UoS ACM Notebook

1 Data Structures

1.1 Binary indexed tree

```
int AIB[Nmax],v[Nmax],M,x,y,z;
int N,num;

inline int zeros(int x) {
  return ((x ^ (x - 1)) & x );
}
inline void Add(int x, int q) {
  for (int i = x; i <= N; i += zeros(i))
    AIB[i]+=q;
}
inline int comp(int x) {
  int i, ret = 0;
  for (i = x; i > 0; i -= zeros(i))
    ret +=AIB[i];
  return ret;
}
```

1.2 Segment tree

```
long long aint[4*Nmax+100],v[Nmax],maxim,M,x,z,y,indic;
long long up[4*Nmax+100];
long long ind[4*Nmax+100];
long long SUM,SUMI;
long long N;
long long inf = (long long)101010000*100;

inline void relax(int nod,int st,int dr) {
  long long mij=(st+dr)/2;
  long long val=up[nod];
  if(st!=dr) {
    up[2*nod]+=val;
    up[2*nod+1]+=val;
  }
  if(st==dr)
    ind[nod]=st;
  aint[nod]+=up[nod];
  up[nod]=0;
}
```

```
void update(int nod,int ist,int idr,int st,int dr,long long val) {
  if(ist<=st&&idr>=dr) {
    aint[nod]+=val;
    if(st!=dr) {
      up[2*nod]+=val;
      up[2*nod+1]+=val;
    }
    else ind[nod]=ist;
  } else {
    if(aint[nod]>0)
      relax(nod,st,dr);
    long long mij=(st+dr)/2;

    if(ist<=mij)
      update(2*nod,ist,idr,st,mij,val);
    if(idr>mij)
      update(2*nod+1,ist,idr,mij+1,dr,val);
    if(up[nod*2])
      relax(nod*2,st,mij);
    if(up[nod*2+1])
      relax(nod*2+1,mij+1,dr);
    if(aint[nod*2]>aint[nod*2+1]) {
      aint[nod]=aint[2*nod+1];
      ind[nod]=ind[2*nod+1];
    } else {
      aint[nod]=aint[2*nod];
      ind[nod]=ind[2*nod];
    }
  }
}
```

1.3 Treap

```
struct T {
  int key, priority,nr;
  T *left, *right;
  T() {}
  T(int key, int priority, T* left, T* right) {
    this->key = key;
    this->priority = priority;
    this->left = left, this->right = right;
    this->nr = 0;
  }
} *R, *nil; // nil indica un nod 'gol'

void init(T* &R) {
  srand(unsigned(time(0)));
  R = nil = new T(0, 0, NULL, NULL);
}
void parc(T* n){
  if(n== nil)
    return;
  parc(n->left);
```

```
  parc(n->right);
}
inline void update(T* &n){
  if(n==nil)
    return;
  n->nr = n->left->nr + n->right->nr + 1;

}
int search(T* n, int key) {
  if (n == nil) return 0;
  if (key == n->key) return 1;
  if (key < n->key)
    return search(n->left, key);
  else
    return search(n->right, key);
  update(n->right);
  update(n->left);
  update(n);
}
void rotleft(T* &n) {
  T *t = n->left;
  n->left = t->right, t->right = n;
  n = t;
  update(t->right);
  update(t->left);
  update(t);
  update(n->right);
  update(n);
}
void rotright(T* &n) {
  T *t = n->right;
  n->right = t->left, t->left = n;
  n = t;
  update(t->right);
  update(t->left);
  update(t);
  update(n->left);
  update(n);
}
int nth(T* &n,int nr){
  if(n==nil)
    return -1;
  if(nr==0)
    return n->key;
  int leftval = n->left->nr;
  if(nr-leftval == 1)
    return n->key;
  if(leftval >= nr)
    return nth(n->left,nr);
  return nth(n->right,nr-leftval-1);
}
void balance(T* &n) {
  if (n->left->priority > n->priority)
    rotleft(n);
```

```
  else if (n->right->priority > n->priority)
    rotright(n);
  update(n->right);
  update(n->left);
  update(n);
}
void insert(T* &n, int key, int priority) {
  if (n == nil) {
    n = new T(key, priority, nil, nil);
    n->nr=1;
    return;
  }
  (n->nr)++;
  if (key <= n->key)
    insert(n->left, key, priority);
  else if (key > n->key)
    insert(n->right, key, priority);
  balance(n);
}
```

