

# Contents

<b>1</b>	<b>Environment</b>	<b>2</b>
1.1	Template . . . . .	2
<b>2</b>	<b>Data Structures</b>	<b>2</b>
2.1	Union Find . . . . .	2
2.2	Fenwick Tree . . . . .	2
<b>3</b>	<b>Numerical</b>	<b>2</b>
3.1	Rational Numbers Class . . . . .	2
3.2	Binary Search . . . . .	2
3.3	De Bruijn . . . . .	2
3.4	Prime Generator . . . . .	3
3.5	Factorisation . . . . .	3

<b>4</b>	<b>Graphs</b>	<b>3</b>
4.1	Single Source Shortest Path . . . . .	3
4.2	Single Source Shortest Path Time Table . . . . .	3
4.3	All pairs shortest path . . . . .	4
4.4	Minimum Spanning Tree . . . . .	4
4.5	Maximum Flow . . . . .	4
4.6	Euler Tour . . . . .	4
<b>5</b>	<b>String processing</b>	<b>4</b>
5.1	String Matching . . . . .	4
<b>6</b>	<b>Geometry</b>	<b>5</b>
6.1	Points Class . . . . .	5
6.2	Graham Scan . . . . .	5
6.3	Convex Hull . . . . .	5

# 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 bool solve(){
26
27     return true;
28 }
29
30 int main(){
31     int tc=1; //scanf("%d", &tc);
32     rep(i, 0, tc) solve();
33
34     return 0;
35 }
36
```

# 2 Data Structures

## 2.1 Union Find

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

## 2.2 Fenwick Tree

```
1 #include <iostream>
2 #include <stdio.h>
3 #include <vector>
4
5 using namespace std;
6
7
8 typedef long long int lli;
9 typedef vector<lli> vi;
10
11
12 #define last_dig(x) (x & (-x))
13
14 void fenwick_create(vi &t, lli n){
15     t.assign(n + 1, 0);
16 }
17 lli fenwick_read(const vi &t, lli b){
18     lli sum = 0;
19     while(b > 0){
20         sum += t[b];
21         b -= last_dig(b);
22     }
23     return sum;
24 }
25
26 void fenwick_update(vi &t, lli k, lli v){
27     while(k <= (lli)t.size()){
28         t[k] += v;
29         k += last_dig(k);
30     }
31 }
32
33 int main(){
34     lli N, Q; scanf("%lld%lld", &N, &Q);
35     vi ft; fenwick_create(ft, N);
36
37     char op; lli a, b;
38     for(lli i = 0; i < Q; i++){
39         scanf("%*[\n\t]%c", &op);
40         switch (op){
41             case '+':
42                 scanf("%lld%lld", &a, &b);
43                 fenwick_update(ft, a+1, b);
44                 break;
45
46             case '?':
47                 scanf("%lld", &a);
48                 printf("%lld\n", fenwick_read(ft, a));
49                 break;
50         }
51     }
52
53     return 0;
54 }
```

# 3 Numerical

## 3.1 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:uden=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         ;
32         return res;
33     }
34     Q operator - (Q a){
35         Q b = * this;
36         Q res;
37         if(a==b) res = Q(0,0);
38         else res = Q((b.p * a.q - a.p * b.q), (a.q * b.
39             q));
40         return res;
41     }
42     Q operator * (Q a){
43         Q b = * this;
44         Q res = Q(a.p * b.p, a.q * b.q);
45         return res;
46     }
47
48     Q operator / (Q a){
49         Q b = * this;
50         Q res = Q(b.p * a.q, b.q * a.p);
51         return res;
52     }
53
54     bool operator == (Q a){
55         Q f = * this;
56         Q s = Q(a.p, a.q);
57         return (f.p == s.p and f.q == s.q);
58     }
59
60     void operator = (Q a){
61         this->p = a.p;
62         this->q = a.q;
63     }
64
65     void print(){
66         printf("%lld\0%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='0'; while (t == '0') scanf("%c", &t);
77         scanf("%d%d", &tp, &tn); Q b = Q(tp, tn);
78
79         switch(t){
80             case '+': (a+b).print(); break;
81             case '-': (a-b).print(); break;
82             case '*': (a*b).print(); break;
83             case '/': (a/b).print(); break;
84         }
85     }
86
87     return 0;
88 }
89
```

## 3.2 Binary Search

```
1 // Example usage of the bsearch
2 #include <stdlib>
3 #include <stdio>
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,
17         1, &check);
18
19     printf("I found: %d\n", found);
20
21     return 0;
22 }
```

## 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 = 1; j < p + 1; 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     }

```

## 3.4 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 = 0x7FFEA810
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 = 0x7FFEA810
14         }
15     }
16 }
17
18 int main(){
19     calcprimes(10000000);
20     for(int i = 0; i < 664579; i++) printf("%d\n",
21         prime[i]);

