Team Reference Document

Encore @ Harbin Institute of Technology

June 25, 2013

Contents

1 String Processing

1.1 AC Automaton

```
#define code(ch) ((ch) - 'A')
const int KIND = 26, MAXN = 3000000;
struct node {
  node* nxt[KIND], *fail;
  int count, id;
} pool[MAXN], *pp, *root, *q[MAXN];
node *newNode() {
  pp->fail = NULL;
  pp->count = 0;
  memset(pp->nxt, 0, sizeof (pp->nxt));
  return pp++;
}
void initialize() {
  pp = pool;
  root = newNode();
}
void insert(const char * str, int id) {
  node * now = root;
  while (*str) {
    int i = code(*str);
    now->nxt[i] = now->nxt[i] == 0 ? newNode() : now->nxt[i];
    now = now->nxt[i];
    str++;
  }
  now->count++, now->id = id;
}
void buildFail(node*& now, int ith) {
  if(now == root) now->nxt[ith]->fail = root;
  node* tmp = now->fail;
  while(tmp) {
    if(tmp->nxt[ith] != NULL) {
      now->nxt[ith]->fail = tmp->nxt[ith];
      return;
    }
    tmp = tmp->fail;
  }
  if(tmp == NULL) now->nxt[ith]->fail = root;
}
void build() {
  int head = 0, tail = 0;
  q[tail++] = root;
  while (head != tail) {
    node * beg = q[head++];
    for (int i = 0; i < KIND; i++) {
      if (beg->nxt[i] == NULL) continue;
      buildFail(beg, i);
      q[tail++] = beg->nxt[i];
    }
  }
}
node* goStatus(node* now, int ith) {
  node * tmp = now;
  while(now->nxt[ith] == NULL && now != root)
    now = now->fail;
  now = now->nxt[ith];
  return now == NULL ? root : now;
}
void query(const char* str) {
  node * p = root, * tmp;
  int tail = 0;
  while (*str) {
    tmp = p = goStatus(p, code(*str));
    while (tmp != root && tmp->count != -1) {
      q[tail++] = tmp;
      tmp->count = -1;
      tmp = tmp->fail;
    }
    str++;
  }
}
```

1.2 Suffix Array

```
const int MAXN = 50001;
int sfx[MAXN], temp[MAXN], key[MAXN][2];
int _rank[MAXN], bucket[MAXN], height[MAXN];
// _rank from 0 to n - 1
void radixSort(int* in, int n, int idx, int* out) {
  memset(bucket, 0, sizeof(int) * (n + 1));
  for (int i = 0; i < n; i++) bucket[key[i][idx]]++;
  for (int i = 1; i <= n; i++) bucket[i] += bucket[i - 1];
  for (int i = n - 1;i >= 0;i--)out[--bucket[key[i][idx]]]=in[i];
}
#define KEY0(i) key[i][0]
#define KEY1(i) key[i][1]
int cmp(int i, int j) {
return KEY0(i) == KEY0(j) ? KEY1(i) < KEY1(j) : KEY0(i) < KEY0(j);
}
/*text can't contain 0, 0 is used as terminal*/
void buildSA(const char* text, int n) {
  for (int i = 0; i < n; i++)
    sfx[i] = i, key[i][0] = text[i], key[i][1] = 0;
  sort(sfx, sfx + n, cmp);
  for(int i = 0;i < n;i++) key[i][0] = text[sfx[i]];
  int wid = 1;
  while (wid < n) {
    _rank[sfx[0]] = 0;
    for (int i = 1; i < n; i++)
      _rank[sfx[i]] = _rank[sfx[i - 1]] + cmp(i - 1, i);
    for (int i = 0; i < n; i++) {
      sfx[i] = i;
      key[i][1] = i + wid < n ? _rank[i + wid]: 0;
    }
    radixSort(sfx, n, 1, temp);
    for(int i = 0;i < n;i++) key[i][0] = _rank[temp[i]];
    radixSort(temp, n, 0, sfx);
    for(int i = 0;i < n;i++) key[i][0] = _rank[sfx[i]];
```

```
    for(int i = 0;i < n;i++)
      key[i][1] = wid + sfx[i] < n ? _rank[sfx[i] + wid] : 0;
    wid <<= 1;
  }
}
void calHeight(const char* text, int* _rank, int n) {
  //height[i] = lcp(suffix(sa[i - 1]), suffix[sa[i]])
  for(int i = 0; i < n; i++) _rank[sfx[i]] = i;
  height[0] = 0;
  for(int i = 0, k = 0, j; i < n; i++) {
    if(_rank[i] != 0) {
      if(k > 0) k-- ;
      for (j = sfx[_rank[i] - 1]; text[i + k] == text[j + k]; k++);
      height[_rank[i]] = k;
    }
  }
}
int RMQ[MAXN][20];
//n = len(text),height[0] means nothing
void buildRMQ(int n, int* height) {
  for(int i = 1; i <= n; i++) RMQ[i][0] = height[i - 1];
  for (int j = 1; j <= log(n + 0.00) / log(2.0); j++)
    for (int i = 1; i + (1 << j) - 1 <= n; i++)
      RMQ[i][j] = min(RMQ[i][j - 1],RMQ[i + (1<<(j-1))][j - 1]);
}
int queryRMQ(int a, int b) {
  int len = log(b - a + 1.0) / log(2.0);
  return min(RMQ[a][len], RMQ[b - (1 << len) + 1][len]);
}
int queryLCP(int a, int b) {
  a = _rank[a] + 1, b = _rank[b] + 1;
  if(a > b) swap(a, b);
  return queryRMQ(a + 1, b);
}
```

1.3 Suffix Automaton

```
namespace SAM {
const int MAXN = 600000;
struct Node {
   Node *ch[26], *f; int l;
} a[MAXN], *root, *acc, *ptr;
void Initial() {
   memset(a, 0, sizeof(a));
   acc = root = a, ptr = a + 1;
}
void AddSuffix(int x) {
   using namespace std;
   Node * cur = ptr++, *fail = acc;
   cur->l = acc->l + 1; acc = cur;
   for(;fail && !fail->ch[x];fail = fail->f)
      fail->ch[x] = cur;
   if(!fail) {
      cur->f = root;
   } else if(fail->l + 1 == fail->ch[x]->l) {
      cur->f = fail->ch[x];
   } else {
      Node* r = ptr++, * q = fail->ch[x];
      *r = *q, r->l = fail->l + 1;
      cur->f = q->f = r;
      for(;fail && fail->ch[x] == q;fail = fail->f)
         fail->ch[x] = r;
   }
}
int lcs(const char * src, const char * dest) {
   Initial();
   int n = strlen(src), m = strlen(dest), ans = 0, mid = 0;
   Node * acc = root;
   for(int i = 0;i < n;i++) {
      SAM::AddSuffix(src[i] - 'a');
   }
   for(int i = 0;i < m;++i) {
      int v = dest[i] - 'a';
      if(acc->ch[v]) {
         ++mid;
         acc = acc->ch[v];
      } else {
         for(;acc && !acc->ch[v];acc = acc->f);
         mid = acc ? acc->l + 1 : 0;
         acc = acc ? acc->ch[v] : root;
      }
      ans = max(ans, mid);
   }
   return ans;
}
}
```

1.4 KMP

```
//be careful with mod string and main string
void prefix(const char *mode, int *next) {
  int m = strlen(mode), k = -1, i;
  next[0] = -1;
  for (i = 1; i < m; i++) {
    while (k > -1 && mode[k + 1] != mode[i]) k = next[k];
    if (mode[k + 1] == mode[i]) k++;
    next[i] = k;
  }
}
int KMP(const char *main, const char *mode) {
  int n = strlen(main), m = strlen(mode), q = -1, ans = 0;
  int next[LEN], i;
  prefix(mode, next);
  for (i = 0; i < n; i++) {
    while (q > -1 && mode[q + 1] != main[i]) q = next[q];
    if (mode[q + 1] == main[i]) q++;
    if (q == m - 1) {
      ans++;
      q = next[q];
    }
  }
  return ans;
}
```

1.5 Algorithm Z

```
#include <cmath>
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
void get_suffix(const char* sub, int len, int next[]) {
  //extend[i] = len(lcp(sub, sub.substr(i)))
  int pos = 1, j = 0;
  while(sub[j + 1] == sub[j]) j++;
  next[0] = len, next[pos] = j;
  for(int i = 2;i < len;i++) {
    int ll = pos + next[pos], cur = next[i - pos];
    if(ll > i + cur) {
      next[i] = cur;
    } else {
      j = max(ll - i, 0);
      while(sub[i + j] == sub[j] && i + j < len) j++;
      next[i] = j;
      pos = i;
    }
  }
}
void extend_kmp(const char* str, int n, const char* sub, int m,
       int extend[], int next[]) {
  get_suffix(sub, m, next);
  int j = 0, pos = 0;
  while(str[j] == sub[j] && j < n && j < m) j++;
  extend[0] = j;
  for(int i = 1;i < n;i++) {
    int ll = pos + extend[pos], cur = next[i - pos];
    if(ll > i + cur) {
      extend[i] = cur;
    } else {
      j = max(ll - i, 0);
      while(str[i + j] == sub[j] && i + j < n && j < m) j++;
      extend[i] = j;
      pos = i;
    }
  }
}
```

2 Network Flow

2.1 Max flow

```
const int V = 1010;
const int E = V*V*2;
const int INF = 1<<29;
typedef struct Edge{
  int v, cap, flow;
  Edge *next, *re;
}Edge;
class MaxFlow{
public:
  Edge edge[E], *adj[V], *pre[V], *arc[V];
  int e, n, d[V], q[V], numb[V];
  void Init(int x){
    n = x;
    for (int i = 0; i < n; ++i) adj[i] = NULL;
    e = 0;
  }
  void Addedge(int x, int y, int f) {
    edge[e].v = y, edge[e].cap = f, edge[e].next = adj[x], edge[e].
          re = &edge[e+1]; adj[x] = &edge[e++];
    edge[e].v = x, edge[e].cap = 0, edge[e].next = adj[y], edge[e].
          re = &edge[e-1]; adj[y] = &edge[e++];
  }
  void Bfs(int v) {
    int front = 0, rear = 0, r = 0, dis = 0;
    for (int i = 0;i < n; ++i) d[i] = n, numb[i] = 0;
    d[v] = 0;++numb[0];
    q[rear++] = v;
    while (front != rear) {
      if (front == r) ++dis, r = rear;
      v = q[front++];
      for (Edge *i = adj[v];i != NULL;i = i->next) {
        int t = i->v;
        if (d[t] == n) d[t] = dis, q[rear++] = t, ++numb[dis];
      }
    }
  }
  int Maxflow(int s, int t){
    int ret = 0, i, j;
    Bfs(t);
    for (i = 0; i < n; ++i) pre[i] = NULL, arc[i] = adj[i];
    for (i = 0; i < e; ++i) edge[i].flow = edge[i].cap;
    i = s;
    while (d[s] < n) {
      while (arc[i] && (d[i] != d[arc[i]->v]+1 || !arc[i]->flow))
            arc[i] = arc[i]->next;
      if (arc[i]) {
        j = arc[i]->v;
        pre[j] = arc[i];
        i = j;
        if (i == t) {
          int update = INF;
          for (Edge *p = pre[t];p != NULL;p = pre[p->re->v])
                checkmin(update, p->flow);
          ret += update;
```

```
        for (Edge *p = pre[t];p != NULL;p = pre[p->re->v]) p->flow
                -= update, p->re->flow += update;
        i = s;
       }
      }
      else {
       int min = n - 1;
       for (Edge *p = adj[i];p != NULL;p = p->next) if(p->flow)
             checkmin(min, d[p->v]);
       if (--numb[d[i]] == 0) return ret;
       d[i] = min + 1;
       ++numb[d[i]];
       arc[i] = adj[i];
       if (i != s) i = pre[i]->re->v;
      }
    }
    return ret;
  }
};
```

