

NAT: Nostalgic Alien Trespassers — TCR NWERC 2013

Filip Strömbäck, Magnus Selin, Carl Einarson

November 24, 2013

Contents

1 Environment	2		
1.1 Template	2		
2 Data Structures	2		
2.1 Union Find	2		
2.2 Fenwick Tree	2		
3 Numerical	2		
3.1 General Utils	2		
3.2 Rational Numbers Class	3		
3.3 Binary Search	3		
3.4 De Bruijn	3		
3.5 Prime Generator	4		
3.6 Factorisation	4		
4 Graphs	4		
4.1 Single Source Shortest Path	4		
4.2 Single Source Shortest Path Time Table	4		
4.3 All Pairs Shortest Path	5		
4.4 Minimum Spanning Tree	5		
4.5 Maximum Flow	6		
		4.6 Euler Tour	6
		4.7 Bipartite Matching	6
		4.8 Strongly Connected Components	6
		5 String processing	7
		5.1 STL	7
		5.2 String Matching	7
		5.3 String Multimatching	7
		6 Geometry	7
		6.1 Points Class	7
		6.2 Matrix Class	7
		6.3 Matrix3d Class	8
		6.4 Points Class	9
		6.5 Graham Scan	9
		6.6 Convex Hull	10
		6.7 Line-point distance	10
		6.8 Polygon Area	11
		7 Misc	11
		7.1 Longest Increasing Subsequence	11
		7.2 Longest Increasing Substring	12

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 ll;
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 }
```

```
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("%c[%u\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 void fenwick_create(vi &t, lli n){
14     t.assign(n + 1, 0);
15 }
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("%c[%u\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             case '?':
46                 scanf("%lld", &a);
47                 printf("%lld\n", fenwick_read(ft, a));
48                 break;
49         }
50     }
51 }
52
53 return 0;
54 }
```

```
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); // 0(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 stre
10 // eller lika alla element i [first, nth[
11 // och *nth r mindre eller lika alla element i ]nth, last[.
12 // Komplexitet: 0(n) i medeltal
13 // BidIt partition(BidIt first, BidIt last, Pred pr); // 0(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]); // 0(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]); // 0(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 NOTFOUND;
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), (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     return 0;
88 }

```

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 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             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)]
35                 * pow((double)2, n-i-1);
36         }
37         cout << sum << '\n';
38     }
39 }
40
41 int main(){
42     int tc;
43     cin >> tc;
44     for(int we = 0; we < tc; we++){
45         cin >> n >> k;
46         a.clear(); seq.clear();
47         de_bruijn();
48     }
49 }

```

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[numprimes]
8          = 46340; // 0xb504*0xb504 = 0x7FEEA810
9      for(int n = 3; n < maxn; n += 2) {
10         for(int i = 1; prime[i]*prime[i] <= n; ++i)
11             if(n % prime[i] == 0) goto not_prime;
12         prime[numprimes++] = n; prime[numprimes] =
13             46340; // 0xb504*0xb504 = 0x7FEEA810
14     not_prime:
15     ;
16 }
17
18 int main(){
19     calcprimes(1000000);
20     for(int i = 0; i < 664579; i++) printf("%d\n",
21         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); numfactors++;
15     }
16 }
17 }

```

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 1000000000
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, const
16         pair<int, T> &b){return (a.second > b.
17             second);}
18 };
19
20 template<class T>
21 vector<T> dijkstras(vector<pair<int, T> > G[],
22     int n, int e, int s){
23     priority_queue<pair<int, T>, vector<pair<int,
24         T>, comp> Q;
25
26     vector<T> c; for(int i = 0; i < n; i++) c.
27         push_back(INF); c[s] = 0;
28     vector<int> p; for(int i = 0; i < n; i++) p.
29         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         u = Q.top().first; Q.pop();
35         sz = G[u].size();
36         for(int i = 0; i < sz; i++){
37             v = G[u][i].first;
38             w = G[u][i].second;
39             if(c[v] > c[u] + w){
40                 c[v] = c[u] + w;
41                 p[v] = u;
42                 Q.push(pair<int, T>(v, c[v]));
43             }
44         }
45     }
46 }
47
48 //printf("Path to follow: ");
49 //for(int i = 0; i < n; i++) printf("%d ", p[
50     i]);
51 //printf("\n");
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