```

## 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); 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 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, const pair
16         <int, T> & b){return (a.second > b.second);};
17 };
18
19 template<class T>
20 vector<T> dijkstras(vector<pair<int, T> > G[], int
21     n, int e, int s){
22     priority_queue<pair<int, T>, vector<pair<int, T>
23         >, comp> Q;
24
25     vector<T> c; for(int i = 0; i < n; i++) c.
26         push_back(INF); c[s] = 0;
27     vector<int> p; for(int i = 0; i < n; i++) p.
28         push_back(-1);
29
30     Q.push(pair<int, T>(s, c[s]));
31     int u, sz, v; T w;
32     while(!Q.empty()){
33         u = Q.top().first; Q.pop();
34         sz = G[u].size();
35         for(int i = 0; i < sz; i++){
36             v = G[u][i].first;
37             w = G[u][i].second;
38             if(c[v] > c[u] + w){
39                 c[v] = c[u] + w;
40                 p[v] = u;
41                 Q.push(pair<int, T>(v, c[v]));
42             }
43         }
44     }
45
46     //printf("Path to follow: ");
47     //for(int i = 0; i < n; i++) printf("%d ", p[i]);
48     //printf("\n");
49
50     return c;
51 }
52
53 int main(){
54     int n, e, q, s;
55     scanf("%d%d%d%d", &n, &e, &q, &s);
56     while(n!=0 or e!=0 or q!=0 or s!=0){
57         vector<ii> G[n];
58         for(int i = 0; i < e; i++){
59             int f, t, w;
60             scanf("%d%d%d", &f, &t, &w);
61             G[f].push_back(ii(t, w));
62         }
63         vector<int> c = dijkstras(G, n, e, s);
64
65         for(int i = 0; i < q; i++) {
66             int d; scanf("%d", &d);
67             if(c[d] == INF) printf("Impossible\n");
68             else printf("%d\n", c[d]);
69         }
70         printf("\n");
71
72         scanf("%d%d%d%d", &n, &e, &q, &s);
73     }
74
75     return 0;
76 }

```

## 4.2 Single Source Shortest Path Time Table

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

```

1 #include <stdio.h>
2 #include <queue>
3 #include <vector>
4
5 #define INF 100000000
6
7 using namespace std;
8
9 struct A{
10     A(int a, int b, int c){t0=a; tn = b; w = 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){return (
20         a.second > b.second);};
21 };
22
23 vector<int> dijkstras(vector<iA> G[], int n, int e,
24     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             tn = G[u][i].second.tn;
40             t0 = G[u][i].second.t0;
41             w = G[u][i].second.w;
42
43             wt = t0 - c[u];
44             if(wt < 0 and tn == 0) continue;
45             while(wt < 0) wt+=tn;
46
47             if(c[v] > c[u] + w + wt){
48                 c[v] = c[u] + w + wt;
49                 p[v] = u;
50                 Q.push(ii(v, c[v]));
51             }
52         }
53     }
54
55     //printf("Path to follow: ");
56     //for(int i = 0; i < n; i++) printf("%d ", p[i]);
57     //printf("\n");
58
59     return c;
60 }
61
62 int main(){
63     int n, e, q, s;
64     scanf("%d%d%d%d", &n, &e, &q, &s);
65     while(n!=0 or e!=0 or q!=0 or s!=0){
66         vector<iA> G[n];
67         for(int i = 0; i < e; i++){
68             int f, t, t0, tn, w;
69             scanf("%d%d%d%d", &f, &t, &t0, &tn, &w);
70             G[f].push_back(iA(t, A(t0, tn, w)));
71         }
72         vector<int> c = dijkstras(G, n, e, s);
73
74         for(int i = 0; i < q; i++) {
75             int d; scanf("%d", &d);
76             if(c[d] == INF) printf("Impossible\n");
77             else printf("%d\n", c[d]);
78         }
79         printf("\n");
80
81         scanf("%d%d%d%d", &n, &e, &q, &s);
82     }
83
84     return 0;
85 }

```

## 4.3 All pairs shortest path

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

Time Complexity  $O(V^3)$

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

## 4.4 Minimum Spanning Tree

```
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 Tree kruskal(Vertex * v, Edge * e, int numv, int
47     n){
48     Tree ans;
49     int sum = 0;
50
51     for(int i = 0; i < numv; ++i){
52         v[i].p = &v[i];
53     }
54
55     sort(&e[0], &e[numv]);
56
57     for(int i = 0; i < numv; ++i){
58         if(v[e[i].f].root() != v[e[i].t].root()){
59             v[e[i].t].root()->p = v[e[i].f].root();
60             ans.w += e[i].w;
61
62             if(e[i].t < e[i].f) ans.e.push_back(AnsEdge(e
63                 [i].t, e[i].f));
64             else ans.e.push_back(AnsEdge(e[i].f, e
65                 [i].t));
66         }
67
68         Vertex * p = v[0].root();
69         for(int i = 0; i < numv; ++i)
70             if(p != v[i].root()){
71                 ans.complete = false;
72                 break;
73             }
74
75         sort(ans.e.begin(), ans.e.end());
76
77         return ans;
78     }
79
80 int main(){
81     int n, m; scanf("%d", &n, &m);
82     while(n or m){
83         Vertex v[n];
84         Edge e[m];
85
86         for(int i = 0; i < m; i++){
87             int f, t;
88             scanf("%d%d%d", &f, &t, &e[i].w);
89             e[i].f = f;
90             e[i].t = t;
91         }
92
93         Tree ans = mst(v, e, n, m);
94
95         if(ans.complete){
96             printf("%d\n", ans.w);
97             for(int i = 0; i < ans.e.size(); i++){
98                 printf("%d_u%d\n", ans.e[i].f, ans.e[i].t);
99             }
100         } else printf("Impossible\n");
101
102         scanf("%d", &n, &m);
103     }
104
105     return 0;
106 }
```

## 4.5 Maximum Flow

