

NAT: Nostalgic Alien Trespassers — TCR NWERC 2013

Filip Strömbäck, Magnus Selin, Carl Einarson

November 24, 2013

Contents

1 Environment	2	4.6 Euler Tour	7
1.1 Template	2	4.7 Bipartite Matching	7
2 Data Structures	2	4.8 Strongly Connected Components	7
2.1 Union Find	2	5 String processing	8
2.2 Fenwick Tree	2	5.1 STL	8
3 Numerical	2	5.2 String Matching	8
3.1 General Utils	2	5.3 String Multimatching	8
3.2 Rational Numbers Class	3	6 Geometry	8
3.3 Binary Search	4	6.1 Points Class	8
3.4 De Bruijn	4	6.2 Matrix Class	8
3.5 Prime Generator	4	6.3 Matrix3d Class	9
3.6 Factorisation	4	6.4 Points Class	10
4 Graphs	4	6.5 Graham Scan	11
4.1 Single Source Shortest Path	4	6.6 Convex Hull	11
4.2 Single Source Shortest Path Time Table	5	6.7 Line-point distance	12
4.3 All Pairs Shortest Path	5	6.8 Polygon Area	13
4.4 Minimum Spanning Tree	6	7 Misc	13
4.5 Maximum Flow	6	7.1 Longest Increasing Subsequence	13
		7.2 Longest Increasing Substring	14

1 Environment

1.1 Template

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

2 Data Structures

2.1 Union Find

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

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

2.2 Fenwick Tree

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

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

3 Numerical

3.1 General Utils

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

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

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

3.2 Rational Numbers Class

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

```

27
28 Q operator + (Q a){
29     Q b = * this;
30     Q res = Q((a.p * b.q + b.p * a.q), (
31         a.q * b.q));
32     return res;
33 }
34 Q operator - (Q a){
35     Q b = * this;
36     Q res;
37     if(a==b) res = Q(0,0);
38     else res = Q((b.p * a.q - a.p * b.q)
39         , (a.q * b.q));
40     return res;
41 }
42 Q operator * (Q a){
43     Q b = * this;
44     Q res = Q(a.p * b.p, a.q * b.q);
45     return res;
46 }
47
48 Q operator / (Q a){
49     Q b = * this;
50     Q res = Q(b.p * a.q, b.q * a.p);
51     return res;
52 }
53
54 bool operator == (Q a){
55     Q f = * this;
56     Q s = Q(a.p, a.q);
57     return (f.p == s.p and f.q == s.q);
58 }
59
60 void operator = (Q a){
61     this->p = a.p;
62     this->q = a.q;
63 }
64
65 void print(){
66     printf("%lld_/%lld\n", p, q);
67 }
68 };
69
70 int main(){
71     int n; scanf("%d", &n);
72     for(int i = 0; i < n; i++){
73         int tp, tn;
74         scanf("%d%d", &tp, &tn); Q a = Q(tp,

```

```

tn);
75
76     char t='_'; while (t == '_') scanf("
77     %c", &t);
78
79     scanf("%d%d", &tp, &tn); Q b = Q(tp,
80     tn);
81
82     switch(t){
83         case '+': (a+b).print(); break;
84         case '-': (a-b).print(); break;
85         case '*': (a*b).print(); break;
86         case '/': (a/b).print(); break;
87     }
88
89     return 0;

```

3.3 Binary Search

```

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

```

3.4 De Bruijn

```

1
2 #include <iostream>
3 #include <vector>
4 #include <cmath>

```

```

5
6 using namespace std;
7 vector<bool> seq;
8 vector<bool> a;
9 int n, k;
10
11 void db(int t, int p){
12     if (t > n){
13         if (n % p == 0)
14             for (int j = 1; j < p + 1; j++)
15                 seq.push_back(a[j]);
16     }
17     else{
18         a[t] = a[t - p];
19         db(t + 1, p);
20         for (int j = a[t - p] + 1; j < 2; j
21             ++){
22             a[t] = j;
23             db(t + 1, t);
24         }
25     }
26 }
27
28 int de_bruijn(){
29     for(int i = 0; i < n; i++)
30         a.push_back(0);
31     db(1, 1);
32
33     int sum = 0;
34     for(int i = 0; i < n; i++){
35         sum += seq[(k+i) % (int)pow((double)
36             2, n)] * pow((double)2, n-i-1);
37     }
38     cout << sum << '\n';
39 }
40
41 int main(){
42     int tc;
43     cin >> tc;
44     for(int we = 0; we < tc; we++){
45         cin >> n >> k;
46         a.clear(); seq.clear();
47         de_bruijn();
48     }
49 }