2.2 Cost flow

```
using namespace std;
typedef long long USETYPE;
const USETYPE INF = numeric_limits<USETYPE>::max();//<limits>
template<typename T = int>
class mincost {
private:
    const static int N = 1000;
    const static int E = 100000;
    struct edge {
        int u, v;
        T cost, cap;
        edge *nxt;
    } pool[E], *g[N], *pp, *pree[N];
    T dist[N];

    bool SPFA(int n,int s, int t) {
        fill(dist, dist + n, INF);
        int tail = 0, q[N] = {s};
        dist[s] = 0;
        bool vst[N] = {false};
        vst[s] = true;
        for(int i = 0; i <= tail; i++) {
            int u = q[i % n];
            for(edge *j = g[u]; j != NULL; j= j->nxt) {
                int v = j->v;
                if(j->cap && dist[u] != INF && dist[v] > dist[u] + j->
                        cost) {
                    dist[v] = dist[u] + j->cost;
                    pree[v] = j;
                    if(!vst[v]) {
                        tail++;
                        q[tail % n] = v;
                        vst[v] = true;
                    }
                }
            }
            vst[u] = false;
        }
        return dist[t] < INF;
    }
public:
#define OP(i) (((i) - pool) ^ 1)
    void addedge(int u, int v, T cap, T cost) {
        pp->u = u, pp->v = v;
        pp->cost = cost, pp->cap = cap;
        pp->nxt = g[u],g[u] = pp++;
    }
    void initialize() {
        CC(g, 0);
        pp = pool;
    }
    pair<T, T> mincostflow(int n, int s, int t) {
        T flow = 0, cost = 0;
        while(SPFA(n, s, t)) {
            T minf = INF;
            for(int i = t; i != s; i = pree[i]->u)
                minf = min(minf, pree[i]->cap);
            for(int i = t; i != s; i = pree[i]->u) {
                pree[i]->cap -= minf;
                pool[OP(pree[i])].cap += minf;
                cost += minf * pree[i]->cost;
            }
            flow += minf;
        }
        return make_pair(flow, cost);
    }
};
```

3 Data Structure

3.1 DLX exact cover

```
const int SIZE = 16, SQRTSIZE = 4;//here
const int ALLSIZE = SIZE * SIZE, ROW = SIZE * SIZE * SIZE;
const int INF = 100000000, COL = SIZE * SIZE * 4;
const int N = ROW * COL, HEAD = 0;
#define BLOCK(r, c) ((r) * SQRTSIZE + c)
#define CROW(r, c, k) ((r) + (c) * SIZE + (k) * SIZE * SIZE)
#define ROWCOL(i, j) ((i) * SIZE + (j))
#define ROWCOLOR(i, k) (ALLSIZE + (i) * SIZE + k)
#define COLCOLOR(j, k) (2 * ALLSIZE + (j) * SIZE + k)
#define BLOCKCOLOR(i, j, k) (3*ALLSIZE+BLOCK((i/SQRTSIZE),(j/
      SQRTSIZE))*SIZE+(k))
int maps[ROW][COL], ans[N];
char sudoku[SIZE][SIZE];
int r[N], l[N], u[N], d[N], c[N], s[N];
int n, m, ansd, row[N];
void resume(const int col) {
    for (int i = u[col]; i != col; i = u[i]) {
        for (int j = l[i]; j != i; j = l[j]) {
            u[d[j]] = j;
            d[u[j]] = j;
            s[c[j]]++;
        }
    }
    r[l[col]] = col;
    l[r[col]] = col;
}
void cover(const int col) {
    r[l[col]] = r[col];
    l[r[col]] = l[col];
    for (int i = d[col]; i != col; i = d[i]) {
        for (int j = r[i]; j != i; j = r[j]) {
            u[d[j]] = u[j];
            d[u[j]] = d[j];
            s[c[j]]--;
        }
    }
}
void initialize(int n, int m) {
    l[HEAD] = m;
    r[HEAD] = 1;
    for (int i = 1; i <= m; i++) {
        if (i == m) {
            r[i] = HEAD;
        } else {
            r[i] = i + 1;
        }
        l[i] = i - 1;
        c[i] = u[i] = d[i] = i;
        s[i] = 0;
    }
    int size = m;
    for (int i = 1; i <= n; i++) {
        int first = 0;
        for (int j = 1; j <= m; j++) {
            if (maps[i - 1][j - 1] == 0) continue;
            size++;
            int tmp = u[j];
            u[j] = size; d[tmp] = size;
            d[size] = j; u[size] = tmp;
            if (!first) {
                first = size;
                l[size] = r[size] = size;
            } else {
                tmp = l[first];
                r[tmp] = size;
                l[size] = tmp;
                l[first] = size;
                r[size] = first;
            }
            row[size] = i;
            s[j]++;
            c[size] = j;
        }
    }
}
bool dfs(int depth) {
    if (r[HEAD] == HEAD) {
        ansd = depth;
        return true;
    }
    int minn = INF, v;
    for (int i = r[HEAD]; i != HEAD; i = r[i]) {
        if (s[i] < minn) {
            v = i;
            minn = s[i];
        }
    }
    cover(v);
    for (int i = d[v]; i != v; i = d[i]) {
        for (int j = r[i]; j != i; j = r[j])
            cover(c[j]);
        ans[depth] = row[i] - 1;
        if (dfs(depth + 1))
            return true;
        for (int j = l[i]; j != i; j = l[j])
            resume(c[j]);
    }
    resume(v);
    ans[depth] = -1;
    return false;
}

int main() {
    n = ROW;
    m = COL;
    while (scanf(" %c", &sudoku[0][0]) == 1) {
        for(int i = 0; i < SIZE; i++) {
            for(int j = 0; j < SIZE; j++) {
                if(i + j) scanf(" %c", &sudoku[i][j]);
            }
        }
        memset(maps, 0, sizeof (maps));
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (sudoku[i][j] == '-') {
                    for (int k = 0; k < SIZE; k++) {
                        maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
                        maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
                        maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
                        maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
                    }
                } else {
                    int k = sudoku[i][j] - 'A';//here
                    maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
                    maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
                    maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
                    maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
                }
            }
            initialize(n, m);
        }
        if (dfs(0)) {
```

```
            for (int i = 0; i < ansd; i++)
               sudoku[ans[i] % SIZE][ans[i] % ALLSIZE / SIZE] = ans[i
                  ] / ALLSIZE + 'A';//here
            for(int i = 0; i < SIZE; i++) {
               for(int j = 0; j < SIZE; j++)
                  putchar(sudoku[i][j]);
               puts("");
            }
         }
         puts("");
      }
      return 0;
   }
```

3.2 DLX fuzzy cover

```
const int ROW = 56;
const int COL = 56;
const int N = ROW * COL, HEAD = 0;
const int INF = 1000000000;
int maps[ROW][COL], ansq[ROW], row[N];
int s[COL], u[N], d[N], l[N], r[N], c[N];
void build(int n, int m) {
   r[HEAD] = l;
   l[HEAD] = m;
   for (int i = 1; i <= m; i++) {
      l[i] = i - 1;
      r[i] = (i + 1) % (m + 1);
      c[i] = d[i] = u[i] = i;
      s[i] = 0;
   }
   int size = m;
   for (int i = 1; i <= n; i++) {
      int first = 0;
      for (int j = 1; j <= m; j++) {
         if (!maps[i - 1][j - 1]) continue;
         size++;
         d[u[j]] = size;
         u[size] = u[j];
         d[size] = j;
         u[j] = size;
         if (!first) {
            first = size;
            l[size] = size;
            r[size] = size;
         } else {
            l[size] = l[first];
            r[size] = first;
            r[l[first]] = size;
            l[first] = size;
         }
         c[size] = j;
         s[j]++;
      }
   }
}
inline void coverc(int col) {
   for(int i = d[col]; i != col; i = d[i]) {
      r[l[i]] = r[i];
      l[r[i]] = l[i];
   }
}
inline void resumec(int col) {
   for(int i = u[col]; i != col; i = u[i]) {
      l[r[i]] = i;
      r[l[i]] = i;
   }
}
bool vis[COL];
int H() {
   int cnt = 0;
   memset(vis,0,sizeof(vis));
   for (int i = r[HEAD]; i != HEAD; i = r[i]) {
      if (vis[i]) continue;
      cnt++;
      vis[i] = 1;
      for (int j = d[i]; j != i; j = d[j])
         for (int k = r[j]; k != j; k = r[k])
            vis[c[k]] = 1;
   }
   return cnt;
}
int cut,nextcut;
bool dfs(int dep) {
   if (!r[HEAD]) return true;
   int now, minn = ROW;
   for (int i = r[HEAD]; i != HEAD; i = r[i])
      if (minn > s[i]) {
         minn = s[i];
         now = i;
      }
   for (int j = d[now]; j != now; j = d[j]) {
      //ansq[dep]=row[rp];
      coverc(j);
      for (int i = r[j]; i != j; i = r[i])
         coverc(i);
      int tmp = dep + 1 + H();
      if(tmp > cut) nextcut = min(tmp, nextcut);
      else if (dfs(dep + 1)) return true;
      for (int i = l[j]; i != j; i = l[i])
         resumec(i);
      resumec(j);
   }
   return false;
}
int IDAstar(int n) {
   cut = H();
   nextcut = n;
   memset(vis,0,sizeof(vis));
   while(!dfs(HEAD)) {
      cut = nextcut;
      nextcut = n;
   }
   return cut;
}
```

3.3 Partition Tree

```
/* NlogN find Kth number in any interval */
class partition_tree {
private:
   static const int N = 100005;
   static const int DEPTH = 20;
   int tree[DEPTH][N * 4], sorted[N];
   int toleft[DEPTH][N * 4];
   int n;
public:
   void initialize(int n, int *array) {
      this->n = n;
      for (int i = 1; i <= n; i++)
         sorted[i] = tree[0][i] = array[i];
      sort(sorted + 1, sorted + n + 1);
   }
   void build(int l, int r, int depth) {
      if (l == r) return;
      int mid = (l + r) / 2, same = 0, less = 0;
      for (int i = l; i <= r; i++)
         less += (tree[depth][i] < sorted[mid]);
      same = mid - l + 1 - less;
      int lpos = l, rpos = mid + 1;
      for (int i = l; i <= r; i++) {
         int w = tree[depth][i];
         if (w < sorted[mid]) tree[depth + 1][lpos++] = w;
         else if (w == sorted[mid] && same) {
            tree[depth + 1][lpos++] = w;
            same--;
         }
         else
            tree[depth + 1][rpos++] = w;
         toleft[depth][i] = toleft[depth][l - 1] + lpos - l;
      }
      build(l, mid, depth + 1);
      build(mid + 1, r, depth + 1);
   }
// ptree.query(1, n, a, b, 0, k) th kth number of [a, b]
   int query(int L, int R, int l, int r, int depth, int k) {
      if (l == r) return tree[depth][l];
      int cnt, mid = (R + L) / 2, tmpl, tmpr;
      cnt = toleft[depth][r] - toleft[depth][l - 1];
      if (cnt >= k) {
         tmpl = L + toleft[depth][l - 1] - toleft[depth][L - 1];
         tmpr = tmpl + cnt - 1;
         return query(L, mid, tmpl, tmpr, depth + 1, k);
      } else {
         tmpr = r + toleft[depth][R] - toleft[depth][r];
         tmpl = tmpr - (r - l - cnt);
         return query(mid + 1, R, tmpl, tmpr, depth + 1, k - cnt);
      }
   }
};
```

