Team Reference Document

Encore @ Harbin Institute of Technology

June 26, 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, E = V*V*2, 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

```
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] = 1;
   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

```cpp
/* 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], 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

```cpp
#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

```cpp
#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

```cpp
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) // 1
      rotate(y, 1), rotate(x, 1);
     else // z
      rotate(x, 0), rotate(x, 1);
    } else {
     if (y->ch[1] == x) // 1
      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;
```

3.7 SplitTree

```
#define TREE(x) root[belong[x]],1,1,length[belong[x]]
#define L seg, id << 1, l, mid
#define R seg, (id<<1)+1, mid+1,r
const int N = 10011, INF = 0x3f3f3f3f;
struct G {
   int v[N << 1], next[N << 1], w[N << 1], id[N << 1], adj[N], ne;
   void init() {
      ne = 2;
      memset(adj, 0, sizeof (adj));
   }
   void addEdge(int a, int b, int c, int d) {
      v[ne] = b;
      w[ne] = c;
      id[ne] = d;
      next[ne] = adj[a];
      adj[a] = ne++;
   }
} g;
class LCA {
private:
   G* g;
   int dfn[N], pos[N], st[20][N << 1], len;
   void dfs(int cur, int p, int d) {
      pos[cur] = len;
      dfn[cur] = d;
      st[0][len++] = cur;
      for (int i = g->adj[cur]; i; i = g->next[i])
         if (g->v[i] != p) {
            dfs(g->v[i], cur, d + 1);
            st[0][len++] = cur;
         }
   }
   void initRMQ() {
      for (int i = 1, k = 2; k < len; ++i, k <<= 1)
         for (int j = 0; j < len; ++j) {
            int sk = j + (k >> 1);
            st[i][j] = st[i - 1][j];
            if (sk < len && dfn[st[i][j]] > dfn[st[i - 1][sk]])
               st[i][j] = st[i - 1][sk];
         }
   }
public:
   void init(G &graph, int root) {
      g = &graph;
      len = 0;
      dfs(root, -1, 0);
      initRMQ();
   }
   int query(int x, int y) {
      x = pos[x], y = pos[y];
      if (x > y) swap(x, y);
      int len = y - x + 1, bl;
      for (bl = -1; len != 0; ++bl, len >>= 1);
      x = st[bl][x];
      y = st[bl][y - (1 << bl) + 1];
      return dfn[x] < dfn[y] ? x : y;
   }
} lca;
int fa[N], size[N], idv[N], wgt[N];
void dfs(int cur) {
   size[cur] = 1;
   for (int i = g.adj[cur]; i; i = g.next[i])
      if (g.v[i] != fa[cur]) {
         fa[g.v[i]] = cur;
         wgt[g.v[i]] = g.w[i];
         idv[g.id[i]] = g.v[i];
         dfs(g.v[i]);
         size[cur] += size[g.v[i]]; // notice that
      }
}
// begin segment tree
int segt[N << 2], length[N], *root[N], *ne;
int* newSegTree(int len) {
   int *res = ne;
   ne += (len << 2);
   return res;
}
int ar[N], belong[N], pos[N];
void build(int* seg, int id, int l, int r) {
   if (l == r) seg[id] = wgt[ar[l]];
   else {
      int mid = l + r >> 1;
      build(L);
      build(R);
      seg[id] = max(seg[id << 1], seg[(id << 1) + 1]);
   }
}
void insert(int* seg, int id, int l, int r, int x, int t) {
   if (l == r) seg[id] = t;
   else {
      int mid = l + r >> 1;
      if (x <= mid) insert(L, x, t);
      else insert(R, x, t);
      seg[id] = max(seg[id << 1], seg[(id << 1) + 1]);
   }
}
int getMax(int *seg, int id, int l, int r, int ll, int rr) {
   if (ll <= l && r <= rr) return seg[id];
   else {
      int mid = l + r >> 1;
      int res = -INF;
      if (ll <= mid) res = max(res, getMax(L, ll, rr));
      if (rr > mid) res = max(res, getMax(R, ll, rr));
      return res;
   }
}
// end segment tree
void createTree(int cur, int len) {
   ar[len] = cur;
   int maxsize = 0, next = 0;
   for (int i = g.adj[cur]; i; i = g.next[i])
      if (g.v[i] != fa[cur] && size[g.v[i]] > maxsize)
         maxsize = size[g.v[i]], next = g.v[i];
   if (next) {
      createTree(next, len + 1);
      for (int i = g.adj[cur]; i; i = g.next[i])
         if (g.v[i] != fa[cur] && g.v[i] != next)
```

```
            createTree(g.v[i], 1);
   } else {
      int p = cur;
      for (int i = 1; i < len; ++i) p = fa[p];
      root[p] = newSegTree(len);
      length[p] = len;
      build(root[p], 1, 1, len);
      for (int i = 1; i <= len; ++i) {
         belong[ar[i]] = p;
         pos[ar[i]] = i;
      }
   }
}
void buildTree() {
   lca.init(g, 1);
   fa[1] = -1;
   wgt[1] = -INF;
   dfs(1);
   ne = segt;
   createTree(1, 1);
}
int query(int a, int b) {
   int res = -INF, t;
   while (a != b) {
      if (belong[a] == belong[b]) {
         t = getMax(TREE(b), pos[a] + 1, pos[b]);
         if (t > res) res = t;
         break;
      } else {
         t = getMax(TREE(b), 1, pos[b]);
         if (t > res) res = t;
         b = fa[belong[b]];
      }
   }
   return res;
}
int main() {
   int n,cas;
   for (cin >> cas; cas; --cas) {
      g.init(); scanf("%d", &n);
      int a, b, c;
      rep(i, 1, n) {
         scanf("%d%d%d", &a, &b, &c);
         g.addEdge(a, b, c, i);
         g.addEdge(b, a, c, i);
      }
      buildTree(); char s[20];
      while (scanf("%s", s), s[0] != ’D’) {
         scanf("%d%d", &a, &b);
         if (s[0] == ’Q’) {
            c = lca.query(a, b);
            printf("%d\n", max(query(c, a), query(c, b)));
         } else {
            int v = idv[a]; // notice that !!!
            insert(TREE(v), pos[v], b);
         }
      }
   }
}
```

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 Eular Circles

```
锘縞
onst int SIZE = 2 * 2000, N = 50;
/* Eular degree & connection
 * fordown 锛宨paths the smallest lexicographic path
 * hoj 1045 John's trip*/
struct edge {
   int v, id;
   bool operator<(const edge a) const {
      return id < a.id;
   }
} edges[SIZE];
vector<edge> adj[N];
int path[SIZE]. E, V, S, deg[N], stp;
bool vst[SIZE];
void dfs(int now) {
   edge tmp;
   for (size_t i = 0; i < adj[now].size(); i++) {
      tmp = adj[now][i];
      if (!vst[tmp.id] && !vst[tmp.id]) {
         vst[tmp.id] = vst[tmp.id] = 1;
         dfs(tmp.v);
         path[stp++] = tmp.id;
      }
   }
}
void solve() { {
   for (int i = 0; i < V; i++)
      sort(adj[i].begin(), adj[i].end());
   dfs(S);
   printf("%d", path[stp - 1]);
   for (int i = stp - 2; i >= 0; i--)
      printf(" %d", path[i]);
   putchar('\n');
}
void initialize(int u, int v) {
   stp = V = E = S = 0;
   for (int i = 0; i < N; i++) adj[i].clear();
   memset(vst, false, sizeof (vst));
   memset(deg, 0, sizeof (deg));
   S = min(v, u);
}
void add_edge(int u, int v, int id, int E) {
   deg[u]++, deg[v]++;
   edges[E].v = v, edges[E].id = id;
   adj[u].push_back(edges[E]);
   edges[E].v = u, edges[E].id = id;
   adj[v].push_back(edges[E]);
}
```

