

NAT: Nostalgic Alien Trespassers — TCR NWERC 2013

Filip Strömbäck, Magnus Selin, Carl Einarson

November 24, 2013

Contents

1 Environment	2	4.6 Euler Tour	9
1.1 Template	2	4.7 Bipartite Matching	9
2 Data Structures	2	4.8 Strongly Connected Components	9
2.1 Union Find	2	5 String processing	10
2.2 Fenwick Tree	2	5.1 STL	10
3 Numerical	3	5.2 String Matching	10
3.1 General Utils	3	5.3 String Multimatching	10
3.2 Rational Numbers Class	3	6 Geometry	10
3.3 Binary Search	4	6.1 Points Class	10
3.4 De Bruijn	4	6.2 Lines	10
3.5 Prime Generator	4	6.3 Matrix Class	12
3.6 Factorisation	5	6.4 Matrix3d Class	13
3.7 Chinese remainder	5	6.5 Points Class	15
3.8 Diophantine equations	5	6.6 Graham Scan	15
4 Graphs	6	6.7 Convex Hull	16
4.1 Single Source Shortest Path	6	6.8 Line-point distance	17
4.2 Single Source Shortest Path Time Table	6	6.9 Polygon Area	18
4.3 All Pairs Shortest Path	7	7 Misc	18
4.4 Minimum Spanning Tree	8	7.1 Longest Increasing Subsequence	18
4.5 Maximum Flow	8	7.2 Longest Increasing Substring	19
		7.3 Knapsack	19

1 Environment

1.1 Template

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <cstdio>
4 #include <cmath>
5 #include <vector>
6 #include <set>
7 #include <map>
8 #include <stack>
9 #include <queue>
10 #include <string>
11 #include <bitset>
12 #include <algorithm>
13 #include <cstring>
14
15 using namespace std;
16
17 #define rep(i, a, b) for(int i = (a); i
    < int(b); ++i)
18 #define trav(it, v) for(typeof((v).begin
    ()) it = (v).begin(); it != (v).end()
    ; ++it)
19
20 typedef double fl;
21 typedef long long lli;
22 typedef pair<int, int> pii;
23 typedef vector<int> vi;
24
25
26 bool solve(){
27
28     return true;
29 }
30
31 int main(){
32     int tc=1; //scanf("%d", &tc);
33     rep(i, 0, tc) solve();
34
35     return 0;
36 }
```

2 Data Structures

2.1 Union Find

```
1 #include <iostream>
2 #include <stdio.h>
```

```
3 #include <string.h>
4 using namespace std;
5
6 int find(int * root, int x){
7     if (root[x] == x) return x;
8     root[x] = find(root, root[x]);
9     return root[x];
10 }
11
12 void uni(int * root, int * deep, int x,
    int y){
13     int a = find(root, x);
14     int b = find(root, y);
15     root[a] = b;
16 }
17
18 bool issame(int * root, int a, int b){
19     return (find(root, a) == find(root, b))
    ;
20 }
21
22 int main(){
23     int n, no; scanf("%d%d", &n, &no);
24     int root[n];
25     for(int i = 0; i < n; i++){
26         root[i] = i;
27     }
28
29     for(int i = 0; i < no; i++){
30         char op; int a, b;
31         scanf("%*[\n\t]%c", &op);
32         scanf("%d%d", &a, &b);
33         if(op == '?'){
34             if(issame(root, a, b)) printf("yes
    \n");
35             else printf("no\n");
36         }
37         if(op == '=')
38             uni(root, deep, a, b);
39     }
40 }
```

2.2 Fenwick Tree

```
1 #include <iostream>
2 #include <stdio.h>
3 #include <vector>
4
5 using namespace std;
6
```

```
7
8 typedef long long int lli;
9 typedef vector<lli> vi;
10
11 #define last_dig(x) (x & (-x))
12
13
14 void fenwick_create(vi &t, lli n){
15     t.assign(n + 1, 0);
16 }
17 lli fenwick_read(const vi &t, lli b){
18     lli sum = 0;
19     while(b > 0){
20         sum += t[b];
21         b -= last_dig(b);
22     }
23     return sum;
24 }
25
26 void fenwick_update(vi &t, lli k, lli v)
    {
27     while(k <= (lli)t.size()){
28         t[k] += v;
29         k += last_dig(k);
30     }
31 }
32
33 int main(){
34     lli N, Q; scanf("%lld%lld", &N, &Q);
35     vi ft; fenwick_create(ft, N);
36
37     char op; lli a, b;
38     for(lli i = 0; i < Q; i++){
39         scanf("%*[\n\t]%c", &op);
40         switch (op){
41             case '+':
42                 scanf("%lld%lld", &a, &b);
43                 fenwick_update(ft, a+1, b);
44                 break;
45
46             case '?':
47                 scanf("%lld", &a);
48                 printf("%lld\n", fenwick_read(ft,
    a));
49                 break;
50             }
51     }
52
53     return 0;
54 }
```

3 Numerical

3.1 General Utils

```
1 // Externa funktioner:
2 // OutIt copy(InIt first, InIt last,
  OutIt x);
3 // Returvrde: x + N, utiteratorn efter
  sista elementet.
4 // void fill(FwdIt first, FwdIt last,
  const T& x);
5 // bool next_permutation(BidIt first,
  BidIt last, Pred pr); // O(n)
6 // Funktion: Permuterar mngden till
  nsta variant enligt lexikal ordning.
7 // Kommentar: Brja med en sorterad
  mngd. Tar ej med dubletter.
8 // void nth_element(RanIt fi, RanIt nth,
  RanIt la [,Pred pr]);
9 // Funktion: Delar upp elementen s att
  *nth r strre
10 // eller lika alla element i [first, nth
  [
11 // och *nth r mindre eller lika alla
  element i ]nth, last[.
12 // Komplexitet: O(n) i medeltal
13 // BidIt partition(BidIt first, BidIt
  last, Pred pr); // O(n)
14 // Returvrde: first + k, iteratorn fr
  frsta elementet i andra intervallet.
15 // Funktion: Delar upp elementen s att
  pr() r sant resp. falskt fr alla
16 // element i intervallen [0, k[
  respektive [k, n[.
17 // FwdIt stable_partition(FwdIt first,
  FwdIt last, Pred pr);
18 // Kommentar: Samma som ovan men bevarar
  inbrdes ordning.
19 // void sort(RanIt first, RanIt last [,
  Pred pr]); // O(n*log(n))
20 // Kommentar: Fr list<> anvnd den
  interna funktionen l.sort().
21 // void stable_sort(RanIt first, RanIt
  last [, Pred pr]);
22 // Kommentar: Samma som ovan men bevarar
  inbrdes ordning.
23 // FwdIt unique(FwdIt first, FwdIt last
  [, Pred pr]); // O(n)
24 // Returvrde: first + k, iteratorn
  efter sista elementet i mngden.
```

```
25 // Funktion: Delar upp elementen s att
  inga p varandra fljande
26 // element i [0, k) r lika.
27 // Elementen i [k, last[ r odefinierade
  .
28 // Kommentar: Fr list<> anvnd den
  interna funktionen l.unique().
29 //
30 // Skning i sorterade mngder
31 // Fljande funktioner har
  tidskomplexiteten O(log(n)) med
  undantaget O(n)
32 // fr list. De tre sista samt funktion
  find() finns internt i map
33 // och set. Returnerar c.end() om inget
  passande element hittas.
34 // bool binary_search(FwdIt first, FwdIt
  last, T& x [, Pred pr]);
35 // Returvrde: true om x finns, annars
  false.
36 // FwdIt lower_bound(FwdIt first, FwdIt
  last, T& x [, Pred pr]);
37 // Returvrde: first + k, frsta
  positionen som x kan sttas
38 // in p s att sorteringen, dvs. varje
  element i [0, k[ r mindre n x.
39 // FwdIt upper_bound(FwdIt first, FwdIt
  last, T& x [, Pred pr]);
40 // Returvrde: first + k, sista
  positionen som x kan sttas
41 // in p s att sorteringen bibehlls,
  dvs. varje element i
42 // [k, n[ r strre n x.
43 // pair<It, It> equal_range(It first, It
  last, T& x [,Pred pr]);
44 // Returvrde: pair(lower_bound(fi, la,
  x),upper_bound(fi, la, x))
45
46 // Binary search (from Wikipedia)
47 // The indices are _inclusive_.
48 int binary_search(T *a, int key, int min
  , int max) {
49     while (min < max) {
50         int mid = (min + max) / 2; // midpoint
          (min, max)
51
52         // assert(mid < max)
53
54         // The condition can be replaced by
          some other function
```

```
55 // depending on mid, eg worksFor(mid +
  1) to search for
56 // the last index "worksFor" returns
  true for.
57 if (a[mid] < key) {
58     min = mid + 1;
59 } else {
60     max = mid;
61 }
62 }
63
64 // Equality test, can be skipped when
  looking for a specific value
65 if ((max == min) && (a[min] == key))
66     return min;
67 else
68     return NOT_FOUND;
69 }
70
71 // Fenwick tree:
```

3.2 Rational Numbers Class