```

3.5 Prime Generator

```

1 #include <cstdio>
2

```

```

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

```

3.6 Factorisation

```

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

```

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<int, T> & b){return (a.
            second > b.second);}
16 };
17
18 template<class T>
19 vector<T> dijkstras(vector<pair<int, T>
    > G[], int n, int e, int s){
20     priority_queue<pair<int, T> , vector<
        pair<int, T> >, comp> Q;
21
22     vector<T> c; for(int i = 0; i < n; i
        ++ ) c.push_back(INF); c[s] = 0;
23     vector<int> p; for(int i = 0; i < n; i
        ++ ) p.push_back(-1);
24
25     Q.push(pair<int, T>(s, c[s]));
26     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 ", p[i]);
44     //printf("\n");
45
46     return c;
47 }
48
49 int main(){
50     int n, e, q, s;
51     scanf("%d%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%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 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 (a.second > b.second);}
20  };
21
22  vector<int> dijkstras(vector<iA> G[],
        int n, int e, int s){
23      priority_queue<ii, vector<ii>, comp> Q
        ;
24
25      vector<int> c; for(int i = 0; i < n; i
        ++ ) c.push_back(INF); c[s] = 0;
26      vector<int> p; for(int i = 0; i < n; i
        ++ ) p.push_back(-1);
27
28      Q.push(ii(s, c[s]));
29      int u, sz, v, t0, tn, w, wt;
30      while(!Q.empty()){
31
32          u = Q.top().first; Q.pop();
33          sz = G[u].size();
34          for(int i = 0; i < sz; i++){
35              v = G[u][i].first;
36              tn = G[u][i].second.tn;
37              t0 = G[u][i].second.t0;
38              w = G[u][i].second.w;
39
40              wt = t0 - c[u];
41              if (wt < 0 and tn == 0) continue;
42              while(wt < 0) wt+=tn;
43
44              if( c[v] > c[u] + w + wt){
45                  c[v] = c[u] + w + wt;
46                  p[v] = u;
47                  Q.push(ii(v, c[v]));
48              }
49          }
50      }
51
52      //printf("Path to follow: ");

```

```

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

```

4.3 All Pairs Shortest Path

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

```

1  // All pairs shortest path (Floyd
    Warshall). Assign nodes which are
    part of a
2  // negative cycle to minus infinity.
3
4  #include <stdio.h>
5  #include <iostream>
6  #include <vector>
7  #include <algorithm>

```

```

8
9  #define INF 1000000000
10 using namespace std;
11
12 template<class T>
13 vector< vector<T>> floyd_warshall(
    vector< vector<T>> d){
14     int n = d.size();
15     for(int i = 0; i < n; i++) d[i][i] =
        0;
16
17     for (int k = 0; k < n; k++)
18         for (int i = 0; i < n; i++)
19             for (int j = 0; j < n; j++)
20                 if (d[i][k] != INF and d[k][j] !=
        INF)
21                     d[i][j] = min(d[i][j], d[i][k]+d
        [k][j]);
22
23     for(int i = 0; i < n; i++)
24         for(int j = 0; j < n; j++)
25             for(int k = 0; d[i][j] != -INF &&
        k < n; k++)
26                 if(d[i][k] != INF && d[k][j] !=
        INF && d[k][k] < 0)
27                     d[i][j] = -INF;
28
29     return d;
30 }
31
32 int main(){
33     int n, m, q; scanf("%d%d%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

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, int nume){
48     Tree ans;
49     int sum = 0;
50
51     for(int i = 0; i < numv; ++i){
52         v[i].p = &v[i];
53     }
54
55     sort(&e[0], &e[nume]);
56
57     for(int i = 0; i < nume; ++i){
58         if(v[e[i].f].root() != v[e[i].t].
            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[i].t, e[i].f));
63             else ans.e.push_back(
                AnsEdge(e[i].f, e[i].t));
64         }
65     }
66
67     Vertex * p = v[0].root();
68     for(int i = 0; i < numv; ++i)
69         if(p != v[i].root()){
70             ans.complete = false;
71             break;
72         }
73
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%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].t);
98             }
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
            (minEdge, res[v][p[v]]));
7          res[p[v]][v] -= f; res[v][p[
            v]] += f; }
8  }
9
10 int solve(){
11     mf = 0; // Max Flow
12

```

```

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

```

4.6 Euler Tour

Time Complexity $O(E + V)$

