a[i][0] * param.get(0,j) +
                    a[i][1] * param.get(1,j) +
                    a[i][2] * param.get(2,j) );
        }
    }

    return temp;
}

std::ostream& operator << ( std::ostream & os, Matrix3d m ){
    for(int i = 0; i < 3; i++){
        os << "(";
        for(int j = 0; j < 3; j++){
            os << m.get(i, j) << ",\t";
        }
        os << ")\n";
    }
    return os;
}
```

6.5 Points Class

```
/* Description: Untested homogenous coordinates
 * transformation geometry.
 * Source: Benjamin Ingberg
 * Usage: Requires homogenous coordinates, handles
 * multiple rotations, translations and scaling in a
 * high precision efficient manner (matrix
 * multiplication) with homogenous coordinates.
 * Also keeps reverse transformation available. */
namespace h { // avoid name collisions
 struct Transform {
  enum ActionType {
   Scale, Rotate, TranslateX, TranslateY
  };
  typedef tuple<ActionType, fp> Action;
  typedef Matrix<fp> M;
  typedef vector<Action> History;
  History hist;
  M to, from;
  Transform(History h = History())
   : to(id<fp>(3,3)), from(id<fp>(3,3)) {
   doTransforms(h);
  }
  H transformTo(H in) {
   return H(to*M(in));
  }
  H transformFrom(H in) {
   return H(from*M(in));
  }
  Transform & scale(fp s) {
   doTransform(Scale, s);
  }
  Transform & translate(fp dx, fp dy) {
   doTransform(TranslateX, dx);
   doTransform(TranslateY, dy);
  }
  Transform & rotate(fp phi) {
   doTransform(Rotate, phi);
  }
  void doTransforms(History & h) {
   trav(it, h) {
    doTransform(get<0>(*it), get<1>(*it));
   }
  }
  void doTransform(ActionType t, fp v) {
   hist.push_back(make_tuple(t, v));
   if(t == Scale)
    doScale(v);
   else if(t == TranslateX)
    doTranslate(0,v);
   else if(t == TranslateY)
    doTranslate(1,v);
   else
    doRotate(v);
  }
 private:
  void doScale(fp s) {
   M sm(id<fp>(3,3)), ism(id<fp>(3,3));
   sm(1,1) = sm(0,0) = s;
   ism(1,1) = ism(1,1) = 1/s;
   to = to*sm; from = ism*from;
  }
  void doTranslate(int c, fp dx) {
   M sm(id<fp>(3,3)), ism(id<fp>(3,3));
   sm(c,2) = dx;
   ism(c,2) = -dx;
   to = to*sm; from = ism*from;
  }
  void doRotate(fp phi) {
   M sm(id<fp>(3,3)), ism(id<fp>(3,3));
   sm(0,0) = sm(1,1) = cos(phi);
   ism(0,0) = ism(1,1) = cos(-phi);
   ism(1,0) = sm(0,1) = sin(phi);
   ism(0,1) = sm(1,0) = sin(-phi);
   to = to*sm; from = ism*from;
  }
 };
}
```

6.6 Graham Scan

```
struct point {
 int x, y;
};
```

```
int det(const point& p1, const point& p2
    , const point& p3)
{
 int x1 = p2.x      p1.x;
 int y1 = p2.y      p1.y;
 int x2 = p3.x      p1.x;
 int y2 = p3.y      p1.y;
 return x1*y2      x2*y1;
}

// bool ccw(const point& p1, const point
    & p2, const point& p3)
// { // Counterclockwise? Compare with
    determinant...
// return (det(p1, p2, p3) > 0);
// }

struct angle_compare {
 point p; // Leftmost lower point
 angle_compare(const point& p) : p(p) {
    }
 bool operator()(const point& lhs, const
     point& rhs) {
  int d = det(p, lhs, rhs);
  if(d == 0) // Furthest first if same
    direction will keep all
   return (x1*x1+y1*y1 > x2*x2+y2*y2);
    // points at the line
  return (d > 0); // Counterclockwise?
 }
};