```
1 #include <stdio.h>
2
3 using namespace std;
4
5 class Q{
6 private:
7     long long int p, q;
8     long long int gcd(long long int a,
        long long int b) {
9         if (a < 0) a = -a;
10        if (b < 0) b = -b;
11        if (0 == b) return a;
12        else return gcd(b, a % b);
13    }
14 public:
15    Q() {}
16    Q(long long int a, long long int b){
17        p = a; q = b;
18        if(q < 0){p = -p; q = -q;}
19        if (p == 0) q = 1;
20        if (q == 0){
21            printf("ERR: _den_=_0!\n");
22            q = 1;
23        }
24        long long int g = gcd(p, q);
25        p /= g; q /= g;
26    }
```

```

27
28 Q operator + (Q a){
29     Q b = * this;
30     Q res = Q((a.p * b.q + b.p * a.q), (
31         a.q * b.q));
32     return res;
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)
39         , (a.q * b.q));
40     return res;
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("
77         %c", &t);
78
79     scanf("%d%d", &tp, &tn); Q b = Q(tp,
80         tn);
81
82     switch(t){
83         case '+': (a+b).print(); break;
84         case '-': (a-b).print(); break;
85         case '*': (a*b).print(); break;
86         case '/': (a/b).print(); break;
87     }
88
89     return 0;

```

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 *
6     elem) {
7     int k = (int)key;
8     int e = (int)elem;
9     printf("Comparing ␣%d␣with␣%d\n", k, e);
10
11     if (k == e) return 0;
12     if (k < e) return -1;
13     return 1;
14 }
15
16 int main() {
17     int found = (int)bsearch((const void *)
18         10, 0, 100, 1, &check);
19
20     printf("I␣found:␣%d\n", found);
21
22     return 0;
23 }

```

3.4 De Bruijn

```

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             ++){
22             a[t] = j;
23             db(t + 1, t);
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)
35                 2, n)] * pow((double)2, n-i-1);
36         }
37         cout << sum << '\n';
38     }
39
40     int main(){
41         int tc;
42         cin >> tc;
43         for(int we = 0; we < tc; we++){
44             cin >> n >> k;
45             a.clear(); seq.clear();
46             de_bruijn();
47         }

```

3.5 Prime Generator

```

1 #include <cstdio>
2

```

```

3 int prime[664579];
4 int numprimes;
5
6 void calcprimes(int maxn){
7     prime[0] = 2; numprimes = 1; prime[
8         numprimes] = 46340; // 0xb504*0xb504
9         = 0x7FFEA810
10    for(int n = 3; n < maxn; n += 2) {
11        for(int i = 1; prime[i]*prime[i] <=
12            n; ++i) {
13            if(n % prime[i] == 0) goto
14                not_prime;
15        }
16    }
17
18    int main(){
19        calcprimes(10000000);
20        for(int i = 0; i < 664579; i++) printf
21            ("%d\n", prime[i]);
22    }

```

3.6 Factorisation

```

1 int factor[1000000];
2 int numf[1000000];
3 int numfactors;
4
5 void calcfactors(int n){
6     numfactors = 0;
7     for(int i = 0; n > 1; ++i){
8         if(n % prime[i] == 0){
9             factor[numfactors] = prime[i];
10            numf[numfactors] = 0;
11            do {
12                numf[numfactors]++;
13                n /= prime[i];
14            } while(n % prime[i] == 0);
15            numfactors++;
16        }
17    }

```

3.7 Chinese remainder

```

1 #pragma once

```

```

2 #include "number.h"
3 #include "modular.h"
4
5 /*
6  * Solve the problem:
7  * x = a (mod m)
8  * x = b (mod n)
9  * ...
10 * Where a, b, m and n are valid modular
11   numbers. A NoSolution is thrown if
12   there is no solution.
13 */
14 Modular<int64> remainder(const vector<
15     Modular<int64> > &eq) {
16     typedef Modular<int64> M;
17
18     M last = eq[0];
19
20     for (nat i = 1; i < eq.size(); i++) {
21         M curr = eq[i];
22
23         /* Given the two currently first
24          equations: */
25         /* r = a (mod m) = b (mod n) */
26         /* r = a + x*m = b + y*n */
27         /* a - b = y*n - x*m */
28         Dioph<int64> sol = diophantine(-last
29             .modulo, curr.modulo, last.value -
30             curr.value);
31         int64 mod = lcm(last.modulo, curr.
32             modulo);
33
34         last = M(last.value, mod) + M(last.
35             modulo, mod) * M(sol.x, mod);
36     }
37
38     return last;
39 }

```

3.8 Diophantine equations

```

1 /*
2  * Find the greatest common divisor of
3  * two positive numbers. T is an
4  * integral type.
5  */
6 template <class T>
7 inline T gcd(T a, T b) {
8     if (a < b) swap(a, b);
9
10    while (b != 0) {
11        T r = a % b;
12        a = b;
13        b = r;
14    }
15    return a;
16 }

```

```

8 while (b != 0) {
9     T r = a % b;
10
11     a = b;
12     b = r;
13 }
14
15 return a;
16 }
17
18 /*
19 * Find the least common multiple of two
20   positive numbers. T is an integral
21   type.
22 */
23 template <class T>
24 inline T lcm(T a, T b) {
25     return a * (b / gcd(a, b));
26 }
27
28 /*
29 * Do the euclid's algorithm both
30   forward and reverse, to find
31   solutions to
32   diophantine equations. T shall be
33   signed, as this algorithm will use
34   negative
35   numbers internally.
36 * Generates one integer solution to the
37   equation a*x + b*y = sum, throws
38   NoSolution
39 * if none exists. "a" and "b" are
40   assumed to be positive.
41 */
42 template <class T>
43 struct Dioph {
44     T x, y;
45
46     Dioph(const T &x = T(), const T &y = T
47         ()) : x(x), y(y) {}
48 };
49
50 template <class T>
51 ostream &operator <<(ostream &to, const
52     Dioph<T> &f) {
53     return to << "{x=" << f.x << ",y="
54         << f.y << "}";
55 }
56
57 }
58
59 }

```

```

46 template <class T>
47 Dioph<T> diophantine(T a, T b, T sum) {
48
49     /* If b == 0, it is easy: */
50     if (b == T()) {
51         /* x = sum / a, iff sum % a != 0 */
52         if (sum % a == 0)
53             return Dioph<T>(sum / a, 0);
54         else
55             throw NoSolution();
56     } else if (a == T()) {
57         if (sum % b == 0)
58             return Dioph<T>(sum / b, 0);
59         else
60             throw NoSolution();
61     }
62
63     /* Compute the form: a = z * b + r */
64     /* We can also say: r = a - z*b */
65     T z = a / b;
66     T r = a % b;
67
68     /* Solve the new equation, b*x + r*y =
69        sum */
69     Dioph<T> sol = diophantine(b, r, sum);
70
71     /* We know: r = a - z*b */
72     /* Substitution gives: b*x + (a - z*b)
73        *y = sum */
73     /* Which equals: b*x + a*y - z*b*y =
74        sum => a*y + b*(x - z*y) */
74     return Dioph<T>(sol.y, sol.x - sol.y*z
75 );
75 }

```

4 Graphs

4.1 Single Source Shortest Path

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

```

1  #include <stdio.h>
2  #include <queue>
3  #include <vector>
4
5  #define INF 100000000
6
7  using namespace std;
8
9  typedef pair<int, int> ii;
10

```

```

11 template<class T>
12
13 class comp{
14 public:
15     int operator()(const pair<int, T> & a,
16                     const pair<int, T> & b){return (a.
17                     second > b.second);}
18 };
19
20 template<class T>
21 vector<T> dijkstras(vector<pair<int, T>
22 > G[], int n, int e, int s){
23     priority_queue<pair<int, T>, vector<
24     pair<int, T>, comp> Q;
25
26     vector<T> c; for(int i = 0; i < n; i
27     ++ ) c.push_back(INF); c[s] = 0;
28     vector<int> p; for(int i = 0; i < n; i
29     ++ ) p.push_back(-1);
30
31     Q.push(pair<int, T>(s, c[s]));
32     int u, sz, v; T w;
33     while(!Q.empty()){
34
35         u = Q.top().first; Q.pop();
36         sz = G[u].size();
37         for(int i = 0; i < sz; i++){
38             v = G[u][i].first;
39             w = G[u][i].second;
40             if( c[v] > c[u] + w ){
41                 c[v] = c[u] + w;
42                 p[v] = u;
43                 Q.push(pair<int, T>(v, c[v]));
44             }
45         }
46     }
47
48     //printf("Path to follow: ");
49     //for(int i = 0; i < n; i++) printf("%
50     d ", p[i]);
51     //printf("\n");
52
53     return c;
54 }
55
56 int main(){
57     int n, e, q, s;
58     scanf("%d%d%d%d", &n, &e, &q, &s);
59     while(n!=0 or e!=0 or q!=0 or s!=0){
60         vector<ii> G[n];

