

TCR: Wir werden nach Delft fahren

Filip Stromback, Magnus Selin, Carl Einarson

November 22, 2013

Contents

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

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 <
18   int(b); ++i)
19 #define trav(it, v) for(typeof((v).begin())
20   it = (v).begin(); it != (v).end(); ++
21   it)
22
23 typedef double fl;
24 typedef long long ll;
25 typedef pair<int, int> pii;
26 typedef vector<int> vi;
27
28 bool solve(){
29     return true;
30 }
31
32 int main(){
33     int tc=1; //scanf("%d", &tc);
34     rep(i, 0, tc) solve();
35
36     return 0;
37 }
```

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
13   y){
14     int a = find(root, x);
15     int b = find(root, y);
16     root[a] = b;
17 }
18
19 bool issame(int * root, int a, int b){
20     return (find(root, a) == find(root, b));
21 }
```

```
20 }
21
22 int main(){
23     int n, no; scanf("%d%d", &n, &no);
24     int root[n];
25     for(int i = 0; i < n; i++){
26         root[i] = i;
27     }
28
29     for(int i = 0; i < no; i++){
30         char op; int a, b;
31         scanf("%*[\n\t]%c", &op);
32         scanf("%d%d", &a, &b);
33         if(op == '?'){
34             if(issame(root, a, b)) printf("yes\n"
35 );
36             else printf("no\n");
37         }
38         if(op == '=')
39             uni(root, deep, a, b);
40     }
41 }
```

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("%*[\n\t]%c", &op);
40         switch (op){
41             case '+':
42                 scanf("%lld%lld", &a, &b);
43                 fenwick_update(ft, a+1, b);
44                 break;
45 }
```

```
46         case '?':
47             scanf("%lld", &a);
48             printf("%lld\n", fenwick_read(ft, a))
49             ;
50             break;
51         }
52     }
53     return 0;
54 }
```

3 Numerical

3.1 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
9       long int b) {
10         if (a < 0) a = -a;
11         if (b < 0) b = -b;
12         if (0 == b) return a;
13         else return gcd(b, a % b);
14 }
15 public:
16     Q(){}
17     Q(long long int a, long long int b){
18         p = a; q = b;
19         if (q < 0){p = -p; q = -q;}
20         if (p == 0) q = 1;
21         if (q == 0){
22             printf("ERR: _den_ = _0!\n");
23             q = 1;
24         }
25         long long int g = gcd(p, q);
26         p /= g; q /= g;
27     }
28
29     Q operator + (Q a){
30         Q b = * this;
31         Q res = Q((a.p * b.q + b.p * a.q), (a.q
32           * b.q));
33         return res;
34 }
35
36     Q operator - (Q a){
37         Q b = * this;
38         Q res;
39         if(a==b) res = Q(0,0);
40         else res = Q((b.p * a.q - a.p * b.q), (
41           a.q * b.q));
42         return res;
43 }
44
45     Q operator * (Q a){
46         Q b = * this;
47         Q res = Q(a.p * b.p, a.q * b.q);
48         return res;
49 }
50
51     Q operator / (Q a){
52         Q b = * this;
53         Q res = Q(b.p * a.q, b.q * a.p);
54         return res;
55 }
```

```

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
77         char t='␣'; while (t == '␣') scanf("%c"
78         , &t);
79
80         scanf("%d%d", &tp, &tn); Q b = Q(tp, tn
81         );
82
83         switch(t){
84             case '+': (a+b).print(); break;
85             case '-': (a-b).print(); break;
86             case '*': (a*b).print(); break;
87             case '/': (a/b).print(); break;
88         }
89     }
90
91     return 0;
92 }

```

3.2 Binary Search

```

1 // Example usage of the bsearch
2 #include <cstdlib>
3 #include <stdio>
4
5 int check(const void *key, const void *elem
6 ) {
7     int k = (int)key;
8     int e = (int)elem;
9     printf("Comparing␣%d␣with␣%d␣\n", k, e);
10
11     if (k == e) return 0;
12     if (k < e) return -1;
13     return 1;
14 }
15
16 int main() {
17     int found = (int)bsearch((const void *)10,
18     0, 100, 1, &check);
19
20     printf("I␣found:␣%d␣\n", found);
21
22     return 0;
23 }

```

3.3 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,
35         n)] * pow((double)2, n-i-1);
36     }
37     cout << sum << '\n';
38 }
39
40 int main(){
41     int tc;
42     cin >> tc;
43     for(int we = 0; we < tc; we++){
44         cin >> n >> k;
45         a.clear(); seq.clear();
46         de_bruijn();
47     }
48 }

```

3.4 Prime Generator