3.4 Leftist Tree

```
#define DIST(v) ((v == NULL) ? -1 : (v->dist))
template<typename T, class Compare = greater<T> >
class LeftistTree {
private:
   class node {
   public:
      T v;
      int dist;
      node *rr, *ll;
      node(){rr = ll = NULL; dist = 0;}
      node(T v){this->v = v; rr = ll = NULL;dist = 0;}
   };
   node* root;
   int s;
   Compare _compare;
   node* Merge(node* left, node* right) {
      if(left == NULL) return right;
      if(right == NULL) return left;
      if(_compare(right->v, left->v)) swap(left, right);
      left->rr = Merge(left->rr, right);
      if(DIST(left->rr)>DIST(left->ll))swap(left->ll, left->rr);
      left->dist = DIST(left->rr) + 1;
      return left;
   }
   void Clear(node*& root) {
      if(root == NULL) return;
      Clear(root->ll);
      Clear(root->rr);
      delete root;
      root = NULL;
   }
public:
   LeftistTree(){root = NULL;s = 0;}
   ~LeftistTree(){Clear(root);}
   void Push(T v) {
      node * newNode = new node(v);
      root = Merge(newNode, root);
      s++;
   }
   void Clear(){Clear(root);}
   int Size(){return this->s;}
   T Top(){return root->v;}
   void Pop() {
      node *tmp = root;
      root = Merge(root->ll, root->rr);
      delete tmp;
      s--;
   }
   void Merge(LeftistTree<T>& tree) {
      this->root = Merge(root, tree.root);
      s += tree.s;
      tree.root = NULL;
   }
};
```

3.5 Cartesian Tree

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <cstring>
using namespace std;
const int N = 100000;
struct node {
  int key, value, id;
  bool operator < (const node& oth) const {
    return key < oth.key;
  }
}nodes[N];
/*lt[i] is nodes[i]'s left son, shouldn't sort again*/
int lt[N], rt[N], parent[N];
void rotate(int i) {
  while(parent[i]!=-1&&nodes[i].value<nodes[parent[i]].value) {
    rt[parent[i]] = lt[i];
    if(lt[i] != -1) parent[lt[i]] = parent[i];
    lt[i] = parent[i];
    int ff = parent[parent[i]];
    if(ff != -1) {
      parent[i] == lt[ff] ? lt[ff] = i : rt[ff] = i;
    }
    parent[i] = ff;
    parent[lt[i]] = i;
  }
}
int key[N], value[N], pos[N];
void build(int n) {
  sort(nodes, nodes + n);
  int rightmost = 0;
  for(int i = 1;i < n;i++) {
    pos[nodes[i].id] = i;
    rt[rightmost] = i;
    parent[i] = rightmost;
    rightmost = i;
    rotate(i);
  }
}
#define V(i) (i == -1 ? 0 : nodes[i].id + 1)
int main() {
  int n;
  while(scanf("%d", &n) == 1) {
    for(int i = 0;i < n;i++) {
      scanf("%d %d", &nodes[i].key, &nodes[i].value);
      nodes[i].id = i;
      key[i] = nodes[i].key;
      value[i] = nodes[i].value;
      lt[i] = rt[i] = parent[i] = -1;
    }
    build(n);
    printf("YES\n");
    for(int i = 0;i < n;i++) {
      printf("%d %d %d\n", V(parent[pos[i]]),
        V(lt[pos[i]]), V(rt[pos[i]]));
    }
  }
  return 0;
}
```

3.6 Splay

```
struct node {
#define __JUDGE if(tot == 0) return
  static const int INF = 100000000;
  node* ch[2], *pre;
  int v, minn, tot, delta, flip;
  node(int v, int tot, node* l, node* r, node* pre)
    : pre(pre), v(v), minn (v), tot(tot), delta(0), flip(0) {
    ch[0] = l, ch[1] = r;
  }
  inline int min_v() { return minn; }
  inline int size() { return tot; }
  void reverse() { __JUDGE; flip ^= 1; }
  void add(int d) { __JUDGE; minn += d, delta += d, v += d; }
  void push_down() {
    __JUDGE;
    if(delta) {
      if(ch[0]->tot) ch[0]->add(delta);
      if(ch[1]->tot) ch[1]->add(delta);
    }
    if(flip) {
      swap(ch[0], ch[1]);
      if(ch[0]->tot) ch[0]->reverse();
      if(ch[1]->tot) ch[1]->reverse();
    }
    flip = delta = 0;
  }
  void push_up() {
    __JUDGE;
    tot = ch[0]->size() + ch[1]->size() + 1;
    minn = min(v, min(ch[0]->min_v(), ch[1]->min_v()));
  }
};
class splay_tree {
public:
  splay_tree() {
    root = null = new node(node::INF, 0, 0, 0, 0);
  }
  ~splay_tree() {
    clear(root);
    delete null;
  }
  // build(0, n + 1, val) make a sequence from 1 to n
  void build(int l, int r, int* val) {
    if(l > r) return;// and make sure val[0] = va[1] = INF;
    build(l, r, root, null, val);
  }
#define centre (root->ch[1]->ch[0])
  int min_value(int a, int b) {
    makeInterval(a, b);
    return centre->min_v();
  }
  void add_value(int a, int b, int value) {
    makeInterval(a, b);
    centre->add(value);
    splay(centre, null);
  }
  void reverse(int a, int b) {
    if(a == b) return;
    makeInterval(a, b);
    centre->reverse();
    splay(centre, null);
  }
  void revolve(int a, int b, int c) {// c < b - a + 1
    if(c == 0) return;
    int len = b - a + 1;
    reverse(a, a + len - c - 1);
    reverse(a + len - c, b), reverse(a, b);
  }
  void insert(int a, int c) {
    makeInterval(a + 1, a);
    centre = new node(c, 1, null, null, root->ch[1]);
    root->ch[1]->push_up(), root->push_up();
    splay(centre, null);
  }
  void erase(int a) {
    makeInterval(a, a);
    delete centre;
    centre = null;
    root->ch[1]->push_up(), root->ch[0]->push_up();
  }
#undef centre
  void clear() { clear(root); }
private:
  node* root, * null;
  void clear(node*& now) {
    if(now == null) return;
    clear(now->ch[0]), clear(now->ch[1]);
    delete now;
    now = null;
  }
  /* 0: right rotate, 1: left rotate*/
  void rotate(node* x, int type) {
    node *y = x->pre;
    y->push_down(), x->push_down();
    y->ch[!type] = x->ch[type];
    if (x->ch[type] != null)
      x->ch[type]->pre = y;
    x->pre = y->pre;
    if (y->pre != null) {
      if(y->pre->ch[1] == y)
        y->pre->ch[1] = x;
      else
        y->pre->ch[0] = x;
    }
    x->ch[type] = y, y->pre = x;
    if (y == root) root = x;
    y->push_up(), x->push_up();
  }
  void splay(node* x, node* f) {
    x->push_down();
    while(x->pre != f) {
      if (x->pre->pre == f) {
        if (x->pre->ch[0] == x)
          rotate(x, 1);
        else
          rotate(x, 0);
      } else {
        node *y = x->pre;
        node *z = y->pre;
        if (z->ch[0] == y) {
          if (y->ch[0] == x) // l
            rotate(y, 1), rotate(x, 1);
          else // z
            rotate(x, 0), rotate(x, 1);
        } else {
          if (y->ch[1] == x) // l
            rotate(y, 0), rotate(x, 0);
          else // z
            rotate(x, 1), rotate(x, 0);
        }
      }
    }
    x->push_up();
  }
  void build(int l, int r, node*& now, node* pre, int* val) {
    if(l > r) return;
    int mid = (l + r) / 2;
    now = new node(val[mid], 1, null, null, pre);
    build(l, mid - 1, now->ch[0], now, val);
    build(mid + 1, r, now->ch[1], now, val);
    now->push_up();
  }
  // the flag node is !not! included, be careful when make
       interval
  void findK(int k, node* pre) {
    node* now = root;
    while(true) {
      now->push_down();
      int s = now->ch[0]->size();
      if(s == k) break;
      else if(s > k)
        now = now->ch[0];
      else {
        now = now->ch[1];
        k -= s + 1;
      }
    }
    splay(now, pre);
  }
  void makeInterval(int a, int b) {
    findK(a - 1, null), findK(b + 1, root);
  }
}tree;
```

4 Graph Theory

4.1 2-Satisfiability

```
/* 2-sat template node is from 0
 * i and i^1 is a bool variable(true or false)
 * conjunctive normal form with 2-sat
 * x V y == 1 => edge(~x-->y) and edge(~y-->x)
 * x V y == 0 => (~x V ~x) & (~y V ~y)
 * x ^ y == (~x V ~y) & (x V y)
 * x & y == 1 (x V x) & (y V y)
 * x & y == 0 (~x V ~y) */
const int V = 20000, E = 20480 * 4;
const int RED = 1, BLUE = 2;
struct edge {
   int v;
   edge * nxt;
} pool[E], *g[V], *pp, *gscc[V];
int st[V], top, tms[V], pt;
bool reach[V];
int dfn[V], low[V], idx[V], sccCnt, depth;
int color[V], pre[V];
void addedge(int a, int b, edge *g[]) {
   pp->v = b;
   pp->nxt = g[a];
   g[a] = pp++;
}
void initialize() {
   memset(reach, 0, sizeof (reach));
   memset(dfn, 0, sizeof (dfn));
   memset(g, 0, sizeof (g));
   top = sccCnt = depth = 0, pp = pool;
}
void dfs(int x) {
   st[++top] = x;
   dfn[x] = low[x] = ++depth;
   int w;
   for (edge * i = g[x]; i != NULL; i = i->nxt) {
      w = i->v;
      if (reach[w]) continue;
      else if (dfn[w] == 0) {
         dfs(w);
         low[x] = min(low[x], low[w]);
      }
      else low[x] = min(low[x], dfn[w]);
   }
   if (low[x] == dfn[x]) {
      sccCnt++;
      do {
         w = st[top--];
         idx[w] = sccCnt - 1;
         reach[w] = true;
      }while (w != x);
   }
}
void toposort(int v) {
   reach[v] = true;
   for (edge *i = gscc[v]; i != NULL; i = i->nxt)
      if (!reach[i->v]) toposort(i->v);
   tms[pt++] = v;
}
void build_regraph(int n)/*anti-graph*/ {
   memset(gscc, 0, sizeof (gscc));//anti-graph scc
   memset(pre, -1, sizeof (pre));//the new node to every scc
   for (int i = 0; i < n; i++) {
      if (pre[idx[i]] == -1) pre[idx[i]] = i;
      for (edge * ptr = g[i]; ptr != NULL; ptr = ptr->nxt) {
         int w = ptr->v;
         if (idx[i] != idx[w]) addedge(idx[w], idx[i], gscc);
      }
   }
}
void becolor(int v) {
   color[v] = BLUE;
   for (edge *i = gscc[v]; i != NULL; i = i->nxt)
      if (!color[i->v]) becolor(i->v);
}
void output(int n)/* Topological Sort */ {
   memset(color, 0, sizeof (color));//color white
   for (int i = 0; i < pt; i++) {
      if (!color[tms[i]])/*color as Topological order*/{
         color[tms[i]] = RED;
         int v = idx[pre[tms[i]] ^ 1];
         if (color[v] == 0) becolor(v);
      }
   }
   for (int i = 0; i < n; i += 2) {
      if (color[idx[i]] == RED)
         printf("%d\n", i + 1);
      else //if (color[idx[i ^ 1]] == RED)
         printf("%d\n", (i ^ 1) + 1);
   }
}
bool solve(int n)/*i and ~i can not be in the same scc */ {
   for (int i = 0; i < n; i++) if (!reach[i]) dfs(i);
   for (int i = 0; i < n; i++) if (idx[i] == idx[i ^ 1])return
         false;
   build_regraph(n);
   pt = 0;
   memset(reach, 0, sizeof (reach));
   for (int i = 0; i < sccCnt; i++)
      if (!reach[i]) toposort(i);
   reverse(tms, tms + pt);
   output(n);
   return true;
}
int main() {
   int n, m;
   while (scanf("%d %d", &n, &m) == 2) {
      initialize();
      n *= 2;
      while (m--) {
         int a, b;
         scanf("%d %d", &a, &b);
         a--, b--;
         addedge(a, b ^ 1, g);
         addedge(b, a ^ 1, g);
      }
      if (!solve(n)) printf("NIE\n");
   }
   return 0;
}
```

4.2 Edge Cut