```

```

54     for(int i = 0; i < e; i++){
55         int f, t, w;
56         scanf("%d%d%d", &f, &t, &w);
57         G[f].push_back(ii(t, w));
58     }
59     vector<int> c = dijkstras(G, n, e, s
60 );
61
62     for(int i = 0; i < q; i++) {
63         int d; scanf("%d", &d);
64         if(c[d] == INF) printf("
65 Impossible\n");
66         else printf("%d\n", c[d]);
67     }
68     printf("\n");
69
70     scanf("%d%d%d%d", &n, &e, &q, &s);
71 }
72
73 return 0;
74 }

```

4.2 Single Source Shortest Path Time Table

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

```

1  #include <stdio.h>
2  #include <queue>
3  #include <vector>
4
5  #define INF 100000000
6
7  using namespace std;
8
9  struct A{
10     A(int a, int b, int c){t0=a; tn = b; w
11     = c;}
12     int t0, tn, w;
13 };
14
15 typedef pair<int, int> ii;
16 typedef pair<int, A> iA;
17
18 class comp{
19 public:
20     int operator()(const ii& a, const ii&
21     b){return (a.second > b.second);}
22 };

```

```

22 vector<int> dijkstras(vector<iA> G[],
    int n, int e, int s){
23     priority_queue<ii, vector<ii>, comp> Q
        ;
24
25     vector<int> c; for(int i = 0; i < n; i
        ++){ c.push_back(INF); c[s] = 0;
26     vector<int> p; for(int i = 0; i < n; i
        ++){ p.push_back(-1);
27
28     Q.push(ii(s, c[s]));
29     int u, sz, v, t0, tn, w, wt;
30     while(!Q.empty()){
31
32         u = Q.top().first; Q.pop();
33         sz = G[u].size();
34         for(int i = 0; i < sz; i++){
35             v = G[u][i].first;
36             tn = G[u][i].second.tn;
37             t0 = G[u][i].second.t0;
38             w = G[u][i].second.w;
39
40             wt = t0 - c[u];
41             if (wt < 0 and tn == 0) continue;
42             while(wt < 0) wt+=tn;
43
44             if( c[v] > c[u] + w + wt){
45                 c[v] = c[u] + w + wt;
46                 p[v] = u;
47                 Q.push(ii(v, c[v]));
48             }
49         }
50     }
51
52     //printf("Path to follow: ");
53     //for(int i = 0; i < n; i++) printf("%
        d ", p[i]);
54     //printf("\n");
55
56     return c;
57 }
58
59 int main(){
60     int n, e, q, s;
61     scanf("%d%d%d", &n, &e, &q, &s);
62     while(n!=0 or e!=0 or q!=0 or s!=0){
63         vector<iA> G[n];
64         for(int i = 0; i < e; i++){
65             int f, t, t0, tn, w;

```

```

66         scanf("%d%d%d%d", &f, &t, &t0, &
            tn, &w);
67         G[f].push_back(iA(t, A(t0, tn, w)
            ));
68     }
69     vector<int> c = dijkstras(G, n, e, s
        );
70
71     for(int i = 0; i < q; i++) {
72         int d; scanf("%d", &d);
73         if(c[d] == INF) printf("
            Impossible\n");
74         else printf("%d\n", c[d]);
75     }
76     printf("\n");
77
78     scanf("%d%d%d", &n, &e, &q, &s);
79 }
80
81     return 0;
82 }

```

4.3 All Pairs Shortest Path

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

```

1 // All pairs shortest path (Floyd
    Warshall). Assign nodes which are
    part of a
2 // negative cycle to minus infinity.
3
4 #include <stdio.h>
5 #include <iostream>
6 #include <vector>
7 #include <algorithm>
8
9 #define INF 1000000000
10 using namespace std;
11
12 template<class T>
13 vector< vector<T> > floyd_warshall(
    vector< vector<T> > d){
14     int n = d.size();
15     for(int i = 0; i < n; i++) d[i][i] =
        0;
16
17     for (int k = 0; k < n; k++)
18         for (int i = 0; i < n; i++)
19             for (int j = 0; j < n; j++)

```

```

20         if (d[i][k] != INF and d[k][j] !=
            INF)
21             d[i][j] = min(d[i][j], d[i][k]+d
                [k][j]);
22
23     for(int i = 0; i < n; i++){
24         for(int j = 0; j < n; j++){
25             for(int k = 0; d[i][j] != -INF &&
                k < n; k++){
26                 if(d[i][k] != INF && d[k][j] !=
                    INF && d[k][k] < 0)
27                     d[i][j] = -INF;
28
29         return d;
30     }
31
32     int main(){
33         int n, m, q; scanf("%d%d", &n, &m, &
            q);
34         while(n!=0 or m!=0 or q!=0){
35             vector< vector<int> > d;
36             d.resize(n);
37             for(int i = 0; i < n; i++){
38                 for(int j = 0; j < n; j++){
39                     d[i].push_back(INF);
40
41                 for(int i = 0; i < m; i++){
42                     int f, t, w; scanf("%d%d", &f, &
                        t, &w);
43                     d[f][t] = min(w, d[f][t]);
44                 }
45
46                 d = floyd_warshall(d, n);
47                 for(int i = 0; i < q; i++){
48                     int f, t; scanf("%d", &f, &t);
49                     if(d[f][t] == INF) printf("
                        Impossible\n");
50                     else if(d[f][t] == -INF) printf("
                        -Infinity\n");
51                     else printf("%d\n", d[f
                        ][t]);
52                 }
53                 printf("\n");
54                 scanf("%d%d", &n, &m, &q);
55             }
56             return 0;
57 }

```

4.4 Minimum Spanning Tree

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

```
1  #include <stdio.h>
2  #include <algorithm>
3  #include <vector>
4
5  using namespace std;
6
7  struct AnsEdge{
8      int f, t;
9      bool operator<(const AnsEdge& oth)
10         const{
11         if(f == oth.f)
12             return(t < oth.t);
13         return(f < oth.f);
14     }
15     AnsEdge(){};
16     AnsEdge(int a, int b){f = a; t = b;};
17 };
18 struct Tree{
19     int w;
20     bool complete;
21     std::vector<AnsEdge> e;
22     Tree(){
23         w = 0;
24         complete = true;
25     }
26 };
27
28 struct Vertex{
29     Vertex *p;
30     Vertex *root(){
31         if(p->p != p)
32             p = p->root();
33         return p;
34     }
35 };
36 struct Edge{
37     int f, t, w;
38
39     bool operator<(const Edge& oth) const{
40         if (w == oth.w)
41             return(t < oth.t);
42         return(w < oth.w);
43     }
44 };
45
46
```

```
47 Tree kruskal(Vertex * v, Edge * e, int
    numv, int nume){
48     Tree ans;
49     int sum = 0;
50
51     for(int i = 0; i < numv; ++i){
52         v[i].p = &v[i];
53     }
54
55     sort(&e[0], &e[nume]);
56
57     for(int i = 0; i < nume; ++i){
58         if(v[e[i].f].root() != v[e[i].t].
59             root()){
60             v[e[i].t].root()->p = v[e[i].f].
61                 root();
62             ans.w += e[i].w;
63
64             if(e[i].t < e[i].f) ans.e.
65                 push_back(AnsEdge(e[i].t, e[i].f));
66             else ans.e.push_back(
67                 AnsEdge(e[i].f, e[i].t));
68         }
69     }
70
71     Vertex * p = v[0].root();
72     for(int i = 0; i < numv; ++i)
73         if(p != v[i].root()){
74             ans.complete = false;
75             break;
76         }
77
78     sort(ans.e.begin(), ans.e.end());
79
80     return ans;
81 }
82
83 int main(){
84     int n, m; scanf("%d%d", &n, &m);
85     while(n or m){
86         Vertex v[n];
87         Edge e[m];
88
89         for(int i = 0; i < m; i++){
90             int f, t;
91             scanf("%d%d%d", &f, &t, &e[i].w);
92             e[i].f = f;
93             e[i].t = t;
94         }
95
96
```

```
97     Tree ans = mst(v, e, n, m);
98
99     if(ans.complete){
100         printf("%d\n", ans.w);
101         for(int i = 0; i < ans.e.size(); i
102             ++){
103             printf("%d□%d\n", ans.e[i].f,
104                 ans.e[i].t);
105         }
106     }
107     else printf("Impossible\n");
108
109     scanf("%d%d", &n, &m);
110 }
111
112 return 0;
113 }
```

4.5 Maximum Flow

Edmonds Karp's Maximum Flow Algorithm

Input: Adjacency Matrix (res)

