

# NAT: Nostalgic Alien Trespassers

Filip Stromback, Magnus Selin, Carl Einarson

November 22, 2013

## Contents

<b>1 Environment</b>	<b>2</b>		
1.1 Template . . . . .	2		
<b>2 Data Structures</b>	<b>2</b>		
2.1 Union Find . . . . .	2	4.4 Minimum Spanning Tree . . . . .	5
2.2 Fenwick Tree . . . . .	2	4.5 Maximum Flow . . . . .	6
<b>3 Numerical</b>	<b>2</b>	4.6 Euler Tour . . . . .	6
3.1 General Utils . . . . .	2	<b>5 String processing</b>	<b>6</b>
3.2 Rational Numbers Class . . . . .	3	5.1 Stl . . . . .	6
3.3 Binary Search . . . . .	3	5.2 String Matching . . . . .	6
3.4 De Bruijn . . . . .	3	5.3 String Multismatching . . . . .	6
3.5 Prime Generator . . . . .	4	<b>6 Geometry</b>	<b>6</b>
3.6 Factorisation . . . . .	4	6.1 Points Class . . . . .	6
<b>4 Graphs</b>	<b>4</b>	6.2 Transformation . . . . .	7
4.1 Single Source Shortest Path . . . . .	4	6.3 Points Class . . . . .	7
4.2 Single Source Shortest Path Time Table . . . . .	4	6.4 Graham Scan . . . . .	7
4.3 All Pairs Shortest Path . . . . .	5	6.5 Convex Hull . . . . .	8
		6.6 Line-point distance . . . . .	8
		<b>7 Misc</b>	<b>9</b>
		7.1 Longest Increasing Subsequence . . . . .	9
		7.2 Longest Increasing Substring . . . . .	10

# 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     return 0;
53 }
```

```
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 strre
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 // Returvrd: true om x finns, annars false.
36 // FwdIt lower_bound(FwdIt first, FwdIt last, T
    & x [, Pred pr]);
37 // Returvrd: 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 // Returvrd: 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 // Returvrd: pair(lower_bound(fi, la, x),
    upper_bound(fi, la, x))

45 // Binary search (from Wikipedia)
46 // The indices are _inclusive_.
47 int binary_search(T *a, int key, int min, int
    max) {
48     while (min < max) {
49         int mid = (min + max) / 2; // midpoint(min,
            max)
50
51         // assert(mid < max)
52
53         // The condition can be replaced by some
            other function
54         // depending on mid, eg worksFor(mid + 1) to
            search for
55         // the last index "worksFor" returns true for
            .
56         if (a[mid] < key) {
57             min = mid + 1;
58         } else {
59             max = mid;
60         }
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 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[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]);

```

### 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
21 template<class T>
22 vector<T> dijkstras(vector<pair<int, T> > G[],
23     int n, int e, int s){
24     priority_queue<pair<int, T>, vector<pair<int,
25         T>, >, comp> Q;
26
27     vector<T> 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(pair<int, T>(s, c[s]));
33     int u, sz, v; T w;
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             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 }

```

```

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

```

```

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%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()){

```

```

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", &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", &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", &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     }
36 }

```

# 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 <<
6          '\n';
7
8  str.replace(str.find(str2), str2.length(), "new
9      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 MAX_N 100010
11
12 char T[MAX_N], P[MAX_N]; // T = text, P =
13     pattern
14 int b[MAX_N], 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         i++; j++;
22         b[i] = j;
23     }
24 }
25
26 void kmpSearch() {
27     int i = 0, j = 0;
28     while(i < n){
29         while(j >= 0 && T[i] != P[j]) j = b[j];
30         i++; j++;
31         if(j == m){
32             printf("P is found at index %d in T\n", i
33                 - j);
34             j = b[j];
35         }
36     }
37 }
38
39 int main(){
40     strcpy(T, "asdhasdhejasdasdhejasdasd");
41     strcpy(P, "hej");
42
43     n = 25; m = 3;
44
45     kmpPreprocess();
46     kmpSearch();
47
48     return 0;
49 }

```

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

```

## 6.2 Calculate Area

```

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

```

## 6.3 Transformation

```

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             rep(k, 0, c)
54                 ret(i,j) += operator()(i, k)*rhs(k,
55                     j);
56     return ret;
57 }
58 M operator*(T rhs) const { // scalar mult
59     Matrix ret(*this);
60     trav(it, ret.data)
61         it = it*rhs;
62     return ret;
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 }