```

1 #include <stdio>
2
3 int prime[664579];
4 int numprimes;
5
6 void calcprimes(int maxn){
7     prime[0] = 2; numprimes = 1; prime[
8     numprimes] = 46340; // 0xb504*0xb504 =
9     0x7FFE810
10     for(int n = 3; n < maxn; n += 2) {
11         for(int i = 1; prime[i]*prime[i] <= n;
12         ++i) {
13             if(n % prime[i] == 0) goto not_prime;
14         }
15         prime[numprimes++] = n; prime[numprimes
16         ] = 46340; // 0xb504*0xb504 = 0
17         x7FFE810
18     }
19     not_prime:
20 }

```

```

15     }
16 }
17
18 int main(){
19     calcprimes(10000000);
20     for(int i = 0; i < 664579; i++) printf("%
21     d␣\n", prime[i]);
22 }

```

3.5 Factorisation

```

1 int factor[1000000];
2 int numf[1000000];
3 int numfactors;
4
5 void calcfactors(int n){
6     numfactors = 0;
7     for(int i = 0; n > 1; ++i){
8         if(n % prime[i] == 0){
9             factor[numfactors] = prime[i];
10             numf[numfactors] = 0;
11             do {
12                 numf[numfactors]++;
13                 n /= prime[i];
14             } while(n % prime[i] == 0);
15             numfactors++;
16         }
17     }
18 }

```

4 Graphs

4.1 Single Source Shortest Path

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

```

1 #include <stdio.h>
2 #include <queue>
3 #include <vector>
4
5 #define INF 100000000
6
7 using namespace std;
8
9 typedef pair<int, int> ii;
10
11 template<class T>
12
13 class comp{
14 public:
15     int operator()(const pair<int, T> & a,
16     const pair<int, T> & b){return (a.
17     second > b.second);}
18 };
19
20 template<class T>
21 vector<T> dijkstras(vector<pair<int, T> > G
22 [], int n, int e, int s){
23     priority_queue<pair<int, T>, vector<pair
24     <int, 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++)
29     p.push_back(-1);
30
31     Q.push(pair<int, T>(s, c[s]));
32     int u, sz, v; T w;

```

```

27 while(!Q.empty()){
28
29     u = Q.top().first; Q.pop();
30     sz = G[u].size();
31     for(int i = 0; i < sz; i++){
32         v = G[u][i].first;
33         w = G[u][i].second;
34         if( c[v] > c[u] + w ){
35             c[v] = c[u] + w;
36             p[v] = u;
37             Q.push(pair<int, T>(v, c[v]));
38         }
39     }
40 }
41
42 //printf("Path to follow: ");
43 //for(int i = 0; i < n; i++) printf("%d
44 ", p[i]);
45 //printf("\n");
46
47 return c;
48 }
49
50 int main(){
51     int n, e, q, s;
52     scanf("%d%d%d", &n, &e, &q, &s);
53     while(n!=0 or e!=0 or q!=0 or s!=0){
54         vector<iA> G[n];
55         for(int i = 0; i < e; i++){
56             int f, t, w;
57             scanf("%d%d%d", &f, &t, &w);
58             G[f].push_back(ii(t, w));
59         }
60         vector<int> c = dijkstras(G, n, e, s);
61
62         for(int i = 0; i < q; i++) {
63             int d; scanf("%d", &d);
64             if(c[d] == INF) printf("Impossible\n");
65             else printf("%d\n", c[d]);
66         }
67         printf("\n");
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 =
11         c;}
12     int t0, tn, w;
13 };
14
15 typedef pair<int, int> ii;
16 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++){
28             c.push_back(INF); c[s] = 0;
29         }
30         vector<int> p; for(int i = 0; i < n; i++){
31             p.push_back(-1);
32
33         Q.push(ii(s, c[s]));
34         int u, sz, v, t0, tn, w, wt;
35         while(!Q.empty()){
36
37             u = Q.top().first; Q.pop();
38             sz = G[u].size();
39             for(int i = 0; i < sz; i++){
40                 v = G[u][i].first;
41                 tn = G[u][i].second.tn;
42                 t0 = G[u][i].second.t0;
43                 w = G[u][i].second.w;
44
45                 wt = t0 - c[u];
46                 if (wt < 0 and tn == 0) continue;
47                 while(wt < 0) wt+=tn;
48
49                 if( c[v] > c[u] + w + wt){
50                     c[v] = c[u] + w + wt;
51                     p[v] = u;
52                     Q.push(ii(v, c[v]));
53                 }
54             }
55         }
56
57         //printf("Path to follow: ");
58         //for(int i = 0; i < n; i++) printf("%d
59 ", p[i]);
60         //printf("\n");
61
62         return c;
63     }
64 }
65
66 int main(){
67     int n, e, q, s;
68     scanf("%d%d%d", &n, &e, &q, &s);
69     while(n!=0 or e!=0 or q!=0 or s!=0){
70         vector<iA> G[n];
71         for(int i = 0; i < e; i++){
72             int f, t, t0, tn, w;
73             scanf("%d%d%d", &f, &t, &t0, &tn,
74                 &w);
75             G[f].push_back(iA(t, A(t0, tn, w)));
76         }
77         vector<int> c = dijkstras(G, n, e, s);
78
79         for(int i = 0; i < q; i++) {
80             int d; scanf("%d", &d);
81             if(c[d] == INF) printf("Impossible\n");
82             else printf("%d\n", c[d]);
83         }
84         printf("\n");
85         scanf("%d%d%d", &n, &e, &q, &s);
86     }
87 }

```