Output: Maximum Flow

Time Complexity: $O(VE^2)$

```
1  int res[MAX_V][MAX_V], mf, f, s, t;
2  vi p;
3
4  void augment(int v, int minEdge) {
5      if(v == s){f = minEdge; return;}
6      else if(p[v] != -1){augment(p[v], min
7          (minEdge, res[v][p[v]]));
8          res[p[v]][v] -= f; res[v][p[
9              v]] += f; }
10 }
11
12 int solve(){
13     mf = 0; // Max Flow
14
15     while(1){
16         f = 0;
17         vi dist(MAX_V, INF); dist[s] = 0;
18         queue<int> q; q.push(s);
19         p.assign(MAX_V, -1);
20         while(!q.empty()){
21             int u = q.front(); q.pop();
22             if(u == t) break;
23             for(int v = 0; v < MAX_V; v++){
24                 if (res[u][v] > 0 && dist[v] ==
25                     INF)
26                     dist[v] = dist[u] + 1, q.push(
27                         v), p[v] = u;
28             }
29         }
30     }
31
32     mf += f;
33     return mf;
34 }
```



```

23     }
24     augment(t, INF);
25     if(f == 0) break;
26     mf += f;
27 }
28
29 printf("%d\n", mf);
30 }

```

4.6 Euler Tour

Time Complexity $O(E + V)$

```

1  #include <cstdlib>
2  #include <cstdio>
3  #include <cmath>
4  #include <list>
5
6  typedef vector<int> vi;
7
8  using namespace std;
9
10 list<int> cyc;
11
12 void euler_tour(list<int>::iterator i,
13     int u) {
14     for(int j = 0; j < (int)AdjList[u].
15         size(); j++){
16         ii v = AdjList[u][j];
17         if (v.second){
18             v.second = 0;
19             for(int k = 0; k < (int)AdjList[v.
20                 first][k]){
21                 ii uu = AdjList[v.first][k];
22                 if(uu.first == u && uu.second) {
23                     uu.second = 0; break;}
24             }
25             euler_tour(cyc.insert(i, u), v.
26                 first)
27         }
28     }
29 }
30
31 int main(){
32     cyc.clear();
33     euler_tour(cyc.begin(), A);
34     for(list<int>::iterator it = cyc.begin
35         (); it != cyc.end(); it++){
36         printf("%d\n", *it);
37     }
38 }

```

4.7 Bipartite Matching

```

1  /* Name: Bipartite DFS
2   * Description: Simple bipartite
3   * matching.
4   * Slower than HopcroftKarp but shorter.
5   * Graph g should be a list of
6   * neighbours
7   * of the left partition.
8   * n is the size of the left partition
9   * and m is the size of the right
10  * partition.
11  * Ifyou want to get the matched pairs,
12  * \lstinline|match[i]| contains match
13  * for vertex i on
14  * the right side or -1 if it's not
15  * matched.
16  * Time:  $O(EV)$ 
17  * Usage example:
18  * \begin{lstlisting}[frame=none,
19  *     aboveskip=-0.6cm, ]
20  * Graph left(n);
21  * trav(it, edges){
22  *     l[it->left].push_back(it->right);
23  * }
24  * dfs_matching(left, size_left,
25  *     size_right);
26  * \end{lstlisting}
27  * Source: KACTL */
28
29 typedef vector<vector<int>> Graph;
30
31 vector<int> match;
32 vector<bool> visited;
33 template<class G>
34 bool find(int j, G &g) {
35     if (match[j] == -1) return true;
36     visited[j] = true; int di = match[j];
37     trav(e, g[di])
38     if (!visited[*e] && find(*e, g)) {
39         match[*e] = di;
40         match[j] = -1;
41         return true;
42     }
43     return false;
44 }
45
46 int dfs_matching(Graph &g, int n, int m)
47 {
48     match.assign(m, -1);
49     rep(i, 0, n) {

```

```

41     visited.assign(m, false);
42     trav(j, g[i])
43     if (find(*j, g)) {
44         match[*j] = i;
45         break;
46     }
47 }
48 return m - count(match.begin(), match.
49     end(), -1);
50 }

```

4.8 Strongly Connected Components

```

1  /* Name: Strongly Connected Components -
2   * Double DFS
3   * Description: Untested SCC algorithm.
4   * Calculates a new graph where all
5   * strongly connected components are
6   * merged. Does not require the graph
7   * to be connected.
8   * Source: Fredrik Svensson - 2009 */
9  struct vertex
10 {
11     vector<vertex*> from, to;
12     bool visited;
13 };
14 vector<vertex> v;
15 vector<vector<vertex*>> res;
16
17 vector<vertex*> sorted;
18 vector<vertex*>::reverse_iterator
19     visitIt;
20 vector<vertex*>* curRes;
21
22 void dfs(vertex* p)
23 {
24     if(p->visited) return;
25     p->visited = true;
26     if(curRes) curRes->push_back(p);
27     for(vector<vertex*>::iterator it
28         = p->to.begin();
29         it != p->to.end(); ++it)
30         dfs(*it);
31     *(visitIt++) = p;
32 }
33
34 void run()
35 {
36     sorted.resize(v.size());
37     visitIt = sorted.rbegin();

```

```

31     for(vector<vertex>::it it = v.
        begin());
32     it != v.end(); ++it)
33         it->visited = false;
34     for(vector<vertex>::it it = v.
        begin());
35     it != v.end(); ++it)
36         dfs(&(*it));
37     for(vector<vertex>::it it = v.
        begin());
38     it != v.end(); ++it)
39     {
40         it->visited = false;
41         it->from.swap(it->to);
42     }
43     for(vector<vertex>::iterator it
        = sorted.begin();
44         it != sorted.end(); ++it
        )
45         if(!(*it)->visited)
46         {
47             curRes = &(*res.
                insert(res.
                    end()));
48             dfs(&(*it));
49         }
50 }

```

5 String processing

5.1 STL

```

1  #include <string>
2
3  std::size_t found = str.find(str2);
4  if (found!=std::string::npos)
5      std::cout << "first_found_at:" <<
        found << '\n';
6
7  str.replace(str.find(str2),str2.length()
        ,"new_word");

```

5.2 String Matching

```

1  // Knuth Morris Prat : Search for a
    string in another one
2  // Alternative STL algorithms : strstr
    in <cstring> find in <string>
3  // Time complexity : O(n)
4
5  #include <cstdio>

```

```

6  #include <cstring>
7
8  #define MAXN 100010
9
10 char T[MAXN], P[MAXN]; // T = text, P
    = pattern
11 int b[MAXN], n, m; // b = back
    table, n = length of T, m = length of
    P
12
13 void kmpPreprocess() {
14     int i = 0, j = -1; b[0] = -1;
15     while (i < m){
16         while(j >= 0 && P[i] != P[j]) j = b[
            j];
17         i++; j++;
18         b[i] = j;
19     }
20 }
21
22 void kmpSearch() {
23     int i = 0, j = 0;
24     while(i < n){
25         while(j >= 0 && T[i] != P[j]) j = b[
            j];
26         i++; j++;
27         if(j==m){
28             printf("P_is_found_at_index_%d_in_
                T\n", i - j);
29             j = b[j];
30         }
31     }
32 }
33
34 int main(){
35     strcpy(T, "asdhasdhejasdasdhejasdasd")
        ;
36     strcpy(P, "hej");
37
38     n = 25; m = 3;
39
40     kmpPreprocess();
41     kmpSearch();
42
43     return 0;
44 }

```

5.3 String Multimatching

6 Geometry

6.1 Points Class

```

1  #include <cmath>
2
3  template<class T>
4  class Vector{
5  public:
6
7      T x, y;
8      Vector(){};
9      Vector(T a, T b){x = a; y = b};
10
11      T abs(){return sqrt(x*x+y*y);}
12      Vector operator* (T oth){ return
        Vector(x*oth, y*oth); }
13      Vector operator/ (T oth){ return
        Vector(x/oth, y/oth); }
14
15      Vector operator+ (Vector oth){ return
        Vector(x+oth.x, y+oth.y); }
16      Vector operator- (Vector oth){ return
        Vector(x+oth.x, y+oth.y); }
17      T operator* (Vector oth){ return x*oth
        .x + y*oth.y; }
18      Vector operator/ (Vector oth){ return
        Vector(x*oth.y-oth.x*y)}
19 };

```

6.2 Lines

```

1  /*
2   * Simple line segment representation.
3   */
4  template <class T>
5  class Line2 {
6  public:
7      typedef T Type;
8      typedef Point2<T> Point;
9
10     Point from, to;
11
12     Line2() {}
13     Line2(const Point &from, const Point &
        to) : from(from), to(to) {}
14
15     /*
16      * The line's direction, measured from
        "from".