```
/*HOJ2360
* idx is new node of the tree
* pool should be big enough */
const int SIZE = 5000, ROOT = 0, E = 80000;
struct edge {
   int v, id;
   edge *nxt;
} pool[E], *g[SIZE], *pp, *bg[SIZE];
stack<int> st;
bool flag[E];//label the edge in case of multi-edge
int depth, ebcc, dfn[SIZE], low[SIZE], idx[SIZE];
void initialize() {
   memset(g, 0, sizeof(g));
   memset(flag, 0, sizeof(flag));
   memset(bg, 0, sizeof(bg));
   memset(dfn, 0, sizeof(dfn));
   pp = pool, depth = 1, ebcc = 0;
}
void addedge(int v, int w, edge *g[], int id = 0) {
   pp->v = w, pp->nxt = g[v];
   pp->id = id, g[v] = pp++;
}
void dfs(int v) {
   st.push(v);
   dfn[v] = low[v] = depth++;
   int w, x;
   for (edge* i = g[v]; i != NULL; i = i->nxt) {
      w = i->v;
      if (flag[i->id]) continue;
      flag[i->id] = true;
      if (dfn[w]) low[v] = min(low[v], dfn[w]);
      else {
         dfs(w);
         low[v] = min(low[v], low[w]);
         if (low[w] > dfn[v]) {
            ebcc++;
            do {
               x = st.top();
               st.pop();
               idx[x] = ebcc;
            }while (x != w);
         }
      }
   }
}
void solve()/*find out the cut and build the tree*/ {
   dfs(ROOT);//ROOT = 0 as usual
   if (!st.empty()) ebcc++;
   while (!st.empty()) {
      idx[st.top()] = ebcc;
      st.pop();
   }
}
```

4.3 Vertex Cut

```
/* hoj 1789 Electricity
 * the graph is not connected
 * cnt records the number of BBC, it's an cut P if != 0*/
const int V = 10000;
vector<int> adj[V];
int low[V], dfn[V], cnt[V], depth;
void initialize(int n)
{
   REP(i, 0, n) adj[i].clear();
   CC(cnt, 0);CC(dfn, 0);
   depth = 0;
}
void dfs(int x, const int ROOT)
{
   low[x] = dfn[x] = ++depth;
   int s = adj[x].size(), w, num = 0;
   REP(i, 0, s)
   {
      w = adj[x][i];
      if (!dfn[w])
      {
         num++;
         dfs(w, ROOT);
         low[x] = min(low[w], low[x]);
         if (x == ROOT && num >= 2)
            cnt[x]++;
         if (x != ROOT && dfn[x] <= low[w])
            cnt[x]++;
      }
      else low[x] = min(low[x], dfn[w]);
   }
}
int solve(int n)
{
   int cc = 0;
   REP(i, 0, n)
   {
      if (dfn[i] == 0)
      {
         dfs(i, i);
         cc++;
      }
   }
   return cc;
```

```
}
int main()
{
   int n, m, x, y;

   while (scanf("%d %d", &n, &m) == 2 && n + m)
   {
      initialize(n);
      REP(i, 0, m)
      {
         scanf("%d %d", &x, &y);
         adj[x].push_back(y);
         adj[y].push_back(x);
      }
      int ans = solve(n);
      if (m == 0) printf("%d\n", n - 1);
      else printf("%d\n", ans + *max_element(cnt, cnt + n));
   }
   return 0;
}
```

4.4 Hopcroft Karp

```
const int N = 500, M = 500, INF = 1 << 29;
bool g[N][M], chk[M];
int Mx[N], My[M], dx[N], dy[M], dis;
bool searchP(int n, int m) {
  queue<int> Q;
  dis = INF;
  CC(dx, -1);CC(dy, -1);
  for (int i = 0; i < n; ++ i)
    if (Mx[i] == -1) {
      Q.push(i);
      dx[i] = 0;
    }
  while (!Q.empty()) {
    int u = Q.front();
    Q.pop();
    if (dx[u] > dis) break;
    for (int v = 0; v < m; ++ v)
      if (g[u][v] && dy[v] == -1) {
        dy[v] = dx[u] + 1;
        if (My[v] == -1) dis = dy[v];
        else {
          dx[My[v]] = dy[v] + 1;
          Q.push(My[v]);
        }
      }
  }
  return dis != INF;
}
bool Augment(int u, const int m) {
  REP(v, 0, m)
    if (g[u][v] && !chk[v] && dy[v] == dx[u] + 1) {
      chk[v] = true;
      if (My[v] != -1 && dy[v] == dis) continue;
      if (My[v] == -1 || Augment(My[v], m)) {
        My[v] = u;
        Mx[u] = v;
        return true;
      }
    }
  return false;
}
int MaxMatch(int n, int m) {
  int ans = 0;
  CC(Mx, -1);CC(My, -1);
  while (searchP(n, m)) {
    CC(chk, false);
    REP(i, 0, n)
      if (Mx[i] == -1 && Augment(i, m)) ++ ans;
  }
  return ans;
}
```

4.5 Hungary Algorithm

```
/*1. simple maximum match
2.min path cover of DAG = |V| - max match
define: find some edge cover all the nodes
build PXP Bipartite graph do the maximum match
3.min path cover of Bipartite graph = max match
define : find some point cover all the edge(konig)
4.chessBoard is a Bipartite graph,then you know
5.max independant set(Bipartite graph)=|V| - max match
v is all the point of (set A and set B)
6.largest cloud(Bipartite graph) = max independant set of
       Complement*/
const int V = 201, E = 10000;
vector<int> adj[V];
int ym[V], chk[V];
bool find_path(int x) {
  FOREACH(adj[x], i) {
    if (chk[*i]) continue;
    chk[*i] = true;
    if (ym[*i] == -1 || find_path(ym[*i])) {
      ym[*i] = x;
      return true;
    }
  }
  return false;
}
int solve(int n) {
  CC(ym, -1);
  int res = 0;
  for (int i = 0; i < n; i++) {
    memset(chk, 0, sizeof (chk));
    if (find_path(i)) res++;
  }
  return res;
}
```

4.6 KM

```
struct Graph {
   int ny, nx;
   double w[N][N];
   double lx[N], ly[N];
   int linky[N];
   int visx[N], visy[N];
   double slack[N];
   void init(int nn,int mm) {
      nx = nn;
      ny = mm;
   }
   bool find(int x) {
      visx[x] = 1;
      for(int y = 1; y <= ny; y++) {
         if(visy[y]) continue;
         double t = lx[x] + ly[y] - w[x][y];
         if(t < eps) {
            visy[y] = 1;
            if(linky[y] == -1 || find(linky[y])) {
               linky[y] = x;
               return true;
            }
         } else if(slack[y] > t) {
            slack[y] = t;
         }
      }
      return false;
   }
   double KM() {
      memset(linky, -1, sizeof(linky));
      for(int i = 1; i <= nx; i++) lx[i] = -INF;
      memset(ly, 0, sizeof(ly));
      for(int i = 1; i <= nx; i++)
         for(int j = 1; j <= ny; j++)
            if(w[i][j] > lx[i]) lx[i] = w[i][j];
      for(int x = 1; x <= nx; x++) {
         for(int i = 1; i <= ny; i++) slack[i] = INF;
         while(true) {
            memset(visx, 0, sizeof(visx));
            memset(visy, 0, sizeof(visy));
            if(find(x)) break;
            double d = INF;
            for(int i = 1; i <= ny; i++)
               if(!visy[i]) d = min(d, slack[i]);
            if(d == INF) return -1;
            for(int i = 1; i <= nx; i++)
               if(visx[i]) lx[i] -=d;
            for(int i = 1; i <= ny; i++)
               if(visy[i]) ly[i] += d;
               else slack[i] -= d;
            }
      }
      int cnt = 0;
      for(int i = 1; i <= ny; i++)
         if(linky[i] != -1) cnt++;
      if(cnt != nx) return -1;
      double tp = 0;
      for(int i = 1; i <= ny; i++)
         if(linky[i] != -1 ) tp += w[linky[i]][i];
      return tp;
   }
}g;
```

4.7 Stable Marriage

```
/* boy[i][j] gg[i] to mm[j]
 * girl[i][j] mm[i] to gg[j]*/
const int N = 26;
const int M = 128;
int boy[N][N], girl[N][N];
int my[N], mx[N], now[N];
void Gale_Shapley(int n) {
   queue<int> q;
   for(int i = 0; i < n; i++) q.push(i);
   while(!q.empty()) {
      int i = q.front();q.pop();
      int j = now[i]++, mm = boy[i][j];
      if(my[mm] == -1 || girl[mm][my[mm]] > girl[mm][i]) {
         if(my[mm] != -1) q.push(my[mm]);
         my[mm] = i, mx[i] = mm;
      }
      else q.push(i);
   }
}
char nameB[N], nameG[N];
void output(int n) {
   for(int i = 0; i < n; i++)
      printf("%c %c\n", nameB[i], nameG[mx[i]]);
}
int hashB[M], hashG[M];
void initialize() {
   memset(hashB,0,sizeof(hashB)),memset(hashG,0,sizeof(hashG));
   memset(my, -1, sizeof(my)), memset(now, 0, sizeof(now));
}
```

4.8 Maximum Clique

```
const int N = 50;
int maps[N][N], found, mc, n;
int c[N], answer[N], record[N];
void dfs(int GraphSize,int *s, int CliqueSize) {
  if(GraphSize == 0) {
    if(CliqueSize > mc) {
      mc = CliqueSize;
      found = true;
      copy(record, record + mc, answer);
    }
    return ;
  }
  for(int i = 0; i < GraphSize; i++) {
    if(CliqueSize + GraphSize <= mc || c[s[i]] + CliqueSize <= mc)
      return;
    int tmps[N],tmpSize = 0;
    record[CliqueSize] = s[i];
    for(int j = i + 1; j < GraphSize; j++)
      if(maps[s[i]][s[j]]) tmps[tmpSize++] = s[j];
    dfs(tmpSize, tmps, CliqueSize + 1);
    if(found) return ;
  }
}
void initialize() {
  memset(maps, false, sizeof(maps));
  mc = 0;
}
int findMaxClique(int n) {
  for(int i = n - 1; i >= 0; i--) {
    found = false;
    int tail = 0, s[N];
    for(int j = i + 1; j < n; j++)
      if(maps[i][j])
        s[tail++] = j;
    record[0] = i;
    dfs(tail, s, 1);
    c[i] = mc;
  }
  return mc;
}
```

4.9 Maximal Clique

```
const static int N = 130;
int n, maps[N][N], cnt;
void CountMaximalClique(int *p, int ps, int *x, int xs) {
   if(ps == 0) {
      if(xs == 0) cnt++;
      return ;
   }
   for(int i = 0; i < xs; i++) {
      int j, v = x[i];
      for(j = 0; j < ps && maps[p[j]][v]; j++);
      if(j == ps) return;
   }
   int tmpp[N], tmpps = 0, tmpx[N], tmpxs = 0;
   for(int i = 0; i < ps; i++) {
      int v = p[i];
      tmpps = tmpxs = 0;
      for(int j = i + 1; j < ps; j++) {
         int u = p[j];
         if(maps[v][u])
            tmpp[tmpps++] = u;
      }
      for(int j = 0; j < xs; j++) {
         int u = x[j];
         if(maps[v][u])
            tmpx[tmpxs++] = u;
      }
      CountMaximalClique(tmpp, tmpps, tmpx, tmpxs);
      if(cnt > 1000) return;
      x[xs++] = v;
   }
}
int CountMaximalClique() {
   cnt = 0;
   int p[N], x[N];
   for(int i = 0; i < n; i++) p[i] = i;
   CountMaximalClique(p, n, x, 0);
   return cnt;
}
```

4.10 Lowest Common Ancestor