```
1 // Edmonds Karp's Maximum Flow Algorithm
```

```
2 // Input: Adjacency Matrix (res)
3 // Output: Maximum Flow
4 // Time Complexity:  $O(VE^2)$ 
5
6 int res[MAX_V][MAX_V], mf, f, s, t;
7 vi p;
8
9 void augment(int v, int minEdge){
10     if(v == s){f = minEdge; return;}
11     else if(p[v] != -1){augment(p[v], min(minEdge,
12         res[v][p[v]]));
13         res[p[v]][v] -= f; res[v][p[v]] += f; }
14 }
15
16 int solve(){
17     mf = 0; // Max Flow
18     while(1){
19         f = 0;
20         vi dist(MAX_V, INF); dist[s] = 0; queue<int> q;
21         q.push(s);
22         p.assign(MAX_V, -1);
23         while(!q.empty()){
24             int u = q.front(); q.pop();
25             if(u == t) break;
26             for(int v = 0; v < MAX_V; v++){
27                 if(res[u][v] > 0 && dist[v] == INF)
28                     dist[v] = dist[u] + 1, q.push(v), p[v] =
29                         u;
30                 augment(t, INF);
31                 if(f == 0) break;
32             }
33             mf += f;
34         }
35         printf("%d\n", mf);
36     }
```

## 4.6 Euler Tour

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

# 5 String processing

## 5.1 String Matching

```
1 // Knuth Morris Prat : Search for a string in
2 // another one
3 // Alternative STL algorithms : strstr in <cstring>
4 // find in <string>
5 // Time complexity :  $O(n)$ 
6
7 #include <stdio.h>
8 #include <string>
9
10 #define MAX_N 100010
```

```

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

```

## 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*oth, y*
        oth); }
13     Vector operator/ (T oth) { return Vector(x/oth, y/
        oth); }
14
15     Vector operator+ (Vector oth) { return Vector(x+
        oth.x, y+oth.y); }
16     Vector operator- (Vector oth) { return Vector(x+
        oth.x, y+oth.y); }
17     T operator* (Vector oth) { return x*oth.x + y*oth.
        y; }
18     Vector operator/ (Vector oth) { return Vector(x*
        oth.y-oth.x*y); }
19 };

```

### 6.2 Graham Scan

```

1 struct point {
2     int x, y;
3 };
4 int det(const point& p1, const point& p2, const
    point& p3)
5 {
6     int x1 = p2.x - p1.x;
7     int y1 = p2.y - p1.y;
8     int x2 = p3.x - p1.x;
9     int y2 = p3.y - p1.y;
10    return x1*y2 - x2*y1;
11 }
12
13 // bool ccw(const point& p1, const point& p2, const
    point& p3)
14 // { // Counterclockwise? Compare with determinant
    ...
15 // return (det(p1, p2, p3) > 0);

```

```

16 // }
17
18 struct angle_compare {
19     point p; // Leftmost lower point
20     angle_compare(const point& p) : p(p) {}
21     bool operator() (const point& lhs, const point& rhs)
    {} {
22         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 }

```

### 6.3 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; }
28 };
29
30 template <class T>
31 struct sort_less {
32     const Point<T> &ref;
33
34     sort_less(const Point<T> &p) : ref(p) {}
35
36     double angle(const Point<T> &p) const {
37         Point<T> delta = p - ref;
38         return atan2(delta.y, delta.x);
39     }
40
41     bool operator() (const Point<T> &a, const 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 }

```

```

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                     output.pop_back();
90                 }
91                 else {
92                     break;
93                 }
94             }
95             else if (c < 0) {
96                 if (output.size() > 1)
97                     output.pop_back();
98                 else {
99                     break;
100                 }
101             }
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         vector<Pt> result = convex_hull(points);
128
129         printf("%d\n", (int)result.size());
130     }

```

```
131     for (nat i = 0; i < result.size(); i++) {
132         printf("%d_%d\n", result[i].x, result[i].y);
133     }
134
135     return true;
```

```
136     }
137
138     int main() {
139         while(solve());
140     }
```

```
141     return 0;
142 }
```