```

## 6.4 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     }
30     H transformFrom(H in) {
31         return H(from*M(in));
32     }
33     Transform & scale(fp s) {
34         doTransform(Scale, s);
35     }
36     Transform & translate(fp dx, fp dy) {
37         doTransform(TranslateX, dx);
38         doTransform(TranslateY, dy);
39     }
40     Transform & rotate(fp phi) {
41         doTransform(Rotate, phi);
42     }
43     void doTransforms(History & h) {
44         trav(it, h) {
45             doTransform(get<0>(*it), get<1>(*it));
46         }
47     }
48     void doTransform(ActionType t, fp v) {
49         hist.push_back(make_tuple(t, v));
50         if(t == Scale)
51             doScale(v);
52         else if(t == TranslateX)
53             doTranslate(0,v);
54         else if(t == TranslateY)
55             doTranslate(1,v);
56         else
57             doRotate(v);
58     }
59 private:
60     void doScale(fp s) {
61         M sm(id<fp>(3,3)), ism(id<fp>(3,3));
62         sm(1,1) = sm(0,0) = s;
63         ism(1,1) = ism(1,1) = 1/s;
64         to = to*sm; from = ism*from;
65     }
66     void doTranslate(int c, fp dx) {
67         M sm(id<fp>(3,3)), ism(id<fp>(3,3));
68         sm(c,2) = dx;
69         ism(c,2) = -dx;
70         to = to*sm; from = ism*from;
71     }
72     void doRotate(fp phi) {
73         M sm(id<fp>(3,3)), ism(id<fp>(3,3));
74         sm(0,0) = sm(1,1) = cos(phi);
75         ism(0,0) = ism(1,1) = cos(-phi);
76         ism(1,0) = sm(0,1) = sin(phi);
77         ism(0,1) = sm(1,0) = sin(-phi);
78         to = to*sm; from = ism*from;
79     }
80 }

```

## 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);
25     if(d == 0) // Furthest first if same
26       direction will keep all
27       return (x1*x1+y1*y1 > x2*x2+y2*y2); //
28       points at the line
29     return (d > 0); // Counterclockwise?
30 }
31 };
32 int ConvexHull(const vector<point>& p, int* res
33 )
34 { // Returns number of points in the convex
35   polygon
36   int best = 0; // Find the first leftmost lower
37   point
38   for(int i = 1; i < p.size(); ++i)
39   {
40     if(p[i].y < p[best].y ||
41        (p[i].y == p[best].y && p[i].x < p[
42        best].x))
43       best = i;
44   }
45   sort(p.begin(), p.end(), angle_compare(p[best
46   ]));
47   for(int i = 0; i < 3; ++i)
48     res[i] = i;
49   int n = 3;
50   for(int i = 3; i < p.size(); ++i)
51   {
52     // All consecutive points should be counter
53     clockwise
54     while(n > 2 && det(res[n-2], res[n-1], i) <
55     0)
56       --n; // Keep if det = 0, i.e. the same
57       line, angle_compare
58     res[n++] = i;
59   }
60   return n;
61 }

```

## 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   Point(T x = T(), T y = T()) : x(x), y(y) {}
15
16   bool operator <(const Point<T> &o) const {
17     if (y != o.y) return y < o.y;
18     return x < o.x;
19   }
20
21   Point<T> operator -(const Point<T> &o) const {
22     return Point<T>(x - o.x, y - o.y); }
23   Point<T> operator +(const Point<T> &o) const {
24     return Point<T>(x + o.x, y + o.y); }
25
26   T lenSq() const { return x*x + y*y; }
27 };
28
29 template <class T>
30 struct sort_less {
31   const Point<T> &ref;
32
33   sort_less(const Point<T> &p) : ref(p) {}
34
35   double angle(const Point<T> &p) const {
36     Point<T> delta = p - ref;
37     return atan2(delta.y, delta.x);
38   }
39
40   bool operator() (const Point<T> &a, const
41   Point<T> &b) const {
42     double aa = angle(a);
43     double ab = angle(b);
44     if (aa != ab) return aa < ab;
45     return (a - ref).lenSq() < (b - ref).lenSq();
46   }
47 };
48
49 template <class T>
50 int ccw(const Point<T> &p1, const Point<T> &p2,
51 const Point<T> &p3) {
52   return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y -
53   p1.y) * (p3.x - p1.x);
54 }
55
56 template <class T>
57 vector<Point<T> > convex_hull(vector<Point<T> >
58 input) {
59   if (input.size() < 2) return input;
60   nat size = input.size();
61
62   vector<Point<T> > output;
63
64   // Find the point with the lowest x and y
65   value.
66   int minIndex = 0;
67   for (int i = 1; i < size; i++) {
68     if (input[i] < input[minIndex]) {
69       minIndex = i;
70     }
71   }
72
73   // This is the "root" point in our traversal.
74   Point<T> p = input[minIndex];
75   output.push_back(p);
76   input.erase(input.begin() + minIndex);
77
78   // Sort the other elements according to the
79   angle with "p"
80   sort(input.begin(), input.end(), sort_less<T>(
81   p));

```