```

```

46     return c;
47 }
48
49 int main(){
50     int n, e, q, s;
51     scanf("%d%d%d", &n, &e, &q, &s);
52     while(n!=0 or e!=0 or q!=0 or s!=0){
53         vector<ii> G[n];
54         for(int i = 0; i < e; i++){
55             int f, t, w;
56             scanf("%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         for(int i = 0; i < q; i++) {
62             int d; scanf("%d", &d);
63             if(c[d] == INF) printf("Impossible\n");
64             else printf("%d\n", c[d]);
65         }
66         printf("\n");
67
68         scanf("%d%d%d", &n, &e, &q, &s);
69     }
70
71     return 0;
72 }

```

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 1000000000
6
7  using namespace std;
8
9  struct A{
10     A(int a, int b, int c){t0=a; tn = b; w = c;}
11     int t0, tn, w;
12 };
13
14 typedef pair<int, int> ii;
15 typedef pair<int, A> iA;
16
17 class comp{
18 public:
19     int operator()(const ii&a, const ii&b){
20         return (a.second > b.second);}
21 };
22
23 vector<int> dijkstras(vector<iA> G[], int n,
24     int e, int s){
25     priority_queue<ii, vector<ii>, comp> Q;
26
27     vector<int> c; for(int i = 0; i < n; i++) c.
28         push_back(INF); c[s] = 0;
29     vector<int> p; for(int i = 0; i < n; i++) p.
30         push_back(-1);
31
32     Q.push(ii(s, c[s]));
33     int u, sz, v, t0, tn, w, wt;
34     while(!Q.empty()){
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             wt = G[u][i].second;
40             if(c[v] > c[u] + wt){
41                 c[v] = c[u] + wt;
42                 p[v] = u;
43                 Q.push(ii(v, c[v]));
44             }
45         }
46     }
47 }
48
49 //printf("Path to follow: ");
50 //for(int i = 0; i < n; i++) printf("%d ", p[
51     i]);
52 //printf("\n");
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

```

```

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[
54     i]);
55 //printf("\n");
56
57 return c;
58 }
59
60 int main(){
61     int n, e, q, s;
62     scanf("%d%d%d%d", &n, &e, &q, &s);
63     while(n!=0 or e!=0 or q!=0 or s!=0){
64         vector<iA> G[n];
65         for(int i = 0; i < e; i++){
66             int f, t, t0, tn, w;
67             scanf("%d%d%d%d%d", &f, &t, &t0, &tn, &w)
68             ;
69             G[f].push_back(iA(t, A(t0, tn, w)));
70         }
71         vector<int> c = dijkstras(G, n, e, s);
72
73         for(int i = 0; i < q; i++) {
74             int d; scanf("%d", &d);
75             if(c[d] == INF) printf("Impossible\n");
76             else printf("%d\n", c[d]);
77         }
78         printf("\n");
79
80         scanf("%d%d%d%d", &n, &e, &q, &s);
81     }
82     return 0;
83 }

```

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).
2 // Assign nodes which are part of a
3 // negative cycle to minus infinity.
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<
14     vector< T> > d){
15     int n = d.size();
16     for(int i = 0; i < n; i++) d[i][i] = 0;
17
18     for (int k = 0; k < n; k++)
19         for (int i = 0; i < n; i++)
20             for (int j = 0; j < n; j++)
21                 if (d[i][k] != INF and d[k][j] != INF)
22                     d[i][j] = min(d[i][j], d[i][k]+d[k][j]);
23
24     for(int i = 0; i < n; i++)
25         for(int j = 0; j < n; j++)
26             for(int k = 0; k < n; k++)
27                 if(d[i][k] != INF && d[k][j] != INF &&
28                     d[k][k] < 0)
29                     d[i][j] = -INF;
30
31     return d;
32 }
33
34 int main(){
35     int n, m, q; scanf("%d%d%d", &n, &m, &q);
36     while(n!=0 or m!=0 or q!=0){
37         vector< vector<int> > d;
38         d.resize(n);
39         for(int i = 0; i < n; i++)
40             for(int j = 0; j < n; j++)
41                 d[i].push_back(INF);
42
43         for(int i = 0; i < m; i++){
44             int f, t, w; scanf("%d%d%d", &f, &t, &w);
45             d[f][t] = min(w, d[f][t]);
46         }
47
48         d = floyd_warshall(d, n);
49         for(int i = 0; i < q; i++){
50             int f, t; scanf("%d%d", &f, &t);
51             if(d[f][t] == INF) printf("Impossible\n");
52             else if(d[f][t] == -INF) printf("-
53                 Infinity\n");
54             else printf("%d\n", d[f][t]);
55         }
56         printf("\n");
57         scanf("%d%d%d", &n, &m, &q);
58     }
59     return 0;
60 }