4.11 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));
}
```

4.12 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;
}
```

4.13 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;
}
```

4.14 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 Dynamic Programing

5.1 Mask DP I

```
const int N = 10, TOT = 1000;
const int MAXN = 1594323;// 3^13
char maps[N][N];
int bit3[N] = {1}, status[TOT], Hash[MAXN], allS = 0;
LL dp[2][TOT];
bool check(int s) {
 int cnt = 0;
 while(s) {
  int n = s % 3;
  if(n == 1) cnt++;
  if(n == 2) cnt--;
  if(cnt < 0) return false;
  s /= 3;
 }
 return (cnt == 0);
}
int getbit(int s, int i) {
 return s / bit3[i] % 3;
}
void transfer(LL& dest, LL add) {
 dest == -1 ? (dest = add) : (dest += add);
}
LL DP(int n, int m) {
 int now = 0, pre = 1, state = (1 + 2 * bit3[m - 1]) * 3;
 CC(dp, 0), dp[0][0] = 1;
 for(int i = 0; i < n; i++) {
  for(int j = 0; j < m; j++) {
   swap(now, pre);CC(dp[now], 0);
   for(int k = 0, s; s = status[k], s < bit3[m + 1]; k++) {
    if(dp[pre][k] == 0) continue;
    int l = getbit(s, j), u = getbit(s, j + 1);
    int nows = s - l * bit3[j] - u * bit3[j + 1];
    if(maps[i][j] != ’.’) {
     if(l == 0 && u == 0) {
      transfer(dp[now][k], dp[pre][k]);
     }
    } else if(l == 0 && u == 0) { //build a pair () when can walk down and right
     if(maps[i][j + 1] == ’.’ && maps[i + 1][j] == ’.’) {
      int nxt = nows + bit3[j] + 2 * bit3[j + 1];
      transfer(dp[now][Hash[nxt]], dp[pre][k]);
     }
    } else if(l == 1 && u == 1) { //merge ((
     int cnt = 0;
     for(int b = j + 2; b <= m; b++) {
      int tmp = getbit(nows, b);
      if(tmp == 2) cnt--;
      if(tmp == 1) cnt++;
      if(cnt == -1) {
       transfer(dp[now][Hash[nows - bit3[b]]], dp[pre][k]);
       break;
      }
     }
    } else if(l == 2 && u == 2) { //merge ))
     int cnt = 0;
     for(int b = j - 1;b >= 0;b--) {
      int tmp = getbit(nows, b);
      if(tmp == 1) cnt++;
      if(tmp == 2) cnt--;
      if(cnt == 1) {
       transfer(dp[now][Hash[nows + bit3[b]]], dp[pre][k]);
       break;
      }
     }
    } else if(l == 2 && u == 1) { //merge )(
     transfer(dp[now][Hash[nows]], dp[pre][k]);
    } else if((!l && u) || (l && !u)) {
     if(maps[i + 1][j] == '.')
      transfer(dp[now][Hash[nows + (l + u) * bit3[j]]], dp[pre][k]);
     if(maps[i][j + 1] == '.')
      transfer(dp[now][Hash[nows + (l + u) * bit3[j + 1]]], dp[pre][k]);
    } else if(l == 1 && u == 2) {/* only happen last step*/}
   }
  }
  swap(pre, now); CC(dp[now], 0);//memset to set illegal state -1
  for(int k = 0, s; s = status[k], s < bit3[m]; k++)
   if(dp[pre][k] != -1) dp[now][Hash[s * 3]] = dp[pre][k];
 }
 return max(0LL, dp[now][Hash[state]]);
}
int main() {
 int n, m;
 REP(i, 1, N) bit3[i] = bit3[i - 1] * 3;
 REP(i, 0, bit3[N - 1]) {
  if(check(i)) {
   Hash[i] = allS;
   status[allS++] = i;
  } else {
   Hash[i] = -1;
  }
 }
 status[allS] = MAXN;
 while(scanf("%d %d", &n, &m) == 2 && (n + m)) {
  CC(maps, 0);
  REP(i, 0, n) scanf("%s", maps[i]);
        maps[n][0] = maps[n][m - 1] = '.';
  printf("%I64d\n", DP(n, m));
 }
 return 0;
}
```

5.2 Mask DP II

```
class HashTable {
private:
 const static int SIZE = 1000000, MOD = 10007;
 struct HashCell {
  int value, idx;
  HashCell *mask;
 } pool[SIZE], *g[MOD], *pp;
#define hashFunction(x) ((x) % MOD)
public:
 void clear() {
  memset(g, 0, sizeof(g));
  pp = pool;
 }
 int find(int x) {
  int hash = hashFunction(x);
  for(HashCell *i = g[hash]; i != NULL; i = i->mask) {
   if(i->value == x) return i->idx;
  }
  return -1;
 }
 void insert(int x, int idx) {
  int hash = hashFunction(x);
  pp->idx = idx, pp->value = x;
  pp->mask = g[hash];
  g[hash] = pp++;
 }
} hashTable;
const int N = 10;
const int STATE_CNT = 1000000;
const int INF = 10000000;
const int HEX = 10;/*BIT[i] = HEX^i*/
const int BIT[] = {1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000};
int state[2][STATE_CNT], dp[2][STATE_CNT];
int encode(const int s[], const int M) {/*repeat down to keep min-notation*/
 int lab[N], cnt = 0, newS = 0;
 memset(lab, -1, sizeof(lab));
 lab[0] = cnt++;/*0 means not be used*/
 for(int i = M - 1; i >= 0; i--) {
  newS *= HEX;
  newS += (lab[s[i]] = lab[s[i]] == -1 ? cnt++ : lab[s[i]]);
 }
 return newS;
}
int cnt[N];/* the number of each plugin */
void decode(int src, int* dest, const int M) {/* decode mask code into array */
 memset(cnt, 0, sizeof(cnt));
 REP(i, 0, M) {
  cnt[src % HEX]++;
  dest[i] = src % HEX;
  src /= HEX;
 }
}
bool isOneBlock(int state) {
 int last = -1;
 while(state) {
  int now = state % HEX;
  if(now != 0 && now != last && last != -1) return false;
  if(now != 0) last = now;
  state /= HEX;
 }
 return true;
}
void transfer(int now, int mask, int val, int& newCnt) {
 int idx = hashTable.find(mask);
 if(idx != -1) {
  dp[now][idx] = max(dp[now][idx], val);
 } else {
  idx = newCnt;
  hashTable.insert(mask, newCnt++);
  state[now][idx] = mask;
  dp[now][idx] = val;
 }
}
int DP(int n, int m, int maps[N][N]) {
 state[0][0] = dp[0][0] = 0;
 int ans = -INF, newS[N], now = 0, pre = 1;
 int oldCnt = 0, newCnt = 1, M = m + 1, mask;
 REP(i, 0, n) {
  REP(j, 0, m) {
   now ^= 1, pre ^= 1;
   oldCnt = 0;
   swap(oldCnt, newCnt);
   hashTable.clear();
   REP(k, 0, oldCnt) {
    decode(state[pre][k], newS, M);
    if(newS[j] == 0 && newS[j + 1] == 0) {
     transfer(now, state[pre][k], dp[pre][k], newCnt);
     newS[j] = newS[j + 1] = *max_element(newS, newS + M) + 1;
     mask = encode(newS, M);
     transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
    } else if(newS[j] == 0 && newS[j + 1]) {
     newS[j] = newS[j + 1];
     mask = encode(newS, M);
     transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
     if(cnt[newS[j + 1]] > 1) {
      newS[j] = newS[j + 1] = 0;
      mask = encode(newS, M);
      transfer(now, mask, dp[pre][k], newCnt);
     }
    } else if(newS[j] && newS[j + 1] == 0) {
     newS[j + 1] = newS[j];
     mask = encode(newS, M);
     transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
     if(cnt[newS[j]] > 1) {
      newS[j] = newS[j + 1] = 0;
      mask = encode(newS, M);
      transfer(now, mask, dp[pre][k], newCnt);
     }
    } else if(newS[j] && newS[j + 1]) {
     /* drop current block */
     int a = 0, b = 0;
     if(newS[j] == newS[j + 1]) {
      if(cnt[newS[j]] > 2) {
       swap(a, newS[j]), swap(b, newS[j + 1]);
       int mask = encode(newS, M);
       transfer(now, mask, dp[pre][k], newCnt);
       swap(a, newS[j]), swap(b, newS[j + 1]);
      }
     } else {
      if(cnt[newS[j]] > 1 && cnt[newS[j + 1]] > 1) {
       swap(a, newS[j]), swap(b, newS[j + 1]);
       int mask = encode(newS, M);
       transfer(now, mask, dp[pre][k], newCnt);
       swap(a, newS[j]), swap(b, newS[j + 1]);
      }
     }
     /* merge two block */
     int minn = min(newS[j], newS[j + 1]);
     for(int b = 0; b <= m; b++) {
      if(newS[b] == newS[j] || newS[b] == newS[j + 1])
       newS[b] = minn;
     }
     mask = encode(newS, M);
     transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
    }
   }
  }
  now ^= 1, pre ^= 1;
  oldCnt = 0;
  swap(oldCnt, newCnt);
  hashTable.clear();/* two different mask can change into one */
  REP(k, 0, oldCnt) {
   if(isOneBlock(state[pre][k]))
    ans = max(ans, dp[pre][k]);
   if(state[pre][k] - BIT[m] > 0) {
    decode((state[pre][k] - BIT[m]) * 10, newS, M);
    if(mask = encode(newS, M))/* if mask != 0 */
     transfer(now, mask, dp[pre][k], newCnt);
   } else if(state[pre][k] != 0) {
    decode(state[pre][k] * 10, newS, M);
    if(mask = encode(newS, M))
     transfer(now, mask, dp[pre][k], newCnt);
   }
  }
 }
 return ans;
}