```

```

17  */
18  Point dir() const { return to - from; }
19
20  /*
21   * The line's angle. This angle is the
22   * angle of "dir", relative the vector
23   * (1, 0).
24   */
25  double angle() const {
26      Point pt = dir();
27      return atan2(pt.y, pt.x);
28  }
29
30  /*
31   * Orthogonal projection of a point
32   * onto this line. Returns where on the
33   * line the point
34   * is located, 0 = from, 1 = to. May be
35   * outside this range.
36   */
37  double project(Point pt) const {
38      return double((pt - from) | dir()) /
39          lengthSq(dir());
40  }
41
42  /*
43   * Orthogonal projection of the two
44   * points of another line onto this line
45   * .
46   */
47  pair<double, double> project(Line2<T>
48      line) const {
49      return make_pair(project(line.from),
50          project(line.to));
51  }
52
53  };
54
55  /*
56   * Point - line-segment intersection.
57   */
58  template <class T>
59  bool intersects(Point2<T> point, Line2<T>
60      > line) {
61      if (line.from == line.to) {
62          if (line.from == point;
63          } else {
64              Point2<T> dir = line.to - line.from;
65              Point2<T> pt = point - line.from;
66              Point2<T> pt2 = point - line.to;

```

```

56      return (dir % pt) == 0 && (dir | pt)
57          >= 0 && (-dir | pt2) >= 0;
58  }
59  }
60
61  /*
62   * Distance between a point and a line
63   * segment.
64   */
65  template <class T>
66  double distance(Point2<T> point, Line2<T>
67      > line) {
68      if (line.from == line.to)
69          return distance(line.from, point);
70
71      /* Project pt unto the line. */
72      double projection = line.project(point)
73          ;
74
75      if (projection <= 0)
76          return distance(line.from, point);
77      else if (projection >= 1)
78          return distance(line.to, point);
79
80      Point2<double> proj = Point2<double>(
81          line.dir()) * projection;
82      return length(proj - Point2<double>(
83          point - line.from));
84  }
85
86  /*
87   * line-segment - line-segment
88   * intersection. Reports wether the
89   * lines intersect
90   * or not.
91   */
92  template <class T>
93  bool intersects(Line2<T> a, Line2<T> b)
94      {
95      typedef Point2<T> Pt;
96
97      Pt aDir = a.dir();
98      Pt bDir = b.dir();
99      Pt delta = a.from - b.from;
100
101      if (aDir == Pt() && bDir == Pt()) {
102          return a.from == b.from;
103      } else if (aDir == Pt()) {
104          /* We can manage if bDir == 0 */

```

```

97      return intersects(b, a);
98  }
99
100  T cross = aDir % bDir;
101  if (cross == 0) {
102      /* Parallel lines, project the two
103      points in B onto A and see where they
104      are located. */
105
106      /* Check if the lines are on the same
107      line. */
108      if (delta % aDir != 0)
109          return false;
110
111      double from = a.project(b.from);
112      double to = a.project(b.to);
113
114      double low = min(from, to);
115      double high = max(from, to);
116
117      return low <= 1 && high >= 0;
118  } else {
119      /* The lines intersect, where? */
120
121      /* Distance along bDir * cross */
122      T u = aDir % delta;
123      /* Distance along aDir * cross */
124      T v = bDir % delta;
125
126      return between(T(0), u, cross) &&
127          between(T(0), v, cross);
128  }
129  }
130
131  /*
132   * Helper to intersection() below. Adds
133   * two points, only if they are
134   * distinct.
135   */
136  template <class T>
137  inline void addPoints(vector<Point2<
138      double> > &result, const Point2<T> &a
139      , const Point2<T> &b) {
140      result << Point2<double>(a);
141      if (a != b) result << Point2<double>(b)
142          ;
143  }
144
145  /*

```

```

137 * Compute the intersection between two
    line segments.
138 * Returns zero, one or two points
    depending on if there is an
    intersection
139 * and if that intersections is a point
    or a line.
140 * If there are two points, this always
    returns the intersection points
141 * sorted along the direction of line a.
142 */
143 template <class T>
144 vector<Point2<double>> intersection(
    Line2<T> a, Line2<T> b) {
145     typedef Point2<T> Pt;
146     typedef Point2<double> PtD;
147
148     vector<PtD> result;
149
150     Pt aDir = a.dir(), bDir = b.dir();
151     Pt delta = a.from - b.from;
152
153     if (aDir == Pt() && bDir == Pt()) {
154         if (a.from == b.from)
155             result << PtD(a.from);
156         return result;
157     } else if (aDir == Pt()) {
158         return intersection(b, a);
159     }
160
161     T cross = aDir % bDir;
162     if (cross == 0) {
163         /* The lines are parallel. */
164
165         /* Are they in the same line? */
166         /* See if vectors from a.from to b.
            from is parallel with a.from to a.to
            */
167         if (delta % aDir == 0) {
168             double low = a.project(b.from);
169             double high = a.project(b.to);
170
171             Pt lowPt(b.from);
172             Pt highPt(b.to);
173
174             if (low > high) {
175                 swap(low, high);
176                 swap(lowPt, highPt);
177             }
178

```

```

179         if (b.from == b.to) {
180             if (low >= 0 && low <= 1)
181                 result << PtD(b.from);
182         } else {
183             if (low < 0 && high >= 0 && high
184                 <= 1)
185                 addPoints(result, a.from,
186                     highPt);
187             else if (low < 0 && high >= 1)
188                 addPoints(result, a.from, a.to)
189                 ;
190             else if (low >= 0 && high <= 1)
191                 addPoints(result, lowPt, highPt
192                     );
193             else if (low >= 0 && low <= 1)
194                 addPoints(result, lowPt, a.to);
195         }
196     }
197     } else {
198         /* Distance along bDir * cross */
199         T u = aDir % delta;
200         /* Distance along aDir * cross */
201         T v = bDir % delta;
202
203         if (between(T(0), u, cross) && between
204             (T(0), v, cross)) {
205             result << PtD(a.from) + PtD(aDir) * v
206                 / double(cross);
207         }
208     }
209     return result;
210 }
211
212 /*
213 * Distance between two line segments.
214 */
215 template <class T>
216 double distance(Line2<T> a, Line2<T> b)
217 {
218     if (intersects(a, b))
219         return 0.0;
220
221     /*
222     * Now that we know that the lines are
223     not intersecting, we can easily
224     * compute the shortest distance
225     between all endpoints on the lines
226     * to the other line.
227     */

```

```

220 double shortest =
221     min(min(distance(a.from, b),
222         distance(a.to, b)),
223         min(distance(b.from, a),
224             distance(b.to, a)));
225
226 return shortest;
227 }

```

6.3 Matrix Class

```

1 /* Description: Untested matrix
    implementation
2 * Source: Benjamin Ingberg */
3 template<typename T>
4 struct Matrix {
5     typedef Matrix<T> const & In;
6     typedef Matrix<T> M;
7
8     int r, c; // rows columns
9     vector<T> data;
10    Matrix(int r_, int c_, T v = T()) : r(
        r_),
11        c(c_), data(r_*c_, v) { }
12    explicit Matrix(Pt3<T> in)
13        : r(3), c(1), data(3*1) {
14        rep(i, 0, 3)
15            data[i] = in[i];
16    }
17    explicit Matrix(Pt2<T> in)
18        : r(2), c(1), data(2*1) {
19        rep(i, 0, 2)
20            data[i] = in[i];
21    }
22    // copy constructor, assignment
23    // and destructor compiler defined
24    T & operator()(int row, int col) {
25        return data[col+row*c];
26    }
27    T const & operator()(int row, int col)
28        const {
29        return data[col+row*c];
30    }
31    // implement as needed
32    bool operator==(In rhs) const {
33        return data == rhs.data;
34    }
35    M operator+(In rhs) const {
36        assert(rhs.r == r && rhs.c == c);
37        Matrix ret(r, c);

```

```

37     rep(i, 0, c*r)
38     ret.data[i] = data[i]*rhs.data[i];
39     return ret;
40 }
41 M operator-(In rhs) const {
42     assert(rhs.r == r && rhs.c == c);
43     Matrix ret(r, c);
44     rep(i, 0, c*r)
45     ret.data[i] = data[i]-rhs.data[i];
46     return ret;
47 }
48 M operator*(In rhs) const { // matrix
49     mult
50     assert(rhs.r == c);
51     Matrix ret(r, rhs.c);
52     rep(i, 0, r)
53     rep(j, 0, rhs.c)
54     rep(k, 0, c)
55     ret(i,j) += operator()(i, k)
56     *rhs(k,j);
57     return ret;
58 }
59 M operator*(T rhs) const { // scalar
60     mult
61     Matrix ret(*this);
62     trav(it, ret.data)
63     it = it*rhs;
64     return ret;
65 }
66 };
67
68 template<typename T> // create identity
69 matrix
70 Matrix<T> id(int r, int c) {
71     Matrix<T> m(r,c);
72     rep(i, 0, r)
73     m(i,i) = T(1);
74 }

```

6.4 Matrix3d Class

```

1  /* 3 dimensional matrix class
2   * with Gauss Elimination and
3   * Eigenvectors
4   * Source: Magnus Selin
5   */
6  #include <cmath>
7
8  class Matrix3d{