```

1 #include <cstdlib>
2 #include <cstdio>
3 #include <cmath>
4 #include <list>
5
6 typedef vector<int> vi;
7
8 using namespace std;
9
10 list<int> cyc;
11
12 void euler_tour(list<int>::iterator i,
13     int u) {
14     for(int j = 0; j < (int)AdjList[u].
15         size(); j++){
16         ii v = AdjList[u][j];
17         if (v.second){
18             v.second = 0;
19             for(int k = 0; k < (int)AdjList[u]
20                 .size(); k++){
21                 ii uu = AdjList[v.first][k];
22                 if(uu.first == u && uu.second) {
23                     uu.second = 0; break;
24                 }
25             }
26         }
27     }
28 }

```

```

21 euler_tour(cyc.insert(i, u), v.
22     first)
23 }
24 }
25
26 int main(){
27     cyc.clear();
28     euler_tour(cyc.begin(), A);
29     for(list<int>::iterator it = cyc.begin
30         (); it != cyc.end(); it++){
31         printf("%d\n", *it);
32     }
33 }

```

4.7 Bipartite Matching

```

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

```

```

28 if (match[j] == -1) return true;
29 visited[j] = true; int di = match[j];
30 trav(e, g[di])
31 if (!visited[*e] && find(*e, g)) {
32     match[*e] = di;
33     match[j] = -1;
34     return true;
35 }
36 return false;
37 }
38 int dfs_matching(Graph &g, int n, int m)
39 {
40     match.assign(m, -1);
41     rep(i, 0, n) {
42         visited.assign(m, false);
43         trav(j, g[i])
44         if (find(*j, g)) {
45             match[*j] = i;
46             break;
47         }
48     }
49     return m - count(match.begin(), match.
50         end(), -1);
51 }

```

4.8 Strongly Connected Components

```

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

```



```

18     if(p->visited) return;
19     p->visited = true;
20     if(curRes) curRes->push_back(p);
21     for(vector<vertex*>::iterator it
22           = p->to.begin();
23           it != p->to.end(); ++it)
24         dfs(*it);
25     *(visitIt++) = p;
26 }
27 void run()
28 {
29     sorted.resize(v.size());
30     visitIt = sorted.rbegin();
31     for(vector<vertex>::it it = v.
32           begin();
33           it != v.end(); ++it)
34         it->visited = false;
35     for(vector<vertex>::it it = v.
36           begin();
37           it != v.end(); ++it)
38         dfs(&(*it));
39     for(vector<vertex>::it it = v.
40           begin();
41           it != v.end(); ++it)
42     {
43         it->visited = false;
44         it->from.swap(it->to);
45     }
46     for(vector<vertex>::iterator it
47           = sorted.begin();
48           it != sorted.end(); ++it)
49         if(!(*it)->visited)
50         {
51             curRes = &(*res.
52                       insert(res.
53                             end()));
54             dfs(&(*it));
55         }
56 }

```

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:" <<
6          found << '\n';
7  str.replace(str.find(str2),str2.length()
8              ,"new_word");

```

5.2 String Matching

```

1  // Knuth Morris Prat : Search for a
2  // string in another one
3  // Alternative STL algorithms : strstr
4  // in <cstring> find in <string>
5  // Time complexity : O(n)
6
7  #include <cstdio>
8  #include <cstring>
9
10 #define MAXN 100010
11
12 char T[MAXN], P[MAXN]; // T = text, P
13 = pattern
14
15 int b[MAXN], n, m; // b = back
16 table, n = length of T, m = length of
17 P
18
19 void kmpPreprocess() {
20     int i = 0, j = -1; b[0] = -1;
21     while (i < m){
22         while(j >= 0 && P[i] != P[j]) j = b[j];
23         i++; j++;
24         b[i] = j;
25     }
26 }
27
28 void kmpSearch() {
29     int i = 0, j = 0;
30     while(i < n){
31         while(j >= 0 && T[i] != P[j]) j = b[j];
32         i++; j++;
33         if(j==m){
34             printf("P_is_found_at_index%d\n", i - j);
35             j = b[j];
36         }
37     }
38 }
39
40 int main(){

```

```

35     strcpy(T, "asdhasdhejasdasdhejasdasd")
36     ;
37     strcpy(P, "hej");
38
39     n = 25; m = 3;
40
41     kmpPreprocess();
42     kmpSearch();
43
44     return 0;
45 }