```

82   // Add the first point from "input" to the "
83   output" as a candidate.
84   output.push_back(input[0]);
85
86   // Start working our way through the points...
87   input.push_back(p);
88   size = input.size();
89   for (nat i = 1; i < size; i++) {
90     while (output.size() >= 2) {
91       nat last = output.size() - 1;
92       int c = ccw(output[last - 1], output[last],
93       input[i]);
94       if (c == 0) {
95         // Colinear points! Take away the
96         closest.
97         if ((output[last - 1] - output[last]).
98         lenSq() <= (output[last - 1] -
99         input[i]).lenSq()) {
100           if (output.size() > 1)
101             output.pop_back();
102           else
103             break;
104         } else {
105           break;
106         }
107       } else if (c < 0) {
108         if (output.size() > 1)
109           output.pop_back();
110         else
111           break;
112       }
113     }
114     output.push_back(input[i]);
115   }
116
117   // Do not take the last point twice.
118   if (i < size - 1)
119     output.push_back(input[i]);
120
121   return output;
122 }
123
124 typedef Point<int> Pt;
125
126 bool solve() {
127   nat count;
128   scanf("%d", &count);
129
130   if (count == 0) return false;
131
132   vector<Pt> points(count);
133   for (nat i = 0; i < count; i++) {
134     scanf("%d %d", &points[i].x, &points[i].y);
135   }
136
137   vector<Pt> result = convex_hull(points);
138
139   printf("%d\n", (int)result.size());
140   for (nat i = 0; i < result.size(); i++) {
141     printf("%d %d\n", result[i].x, result[i].y);
142   }
143
144   return true;
145 }

```



```

138 int main() {
139     while(solve());
140
141     return 0;
142 }

```

## 6.7 Line-point distance

```

1 // Problem 12173 on UVA (accepted there)
2
3 #include <stdio>
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 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(
35         pt.y), z(z) {}
36     Vector(T x, T y, T z) : x(x), y(y), z(z) {}
37
38     Vector<T> operator -(const Vector &o) const {
39         return Vector<T>(x - o.x, y - o.y, z - o.z)
40         ; }
41     Vector<T> operator /(T o) const { return
42         Vector<T>(x / o, y / o, z / o); }
43     T operator |(const Vector &o) const { return x
44         * o.x + y * o.y + z * o.z; }
45     Vector<T> operator %(const Vector &o) const {
46         return Vector<T>(y*o.z - z*o.y, z*o.x - x*o.z
47         , x*o.y - y*o.x); }
48 };
49
50 // distance between two points or vectors.
51 template <class T>
52 T dist(const Point<T> &a, const Point<T> &b) {
53     Point<T> d = a - b;
54     return sqrt(d | d);
55 }
56
57 // 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> &
67     lineFrom, const Point<T> &lineTo) {
68     // Outside first endpoint?
69     if (((point - lineFrom) | (lineTo - lineFrom))
70         < 0) {
71         return dist(point, lineFrom);
72     }
73     // Outside second endpoint?
74     if (((point - lineTo) | (lineFrom - lineTo)) <
75         0) {
76         return dist(point, lineTo);
77     }
78     // Ok, in the middle of the line!
79     // Create the homogenous representation of the
80     line...
81     Vector<T> line = Vector<T>(lineFrom, 1) %
82         Vector<T>(lineTo, 1);
83     // The signed distance is then the dot product
84     of the line
85     // and the point.
86     normLine(line);
87     T distance = Vector<T>(point, 1) | line;
88     // Don't return negative distances...
89     return abs(distance);
90 }
91
92 vector<Point<double>> readPoints() {
93     nat size = 0;
94     scanf("%d", &size);
95
96     vector<Point<double>> result;
97
98     for (nat i = 0; i < size; i++) {
99         double x, y;
100         scanf("%lf%lf", &x, &y);
101         result.push_back(Point<double>(x, y));
102     }
103
104     return result;
105 }
106
107 void solve() {
108     vector<Point<double>> inner = readPoints();
109     vector<Point<double>> outer = readPoints();
110
111     double longest = 1e100;
112
113     for (nat i = 0; i < inner.size(); i++) {
114         nat iNext = (i + 1) % inner.size();
115         for (nat j = 0; j < outer.size(); j++) {
116             nat jNext = (j + 1) % outer.size();

```

```

116         longest = min(longest, dist(outer[j], inner[
117             i], inner[iNext]));
118         longest = min(longest, dist(inner[i], outer[
119             j], outer[jNext]));
120     }
121 }
122
123 printf("%.8lf\n", longest / 2.0);
124 }
125
126 int main() {
127     int tc;
128     scanf("%d", &tc);
129
130     while (tc--) solve();
131
132     return 0;
133 }

```

## 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         else{
39             int pos = bin_search(a, c, -1, s-1, a[i])
40             ;
41             p[i] = c[pos-1];
42             c[pos] = i;

```

```

41     }
42 }
43
44 int d = c[s-1];
45 for( int i = 0; i < s; i++ ){
46     lis.push_back(d);
47     d = p[d];
48 }
49
50
51 reverse(lis.begin(),lis.end());
52 return lis;
53 }
54 int main(){
55     int n;
56     while(scanf("%d", &n) == 1){
57         int a[n]; for(int i = 0; i < n; i++) scanf(
58             "%d", &a[i]);
59         vector<int> lseq = lis(a, n);
60

```

```

61         printf("%d\n", (int)lseq.size());
62         for(int i = 0; i < lseq.size(); i++){
63             printf("%d_", lseq[i]);
64         }
65         printf("\n");
66     }
67     lseq.clear();
68 }
69 }

```

## 7.2 Longest Increasing Substring

```

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

```

```

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

```