```

80
81     return 0;
82 }

```

4.3 All Pairs Shortest Path

Floyd Warshall's algorithm. Assign nodes which are part of a negative cycle to minus infinity.

Time Complexity $O(V^3)$

```

1 // All pairs shortest path (Floyd Warshall)
2 // Assign nodes which are part of a
3 // negative cycle to minus infinity.
4
5 #include <stdio.h>
6 #include <iostream>
7 #include <vector>
8 #include <algorithm>
9
10 #define INF 1000000000
11 using namespace std;
12
13 template<class T>
14 vector< vector<T> > floyd_warshall(vector<
15     vector<T> > d){
16     int n = d.size();
17     for(int i = 0; i < n; i++) d[i][i] = 0;
18
19     for (int k = 0; k < n; k++)
20         for (int i = 0; i < n; i++)
21             for (int j = 0; j < n; j++)
22                 if (d[i][k] != INF and d[k][j] != INF)
23                     d[i][j] = min(d[i][j], d[i][k]+d[k][j]);
24
25     for(int i = 0; i < n; i++)
26         for(int j = 0; j < n; j++)
27             if(d[i][j] != -INF and d[j][i] != -INF)
28                 d[i][j] = min(d[i][j], d[i][i]+d[j][j]);
29
30     return d;
31 }
32
33 int main(){
34     int n, m, q; scanf("%d%d", &n, &m, &q);
35     while(n!=0 or m!=0 or q!=0){
36         vector< vector<int> > d;
37         d.resize(n);
38         for(int i = 0; i < n; i++)
39             d[i].push_back(INF);
40
41         for(int i = 0; i < m; i++){
42             int f, t, w; scanf("%d%d", &f, &t,
43                 &w);
44             d[f][t] = min(w, d[f][t]);
45         }
46
47         d = floyd_warshall(d, n);
48         for(int i = 0; i < q; i++){
49             int f, t; scanf("%d", &f, &t);
50             if(d[f][t] == INF) printf("Impossible\n");
51             else if(d[f][t] == -INF) printf("-Infinity\n");
52         }
53     }
54 }

```

```

51         else                printf("%d\n", d[f][t
52     });
53     }
54     printf("\n");
55     scanf("%d%d%d", &n, &m, &q);
56 }
57 return 0;
58 }

```

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 struct Tree{
19     int w;
20     bool complete;
21     std::vector<AnsEdge> e;
22     Tree(){
23         w = 0;
24         complete = true;
25     }
26 };
27
28 struct Vertex{
29     Vertex *p;
30     Vertex *root() {
31         if(p->p != p)
32             p = p->root();
33         return p;
34     }
35 };
36 struct Edge{
37     int f, t, w;
38
39     bool operator<(const Edge& oth) const{
40         if (w == oth.w)
41             return(t < oth.t);
42         return(w < oth.w);
43     }
44 };
45
46
47 Tree kruskal(Vertex * v, Edge * e, int numv
48     , int nume){
49     Tree ans;
50     int sum = 0;
51
52     for(int i = 0; i < numv; ++i){
53         v[i].p = &v[i];
54     }
55
56     sort(&e[0], &e[nume]);