```

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) const{
10         if(f == oth.f)
11             return(t < oth.t);
12         return(f < oth.f);
13     }
14
15     AnsEdge(){};
16     AnsEdge(int a, int b){f = a; t = b;};
17 };
18
19 struct Tree{
20     int w;
21     bool complete;
22     std::vector<AnsEdge> e;
23     Tree(){
24         w = 0;
25         complete = true;
26     }
27 };
28
29 struct Vertex{
30     Vertex *p;
31     Vertex *root() {
32         if(p->p != p)
33             p = p->root();
34         return p;
35     }
36 };
37
38 struct Edge{
39     int f, t, w;
40
41     bool operator<(const Edge& oth) const{
42         if(w == oth.w)
43             return(t < oth.t);
44         return(w < oth.w);
45     }
46 };
47
48 Tree kruskal(Vertex * v, Edge * e, int numv,
49     int nume){
50     Tree ans;
51     int sum = 0;
52
53     for(int i = 0; i < numv; ++i){
54         v[i].p = &v[i];
55     }
56
57     sort(&e[0], &e[nume]);
58
59     for(int i = 0; i < nume; ++i){
60         if(v[e[i].f].root() != v[e[i].t].root()){
61             v[e[i].t].root()->p = v[e[i].f].root();
62             ans.w += e[i].w;
63
64             if(e[i].t < e[i].f) ans.e.push_back(
65                 AnsEdge(e[i].t, e[i].f));
66             else ans.e.push_back(AnsEdge(e[i].
67                 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 }

```

```

74     sort(ans.e.begin(), ans.e.end());
75
76     return ans;
77 }
78
79 int main(){
80     int n, m; scanf("%d%d", &n, &m);
81     while(n or m){
82         Vertex v[n];
83         Edge e[m];
84
85         for(int i = 0; i < m; i++){
86             int f, t;
87             scanf("%d%d", &f, &t, &e[i].w);
88             e[i].f = f;
89             e[i].t = t;
90         }
91
92         Tree ans = mst(v, e, n, m);
93
94         if(ans.complete){
95             printf("%d\n", ans.w);
96             for(int i = 0; i < ans.e.size(); i++){
97                 printf("%d␣%d\n", ans.e[i].f, ans.e[i].
98                     t);
99             }
100         } else printf("Impossible\n");
101
102         scanf("%d%d", &n, &m);
103     }
104
105     return 0;
106 }

```

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[v]] +=
9              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; queue<int>
18             > 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] == INF)
25                     dist[v] = dist[u] + 1, q.push(v), p[v]
26                         = u;

```

```

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, int u) {
13     for(int j = 0; j < (int)AdjList[u].size(); j
14         ++){
15         ii v = AdjList[u][j];
16         if (v.second){
17             v.second = 0;
18             for(int k = 0; k < (int)AdjList[u].size()
19                 ; k++){
20                 ii uu = AdjList[v.first][k];
21                 if(uu.first == u && uu.second) {uu.
22                     second = 0; break;}
23             }
24             euler_tour(cyc.insert(i, u), v.first)
25         }
26     }
27 }
28
29 int main(){
30     cyc.clear();
31     euler_tour(cyc.begin(), A);
32     for(list<int>::iterator it = cyc.begin(); it
33         != cyc.end(); it++){
34         printf("%d\n", *it);
35     }