int main() {
 int n;
 while(scanf("%d", &n) == 1 && n) {
  int maps[N][N], ans = -INF;
  REP(i, 0, n) {
   REP(j, 0, n) {
    scanf("%d", &maps[i][j]);
    ans = max(ans, maps[i][j]);
   }
  }
  if(ans <= 0)
   printf("%d\n", ans);
  else
   printf("%d\n", DP(n, n, maps));
 }
 return 0;
}
```

5.3 Veterx Pair on Tree

```
const int V = 10000, E = V << 1;
struct edge {
  int v, c;
  edge *nxt;
}pool[E], *g[V], *pp;
//initialize and add_edge Function
int _size[V], vis[V], dist[V], root, maxn;
void select(int v, int pre, int tree_size) {
  int max_sub = 0;
  _size[v] = 1;
  for(edge* i = g[v];i != NULL;i = i->nxt) {
    if(i->v == pre || vis[i->v]) continue;
    select(i->v, v, tree_size);
    _size[v] += _size[i->v];
    checkmax(max_sub, _size[i->v]);
  }
  checkmax(max_sub, tree_size - _size[v]);
  if(checkmin(maxn, max_sub))
    root = v;
}
int _count(int beg, int end, int k) {
  int ret = 0, lo = beg, hi = end - 1;
  sort(dist + beg, dist + end);
  while(lo < hi) {
    if(dist[hi] + dist[lo] <= k)
      ret += hi - lo++;
    else
      hi--;
  }
  return ret;
}
void dfs(int root, int pre, int len, int& end) {
  dist[end++] = len, _size[root] = 1;
  for(edge *i = g[root];i != NULL;i = i->nxt) {
    if(i->v == pre || vis[i->v]) continue;
    dfs(i->v, root, len + i->c, end);
    _size[root] += _size[i->v];
  }
}
int get_sub_solve(int root, int k) {
  int beg = 0, end = 0, res = 0;
  for(edge*i = g[root];i != NULL;i = i->nxt) {
    if(vis[i->v]) continue;
    dfs(i->v, root, i->c, end);
    res -= _count(beg, end, k);
    beg = end;
  }
  dist[end++] = 0;
  res += _count(0, end, k);
  return res;
}
//number of path less than k
void solve(int n, int k) {
  queue<int> q;
  q.push(0);
  _size[0] = n;
  CC(vis, 0);
  int res = 0;
  while(q.empty() == false) {
    int now = q.front();
    q.pop();
    maxn = numeric_limits<int>::max();
    select(now, -1, _size[now]);
    vis[root] = 1;
    for(edge*i = g[root];i != NULL;i = i->nxt)
      if(vis[i->v] == false) q.push(i->v);
    res += get_sub_solve(root, k);
  }
  printf("%d\n", res);
}
```

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

```
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;
}

/// here is the fast code.
ll val[1<<20];
ll p_rho(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;
         }
      }
   }
   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;
}
```

6.2.5 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

```
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

```
锘縧
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 Romberg

```
/*求 f(x) 在区间[aa, bb] 上的积分*/
#include <iostream>
#include <cstdio>
#include <cmath>
using namespace std;
const int N = 11, MAXR = 17; //迭代次数
const double eps = 1e-12; //精度
int n;
double a[N];
double f(double x){
   double k = 0.0, xx = 1;
   for(int i = 1;i <= n;++i){
      k += a[i] * xx;
      xx *= x;
   }
   return sqrt(1 + k * k);
}
double Romberg(double aa, double bb){ //aa bb 积分下上限
   int m, n;//m 控制迭代次数, 而n 控制复化梯形积分的分点数n=2^m
   double h, x;
   double s, q;
   double ep; //精度要求
   double y[MAXR]; //为节省空间只需一维数组,
    //每次循环依次存储 Romberg 计算表的每行元素以供计算下一行算完后更新,,
    double p; // p 总是指示待计算元素的前一个元素-> 同一行
    //迭代初值
    h = bb - aa;
    y[0] = 0.5 * h * (f(aa) + f(bb));
    m = n = 1;
    ep = eps + 1.0;
    //迭代计算
    while (ep >= eps && m < MAXR){
        //复化积分公式求 T2n -> Romberg 计算表中的第一列n 初始为1 以后倍增
        p = 0.0;
        for (int i = 0; i < n; ++i)//求 Hn{
            x = aa + (i + 0.5) * h;
            p += f(x);
        }
        p = 0.5 * (y[0] + h * p); //求 T2n = 1/2( Tn + Hn ) 用p 指示

        //求第 m 行元素根据Romberg 计算表本行的前一个元素( p 指示)
        //和上一行左上角元素 -> y[k - 1] 指示求得
        s = 1.0;
        for (int k = 1; k <= m;++k){
            s = pow(4.0, k); //!!!
            //s *= 4.0;
            q = (pow(4.0, k) * p - y[k - 1]) / (pow(4.0, k) - 1.0);
            y[k - 1] = p;
            p = q;
        }
        ep = fabs(q - y[m - 1]);
        y[m++] = q;
        n *= 2;
        h *= 0.5;
    }
    return q;
}
int main()
{
    while(1 == scanf("%d", &n)){
        for(int i = 1;i <= n;++i){
            scanf("%lf", a + i);
            a[i] *= i;
        }
        printf("%.2lf\n", Romberg(0.0, 10.0));
    }
    return 0;
}
```

7 Computational Geometry

7.1 Formula of Geometry