```
const int N = 100000;
int father[N], chk[N], dgr[N];
vector<vector<int> > adj, query;
int set_find(int i) {
   return father[i] = i == father[i] ? i : set_find(father[i]);
}
void initialize(int n) {
   adj.assign(n, vector<int>());
   query.assign(n, vector<int>());
   CC(dgr, 0);CC(chk, 0);
}
void LCA(int u) {
   father[u] = u;
   FOREACH(adj[u], i) {
      LCA(*i),father[*i] = u;
   }
   chk[u] = 1;
   FOREACH(query[u], i)if(chk[*i])
         printf("%d\n", set_find(*i));
}
```

4.11 Minimum Cut Algorithm

```
const int V = 501, INF = 100000000, S = 1;
int maps[V][V], dist[V], pre;
bool vst[V], del[V];
void intialize()/* start with 1 */ {
   memset(del, false, sizeof (del));
   memset(maps, 0, sizeof (maps));
}
int maxinum_adjacency_search(int t, int n) {
   for (int i = 1; i <= n; i++)
      if (!del[i]) dist[i] = maps[S][i];
   memset(vst, false, sizeof (vst));
   vst[S] = true;
   int k = S;
   for (int j = 1; j <= n - t; j++) {
      int tmp = -INF;
      pre = k;
      for (int i = 1; i <= n; i++)
         if (!vst[i] && !del[i] && tmp < dist[i]) {
            tmp = dist[i];
            k = i;
         }
      vst[k] = true;
      for (int i = 1; i <= n; i++)
         if (!vst[i] && !del[i]) dist[i] += maps[k][i];
   }
   return k;
}
int Stoer_Wagner(int n) {
   int mcut = INF;
   for (int i = 1; i < n; i++) {
      int idx = maxinum_adjacency_search(i, n);
      mcut = min(mcut, dist[idx]);
      del[idx] = true;
      for (int i = 1; i <= n; i++) {
         if (!del[i] && i != pre) {
            maps[pre][i] += maps[idx][i];
            maps[i][pre] = maps[pre][i];
         }
      }
   }
   return mcut;
}
```

4.12 Degree-constrained Spanning Tree

```
const int N = 25, LEN = 15, INF = 1<<29;
int dis[N][N]= {}, f[N]= {}, father[N]= {}, n;
bool visit[N]= {};
bool used[N][N]= {};
void Dfs(int last, int v) {
   visit[v] = 1;
   if (!father[v]) f[v] = -INF;
   else f[v] = max(dis[last][v], f[father[v]]);
   for (int i = 0; i < n; ++i)
      if (!visit[i] && used[v][i])
         father[i] = v, Dfs(v, i);
}
int DegreeLimitMST(int k) {
   int ret = 0, path[N], group[N]= {}, g = 0, pre[N], degree = 0;
   memset(used, 0, sizeof(used));
   for (int i = 1; i < n; ++i)
      if (!group[i]) {
         group[i] = ++g;
         for (int j = 0; j < n; ++j)
            path[j] = dis[i][j], pre[j] = i;
         while (1) {
            int tmp = INF, mark = -1;
            for (int j = 1; j < n; ++j)
               if (!group[j] && path[j] < tmp)
                  tmp = path[j], mark = j;
            if (mark == -1) break;
            used[pre[mark]][mark] = 1, used[mark][pre[mark]] = 1;
            ret += tmp;
            group[mark] = g;
            for (int j = 1; j < n; ++j)
               if (!group[j] && path[j] > dis[mark][j])
                  path[j] = dis[mark][j], pre[j] = mark;
         }
      }
   for (int i = 1; i <= g; ++i) {
      int tmp = INF, mark = -1;
      for (int j = 1; j < n; ++j)
         if (group[j] == i && tmp > dis[0][j])
            tmp = dis[0][j], mark = j;
      used[0][mark] = used[mark][0] = 1;
      ret += tmp;
      ++degree;
   }
   while (degree < k) {
      memset(visit, 0, sizeof(visit));
      Dfs(0, 0);
      int tmp = INF, mark = -1, t;
      for (int i = 1; i < n; ++i)
         if (!used[0][i] && dis[0][i] != INF) {
            t = ret+dis[0][i]-f[i];
            if (tmp > t) tmp = t, mark = i;
         }
      if (ret <= tmp) break;
      ret = tmp;
      used[0][mark] = used[mark][0] = 1;
      tmp = f[mark];
      while (dis[father[mark]][mark] != tmp) mark = father[mark];
      used[mark][father[mark]] = used[father[mark]][mark] = 0;
      ++degree;
   }
   return ret;
}
```

4.13 Minimum Directed Tree

```
const int N = 1010, E = N * N;
const LL INF = 10000000000LL;
template<typename T>
struct Edge {
   int u, v;
   T c;
};
Edge<LL> edge[E];
int label[N], pre[N], visit[N];
template<typename T>
T treeGraph(int n, int m, int root, Edge<T>* edge) {
   int cnt = 0;
   T inEdge[N], ans = 0;
   while(true) {
      fill(inEdge, inEdge + n, INF);
      REP(i, 0, m) {
         int u = edge[i].u, v = edge[i].v;
         if(v != u && edge[i].c < inEdge[v])
         {
            pre[v] = u;
            inEdge[v] = edge[i].c;
         }
      }
      REP(i, 0, n) {
         if(i == root) continue;
         if(inEdge[i] == INF) return -1;
      }
      int now = 0;
      CC(label, -1), CC(visit, -1);
      inEdge[root] = 0;
      REP(i, 0, n) {
         ans += inEdge[i];
         int v = i;
         while(visit[v] != i && label[v] == -1 && v != root) {
            visit[v] = i;
            v = pre[v];
         }
         if(v != root && label[v] == -1) {
            for(int u = pre[v]; u != v; u = pre[u])
               label[u] = now;
            label[v] = now++;
         }
      }
      if(now == 0) break;
      REP(i, 0, n) if(label[i] == -1) label[i] = now++;
      REP(i, 0, m) {
         int v = edge[i].v;
         edge[i].v = label[edge[i].v];
         edge[i].u = label[edge[i].u];
         if(edge[i].v != edge[i].u) edge[i].c -= inEdge[v];
      }
      root = label[root];
      n = now;
   }
   return ans;
}
```

5 Math

5.1 Minimum Directed Tree

```
const int N = 1010, E = N * N;
const LL INF = 10000000000LL;
template<typename T>
struct Edge {
   int u, v;
   T c;
};
Edge<LL> edge[E];
int label[N], pre[N], visit[N];
template<typename T>
T treeGraph(int n, int m, int root, Edge<T>* edge) {
   int cnt = 0;
   T inEdge[N], ans = 0;
   while(true) {
      fill(inEdge, inEdge + n, INF);
      REP(i, 0, m) {
         int u = edge[i].u, v = edge[i].v;
         if(v != u && edge[i].c < inEdge[v])
         {
            pre[v] = u;
            inEdge[v] = edge[i].c;
         }
      }
      REP(i, 0, n) {
         if(i == root) continue;
         if(inEdge[i] == INF) return -1;
      }
      int now = 0;
      CC(label, -1), CC(visit, -1);
      inEdge[root] = 0;
      REP(i, 0, n) {
         ans += inEdge[i];
         int v = i;
         while(visit[v] != i && label[v] == -1 && v != root) {
            visit[v] = i;
            v = pre[v];
         }
         if(v != root && label[v] == -1) {
            for(int u = pre[v]; u != v; u = pre[u])
               label[u] = now;
            label[v] = now++;
         }
      }
      if(now == 0) break;
      REP(i, 0, n) if(label[i] == -1) label[i] = now++;
      REP(i, 0, m) {
         int v = edge[i].v;
         edge[i].v = label[edge[i].v];
         edge[i].u = label[edge[i].u];
         if(edge[i].v != edge[i].u) edge[i].c -= inEdge[v];
      }
      root = label[root];
      n = now;
   }
   return ans;
}
```

6 Math

6.1 Matrix

```
#define rep(i,a,b) for(int i=a;i<b;++i)
typedef long long ll;

const int mod = 1000000;
struct Matrix {
   static const int N = 27;
   ll v[N][N];
   int s;

   Matrix(int ss) {
      s = ss;
      memset(v, 0, sizeof (v));
   }

   inline void setE() {
      rep(i, 0, s) v[i][i] = 1;
   }

   Matrix operator +(const Matrix & b) {
      Matrix res = *this;
      rep(i, 0, s) rep(j, 0, s) {
         res.v[i][j] += b.v[i][j];
         if (res.v[i][j] >= mod)
            res.v[i][j] -= mod;
      }
      return res;
   }

   Matrix operator -(const Matrix & b) {
      Matrix res = *this;
      rep(i, 0, s) rep(j, 0, s) {
         res.v[i][j] -= b.v[i][j];
         if (res.v[i][j] < 0)
            res.v[i][j] += mod;
      }
      return res;
   }

   Matrix operator *(const Matrix & b) {
      Matrix res(s);
      rep(i, 0, s)rep(j, 0, s) {
         ll temp = 0;
         rep(k, 0, s) temp += v[i][k] * b.v[k][j];
         res.v[i][j] = temp % mod;
      }
      return res;
   }

   Matrix pow(int b) {
      Matrix res(s), t = *this;
      res.setE();
      while (b) {
         if (b & 1) res = res * t;
         t = t*t;
         b >>= 1;
      }
      return res;
   }

   Matrix powS(int b) {
      Matrix res(s), t = *this, p = t;
      res.setE();
      while (b) {
         if (b & 1) {
            res = res + t;
            t = t* p;
         }
         t = t + t*p;
         b >>= 1;
         p = p * p;
      }
      return res;
   }
};
```

6.2 Number Thoery

6.2.1 Phi

```
typedef long long LL;
const int N = 1000001;
int prime[N], np;
bool vis[N];
LL phi[N];

void getPhi() {
   int t;
   np = 0;
   memset(vis, 0, sizeof (vis));
   for (int i = 1; i < N; ++i)phi[i] = i;
   for (int i = 2; i < N; ++i) {
```

```
        if (!vis[i]) {
            prime[np++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; j < np && (t = i * prime[j]) < N; ++j) {
            vis[t] = 1;
            if (i % prime[j] == 0) {
                phi[t] = phi[i] * prime[j];
                break;
            }
            phi[t] = phi[i]*(prime[j] - 1);
        }
    }
}
```

6.2.2 $a^x == b(modn)$

$a^x = b(modp)$ p is prime number:Baby-step-gaint-step

```
typedef long long llong;
int mod_pow(int a, int b, int n) {
    llong res(1), t(a);
    while (b) {
        if (b & 1) res = res * t % n;
        t = t * t % n, b >>= 1;
    }
    return res;
}
const int N = 50003;
int mexp[N], id[N];
bool log_cmp(const int a, const int b) { return mexp[a] < mexp[b
    ];}

// a^x == b(mod p); p is prime and 1 <= a < p; x>=0
int mod_log(int a, int b, int p) {
    if (b == 1) return 0;
    int i, j, m = (int) ceil(sqrt(p)),inv = mod_pow(mod_pow(a, m, p
        ), p - 2, p);
    for (id[0] = 0, mexp[0] = i = 1; i < m; ++i) {
        id[i] = i; mexp[i] = mexp[i - 1]*(llong) a % p;
        if (mexp[i] == b) return i;
    }
    stable_sort(id, id + m, log_cmp);
    sort(mexp, mexp + m);
    for (i = 0; i < m; ++i) {
        j = lower_bound(mexp, mexp + m, b) - mexp;
        if (j < m && mexp[j] == b) return i * m + id[j];
        b = b * (llong) inv % p;
    }
    return -1;
}
```

$a^x = b(modn)$ a,b,n can be any integer.Baby-step-gaint-step