```

4.7 Bipartite Matching

```

1  /* Name: Bipartite DFS
2  * Description: Simple bipartite matching.
3  * Slower than HopcroftKarp but shorter.
4  * Graph g should be a list of neighbours
5  * of the left partition.
6  * n is the size of the left partition
7  * and m is the size of the right partition.
8  * If you want to get the matched pairs,
9  * \lstinline|match[i]| contains match for
10     vertex i on
11     * the right side or -1 if it's not matched.
12     * Time:  $O(EV)$ 
13     * Usage example:
14     * \begin{lstlisting}[frame=none, aboveskip
15         =-0.6cm, ]

```

```

14 * Graph left(n);
15 * trav(it, edges){
16 *     l[it->left].push_back(it->right);
17 * }
18 * dfs_matching(left, size_left, size_right);
19 * \end{lstlisting}
20 * Source: KACTL */
21
22 typedef vector<vector<int>> Graph;
23
24 vector<int> match;
25 vector<bool> visited;
26 template<class G>
27 bool find(int j, G &g) {
28     if (match[j] == -1) return true;
29     visited[j] = true; int di = match[j];
30     trav(e, g[di])
31     if (!visited[*e] && find(*e, g)) {
32         match[*e] = di;
33         match[j] = -1;
34         return true;
35     }
36     return false;
37 }
38 int dfs_matching(Graph &g, int n, int m) {
39     match.assign(m, -1);
40     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.end(),
49         -1);

```

4.8 Strongly Connected Components

```

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

```

```

24         *(visitIt++) = p;
25     }
26
27 void run()
28 {
29     sorted.resize(v.size());
30     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 =
44         sorted.begin();
45         it != sorted.end(); ++it)
46         if(!(*it->visited)
47         {
48             curRes = &(*res.insert(
49                 res.end()));
50             dfs(&(*it));
51         }
52 }

```

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
2 // another one
3 // Alternative STL algorithms : strstr in <
4 // cstring> find in <string>
5 // Time complexity : O(n)
6
7 #include <cstdio>
8 #include <cstring>
9
10 #define MAXN 100010
11
12 char T[MAXN], P[MAXN]; // T = text, P =
13 // pattern
14 int b[MAXN], n, m; // b = back table, n =
15 // length of T, m = length of P
16
17 void kmpPreprocess() {
18     int i = 0, j = -1; b[0] = -1;
19     while (i < m){
20         while(j >= 0 && P[i] != P[j]) j = b[j];
21         b[i] = j;
22         ++i;
23     }
24 }

```

```

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
29                 - j);
30             j = b[j];
31         }
32     }
33 }
34
35 int main(){
36     strcpy(T, "asdhasdhejasdasdhejasdasd");
37     strcpy(P, "hej");
38
39     n = 25; m = 3;
40
41     kmpPreprocess();
42     kmpSearch();
43
44     return 0;
45 }

```

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
13         , y*oth); }
14     Vector operator/ (T oth){ return Vector(x/oth
15         , y/oth); }
16
17     Vector operator+ (Vector oth){ return Vector(
18         x+oth.x, y+oth.y); }
19     Vector operator- (Vector oth){ return Vector(
20         x+oth.x, y+oth.y); }
21     T operator* (Vector oth){ return x*oth.x + y
22         *oth.y; }
23     Vector operator/ (Vector oth){ return Vector(
24         x*oth.y-y*oth.x/y)}
25 };

```

6.2 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) const {
28         return data[col+row*c];
29     }
30     // implement as needed
31     bool operator==(In rhs) const {
32         return data == rhs.data;
33     }
34     M operator+(In rhs) const {
35         assert(rhs.r == r && rhs.c == c);
36         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 mult
49         assert(rhs.r == c);
50         Matrix ret(r, rhs.c);
51         rep(i, 0, r)
52             rep(j, 0, rhs.c)
53                 ret(i, j) = operator()(i, k)*rhs(k,
54                     j);
55         return ret;
56     }
57     M operator*(T rhs) const { // scalar mult
58         Matrix ret(*this);
59         trav(it, ret.data)
60             it = it*rhs;
61         return ret;
62     }
63 };
64
65 template<typename T> // create identity matrix
66 Matrix<T> id(int r, int c) {
67     Matrix<T> m(r,c);
68     rep(i, 0, r)

```