```

```

57     for(int i = 0; i < nume; ++i){
58         if(v[e[i].f].root() != v[e[i].t].root())
59         ){
60             v[e[i].t].root()->p = v[e[i].f].root
61             ();
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
67             [i].f, e[i].t));
68         }
69     }
70
71     Vertex * p = v[0].root();
72     for(int i = 0; i < numv; ++i)
73     {
74         if(p != v[i].root()){
75             ans.complete = false;
76             break;
77         }
78     }
79
80     sort(ans.e.begin(), ans.e.end());
81
82     return ans;
83 }
84
85 int main(){
86     int n, m; scanf("%d%d", &n, &m);
87     while(n or m){
88         Vertex v[n];
89         Edge e[m];
90
91         for(int i = 0; i < m; ++i){
92             int f, t;
93             scanf("%d%d", &f, &t, &e[i].w);
94             e[i].f = f;
95             e[i].t = t;
96         }
97
98         Tree ans = mst(v, e, n, m);
99
100         if(ans.complete){
101             printf("%d\n", ans.w);
102             for(int i = 0; i < ans.e.size(); ++i)
103             {
104                 printf("%d_ %d\n", ans.e[i].f, ans.e
105                 [i].t);
106             }
107         }
108         else printf("Impossible\n");
109
110         scanf("%d", &n, &m);
111     }
112
113     return 0;
114 }

```

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
18         <int> q; q.push(s);
19         p.assign(MAX_V, -1);
20         while(!q.empty()){
21             int u = q.front(); q.pop();
22             if(u == t) break;
23             for(int v = 0; v < MAX_V; v++)
24                 if (res[u][v] > 0 && dist[v] == INF
25                     )
26                     dist[v] = dist[u] + 1, q.push(v),
27                     p[v] = u;
28         }
29         augment(t, INF);
30         if(f == 0) break;
31         mf += f;
32     }
33
34     printf("%d\n", mf);
35 }

```

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

```

```
31 }
```

5 String processing

5.1 String Matching

```
1 // Knuth Morris Prat : Search for a string
  in another one
2 // Alternative STL algorithms : strstr in <
  cstring> find in <string>
3 // Time complexity : O(n)
4
5 #include <stdio>
6 #include <cstring>
7
8 #define MAXN 100010
9
10 char T[MAXN], P[MAXN]; // T = text, P =
   pattern
11 int b[MAXN], n, m; // b = back table,
   n = length of T, m = length of P
12
13 void kmpPreprocess() {
14     int i = 0, j = -1; b[0] = -1;
15     while (i < m){
16         while(j >= 0 && P[i] != P[j]) j = b[j];
17         i++; j++;
18         b[i] = j;
19     }
20 }
21
22 void kmpSearch() {
23     int i = 0, j = 0;
24     while(i < n){
25         while(j >= 0 && T[i] != P[j]) j = b[j];
26         i++; j++;
27         if(j==m){
28             printf("P is found at index %d in T\n",
29                    i - 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 }
```

6 Geometry

6.1 Points Class

```
1 #include <cmath>
2
3 template<class T>
4 class Vector{
5 private:
6     T x, T y;
```

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

6.2 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)
28         const {
29         return data[col+row*c];
30     }
31     // implement as needed
32     bool operator==(In rhs) const {
33         return data == rhs.data;
34     }
35     M operator+(In rhs) const {
36         assert(rhs.r == r && rhs.c == c);
37         Matrix ret(r, c);
38         rep(i, 0, c*r)
39             ret.data[i] = data[i]*rhs.data[i];
40         return ret;
41     }
42     M operator-(In rhs) const {
43         assert(rhs.r == r && rhs.c == c);
44         Matrix ret(r, c);
45         rep(i, 0, c*r)
46             ret.data[i] = data[i]-rhs.data[i];
47         return ret;
```

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

6.3 Points Class

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

```

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

```

6.4 Graham Scan

```

1  struct point {
2      int x, y;
3  };
4  int det(const point& p1, const point& p2,
5         const 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&
14 // p2, 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
23         point& rhs) {
24         int d = det(p, lhs, rhs);

```

```

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

```

6.5 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
18     bool operator <(const Point<T> &o) const {
19         if (y != o.y) return y < o.y;
20         return x < o.x;
21     }
22
23     Point<T> operator -(const Point<T> &o)
24         const { return Point<T>(x - o.x, y - o.