```
三角形
:
1. 半周长P=(a+b+c)/2
2. 面积S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3. 中线Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4. 角平分线Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5. 高线Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6. 内切圆半径r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
            =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
            =Ptan(A/2)tan(B/2)tan(C/2)
7. 外接圆半径R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))四边形

:
D1,为对角线D2,对角线中点连线M,为对角线夹角A
1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2以下对圆的内接四边形
()
3. ac+bd=D1D2
4. S=sqrt((P-a)(P-b)(P-c)(P-d)),为半周长P正边形

n:为外接圆半径
R,为内切圆半径r
1. 中心角A=2PI/n
2. 内角C=(n-2)PI/n
3. 边长a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))圆

:
1. 弧长l=rA
2. 弦长a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积S1=rl/2=r^2A/2
5. 弓形面积S2=(rl-a(r-h))/2=r^2(A-sin(A))/2棱柱

:
1. 体积V=Ah,为底面积A,为高h
2. 侧面积S=lp,为棱长l,为直截面周长p
3. 全面积T=S+2A棱锥

:
1. 体积V=Ah/3,为底面积A,为高h以下对正棱锥
()
2. 侧面积S=lp/2,为斜高l,为底面周长p
3. 全面积T=S+A棱台

:
1. 体积V=(A1+A2+sqrt(A1A2))h/3,A1.为上下底面积A2,为高h以下为正棱台
()
2. 侧面积S=(p1+p2)l/2,p1.为上下底面周长p2,为斜高l
3. 全面积T=S+A1+A2圆柱

:
1. 侧面积S=2PIrh
2. 全面积T=2PIr(h+r)
3. 体积V=PIr^2h圆锥

:
1. 母线l=sqrt(h^2+r^2)
2. 侧面积S=PIrl
3. 全面积T=PIr(l+r)
4. 体积V=PIr^2h/3圆台

:
1. 母线l=sqrt(h^2+(r1-r2)^2)
2. 侧面积S=PI(r1+r2)l
3. 全面积T=PIr1(l+r1)+PIr2(l+r2)
4. 体积V=PI(r1^2+r2^2+r1r2)h/3球

:
1. 全面积T=4PIr^2
2. 体积V=4PIr^3/3球台

:
1. 侧面积S=2PIrh
2. 全面积T=PI(2rh+r1^2+r2^2)
3. 体积V=PIh(3(r1^2+r2^2)+h^2)/6球扇形

:
1. 全面积T=PIr(2h+r0),为球冠高h,为球冠底面半径r0
2. 体积V=2PIr^2h/3
```

7.2 Float Method

```
//浮点几何函数库
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};
struct line{point a,b;};
//计算cross product (P1-P0)x(P2-P0)
double xmult(point p1,point p2,point p0){
  return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,
       double y0){
  return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
//计算dot product (P1-P0).(P2-P0)
double dmult(point p1,point p2,point p0){
  return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
double dmult(double x1,double y1,double x2,double y2,double x0,
       double y0){
  return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}
//两点距离
double distance(point p1,point p2){
  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double distance(double x1,double y1,double x2,double y2){
  return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
//判三点共线
int dots_inline(point p1,point p2,point p3){
  return zero(xmult(p1,p2,p3));
}
int dots_inline(double x1,double y1,double x2,double y2,double x3,
       double y3){
  return zero(xmult(x1,y1,x2,y2,x3,y3));
}
//判点是否在线段上包括端点,
int dot_online_in(point p,line l){
  return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a
         .y-p.y)*(l.b.y-p.y)<eps;
}
int dot_online_in(point p,point l1,point l2){
  return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.
         y)*(l2.y-p.y)<eps;
}
int dot_online_in(double x,double y,double x1,double y1,double x2,
       double y2){
  return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)*(x2-x)<eps&&(y1-y)*(
         y2-y)<eps;
}
//判点是否在线段上不包括端点,
int dot_online_ex(point p,line l){
  return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))
         &&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));
}
int dot_online_ex(point p,point l1,point l2){
  return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y)
         )&&(!zero(p.x-l2.x)||!zero(p.y-l2.y));
}
int dot_online_ex(double x,double y,double x1,double y1,double x2,
       double y2){
  return dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1)
         )&&(!zero(x-x2)||!zero(y-y2));
}
//判两点在线段同侧点在线段上返回,0
int same_side(point p1,point p2,line l){
  return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
}
int same_side(point p1,point p2,point l1,point l2){
  return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}
//判两点在线段异侧点在线段上返回,0
int opposite_side(point p1,point p2,line l){
  return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
}
int opposite_side(point p1,point p2,point l1,point l2){
  return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
// 点关于直线的对称点// by lyt
// 缺点：用了斜率
// 也可以利用点到直线上的最近点来做，避免使用斜率。""
point symmetric_point(point p1, point l1, point l2) {
  point ret;
  if (l1.x > l2.x - eps && l1.x < l2.x + eps) {
    ret.x = (2 * l1.x - p1.x);
  ret.y = p1.y;
  } else {
    double k = (l1.y - l2.y ) / (l1.x - l2.x);
    ret.x = (2*k*k*l1.x + 2*k*p1.y - 2*k*l1.y - k*k*p1.x + p1.x) /
           (1 + k*k);
    ret.y = p1.y - (ret.x - p1.x ) / k;
  }
  return ret;
}
//判两直线平行
int parallel(line u,line v){
 return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b.y));
}
int parallel(point u1,point u2,point v1,point v2){
 return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
}
//判两直线垂直
int perpendicular(line u,line v){
 return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b.y));
}
int perpendicular(point u1,point u2,point v1,point v2){
 return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
}
//判两线段相交包括端点和部分重合,
int intersect_in(line u,line v){
 if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
  return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
 return dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
 if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
  return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
 return dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u2);
}
//判两线段相交不包括端点和部分重合,
int intersect_ex(line u,line v){
 return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
 return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}
//计算两直线交点注意事先判断直线是否平行,!
//线段交点请另外判线段相交同时还是要判断是否平行(!)
point intersection(line u,line v){
 point ret=u.a;
 double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
    /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
 ret.x+=(u.b.x-u.a.x)*t;
 ret.y+=(u.b.y-u.a.y)*t;
 return ret;
}
point intersection(point u1,point u2,point v1,point v2){
 point ret=u1;
 double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
    /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
 ret.x+=(u2.x-u1.x)*t;
 ret.y+=(u2.y-u1.y)*t;
 return ret;
}
//点到直线上的最近点
point ptoline(point p,line l){
 point t=p;
 t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
 return intersection(p,t,l.a,l.b);
}
point ptoline(point p,point l1,point l2){
 point t=p;
 t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
 return intersection(p,t,l1,l2);
}
//点到直线距离
double disptoline(point p,line l){
 return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoline(point p,point l1,point l2){
 return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
double disptoline(double x,double y,double x1,double y1,double x2,double y2){
 return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);
}
//点到线段上的最近点
point ptoseg(point p,line l){
 point t=p;
 t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
 if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
  return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
 return intersection(p,t,l.a,l.b);
}
point ptoseg(point p,point l1,point l2){
 point t=p;
 t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
 if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
  return distance(p,l1)<distance(p,l2)?l1:l2;
 return intersection(p,t,l1,l2);
}
//点到线段距离
double disptoseg(point p,line l){
 point t=p;
 t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
 if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
  return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
 return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoseg(point p,point l1,point l2){
 point t=p;
 t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
 if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
  return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);
 return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
//矢量以为顶点逆时针旋转并放大倍VPanglescale
point rotate(point v,point p,double angle,double scale){
 point ret=p;
 v.x-=p.x,v.y-=p.y;
 p.x=scale*cos(angle);
 p.y=scale*sin(angle);
 ret.x+=v.x*p.x-v.y*p.y;
 ret.y+=v.x*p.y+v.y*p.x;
 return ret;
}
```

7.3 Int Method