```

69     m(i, i) = T(1);
70 }

```

6.3 Matrix3d Class

```

1  /* 3 dimensional matrix class
2   * with Gauss Elimination and Eigenvectors
3   * Source: Magnus Selin
4   */
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 && i < 3)
18                      a[i][j] = 0;
19              }
20          }
21
22          double get(int i, int j) { if(j >= 0 &&
23              j < 3 && i >= 0 && i < 3) return a[i][j];
24              else std::cerr << "Out_of_bounds!\n"; }
25          void set(int i, int j, int v) { if(j >= 0 &&
26              j < 3 && i >= 0 && i < 3) a[i][j] = v;
27              else std::cerr << "Out_of_bounds!\n"; };
28
29          void chg_row(int x, int y);
30          void mult_row(int x, double c);
31          void add_row(int x, int y, double c);
32
33          Matrix3d gauss();
34          Matrix3d get_inverse();
35
36          double get_det();
37          void get_eigenvectors();
38
39          Matrix3d operator= (Matrix3d);
40          Matrix3d operator+ (Matrix3d);
41          Matrix3d operator- (Matrix3d);
42          Matrix3d operator* (Matrix3d);
43          Matrix3d operator* (double);
44      };
45
46      void Matrix3d::chg_row(int x, int y){
47          int temp[3] = {a[x][0], a[x][1], a[x][2]};
48
49          for(int i = 0; i < 3; i++){
50              a[x][i] = a[y][i];
51              a[y][i] = temp[i];
52          }
53      }
54
55      void Matrix3d::mult_row(int x, double c){
56          for(int i = 0; i < 3; i++){
57              a[x][i] *= c;

```

```

58      }
59  }
60
61  void Matrix3d::get_eigenvectors(){
62      double eig[3];
63      double p = a[0][1] * a[0][1] + a[0][2] * a
64          [0][2] + a[1][2] * a[1][2];
65      double q, r, phi;
66      Matrix3d B; Matrix3d I;
67      for (int i = 0; i < 3; i++) I.set(i, i, 1);
68
69      if ( p == 0 ){
70          eig[0] = a[0][0];
71          eig[1] = a[1][1];
72          eig[2] = a[2][2];
73      }
74      else {
75          q = (a[0][0] + a[1][1] + a[2][2]) / 3;
76          p = (a[0][0] - q) * (a[0][0] - q) +
77              (a[1][1] - q) * (a[1][1] - q) +
78              (a[2][2] - q) * (a[2][2] - q) + 2 * q;
79          p = sqrt( p / 6 );
80
81          B = ((*this) - I * q);
82          B = B * (1 / p);
83          r = B.get_det();
84
85          if (r <= -1)
86              phi = M_PI / 3;
87          else if (r >= 1)
88              phi = 0;
89          else
90              phi = acos(r) / 3;
91
92          eig[0] = q + 2 * p * cos(phi);
93          eig[2] = q + 2 * p * cos(phi + M_PI * (2/3)
94              );
95          eig[1] = 3 * q - eig[0] - eig[2];
96      }
97
98      std::cout << eig[0] << '\n' << eig[1] << '\n'
99          << eig[2] << '\n';
100
101      for (int i = 0; i < 3; i++) {
102          Matrix3d temp = (*this);
103
104          temp.set(0, 0, temp.get(0, 0) - eig[i]);
105          temp.set(1, 1, temp.get(1, 1) - eig[i]);
106          temp.set(2, 2, temp.get(2, 2) - eig[i]);
107          temp = temp.gauss();
108
109          std::cout << "Temp\n" << i << "\n" << temp
110              << "\n";
111      }
112
113      double Matrix3d::get_det(){
114          return a[0][0] * a[1][1] * a[2][2] + a[0][1]
115              * a[1][2] * a[2][0] + a[0][2] * a[1][0] * a
116              [2][1] -
117              a[0][2] * a[1][1] * a[2][0] - a[0][1] * a
118              [1][0] * a[2][2] - a[0][0] * a[1][2] * a
119              [2][1];
120      }
121
122      Matrix3d Matrix3d::gauss(){
123          Matrix3d * temp = new Matrix3d;
124          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          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, i);
141                  temp->add_row(j, i, mult_val);
142              }
143          }
144      }
145
146      std::cout << "Temp\n" << "\n" << *temp << "\n
147          ";
148      std::cout << "This\n" << "\n" << *this << "\n
149          ";
150
151      return *temp;
152  }
153
154      Matrix3d Matrix3d::get_inverse(){
155          Matrix3d temp = (*this), inverse;
156          for (int i = 0; i < 3; i++) inverse.set(i, i,
157              1);
158
159          for (int i = 0; i < 3; i++){
160              if(temp.get(i, i) == 0){
161                  for (int j = i; j < 3; j++){
162                      if(temp.get(j, i) != 0){
163                          temp.chg_row(i, j);
164                          inverse.chg_row(i, j);
165
166                          std::cout << "Change row\n" << i << "\n
167                              and\n" << j << "\n";
168                          std::cout << temp << '\n';
169                          std::cout << inverse << '\n';
170
171                          break;
172                      }
173                  }
174
175                  if(temp.get(i, i) == 0){
176                      std::cout << "Singularity!\n";
177                      break;
178                  }
179              }
180
181              double mult_val = temp.get(i, i);
182              temp.mult_row(i, 1 / mult_val );
183              inverse.mult_row(i, 1 / mult_val );

```