```

5.3 String Multimatching

6 Geometry

6.1 Points Class

```

1  #include <cmath>
2
3  template<class T>
4  class Vector{
5  public:
6
7      T x, y;
8      Vector(){};
9      Vector(T a, T b){x = a; y = b};
10
11      T abs(){return sqrt(x*x+y*y);}
12      Vector operator* (T oth){ return
13          Vector(x*oth, y*oth); }
14      Vector operator/ (T oth){ return
15          Vector(x/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
22          .x + y*oth.y; }
23      Vector operator* (Vector oth){ return
24          Vector(x*oth.y-oth.x*y)}
25 };

```

6.2 Matrix Class

```

1  /* Description: Untested matrix
2  implementation
3  * Source: Benjamin Ingberg */
4  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(
11        r_),
12        c(c_), data(r*c_, v) { }
13    explicit Matrix(Pt3<T> in)
14        : r(3), c(1), data(3*1) {
15        rep(i, 0, 3)
16        data[i] = in[i];
17    }
18    explicit Matrix(Pt2<T> in)
19        : r(2), c(1), data(2*1) {
20        rep(i, 0, 2)
21        data[i] = in[i];
22    }
23    // copy constructor, assignment
24    // and destructor compiler defined
25    T & operator()(int row, int col) {
26        return data[col+row*c];
27    }
28    T const & operator()(int row, int col)
29        const {
30        return data[col+row*c];
31    }
32    // implement as needed
33    bool operator==(In rhs) const {
34        return data == rhs.data;
35    }
36    M operator+(In rhs) const {
37        assert(rhs.r == r && rhs.c == c);
38        Matrix ret(r, c);
39        rep(i, 0, c*r)
40        ret.data[i] = data[i]*rhs.data[i];
41        return ret;
42    }
43    M operator-(In rhs) const {
44        assert(rhs.r == r && rhs.c == c);
45        Matrix ret(r, c);
46        rep(i, 0, c*r)
47        ret.data[i] = data[i]-rhs.data[i];
48        return ret;
49    }
50    M operator*(In rhs) const { // matrix
51        mult
52        assert(rhs.r == c);
53        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)
55        *rhs(k,j);
56    }
57    M operator*(T rhs) const { // scalar
58        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
66 matrix
67 Matrix<T> id(int r, int c) {
68     Matrix<T> m(r,c);
69     rep(i, 0, r)
70     m(i,i) = T(1);

```

6.3 Matrix3d Class

```

1  /* 3 dimensional matrix class
2   * with Gauss Elimination and
3   * Eigenvectors
4   * Source: Magnus Selin
5   */
6  #include <cmath>
7
8  class Matrix3d{
9      friend std::ostream& operator<< ( std
10      ::ostream& os, Matrix3d fb );
11  private:
12      double a[3][3];
13  public:
14      Matrix3d(){
15          for(int i = 0; i < 3; i++){
16              for(int j = 0; j < 3; j++){
17                  if(j >= 0 && j < 3 && i >= 0 &&
18                  i < 3) a[i][j] = 0;
19              }
20          }

```

```

21      double get(int i, int j) { if(j
22      >= 0 && j < 3 && i >= 0 && i < 3)
23      return a[i][j]; else std::cerr << "
24      Out_of_bounds!\n"; }
25      void set(int i, int j, int v) { if(j
26      >= 0 && j < 3 && i >= 0 && i < 3) a[i
27      ][j] = v; else std::cerr << "Out_
28      of_bounds!\n"; };
29
30      void chg_row(int x, int y);
31      void mult_row(int x, double c);
32      void add_row(int x, int y, double c);
33
34      Matrix3d gauss();
35      Matrix3d get_inverse();
36
37      double get_det();
38      void get_eigenvectors();
39
40      Matrix3d operator= (Matrix3d);
41      Matrix3d operator+ (Matrix3d);
42      Matrix3d operator- (Matrix3d);
43      Matrix3d operator* (Matrix3d);
44      Matrix3d operator* (double);
45      };
46
47      void Matrix3d::chg_row(int x, int y){
48          int temp[3] = {a[x][0], a[x][1], a[x
49          ][2]};
50
51          for(int i = 0; i < 3; i++){
52              a[x][i] = a[y][i];
53              a[y][i] = temp[i];
54          }
55      }
56      void Matrix3d::mult_row(int x, double c)
57      {
58          for(int i = 0; i < 3; i++){
59              a[x][i] *= c;
60          }
61      }
62      void Matrix3d::add_row(int x, int y,
63      double c){
64          for(int i = 0; i < 3; i++){
65              a[x][i] += c * a[y][i];
66          }
67      }
68      void Matrix3d::get_eigenvectors(){