```
typedef long long llong;
const int N = (1<<14)-1, M = 40000;
//spoj 3105
struct Hash {
    int g[N], next[M], v[M], vu[M], ne;

    void init() {
        ne = 2; memset(g, 0, sizeof (g));
    }
    int find(int t) {
        for (int i = g[t&N]; i; i = next[i]) if (t == v[i]) return
            vu[i];
        return -1;
    }
    void insert(int t, int val) {
        int key = t&N;
        v[ne] = t;
        vu[ne] = val;
        next[ne] = g[key];
        g[key] = ne++;
    }
} S;

void extend_gcd(llong a, llong b, llong &d, llong &x, llong& y) {
    if (b) {
        extend_gcd(b, a % b, d, y, x);
        y -= a / b*x;
    } else d = a, x = 1, y = 0;
}
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
// a^x == b (mod n),n is not need to be prime;
int mod_log(int a, int b, int n) {
    b %= n, a %= n;
    llong t, x, y, d, r, res;
    int i, tmp;
    for (i = 0, t = 1 % n; i < 100; ++i, t = t * a % n) if (t == b)
        return i;
    for (r = 1, res = 0; (tmp = gcd(a, n)) > 1; ++res) {
        if (b % tmp) return -1;
        b /= tmp; n /= tmp; r = r * a / tmp % n;
    }
    S.init();
    extend_gcd(r, n, d, x, y);
    b = (b * x % n + n) % n;
    int s = (int) ceil(sqrt(n+0.0));
    for (i = 0, t = 1; i < s; ++i, t = t * a % n) {
        if (t == b) return i + res;
        if (S.find(t) == -1) S.insert(t, i);
        else return -1;
    }
    extend_gcd(t, n, d, x, y);
    x = (x % n + n) % n;
    for (i = 0; i < s; ++i) {
        tmp = S.find(b);
        if (tmp != -1) return i * s + res + tmp;
        b = b * x % n;
    }
    return -1;
}
```

6.2.3 $x * x == a(modp)$

```
typedef long long llong;
int mod_pow(int a, int b, int mod) {
    llong res(1), t(a);
    while (b) {
        if (b & 1) res = res * t % mod;
        t = t * t % mod; b >>= 1;
    }
    return res;
}
// x*x == a (mod n) n should be a prime and gcd(a,n) == 1
int mod_sqrt(int a, int n) {
    int res;
    if (2 == n) return a % n;
    if (1 == mod_pow(a, (n - 1) / 2, n)) {
        if (3 == n % 4) res = mod_pow(a, (n + 1) / 4, n);
        else {
            int b = 1, k = 0, i = (n - 1) / 2;
            while (1 == mod_pow(b, (n - 1) / 2, n)) ++b;
            do {
                i /= 2, k /= 2;
                if ((0 == mod_pow(a, i, n)*(llong)mod_pow(b, k, n)+1)
                        % n) k += (n-1)/2;
            } while (i % 2 == 0);
            res = (mod_pow(a, (i + 1) / 2, n)*(llong) mod_pow(b, k /
                    2, n)) % n;
        }
        return min(res, n - res); // make that res <= n/2
    }
    return -1;
}
int x, n;
int main() {
    while (cin >> x >> n) {
        cout << mod_sqrt(x, n) << endl;
    }
}
```

6.2.4 Miller and Pollard

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <ctime>
#include <cstdlib>


using namespace std;
typedef long long ll;

ll mult(ll a,ll b, ll mod) {
    if (a >= mod) a %= mod;
    if (b >= mod) b %= mod;
    if (a <= (1LL<<31) && b <= (1LL <<31)) return a*b%mod;

    ll res = 0;
    while (b) {
        if (b&1) {
            res += a;
            if (res >= mod) res -= mod;
        }
        a <<= 1;
        if (a >= mod) a -= mod;
        b >>= 1;
    }
    return res;
}

ll gcd(ll a, ll b) {
    return b? gcd(b,a%b):a;
}

int cnt1, cnt2, cnt0;
ll p_rho(ll n, int limit = 1 << 17) {
    if (0 == (n&1)) return 2; // must
    ll x, y, d;
    for (ll c = 1; c < n; ++c) {
        x = y = 2;
        while (true) {
            x = (mult(x,x,n) + c) % n;
            y = (mult(y,y,n) + c) % n;
            y = (mult(y,y,n) + c) % n;
            d = gcd(y - x + n, n);
            if (d == n) break;
            if (d > 1) return d;
            ++cnt1;
        }
    }
    return n;
}
/// here is the fast code.
ll val[1<<20];
ll p_rho0(ll n, int limit = 1 << 17) {
    if (0 == (n&1)) return 2; // must
    ll d;
    for (ll c = 1; c < n; ++c) {
        val[0] = 2;
        for (int i = 1; i < limit; ++i) {
            val[i] = (mult(val[i - 1], val[i - 1],n) + c)%n;
            if (0 == (i & 1)) {
                d = gcd(val[i] - val[i>>1] + n, n);
                if (d == n) break;
                if (d > 1) return d;
```

```
            }
            ++cnt0;
        }
    }
    return n;
}


ll p_rho2(ll n, int limit = 8192) {
    if (0 == (n&1)) return 2; // must
    ll x,y,d,i,k;
    for (ll c = 1; c < n; ++c) {
        i = 0;
        k = x = y = 2;
        while (i++ < limit) {
            x = (mult(x,x,n) + c) % n;
            d = gcd(x - y + n, n);
            if (d == n) break;
            if (d != n && d > 1) return d;
            if (i == k) y = x, k <<= 1;
            ++cnt2;
        }
    }
    return n;
}

ll power(ll a, ll b, ll mod) {
    ll res = 1, t = a;
    while (b) {
        if (b&1) res = mult(res,t,mod);
        t = mult(t,t,mod);
        b >>= 1;
    }
    return res;
}


ll witness(ll a, ll n) {
    ll b = n - 1;
    int cnt = 0;
    while (0 == (b&1)) ++cnt, b >>= 1;
    ll x = power(a,b,n), y;
    for (int i = 0; i < cnt; ++i) {
        y = mult(x,x,n);
        if(y == 1) {
            if(x != 1 && x != n - 1) return 0;
            return 1;
        }
        x = y;
    }
    return x;
}

bool is_prime(ll n) {
    if(n == 2) return true;
    if(n < 2 || (n&1) == 0) return false;
    int p[5] = {3,5,7,11,13};
    for (int i = 0; i < 5; ++i) {
        if (n == p[i]) return true;
        if (n % p[i] == 0) return false;
    }

    for (int i = 0; i < 10; ++i)
        if (witness(rand()%(n-2) + 2,n)!= 1) return false;
    return true;
}

int main() {
    ll n;
    ll sum0 = 0, sum1 = 0, sum2 = 0;
    for (int v = 0; v < 1000; ++v){
    srand(time(0));
    for (int i = 3; i < 1000; i+=2) {

        n = rand();
        if ( v % 5 != 0 ) n = n*n;
        if (is_prime(n)) {
            n = n*n;
            cnt0 = cnt1 = cnt2 = 0;
            if(p_rho0(n) == n) cout << "ERROR0" << " " << n << endl;
            if(p_rho(n) == n) cout << "ERROR1" << " " << n << endl;
            if(p_rho2(n) == n) cout << "ERROR2" << " " << n << endl;
            sum0 += cnt0;
            sum1 += 3*cnt1;
            sum2 += cnt2;
        } else {
            cnt0 = cnt1 = cnt2 = 0;
            if(p_rho0(n) == n) cout << "ERROR0" << " " << n << endl;
            if(p_rho(n) == n) cout << "ERROR1" << " " << n << endl;
            if(p_rho2(n) == n) cout << "ERROR2" << " " << n << endl;
            sum0 += cnt0;
            sum1 += 3*cnt1;
            sum2 += cnt2;
        }
    }
    }
    cout << "over" << endl;
    cout << sum0 <<"\n"<<sum1 <<"\n" << sum2 << endl;
}
```

6.2.5 Get prime in range

```
//spoj 8586. Prime Generator2
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;
const int N = 46340;
bool vis[N];
int prime[5000], np;

void cp() {
    memset(vis, 0, sizeof (vis));
    np = 0;
    int t;
    for (int i = 2; i < N; ++i) {
        if (!vis[i]) prime[np++] = i;
        for (int j = 0; j < np && (t = i * prime[j]) < N; ++j) {
            vis[t] = 1;
            if (i % prime[j] == 0) break;
        }
    }
    //printf("%d\n",np);
}
int a, b;

int solve() {
    int ret = 0, k, t;
    memset(vis, 0, sizeof (vis));
    if (2 >= a && 2 <= b) puts("2");// note that
    if ((a & 1) == 0) a++;
    for (int i = 1; i < np && prime[i] <= b; ++i) {
        if (a % prime[i] == 0) k = a;
        else {
            if (a <= b - (prime[i]-(a % prime[i]))) {
                k = a + prime[i]-(a % prime[i]);
                if ((k & 1) == 0) {
                    if (k <= b - prime[i])
                        k += prime[i];
                    else continue;
                }
            } else continue;
        }
        if (k != prime[i] && k <= b) vis[k - a] = 1;
        t = prime[i]*2;
        int j = k;
        while (j <= b - t) {
            j += t;
            vis[j - a] = 1;
        }
    }
    for (int i = 0; i <= b - a; i += 2)
        if (!vis[i] && i + a >= 2) printf("%d\n", a + i);
}

int main() {
    cp();
    int cas;
    bool flag = false;
    for (scanf("%d", &cas); cas; --cas) {
        if(flag) puts("");
        else flag = true;
        scanf("%d%d", &a, &b);
        solve();
    }
}
```

6.2.6 Mod Equation

```
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;

// #define __int64 long long
void eGcd(__int64 a, __int64 b, __int64 &d, __int64 &x, __int64 &y
    ) {
    if (!b) {
        x = 1;
        y = 0;
        d = a;
    } else {
        eGcd(b, a % b, d, y, x);
        y -= a / b*x;
    }
}

__int64 getAns(__int64* m, __int64* r, int n) {
    __int64 ret = r[0], mod = m[0], t, te, x, y, d;
    for (int i = 1; i < n; ++i) {
        t = r[i] - ret;
        eGcd(mod, m[i], d, x, y);
        if (t % d) return -1;
        te = m[i] / d;
        x = (t / d * x % te + te) % te;
        ret += mod*x;
        mod *= te;
    }
    return ret;
}

int n;
__int64 m[10000], r[10000];

int main() {
    while (scanf("%d", &n) == 1) {
        for (int i = 0; i < n; ++i)
            scanf("%I64d%I64d", &m[i], &r[i]);
        printf("%I64d\n", getAns(m, r, n));
    }
    return 0;
}
```

6.3 Fraction

6.3.1 $a/b < x/y < c/d$

```
//smallest denominator
typedef long long ll;
void max_fac(int a,int b,int c,int d,int &x, int &y) {
    int t = a/b;
    if ( (t + 1)*(ll)d < c) {
```

```
        x = t + 1, y = 1;
    } else {
        max_fac(d,c-t*d,b,a-t*b,y,x);
        x += t*y;
    }
}
```

6.3.2 $x^2 - n * y^2 = 1$

n is a non-squre-number, solve the minimum $(x1, y1)$
all (x_i, y_i) satisfies:

$$x_i + y_i\sqrt{n} = (x_1 + y_1\sqrt{n})^i$$

$$x_{i+1} = x_1x_i + ny_1y_i$$

$$y_{i+1} = x_1y_i + y_1x_i$$

```
//always need BigInteger
typedef long long ll;
void getAns(ll &x, ll &y, int n) {
    ll p0 = 0, p1 = 1, p2;
    ll q0 = 1, q1 = 0, q2;
    ll g1 = 0, h1 = 1, g2, h2;
    ll a0 = (int)(sqrt(n+0.5)), a2 = a0, a3;

    for (int i = 2;; ++i) {
        g2 = a2*h1 - g1;
        h2 = (n - g2*g2)/h1;
        a3 = (g2+a0)/h2;
        p2 = a2*p1+p0;
        q2 = a2*q1+q0;
        if (p2*p2-n*q2*q2 == 1) {
            x = p2;
            y = q2;
            return ;
        }
        g1 = g2, h1 = h2, a2 = a3;
        p0 = p1, p1 = p2;
        q0 = q1, q1 = q2;
    }
}

/*
    static BigInteger x, y;

  public static void getAns(int n)
  {
    BigInteger p0 = BigInteger.ZERO, p1 = BigInteger.ONE, p2;
    BigInteger q0 = BigInteger.ONE, q1 = BigInteger.ZERO, q2;
    BigInteger g1 = BigInteger.ZERO, h1 = BigInteger.ONE, g2, h2;
    BigInteger a0 = BigInteger.valueOf((int)(Math.sqrt(n + 0.5))),
           a2 = a0, a3;
    BigInteger bn = BigInteger.valueOf(n);
    while (true)
    {
      g2 = a2.multiply(h1).subtract(g1);
      h2 = (bn.subtract(g2.multiply(g2))).divide(h1);
      a3 = (g2.add(a0)).divide(h2);
      p2 = a2.multiply(p1).add(p0);
      q2 = a2.multiply(q1).add(q0);
      if(p2.multiply(p2).subtract(bn.multiply(q2).multiply(q2)).
            equals(BigInteger.ONE)){ // notice use equals!!!!!
        x = p2;
        y = q2;
        return ;
      }

      g1 = g2; h1 = h2; a2 = a3;
      p0 = p1; p1 = p2;
      q0 = q1; q1 = q2;
    }
  }
*/
```