2 Maximum Flows

2.1 Max flow

```
int flux[1010][1010];
int c[1010][1010];
int tata[1010];
int viz[1010],flow;
int coad[1015];
int Q,x,y,z,N,act,M,flow_min;

vector<int> g[1010];

int BF() {
  for(int i=1;i<=N;++i) {
    viz[i]=0;
  }
  coad[0]=1;
  int st=0,dr=1;
  viz[1]=1;
  while(st<dr) {
    act=coad[st];
    if(act!=N)
      for(int i=0;i<g[act].size();++i) {
        Q = g[act][i];
        if(c[act][Q] == flux[act][Q] || viz[Q])
          continue;
        viz[Q]=1;
        coad[dr++]=Q;
        tata[Q]=act;
      }
    ++st;
```

```
  }
  return viz[N];
}
int main() {
  for(flow=0; BF();) {
    for(int i=0;i<g[N].size();++i) {
      act=g[N][i];
      if(flux[act][N]==c[act][N] ||!viz[act])
        continue;
      tata[N]=act;

      flow_min=10101000;
      for(int nod=N;nod!=1;nod=tata[nod])
        flow_min=min(flow_min,c[tata[nod]][nod]-flux[tata[nod]][nod]);
      if(flow_min==0)
        continue;
      for(int nod=N;nod!=1;nod=tata[nod]) {
        flux[tata[nod]][nod]+=flow_min;
        flux[nod][tata[nod]]-=flow_min;
      }
      flow+=flow_min;
    }
  }
}
```

2.2 Min cost max flow

```
int N, M, S, D;
vector<int> G[MAXN];

int cap[MAXN][MAXN];
int cost[MAXN][MAXN];
int flow[MAXN][MAXN];

int d[MAXN];
int prev[MAXN];
bool found;

int bellman_ford() {
  vector<bool> inqueue(N + 1, false);
  queue<int> q;
  q.push(S);
  inqueue[S] = true;
  for (int i = 1; i <= N; ++i) {
    prev[i] = -1;
    d[i] = INF;
  }
  d[S] = 0;

  while (!q.empty()) {
    int node = q.front();
    q.pop();
    inqueue[node] = false;
```

```
    vector<int>::iterator it;
    for (it = G[node].begin(); it != G[node].end(); ++it) {
      if (cap[node][*it] - flow[node][*it] <= 0)
        continue;
      if (cost[node][*it] + d[node] < d[*it]) {
        prev[*it] = node;
        d[*it] = cost[node][*it] + d[node];
        if (!inqueue[*it]) {
          q.push(*it);
          inqueue[*it] = true;
        }
      }
    }
  }

  if (d[D] < INF / 2) {
    found = true;
    int fmin = INF;
    for (int node = D; node != S; node = prev[node])
      fmin = min(fmin, cap[prev[node]][node] - flow[prev[node]][node]);
    for (int node = D; node != S; node = prev[node]) {
      flow[prev[node]][node] += fmin;
      flow[node][prev[node]] -= fmin;
    }
    return d[D] * fmin;
  }
  return 0;
}
long long mfmc() {
  long long result = 0;
  found = true;
  while (found) {
    found = false;
    result += bellman_ford();
  }
  return result;
}
```

3 Graphs

3.1 Bellman-Ford

```
int bellmanford() {
  for (int i = 1; i <= n; ++i)
      dist[i] = INF;
  dist[1] = 0;

  queue<int> q;
  q.push(1);
  inqueue[1] = true;
  cnt[1] = 1;
  while (!q.empty()) {
```

```
    int node = q.front();
    q.pop();
    inqueue[node] = false;
    for (int i = 0; i < G[node].size(); ++i) {
      int next = G[node][i];
      if (dist[node] + C[node][i] < dist[next]) {
        dist[next] = dist[node] + C[node][i];
        if (!inqueue[next]) {
          if (cnt[next] > n)
            return -1;
          q.push(next);
          inqueue[next] = true;
          cnt[next]++;
        }
      }
    }
  }
  return 1;
}
```

3.2 Euler cycle

```
list<int> G[MAXN];
vector<int> sol;
int deg[MAXN];
bool vis[MAXN];
void dfs(int node) {
  vis[node] = true;
  list<int>::iterator it;
  for (it = G[node].begin(); it != G[node].end(); ++it) {
    if (vis[*it])
      continue;
    dfs(*it);
  }
}
void rem_edge(int v, int w) {
  G[v].pop_front();
  list<int>::iterator it;
  for (it = G[w].begin(); it != G[w].end(); ++it)
    if (*it == v) {
      G[w].erase(it);
      break;
    }
}

stack<int> st;
void euler(int v) {
  while (!G[v].empty()) {
    int w = *G[v].begin();
    rem_edge(v, w);
    st.push(v);
    v = w;
  }
```

```
}
int main() {
  st.push(1);
  while (!st.empty()) {
    int v = st.top();
    st.pop();
    euler(v);
    sol.push_back(v);
  }
  reverse(sol.begin(), sol.end());
}
```

