Contents

~/.vimrc

```
set nocp mouse=a bs=2
set nu hls ru ls=2 cul sb spr
set ai ts=4 sw=4 sts=4 et sta
set nobk noswf ar
set noea noeb novb t_vb=

sy on
hi CursorLine ctermbg=233
hi LineNr     ctermfg=236
hi Search     ctermbg=52
hi MatchParen ctermbg=234

 map o oo<bs>
 map O OO<bs>
imap <cr> <cr>a<bs>

vmap > >gv
vmap < <gv
 map <f9>      :w<cr>:!g++ -lm -std=c++11 -o %:r -Wall %<cr>
imap <f9> <esc>:w<cr>:!g++ -lm -std=c++11 -o %:r -Wall %<cr>
 map <f1> <esc>
imap <f1> <esc>

map \\ :silent! /asdfadfeifj<cr>
map \x mzHmx:silent! :%s/[ \t][ \t]*$//g<CR>`xzt`z\\
```

Memo

- Binary Search
- stack, queue, deque, DP
- Split N to two N/2
- Find the center of gravity
- continue?break?
- divide by zero, special case
- initialize
-
-
-
-

KMP

```
/*********************************************/
/* [usage]: kmp(str1, str2)                  */
/* [check]: return true if str1 contains str2 */
/*********************************************/
#include<string.h>
int kmp_fail[maxn];
bool kmp(char* s, char* t){
  int s_len = strlen(s), t_len = strlen(t);
  int tmp, s_pos, t_pos;
  kmp_fail[0] = -1;
  for(int i=1; i<t_len; i++){
    tmp = kmp_fail[i-1];
    while( t[i] != t[tmp+1] && tmp!=-1) tmp = kmp_fail[tmp];
    if(t[i] == t[tmp+1]) kmp_fail[i] = tmp+1;
    else kmp_fail[i] = -1;
  }
  s_pos = t_pos = 0;
  while(s_pos < s_len){
    if(s[s_pos] == t[t_pos]){
      s_pos++; t_pos++;
      if(t_pos == t_len) return true;
    }
    else if(t_pos == 0) s_pos++;
    else t_pos = kmp_fail[t_pos-1] + 1;
  }
  return false;
}
```

Gaussian Elimination

```

lnt Madd(lnt a, lnt b, lnt mod) {
  return (a + b + mod) % mod;
}
lnt Mmul(lnt a, lnt b, lnt mod) {
  return (a * b) % mod;
}

int Pow(int a, int b, int mod) {
  lnt r = 1, t = a;
  for( ; b != 0;
      b >>= 1, t = Mmul(t, t, mod))
    if((b&1) != 0) r=Mmul(r, t, mod);
  return (int)r;
}

int Inverse(int a, int mod) {
  return Pow(a, mod - 2, mod);
}

void RowSwi(int coe[][MAXN+1],
            int n, int mod,
            int a, int b) {
  for(int i = 1; i <= n+1; ++i)
    swap(coe[a][i], coe[b][i]);
}

void RowMul(int coe[][MAXN + 1],
            int n, int mod,
            int a, int c) {
  for(int i=1; i<=n + 1; ++i)
    coe[a][i]=Mmul(coe[a][i],c,
                   mod);
}

void RowAdd(int coe[][MAXN + 1],
            int n, int mod,
            int a,int b,int c) {
  for(int i=1; i <= n+1; ++i)
    coe[b][i]=Madd(coe[b][i],
                   Mmul(coe[a][i],
                        c,
                        mod),
                   mod);
}

void GaussElimination(
    int n, int mod,
    int coe[][MAXN + 1]) {
  for(int j=1, k=1; j<=n; ++j){
    for(i = k; i <= n; ++i)
      if(coe[i][j] != 0) break;
    if(i > n) continue;

    RowSwi(coe, n, mod, i, k);
    RowMul(coe, n, mod, k,
           Inverse(coe[k][j], mod));
    for(i = 1; i <= n; ++i) {
      if(i == k) continue;
      RowAdd(coe, n, mod, k, i,
             Madd(mod,
                  -coe[i][j],
                  mod));
    }
    ++k;
  }
}
```

Tree DC

```cpp
#include <algorithm>
#include <vector>

using std::max;
using std::vector;

int n;
std::vector< int > edg[SIZE_N];

int exi[SIZE_N];
int sons[SIZE_N], mson[SIZE_N];

int find_G(int now, int fa, int siz) {
  if (mson[now] * 2 <= siz &&
      (siz - sons[now]) * 2 <= siz)
    return now;
  for(int i=0; i<edg[now].size(); ++i) {
    int ne = edg[now][i];
    if(exi[ne] == 0 || ne == fa)
      continue;
    int t = find_G(ne, now, siz);
    if(t != -1) return t;
  }
  return -1;
}

void calc_son(int now, int fa) {
  sons[now] = 1, mson[now] = 0;
  for(int i=0; i<edg[now].size(); ++i) {
    int ne = edg[now][i];
    if(exi[ne] == 0 || ne == fa)
      continue;
    calc_son(ne, now);
    sons[now] += sons[ne];
    mson[now] =max(mson[now], sons[ne]);
  }
}

void TDC(int now) {
  calc_son(now, 0);
  now = find_G(now, 0, sons[now]);
  exi[now] = 0;
  for(int i=0; i<edg[now].size(); ++i) {
    int ne = edg[now][i];
    if(exi[ne] == 0) continue;
    TDC(ne);
  }
  /* do something */
  exi[now] = 1;
}
```

Tarjar On Graph

```cpp
/*****************************************/
/*  Tarjan's algorithm                   */
/*  Find articulation points, bridges,   */
/*       BCC, SCC, and solve 2-SAT       */
/*****************************************/

#include <stdio.h>
#include <stack>
#include <vector>
#include <utility>

#define min(x, y) ((x) < (y) ? (x) : (y))
#define V 1000007
#define E 1000007

using namespace std;

typedef pair<int, int> pii;

vector<pii> edge[V];    // adjacent list
// first: destination, second: edge index
stack<int> stk;         // stack for BCC/SCC
int time, new_id;       // global counters
int stamp[V];           // timestamp (use global "time")
int low[V];             // low function
bool instack[V];        // only needed in SCC
bool is_articulation[V];// articulation point result
bool is_bridge[E];      // bridge result
int contract[V];        // contracted point id (use global "new_id")
vector<int> group[V];   // SCC groups

vector<pair<int, int> > edge_arr;

void add_edge(int st, int ed, int edge_idx = 0){
  edge[st].push_back(make_pair(ed, edge_idx));
}

void dfs(int now, int par){
  stamp[now] = low[now] = time++;
```

```
  stk.push(now);
  instack[now] = true;
  bool flg = false;
  int child_cnt = 0;
  for(unsigned i=0, n=edge[now].size(); i<n; ++i){
    int nxt = edge[now][i].first;
    int edge_idx = edge[now][i].second;
    // [Important] don't need this "if" in SCC/2-SAT
    if(nxt != par){
      // tree edge
      if(stamp[nxt] == -1){
        child_cnt++;
        dfs(nxt, now);
        low[now] = min(low[now], low[nxt]);
        if(low[nxt] >= stamp[now]) flg = true;
        if(low[nxt] > stamp[now]) is_bridge[edge_idx] = true;
      }
      // back edge in BCC, always true (can change to "else")
      // back / forward / cross edge in SCC
      else if(instack[nxt])
        low[now] = min(low[now], stamp[nxt]);
    }
  }
  if( (now == par && child_cnt >= 2) ||
      (now != par && flg)) is_articulation[now] = true;
  if(low[now] == stamp[now]){
    int v;
    do{
      v = stk.top(); stk.pop();
      instack[v] = false;
      contract[v] = new_id;
      group[new_id].push_back(v);
    } while(v != now);
    new_id++;
  }
}

// if don't need bridges, use tarjan(v)
void tarjan(int v, int e = 0){
  time = 0;
  new_id = 0;
  while(!stk.empty()) stk.pop();
  for(int i=0; i<v; ++i) stamp[i] = -1;
  for(int i=0; i<v; ++i) instack[i] = false;
  for(int i=0; i<v; ++i) is_articulation[i] = false;
  for(int i=0; i<e; ++i) is_bridge[i] = false;
  for(int i=0; i<v; ++i) group[i].clear();
  for(int i=0; i<v; ++i)
    if(stamp[i] == -1)
      dfs(i, i);
}

/**** 2-SAT part  ***/
// lit = var*2 (positive), var*2+1 (negative)
#define inv(x) ((x)^1)
vector<int> order;      // reversed topological order, only needed in 2-SAT
bool visit[V];          // visit tag
int scc_assignment[V];  // assignments for SCC groups
int var_assignment[V];  // assignments for variables

void toposort(int now){
  if(visit[now]) return;
  visit[now] = true;
  for(int i=0, n=group[now].size(); i<n; ++i){
    int node = group[now][i];
    for(int j=0, m=edge[node].size(); j<m; ++j){
      int nxt = edge[node][j].first;
      toposort(contract[nxt]);
    }
  }
  for(int i=0, n=group[now].size(); i<n; ++i)
    order.push_back(group[now][i]);
}