```
//整数几何函数库
//注意某些情况下整数运算会出界!
#define sign(a) ((a)>0?1:(((a)<0?-1:0)))
struct point{int x,y;};
struct line{point a,b;};
//计算cross product (P1-P0)x(P2-P0)
int xmult(point p1,point p2,point p0){
 return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
int xmult(int x1,int y1,int x2,int y2,int x0,int y0){
 return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
//计算dot product (P1-P0).(P2-P0)
int dmult(point p1,point p2,point p0){
 return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
int dmult(int x1,int y1,int x2,int y2,int x0,int y0){
 return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}
//判三点共线
int dots_inline(point p1,point p2,point p3){
 return !xmult(p1,p2,p3);
}
int dots_inline(int x1,int y1,int x2,int y2,int x3,int y3){
 return !xmult(x1,y1,x2,y2,x3,y3);
}
//判点是否在线段上包括端点和部分重合,
int dot_online_in(point p,line l){
 return !xmult(p,l.a,l.b)&&(l.a.x-p.x)*(l.b.x-p.x)<=0&&(l.a.y-p.y)*(l.b.y-p.y)<=0;
}
int dot_online_in(point p,point l1,point l2){
 return !xmult(p,l1,l2)&&(l1.x-p.x)*(l2.x-p.x)<=0&&(l1.y-p.y)*(l2.y-p.y)<=0;
}
int dot_online_in(int x,int y,int x1,int y1,int x2,int y2){
 return !xmult(x,y,x1,y1,x2,y2)&&(x1-x)*(x2-x)<=0&&(y1-y)*(y2-y)<=0;
}
//判点是否在线段上不包括端点,
int dot_online_ex(point p,line l){
 return dot_online_in(p,l)&&(p.x!=l.a.x||p.y!=l.a.y)&&(p.x!=l.b.x||p.y!=l.b.y);
}
int dot_online_ex(point p,point l1,point l2){
 return dot_online_in(p,l1,l2)&&(p.x!=l1.x||p.y!=l1.y)&&(p.x!=l2.x||p.y!=l2.y);
}
int dot_online_ex(int x,int y,int x1,int y1,int x2,int y2){
 return dot_online_in(x,y,x1,y1,x2,y2)&&(x!=x1||y!=y1)&&(x!=x2||y!=y2);
}
//判两点在直线同侧点在直线上返回,0
int same_side(point p1,point p2,line l){
 return sign(xmult(l.a,p1,l.b))*xmult(l.a,p2,l.b)>0;
}
int same_side(point p1,point p2,point l1,point l2){
 return sign(xmult(l1,p1,l2))*xmult(l1,p2,l2)>0;
}
//判两点在直线异侧点在直线上返回,0
int opposite_side(point p1,point p2,line l){
 return sign(xmult(l.a,p1,l.b))*xmult(l.a,p2,l.b)<0;
}
int opposite_side(point p1,point p2,point l1,point l2){
 return sign(xmult(l1,p1,l2))*xmult(l1,p2,l2)<0;
}
//判两直线平行
int parallel(line u,line v){
 return (u.a.x-u.b.x)*(v.a.y-v.b.y)==(v.a.x-v.b.x)*(u.a.y-u.b.y);
}
int parallel(point u1,point u2,point v1,point v2){
 return (u1.x-u2.x)*(v1.y-v2.y)==(v1.x-v2.x)*(u1.y-u2.y);
}
//判两直线垂直
int perpendicular(line u,line v){
 return (u.a.x-u.b.x)*(v.a.x-v.b.x)==-(u.a.y-u.b.y)*(v.a.y-v.b.y);
}
int perpendicular(point u1,point u2,point v1,point v2){
 return (u1.x-u2.x)*(v1.x-v2.x)==-(u1.y-u2.y)*(v1.y-v2.y);
}
//判两线段相交包括端点和部分重合,
int intersect_in(line u,line v){
 if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
  return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
 return dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
 if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
  return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
 return dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u2);
}
//判两线段相交不包括端点和部分重合,
int intersect_ex(line u,line v){
 return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
 return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}
```

7.4 Polygon Method

```
#include <stdlib.h>
#include <math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};
double xmult(point p1,point p2,point p0){
 return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
//判定凸多边形顶点按顺时针或逆时针给出允许相邻边共线,,
int is_convex(int n,point* p){
 int i,s[3]={1,1,1};
 for (i=0;i<n&&s[1]|s[2];i++)
  s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
 return s[1]|s[2];
}
//判定凸多边形顶点按顺时针或逆时针给出不允许相邻边共线,,
int is_convex_v2(int n,point* p){
 int i,s[3]={1,1,1};
 for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
  s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
 return s[0]&&s[1]|s[2];
}
//判点在凸多边形内或多边形边上顶点按顺时针或逆时针给出,
int inside_convex(point q,int n,point* p){
 int i,s[3]={1,1,1};
 for (i=0;i<n&&s[1]|s[2];i++)
  s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
 return s[1]|s[2];
}
//判点在凸多边形内顶点按顺时针或逆时针给出在多边形边上返回,,0
int inside_convex_v2(point q,int n,point* p){
 int i,s[3]={1,1,1};
 for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
  s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
 return s[0]&&s[1]|s[2];
}
//判点在任意多边形内顶点按顺时针或逆时针给出,
//表示点在多边形边上时的返回值on_edge,为多边形坐标上限offset
int inside_polygon(point q,int n,point* p,int on_edge=1){
 point q2;
 int i=0,count;
 while (i<n)
  for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
   if (zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)*(p[(i+1)%n].y-q.y)<eps)
    return on_edge;
   else if (zero(xmult(q,q2,p[i])))
    break;
   else if (xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[(i+1)%n])<-eps)
    count++;
 return count&1;
}
inline int opposite_side(point p1,point p2,point l1,point l2){
 return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
inline int dot_online_in(point p,point l1,point l2){
 return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
//判线段在任意多边形内顶点按顺时针或逆时针给出与边界相交返回,,1
int inside_polygon(point l1,point l2,int n,point* p){
 point t[MAXN],tt;
 int i,j,k=0;
 if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
  return 0;
 for (i=0;i<n;i++)
  if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
   return 0;
  else if (dot_online_in(l1,p[i],p[(i+1)%n]))
   t[k++]=l1;
  else if (dot_online_in(l2,p[i],p[(i+1)%n]))
   t[k++]=l2;
  else if (dot_online_in(p[i],l1,l2))
   t[k++]=p[i];
 for (i=0;i<k;i++)
  for (j=i+1;j<k;j++){
   tt.x=(t[i].x+t[j].x)/2;
   tt.y=(t[i].y+t[j].y)/2;
   if (!inside_polygon(tt,n,p))
    return 0;
  }
 return 1;
}
point intersection(line u,line v){
 point ret=u.a;
 double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
   /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
 ret.x+=(u.b.x-u.a.x)*t;
 ret.y+=(u.b.y-u.a.y)*t;
 return ret;
}
point barycenter(point a,point b,point c){
 line u,v;
 u.a.x=(a.x+b.x)/2;
 u.a.y=(a.y+b.y)/2;
 u.b=c;
 v.a.x=(a.x+c.x)/2;
 v.a.y=(a.y+c.y)/2;
 v.b=b;
 return intersection(u,v);
}
//多边形重心
point barycenter(int n,point* p){
 point ret,t;
 double t1=0,t2;
 int i;
 ret.x=ret.y=0;
 for (i=1;i<n-1;i++)
  if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
   t=barycenter(p[0],p[i],p[i+1]);
   ret.x+=t.x*t2;
   ret.y+=t.y*t2;
   t1+=t2;
  }
 if (fabs(t1)>eps)
  ret.x/=t1,ret.y/=t1;
 return ret;
}
```

7.5 Circles Method