6.3.3 $\sum_{k=0}^{n-1}\lfloor(a + d * k)/m\rfloor$

```
typedef long long ll;
ll rec(ll a, ll d, ll m, ll n) {
    ll res = 0;
    if (a >= m) {
        res += (a/m)*n;
        a %= m;
    }
    if (d >= m) {
        res += (d/m)*(n*(n-1)/2);
        d %= m;
    }

    if (d == 0) return res;
    ll top = a + d*n;
    return res + rec(top%m, m, d, top/m);
}
```

6.4 Linear Equaton

6.4.1 Xor Equation

1 1 0 0 = 0 (here when r = 0, i = 0)
0 0 1 0 = 1 (here when r = 1, i = 2)
0 0 0 1 = 1 (here when r = 2, i = 3)

```
const int N = 33;
int m,nn,num,list[N];
// m equations, nn variables
int a[N][N];

int reduce(){
    int i,j,k,r;
    for (i = r = 0; i < nn; ++i){
        for (j = r; j < m && !a[j][i]; ++j); if (j >= m) continue;
        if (j > r) for (k = 0; k <= nn; ++k) swap(a[r][k],a[j][k]);
        for (num = 0, k = i; k <= nn; ++k) if (a[r][k]) list[num++]
              = k;
        for (j = 0; j < m; ++j) if (j != r && a[j][i])
            for (k = 0; k < num; ++k) a[j][list[k]] ^= 1;
        ++r;
    }
    for (i = 0; i < m; ++i)
        if (a[i][nn]) {
            for(j = 0; i < nn && !a[i][j]; ++j);
            if(j == nn) return 0; // else x[j] = a[i][nn]/a[i][j];
        }
    // the number of free variable = (nn - r)
    return 1;
}
```

be carefully when use long long and int

```
const int N = 33;
ll a[N];
int m,nn; // nn variables(0 -> nn -1), m equations
inline int bit2(ll v,int n){ return v >> n & 1;};
inline void set_bit(ll &v,int n,ll t) { v |= t << n;} // notice v
      will be ll
int reduce(){
    int i,j,r;
    for (i=r=0; i < nn; ++i){
        for(j=r; j < m && !bit2(a[j],i); ++j); if (j >= m) continue;
        if (j > r) swap(a[r],a[j]);
        for (int j = 0; j < m; ++j) if(j!=r && bit2(a[j],i)) a[j] ^=
              a[r];
        ++r;
    }
    for (i = 0; i < m; ++i) if(bit2(a[i],nn)) {
          if (a[i]^(1LL << nn)); // notice 1LL !!!
          //x[j] = bit2(a[i],nn)/ bit2(a[i],j); j's the first 1
                _value bit
          else return 0;
       }
    return 1;
}
```

6.4.2 Equation in Z

if in Q , change integer to fracton and no clean()

```
#include <iostream>
#include <algorithm>
using namespace std;

#define rep(i,a,b) for(int i=a;i<b;++i)
typedef long long ll;
const int N = 101;

/* reduce() == 1 has integer value , they are A[i][cols]
         == 2 has free value,the number is nfree
         == 0 no value
         == -1 has fractional value
 */
struct Equation {
    int rows, cols, nfree, frac, A[N][N];

    void init(){
        memset(A,0,sizeof(A));
    }

    inline int gcd(int a, int b) {
        return b ? gcd(b, a % b) : a;
    }

    inline void clean(int *b) {
        int g = 0;
        rep(i, 0, cols + 1) if (b[i])
            if (g) g = gcd(b[i], g);
            else g = b[i];
        if (g) rep(i, 0, cols + 1) b[i] /= g;
    }

    inline void swapRow(int a, int b) {
        if (a == b) return;
        rep(i, 0, cols + 1) swap(A[a][i], A[b][i]);
    }

    inline void pivot(int r, int c) {
        int u, v;
        if (A[r][c] == 0) exit(-1);
        rep(i, 0, rows) if (i != r && A[i][c]) {
            v = gcd(A[i][c], A[r][c]);
            u = A[r][c] / v;
```

```
            v = A[i][c] / v;
            rep(j, 0, cols + 1) A[i][j] = A[i][j] * u - v * A[r][j];
            clean(A[i]);
        }
    }

    int reduce(int nrow, int ncol) {
        rows = nrow;
        cols = ncol;
        nfree = frac = 0;
        int r = 0, c = 0, ind = 0;
        for (; c < cols; ++c, ++r) {
            for (ind = r; ind < rows && !A[ind][c]; ++ind);
            if (ind >= rows) {
                --r;
                ++nfree;
                continue;
            }
            swapRow(r, ind);
            pivot(r, c);
            // this->print();
        }
        for (int i = r; i < rows; ++i) if (A[i][cols]) return 0;
        if (nfree) return 2;
        for (r = 0; r < rows; ++r) if (A[r][cols]) {
                for (c = 0; c < cols && !A[r][c]; ++c);
                if (c == cols) return 0;
                if (A[r][cols] % A[r][c]) frac = 1;
                if (!frac) A[r][cols] /= A[r][c]; // get the answer
            }
        if (frac) return -1;
        return 1;
    }

    void print() {
        rep(i, 0, rows) {
            rep(j, 0, cols + 1) cout << A[i][j] << " ";
            cout << endl;
        }
        cout << endl;
    }
};

Equation test;

int main() {
    int n, m, t;
    while (cin >> n >> m) {
        rep(i, 0, n) {
            rep(j, 0, m + 1) cin >> test.A[i][j];
        }
        cout << test.reduce(n, m) << endl;
        test.print();
    }
}
```

6.4.3 Equation in R

```
#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

#define rep(i,a,b) for(int i=a;i<b;++i)
#define TEST freopen("in","r",stdin);
typedef long long ll;
const int N = 101;
const double eps = 1e-8;

/* reduce() == 0 has no value
        == 1 has values,they are A[i][cols]
 */


struct Equation {
    double A[N][N];
    int rows, cols, id[N];

    void init(){
        memset(A,0,sizeof(A));
    }

    inline bool zero(double x) {
        return fabs(x) < eps;
    }

    inline void swapA(int r, int c, int ir, int ic) {
        if (r != ir)
            rep(i, 0, cols + 1) swap(A[r][i], A[ir][i]);
        if (c != ic) {
            rep(i, 0, rows) swap(A[i][c], A[i][ic]);
            swap(id[c], id[ic]);
        }
    }

    inline void pivot(int r, int c) {
        if (fabs(A[r][c]) < eps) exit(-1);
        double p;
        rep(i, 0, rows) if (i != r && fabs(A[i][c]) > eps) {
            p = A[i][c] / A[r][c];
            rep(j, 0, cols + 1) A[i][j] -= p * A[r][j];
        }
    }

    int reduce(int nrow, int ncol) {
        rows = nrow;
        cols = ncol;
        rep(i, 0, cols) id[i] = i;
        int r = 0, c = 0, indr = 0, indc = 0;
        double maxp = 0;
        for (; c < cols; ++c, ++r) {
            maxp = 0;
            rep(i, r, rows) rep(j, c, cols) if (fabs(A[i][j]) > fabs(
                    maxp)) {
                maxp = A[i][j];
                indr = i;
                indc = j;
            }
            if (zero(maxp)) return 0;
            swapA(r, c, indr, indc);
            pivot(r, c);
        }
        //this->print();
        rep(i, 0, cols) A[i][0] = A[i][cols] / A[i][i];
        rep(i, 0, cols) A[id[i]][cols] = A[i][0];
        return 1;
    }

    void print() {
        rep(i, 0, rows) {
            rep(j, 0, cols + 1) cout << A[i][j] << " ";
            cout << endl;
        }
        cout << endl;
    }
};

Equation test;

int main() {
    int n, m, t;
    TEST
    while (cin >> n >> m) {
        rep(i, 0, n) {
            rep(j, 0, m + 1) cin >> test.A[i][j];
        }
        cout << test.reduce(n, m) << endl;
        test.print();

        rep(i, 0, m) cout << test.A[i][m] << endl;
        cout << "Over" << endl;
    }
}
```

6.4.4 det

```
锘緌
l det(ll a[][N],int n) {
    for(int i=0; i<n; i++)for(int j=0; j<n; j++)a[i][j]%=mod;
    ll ret=1;
    for(int i=1; i<n; i++) {
        for(int j=i+1; j<n; j++)
            while(a[j][i]) {
                ll t=a[i][i]/a[j][i];
                for(int k=i; k<n; k++)
                    a[i][k]=(a[i][k]-a[j][k]*t)%mod;
                for(int k=i; k<n; k++)
                    swap(a[i][k],a[j][k]);
                ret=-ret;
            }
        if(a[i][i]==0)return 0;
        ret=ret*a[i][i]%mod;
    }
    return (ret+mod)%mod;
}
```

6.5 Anti-Nim

Anti-Nim: $res = \oplus_i sg(i)$
$cnt = \sum_i [sg(i) <= 1]$
first player wins when ($res = 0$ and $cnt = n$) $\|$ ($res \neq 0$ and $cnt \neq n$)

6.6 nim multiply

$x \otimes y = \text{mex}\{(x \otimes a) \oplus (b \otimes y) \oplus (a \otimes b) | 0 \leq a < x, 0 \leq b < y\}$

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <vector>
#include <map>
#include <algorithm>
#include <set>
using namespace std;
typedef long long ll;


const int N = 1010;
const int M = (1<<8);

bool vis[1<<8];
int sgv[1<<4][1<<4];
int sg(int x,int y) {
    if (x == 0 || y == 0) return 0;
    if (sgv[x][y] != -1) return sgv[x][y];

    memset(vis, 0 , sizeof (vis));
    for (int i = 0; i < x; ++i) {
        vis[sg(i,y)] = 1;
    }

```

```
    for (int i = 0; i < y; ++i) {
       vis[sg(x,i)] = 1;
    }

    for (int xx = 1; xx < x; ++xx) {
       for (int yy = 1; yy < y; ++yy) {
          vis[sg(xx,y)^sg(x,yy)^sg(xx,yy)] = 1;
       }
    }

    for (int i = 0; i < M; ++i) if (!vis[i]) return i;
    return M;
}




void init() {
    memset(sgv, -1, sizeof(sgv));
    for (int i = 0; i < 16; ++i) {
       for (int j = 0; j < 16; ++j) {
          sgv[i][j] = sg(i,j);
       }
    }
}



int nim_mult_power(int x,int y) { // x is a power of 2
    //cout << x<<" " << y << endl;
    if (x < 16) return sg(x,y);
    int a, m , p, s, t, d1, d2;
    for (a = 1; (1LL << a) <= x ; a <<= 1);
    a >>= 1, m = (1<<a);
    p = x/m;
    s = y/m, t = y&(m-1);
    d1 = nim_mult_power(p,s);
    d2 = nim_mult_power(p,t);
    return ((d1^d2) << a) ^ nim_mult_power(m/2,d1);
}

int nim_mult(int x,int y) {
    if (x < y) swap(x,y);
    if (x < 16) return sg(x,y);
    int a, m, p, q, s, t, c1, c2, c3;
    for (a = 1; (1LL << a) <= x; a <<= 1);
    a >>= 1, m = (1<<a);
    p = x/m, q = x&(m-1);
    s = y/m, t = y&(m-1);
    c1 = nim_mult(p,s);
    c2 = nim_mult(p,t)^nim_mult(q,s);
    c3 = nim_mult(q,t);
    return ((c1^c2) << a)^c3^nim_mult_power(m/2,c1);
}

int n;
int main() {
    init();
    //test();
    int cas;
    for (cin >> cas; cas; --cas) {
       scanf("%d",&n);
       int res = 0, x, y;

       for (int i = 0; i < n; ++i) {
          scanf("%d%d",&x,&y);
          res ^= nim_mult(x,y);
       }
       if (res) puts("Have a try, lxhgww.");
       else puts("Don't waste your time.");
    }
}
```

6.7 FFT