```

```

9      friend std::ostream& operator<< ( std
10      ::ostream& os, Matrix3d fb );
11  private:
12      double a[3][3];
13  public:
14      Matrix3d(){
15          for(int i = 0; i < 3; i++){
16              for(int j = 0; j < 3; j++){
17                  if(j >= 0 && j < 3 && i >= 0 &&
18                  i < 3) a[i][j] = 0;
19              }
20          }
21      }
22      double get(int i, int j) { if(j
23      >= 0 && j < 3 && i >= 0 && i < 3)
24      return a[i][j]; else std::cerr << "
25      Out of bounds!\n"; }
26      void set(int i, int j, int v) { if(j
27      >= 0 && j < 3 && i >= 0 && i < 3) a[i
28      ][j] = v; else std::cerr << "Out
29      of bounds!\n"; };
30
31      void chg_row(int x, int y);
32      void mult_row(int x, double c);
33      void add_row(int x, int y, double c);
34
35      Matrix3d gauss();
36      Matrix3d get_inverse();
37
38      double get_det();
39      void get_eigenvectors();
40
41      Matrix3d operator= (Matrix3d);
42      Matrix3d operator+ (Matrix3d);
43      Matrix3d operator- (Matrix3d);
44      Matrix3d operator* (Matrix3d);
45      Matrix3d operator* (double);
46  };
47
48 void Matrix3d::chg_row(int x, int y){
49     int temp[3] = {a[x][0], a[x][1], a[x]
50     ][2]};
51
52     for(int i = 0; i < 3; i++){
53         a[x][i] = a[y][i];
54         a[y][i] = temp[i];
55     }
56 }

```

```

57 void Matrix3d::mult_row(int x, double c)
58 {
59     for(int i = 0; i < 3; i++){
60         a[x][i] *= c;
61     }
62 }
63 void Matrix3d::add_row(int x, int y,
64     double c){
65     for(int i = 0; i < 3; i++){
66         a[x][i] += c * a[y][i];
67     }
68 }
69 void Matrix3d::get_eigenvectors(){
70     double eig[3];
71     double p = a[0][1] * a[0][1] + a[0][2]
72     * a[0][2] + a[1][2] * a[1][2];
73     double q, r, phi;
74     Matrix3d B; Matrix3d I;
75     for (int i = 0; i < 3; i++) I.set(i, i
76     , 1);
77
78     if ( p == 0 ){
79         eig[0] = a[0][0];
80         eig[1] = a[1][1];
81         eig[2] = a[2][2];
82     }
83     else {
84         q = (a[0][0] + a[1][1] + a[2][2]) /
85         3;
86         p = (a[0][0] - q) * (a[0][0] - q) +
87         (a[1][1] - q) * (a[1][1] - q) +
88         (a[2][2] - q) * (a[2][2] - q) + 2
89         * q;
90         p = sqrt( p / 6 );
91
92         B = ((*this) - I * q);
93         B = B * (1 / p);
94         r = B.get_det();
95
96         if (r <= -1)
97             phi = M_PI / 3;
98         else if (r >= 1)
99             phi = 0;
100        else
101            phi = acos(r) / 3;
102
103        eig[0] = q + 2 * p * cos(phi);
104        eig[2] = q + 2 * p * cos(phi + M_PI
105        * (2/3));

```

```

93     eig[1] = 3 * q - eig[0] - eig[2];
94 }
95
96 std::cout << eig[0] << '␣' << eig[1]
97     << '␣' << eig[2] << '␣';
98
99 for (int i = 0; i < 3; i++) {
100     Matrix3d temp = (*this);
101
102     temp.set(0, 0, temp.get(0, 0) - eig[
103         i]);
104     temp.set(1, 1, temp.get(1, 1) - eig[
105         i]);
106     temp.set(2, 2, temp.get(2, 2) - eig[
107         i]);
108     temp = temp.gauss();
109
110     std::cout << "Temp␣" << i << ":\n"
111         << temp << "\n";
112 }
113 }
114
115 double Matrix3d::get_det(){
116     return a[0][0] * a[1][1] * a[2][2] + a
117         [0][1] * a[1][2] * a[2][0] + a[0][2]
118         * a[1][0] * a[2][1] -
119         a[0][2] * a[1][1] * a[2][0] - a
120         [0][1] * a[1][0] * a[2][2] - a[0][0]
121         * a[1][2] * a[2][1];
122 }
123
124 Matrix3d Matrix3d::gauss(){
125     Matrix3d * temp = new Matrix3d;
126     temp = this;
127
128     for (int i = 0; i < 3; i++){
129         if(temp->get(i, i) == 0){
130             for (int j = i; j < 3; j++){
131                 if(temp->get(j, i) != 0){
132                     temp->chg_row(i, j);
133                     break;
134                 }
135             }
136
137             if(temp->get(i, i) == 0){
138                 std::cout << "Parameter␣solotion
139                     !!\n";
140                 break;
141             }
142         }
143     }

```

```

133     }
134
135     double mult_val = temp->get(i, i);
136     temp->mult_row(i, 1 / mult_val);
137
138     for (int j = 0; j < 3; j++){
139         if(i != j){
140             double mult_val = -temp->get(j,
141                 i);
142             temp->add_row(j, i, mult_val);
143         }
144     }
145 }
146
147 std::cout << "Temp␣" << ":\n" << *temp
148     << "\n";
149 std::cout << "This␣" << ":\n" << *this
150     << "\n";
151
152 return *temp;
153 }
154
155 Matrix3d Matrix3d::get_inverse(){
156     Matrix3d temp = (*this), inverse;
157     for (int i = 0; i < 3; i++) inverse.
158         set(i, i, 1);
159
160     for (int i = 0; i < 3; i++){
161         if(temp.get(i, i) == 0){
162             for (int j = i; j < 3; j++){
163                 if(temp.get(j, i) != 0){
164                     temp.chg_row(i, j);
165                     inverse.chg_row(i, j);
166
167                     std::cout << "Change␣row␣" <<
168                         i << "␣and␣" << j << ":\n";
169                     std::cout << temp << '\n';
170                     std::cout << inverse << '\n';
171
172                     break;
173                 }
174             }
175
176             if(temp.get(i, i) == 0){
177                 std::cout << "Singularity!\n";
178                 break;
179             }
180         }
181     }

```

```

178     double mult_val = temp.get(i, i);
179     temp.mult_row(i, 1 / mult_val);
180     inverse.mult_row(i, 1 / mult_val);
181
182     std::cout << "Divide␣row␣" << i << "
183         ␣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             double mult_val = -temp.get(j, i
190                 );
191
192             temp.add_row(j, i, mult_val);
193             inverse.add_row(j, i, mult_val);
194
195             std::cout << "Multiply␣row␣" <<
196                 i << "␣by␣" << mult_val << "␣and␣
197                 adding␣it␣to␣" << j << ":\n";
198             std::cout << temp << '\n';
199             std::cout << inverse << '\n';
200         }
201     }
202
203     return inverse;
204 }
205
206 Matrix3d Matrix3d::operator= (Matrix3d
207     param){
208     Matrix3d temp;
209     for(int i = 0; i < 3; i++){
210         for(int j = 0; j < 3; j++){
211             temp.set(i, j, param.get(i, j));
212         }
213     }
214
215     return temp;
216 }
217
218 Matrix3d Matrix3d::operator+ (Matrix3d
219     param){
220     Matrix3d temp;
221     for(int i = 0; i < 3; i++){
222         for(int j = 0; j < 3; j++){
223             temp.set(i, j, a[i][j] + param.get
224                 (i, j));
225         }
226     }

```

```

221     }
222
223     return temp;
224 }
225 Matrix3d Matrix3d::operator- (Matrix3d
    param){
226     Matrix3d temp;
227     for(int i = 0; i < 3; i++){
228         for(int j = 0; j < 3; j++){
229             temp.set(i, j, a[i][j] - param.get
                (i, j));
230         }
231     }
232
233     return temp;
234 }
235 Matrix3d Matrix3d::operator* (double
    param){
236     Matrix3d temp;
237     for(int i = 0; i < 3; i++){
238         for(int j = 0; j < 3; j++){
239             temp.set(i, j, a[i][j] * param);
240         }
241     }
242
243     return temp;
244 }
245 Matrix3d Matrix3d::operator* (Matrix3d
    param){
246     Matrix3d temp;
247
248     for(int i = 0; i < 3; i++){
249         for(int j = 0; j < 3; j++){
250             temp.set(i, j, a[i][0] * param.get
                (0,j) +
251                 a[i][1] * param.get(1,j) +
252                 a[i][2] * param.get(2,j) )
                ;
253         }
254     }
255
256     return temp;
257 }
258
259 std::ostream& operator << ( std::ostream
    & os, Matrix3d m ){
260     for(int i = 0; i < 3; i++){
261         os << "(";
262         for(int j = 0; j < 3; j++){
263             os << m.get(i, j) << ",\t";

```

```

264         }
265         os << ")\n";
266     }
267     return os;
268 }

```

6.5 Points Class

```

1  /* Description: Untested homogenous
    coordinates
2  * transformation geometry.
3  * Source: Benjamin Ingberg
4  * Usage: Requires homogenous
    coordinates, handles
5  * multiple rotations, translations and
    scaling in a
6  * high precision efficient manner (
    matrix
7  * multiplication) with homogenous
    coordinates.
8  * Also keeps reverse transformation
    available. */
9  namespace h { // avoid name collisions
10     struct Transform {
11         enum ActionType {
12             Scale, Rotate, TranslateX, TranslateY
13         };
14         typedef tuple<ActionType, fp> Action;
15         typedef Matrix<fp> M;
16         typedef vector<Action> History;
17         History hist;
18         M to, from;
19         Transform(History h = History())
20             : to(id<fp>(3,3)), from(id<fp>(3,3))
                {
21             doTransforms(h);
22         }
23         H transformTo(H in) {
24             return H(to*M(in));
25         }
26         H transformFrom(H in) {
27             return H(from*M(in));
28         }
29         Transform & scale(fp s) {
30             doTransform(Scale, s);
31         }
32         Transform & translate(fp dx, fp dy) {
33             doTransform(TranslateX, dx);
34             doTransform(TranslateY, dy);
35         }