```
#include <math.h>
#define eps 1e-8
struct point{double x,y;};
double xmult(point p1,point p2,point p0){
 return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double distance(point p1,point p2){
 return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double disptoline(point p,point l1,point l2){
 return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
point intersection(point u1,point u2,point v1,point v2){
 point ret=u1;
 double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
   /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
 ret.x+=(u2.x-u1.x)*t;
 ret.y+=(u2.y-u1.y)*t;
 return ret;
}
//判直线和圆相交包括相切,
int intersect_line_circle(point c,double r,point l1,point l2){
 return disptoline(c,l1,l2)<r+eps;
}
//判线段和圆相交包括端点和相切,
int intersect_seg_circle(point c,double r,point l1,point l2){
 double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
 point t=c;
 if (t1<eps||t2<eps)
  return t1>-eps||t2>-eps;
 t.x+=l1.y-l2.y;
 t.y+=l2.x-l1.x;
 return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
}
//判圆和圆相交包括相切,
int intersect_circle_circle(point c1,double r1,point c2,double r2){
 return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
}
//计算圆上到点最近点p如,与圆心重合p返回,本身p
point dot_to_circle(point c,double r,point p){
 point u,v;
 if (distance(p,c)<eps)
  return p;
 u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
 u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
 v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
 v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
 return distance(u,p)<distance(v,p)?u:v;
}
//计算直线与圆的交点保证直线与圆有交点,
//计算线段与圆的交点可用这个函数后判点是否在线段上
void intersection_line_circle(point c,double r,point l1,point l2,point& p1,point& p2){
 point p=c;
 double t;
 p.x+=l1.y-l2.y;
 p.y+=l2.x-l1.x;
 p=intersection(p,c,l1,l2);
 t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(l1,l2);
 p1.x=p.x+(l2.x-l1.x)*t;
 p1.y=p.y+(l2.y-l1.y)*t;
 p2.x=p.x-(l2.x-l1.x)*t;
 p2.y=p.y-(l2.y-l1.y)*t;
}
//计算圆与圆的交点保证圆与圆有交点圆心不重合,,
void intersection_circle_circle(point c1,double r1,point c2,double r2,point& p1,point& p2){
 point u,v;
 double t;
 t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(c1,c2))/2;
 u.x=c1.x+(c2.x-c1.x)*t;
 u.y=c1.y+(c2.y-c1.y)*t;
 v.x=u.x+c1.y-c2.y;
 v.y=u.y-c1.x+c2.x;
 intersection_line_circle(c1,r1,u,v,p1,p2);
}
```

7.6 Scan Line

```
#include <cstdio>
#include <set>
#include <cmath>
#include <algorithm>
using namespace std;
const int N = 50010;
const double EPS = 1e-8;
int X;
double a, b;
int x[N], y[N], r[N];
typedef struct Node{
   int id;
   bool tag;
   bool operator < (const Node& other)const{
      a = sqrt(1.0*r[id]*r[id]-1.0*(X-x[id])*(X-x[id]));
      b = sqrt(1.0*r[other.id]*r[other.id]-1.0*(X-x[other.id])*(X-x[other.id]));
      a = tag ? y[id]+a : y[id]-a;
      b = other.tag ? y[other.id]+b : y[other.id]-b;
      if (fabs(a-b) > EPS) return a < b;
      else return tag < other.tag;
   }
}Node;
typedef struct Event{
   int id, value;
   bool in;
   bool operator < (const Event &other)const{
      return value < other.value;
   }
}Event;
Event arr[2*N];
int n;
int f[N]={};
int main()
{
   Event e;
   Node node;
   while (scanf("%d", &n) == 1){
      set<Node>st;
      set<Node>::iterator it1, it2;
      for (int i = 0; i < n; ++i){
         f[i] = 0;
         scanf("%d %d %d", &x[i], &y[i], &r[i]);
         e.id = i, e.value = x[i]-r[i], e.in = 1;
         arr[i] = e;
         e.id = i, e.value = x[i]+r[i], e.in = 0;
         arr[i+n] = e;
      }
      n *= 2;
      sort(arr, arr+n);
      for (int i = 0; i < n; ++i){
         e = arr[i];
         X = e.value;
         if (e.in == 0){
            node.id = e.id;
            node.tag = 0;
            st.erase(node);
            node.tag = 1;
            st.erase(node);
         }
         else{
            node.id = e.id;
            node.tag = 1;
            it2 = st.lower_bound(node);
            if (it2 == st.begin() || it2 == st.end()){
               f[e.id] = 1;
            }
            else{
               it1 = it2;
               --it1;
               if (it1->id == it2->id) f[e.id] = f[it1->id]+1;
               else f[e.id] = max(f[it1->id], f[it2->id]);
            }
            node.tag = 0;
            st.insert(node);
            node.tag = 1;
            st.insert(node);
         }
      }
      int ans = 0;
      n /= 2;
      for (int i = 0; i < n; ++i) if (ans < f[i]) ans = f[i];
      printf("%d\n", ans);
   }
   return 0;
}
```

7.7 Spherical Distance

```
/*
   Point 表示点的位置其中
    x 表示经度，表示纬度y
   R 为球的半径
   dist 返回两点的球面距离
*/
#include<cstdio>
#include<cmath>
const double R = 1000;
const double PI = acos(-1);
typedef struct Point{
   double x, y;
}Point;
double dist(Point a, Point b)
{
   a.x = a.x*2*PI/360;
   a.y = a.y*2*PI/360;
   b.x = b.x*2*PI/360;
   b.y = b.y*2*PI/360;
   if (fabs(a.x - b.x) < 1e-6) return R*fabs(a.y - b.y);
   else return R*acos(sin(a.y)*sin(b.y)+cos(a.y)*cos(b.y)*cos(a.x-b.x));
}
double d[1010][1010];
int main()
{
   Point pt[1010];
   int n;
   while (scanf("%d", &n) == 1){
      for (int i = 1; i <= n; i++) scanf("%lf %lf", &pt[i].y, &pt[i].x);
      for (int i = 1; i <= n; i++){
         d[i][i] = 0;
         for (int j = i+1; j <= n; j++)
            d[i][j] = d[j][i] = dist(pt[i],pt[j]);
      }
      double tmp = 9999999;
      int mark = 1;
      for (int i = 1; i <= n; i++){
         double max = 0;
         for (int j = 1; j <= n; j++) if (max < d[i][j]) max = d[i][j];
         if (fabs(max-tmp) >= 0.005 && max < tmp) tmp = max, mark = i;
      }
      printf("%.2lf %.2lf\n", pt[mark].y, pt[mark].x);
   }
   return 0;
}
```

7.8 Minimum Circle Cover