```
// hdu 1402
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#define N 300005
#define pi acos(-1.0)
using namespace std;

struct Complex {
    double r, i;

    Complex(double real = 0.0, double image = 0.0) {
       r = real;
       i = image;
    }

    Complex operator +(const Complex o) {
       return Complex(r + o.r, i + o.i);
    }

    Complex operator -(const Complex o) {
       return Complex(r - o.r, i - o.i);
    }

    Complex operator *(const Complex o) {
       return Complex(r * o.r - i * o.i, r * o.i + i * o.r);
    }

    void setValue(double real = 0.0, double image = 0.0) {
       r = real;
       i = image;
    }
} xa[N], xb[N];

void brc(Complex *y, int len) {
    register int i, j, k;
    for (i = 1, j = len >> 1; i < len - 1; ++i) {
       if (i < j) swap(y[i], y[j]);

       k = len >> 1;
       while (j >= k) {
          j -= k;
          k >>= 1;
       }
       if (j < k) j += k;
    }
}

void fft(Complex *y, int len, double on) // FFT O(nlogn)
// if on==1 DFT if on==-1 DFT
{
    register int h, hh, i, j, k;
    Complex w ,u, t, wn;
    brc(y, len);
    for (h = 1, hh = 2; hh <= len; h = hh, hh <<= 1) {
       wn.setValue(cos(on * pi / h), sin(on * pi / h));
       for (j = 0; j < len; j += hh) {
          w.setValue(1, 0);
          for (k = j; k < j + h; k++) {
             u = y[k];
             t = w * y[k + h];
             y[k] = u + t;
             y[k + h] = u - t;
             w = w*wn;
          }
       }
    }
    if (on == -1) for (i = 0; i < len; ++i) y[i].r /= len;
}

void multi(char* a, char* b, int* sum, int &len) {
    int la, lb, i;
    la = strlen(a);
    lb = strlen(b);
    len = 1;
    while (len < la * 2 || len < lb * 2) len <<= 1;
    for (i = 0; i < len; ++i) {
       xa[i].r = (i < la) ? a[la - i - 1] - '0' : 0.0;
       xb[i].r = (i < lb) ? b[lb - i - 1] - '0' : 0.0;
       xa[i].i = xb[i].i = 0.0;
    }
    fft(xa, len, 1); // DFT(a)
    fft(xb, len, 1); // DFT(b)
    for (i = 0; i < len; ++i) xa[i] = xa[i] * xb[i]; // a = a*b
    fft(xa, len, -1); // IDFT(a*b)
    for (i = 0; i < len; ++i) sum[i] = (int) (xa[i].r + 0.5); //
          sum = a
    for (i = 0; i < len; ++i) //
    {
       sum[i + 1] += sum[i] / 10;
       sum[i] %= 10;
    }
    len = la + lb - 1;
    while (sum[len] <= 0 && len > 0) --len;
}

char a[N / 2], b[N / 2];
int sum[N]; // result

int main(void) {
    int l;
    register int i;
    while (scanf("%s%s", a, b) == 2) {
       multi(a, b, sum, l);
       for (int i = l; i >= 0; i--) putchar(sum[i] + '0');
       putchar('\n');
    }
    return 0;
}
```

6.8 Popynomial

```
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <cstring>

int flag = 0;
typedef long long typec;
typec MOD = 999983;
const int RANKLIM = 50;
const int EQLIM = 10;

typec gcd(typec a, typec b) {
    return b ? gcd(b, a % b) : a;
}

typec extendGCD(typec a, typec b, typec& x, typec& y) {
    if (!b) return x = 1, y = 0, a;
    typec res = extendGCD(b, a % b, x, y), tmp = x;
    x = y, y = tmp - (a / b) * y;
    return res;
}
///ax = b mod m, be sure that (b, m) = (a, m)

typec modEquation(typec a, typec b, typec m) {
    typec x, y;
    y = extendGCD(a, m, x, y);
    while (x < 0) x += m;
    return (x * (b / y)) % m;
}

class Polynomial {
public:
    Polynomial();
    void set(int ra, typec* coe);
    void simplify();

    int getRank() const {
       return rank;
    }
```

```
    typec getValue(typec x) const;

    bool isOne() const {
        return rank == 0 && coefficient[0];
    }

    bool isZero() const {
        return rank == 0 && coefficient[0] == 0;
    }
    bool operator==(const Polynomial& a) const;

    bool operator!=(const Polynomial& a) const {
        return !(*this == a);
    }

    bool rankEqual(const Polynomial& a) const {
        return rank == a.rank;
    }

    bool operator>(const Polynomial& a) const {
        return rank > a.rank;
    }

    bool operator<(const Polynomial& a) const {
        return rank < a.rank;
    }

    bool operator>=(const Polynomial& a) const {
        return rank >= a.rank;
    }

    bool operator<=(const Polynomial& a) const {
        return rank <= a.rank;
    }
    Polynomial & operator+=(const Polynomial&);
    Polynomial operator+(const Polynomial&) const;
    Polynomial & operator-=(const Polynomial&);
    Polynomial operator-(const Polynomial&) const;
    Polynomial operator*(const Polynomial&) const;

    Polynomial & operator*=(typec ti) {
        for (int i = 0; i <= rank; i++) coefficient[i] *= ti;
        return *this;
    }

    Polynomial operator*(typec ti) const {
        Polynomial res = *this;
        return res *= ti;
    }
    Polynomial& mulAtRank(typec ti, int ra);
    Polynomial nMulAtRank(typec ti, int ra) const;
    Polynomial & operator%=(const Polynomial&);
    Polynomial operator%(const Polynomial&) const;
    void print() const;
    void print2() const;

private:
    typec coefficient[RANKLIM + 1];
    int rank;
};

Polynomial::Polynomial() : rank(0) {
    for (int i = 0; i < RANKLIM + 1; i++)
        coefficient[i] = 0;
}

void Polynomial::set(int ra, typec* coe) {
    for (int i = 0; i <= rank; i++)
        coefficient[i] = 0;
    rank = ra;
    for (int i = 0; i <= rank; i++)
        coefficient[ra - i] = (coe[i] % MOD + MOD) % MOD;
    while (rank > MOD - 2) {
        coefficient[rank % (MOD - 1)] += coefficient[rank];
        coefficient[rank % (MOD - 1)] %= MOD;
        coefficient[rank--] = 0;
    }
    while (coefficient[rank] == 0 && rank) rank--;
}

void Polynomial::simplify() {
    typec g = coefficient[0];
    if (g == 0) return;
    for (int i = 1; i <= rank; i++)
        g = gcd(g, coefficient[i]);
    for (int i = 0; i <= rank; i++)
        coefficient[i] /= g;
}

typec Polynomial::getValue(typec x) const {
    typec res = 0;
    for (int i = rank; i >= 0; i++)
        res *= x, res += coefficient[i], res %= MOD;
    return res;
}

bool Polynomial::operator==(const Polynomial& a) const {
    if (rank != a.rank) return false;
    for (int i = 0; i <= rank; i++)
        if (coefficient[i] != a.coefficient[i])
            return false;
    return true;
}

Polynomial& Polynomial::operator+=(const Polynomial& a) {
    if (a.rank > rank) rank = a.rank;
    for (int i = 0; i <= rank; i++)
        coefficient[i] += a.coefficient[i], coefficient[i] %= MOD;
    while (coefficient[rank] == 0 && rank) rank--;
    return *this;
}

Polynomial Polynomial::operator+(const Polynomial& a) const {
    Polynomial res = *this;
    return res += a;
}

Polynomial& Polynomial::operator-=(const Polynomial& a) {
    if (a.rank > rank) rank = a.rank;
    for (int i = rank; i >= 0; i--) {
        coefficient[i] -= a.coefficient[i];
        while (coefficient[i] < 0)
            coefficient[i] += MOD;
    }
    while (coefficient[rank] == 0 && rank) rank--;
    return *this;
}

Polynomial Polynomial::operator-(const Polynomial& a) const {
    Polynomial res = *this;
    return res -= a;
}

Polynomial Polynomial::operator*(const Polynomial& a) const {
    Polynomial res;
    res.rank = rank + a.rank;
    if (res.rank > MOD - 2) res.rank = MOD - 2;
    for (int i = 0; i <= rank; i++)
        for (int j = 0; j <= a.rank; j++) {
            res.coefficient[(i + j) % (MOD - 1)] += coefficient[i] *
                    a.coefficient[j];
            res.coefficient[(i + j) % (MOD - 1)] %= MOD;
        }
    while (res.coefficient[res.rank] == 0 && res.rank) res.rank--;
    return res;
}

Polynomial& Polynomial::mulAtRank(typec ti, int ra) {
    for (int i = 0; i <= rank; i++)
        coefficient[rank + ra - i] = (coefficient[rank - i] * ti) %
                MOD;
    for (int i = 0; i < ra; i++)
        coefficient[i] = 0;
    rank += ra;
    if (ti == 0) rank = 0;
    while (rank >= MOD - 1) {
        coefficient[rank % (MOD - 1)] += coefficient[rank];
        coefficient[rank % (MOD - 1)] %= MOD;
        coefficient[rank--] = 0;
    }
    return *this;
}

Polynomial Polynomial::nMulAtRank(typec ti, int ra) const {
    Polynomial res = *this;
    return res.mulAtRank(ti, ra);
}

Polynomial& Polynomial::operator%=(const Polynomial& divisor) {
    typec ti;
    while (*this >= divisor && !(this->isZero())) {
        ti = modEquation(divisor.coefficient[divisor.rank],
                coefficient[rank], MOD);
        *this -= divisor.nMulAtRank(ti, rank - divisor.rank);
    }
    return *this;
}

Polynomial Polynomial::operator%(const Polynomial& divisor) const
        {
    Polynomial res = *this;
    return res %= divisor;
}

void Polynomial::print() const {
    for (int i = rank; i >= 0; i--) {
        if (coefficient[i] == 0) continue;
        if (i != rank) printf(" + ");
        if (!i || coefficient[i] != 1)
            printf("%lld", coefficient[i]);
        if (i) printf("x");
        if (i > 1) printf("^%d", i);
    }
    printf("\n");
}

void Polynomial::print2() const {
    printf("%d", rank);
    for (int i = rank; i >= 0; i--)
        printf(" %lld", coefficient[i]);
    printf("\n");
}

Polynomial Pgcd(Polynomial a, Polynomial b) {
    Polynomial swap, zero;
    while (b != zero) {
        a %= b;
        swap = b, b = a, a = swap;
    }
    return a;
}


int main() {
    int n, ra;
    typec co[100];
    Polynomial pol[100], g;
    while (scanf("%d", &n) != EOF) {
        if (flag) break;
        for (int i = 0; i < n; i++) {
            scanf("%d", &ra);
            for (int j = 0; j <= ra; j++)
                scanf("%I64d", co + j);
            pol[i].set(ra, co);
        }
        g = pol[0];
        for (int i = 1; i < n; i++) {
            g = Pgcd(pol[i], g);
            if (g.isOne()) break;
        }
        g.simplify();
        if (g.isOne()) printf("NO\n");
        else printf("YES\n");
    }
    return 0;
}
```