3.3 Maximum matching

```
int N,M,K;
int v[25000],x,p=0;
char car;
vector <int> g[25000];
int l[25000],r[25000],u[25000],was[25000],S;
int cupj(int q) {
  if(was[q])
    return 0;
  was[q]=1;
  for(int i=0;i<g[q].size();++i) {
    if(!r[g[q][i]]) {
      l[q]=g[q][i];
      r[g[q][i]]=q;
      return 1;
    }
  }
  for(int i=0;i<g[q].size();++i) {
    if(cupj(r[g[q][i]])) {
      l[q]=g[q][i];
      r[g[q][i]]=q;
      return 1;
    }
  }
  return 0;
}
```

3.4 Hamiltonian path

```
const int inf = 1000000000;
int N,M,x,y,z,Sol,b[262150][22],c[22][22];
vector<int> a[22];

int best(int conf, int last) {
  if(b[conf][last]>=0)
    return b[conf][last];
  b[conf][last]=inf;
  for(int i=0;i<a[last].size();++i)
    if(conf & (1<<a[last][i])) {
```

```
      if(a[last][i]==0 && conf!=(1<<last)+1)
        continue;
      if(b[conf][last] > best(conf^(1<<last),a[last][i])+c[a[last][i]][last])
        b[conf][last] = best(conf^(1<<last),a[last][i])+c[a[last][i]][last];
    }
  return b[conf][last];
}
```

3.5 Heavy path decomposition

```
int N, M, nL;
int v[MAXN], fol[MAXN], niv[MAXN], w[MAXN], l[MAXN];
int aint[4*MAXN];
int lTata[MAXN], lNiv[MAXN], lDim[MAXN], lPoz[MAXN];
vector<int> G[MAXN], P[MAXN];
pair<int, pair<int, int> > op[MAXN];

void df(int nod) {
  fol[nod] = 1;
  w[nod] = 1;
  int hN = -1, frunza = 1;

  for(vector<int> :: iterator it = G[nod].begin(); it != G[nod].end(); ++it) {
    if(fol[*it])
      continue;
    frunza = 0;
    niv[*it] = niv[nod] + 1;
    df(*it);
    w[nod] += w[*it];
    if(hN == -1)
      hN = *it;
    else if(w[hN] < w[*it])
      hN = *it;
  }
  if(frunza) {
    l[nod] = ++nL;
    lDim[nL]=1;
    P[nL].push_back(nod);
    return;
  }
  l[nod] = l[hN];
  ++lDim[l[nod]];
  P[l[nod]].push_back(nod);

  for(vector<int> :: iterator it = G[nod].begin(); it != G[nod].end(); ++it) {
    if((*it) == hN || niv[*it] < niv[nod])
      continue;

    lTata[l[*it]] = nod;
    lNiv[l[*it]] = niv[nod];
  }
}
void build(int nod, int left, int right, int decalaj, int lant) {
```

```
  if(left == right) {
    aint[nod + decalaj] = v[ P[lant][left - 1] ];
    return;
  }
  int med = (left + right) / 2;
  build(nod * 2, left, med, decalaj, lant);
  build(nod * 2 + 1, med+1, right, decalaj, lant);
  aint[nod + decalaj] = max(aint[nod * 2 + decalaj], aint[nod * 2 + 1 + decalaj]);
}
void make_paths() {
  niv[1] = 1;
  df(1);
  for(int i = 1; i <= nL; ++i) {
    reverse(P[i].begin(), P[i].end());
    if(i > 1)
      lPoz[i] = lPoz[i-1] + lDim[i-1] * 4;
    build(1, 1, lDim[i], lPoz[i], i);
  }
}
void update(int nod, int left, int right, int poz, int val, int decalaj) {
  if(left == right) {
    aint[nod + decalaj] = val;
    return;
  }
  int med = (left + right) / 2;
  if(poz<=med)
    update(nod * 2, left, med, poz, val, decalaj);
  else
    update(nod * 2 + 1, med+1, right, poz, val, decalaj);
  aint[nod + decalaj] = max(aint[nod * 2 + decalaj], aint[nod * 2 + 1 + decalaj]);
}
int query(int nod, int left, int right, int qleft, int qright, int decalaj) {
  if(qleft <= left && right <= qright)
    return aint[nod + decalaj];
  int med = (left + right) / 2, rez = 0;
  if(qleft <= med)
    rez = max(rez, query(nod * 2, left, med, qleft, qright, decalaj) );
  if(med < qright)
    rez = max(rez, query(nod * 2 + 1, med + 1, right, qleft, qright, decalaj) );
  return rez;
}
void solve() {
  int t, x, y, sol = 0;
  for(int i = 1; i <= M; ++i) {
    t = op[i].first; x = op[i].second.first, y = op[i].second.second;
    if(t==0) {
      update(1, 1, lDim[l[x]], niv[x] - lNiv[l[x]], y, lPoz[l[x]]);
    } else {
      sol = 0;
      while(1) {
        if(l[x] == l[y]) {
          if(niv[x] > niv[y])
            swap(x, y);
          sol = max(sol, query(1, 1, lDim[l[x]], niv[x] - lNiv[l[x]], niv[y] - lNiv[l[x]], lPoz[l[x]]))
```

```
          break;
        }
        if(lNiv[l[x]] < lNiv[l[y]])
          swap(x, y);
        sol = max(sol, query(1, 1, lDim[l[x]], 1, niv[x] - lNiv[l[x]], lPoz[l[x]]));
        x = lTata[l[x]];
      }
    }
  }
}
```