```
#include <stdio.h>
#include <math.h>
const int maxn = 1005;
//const double eps = 1e-6;
struct TPoint{
  double x, y;
  TPoint operator-(TPoint &a){
    TPoint p1;
    p1.x = x - a.x;
    p1.y = y - a.y;
    return p1;
  }
};
struct TCircle{
  double r;
  TPoint centre;
};
struct TTriangle{
  TPoint t[3];
};
TCircle c;
TPoint a[maxn];
double distance(TPoint p1, TPoint p2){
  TPoint p3;
  p3.x = p2.x - p1.x;
  p3.y = p2.y - p1.y;
  return sqrt(p3.x * p3.x + p3.y * p3.y);
}
double triangleArea(TTriangle t){
  TPoint p1, p2;
  p1 = t.t[1] - t.t[0];
  p2 = t.t[2] - t.t[0];
  return fabs(p1.x * p2.y - p1.y * p2.x) / 2;
}
TCircle circumcircleOfTriangle(TTriangle t){
    //三角形的外接圆
    TCircle tmp;
    double a, b, c, c1, c2;
    double xA, yA, xB, yB, xC, yC;
    a = distance(t.t[0], t.t[1]);
    b = distance(t.t[1], t.t[2]);
    c = distance(t.t[2], t.t[0]);
    //根据S = a * b * c / R / 求半径4;R
    tmp.r = a * b * c / triangleArea(t) / 4;
    xA = t.t[0].x; yA = t.t[0].y;
    xB = t.t[1].x; yB = t.t[1].y;
    xC = t.t[2].x; yC = t.t[2].y;
    c1 = (xA * xA + yA * yA - xB * xB - yB * yB) / 2;
    c2 = (xA * xA + yA * yA - xC * xC - yC * yC) / 2;
    tmp.centre.x = (c1 * (yA - yC) - c2 * (yA - yB)) /
        ((xA - xB) * (yA - yC) - (xA - xC) * (yA - yB));
    tmp.centre.y = (c1 * (xA - xC) - c2 * (xA - xB)) /
        ((yA - yB) * (xA - xC) - (yA - yC) * (xA - xB));
    return tmp;
}
TCircle MinCircle2(int tce, TTriangle ce){
  TCircle tmp;
  if(tce == 0) tmp.r = -2;
  else if(tce == 1){
    tmp.centre = ce.t[0];
    tmp.r = 0;
  }
  else if(tce == 2){
    tmp.r = distance(ce.t[0], ce.t[1]) / 2;
    tmp.centre.x = (ce.t[0].x + ce.t[1].x) / 2;
    tmp.centre.y = (ce.t[0].y + ce.t[1].y) / 2;
  }
  else if(tce == 3) tmp = circumcircleOfTriangle(ce);
  return tmp;
}
void MinCircle(int t, int tce, TTriangle ce){
  int i, j;
  TPoint tmp;
  c = MinCircle2(tce, ce);
  if(tce == 3) return;
  for(i = 1;i <= t;i++){
    if(distance(a[i], c.centre) > c.r){
      ce.t[tce] = a[i];
      MinCircle(i - 1, tce + 1, ce);
      tmp = a[i];
      for(j = i;j >= 2;j--){
        a[j] = a[j - 1];
      }
      a[1] = tmp;
    }
  }
}
void run(int n){
  TTriangle ce;
  int i;
  MinCircle(n, 0, ce);
  printf("%.2lf %.2lf %.2lf\n", c.centre.x, c.centre.y, c.r);
}
int main()
{
    int n;
  while(scanf("%d", &n) != 1 && n){
    for(int i = 1;i <= n;i++)
      scanf("%lf%lf", &a[i].x, &a[i].y);
    run(n);
  }
  return 0;
}
```

7.9 N-D Volume

```
#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long LL;
const int M = 110;
const int N = 10;
const int MOD = 14121413;
typedef struct VT{
   int a[N], b[N];
}VT;
VT vt[M];
int m, n, size;
VT ans[M*M];
bool Cross(VT &x, VT &y){
   for (int i = 0; i < n; ++i) if (x.a[i] >= y.b[i] || x.b[i] <= y.a[i]) return 0;
   return 1;
}
void Cut (VT &v){
   int s = size, k1, k2;
   for (int i = 0; i < s; ++i){
      if (!Cross(ans[i], v)) continue;
      for (int j = 0; j < n; ++j){
         k1 = max(ans[i].a[j], v.a[j]);
         k2 = min(ans[i].b[j], v.b[j]);
         if (ans[i].a[j] < k1){ans[size] = ans[i];ans[size++].b[j] = k1;}
         if (k2 < ans[i].b[j]){ans[size] = ans[i];ans[size++].a[j] = k2;}
         ans[i].a[j] = k1, ans[i].b[j] = k2;
      }
      ans[i] = ans[--size], --i;
      if (s > size) s = size;
   }
}
int main()
{
   while (scanf("%d %d", &m, &n) == 2){
      for (int i = 0; i < m; ++i){
         for (int j = 0; j < n; ++j) scanf("%d", &vt[i].a[j]);
         for (int j = 0; j < n; ++j) scanf("%d", &vt[i].b[j]);
      }
      int ret = 0;
      for (int i = 0; i < m; ++i){
         size = 0;
         ans[size++] = vt[i];
         for (int j = 0; j < i; ++j) Cut(vt[j]);
         for (int j = 0; j < size; ++j){
            LL tmp = 1;
            for (int k = 0; k < n; ++k) tmp = (tmp*(ans[j].b[k]-ans[j].a[k]))%MOD;
            ret = (tmp+ret)%MOD;
         }
      }
      printf("%d\n", ret);
   }
   return 0;
}
```

7.10 Perimeter Of Circles

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
const int MAX = 110;
const double EPS = 1e-8;
const double PI = acos(-1.0);
typedef struct Node{
   double x, y;
   bool operator <(const Node & other)const{return x < other.x;}
}Node;
double r[MAX], x[MAX], y[MAX];
inline double Dis(double x1, double y1, double x2, double y2){
   return (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2);
}
int main()
{
   int n;
   vector<Node>vt;
   while (scanf("%d", &n) == 1){
      for (int i = 0; i < n; ++i) scanf("%lf %lf %lf", &r[i], &x[i], &y[i]);
      double ans = 0;
      for (int i = 0; i < n; ++i){
         vt.clear();
         Node tmp;
         bool cover = 0;
         for (int j = i+1; j < n; ++j){
            double dis = Dis(x[i], y[i], x[j], y[j]);
            if (dis < (r[i]+r[j])*(r[i]+r[j])+EPS){
               if (sqrt(dis)+min(r[i],r[j]) <= max(r[i],r[j])+EPS){
                  if (r[i] < r[j]+EPS) {cover = 1;break;}
                  continue;
               }
               double angle = atan2(y[j]-y[i],x[j]-x[i]);
               if (angle < 0) angle += 2*PI;
               double da = acos((r[i]*r[i]+dis-r[j]*r[j])/(2*r[i]*sqrt(dis)));
               double a = angle-da, b = angle+da;
               if (a < 0){
                  tmp.x = a+2*PI;
                  tmp.y = 2*PI;
                  vt.push_back(tmp);
                  a = 0;
               }
               if (b > 2*PI){
                  tmp.x = 0;
                  tmp.y = b-2*PI;
                  vt.push_back(tmp);
                  b = 2*PI;
               }
               tmp.x = a, tmp.y = b;
               vt.push_back(tmp);
            }
         }
         if (cover) continue;
         sort(vt.begin(), vt.end());
         double l = 0, a = 0, b = 0;
         int size = vt.size();
         for (int k = 0; k < size; ++k){
            if (vt[k].x <= b){
               if (vt[k].y > b) b = vt[k].y;
            }
            else{
               l += b-a;
               a = vt[k].x, b = vt[k].y;
            }
         }
         l += b-a;
         ans += r[i]*(2*PI-l);
      }
      printf("%.3lf\n", ans);
   }
   return 0;
}
```