```

```
bool two_sat(int nVar){
  tarjan(nVar*2);
  // check SAT
  for(int i=0; i<nVar; ++i)
    if(contract[2*i] == contract[2*i+1])
      return false;
  // topological order
  order.clear();
  for(int i=0; i<new_id; ++i) visit[i] = false;
  for(int i=0; i<new_id; ++i)
    if(!visit[i])
      toposort(i);
  // initialize assignment
  for(int i=0; i<new_id; ++i) scc_assignment[i] = -1;
  // SCC assignment
  for(int i=0; i<2*nVar; ++i){
    int lit = order[i];
    if(scc_assignment[contract[lit]] == -1){
      scc_assignment[contract[lit]] = 1;
      scc_assignment[contract[inv(lit)]] = 0;
    }
  }
  // variable assignment
  for(int i=0; i<nVar; ++i)
    var_assignment[i] = scc_assignment[contract[2*i]];
  return true;
}
```

Dinic

```
#include <vector>
#include <algorithm>
#define N
#define M
#define INF
using std::min;
using std::vector;

struct Edge{int p[2] , f[2];};

Edge base[M];
int bn; //remember to reset
vector <int> E[N];
int dist[N];
int n;
int q[N];

void bfs(int t){
  for(int i=0;i<n;++i)dist[i]=INF;
  dist[t] = 0;
  int qf = 0 , qn = 0;
  q[qn++] = t;
  while( qf < qn ){
    int now = q[qf++];
    int size = E[now].size();
    for(int i=0;i<size;++i){
      Edge *p = base+E[now][i];
      int c = (p->p[0] == now)?0:1;
      if(p->f[1-c] > 0 &&
         dist[p->p[1-c]]>
         dist[p->p[c]]+1){
        dist[p->p[1-c]]=
            dist[p->p[c]]+1;
        q[qn++] = p->p[1-c];
      }
    }
  }
}

int dfs(int now,int f,int t){
  if( now == t ) return f;
  int re = 0;
  int size = E[now].size();
  for(int i=0;i<size;++i){
    Edge *p = base+E[now][i];
    int c = (p->p[0] == now)?0:1;
    if( p->f[c] > 0 &&
       dist[p->p[1-c]]==
       dist[p->p[c]]-1){
      int tf = dfs(
          p->p[1-c],
          min(f,p->f[c]), t);
      p->f[c] -= tf;
      p->f[1-c] += tf;
      f -= tf;
      re += tf;
    }
  }
  return re;
}

int dinic(int s,int t){
  int re = 0;
  while(1){
    bfs(t);
    if( dist[s] == INF )
      return re;
    else re += dfs(s,INF,t);
  }
}
```

Hungarian

```cpp
#define NIL -1
#define INF 100000000
int n,matched;
int cost[MAXNUM][MAXNUM];
bool sets[MAXNUM]; // whether x is in set S
bool sett[MAXNUM]; // whether y is in set T
int xlabel[MAXNUM],ylabel[MAXNUM];
int xy[MAXNUM],yx[MAXNUM]; // matched with whom
// given y: min{xlabel[x]+ylabel[y]-cost [x][y]} | x not in S
int slack[MAXNUM];
int prev[MAXNUM]; // for augmenting matching
inline void relabel() {
  int i,delta=INF;
  for(i=0;i<n;i++) if(!sett[i]) delta=min(slack[i],delta);
  for(i=0;i<n;i++) if(sets[i]) xlabel[i]-=delta;
  for(i=0;i<n;i++) {
    if(sett[i]) ylabel[i]+=delta;
    else slack[i]-=delta;
} }
inline void add_sets(int x) {
  int i;
  sets[x]=1;
  for(i=0;i<n;i++) {
    if(xlabel[x]+ylabel[i]-cost[x][i]<slack[i]) {
      slack[i]=xlabel[x]+ylabel[i]-cost[x][i];
      prev[i]=x;
} } }
inline void augment(int final) {
  int x=prev[final],y=final,tmp;
  matched++;
  while(1) {
    tmp=xy[x]; xy[x]=y; yx[y]=x; y=tmp;
    if(y==NIL) return;
    x=prev[y];
} }
inline void phase() {
  int i,y,root;
  for(i=0;i<n;i++) { sets[i]=sett[i]=0; slack[i]=INF; }
  for(root=0;root<n&&xy[root]!=NIL;root++);
  add_sets(root);
  while(1) {
    relabel();
    for(y=0;y<n;y++) if(!sett[y]&&slack[y]==0) break;
    if(yx[y]==NIL) { augment(y); return; }
    else { add_sets(yx[y]); sett[y]=1; }
} }
inline int hungarian() {
  int i,j,c=0;
  for(i=0;i<n;i++) {
    xy[i]=yx[i]=NIL;
    xlabel[i]=ylabel[i]=0;
    for(j=0;j<n;j++) xlabel[i]=max(cost[i][j],xlabel[i]);
  }
  for(i=0;i<n;i++) phase();
  for(i=0;i<n;i++) c+=cost[i][xy[i]];
  return c;
}
```

Poly

$|Y| = t$
$c(g) =$ 循環節個數
著色方案數 $\times$ 置換的個數 $=$

$\sum_{\text{所有的位置交換}}$ 顏色的個數$^{\text{置換的循環節個個數}}$

$$|Y^X/G| = \frac{1}{|G'|}\sum_{g\in G} t^{c(g)}$$

DMST

```
const int NO_SOLUTION = -1;
class Edge { public:
  int v,u,l;
};
Edge e[MAXEDGE],pred[MAXNUM];
int n,m,incycle[MAXNUM]={0},cycid;
bool contracted[MAXNUM];
inline int dmst(int s){
  int v,u,i,cost=0;
  for(i=0;i<n;i++) { pred[i].v=-1; contracted[i]=0; }
  while(1) {
    // find a uncontracted node with no in arc: v
    for(v=0;v<n;v++)
      if(v!=s&&!contracted[v]&&pred[v].v==-1) break;
    if(v==n) return cost; // done
    // find least-weighted in arc
    pred[v].l=INF;
    for(i=0;i<m;i++)
      if(e[i].u==v&&e[i].l<pred[v].l) pred[v]=e[i];
    if(pred[v].l==INF) return NO_SOLUTION;
    // append arc, check cycle
    cost+=pred[v].l;
    for(u=pred[v].v;u!=v&&u!=-1;u=pred[u].v);
    if(u==-1) continue;
    // trace and contract nodes in cycle
    incycle[v]=++cycid;
    for(u=pred[v].v;u!=v;u=pred[u].v) {
      contracted[u]=1;
      incycle[u]=cycid;
    }
    // update arc costs into the cycle
    for(i=0;i<m;i++)
      if(incycle[e[i].v]!=cycid&&incycle[e[i].u]==cycid)
        e[i].l-=pred[e[i].u].l;
    // contract: update labels
    pred[v].v=-1;
    for(i=0;i<m;i++) {
      if(incycle[e[i].v]==cycid) e[i].v=v;
      if(incycle[e[i].u]==cycid) e[i].u=v;
      if(e[i].v==e[i].u) e[i--]=e[--m];
    }
    for(i=0;i<n;i++) {
      if(contracted[i]) continue;
      if(pred[i].v>=0&&incycle[pred[i].v]==cycid) pred[i].v=v;
    }
  }
}
```