3.6 Lowest common ancestor

```
int K, N, M, x, y, L[200010], H[200010], Lg[200010], First[100010], Rmq[20][400010];
vector<int> G[100010];

void dfs(int nod, int lev) {
  H[++K] = nod;
  L[K] = lev;
  First[nod] = K;
  int z = G[nod].size();
  for(int i=0;i<z;++i) {
    dfs(G[nod][i],lev+1);
    H[++K] = nod;
    L[K] = lev;
  }
}
void rmq() {
  for(int i=2;i<=K;++i)
    Lg[i] = Lg[i/2]+1;
  for(int i=1;i<=K;++i)
    Rmq[0][i]=i;
  for(int i=1;(1<<i) < K; ++i)
    for(int j=1;j<=K-(1<<i);++j) {
      int l = 1<< (i-1);
      Rmq[i][j] = Rmq[i-1][j];
      if(L[Rmq[i-1][j + l]] < L[Rmq[i][j]])
        Rmq[i][j] = Rmq[i-1][j + l];
    }
}
int lca(int x, int y) {
  int a = First[x], b = First[y];
  if(a>b) {
    int c=a;
    a=b;
    b=c;
  }
  int diff = b - a + 1;
  int l = Lg[diff];
  int sol = Rmq[l][a];
  int sh = diff - (1 << l);
  if(L[sol] > L[Rmq[l][a + sh]])
    sol = Rmq[l][a + sh];
```

```
  return H[sol];
}
```

4 Mathematics

4.1 Number theoretic algorithms

```
typedef vector<int> VI;
typedef pair<int,int> PII;

int mod(int a, int b) {
  return ((a%b)+b)%b;
}
int gcd(int a, int b) {
  int tmp;
  while(b){a%=b; tmp=a; a=b; b=tmp;}
  return a;
}
int lcm(int a, int b) {
  return a/gcd(a,b)*b;
}
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a/b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  }
  return a;
}
VI modular_linear_equation_solver(int a, int b, int n) {
  int x, y;
  VI solutions;
  int d = extended_euclid(a, n, x, y);
  if (!(b%d)) {
    x = mod (x*(b/d), n);
    for (int i = 0; i < d; i++)
      solutions.push_back(mod(x + i*(n/d), n));
  }
  return solutions;
}
int mod_inverse(int a, int n) {
  int x, y;
  int d = extended_euclid(a, n, x, y);
  if (d > 1) return -1;
  return mod(x,n);
}
PII chinese_remainder_theorem(int x, int a, int y, int b) {
  int s, t;
  int d = extended_euclid(x, y, s, t);
```

```
  if (a%d != b%d) return make_pair(0, -1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}
PII chinese_remainder_theorem(const VI &x, const VI &a) {
  PII ret = make_pair(a[0], x[0]);
  for (int i = 1; i < x.size(); i++) {
    ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
    if (ret.second == -1) break;
  }
  return ret;
}
void linear_diophantine(int a, int b, int c, int &x, int &y) {
  int d = gcd(a,b);
  if (c%d) {
    x = y = -1;
  } else {
    x = c/d * mod_inverse(a/d, b/d);
    y = (c-a*x)/b;
  }
}
```