```

183     std::cout << "Divide_row" << i << "by"
184     << mult_val << ".\n";
185     std::cout << temp << '\n';
186     std::cout << inverse << '\n';
187     for (int j = 0; j < 3; j++){
188         if (i != j){
189             double mult_val = -temp.get(j, i);
190
191             temp.add_row(j, i, mult_val);
192             inverse.add_row(j, i, mult_val);
193
194             std::cout << "Multiply_row" << i << "by"
195             << mult_val << "and adding it to" <<
196             j << ".\n";
197             std::cout << temp << '\n';
198             std::cout << inverse << '\n';
199         }
200     }
201     return inverse;
202 }
203
204 Matrix3d Matrix3d::operator= (Matrix3d param){
205     Matrix3d temp;
206     for(int i = 0; i < 3; i++){
207         for(int j = 0; j < 3; j++){
208             temp.set(i, j, param.get(i, j));
209         }
210     }
211     return temp;
212 }
213
214 Matrix3d Matrix3d::operator+ (Matrix3d param){
215     Matrix3d temp;
216     for(int i = 0; i < 3; i++){
217         for(int j = 0; j < 3; j++){
218             temp.set(i, j, a[i][j] + param.get(i, j))
219         }
220     }
221     return temp;
222 }
223
224 Matrix3d Matrix3d::operator- (Matrix3d param){
225     Matrix3d temp;
226     for(int i = 0; i < 3; i++){
227         for(int j = 0; j < 3; j++){
228             temp.set(i, j, a[i][j] - param.get(i, j))
229         }
230     }
231     return temp;
232 }
233
234 Matrix3d Matrix3d::operator* (double param){
235     Matrix3d temp;
236     for(int i = 0; i < 3; i++){
237         for(int j = 0; j < 3; j++){
238             temp.set(i, j, a[i][j] * param);
239         }
240     }
241     return temp;
242 }
243
244 Matrix3d Matrix3d::operator* (Matrix3d param){
245     Matrix3d temp;

```

```

247     for(int i = 0; i < 3; i++){
248         for(int j = 0; j < 3; j++){
249             temp.set(i, j, a[i][0] * param.get(0,j) +
250             a[i][1] * param.get(1,j) +
251             a[i][2] * param.get(2,j) );
252         }
253     }
254     return temp;
255 }
256
257 std::ostream& operator << ( std::ostream& os,
258     Matrix3d m ){
259     for(int i = 0; i < 3; i++){
260         os << "(";
261         for(int j = 0; j < 3; j++){
262             os << m.get(i, j) << ",\t";
263         }
264         os << ")\n";
265     }
266     return os;
267 }
268

```

6.4 Points Class