```

```

25     T lenSq() const { return x*x + y*y; }
26 };
27
28 template <class T>
29 struct sort_less {
30     const Point<T> &ref;
31
32     sort_less(const Point<T> &p) : ref(p) {}
33
34     double angle(const Point<T> &p) const {
35         Point<T> delta = p - ref;
36         return atan2(delta.y, delta.x);
37     }
38
39     bool operator() (const Point<T> &a, const
40         Point<T> &b) const {
41         double aa = angle(a);
42         double ab = angle(b);
43         if (aa != ab) return aa < ab;
44         return (a - ref).lenSq() < (b - ref).
45             lenSq();
46     }
47 };
48
49 template <class T>
50 int ccw(const Point<T> &p1, const Point<T>
51     &p2, const Point<T> &p3) {
52     return (p2.x - p1.x) * (p3.y - p1.y) - (p2
53         .y - p1.y) * (p3.x - p1.x);
54 }
55
56 template <class T>
57 vector<Point<T> > convex_hull(vector<Point<
58     T> > 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
74     traversal.
75     Point<T> p = input[minIndex];
76     output.push_back(p);
77     input.erase(input.begin() + minIndex);
78
79     // Sort the other elements according to
80     the angle with "p"
81     sort(input.begin(), input.end(), sort_less
82         <T>(p));
83
84     // Add the first point from "input" to the
85     "output" as a candidate.
86     output.push_back(input[0]);
87
88     // Start working our way through the
89     points...
90     input.push_back(p);
91     size = input.size();
92     for (nat i = 1; i < size; i++) {
93         while (output.size() >= 2) {
94             nat last = output.size() - 1;

```



```

84     int c = ccw(output[last - 1], output[
85         last], input[i]);
86     if (c == 0) {
87         // Collinear points! Take away the
88         // closest.
89         if ((output[last - 1] - output[last
90             ]).lenSq() <= (output[last - 1]
91                 - input[i]).lenSq()) {
92             if (output.size() > 1)
93                 output.pop_back();
94             else
95                 break;
96         } else {
97             if (c < 0) {
98                 if (output.size() > 1)
99                     output.pop_back();
100                 else
101                     break;
102             } else {
103                 break;
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
128     vector<Pt> result = convex_hull(points);
129
130     printf("%d\n", (int)result.size());
131     for (nat i = 0; i < result.size(); i++) {
132         printf("%d_%d\n", result[i].x, result[i].
133             y);
134     }
135     return true;
136 }

```

```

137
138 int main() {
139     while(solve());
140 }
141 return 0;
142 }

```

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
8      r, 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 vector<int> lis(int a[], int n){
20     std::vector<int> lis;
21     if(n == 0) return lis;
22     int c[n]; memset(c, 0, sizeof(c));
23     int p[n]; memset(p, 0xFF, sizeof(p));
24     int s = 1;
25
26     c[0] = 0;
27     p[0] = -1;
28     for(int i = 1; i < n; i++){
29         if(a[i] < a[c[0]]){
30             c[0] = i;
31         }
32         else if(a[i] > a[c[s-1]]){
33             p[i] = c[s-1];
34             c[s] = i;
35             s++;
36         }
37         else{
38             int pos = bin_search(a, c, -1, s-1, a
39                 [i]);
40             p[i] = c[pos-1];
41             c[pos] = i;
42         }
43     }
44 }

```

```

45     int d = c[s-1];
46     for( int i = 0; i < s; i++){
47         lis.push_back(d);
48         d = p[d];
49     }
50
51     reverse(lis.begin(), lis.end());
52     return lis;
53 }
54
55 int main(){
56     int n;
57     while(scanf("%d", &n) == 1){
58         int a[n]; for(int i = 0; i < n; i++){
59             scanf("%d", &a[i]);
60             vector<int> lseq = lis(a, n);
61
62             printf("%d\n", (int)lseq.size());
63             for(int i = 0; i < lseq.size(); i++){
64                 printf("%d_", lseq[i]);
65             }
66             printf("\n");
67             lseq.clear();
68         }
69     }

```

7.2 Longest Increasing Substring

```

1  /* Longest common substring. */
2  int HadenIngberg(string const &s, string
3      const &t){
4      int n = s.size(), m = t.size(), best;
5      for(int i = 0; i < n-best; ++i) { // Go
6          through s
7          int cur = 0;
8          int e = min(n-i, m);
9
10         // Can best grow?
11         for(int j = 0; j < e && best+j < cur+e;
12             ++j)
13             best = max(best,
14                 cur = (s[i+j] == t[j] ? cur+1 : 0));
15     }
16
17     for(int i = 1; i < m-best; ++i) { // Go
18         through t
19         int cur = 0;
20         int e = min(m-i, n);
21         // Can best grow?
22         for(int j = 0; j < e && best+j < cur+e;
23             ++j)
24             best = max(best, cur = (t[i+j] == s[j] ? cur
25                 +1 : 0));
26     }
27     return best;
28 }

```