4.2 Gaussian elimination

```
int i,j,k;
int N,M;
double A[303][303];
double X[303];

int main() {
  for(int i=1;i<=N;++i) {
    for(int j=1;j<=M+1;++j) {
      scanf("%lf",&A[i][j]);
    }
  }
  i=1,j=1;
  while(i<=N && j<=M) {
    for(k=i;k<=N;++k)
      if(A[k][j]<-EPS||A[k][j]>EPS)
        break;
    if(k==N+1) {
      ++j;
      continue;
    }
    if(k!=i) {
      for(int q=1;q<=M+1;++q) {
        double aux = A[i][q];
        A[i][q]= A[k][q];
        A[k][q]= aux;
      }
    }
    for(int q=j+1;q<=M+1;++q) {
      A[i][q]=A[i][q]/A[i][j];
    }
```

```
    A[i][j]=1;
    for(int u=i+1;u<=N;++u) {
      for(int q=j+1;q<=M+1;++q) {
        A[u][q]-=A[u][j]*A[i][q];
      }
      A[u][j]=0;
    }
    ++i;++j;
  }
}
```

5 Strings

5.1 Knuth-Morris-Pratt

```
vector<int> prefix(const string& str) {
  vector<int> pi(str.size(), 0);
  int k = 0;
  for (int i = 1; i < str.size(); ++i) {
    while (k > 0 && str[i] != str[k])
      k = pi[k - 1];
    if (str[i] == str[k])
      ++k;
    pi[i] = k;
  }
  return pi;
}
vector<int> match(const string& str, const string& patt) {
  vector<int> matches;
  vector<int> pi = prefix(patt);
  int k = 0;
  for (int i = 0; i < str.size(); ++i) {
    while (k > 0 && str[i] != patt[k])
      k = pi[k - 1];
    if (str[i] == patt[k])
      ++k;
    if (k == patt.size())
      matches.push_back(i - patt.size() + 1);
  }
  return matches;
}
```

5.2 Rabin-Karp

```
char A[MAXN], B[MAXN];
int NA, NB;
int hashA1, hashA2, P1, P2;
char match[MAXN];

int main() {
  P1 = P2 = 1;
```

```
  hashA1 = hashA2 = 0;
  for (int i = 0; i < NA; i++) {
    hashA1 = (hashA1 * P + A[i]) % MOD1;
    hashA2 = (hashA2 * P + A[i]) % MOD2;
    if (i != 0)
      P1 = (P1 * P) % MOD1,
         P2 = (P2 * P) % MOD2;
  }
  int hash1 = 0, hash2 = 0;
  for (int i = 0; i < NA; i++)
    hash1 = (hash1 * P + B[i]) % MOD1,
          hash2 = (hash2 * P + B[i]) % MOD2;
  int Nr = 0;
  if (hash1 == hashA1 && hash2 == hashA2)
    match[0] = 1, Nr++;
  for (int i = NA; i < NB; i++) {
    hash1 = ((hash1 - (B[i - NA] * P1) % MOD1 + MOD1) * P + B[i]) % MOD1;
    hash2 = ((hash2 - (B[i - NA] * P2) % MOD2 + MOD2) * P + B[i]) % MOD2;
    if (hash1 == hashA1 && hash2 == hashA2)
      match[ i - NA + 1 ] = 1, Nr++;
  }
}
```

5.3 Longest palindromic substring

```
int N;
char s[MAXN];
int dp[2][MAXN];
long long res = 0;

int explode(int l, int r) {
  int len = 0;
  for (; l >= 0 && r < N && s[l] == s[r]; --l, ++r, ++len);
  return len;
}
void odd_center() {
  int last = -1, right = -1;
  for (int i = 0; i < N; ++i) {
    if (right >= i)
      dp[0][i] = min(dp[0][2 * last - i], right - i);

    int l = i - dp[0][i];
    int r = i + dp[0][i];
    dp[0][i] += explode(l, r);
    if (i + dp[0][i] > right) {
      last = i;
      right = i + dp[0][i];
    }
    res += (long long) dp[0][i];
  }
}
void even_center() {
  int last = -1, right = -1;
```

```
  for (int i = 0; i < N; ++i) {
    if (s[i] != s[i + 1])
      continue;
    if (right > i)
      dp[1][i] = min(dp[1][2 * last - i], right - i - 1);
    int l = i - dp[1][i];
    int r = i + dp[1][i] + 1;
    dp[1][i] += explode(l, r);
    if (i + dp[1][i] + 1 > right) {
      last = i;
      right = i + dp[1][i] + 1;
    }
    res += (long long) dp[1][i];
  }
}
int main() {
  odd_center();
  even_center();
}
```