```

```

62 double eig[3];
63 double p = a[0][1] * a[0][1] + a[0][2]
    * a[0][2] + a[1][2] * a[1][2];
64 double q, r, phi;
65 Matrix3d B; Matrix3d I;
66 for (int i = 0; i < 3; i++) I.set(i, i
    , 1);
67
68 if ( p == 0 ){
69     eig[0] = a[0][0];
70     eig[1] = a[1][1];
71     eig[2] = a[2][2];
72 }
73 else {
74     q = (a[0][0] + a[1][1] + a[2][2]) /
    3;
75     p = (a[0][0] - q) * (a[0][0] - q) +
    (a[1][1] - q) * (a[1][1] - q) +
76     (a[2][2] - q) * (a[2][2] - q) + 2
77     * q;
78     p = sqrt( p / 6 );
79
80     B = ((*this) - I * q);
81     B = B * (1 / p);
82     r = B.get_det();
83
84     if (r <= -1)
85         phi = M_PI / 3;
86     else if (r >= 1)
87         phi = 0;
88     else
89         phi = acos(r) / 3;
90
91     eig[0] = q + 2 * p * cos(phi);
92     eig[2] = q + 2 * p * cos(phi + M_PI
    * (2/3));
93     eig[1] = 3 * q - eig[0] - eig[2];
94 }
95
96 std::cout << eig[0] << '\u' << eig[1]
    << '\u' << eig[2] << '\u';
97
98 for (int i = 0; i < 3; i++) {
99     Matrix3d temp = (*this);
100
101     temp.set(0, 0, temp.get(0, 0) - eig[
    i]);
102     temp.set(1, 1, temp.get(1, 1) - eig[
    i]);

```

```

103     temp.set(2, 2, temp.get(2, 2) - eig[
    i]);
104     temp = temp.gauss();
105
106     std::cout << "Temp\u" << i << ":\n"
    << temp << "\n";
107 }
108 }
109
110 double Matrix3d::get_det(){
111     return a[0][0] * a[1][1] * a[2][2] + a
    [0][1] * a[1][2] * a[2][0] + a[0][2]
    * a[1][0] * a[2][1] -
112     a[0][2] * a[1][1] * a[2][0] - a
    [0][1] * a[1][0] * a[2][2] - a[0][0]
    * a[1][2] * a[2][1];
113 }
114
115 Matrix3d Matrix3d::gauss(){
116     Matrix3d * temp = new Matrix3d;
117     temp = this;
118
119     for (int i = 0; i < 3; i++){
120         if(temp->get(i, i) == 0){
121             for (int j = i; j < 3; j++){
122                 if(temp->get(j, i) != 0){
123                     temp->chg_row(i, j);
124                     break;
125                 }
126             }
127         }
128
129         if(temp->get(i, i) == 0){
130             std::cout << "Parameter\usolotion
    !!\n";
131             break;
132         }
133     }
134
135     double mult_val = temp->get(i, i);
136     temp->mult_row(i, 1 / mult_val);
137
138     for (int j = 0; j < 3; j++){
139         if(i != j){
140             double mult_val = -temp->get(j,
    i);
141
142             temp->add_row(j, i, mult_val);
143         }
144     }

```

```

145 }
146
147 std::cout << "Temp\u" << ":\n" << *temp
    << "\n";
148 std::cout << "This\u" << ":\n" << *this
    << "\n";
149
150 return *temp;
151 }
152
153 Matrix3d Matrix3d::get_inverse(){
154     Matrix3d temp = (*this), inverse;
155     for (int i = 0; i < 3; i++) inverse.
    set(i, i, 1);
156
157     for (int i = 0; i < 3; i++){
158         if(temp.get(i, i) == 0){
159             for (int j = i; j < 3; j++){
160                 if(temp.get(j, i) != 0){
161                     temp.chg_row(i, j);
162                     inverse.chg_row(i, j);
163
164                     std::cout << "Change\urow\u" <<
    i << "\uand\u" << j << ":\n";
165                     std::cout << temp << '\n';
166                     std::cout << inverse << '\n';
167
168                     break;
169                 }
170             }
171
172             if(temp.get(i, i) == 0){
173                 std::cout << "Singularity!\n";
174                 break;
175             }
176         }
177
178         double mult_val = temp.get(i, i);
179         temp.mult_row(i, 1 / mult_val);
180         inverse.mult_row(i, 1 / mult_val);
181
182         std::cout << "Divide\urow\u" << i << "
    \uby\u" << mult_val << ":\n";
183         std::cout << temp << '\n';
184         std::cout << inverse << '\n';
185
186         for (int j = 0; j < 3; j++){
187             if(i != j){
188
189