```

```

36     Transform & rotate(fp phi) {
37         doTransform(Rotate, phi);
38     }
39     void doTransforms(History & h) {
40         trav(it, h) {
41             doTransform(get<0>(*it), get<1>(*it)
                );
42         }
43     }
44     void doTransform(ActionType t, fp v) {
45         hist.push_back(make_tuple(t, v));
46         if(t == Scale)
47             doScale(v);
48         else if(t == TranslateX)
49             doTranslate(0,v);
50         else if(t == TranslateY)
51             doTranslate(1,v);
52         else
53             doRotate(v);
54     }
55 private:
56     void doScale(fp s) {
57         M sm(id<fp>(3,3)), ism(id<fp>(3,3));
58         sm(1,1) = sm(0,0) = s;
59         ism(1,1) = ism(1,1) = 1/s;
60         to = to*sm; from = ism*from;
61     }
62     void doTranslate(int c, fp dx) {
63         M sm(id<fp>(3,3)), ism(id<fp>(3,3));
64         sm(c,2) = dx;
65         ism(c,2) = -dx;
66         to = to*sm; from = ism*from;
67     }
68     void doRotate(fp phi) {
69         M sm(id<fp>(3,3)), ism(id<fp>(3,3));
70         sm(0,0) = sm(1,1) = cos(phi);
71         ism(0,0) = ism(1,1) = cos(-phi);
72         sm(1,0) = sm(0,1) = sin(phi);
73         ism(0,1) = sm(1,0) = sin(-phi);
74         to = to*sm; from = ism*from;
75     }
76 }
77 }

```

6.6 Graham Scan

```

1  struct point {
2      int x, y;
3  };

```



```

4  int det(const point& p1, const point& p2
      , const point& p3)
5  {
6      int x1 = p2.x    p1.x;
7      int y1 = p2.y    p1.y;
8      int x2 = p3.x    p1.x;
9      int y2 = p3.y    p1.y;
10     return x1*y2    x2*y1;
11 }
12
13 // bool ccw(const point& p1, const point
      & p2, const point& p3)
14 // { // Counterclockwise? Compare with
      determinant...
15 // return (det(p1, p2, p3) > 0);
16 // }
17
18 struct angle_compare {
19     point p; // Leftmost lower point
20     angle_compare(const point& p) : p(p) {
21     }
22     bool operator()(const point& lhs, const
        point& rhs) {
23         int d = det(p, lhs, rhs);
24         if(d == 0) // Furthest first if same
            direction will keep all
25         return (x1*x1+y1*y1 > x2*x2+y2*y2);
26         // points at the line
27         return (d > 0); // Counterclockwise?
28     }
29 };
30
31 int ConvexHull(const vector<point>& p,
        int* res)
32 { // Returns number of points in the
        convex polygon
33     int best = 0; // Find the first
        leftmost lower point
34     for(int i = 1; i < p.size(); ++i)
35     {
36         if(p[i].y < p[best].y ||
37             (p[i].y == p[best].y && p[i].x
38              < p[best].x))
39             best = i;
40     }
41     sort(p.begin(), p.end(), angle_compare(
        p[best]));
42     for(int i = 0; i < 3; ++i)
43         res[i] = i;
44     int n = 3;

```

```

42     for(int i = 3; i < p.size(); ++i)
43     {
44         // All consecutive points should be
            counter clockwise
45         while(n > 2 && det(res[n-2], res[n
            -1], i) < 0)
46             --n; // Keep if det = 0, i.e.
                the same line, angle_compare
47         res[n++] = i;
48     }
49     return n;
50 }

```

6.7 Convex Hull

```

1  #include <iostream>
2  #include <cstdio>
3  #include <vector>
4  #include <cmath>
5  #include <algorithm>
6
7  using namespace std;
8
9  typedef unsigned int nat;
10
11 template <class T>
12 struct Point {
13     T x, y;
14
15     Point(T x = T(), T y = T()) : x(x), y(y)
        {}
16
17     bool operator <(const Point<T> &o)
        const {
18         if (y != o.y) return y < o.y;
19         return x < o.x;
20     }
21
22     Point<T> operator -(const Point<T> &o)
        const { return Point<T>(x - o.x, y -
        o.y); }
23     Point<T> operator +(const Point<T> &o)
        const { return Point<T>(x + o.x, y +
        o.y); }
24
25     T lenSq() const { return x*x + y*y; }
26 };
27
28 template <class T>
29 struct sort_less {

```

```

30     const Point<T> &ref;
31
32     sort_less(const Point<T> &p) : ref(p)
        {}
33
34     double angle(const Point<T> &p) const {
35         Point<T> delta = p - ref;
36         return atan2(delta.y, delta.x);
37     }
38
39     bool operator() (const Point<T> &a,
        const Point<T> &b) const {
40         double aa = angle(a);
41         double ab = angle(b);
42         if (aa != ab) return aa < ab;
43         return (a - ref).lenSq() < (b - ref).
            lenSq();
44     }
45 };
46
47 template <class T>
48 int ccw(const Point<T> &p1, const Point<
    T> &p2, const Point<T> &p3) {
49     return (p2.x - p1.x) * (p3.y - p1.y) -
        (p2.y - p1.y) * (p3.x - p1.x);
50 }
51
52 template <class T>
53 vector<Point<T> > convex_hull(vector<
    Point<T> > input) {
54     if (input.size() < 2) return input;
55     nat size = input.size();
56
57     vector<Point<T> > output;
58
59     // Find the point with the lowest x and
        y value.
60     int minIndex = 0;
61     for (int i = 1; i < size; i++) {
62         if (input[i] < input[minIndex]) {
63             minIndex = i;
64         }
65     }
66
67     // This is the "root" point in our
        traversal.
68     Point<T> p = input[minIndex];
69     output.push_back(p);
70     input.erase(input.begin() + minIndex);
71

```



```

72 // Sort the other elements according to
   the angle with "p"
73 sort(input.begin(), input.end(),
       sort_less<T>(p));
74
75 // Add the first point from "input" to
   the "output" as a candidate.
76 output.push_back(input[0]);
77
78 // Start working our way through the
   points...
79 input.push_back(p);
80 size = input.size();
81 for (nat i = 1; i < size; i++) {
82     while (output.size() >= 2) {
83         nat last = output.size() - 1;
84         int c = ccw(output[last - 1], output[
            last], input[i]);
85
86         if (c == 0) {
87             // Colinear points! Take away
               the closest.
88             if ((output[last - 1] - output[
                last]).lenSq() <= (output[
                    last - 1] - input[i]).lenSq()) {
89                 if (output.size() > 1)
90                     output.pop_back();
91                 else
92                     break;
93             } else {
94                 break;
95             }
96         } else if (c < 0) {
97             if (output.size() > 1)
98                 output.pop_back();
99             else
100                 break;
101         } else {
102             break;
103         }
104     }
105
106     // Do not take the last point twice.
107     if (i < size - 1)
108         output.push_back(input[i]);
109 }
110
111 return output;
112 }

```

```

113
114
115 typedef Point<int> Pt;
116
117 bool solve() {
118     nat count;
119     scanf("%d", &count);
120
121     if (count == 0) return false;
122
123     vector<Pt> points(count);
124     for (nat i = 0; i < count; i++) {
125         scanf("%d%d", &points[i].x, &points[i
            ].y);
126     }
127
128     vector<Pt> result = convex_hull(points)
        ;
129
130     printf("%d\n", (int)result.size());
131     for (nat i = 0; i < result.size(); i++)
132         printf("%d%d\n", result[i].x, result[
            i].y);
133 }
134
135 return true;
136 }
137
138 int main() {
139     while(solve());
140
141     return 0;
142 }

```

6.8 Line-point distance

```

1 // Problem 12173 on UVa (accepted there)
2
3 #include <cstdio>
4 #include <vector>
5 #include <cmath>
6 #include <iostream>
7
8 using namespace std;
9
10 typedef unsigned int nat;
11
12 template <class T>
13 class Point {

```

```

14 public:
15     T x, y;
16
17     Point() : x(), y() {}
18     Point(T x, T y) : x(x), y(y) {}
19
20     Point<T> operator -(const Point &o)
        const { return Point<T>(x - o.x, y -
            o.y); }
21     Point<T> operator /(T o) const { return
        Point<T>(x / o, y / o); }
22     T operator |(const Point &o) const {
23         return x * o.x + y * o.y;
24     }
25 };
26
27
28 template <class T>
29 class Vector {
30 public:
31     T x, y, z;
32
33     Vector() : x(), y(), z() {}
34     Vector(const Point<T> &pt, T z) : x(pt.
        x), y(pt.y), z(z) {}
35     Vector(T x, T y, T z) : x(x), y(y), z(z
        ) {}
36
37     Vector<T> operator -(const Vector &o)
        const { return Vector<T>(x - o.x, y -
            o.y, z - o.z); }
38     Vector<T> operator /(T o) const {
39         return Vector<T>(x / o, y / o, z / o
            ); }
40     T operator |(const Vector &o) const {
41         return x * o.x + y * o.y + z * o.z; }
42     Vector<T> operator %(const Vector &o)
        const {
43         return Vector<T>(y*o.z - z*o.y, z*o.x
            - x*o.z, x*o.y - y*o.x);
44     }
45 };
46
47 // distance between two points or
   vectors.
48 template <class T>
49 T dist(const Point<T> &a, const Point<T>
    &b) {
50     Point<T> d = a - b;
51     return sqrt(d | d);