HFTree

```
struct HFTree {
  /* given a sequence ary[1...n] */
  /* query the index of k-th smallest element in ary[ql...qr] */
  /* 1-th smallest means the smallest */

  struct dat {
    int v, id;
    dat(int _v = 0, int _i = 0) : v(_v), id(_i) {}
    bool operator < (const dat &b) const { return v < b.v; }
  } ary[SIZE_N];

  int n;
  int ind[SIZE_LN][SIZE_N], cnt[SIZE_LN][SIZE_N];

  void clear(void){ n = 0; }

  void build(int _n, int _ary[]) {
    n = _n;
```

```
    for(int i = 1; i <= n; ++i) ary[i] = dat(_ary[i], i);
    std::sort(ary + 1, ary + n + 1);
    build(0, 1, n);
  }
  void build(int d, int le, int ri) {
    if(le == ri) {
      ind[d][le] = ary[le].id, cnt[d][le] = 0;
      return;
    }

    int mi = (le + ri >> 1);
    build(d + 1, le, mi);
    build(d + 1, mi + 1, ri);

    int p1 = le, p2 = mi + 1, p3 = le, sum = 0;
    for(; p3 <= ri; ++p3) {
      if(p1 == mi + 1 || (p2 <= ri && ind[d + 1][p1] > ind[d + 1][p2])) {
        ind[d][p3] = ind[d + 1][p2], cnt[d][p3] = sum;
        ++p2;
      } else {
        ind[d][p3] = ind[d + 1][p1], cnt[d][p3] = sum + 1;
        ++p1, ++sum;
      }
    }
  }

  int query(int ql, int qr, int k){ return query(0, 1, n, ql, qr, k); }
  int query(int d, int le, int ri, int ql, int qr, int k) {
    if(ql == qr) return ind[d][ql];
    int s1 = 0, s2 = cnt[d][qr];
    if(le != ql) s1 = cnt[d][ql - 1], s2 -= cnt[d][ql - 1];
    int mi = (le + ri >> 1);
    if(s2 >= k) return query(d + 1,
                             le, mi,
                             le + s1, le + s1 + s2 - 1,
                             k);
    else return query(d + 1,
                      mi + 1, ri,
                      mi - le + ql - s1 + 1, mi + qr - le - s1 - s2 + 1,
                      k - s2);
  }
};
```

Treap

```
struct Treap{
  Treap *l, *r;
  int pri, key;
  Treap(){};
  Treap( int _key ) :
      l(NULL), r(NULL),
      pri(rand()), key(_key){}
};
Treap* merge(Treap*a,Treap*b){
  if(!a || !b ) return a?a:b;
  if( a->pri > b->pri ){
    a->r = merge( a->r, b );
    return a;
  } else {
    b->l = merge( a, b->l );
    return b;
  }
}
/* split t into two parts,
 * part a with elements <= k,
 * part b with elements > k*/
void split(Treap*t,int k,Treap*&a,
           Treap *&b) {
  if( !t ) a = b = NULL;
  else if( t->key <= k ){
    a = t;
    split( t->r, k, a->r, b );
  } else {
    b = t;
    split( t->l, k, a, b->l);
  }
}
Treap* insert(Treap *t, int k){
  Treap *a, *b;
  split( t, k, a, b);
  return merge(merge(
           a, new Treap(k)), b);
}
Treap remove(Treap *t, int k){
  Treap *a, *b, *c;
  split(t, k-1, a, b);
  split(b, k, b, c);
  return merge(a, c);
}
```

Splay Tree

```
#include <functional>
template<typename T,
    typename Comp=std::less<T>>
class splay_tree {
private:
  Comp comp;
  unsigned long p_size;

  struct node {
    node *left, *right;
    node *parent;
    T key;
    node(const T& init = T()):
        left(0), right(0),
        parent(0),key(init){}
    ~node() {
      if(left) delete left;
      if(right) delete right;
      if(parent)
        delete parent;
    }
  } *root;

  void left_rotate(node*x){
    node *y = x->right;
    if(y) {
      x->right = y->left;
      if(y->left)
        y->left->parent=x;
      y->parent = x->parent;
    }

    if(!x->parent) root = y;
    else if(x==x->parent->left)
      x->parent->left=y;
    else x->parent->right = y;
    if(y) y->left = x;
    x->parent = y;
  }

  void right_rotate(node *x){
    node *y = x->left;
    if(y) {
      x->left = y->right;
      if(y->right)
        y->right->parent = x;
      y->parent = x->parent;
    }
    if(!x->parent) root = y;
    else if(x==x->parent->left)
      x->parent->left = y;
    else x->parent->right = y;
    if(y) y->right = x;
    x->parent = y;
  }

  void splay(node *x) {
    while(x->parent) {
if(!x->parent->parent) {
  if(x->parent->left == x)
    right_rotate(x->parent);
  else left_rotate(x->parent);
} else if(x->parent->left==x&&
          x->parent->parent->left
          ==x->parent) {
  right_rotate(x->parent->parent);
  right_rotate(x->parent);
} else if(x->parent->right==x&&
          x->parent->parent->right
          ==x->parent) {
  left_rotate(x->parent->parent);
  left_rotate(x->parent);
} else if(x->parent->left==x &&
          x->parent->parent->right
          ==x->parent) {
  right_rotate(x->parent);
  left_rotate(x->parent);
} else {
  left_rotate(x->parent);
  right_rotate(x->parent);
}
    }
  }

  void replace(node*u, node*v){
    if(!u->parent) root = v;
    else if(u==u->parent->left)
      u->parent->left = v;
    else u->parent->right = v;
    if(v) v->parent = u->parent;
  }

  node* subtree_minimum(node*u){
    while(u->left) u = u->left;
    return u;
  }

  node* subtree_maximum(node*u){
    while(u->right) u=u->right;
    return u;
  }
public:
  splay_tree() :
      root(0), p_size(0) { }

  void insert(const T &key){
    node *z = root;
    node *p = 0;

    while(z) {
      p = z;
      if(comp(z->key, key))
        z = z->right;
      else z = z->left;
    }

    z = new node(key);
    z->parent = p;

    if(!p) root = z;
    else if(
        comp(p->key,z->key))
      p->right = z;
    else p->left = z;

    splay(z);
    p_size++;
  }

  node* find(const T &key){
    node *z = root;
    while(z) {
```

```
      if(comp(z->key, key))
        z = z->right;
      else if(
          comp(key,z->key))
        z = z->left;      else return z;
    }
    return 0;
  }

  void erase(const T &key) {
    node *z = find(key);
    if(!z) return;

    splay(z);

    if(!z->left)
      replace(z, z->right);
    else if(!z->right)
      replace(z, z->left);
    else {
      node*y=subtree_minimum(
          z->right);
      if(y->parent != z) {
        replace(y, y->right);
        y->right = z->right;
        y->right->parent = y;
      }
      replace(z, y);
      y->left = z->left;
      y->left->parent = y;
    }

    delete z;
    p_size--;
  }

  const T& minimum() {
    return subtree_minimum(
        root)->key;
  }
  const T& maximum() {
    return subtree_maximum(
        root)->key;
  }

  bool empty() const {
    return root == 0;
  }
  unsigned long size()
      const { return p_size; }
};
```

Dancing Link

```
int L[maxn],R[maxn],U[maxn],D[maxn],
    S[maxn],C[maxn],sum;
void insert(int x,int c){
  /*insert element x into column c*/
  D[U[c]]=x; U[x]=U[c]; U[c]=x; D[x]=c;
  C[x]=c; S[c]++;
}
void remove(int x){
  L[R[x]]=L[x];
  R[L[x]]=R[x];
  for (int i=D[x];i!=x;i=D[i])
    for (int j=R[i];j!=i;j=R[j]){
      S[C[j]]--;
      U[D[j]]=U[j]; D[U[j]]=D[j];
    }
}
void resume(int x){
  for (int i=U[x];i!=x;i=U[i])
    for (int j=L[i];j!=i;j=L[j]){
      S[C[j]]++;
      U[D[j]]=j; D[U[j]]=j;
    }
  L[R[x]]=x;
  R[L[x]]=x;
}
void dfs(int x){ //x means dfs level
  if (R[0]==0){
    /* found a solution */
    return;
  }
  int mini=2147483647,c;
  for (int i=R[0];i!=0;i=R[i])
    if (S[i]<mini){
      mini=S[i];
      c=i;
    }
  remove(c);
  for (int i=D[c];i!=c;i=D[i]){
    for (int j=R[i];j!=i;j=R[j])
      remove(C[j]);
    dfs(x+1);
    if (/* found a solution*/)
      return;
    for (int j=L[i];j!=i;j=L[j])
      resume(C[j]);
  }
  resume(c);
}
int main()
{
  int n,m,cases,num;
  /* initialize from 1 to n*/
  num=n+1;
  for (int i=0;i<=n;i++){
    U[i]=D[i]=i;
    L[i]=i-1,R[i]=i+1;
    S[i]=0;
  }
  R[n]=0; L[0]=n;
  for (int i=0;i<m;i++){
    /* Setting the element's L and
     * R in diffierent problem
     * situations */
    insert(num,a);
    insert(num+1,b);
    L[num]=R[num]=num+1;
    L[num+1]=R[num+1]=num;
    num+=2;
  }
}
```