```

```

190     double mult_val = -temp.get(j, i
    );
191
192     temp.add_row(j, i, mult_val);
193     inverse.add_row(j, i, mult_val);
194
195     std::cout << "Multiply_row" <<
    i << "by" << mult_val << "and
    adding_it_to" << j << ".\n";
196     std::cout << temp << '\n';
197     std::cout << inverse << '\n';
198 }
199 }
200 }
201
202 return inverse;
203 }
204
205 Matrix3d Matrix3d::operator= (Matrix3d
    param){
206     Matrix3d temp;
207     for(int i = 0; i < 3; i++){
208         for(int j = 0; j < 3; j++){
209             temp.set(i, j, param.get(i, j));
210         }
211     }
212
213     return temp;
214 }
215
216 Matrix3d Matrix3d::operator+ (Matrix3d
    param){
217     Matrix3d temp;
218     for(int i = 0; i < 3; i++){
219         for(int j = 0; j < 3; j++){
220             temp.set(i, j, a[i][j] + param.get
                (i, j));
221         }
222     }
223
224     return temp;
225 }
226
227 Matrix3d Matrix3d::operator- (Matrix3d
    param){
228     Matrix3d temp;
229     for(int i = 0; i < 3; i++){
230         for(int j = 0; j < 3; j++){
231             temp.set(i, j, a[i][j] - param.get
                (i, j));
232         }
233     }

```

```

232     return temp;
233 }
234
235 Matrix3d Matrix3d::operator* (double
    param){
236     Matrix3d temp;
237     for(int i = 0; i < 3; i++){
238         for(int j = 0; j < 3; j++){
239             temp.set(i, j, a[i][j] * param);
240         }
241     }
242
243     return temp;
244 }
245
246 Matrix3d Matrix3d::operator* (Matrix3d
    param){
247     Matrix3d temp;
248
249     for(int i = 0; i < 3; i++){
250         for(int j = 0; j < 3; j++){
251             temp.set(i, j, a[i][0] * param.get
                (0,j) +
                a[i][1] * param.get(1,j) +
                a[i][2] * param.get(2,j) )
252             ;
253         }
254     }
255
256     return temp;
257 }
258
259 std::ostream& operator << ( std::ostream
    & os, Matrix3d m ){
260     for(int i = 0; i < 3; i++){
261         os << "(";
262         for(int j = 0; j < 3; j++){
263             os << m.get(i, j) << ",\t";
264         }
265         os << ")\n";
266     }
267     return os;
268 }

```

6.4 Points Class

```

1  /* Description: Untested homogenous
2     coordinates
3     * transformation geometry.
4     * Source: Benjamin Ingberg

```

```

4     * Usage: Requires homogenous
5     coordinates, handles
6     * multiple rotations, translations and
7     scaling in a
8     * high precision efficient manner (
9     matrix
10    * multiplication) with homogenous
11    coordinates.
12    * Also keeps reverse transformation
13    available. */
14
15 namespace h { // avoid name collisions
16
17 struct Transform {
18     enum ActionType {
19         Scale, Rotate, TranslateX, TranslateY
20     };
21     typedef tuple<ActionType, fp> Action;
22     typedef Matrix<fp> M;
23     typedef vector<Action> History;
24     History hist;
25     M to, from;
26     Transform(History h = History())
27     : to(id<fp>(3,3)), from(id<fp>(3,3))
28     {
29         doTransforms(h);
30     }
31     H transformTo(H in) {
32         return H(to*M(in));
33     }
34     H transformFrom(H in) {
35         return H(from*M(in));
36     }
37     Transform & scale(fp s) {
38         doTransform(Scale, s);
39     }
40     Transform & translate(fp dx, fp dy) {
41         doTransform(TranslateX, dx);
42         doTransform(TranslateY, dy);
43     }
44     Transform & rotate(fp phi) {
45         doTransform(Rotate, phi);
46     }
47     void doTransforms(History & h) {
48         trav(it, h) {
49             doTransform(get<0>(*it), get<1>(*it)
50             );
51         }
52     }
53     void doTransform(ActionType t, fp v) {
54         hist.push_back(make_tuple(t, v));
55         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     ism(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 }