5.4 Trie

```
struct Trie {
  int cnt, nrsons;
  Trie *son[26];

  Trie() {
    cnt = nrsons = 0;
    memset(son, 0, sizeof(son));
  }
};

Trie *T = new Trie;
void ins(Trie *node, char *s) {
  if (*s == '\0') {
    node->cnt++;
    return;
  }
  if (node->son[ch] == 0) {
    node->son[ch] = new Trie;
    node->nrsons++;
  }
  ins(node->son[ch], s + 1);
}

int del(Trie *node, char *s) {
  if (*s == '\0')
    node->cnt--;
  else if (del(node->son[ch], s + 1)) {
    node->son[ch] = 0;
    node->nrsons--;
  }
  if (node->cnt == 0 && node->nrsons == 0 && node != T) {
```

```
    delete node;
    return 1;
  }
}
int freq(Trie *node, char *s) {
  if (*s == '\0')
    return node->cnt;
  if (node->son[ch])
    return freq(node->son[ch], s + 1);
  return 0;
}
int pref(Trie *node, char *s, int k) {
  if (*s == '\0' || node->son[ch] == 0)
    return k;
  return pref(node->son[ch], s + 1, k + 1);
}
```

6 Geometry

6.1 Convex hull

```
typedef pair<double, double> point;

int n;
point v[MAXN];
int head;
point stack[MAXN];

inline double cross_prod(const point& A, const point& B, const point& C) {
  return (B.x - A.x) * (C.y - A.y) - (B.y - A.y) * (C.x - A.x);
}
inline bool comp(const point& A, const point& B) {
  return cross_prod(v[1], A, B) < 0;
}
void sort_points() {
  int pos = 1;
  for (int i = 2; i <= n; ++i)
    if (v[i] < v[pos])
      pos = i;
  swap(v[1], v[pos]);
  sort(v + 2, v + n + 1, comp);
}
void convex_hull() {
  sort_points();

  stack[1] = v[1];
  stack[2] = v[2];
  head = 2;
  for (int i = 3; i <= n; ++i) {
    while (head >= 2 && cross_prod(stack[head - 1], stack[head], v[i]) > 0)
      --head;
    stack[++head] = v[i];
```

```
  }
}
```