Hungarian Unbalanced

```
const int nil = -1;
const int inf = 1000000000;
int xn,yn,matched;
int cost[MAXN][MAXN];
bool sets[MAXN]; // whether x is in set S
bool sett[MAXN]; // whether y is in set T
int xlabel[MAXN],ylabel[MAXN];
int xy[MAXN],yx[MAXN]; // matched with whom
int slack[MAXN]; // given y: min{xlabel[x]+ylabel[y]-cost[x][y]} | x not in S
int prev[MAXN]; // for augmenting matching

inline void relabel() {
  int i,delta=inf;
  for(i=0;i<yn;i++) if(!sett[i]) delta=min(slack[i],delta);
  for(i=0;i<xn;i++) if(sets[i]) xlabel[i]-=delta;
  for(i=0;i<yn;i++) {
    if(sett[i]) ylabel[i]+=delta;
    else slack[i]-=delta;
}   }
inline void add_sets(int x) {
  int i;
  sets[x]=1;
  for(i=0;i<yn;i++) {
    if(xlabel[x]+ylabel[i]-cost[x][i]<slack[i]) {
      slack[i]=xlabel[x]+ylabel[i]-cost[x][i];
      prev[i]=x;
}   }   }
inline void augment(int final) {
  int x=prev[final],y=final,tmp;
  matched++;
  while(1) {
    tmp=xy[x]; xy[x]=y; yx[y]=x; y=tmp;
    if(y==nil) return;
    x=prev[y];
}   }
inline void phase() {
  int i,y,root;
  for(i=0;i<xn;i++) sets[i]=0;
  for(i=0;i<yn;i++) { sett[i]=0; slack[i]=inf; }
  for(root=0;root<xn&&xy[root]!=nil;root++);
  add_sets(root);
  while(1) {
    relabel();
    for(y=0;y<yn;y++) if(!sett[y]&&slack[y]==0) break;
    if(yx[y]==nil) { augment(y); return; }
    else { add_sets(yx[y]); sett[y]=1; }
}   }
inline int hungarian() {
  int i,j,c=0;
  matched=0;
  // we must have "xn<yn"
  bool swapxy=0;
  if(xn>yn) {
    swapxy=1;
    int mn=max(xn,yn);
    swap(xn,yn);
    for(int i=0;i<mn;i++) for(int j=0;j<i;j++) swap(cost[i][j],cost[j][i]);
  }
  for(i=0;i<xn;i++) {
    xy[i]=nil;
    xlabel[i]=0;
    for(j=0;j<yn;j++) xlabel[i]=max(cost[i][j],xlabel[i]);
  }
  for(i=0;i<yn;i++) {
    yx[i]=nil;
    ylabel[i]=0;
  }
  for(i=0;i<xn;i++) phase();
  for(i=0;i<xn;i++) c+=cost[i][xy[i]];
  // recover cost matrix (if necessary)
  if(swapxy) {
```

```
72      int mn=max(xn,yn);
73      swap(xn,yn);
74          for(int i=0;i<mn;i++) for(int j=0;j<i;j++) swap(cost[i][j],cost[j][i]);
75    }
76    // need special recovery if we want more info than matching value
77    return c;
78  }
```

General Graph Max Matching

```
1  #define N 256 // max vertex num
2  class Graph {
3   public:
4    // n,g[i][j]=0/1, match() => match: (i,mate[i]) (or mate[ i]=-1)
5    int n, mate[N];
6    bool g[N][N], inQ[N], inBlo[N];
7    queue<int> Q;
8    int start, newBase, prev[N], base[N];
9    int lca(int u, int v) {
10     bool path[N] = { false };
11     while(true) {
12       u = base[u]; path[u] = true;
13       if(u == start) break;
14       u = prev[mate[u]];
15     }
16     while(true) {
17       v = base[v];
18       if(path[v]) break;
19       v = prev[mate[v]];
20     }
21     return v;
22   }
23   void trace(int u) {
24     while(base[u] != newBase) {
25       int v = mate[u];
26       inBlo[base[u]] = inBlo[base[v]] = true;
27       u = prev[v];
28       if(base[u] != newBase) prev[u] = v;
29     }
30   }
31   void contract(int u, int v) {
32     newBase = lca(u, v);
33     memset(inBlo, false, sizeof(inBlo));
34     trace(u); trace(v);
35     if(base[u] != newBase) prev[u] = v;
36     if(base[v] != newBase) prev[v] = u;
37     for(int i = 0; i < n; i++)
38       if(inBlo[base[i]]) {
39         base[i] = newBase;
40         if(!inQ[i]) { Q.push(i); inQ[i] = true; }
41       }
42   }
43   bool search() {
44     memset(inQ, false, sizeof(inQ));
45     memset(prev, -1, sizeof(prev));
46     for(int i = 0; i < n; i++) base[i] = i;
47     while(!Q.empty()) Q.pop();
48     Q.push(start); inQ[start] = true;
49     while(!Q.empty()) {
50       int u = Q.front(); Q.pop();
51       for(int i = 0; i < n; i++)
52         if(g[u][i] && base[u] != base[i] && mate[u] != i){
53           if(i == start || (mate[i] >= 0 && prev[mate[i]] >= 0))
54             contract(u, i);
55           else if(prev[i] < 0) {
56             prev[i] = u;
57             if(mate[i] != -1) { Q.push(mate[i]); inQ[mate[i]] = true; }
58             else { augment(i); return true; }
59           }
60         }
61     }
```

```
    return false ;
  }
  void augment(int u) {
    while(u >= 0) {
      int v = prev[u], w = mate[v];
      mate[v] = u; mate[u] = v; u = w;
    }
  }
  int match() {
    memset(mate, -1, sizeof(mate));
    int mth = 0;
    for(int i = 0; i < n; i++) {
      if(mate[i] >= 0) continue;
      start = i;
      if(search()) mth++;
    }
    return mth;
  }
};
```

Hopcroft-Karp

```
// num of v in X, num of v in Y
int nx, ny;
// party of each x/y
int mx[100], my[100];
// forest
int px[100], py[100];
// simpl... adj
bool adj[100][100];

int trace(int y) {
  int x = py[y], yy = px[x];
  py[y] = px[x] = -1;
  if (mx[x] == -1 ||
      (yy != -1 && trace(yy))){
    mx[x] = y; my[y] = x;
    return 1;
  }
  return 0;
}

int bipartite_matching() {
  memset(mx, -1, sizeof(mx));
  memset(my, -1, sizeof(my));

  int q[100], *qf, *qb;
  int c = 0;
  while (true) {
    memset(px, -1, sizeof(px));
    memset(py, -1, sizeof(py));
    qf = qb = q;

    for (int x=0; x<nx; ++x)
      if (mx[x] == -1) {
        *qb++ = x;
        // px[x] = -2;
      }

    bool ap = false;
    for (int* tqb=qb;
         qf<tqb && !ap; tqb = qb)
      for(int x=*qf++,y=0;y<ny;++y)
        if (adj[x][y]
            /*&& mx[x] != y*/
            && py[y] == -1){
          py[y] = x;
          if (my[y] == -1)
            ap = true;
          else *qb++ = my[y],
            px[my[y]] = y;
        }
    if (!ap) break;

    for (int y=0; y<ny; ++y)
      if (my[y]==-1 && py[y]!=-1)
        c += trace(y);
  }
  return c;
}
```

Flow with upper and lower bound

有源匯上下界最大流:

1. 建立附加網路
2. 求 max_flow(S',T')
3. 若有解，求 max_flow(S,T)
4. nop
5. nop

有源匯上下界最小流

1. 建立附加網路 (無 (T,S) 邊)
2. 求 max_flow(S',T')
3. 加入 (T,S) 邊
4. 求 max_flow(S',T')
5. 若 S'、T' 滿流，則 (T,S) 即為答案

rope

```
#include <algorithm>
#include <ext/rope>

typedef __gnu_cxx::rope<char> crop;

crop str[SIZE_N];

int main(void) {
  for(int count=1;
      scanf("%d",&n)!=EOF;
      ++count, clear()) {
    int ver = 0, cnt = 0;
    for(int i = 0; i < n; ++i) {
      int t, v, p, c;
      scanf("%d", &t);
      if(t == 1) {
        scanf("%d", &p);
        scanf("%s", temp);
        ++ver, p -= cnt;
        crop t(temp);
        str[ver] = str[ver - 1];
        str[ver].insert(p, t);
      } else if(t == 2) {
        scanf("%d%d", &p, &c);
        ++ver;
        p -= cnt;
        c -= cnt;
        str[ver] = str[ver-1];
        str[ver].erase(p-1,c);
      } else if(t == 3) {
        scanf("%d%d%d",&v,&p,&c);
        v -= cnt;
        p -= cnt;
        c -= cnt;
        for(int j=p-1, jl=p+c-1;
            j<jl; ++j) {
          char c = str[v][j];
          printf("%c", c);
          if(c == 'c') ++cnt;
        }
        puts("");
      }
    }
  }
}
```

二分匹配 -交錯軌

```
#include <cstdio>

int n, k;
bool adj[NMAX + 1][NMAX + 1];
bool vis[NMAX + 1];
int  lst[NMAX + 1];

bool f(int w){
  for(int i = 1; i <= N; i++)
    if(adj[w][i] && !vis[i]){
      vis[i] = true;
      if(lst[i] == -1 || f(lst[i])){
        lst[i] = w;
        return true;
      }
    }
  return false;
}

//////////////

scanf("%d %d", &N, &M);
for(int i = 1; i <= N; i++){
  for(int j = 1; j <= M; j++)
    adj[i][j] = false;
  lst[i] = -1;
}
while(M--){
  scanf("%d %d", &a, &b);
  adj[a][b] = true;
}
int ans = 0;
for(int i = 1; i <= N; i++){
  for(int j = 1; j <= N; j++)
    vis[j] = false;
  if(f(i)) ans++;
}
```

Linux Stack Size

```
#include <sys/resource.h>

void increase_stack_size() {
  const rlim_t kStackSize = 32 * 1024 * 1024; // min stack size = 32 MB
  struct rlimit rl;
  int result;
  result = getrlimit(RLIMIT_STACK, &rl);
  if (result == 0) {
    if (rl.rlim_cur < kStackSize) {
      rl.rlim_cur = kStackSize;
      result = setrlimit(RLIMIT_STACK, &rl);
      if (result != 0)
        fprintf(stderr, "setrlimit returned result = %d\n", result);
} } }
```

AC Autamata

```
////////////////////////////////////////
// [usg] init()->add()*n->link()->match()
//       remember to free() in the end!!
// [chk] str[i] is matched if ( occ[i] ||
//      ( equ[i] != -1 && occ[equ[i]] ) )
////////////////////////////////////////

#include<stdio.h>
#include<string.h>

// # of characters
// need to map all characters to 0~(C-1)
#define C 53

struct NODE{
  NODE *next[C], *fail, *rec;
  int ind;
  NODE(){
    for(int i=0; i<C; i++)
      next[i] = NULL;
    fail = rec = NULL;
    ind = -1;
  }
}*root;
bool occ[1001]; // yes or no
int equ[1001];  // repeated strings
void init(){
  for(int i=0; i<C; i++) {
    occ[i] = false; equ[i] = -1;
  }
  root = new NODE;
}
void free(NODE *p = root){
  for(int i=0; i<C; i++){
    if(p->next[i] != NULL)
      free(p->next[i]);
  }
  delete p;
}
void add(char *s, int mrk){
  int len=strlen(s);
  NODE *p = root;
  for(int i=0; i<len; i++){
    if(p->next[s[i]] == NULL)
      p->next[s[i]] = new NODE;
    p = p->next[s[i]];
  }
  if(p->ind == -1) p->ind = mrk;
  else equ[mrk] = p->ind;
}
void link(NODE *p = root){
  NODE *tmp;
  for(int i=0; i<C; i++){
    if(p->next[i] == NULL) continue;
    tmp = p->fail;
    while(tmp != NULL &&
          tmp->next[i] == NULL)
      tmp = tmp->fail;
    if(tmp != NULL)
      p->next[i]->fail = tmp->next[i];
    else p->next[i]->fail = root;
    if(p->next[i]->fail->ind != -1)
      p->rec = p->next[i]->fail;
    else p->rec = p->next[i]->fail->rec;
  }
  for(int i=0; i<C; i++){
    if(p->next[i] != NULL)
      link(p->next[i]);
  }
}
void match(char *s){
  NODE *p = root;
  int len = strlen(s);
  NODE* tmp;
  for(int i=0; i<len; i++){
    while(p != NULL &&
          p->next[s[i]] == NULL)
      p = p->fail;
    if(p != NULL) p = p->next[s[i]];
    else p = root;
    tmp = p;
    while(tmp != NULL){
      if(tmp->ind != -1)
        occ[tmp->ind] = true;
      tmp = tmp->rec;
    }
  }
}
```

神奇小技能

```
/* sub will iterate through
 * all the subset of sup */
int sub = sup ;
do {
  /* do something */
  sub = ((sub - 1) & sup) ;
} while(sub != sup) ;

/* comb will iterate through all
 * the subset of size k of the
 * set of size n */
int comb = (1 << k) - 1 ;
while(comb < (1 << n)) {
  /* do something */
  int x=(comb&-comb),y=comb+x;
  comb=((comb & ~y)/x >> 1)|y;
}

/* discretize */
vec.erase(unique(vec.begin(),
                 vec.end()) ,
          vec.end());
```

Extend GCD

```
void extGCD(int a, int b, int& t1, int&
    t2, int& g){
  if(b == 0){ g = a; t1 = 1; t2 = 0; }
  else{
    extGCD(b, a%b, t1, t2, g);
    int tmp = t1;
    /* g = a*t1 + b*t2  */
    t1 = t2; t2 = tmp - (a/b)*t2;
    /* g = a*t1 - b*t2 */
    // t1 = -t2; t2 = -tmp - (a/b)*t2;
  }
}
```

Suffix Array

```
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int sacmp(int *r,int a,int b,int l)
{return r[a]==r[b]&&r[a+l]==r[b+l];}
//r the string, r[i]<m, r[n-1] must be 0
//sa the result, current x is the rank
void da(int *r,int *sa,int n,int m){
  int i,j,p,*x=wa,*y=wb,*t;
  for(i=0;i<m;i++) ws[i]=0;
  for(i=0;i<n;i++) ws[x[i]=r[i]]++;
  for(i=1;i<m;i++) ws[i]+=ws[i-1];
  for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
  for(j=1,p=1;p<n;j*=2,m=p){
    for(p=0,i=n-j;i<n;i++) y[p++]=i;
    for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
    for(i=0;i<n;i++) wv[i]=x[y[i]];
    for(i=0;i<m;i++) ws[i]=0;
    for(i=0;i<n;i++) ws[wv[i]]++;
    for(i=1;i<m;i++) ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i];
    for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
      x[sa[i]]=sacmp(y,sa[i-1],sa[i],j)?p-1:p++;
  }
  return;
}
//// lcp
int rank [maxn] , height [maxn ] ;
void calheight(int *r ,int *sa ,int n) {
  int i, j, k = 0;
  for (i =1;i<=n ; i++) rank[sa[i ]]= i ;
  for (i =0;i<n ; height [rank[i ++]]=k)
    for (k?k--:0, j=sa[rank[i] - 1]; r[i+k] == r[j+k]; k++);
  return ;
}
```