```

1  /* Description: Untested homogenous coordinates
2  * transformation geometry.
3  * Source: Benjamin Ingberg
4  * Usage: Requires homogenous coordinates,
5  * handles
6  * multiple rotations, translations and scaling
7  * in a
8  * high precision efficient manner (matrix
9  * multiplication) with homogenous coordinates.
10 * Also keeps reverse transformation available.
11 */
12 namespace h { // avoid name collisions
13 struct Transform {
14     enum ActionType {
15         Scale, Rotate, TranslateX, TranslateY
16     };
17 typedef tuple<ActionType, fp> Action;
18 typedef Matrix<fp> M;
19 typedef vector<Action> History;
20 History hist;
21 M to, from;
22 Transform(History h = History())
23 : to(id<fp>(3,3)), from(id<fp>(3,3)) {
24     doTransforms(h);
25 }
26 H transformTo(H in) {
27     return H(to*M(in));
28 }
29 H transformFrom(H in) {
30     return H(from*M(in));
31 }
32 Transform & scale(fp s) {
33     doTransform(Scale, s);
34 }
35 Transform & translate(fp dx, fp dy) {
36     doTransform(TranslateX, dx);
37     doTransform(TranslateY, dy);
38 }
39 Transform & rotate(fp phi) {
40     doTransform(Rotate, phi);
41 }
42 void doTransforms(History & h) {

```

```

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

```

6.5 Graham Scan

```

1  struct point {
2      int x, y;
3  };
4  int det(const point& p1, const point& p2, const
5      point& p3)
6  {
7      int x1 = p2.x - p1.x;
8      int y1 = p2.y - p1.y;
9      int x2 = p3.x - p1.x;
10     int y2 = p3.y - p1.y;
11     return x1*y2 - x2*y1;
12 }
13 // bool ccw(const point& p1, const point& p2,
14 // const point& p3)
15 // { // Counterclockwise? Compare with
16 // determinant...
17 // return (det(p1, p2, p3) > 0);
18 // }
19 struct angle_compare {
20     point p; // Leftmost lower point
21     angle_compare(const point& p) : p(p) {}
22     bool operator()(const point& lhs, const point&
23         rhs) {
24         int d = det(p, lhs, rhs);

```

```

23     if(d == 0) // Furthest first if same
        direction will keep all
24     return (x1*x1+y1*y1 > x2*x2+y2*y2); //
        points at the line
25     return (d > 0); // Counterclockwise?
26 }
27 };
28
29 int ConvexHull(const vector<point>& p, int* res
    )
30 { // Returns number of points in the convex
    polygon
31     int best = 0; // Find the first leftmost lower
        point
32     for(int i = 1; i < p.size(); ++i)
33     {
34         if(p[i].y < p[best].y ||
35            (p[i].y == p[best].y && p[i].x < p[
                best].x))
36             best = i;
37     }
38     sort(p.begin(), p.end(), angle_compare(p[best]
        ));
39     for(int i = 0; i < 3; ++i)
40         res[i] = i;
41     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 }

```

6.6 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 {
23         return Point<T>(x - o.x, y - o.y); }
24     Point<T> operator +(const Point<T> &o) const {
25         return Point<T>(x + o.x, y + o.y); }
26
27     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 typedef Point<int> Pt;
115
116 bool solve() {
117     nat count;
118     scanf("%d", &count);
119
120     if (count == 0) return false;
121
122     vector<Pt> points(count);
123     for (nat i = 0; i < count; i++) {
124         scanf("%d%d", &points[i].x, &points[i].y);
125     }
126
127     vector<Pt> result = convex_hull(points);
128
129     printf("%d\n", (int)result.size());
130     for (nat i = 0; i < result.size(); i++) {
131         printf("%d%d\n", result[i].x, result[i].y);
132     }
133
134     return true;
135 }
136
137 int main() {
138     while(solve());
139
140     return 0;
141 }