6.2 Miscellaneous geometry

```
struct point {
  double x, y;
  point() {}
  point(double x_, double y_): x(x_), y(y_) {}
  point(const point& p): x(p.x), y(p.y) {}

  point operator+(const point& p) const { return point(x + p.x, y + p.y); }
  point operator-(const point& p) const { return point(x - p.x, y - p.y); }
  point operator*(double c) const { return point(x * c, y * c); }
  point operator/(double c) const { return point(x / c, y / c); }
};
ostream &operator<<(ostream &os, const point& p) {
  os << "(" << p.x << "," << p.y << ")";
}

double dot(point p, point q) { return p.x * q.x + p.y * q.y; }
double dist2(point p, point q) { return dot(p - q, p - q); }
double dist(point p, point q) { return sqrt(dist2(p, q)); }
double cross(point p, point q) { return p.x * q.y - p.y * q.x; }

double is_left(point a, point b, point c) {
  return cross(b - a, c - a);
}

point rotate_cw_90(point p) { return point(p.y, -p.x); }
point rotate_ccw_90(point p) { return point(-p.y, p.x); }
point rotate(point p, double a) {
  return point(p.x * cos(a) - p.y * sin(a), p.x * sin(a) + p.y * cos(a));
}

point project_point_line(point a, point b, point c) {
  return a + (b - a) * dot(c - a, b - a) / dot(b - a, b - a);
}
point project_point_segment(point a, point b, point c) {
  double d = dot(b - a, b - a);
  if (abs(d) < EPS) return a;
  d = dot(c - a, b - a) / d;
  if (d < 0) return a;
  if (d > 1) return b;
  return a + (b - a) * d;
}

double distance_point_segment(point a, point b, point c) {
  return sqrt(dist2(c, project_point_segment(a, b, c)));
}
double distance_point_plane(double x, double y, double z,
    double a, double b, double c, double d) {
  return abs(a * x + b * y + c * z - d) / sqrt(a * a + b * b + c * c);
```

```
}

bool lines_parallel(point a, point b, point c, point d) {
  return abs(cross(b - a, c - d)) < EPS;
}
bool lines_collinear(point a, point b, point c, point d) {
  return lines_parallel(a, b, c, d) &&
    abs(cross(a - b, a - c)) < EPS &&
    abs(cross(c - d, c - a)) < EPS;
}

bool segments_intersect(point a, point b, point c, point d) {
  if (lines_collinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
        dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c - a, c - b) > 0 && dot(d - a, d - b) > 0 &&
        dot(c - b, d - b) > 0) return false;
    return true;
  }
  if (cross(d - a, b - a) * cross(c - a, b - a) > 0) return false;
  if (cross(a - c, d - c) * cross(b - c, d - c) > 0) return false;
  return true;
}
point compute_line_intersection(point a, point b, point c, point d) {
  b = b - a; d = c - d; c = c - a;
  return a + b * cross(c, d) / cross(b, d);
}
point compute_circle_center(point a, point b, point c) {
  b = (a + b) / 2;
  c = (a + c) / 2;
  return compute_line_intersection(b, b + rotate_cw_90(a - b),
      c, c + rotate_cw_90(a - c));
}

bool point_in_poly(point p, const vector<point>& v) {
  int wn = 0;
  int n = static_cast<int>(v.size());
  for (int i = 0; i < n; ++i) {
    int j = (i + 1) % n;
    if (v[i].y <= p.y) {
      if (v[j].y > p.y && is_left(v[i], v[j], p) > 0)
        ++wn;
    } else {
      if (v[j].y <= p.y && is_left(v[i], v[j], p) < 0)
        --wn;
    }
  }
  return wn != 0;
}
bool point_on_polygon(point p, const vector<point>& v) {
  int n = static_cast<int>(v.size());
  for (int i = 0; i < n; ++i) {
    int j = (i + 1) % n;
    if (dist2(project_point_segment(v[i], v[j], p), p) < EPS)
```

```
      return true;
  }
  return false;
}

double signed_area(const vector<point>& v) {
  int n = static_cast<int>(v.size());
  double area = 0;
  for (int i = 0; i < n; ++i) {
    int j = (i + 1) % n;
    area += v[i].x * v[j].y - v[j].x * v[i].y;
  }
  return area / 2.0;
}
double area(const vector<point>& v) {
  return abs(signed_area(v));
}
point centroid(const vector<point>& v) {
  int n = static_cast<int>(v.size());
  point c(0, 0);
  double scale = 6.0 * signed_area(v);
  for (int i = 0; i < n; ++i) {
    int j = (i + 1) % n;
    c = c + (v[i] + v[j]) * (v[i].x * v[j].y - v[j].x * v[i].y);
  }
  return c / scale;
}
bool is_simple(const vector<point>& v) {
  int n = static_cast<int>(v.size());
  for (int i = 0; i < n; ++i) {
    for (int k = i + 1; k < n; ++k) {
      int j = (i + 1) % n;
      int l = (k + 1) % n;
      if (i == l || j == k) continue;
      if (segments_intersect(v[i], v[j], v[k], v[l]))
        return false;
    }
  }
  return true;
}
```

7 Other

7.1 2SAT

```
vector<int> g[202020],gx[202020],last,stackx,viz,iss,low,aux,tare,gr_in;
vector<vector<int> > comp;
int N,index=1,k,Nn,M,x,y;
int rez[201010];
int notn(int x){
  if(x<=N){
    return x + N;
```

```
  }
  return x-N;
}
void df(int x){
  viz[x] = index;
  low[x] = index;
  stackx[++k] = x;
  iss[x] = 1;
  ++index;
  for(int i=0;i<g[x].size();++i){
    if(viz[g[x][i]] == 0){
      df(g[x][i]);
      low[x] = min(low[x],low[g[x][i]]);
    } else{
      if(iss[g[x][i]]){
        low[x] = min(low[x],low[g[x][i]]);
      }
    }
  }
  if(low[x] == viz[x]){
    aux.clear();
    do{
      aux.pb(stackx[k]);
      iss[stackx[k]] = 0;
      --k;
    }while(stackx[k+1] != x);
    comp.pb(aux);
  }
}
void init(){
  index = 1;
  stackx.resize(Nn+10);
  viz.resize(Nn+10);
  iss.resize(Nn+10);
  low.resize(Nn+10);
  tare.resize(Nn+10);
  last.resize(Nn+10);
  gr_in.resize(Nn+10);
}
void make_ctc(){
  Nn=2*N;
  init();
  for(int i=1;i<=Nn;++i){
    if(viz[i]==0){
      df(i);
    }
  }
  for(int i=0;i<comp.size();++i){
    for(int j=0;j<comp[i].size();++j){
      tare[comp[i][j]] = i+1;
    }
  }
}
void solve(){
```

```
  for(int i=1;i<=N;++i){
    if(tare[i] == tare[notn(i)]){
      printf("-1\n");
      return;
    }
  }
  int nod;
  for(int i=0;i<comp.size();++i){
    for(int j=0;j<comp[i].size();++j){
      nod = comp[i][j];
      for(int k=0;k<g[nod].size();++k){
        int compv = tare[g[nod][k]];
        if ( last[compv] != i+1 && compv != i+1 ){
          gx[i+1].pb(compv);
          ++gr_in[compv];
          last[compv]=i+1;
        }
      }
    }
  }
  queue<int> Q;
  for(int i=1;i<=comp.size();++i){
    if(gr_in[i]==0){
      Q.push(i);
    }
  }
  int nr = 0;
  while(!Q.empty()){
    ++nr;
    nod = Q.front(); Q.pop();
    for (int i=0;i<gx[nod].size();++i){
      int nodv = gx[nod][i];
      --gr_in[nodv];
      if (gr_in[nodv]==0)
        Q.push(nodv);
    }
    if ( last[nod] == -1 )  continue ;
    for (int i=0;i<comp[nod-1].size();++i){
      rez[comp[nod-1][i]] = 0;
      rez[notn(comp[nod-1][i])] = 1;
    }
    nod = tare[notn(comp[nod-1][0])];
    last[nod] = -1;
  }
}
```