Lower Concave Hull

```
/* LowerConcaveHull: test with CF gym "travel" *
 * maintain a "concave hull" that support the following *
 * 1. insertion of a line *
 * 2. query of height(y) on specific x on the hull */
/* set as needed */

#include <set>

const long double EPS=1e-9;
const long double INF=1e19;
class Segment { public:
    long double m,c,x1,x2; // y=mx+c
    bool flag;
    Segment(long double _m,long double _c,long double _x1 = -INF,long double _x2=INF,bool
    _flag=0)
        :m(_m),c(_c),x1(_x1),x2(_x2),flag(_flag) {}
    long double evaly(long double x) const { return m*x+c; }
    const bool operator<(long double x) const { return x2-eps<x; }
    const bool operator < (const Segment &b) const {
        if(flag||b.flag) return *this<b.x1;
        return m+eps<b.m;
    }
};
class LowerConcaveHull { public: // maintain a hull like: \__/
    set<Segment> hull;
    /* functions */
    long double xintersection(Segment a,Segment b){ return (a.c-b.c)/(b.m-a.m); }
    inline set<Segment>::iterator replace(set<Segment> &hull,set<Segment>::iterator
    it,Segment s) {
        hull.erase(it);
        return hull.insert(s).first;
    }
```

```
    void insert(Segment s) {
        // insert a line and update hull
        set<Segment>::iterator it=hull.find(s);
        // check for same slope
        if(it!=hull.end()) {
            if(it->c+eps>=s.c) return;
            hull.erase(it);
        }
        // check if below whole hull
        it=hull.lower_bound(s);
        if(it!=hull.end()&&s.evaly(it->x1)<=it->evaly(it->x1)+EPS) return;
        // update right hull
        while(it!=hull.end()) {
            long double x=xintersection(s,*it);
            if(x>=it->x2-eps) hull.erase(it++);
            else {
                s.x2=x;
                it=replace(hull,it,Segment(it->m,it->c,x,it->x2));
                break;
            }
        }
        // update left hull
        while(it!=hull.begin()) {
            long double x=xintersection(s,*(--it));
            if(x<=it->x1+eps) hull.erase(it++);
            else {
                s.x1=x;
                it=replace(hull,it,Segment(it->m,it->c,it->x1,x));
                break;
            }
        }
        // insert s
        hull.insert(s);
    }
    void insert(long double m,long double c) { insert(Segment(m,c)); }
    long double query(long double x) {
        // return y @ given x
        set<Segment>::iterator it=hull.lower_bound(Segment(0.0,0.0,x,x,1));
        return it->evaly(x);
    }
};
```

Stoer-Wanger

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>

const int SIZE_N = (150 + 10);
const int INF = (1000000000);

int T, n, m;
int edg[SIZE_N][SIZE_N];

int nod[SIZE_N], dis[SIZE_N],
    use[SIZE_N];

int SW(void) {
  int V = n, cut = INF;
  for(int i = 0; i <= V; ++i)
    nod[i] = i;
  while(V > 1) {
    dis[nod[1]] = 0;
    use[nod[1]] = 1;
    for(int i=2; i<=V; ++i) {
      dis[nod[i]] = 0;
      use[nod[i]] = 0;
    }

    int las = 1;
    for(int cnt=2;cnt<=V;++cnt){
      for(int i=1; i<=V; ++i) {
        if(use[nod[i]] == 1)
          continue;
        dis[nod[i]] +=
            edg[nod[las]][nod[i]];
      }

      int max = -1, now = 0;
      for(int i=1; i<=V; ++i){
        if(use[nod[i]] == 1)
          continue;
        if(dis[nod[i]] > max)
          max=dis[nod[i]], now=i;
      }

      if(cnt == V) {
        for(int i=1; i<=V; ++i) {
          edg[nod[i]][nod[las]]+=
              edg[nod[i]][nod[now]];
          edg[nod[las]][nod[i]]=
              edg[nod[i]][nod[las]];
```

```
        }
        cut=min(cut,dis[nod[now]]);
        swap(nod[now], nod[V]);
        --V;
      } else {        las = now;
        use[nod[las]] = 1;
      }
    }
  }
  return cut;
}
int main(void) {
  scanf("%d%d", &n, &m);
  for(int i=0,a,b,c;i<m;++i){
    scanf("%d%d%d",&a,&b,&c);
    edg[a][b]+=c;
    edg[b][a]+=c;
  }

  printf("%d\n", SW());

  clear();
  return 0;
}
```

Miller-Rabin

```
/* miller rabin */
typedef long long LL

LL power(LL x,LL p,LL mod){
  LL s=1,m=x;
  while(p) {
    if(p&1) s=mult(s,m,mod);
    p>>=1;
    m=mult(m,m,mod);
  }
  return s;
}

bool witness(LL a,LL n,LL u,int t){
  LL x=power(a,u,n);
  for(int i=0;i<t;i++){
    LL nx=mult(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}

bool miller_rabin(LL n,int s=100){
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n==2;
  LL u=n-1;
  int t=0;
  // n-1 = u*2^t
  while(u&1) {
    u>>=1;
    t++;
  }
  while(s--) {
    LL a=randll()%(n-1)+1;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

Exact Cover

```
const int SIZE_N = 200 + 10;
const int SIZE_M = SIZE_N;
const int INF = 1000000000;

struct node {
  int U, D, L, R;
  node(int U = 0, int D = 0,
       int L = 0, int R = 0)
      : U(U), D(D), L(L), R(R) {}
};
struct DLX {
  int n, m;
  int cnt[SIZE_M];
  node ary[SIZE_N][SIZE_M];

  void init(int _n, int _m,
            int mat[SIZE_N][SIZE_M]) {
    n = _n, m = _m;
    for(int i = 0; i <= m; ++i)
      cnt[i] = (i == 0 ? m : 0);
    for(int i = 0; i <= n; ++i)
      for(int j = 0; j <= m; ++j)
        ary[i][j] = node(i, i, j, j);

    for(int i = 0; i <= m; ++i)
      ary[0][i].L = (i+m) % (m+1),
          ary[0][i].R=(i+1) % (m+1);

    for(int i = 1; i <= n; ++i)
      for(int j = 1; j <= m; ++j)
        if(mat[i][j] != 0) {
          ++cnt[j];
          ary[i][j].D=0;
          ary[i][j].U=ary[0][j].U;
          ary[ary[i][j].U][j].D=i;
          ary[0][j].U=i;
        }
    for(int i=1, j, st, las; i <= n; ++i)
      for(st=las=-1, j=1; j <= m; ++j)
        if(mat[i][j] == 0) continue;
        else if(las == -1) st=las=j;
        else {
          ary[i][j].L=las, ary[i][j].R=st;
          ary[i][st].L=ary[i][las].R=j;
          las=j;
        }}
  void del(int col) {
    int r=0, c = col;
    ary[0][ary[0][c].L].R=ary[0][c].R;
    ary[0][ary[0][c].R].L=ary[0][c].L;
```

```
51    for(r=ary[0][c].D;
52        r != 0; r=ary[r][c].D)
53      for(c=ary[r][c].R;
54          c != col; c=ary[r][c].R) {
       ary[ary[r][c].U][c].D=ary[r][c].D;
55        ary[ary[r][c].D][c].U=ary[r][c].U;
56        --cnt[c];
57      }}
58  void add(int col) {
59    int r = 0, c = col;
60    for(r=ary[r][c].U;
61        r!=0; r=ary[r][c].U)
62      for(c=ary[r][c].L;
63          c!=col; c=ary[r][c].L) {
64        ary[ary[r][c].U][c].D=r;
65        ary[ary[r][c].D][c].U=r;
66        ++cnt[c];
67      }
68    ary[0][ary[0][c].L].R=
69        ary[0][ary[0][c].R].L=c;
70  }
71  int find_EC() {
72    if(ary[0][0].L == 0 &&
73       ary[0][0].R == 0) return 0;
74    int r = 0, c = 0, ret = INF;
75    int min = INF, minp = -1;
76    for(c=ary[0][0].R;
77        c != 0; c=ary[0][c].R)
78      if(cnt[c] < min) min=cnt[c], minp=c;
79    if(min == 0) return INF;
80    del(c = minp);
81    for(r=ary[0][c].D;
82        r != 0; r=ary[r][c].D) {
83      for(int i=ary[r][c].R;
84          i != c;
85          i=ary[r][i].R) del(i);
86      int t = find_EC();
87      if(t != INF) ret=std::min(ret, t+1);
88      for(int i=ary[r][c].L;
89          i != c;
90          i=ary[r][i].L) add(i);
91    }
92    add(c);
93    return ret;
94  }
95 };
```