```

6.5 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)

```

```

14 // { // Counterclockwise? Compare with
15     determinant...
16 // return (det(p1, p2, p3) > 0);
17 // }
18 struct angle_compare {
19     point p; // Leftmost lower point
20     angle_compare(const point& p) : p(p) {
21     }
22     bool operator()(const point& lhs, const
23                     point& 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,
33               int* res)
34 { // Returns number of points in the
35   convex polygon
36   int best = 0; // Find the first
37   leftmost lower 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
42           < p[best].x))
43           best = i;
44   }
45   sort(p.begin(), p.end(), angle_compare(
46       p[best]));
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
53       counter clockwise
54       while(n > 2 && det(res[n-2], res[n
55           -1], i) < 0)
56           --n; // Keep if det = 0, i.e.
57               the same 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
15     Point(T x = T(), T y = T()) : x(x), y(y)
16     {}
17
18     bool operator <(const Point<T> &o)
19         const {
20         if (y != o.y) return y < o.y;
21         return x < o.x;
22     }
23
24     Point<T> operator -(const Point<T> &o)
25         const { return Point<T>(x - o.x, y -
26             o.y); }
27     Point<T> operator +(const Point<T> &o)
28         const { return Point<T>(x + o.x, y +
29             o.y); }
30
31     T lenSq() const { return x*x + y*y; }
32 };
33
34 template <class T>
35 struct sort_less {
36     const Point<T> &ref;
37
38     sort_less(const Point<T> &p) : ref(p)
39     {}
40
41     double angle(const Point<T> &p) const {
42     Point<T> delta = p - ref;
43     return atan2(delta.y, delta.x);
44     }
45
46     bool operator() (const Point<T> &a,
47                     const Point<T> &b) const {
48     double aa = angle(a);

```

```

41     double ab = angle(b);
42     if (aa != ab) return aa < ab;
43     return (a - ref).lenSq() < (b - ref).
        lenSq();
44 }
45 };
46
47 template <class T>
48 int ccw(const Point<T> &p1, const Point<
    T> &p2, const Point<T> &p3) {
49     return (p2.x - p1.x) * (p3.y - p1.y) -
        (p2.y - p1.y) * (p3.x - p1.x);
50 }
51
52 template <class T>
53 vector<Point<T> > convex_hull(vector<
    Point<T> > input) {
54     if (input.size() < 2) return input;
55     nat size = input.size();
56
57     vector<Point<T> > output;
58
59     // Find the point with the lowest x and
        y value.
60     int minIndex = 0;
61     for (int i = 1; i < size; i++) {
62         if (input[i] < input[minIndex]) {
63             minIndex = i;
64         }
65     }
66
67     // This is the "root" point in our
        traversal.
68     Point<T> p = input[minIndex];
69     output.push_back(p);
70     input.erase(input.begin() + minIndex);
71
72     // Sort the other elements according to
        the angle with "p"
73     sort(input.begin(), input.end(),
        sort_less<T>(p));
74
75     // Add the first point from "input" to
        the "output" as a candidate.
76     output.push_back(input[0]);
77
78     // Start working our way through the
        points...
79     input.push_back(p);
80     size = input.size();

```

```

81     for (nat i = 1; i < size; i++) {
82         while (output.size() >= 2) {
83             nat last = output.size() - 1;
84             int c = ccw(output[last - 1], output[
                last], input[i]);
85
86             if (c == 0) {
87                 // Colinear points! Take away
                    the closest.
88                 if ((output[last - 1] - output[
                    last]).lenSq() <= (output[
                    last - 1] - input[i]).lenSq()
                    ) {
89                     if (output.size() > 1)
90                         output.pop_back();
91                     else
92                         break;
93                 } else {
94                     break;
95                 }
96             } else if (c < 0) {
97                 if (output.size() > 1)
98                     output.pop_back();
99                 else
100                     break;
101             } else {
102                 break;
103             }
104         }
105
106         // Do not take the last point twice.
107         if (i < size - 1)
108             output.push_back(input[i]);
109     }
110
111     return output;
112 }
113
114 typedef Point<int> Pt;
115
116 bool solve() {
117     nat count;
118     scanf("%d", &count);
119
120     if (count == 0) return false;
121
122     vector<Pt> points(count);
123     for (nat i = 0; i < count; i++) {

```

```

125         scanf("%d%d", &points[i].x, &points[i]
            ].y);
126     }
127
128     vector<Pt> result = convex_hull(points)
        ;
129
130     printf("%d\n", (int)result.size());
131     for (nat i = 0; i < result.size(); i++)
132     {
133         printf("%d%d\n", result[i].x, result[
            i].y);
134     }
135     return true;
136 }
137
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 { return Point<T>(x - o.x, y -
            o.y); }
21     Point<T> operator /(T o) const { return
        Point<T>(x / o, y / o); }
22     T operator |(const Point &o) const {

```