```

```

50 }
51
52 // Normalize a line
53 template <class T>
54 void normLine(Vector<T> &v) {
55     T l = sqrt(v.x * v.x + v.y * v.y);
56     v = v / l;
57 }
58
59 // Normalize a point
60 template <class T>
61 void normPoint(Vector<T> &v) {
62     v = v / v.z;
63 }
64
65 template <class T>
66 T dist(const Point<T> &point, const
        Point<T> &lineFrom, const Point<T> &
        lineTo) {
67     // Outside first endpoint?
68     if (((point - lineFrom) | (lineTo -
        lineFrom)) < 0) {
69         return dist(point, lineFrom);
70     }
71
72     // Outside second endpoint?
73     if (((point - lineTo) | (lineFrom -
        lineTo)) < 0) {
74         return dist(point, lineTo);
75     }
76
77     // Ok, in the middle of the line!
78
79     // Create the homogenous representation
        of the line...
80     Vector<T> line = Vector<T>(lineFrom, 1)
        % Vector<T>(lineTo, 1);
81
82     // The signed distance is then the dot
        product of the line
83     // and the point.
84     normLine(line);
85     T distance = Vector<T>(point, 1) | line
        ;
86
87     // Don't return negative distances...
88     return abs(distance);
89 }
90
91 vector<Point<double>> readPoints() {

```

```

92     nat size = 0;
93     scanf("%d", &size);
94
95     vector<Point<double>> result;
96
97     for (nat i = 0; i < size; i++) {
98         double x, y;
99         scanf("%lf%lf", &x, &y);
100         result.push_back(Point<double>(x, y));
101     }
102
103     return result;
104 }
105
106 void solve() {
107     vector<Point<double>> inner =
        readPoints();
108     vector<Point<double>> outer =
        readPoints();
109
110     double longest = 1e100;
111
112     for (nat i = 0; i < inner.size(); i++)
113     {
114         nat iNext = (i + 1) % inner.size();
115         for (nat j = 0; j < outer.size(); j++)
116         {
117             nat jNext = (j + 1) % outer.size();
118
119             longest = min(longest, dist(outer[j],
120             inner[i], inner[iNext]));
121             longest = min(longest, dist(inner[i],
122             outer[j], outer[jNext]));
123         }
124     }
125
126     printf("%.8lf\n", longest / 2.0);
127 }
128
129 int main() {
130     int tc;
131     scanf("%d", &tc);
132
133     while (tc--) solve();
134
135     return 0;
136 }

```

6.9 Polygon Area

```

1  /* Calculate the area of an arbitrary
    polygon
2  * <vector> and "geometry.cpp" must be
    included
3  * source: Magnus Selin
4  */
5
6  template <class T>
7  int area(vector<Vector<T>> v){
8      int area = 0;
9      for(int i = 0; i < v.size()-1; i++)
10          area += (v[i] % v[i+1]).z;
11      area += (v[v.size()-1] % v[0]).z;
12      return area;
13 }

```

7 Misc

7.1 Longest Increasing Subsequence

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <vector>
4  #include <algorithm>
5
6  using namespace std;
7  int bin_search(int a[], int t[], int l,
    int r, int k) {
8      int m;
9      while( r - l > 1 ) {
10         m = l + (r - l)/2;
11         if( a[t[m]] >= k )
12             r = m;
13         else
14             l = m;
15     }
16     return r;
17 }
18
19 vector<int> lis(int a[], int n){
20     std::vector<int> lis;
21     if(n == 0) return lis;
22     int c[n]; memset(c, 0, sizeof(c));
23     int p[n]; memset(p, 0xFF, sizeof(p));
24     int s = 1;
25
26     c[0] = 0;
27     p[0] = -1;
28     for(int i = 1; i < n; i++){
29         if(a[i] < a[c[0]]){

```

```

30     c[0] = i;
31 }
32 else if(a[i] > a[c[s-1]]){
33     p[i] = c[s-1];
34     c[s] = i;
35     s++;
36 }
37 else{
38     int pos = bin_search(a, c, -1, s
39 -1, a[i]);
40     p[i] = c[pos-1];
41     c[pos] = i;
42 }
43 }
44
45 int d = c[s-1];
46 for( int i = 0; i < s; i++){
47     lis.push_back(d);
48     d = p[d];
49 }
50
51
52 reverse(lis.begin(), lis.end());
53 return lis;
54 }
55 int main(){
56     int n;
57     while(scanf("%d", &n) == 1){
58         int a[n]; for(int i = 0; i < n; i++){
59             scanf("%d", &a[i]);
60             vector<int> lseq = lis(a, n);
61
62             printf("%d\n", (int)lseq.size());
63             for(int i = 0; i < lseq.size(); i++){
64                 printf("%d ", lseq[i]);
65             }
66             printf("\n");
67
68             lseq.clear();
69         }
70     }

```

7.2 Longest Increasing Substring

```

1  /* Longest common substring. */
2  int HadenIngberg(string const & s,
3                  string const & t){
4      int n = s.size(), m = t.size(), best;

```

```

4      for(int i = 0; i < n-best; ++i) { //
5          Go through s
6          int cur = 0;
7          int e = min(n-i, m);
8
9          // Can best grow?
10         for(int j = 0; j < e && best+j < cur
11             +e; ++j)
12             best = max(best,
13                 cur = (s[i+j] == t[j] ? cur+1 : 0))
14             ;
15     }
16     for(int i = 1; i < m-best; ++i) { //
17         Go through t
18         int cur = 0;
19         int e = min(m-i, n);
20         // Can best grow?
21         for(int j = 0; j < e && best+j < cur
22             +e; ++j)
23             best = max(best, cur=(t[i+j] == s[j]?
24                 cur+1:0));
25     }
26     return best;
27 }

```

7.3 Knapsack

```

1  /*
2   * Magnus Selin 2013-06-02
3   * AAPS LiU
4   * Task 1.2: Knapsack
5   */
6
7  #include <iostream>
8  #include <vector>
9  #include <string.h>
10
11 using namespace std;
12
13 typedef vector<int> vi;
14
15 vi knapsack(int C, int n, int * v, int *
16             w){
17     int m[n+1][C+1]; memset(m[0], 0,
18         sizeof(m[0]));
19     bool k[n+1][C+1]; memset(k, 0, sizeof(
20         k));
21 }

```

```

19 // Starting with the first item, then
20 // the first and second etc. until all
21 // items are included
22 for (int i = 1; i <= n; i++){
23     // Starting with a bag of size 0
24     // then 1 up to C
25     for (int j = 0; j <= C; j++){
26         // If you chose to pack item i-1
27         // or stay with the ones before
28         if (j >= w[i-1]){
29             if(m[i-1][j] >= m[i-1][j-w[i-1]]
30                 + v[i-1]){
31                 m[i][j] = m[i-1][j];
32                 k[i][j] = false;
33             }
34             else{
35                 m[i][j] = m[i-1][j-w[i-1]] + v
36                 [i-1];
37                 k[i][j] = true;
38             }
39         }
40         else{
41             m[i][j] = m[i-1][j];
42             k[i][j] = false;
43         }
44     }
45 }
46
47 for(int i = 0; i < n+1; i++){
48     for(int j = 0; j < C+1; j++){
49         cout << m[i][j] << ' ';
50     }
51     cout << '\n';
52 }
53
54 cout << '\n';
55 for(int i = 0; i < n+1; i++){
56     for(int j = 0; j < C+1; j++){
57         cout << k[i][j] << ' ';
58     }
59     cout << '\n';
60 }
61
62 int c = (int)C;
63 vi obj;
64 // Get the index of which items too
65 // pack
66 for(int i = n; i > 0; i--){
67     if(k[i][c]){
68         obj.push_back(i-1);
69         c -= w[i-1];
70     }
71 }

```

```

62         if(c <= 0) break;
63     }
64 }
65 return obj;
66 }
67
68 int main(){
69     double C; int n;
70     while(cin >> C >> n){

```

```

71     int v[n], w[n];
72     for(int i = 0; i < n; i++) cin >> v[
73         i] >> w[i];
74
75     vi obj = knapsack(C, n, v, w);
76
77     cout << obj.size() << '\n';
78     for (int i = 0; i < obj.size(); i++)
79     {

```

```

78         cout << obj[i] << '␣';
79     }
80     cout << "\n";
81 }
82
83 return 0;
84 }

```