2 Patterns

```
 1 typedef long long lnt;
 2 const int NMAX = 25000;
 3 using std::sort;
 4 using std::swap;
 5 template<class T>
 6 inline T squ(T x);
 7
 8 struct Vector2D{
 9   lnt x, y;
10   Vector2D(lnt x=0, lnt y=0);
11   Vector2D(const Vector2D &b);
12   ~Vector2D(){}
13   Vector2D& operator =();
14   Vector2D& operator+=();
15   Vector2D& operator-=();
16   Vector2D& operator*=();
17   Vector2D& operator/=();
18   int where() const {}
19   lnt length2() const {}
20   lnt dot  (const Vector2D&b);
21   lnt cross(const Vector2D&b);
22   bool operator<(
23       const Vector2D&b)
24       const {
25     int w1=where();
26     int w2=b.where();
27     if(w1!=w2) return w1<w2;
28     if(w1==0) return false;
29     if(w1==1||w1==3||
30        w1==5||w1==7)
31       return length2()<
32              b.length2();
33     if(cross(b)>0)return true;
34     if(cross(b)<0)return false;
35     return length2()<b.length2();
36   }
37 };
38 struct Char{
39   lnt d;
40   lnt cosA, sinA, B;
41
42   Char& operator=(const Char &b);
43   bool operator==(const Char&b)
44       const{ return (d == b.d &&
45              cosA*b.B==b.cosA*B&&
46              sinA*b.B==b.sinA*B);
47   }
48   bool operator!=(const Char &b)
49       const{}
50 };
51 inline lnt deal(int n,
52                 Vector2D*ps){
53   Vector2D pAvr(0, 0);
54   for(int i = 0; i < n; i++){
55     pAvr  += ps[i];
56     ps[i] *= n;
57   }
58   for(int i=0; i<n; i++)
59     ps[i]-=pAvr;
60   sort(ps, ps + n);
61   lnt len2Sum = 0;
62   for(int i = 0; i < n; i++)
63       len2Sum +=ps[i].length2();
64   return len2Sum;
65 }
66 void deal2(int n,
67            const Vector2D *ps,
68            lnt sc,Char *out){
69  for(int i=0,j=1;
70      i<n;i++,j++,j%=n){
71   out[i].d = ps[i].length2()*sc;
72   out[i].cosA=
73       squ(ps[i].dot  (ps[j]));
74   out[i].sinA=
75       squ(ps[i].cross(ps[j]));
76   out[i].B=ps[i].length2()*
77            ps[j].length2();
78 }}
```

```
void preKMP(const Char*s,int N,
            int*out){
  out[1] = 0;
  for(int k=0,i=2;i<=N;i++){
    while(k>0&&s[k+1]!=s[i])      k=out[k];
    if(s[k+1] == s[i]) k++;
    out[i] = k;
} }
bool kmp(const Char*str,int N,
         const Char*pat,int M){
  static int pre[NMAX*2+1];
  preKMP(pat, M, pre);
  for(int k = 0, i=1; i<=N;i++){
    while(k>0&&pat[k+1]!=str[i])
      k=pre[k];
    if(pat[k+1] == str[i])k++;
    if(k == M) return true;
  }
  return false;
}
int main(){
  for(int Z,zz=scanf("%d",&Z);Z--;){
    static int N;
    static Vector2D points[NMAX];
    static Vector2D ptmp[NMAX];
    static Char chars[NMAX*2+1];
    static Char schars[NMAX+1];
    static Char tmp[NMAX*2+1];
    static lnt ls;

    scanf("%d", &N);
    for(int i=0,x,y; i<N; i++){
        scanf("%d %d", &x, &y);
        points[i] = Vector2D(x,y);
    }
    ls = deal(N, points);
    deal2(N,points,1,tmp+1);
    for(int i = 1; i <= N; i++)
        tmp[N + i] = tmp[i];
    for(int T,tt=scanf("%d",&T);
        T--;){
      static int M;
      scanf("%d", &M);
      for(int i=0,x,y;i<M;i++){
          scanf("%d %d",&x,&y);
          ptmp[i]=Vector2D(x,y);
      }
      if(N != M){
        printf("No\n");
        continue;
      }
      for(int i=0;i<M;i++)
          points[i]=ptmp[i];
      lnt sls=deal(M,points);
      for(int i=1;i<=N*2;i++){
          chars[i] = tmp[i];
          chars[i].d *= sls;
      }
      deal2(M,points,ls,schars+1);
      bool ok = false;
      ok=ok||kmp(chars,N*2,schars,M);
      for(int i=0; i<M; i++){
          points[i] = ptmp[i];
          points[i].x *= -1;
      }
      deal (M,points);
      deal2(M,points,ls,schars+1);
      ok=ok||kmp(chars,N*2,schars,M);
      printf(ok ? "Yes\n":"No\n");
    }
    printf("\n");
  }
  return 0;
}
```

Line Segment

```
struct Segment{
  Vector2D a, b;

  Segment&operator()(const Vector2D& __a, const Vector2D& __b){
    a = __a; b = __b;
    return *this;
  }

  double dist(const Vector2D& p) const {
    if((b - a).dot(p - a) <= 0) return (p - a).length();
    if((a - b).dot(p - b) <= 0) return (p - b).length();
    return fabs((b - a).cross(p - a)) / (b - a).length();
  }
  double dist(const Segment& s) const {
    if(cross(s)) return 0;
    return min(min(min(
                (a - s.a).length(),
                (a - s.b).length()),
            (b - s.a).length()),
        (b - s.b).length());
  }
  bool onA(const Vector2D& p) const {
    return ((a - p).cross(b - p) == 0 &&
            (a - p).dot(b - p) <= 0 && p != b);
  }
  bool onB(const Vector2D& p) const {
    return ((a - p).cross(b - p) == 0 &&
            (a - p).dot(b - p) <= 0 && p != a);
```

```
  }
  bool onAB(const Vector2D& p) const {
    return ((a - p).cross(b - p) == 0 && (a - p).dot(b - p) <= 0);
  }
  bool cross(const Segment& s) const {
    double c1 = (b - a).cross(s.a - a);
    double c2 = (b - a).cross(s.b - a);
    double c3 = (s.b - s.a).cross(a - s.a);
    double c4 = (s.b - s.a).cross(b - s.a);
    return (c1 * c2 <= 0 && c3 * c4 <= 0);
  }
};

struct Line{
  Vector2D a, b;

  Line(){ }
  Line(const Vector2D &__a, const Vector2D &__b): a(__a), b(__b) { }
  Line(const Line &l)                           : a(l.a), b(l.b) { }
  Line(const Segment &s)                        : a(s.a), b(s.b) { }

  Line&operator()(const Vector2D& __a, const Vector2D& __b){
    a = __a; b = __b;
    return *this;
  }

  double dist(const Vector2D& p) const {
    return fabs((b - a).cross(p - a)) / (b - a).length();
  }
  double dist(const Line& l) const {
    if((b - a).cross(l.b - l.a) != 0) return 0;
    return dist(l.a);
  }

  Vector2D inter(const Line& l) const {
    Vector2D u = b - a, v = l.b - l.a, w = l.a - a;
    if(u.cross(v) == 0) return Vector2D(DNF, DNF);
    return a + u * w.cross(v) / u.cross(v);
  }
};
```

Polygon

```
struct SimplePolygon{
  Vector2Ds verts;
  int create(const Vector2Ds& pos, int N){
    verts.clear();
    for(int i = 0; i < N; i++){
      int pre = (i - 1 + N) % N;
      int nxt = (i + 1 + N) % N;
      if((pos[pre] - pos[i]).cross(pos[nxt] - pos[i]) == 0) continue;
      verts.push_back(pos[i]);
    }
    return verts.size();
  }
  Vector2D findG() const {
    int sz = verts.size();
    Vector2D p(0, 0);
    if(sz <= 3){
      for(int i = 0, ii = verts.size(); i < ii; i++) p += verts[i];
      return p / (double)sz;
    }
    double all = 0, delta;
    for(int i = 1; i + 1 < sz; i++){
      delta = (verts[i] - verts[0]).cross(verts[i + 1] - verts[0]);
      p += (verts[0] +verts[i] + verts[i + 1]) * delta / 3.0;
      all += delta;
    }
    return p / all;
  }
  bool isIn(const Vector2D &b) const {
```

```
    bool odd = false;
    for(int i = 0, ii = verts.size(); i < ii; i++){
      const Vector2D &v1 = verts[i];
      const Vector2D &v2 = verts[(i + 1) % ii];
      if((b - v1).dot  (v2 - v1) >= 0 &&
         (b - v2).dot  (v1 - v2) >= 0 &&
         (b - v1).cross(v1 - v2) == 0) return true;
      if(v1.y < b.y && b.y <= v2.y && (v2 - v1).cross(b - v1) > 0 ||
         v2.y < b.y && b.y <= v1.y && (v1 - v2).cross(b - v2) > 0)
        odd = !odd;
    }
    return (odd == true);
  }
  double area() const {
    double ret = 0;
    for(int i = 0, ii = verts.size(); i < ii; i++){
      ret += verts[i].cross(verts[(i + 1) % ii]);
    }
    return 0.5 * ret;
  }
  double area2() const {
    double ret = 0;
    for(int i = 0, ii = verts.size(); i < ii; i++){
      ret += verts[i].cross(verts[(i + 1) % ii]);
    }
    return ret;
  }
};
struct ConvexPolygon : SimplePolygon{
  int create(Vector2Ds& pos, int N){
    sort(pos.begin(), pos.begin() + N, sort_xy);
    verts.clear();
    int top0 = 0, top = -1;
    for(int i = 0; i < N; i++){
      while(top-1>=top0 && (verts[top]-verts[top-1]).cross(pos[i]-verts[top-1]) <= 0){
        verts.pop_back(); top--;
      }
      verts.push_back(pos[i]); top++;
    }
    top0 = top;
    for(int i = N - 2; i >= 0; i--){
      while(top-1>=top0 && (verts[top]-verts[top-1]).cross(pos[i]-verts[top-1]) <= 0){
        verts.pop_back(); top--;
      }
      verts.push_back(pos[i]); top++;
    }
    verts.pop_back();
    return verts.size();
  }
};
```