```

23     return x * o.x + y * o.y;
24 }
25 };
26
27
28 template <class T>
29 class Vector {
30 public:
31     T x, y, z;
32
33     Vector() : x(), y(), z() {}
34     Vector(const Point<T> &pt, T z) : x(pt.x), y(pt.y), z(z) {}
35     Vector(T x, T y, T z) : x(x), y(y), z(z) {}
36
37     Vector<T> operator -(const Vector &o) const { return Vector<T>(x - o.x, y - o.y, z - o.z); }
38     Vector<T> operator /(T o) const { return Vector<T>(x / o, y / o, z / o); }
39     T operator |(const Vector &o) const { return x * o.x + y * o.y + z * o.z; }
40     Vector<T> operator %(const Vector &o) const { return Vector<T>(y*o.z - z*o.y, z*o.x - x*o.z, x*o.y - y*o.x); }
41 }
42 };
43
44 // distance between two points or vectors.
45 template <class T>
46 T dist(const Point<T> &a, const Point<T> &b) {
47     Point<T> d = a - b;
48     return sqrt(d | d);
49 }
50
51 // Normalize a line
52 template <class T>
53 void normLine(Vector<T> &v) {
54     T l = sqrt(v.x * v.x + v.y * v.y);
55     v = v / l;
56 }
57
58 // Normalize a point
59 template <class T>
60 void normPoint(Vector<T> &v) {

```

```

61     v = v / v.z;
62 }
63
64 template <class T>
65 T dist(const Point<T> &point, const Point<T> &lineFrom, const Point<T> &lineTo) {
66     // Outside first endpoint?
67     if (((point - lineFrom) | (lineTo - lineFrom)) < 0) {
68         return dist(point, lineFrom);
69     }
70
71     // Outside second endpoint?
72     if (((point - lineTo) | (lineFrom - lineTo)) < 0) {
73         return dist(point, lineTo);
74     }
75
76     // Ok, in the middle of the line!
77
78     // Create the homogenous representation of the line...
79     Vector<T> line = Vector<T>(lineFrom, 1) % Vector<T>(lineTo, 1);
80
81     // The signed distance is then the dot product of the line
82     // and the point.
83     normLine(line);
84     T distance = Vector<T>(point, 1) | line;
85
86     // Don't return negative distances...
87     return abs(distance);
88 }
89
90 vector<Point<double>> readPoints() {
91     nat size = 0;
92     scanf("%d", &size);
93
94     vector<Point<double>> result;
95
96     for (nat i = 0; i < size; i++) {
97         double x, y;
98         scanf("%lf%lf", &x, &y);
99         result.push_back(Point<double>(x, y));
100     }
101
102     return result;
103 }

```

```

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

```

6.8 Polygon Area

```

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

```



```

10     area += (v[i] % v[i+1]).z;
11     area += (v[v.size()-1] % v[0]).z;
12     return area;
13 }

```

7 Misc

7.1 Longest Increasing Subsequence

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <vector>
4  #include <algorithm>
5
6  using namespace std;
7  int bin_search(int a[], int t[], int l,
8               int 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
39                             -1, a[i]);
40         p[i] = c[pos-1];
41         c[pos] = i;
42     }
43 }
44
45 int d = c[s-1];
46 for( int i = 0; i < s; i++ ){
47     lis.push_back(d);
48     d = p[d];
49 }
50
51 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             {
65                 printf("%d_", lseq[i]);
66             }
67     }
68 }

```

```

65     printf("\n");
66
67     lseq.clear();
68 }
69 }

```

7.2 Longest Increasing Substring

```

1  /* Longest common substring. */
2  int HadenIngberg(string const & s,
3                  string const & t){
4      int n = s.size(), m = t.size(), best;
5      for(int i = 0; i < n-best; ++i) { //
6          Go 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
12             +e; ++j)
13             best = max(best,
14                 cur = (s[i+j] == t[j] ? cur+1 : 0))
15             ;
16     }
17
18     for(int i = 1; i < m-best; ++i) { //
19         Go through t
20         int cur = 0;
21         int e = min(m-i, n);
22         // Can best grow?
23         for(int j = 0; j < e && best+j < cur
24             +e; ++j)
25             best = max(best, cur=(t[i+j] == s[j]?
26                 cur+1:0));
27     }
28     return best;
29 }

```