int ConvexHull(const vector<point>& p,
    int* res)
{ // Returns number of points in the
    convex polygon
 int best = 0; // Find the first
    leftmost lower point
 for(int i = 1; i < p.size(); ++i)
  {
   if(p[i].y < p[best].y ||
         (p[i].y == p[best].y && p[i].x
            < p[best].x))
        best = i;
  }
 sort(p.begin(), p.end(), angle_compare(
    p[best]));
 for(int i = 0; i < 3; ++i)
  res[i] = i;
 int n = 3;
 for(int i = 3; i < p.size(); ++i)
  {
   // All consecutive points should be
    counter clockwise
   while(n > 2 && det(res[n-2], res[n
    -1], i) < 0)
        --n; // Keep if det = 0, i.e.
            the same line, angle_compare
   res[n++] = i;
  }
 return n;
}
```

6.7 Convex Hull

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

using namespace std;

typedef unsigned int nat;

template <class T>
struct Point {
 T x, y;

 Point(T x = T(), T y = T()) : x(x), y(y
    ) {}

 bool operator <(const Point<T> &o)
    const {
  if (y != o.y) return y < o.y;
  return x < o.x;
 }

 Point<T> operator -(const Point<T> &o)
    const { return Point<T>(x - o.x, y -
    o.y); }
 Point<T> operator +(const Point<T> &o)
    const { return Point<T>(x + o.x, y +
    o.y); }

 T lenSq() const { return x*x + y*y; }
};

template <class T>
struct sort_less {
 const Point<T> &ref;

 sort_less(const Point<T> &p) : ref(p)
    {}

 double angle(const Point<T> &p) const {
  Point<T> delta = p - ref;
  return atan2(delta.y, delta.x);
 }

 bool operator() (const Point<T> &a,
    const Point<T> &b) const {
  double aa = angle(a);
  double ab = angle(b);
  if (aa != ab) return aa < ab;
  return (a - ref).lenSq() < (b - ref).
    lenSq();
 }
};

template <class T>
int ccw(const Point<T> &p1, const Point<
    T> &p2, const Point<T> &p3) {
 return (p2.x - p1.x) * (p3.y - p1.y) -
    (p2.y - p1.y) * (p3.x - p1.x);
}

template <class T>
vector<Point<T> > convex_hull(vector<
    Point<T> > input) {
 if (input.size() < 2) return input;
 nat size = input.size();

 vector<Point<T> > output;

 // Find the point with the lowest x and
     y value.
 int minIndex = 0;
 for (int i = 1; i < size; i++) {
  if (input[i] < input[minIndex]) {
   minIndex = i;
  }
 }

 // This is the "root" point in our
    traversal.
 Point<T> p = input[minIndex];
 output.push_back(p);
 input.erase(input.begin() + minIndex);

```

```
  // Sort the other elements according to the angle with "p"
  sort(input.begin(), input.end(), sort_less<T>(p));

  // Add the first point from "input" to the "output" as a candidate.
  output.push_back(input[0]);

  // Start working our way through the points...
  input.push_back(p);
  size = input.size();
  for (nat i = 1; i < size; i++) {
   while (output.size() >= 2) {
    nat last = output.size() - 1;
    int c = ccw(output[last - 1], output[last], input[i]);

    if (c == 0) {
        // Colinear points! Take away the closest.
        if ((output[last - 1] - output[last]).lenSq() <= (output[last - 1] - input[i]).lenSq()) {
         if (output.size() > 1)
          output.pop_back();
         else
          break;
        } else {
         break;
        }
    } else if (c < 0) {
        if (output.size() > 1)
         output.pop_back();
        else
         break;
    } else {
        break;
    }
   }

   // Do not take the last point twice.
   if (i < size - 1)
    output.push_back(input[i]);
  }

  return output;
}
```

```


typedef Point<int> Pt;

bool solve() {
 nat count;
 scanf("%d", &count);

 if (count == 0) return false;

 vector<Pt> points(count);
 for (nat i = 0; i < count; i++) {
  scanf("%d %d", &points[i].x, &points[i].y);
 }

 vector<Pt> result = convex_hull(points);

 printf("%d\n", (int)result.size());
 for (nat i = 0; i < result.size(); i++) {
  printf("%d %d\n", result[i].x, result[i].y);
 }

 return true;
}

int main() {
 while(solve());

 return 0;
}
```