計算幾何

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <algorithm>
#include <vector>
#include <deque>

#define max(x,y) (x>y?(x):(y))
#define min(x,y) (x<y?(x):(y))
#define sqr(x) (x)*(x)
#define INF (1e20)
#define EPS (1e-9)

double noEps(double n){ return (n<EPS && n>-EPS)? 0 : n; }


struct Line2D{
  // member variable
```

```
  Vector2D p0, dir;
  // constructor
  Line2D() {}
  Line2D(const Vector2D& _p0, const Vector2D& _dir): p0(_p0), dir(_dir) {}
  Line2D(const Line2D& l): p0(l.p0), dir(l.dir) {}
  // function
  Vector2D norm(){ return ~dir; }
  int side(const Vector2D& v) const {
    // on line: 0, left side: 1, right side: 0
    double tmp = dir.cross(v-p0);
    return tmp == 0? 0 : (tmp > 0? 1 : -1);
  }
  double ang() const { return dir.ang(); }
  double dist(const Vector2D& v) const { return fabs(dir.cross(v-p0)) / dir.length(); }
  bool onLine(const Vector2D& v) const { return noEps(dist(v)) == 0; }
  bool parallel  (const Line2D& l) const { return dir.parallel(l.dir); }
  bool orthogonal(const Line2D& l) const { return dir.orthogonal(l.dir); }
  bool overlap   (const Line2D& l) const { return onLine(l.p0) && parallel(l); }
  bool intersect(const Line2D& l, Vector2D& v) const {
    if(parallel(l)) return false;
    double t = (l.p0-p0).cross(l.dir) / dir.cross(l.dir);
    v = p0 + dir * t;
    return true;
  }
  Vector2D intersect(const Line2D& l) const { // guarantee not parallel
    Vector2D ret = Vector2D(INF, INF);
    intersect(l, ret);
    return ret;
  }
  // debug
  void print() const { printf("line: ["); p0.print(); dir.print(); printf("]\n"); }
};

struct Circle2D{
  // member variable
  Vector2D center;
  double r;
  // constructor
  Circle2D() {}
  Circle2D(const Vector2D& _center, double _r): center(_center), r(_r) {}
  Circle2D(const Circle2D& c): center(c.center), r(c.r) {}
  // function
  int intersect(const Line2D& l, Vector2D& p1, Vector2D& p2){
    Vector2D v0 = Vector2D(l.dir);
    Vector2D v1 = Vector2D(l.p0 - center);
    double a = v0.dot(v0);
    double b = 2*(v0.dot(v1));
    double c = v1.dot(v1) - sqr(r);
    double d = noEps(b*b - 4*a*c);
    if(d < 0) return 0;
    else if(d == 0){
      p1 = l.p0 + l.dir * (-b) / (2*a);
      return 1;
    }
    else{
      p1 = l.p0 + l.dir * (-b-sqrt(d)) / (2*a);
      p2 = l.p0 + l.dir * (-b+sqrt(d)) / (2*a);
      return 2;
    }
  }
  int intersect(const Circle2D& c, Vector2D& p1, Vector2D& p2){
    if(center == c.center){
      if(noEps(r - c.r) == 0) return -1; // overlap
      else return 0;
    }
    Vector2D v0 = Vector2D((c.center - center)*2);
    double u = c.center.dot(c.center) - center.dot(center) + sqr(r) - sqr(c.r);
    if(noEps(v0.x) != 0) return intersect(Line2D(Vector2D(u/v0.x, 0), ~v0), p1, p2);
    else return intersect(Line2D(Vector2D(0, u/v0.y), ~v0), p1, p2);
  }
  // debug
  void print() const { printf("Circle: ["); center.print(); printf("%lf ]\n", r); }
};

```

```
93 struct Polygon2D{
94   // member variable
95   std::vector<Vector2D> vList;
96   // constructor
97   Polygon2D(){ vList.clear(); }
98   Polygon2D(const std::vector<Vector2D> _vList): vList(_vList) {}
99   template<class It> Polygon2D(It first, It last): vList(first, last) {}
100  // member function
101  void clear(){ vList.clear(); }
102  void push(Vector2D v){ vList.push_back(v); }
103  double area() const {
104    double ret = 0.0;
105    int n = (int)vList.size();
106    for(int i=0; i<n-1; ++i)
107      ret += vList[i].cross(vList[i+1]) * 0.5;
108    ret += vList[n-1].cross(vList[0]) * 0.5;
109    return ret;
110  }
111  Vector2D center() const {
112    Vector2D ret;
113    int n = (int)vList.size();
114    for(int i=0; i<n-1; ++i)
115      ret = ret + (vList[i] + vList[i+1]) * vList[i].cross(vList[i+1]) / 6.0;
116    ret = ret + (vList[n-1] + vList[0]) * vList[n-1].cross(vList[0]) / 6.0;
117    return ret;
118  }
119  // debug
120  void print(){
121    for(unsigned i=0; i<vList.size(); ++i){ vList[i].print(); printf("\n"); }
122  }
123 };
124
125 // half plane intersection
126 bool cmpBySlope(Line2D line1, Line2D line2){
127   return line1.ang() < line2.ang();
128 }
129 Polygon2D halfBanana(std::vector<Line2D>& lines){
130   int n = (int)lines.size() + 4;
131   int l, r;
132   std::vector<Line2D> deq(n);
133   // infinite square boundary
134   lines.push_back(Line2D(Vector2D(-INF, -INF), Vector2D( 1,  0)));
135   lines.push_back(Line2D(Vector2D( INF, -INF), Vector2D( 0,  1)));
136   lines.push_back(Line2D(Vector2D( INF,  INF), Vector2D(-1,  0)));
137   lines.push_back(Line2D(Vector2D(-INF,  INF), Vector2D( 0, -1)));
138   // sort
139   std::sort(lines.begin(), lines.end(), cmpBySlope);
140   // find intersection result
141   l = 0; r = -1;
142 #define deqSize (r-l+1)
143   for(int i=0; i<n; ++i){
144     if(deqSize > 0 && lines[i].parallel(deq[r])){
145       if(lines[i].side(deq[r].p0) >= 0) continue;
146       if(deqSize == 1){ deq[r] = Line2D(lines[i]); continue; }
147     }
148     while(deqSize > 1 && lines[i].side(deq[r].intersect(deq[r-1])) <= 0) r--;
149     while(deqSize > 1 && lines[i].side(deq[l].intersect(deq[l+1])) <= 0) l++;
150     if(deqSize == 1 && deq[r].ang() + M_PI < lines[i].ang()) return Polygon2D();
151     deq[++r] = Line2D(lines[i]);
152   }
153   while(deqSize > 1 && deq[l].side(deq[r].intersect(deq[r-1])) <= 0) r--;
154   if(deqSize < 3) return Polygon2D();
155   // generate polygon
156   Polygon2D ret;
157   for(int i=l; i<r; ++i) ret.push(deq[i].intersect(deq[i+1]));
158   ret.push(deq[l].intersect(deq[r]));
159   return ret;
160 }
161
162 int main(){}
```