7.2 KD-tree

```
const ntype sentry = numeric_limits<ntype>::max();
struct point {
  ntype x, y;
  point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};
```

```
bool operator==(const point &a, const point &b) {
  return a.x == b.x && a.y == b.y;
}
bool on_x(const point &a, const point &b) {
  return a.x < b.x;
}
bool on_y(const point &a, const point &b) {
  return a.y < b.y;
}
ntype pdist2(const point &a, const point &b) {
  ntype dx = a.x-b.x, dy = a.y-b.y;
  return dx*dx + dy*dy;
}
struct bbox {
  ntype x0, x1, y0, y1;
  bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}
  void compute(const vector<point> &v) {
    for (int i = 0; i < v.size(); ++i) {
      x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
      y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
    }
  }
  ntype distance(const point &p) {
    if (p.x < x0) {
      if (p.y < y0)       return pdist2(point(x0, y0), p);
      else if (p.y > y1)  return pdist2(point(x0, y1), p);
      else                return pdist2(point(x0, p.y), p);
    } else if (p.x > x1) {
      if (p.y < y0)       return pdist2(point(x1, y0), p);
      else if (p.y > y1)  return pdist2(point(x1, y1), p);
      else                return pdist2(point(x1, p.y), p);
    } else {
      if (p.y < y0)       return pdist2(point(p.x, y0), p);
      else if (p.y > y1)  return pdist2(point(p.x, y1), p);
      else                return 0;
    }
  }
};
struct kdnode {
  bool leaf;      // true if this is a leaf node (has one point)
  point pt;       // the single point of this is a leaf
  bbox bound;     // bounding box for set of points in children

  kdnode *first, *second; // two children of this kd-node

  kdnode() : leaf(false), first(0), second(0) {}
  ~kdnode() { if (first) delete first; if (second) delete second; }

  ntype intersect(const point &p) {
    return bound.distance(p);
  }
  void construct(vector<point> &vp) {
    bound.compute(vp);
    if (vp.size() == 1) {
```

```
      leaf = true;
      pt = vp[0];
    } else {
      if (bound.x1-bound.x0 >= bound.y1-bound.y0)
        sort(vp.begin(), vp.end(), on_x);
      else
        sort(vp.begin(), vp.end(), on_y);

      int half = vp.size()/2;
      vector<point> vl(vp.begin(), vp.begin()+half);
      vector<point> vr(vp.begin()+half, vp.end());
      first = new kdnode();   first->construct(vl);
      second = new kdnode();  second->construct(vr);
    }
  }
};

struct kdtree {
  kdnode *root;
  kdtree(const vector<point> &vp) {
    vector<point> v(vp.begin(), vp.end());
    root = new kdnode();
    root->construct(v);
  }
  ~kdtree() { delete root; }
  ntype search(kdnode *node, const point &p) {
    if (node->leaf) {
      return pdist2(p, node->pt);
    }
    ntype bfirst = node->first->intersect(p);
    ntype bsecond = node->second->intersect(p);

    if (bfirst < bsecond) {
      ntype best = search(node->first, p);
      if (bsecond < best)
        best = min(best, search(node->second, p));
      return best;
    } else {
      ntype best = search(node->second, p);
      if (bfirst < best)
        best = min(best, search(node->first, p));
      return best;
    }
  }
  ntype nearest(const point &p) {
    return search(root, p);
  }
};
```