6.8 Line-point distance

```
// Problem 12173 on UVa (accepted there)

#include <cstdio>
#include <vector>
#include <cmath>
#include <iostream>

using namespace std;

typedef unsigned int nat;

template <class T>
class Point {
public:
 T x, y;

 Point() : x(), y() {}
 Point(T x, T y) : x(x), y(y) {}

 Point<T> operator -(const Point &o) const { return Point<T>(x - o.x, y - o.y); }
 Point<T> operator /(T o) const { return Point<T>(x / o, y / o); }
 T operator |(const Point &o) const {
  return x * o.x + y * o.y;
 }
};


template <class T>
class Vector {
public:
 T x, y, z;

 Vector() : x(), y(), z() {}
 Vector(const Point<T> &pt, T z) : x(pt.x), y(pt.y), z(z) {}
 Vector(T x, T y, T z) : x(x), y(y), z(z) {}

 Vector<T> operator -(const Vector &o) const { return Vector<T>(x - o.x, y - o.y, z - o.z); }
 Vector<T> operator /(T o) const { return Vector<T>(x / o, y / o, z / o); }
 T operator |(const Vector &o) const { return x * o.x + y * o.y + z * o.z; }
 Vector<T> operator %(const Vector &o) const {
  return Vector<T>(y*o.z - z*o.y, z*o.x - x*o.z, x*o.y - y*o.x);
 }
};

// distance between two points or vectors.
template <class T>
T dist(const Point<T> &a, const Point<T> &b) {
 Point<T> d = a - b;
 return sqrt(d | d);
```

```
}

// Normalize a line
template <class T>
void normLine(Vector<T> &v) {
 T l = sqrt(v.x * v.x + v.y * v.y);
 v = v / l;
}

// Normalize a point
template <class T>
void normPoint(Vector<T> &v) {
 v = v / v.z;
}

template <class T>
T dist(const Point<T> &point, const Point<T> &lineFrom, const Point<T> &lineTo) {
 // Outside first endpoint?
 if (((point - lineFrom) | (lineTo - lineFrom)) < 0) {
  return dist(point, lineFrom);
 }

 // Outside second endpoint?
 if (((point - lineTo) | (lineFrom - lineTo)) < 0) {
  return dist(point, lineTo);
 }

 // Ok, in the middle of the line!

 // Create the homogenous representation of the line...
 Vector<T> line = Vector<T>(lineFrom, 1) % Vector<T>(lineTo, 1);

 // The signed distance is then the dot product of the line
 // and the point.
 normLine(line);
 T distance = Vector<T>(point, 1) | line;

 // Don't return negative distances...
 return abs(distance);
}

vector<Point<double> > readPoints() {
 nat size = 0;
 scanf("%d", &size);

 vector<Point<double> > result;

 for (nat i = 0; i < size; i++) {
  double x, y;
  scanf("%lf %lf", &x, &y);
  result.push_back(Point<double>(x, y));
 }

 return result;
}

void solve() {
 vector<Point<double> > inner = readPoints();
 vector<Point<double> > outer = readPoints();

 double longest = 1e100;

 for (nat i = 0; i < inner.size(); i++) {
  nat iNext = (i + 1) % inner.size();
  for (nat j = 0; j < outer.size(); j++) {
   nat jNext = (j + 1) % outer.size();

   longest = min(longest, dist(outer[j], inner[i], inner[iNext]));
   longest = min(longest, dist(inner[i], outer[j], outer[jNext]));
  }
 }

 printf("%.8lf\n", longest / 2.0);
}

int main() {

 int tc;
 scanf("%d", &tc);

 while (tc--) solve();

 return 0;
}
```

6.9 Polygon Area

```
/* Calculate the area of an arbitrary polygon
 * <vector> and "geometry.cpp" must be included
 * source: Magnus Selin
 */

template <class T>
int area(vector<Vector<T> > v){
  int area = 0;
  for(int i = 0; i < v.size()-1; i++)
    area += (v[i] % v[i+1]).z;
  area += (v[v.size()-1] % v[0]).z;
  return area;
}
```