```

6.7 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 {
21         return Point<T>(x - o.x, y - o.y); }
22     Point<T> operator /(T o) const { return Point<
23         T>(x / o, y / o); }
24     T operator |(const Point &o) const {
25         return x * o.x + y * o.y;
26     }
27 };
28
29 template <class T>
30 class Vector {
31 public:
32     T x, y, z;
33
34     Vector() : x(), y(), z() {}
35     Vector(const Point<T> &pt, T z) : x(pt.x), y(
36         pt.y), z(z) {}
37     Vector(T x, T y, T z) : x(x), y(y), z(z) {}
38
39     Vector<T> operator -(const Vector &o) const {
40         return Vector<T>(x - o.x, y - o.y, z - o.z)
41         ; }
42     Vector<T> operator /(T o) const { return
43         Vector<T>(x / o, y / o, z / o); }
44     T operator |(const Vector &o) const { return x
45         * o.x + y * o.y + z * o.z; }
46     Vector<T> operator %(const Vector &o) const {
47         return Vector<T>(y*o.z - z*o.y, z*o.x - x*o.z
48         , x*o.y - y*o.x);
49     }
50 };
51
52 // distance between two points or vectors.
53 template <class T>
54 T dist(const Point<T> &a, const Point<T> &b) {
55     Point<T> d = a - b;
56     return sqrt(d | d);
57 }
58
59 // Normalize a line
60 template <class T>
61 void normLine(Vector<T> &v) {
62     T l = sqrt(v.x * v.x + v.y * v.y);
63     v = v / l;
64 }
65
66 // Normalize a point
67 template <class T>
68 void normPoint(Vector<T> &v) {
69     v = v / v.z;
70 }
71
72 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))
69         < 0) {
70         return dist(point, lineFrom);
71     }
72     // Outside second endpoint?
73     if (((point - lineTo) | (lineFrom - lineTo)) <
74         0) {
75         return dist(point, lineTo);
76     }
77     // Ok, in the middle of the line!
78
79     // Create the homogenous representation of the
80     line...
81     Vector<T> line = Vector<T>(lineFrom, 1) %
82         Vector<T>(lineTo, 1);
83
84     // The signed distance is then the dot product
85     of the line
86     and the point.
87     normLine(line);
88     T distance = Vector<T>(point, 1) | line;
89
90     // Don't return negative distances...
91     return abs(distance);
92 }
93
94 vector<Point<double>> readPoints() {
95     nat size = 0;
96     scanf("%d", &size);
97
98     vector<Point<double>> result;
99
100     for (nat i = 0; i < size; i++) {
101         double x, y;
102         scanf("%lf%lf", &x, &y);
103         result.push_back(Point<double>(x, y));
104     }
105
106     return result;
107 }
108
109 void solve() {
110     vector<Point<double>> inner = readPoints();
111     vector<Point<double>> outer = readPoints();
112
113     double longest = 1e100;
114
115     for (nat i = 0; i < inner.size(); i++) {
116         nat iNext = (i + 1) % inner.size();
117         for (nat j = 0; j < outer.size(); j++) {
118             nat jNext = (j + 1) % outer.size();
119
120             longest = min(longest, dist(outer[j], inner[
121                 i], inner[iNext]));
122             longest = min(longest, dist(inner[i], outer[
123                 j], outer[jNext]));
124         }
125     }
126
127     printf("%.8lf\n", longest / 2.0);
128 }
129
130 int main() {
131
132

```

```

127 int tc;
128 scanf("%d", &tc);
129
130 while (tc--) solve();
131
132 return 0;
133 }

```

6.8 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,
8      int k) {
9      int m;
10     while( r - l > 1 ) {
11         m = l + (r - l)/2;
12         if( a[t[m]] >= k )
13             r = m;
14         else
15             l = m;
16     }
17     return r;
18 }
19
20 vector<int> lis(int a[], int n){
21     std::vector<int> lis;
22     if(n == 0) return lis;
23     int c[n]; memset(c, 0, sizeof(c));
24     int p[n]; memset(p, 0xFF, sizeof(p));
25     int s = 1;
26
27     c[0] = 0;
28     p[0] = -1;
29     for(int i = 1; i < n; i++){
30         if(a[i] < a[c[0]]){
31             c[0] = i;
32         }
33         else if(a[i] > a[c[s-1]]){
34             p[i] = c[s-1];
35             c[s] = i;
36             s++;
37         }
38     }
39     return lis;

```