7 Misc

7.1 Longest Increasing Subsequence

```
#include <stdio.h>
#include <string.h>
#include <vector>
#include <algorithm>

using namespace std;
int bin_search(int a[], int t[], int l, int r, int k) {
  int m;
  while( r - l > 1 ) {
   m = l + (r - l)/2;
   if( a[t[m]] >= k )
     r = m;
   else
     l = m;
  }
  return r;
}

vector<int> lis(int a[], int n){
  std::vector<int> lis;
  if(n == 0) return lis;
  int c[n]; memset(c, 0, sizeof(c));
  int p[n]; memset(p, 0xFF, sizeof(p));
  int s = 1;

  c[0] = 0;
  p[0] = -1;
  for(int i = 1; i < n; i++){
    if(a[i] < a[c[0]]){
```

```
      c[0] = i;
    }
    else if(a[i] > a[c[s-1]]){
      p[i] = c[s-1];
      c[s] = i;
      s++;
    }
    else{
      int pos = bin_search(a, c, -1, s
    -1, a[i]);
      p[i] = c[pos-1];
      c[pos] = i;
    }
  }


  int d = c[s-1];
  for( int i = 0; i < s; i++ ){
    lis.push_back(d);
    d = p[d];
  }


  reverse(lis.begin(),lis.end());
  return lis;
}
int main(){
  int n;
  while(scanf("%d", &n) == 1){
    int a[n]; for(int i = 0; i < n; i++)
     scanf("%d", &a[i]);
    vector<int> lseq = lis(a, n);

    printf("%d\n", (int)lseq.size());
    for(int i = 0; i < lseq.size(); i++)
    {
      printf("%d␣", lseq[i]);
    }
    printf("\n");

    lseq.clear();
  }
}
```

7.2 Longest Increasing Substring

```
/* Longest common substring. */
int HadenIngberg(string const & s,
    string const & t){
  int n = s.size(), m = t.size(), best;
  for(int i = 0; i < n-best; ++i) { //
    Go through s
    int cur = 0;
    int e = min(n-i, m);

  // Can best grow?
    for(int j = 0; j < e && best+j < cur
    +e; ++j)
      best = max(best,
      cur = (s[i+j] == t[j] ? cur+1 : 0))
    ;
  }

  for(int i = 1; i < m-best; ++i) { //
    Go through t
    int cur = 0;
    int e = min(m-i, n);
  // Can best grow?
    for(int j = 0; j < e && best+j < cur
    +e; ++j)
    best = max(best,cur=(t[i+j] == s[j]?
    cur+1:0));
  }
  return best;
}
```

7.3 Knapsack

```
/*
 * Magnus Selin 2013-06-02
 * AAPS LiU
 * Task 1.2: Knapsack
 */

#include <iostream>
#include <vector>
#include <string.h>

using namespace std;

typedef vector<int> vi;

vi knapsack(int C, int n, int * v, int *
    w){
  int m[n+1][C+1]; memset(m[0], 0,
    sizeof(m[0]));
  bool k[n+1][C+1]; memset(k, 0, sizeof(
    k));

  // Starting with the first item, then
    the first and second etc. until all
    items are included
  for (int i = 1; i <= n; i++){
    // Starting with a bag of size 0
    then 1 up to C
    for (int j = 0; j <= C; j++){
      // If you chose to pack item i-1
    or stay with the ones before
      if (j >= w[i-1]){
        if(m[i-1][j] >= m[i-1][j-w[i-1]]
     + v[i-1]){
          m[i][j] = m[i-1][j];
          k[i][j] = false;
        }
        else{
          m[i][j] = m[i-1][j-w[i-1]] + v
    [i-1];
          k[i][j] = true;
        }
      }
      else{
        m[i][j] = m[i-1][j];
        k[i][j] = false;
      }
    }
  }

  for(int i = 0; i < n+1; i++){
    for(int j = 0; j < C+1; j++){
      cout << m[i][j] << '␣';
    }
    cout << '\n';
  }
  cout << '\n';
  for(int i = 0; i < n+1; i++){
    for(int j = 0; j < C+1; j++){
      cout << k[i][j] << '␣';
    }
    cout << '\n';
  }

  int c = (int)C;
  vi obj;
  // Get the index of which items too
    pack
  for(int i = n; i > 0; i--){
    if(k[i][c]){
      obj.push_back(i-1);
      c -= w[i-1];
```

```
        if(c <= 0) break;
      }
    }
    return obj;
}

int main(){
    double C; int n;
    while(cin >> C >> n){
        int v[n], w[n];
        for(int i = 0; i < n; i++) cin >> v[
        i] >> w[i];

        vi obj = knapsack(C, n, v, w);

        cout << obj.size() << '\n';
        for (int i = 0; i < obj.size(); i++)
        {
            cout << obj[i] << ' ';
        }
        cout << "\n";
    }

    return 0;
}
```