7.11 Center of Triangle

```
#include <cstdio>
#include <cmath>
const double PI = acos(-1.0);
double Area(double a, double b, double c){
   double p = (a+b+c)/2;
   return sqrt(p*(p-a)*(p-b)*(p-c));
}
double Fema(double a, double b, double c){
   double anga = acos((b*b+c*c-a*a)/(2*b*c));
   double angb = acos((a*a+c*c-b*b)/(2*a*c));
   double angc = acos((a*a+b*b-c*c)/(2*a*b));
   if (anga >= PI*2/3) return b+c;
   if (angb >= PI*2/3) return a+c;
   if (angc >= PI*2/3) return a+b;
   double ang = angc+PI/3;
   return sqrt(a*a+b*b-2*a*b*cos(ang));
}
double Inner(double a, double b, double c){
   double s = Area(a, b, c);
   double r = 2*s/(a+b+c);
   double anga = acos((b*b+c*c-a*a)/(2*b*c));
   double angb = acos((a*a+c*c-b*b)/(2*a*c));
   double angc = acos((a*a+b*b-c*c)/(2*a*b));
   return r/sin(anga/2)+r/sin(angb/2)+r/sin(angc/2);
}
double Center(double a, double b, double c){
   return sqrt(2*a*a+2*b*b-c*c)/3+sqrt(2*a*a+2*c*c-b*b)/3+sqrt(2*c*c+2*b*b-a*a)/3;
}
double Outer(double a, double b, double c){
   double s = Area(a, b, c);
   return 3*a*b*c/4/s;
}
int main()
{
   int a, b, c, t;
   scanf("%d", &t);
   while (t--){
      scanf("%d %d %d", &a, &b, &c);
      printf("%.3lf %.3lf %.3lf %.3lf\n",
      Fema(a, b, c), Inner(a, b, c), Center(a, b, c), Outer(a, b, c));
   }
   return 0;
}
```

7.12 3D Vector Projection

```
cos(a,b) = (x1*x2+y1*y2+z1*z2)/(sqrt(x1*x1+y1*y1+z1*z1)*sqrt(x2*x2+y2*y2+z2*z2));
a.b = |a|*cos(a,b)
```

7.13 3D Convexhull

```
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <iostream>
using namespace std;
const int MAXN = 1111;
const double EPS = 1e-6;
inline int sgn(double x) {
   return (x > EPS) - (x < -EPS);
}
struct P {
   double x, y, z;
   P() {}
   P(double a, double b, double c) :x(a), y(b), z(c) {}
   P operator - (const P& a) const {
      return P(x - a.x, y - a.y, z - a.z);
   }
   P operator + (const P& a) const {
      return P(x + a.x, y + a.y, z + a.z);
   }
   double len () {
      return sqrt(x*x + y*y + z*z);
   }
};
double dot(const P& a, const P& b) {
   return a.x*b.x + a.y*b.y + a.z*b.z;
}
P det(const P& a, const P& b) {
   return P(a.y*b.z - a.z*b.y
         ,a.z*b.x - a.x*b.z
         ,a.x*b.y - a.y*b.x);
}
P cross(const P &a, const P& b, const P& c) {
   return det(b-a,c-a);
}
double area (P& a, P& b, P& c) {
   return cross(a,b,c).len();
}
double volume (P &u, P& v, P& w, P& p) {
   return dot(cross(u,v,w), p - u);
}
bool coplane(P &a, P& b, P& c, P& d) {
   return sgn(dot(det(c-a,b-a), d-a)) == 0;
}
struct F {
   int a,b,c;
   bool ok;
   F() {}
   F(int aa, int bb,int cc, bool k)
      :a(aa), b(bb), c(cc), ok(k) {}
};
struct CovexHull {
   int n;
   P p[MAXN];
   int cnt;
   F f[MAXN];
   int to[MAXN][MAXN];

   double dir(F & t, P & u) {
      return volume(p[t.a],p[t.b],p[t.c],u);
   }
   void deal(int t,int a, int b) {
      int fid = to[a][b];
      if (!f[fid].ok) return ;
      if (dir(f[fid], p[t]) > EPS) dfs(t, fid);
      else {
         to[t][b] = to[a][t] = to[b][a] = cnt;
         f[cnt++] = F(b,a,t,1);
      }
   }
   void dfs(int t,int cur) {
      f[cur].ok = 0;
      deal(t, f[cur].b, f[cur].a);
      deal(t, f[cur].c, f[cur].b);
      deal(t, f[cur].a, f[cur].c);
   }
   void got() {
      cnt = 0;
      if (n < 4) return ;
      random_shuffle(p, p+n);
      for (int i = 1, j = 1; i < 4; ++i) {
         bool flag = true;
         for (; flag && j < n; ++j) {
            if (i == 1 && (p[0] -p[j]).len() > EPS) {
               swap(p[i],p[j]); flag = false;
            } else if (i == 2 && cross(p[0], p[1], p[j]).len() >
                 EPS) {
               swap(p[i],p[j]); flag = false;
            } else if (i == 3 && !coplane(p[0],p[1],p[2], p[j])) {
               swap(p[i],p[j]); flag = false;
            }
         }
         if (flag) return;
      }
      F now;
      for (int i = 0; i < 4; ++i) {
         now = F(i+1&3, i+2&3, i+3&3, 1);
         if (dir(now, p[i]) > 0) swap(now.b, now.c);
         to[now.a][now.b] = to[now.b][now.c] = to[now.c][now.a] =
               cnt;
         f[cnt++] = now;
      }
      for(int i = 4; i < n; ++i) {
         for (int j=0; j<cnt; ++j) {
            if (f[j].ok && dir(f[j], p[i]) > EPS) {
               dfs(i,j); break;
            }
         }
      }
      int tmp = cnt;
      cnt = 0;
      for (int i = 0; i < tmp; ++i) if (f[i].ok) f[cnt++] = f[i];
   }
   double area() {
      double res = 0;
      for (int i=0; i<cnt; ++i) {
         res += ::area(p[f[i].a], p[f[i].b], p[f[i].c]);
      }
      return res*0.5;
   }
   double vol() {
      P o = P(0,0,0);
      double res = 0;
      for(int i = 0; i < cnt; ++i) {
         res += volume(o, p[f[i].a], p[f[i].b], p[f[i].c]);
      }
      return fabs(res/6);
   }
   bool same(int u,int v) {
      P a = p[f[u].a], b = p[f[u].b], c = p[f[u].c];
      return coplane(a,b,c,p[f[v].a])
         && coplane(a,b,c,p[f[v].b])
         && coplane(a,b,c,p[f[v].c]);
   }
   int face_cnt() {
      int res = 0;
      for (int i=0; i<cnt; ++i) {
         bool t = 1;
         for (int j=0; t && j<i; ++j) {
            if(same(i,j)) t = 0;
         }
         res += t;
      }
      return res;
   }
   P center() {
      P res(0,0,0), pt = p[f[0].a];
      double v = 0, t;
      for (int i=0; i<cnt; ++i) {
         P a = p[f[i].a], b = p[f[i].b], c = p[f[i].c];
         t = volume(pt, a, b, c)/6.0;
         if (t > 0) {
            res.x += (a.x + b.x + c.x + pt.x)*t;
            res.y += (a.y + b.y + c.y + pt.y)*t;
            res.z += (a.z + b.z + c.z + pt.z)*t;
            v += t;
         }
      }
      res.x /= (4*v), res.y /= (4*v), res.z /= (4*v);
      return res;
   }
};
CovexHull ch;
double get_dis(P & pt) {
   double res = 1e100;
   P h;
   double tmp;
   for (int i = 0; i < ch.cnt; ++i) {
      h = cross(ch.p[ch.f[i].a], ch.p[ch.f[i].b], ch.p[ch.f[i].c])
           ;
      tmp = fabs(dot(h,pt-ch.p[ch.f[i].a])/h.len());
      res = min(tmp, res);
   }
   return res;
}
int main() {
   while (cin >> ch.n) {
      for (int i = 0; i < ch.n; ++i) {
         scanf("%lf%lf%lf", &ch.p[i].x, &ch.p[i].y, &ch.p[i].z);
      }
      ch.got();
      P cen = ch.center();
      printf("%.3lf\n", get_dis(cen));
   }
}
```

7.14 2D Convexhull

```
bool g_cmp(const P& a, const P& b) {
   if(sig(a.y - b.y) != 0) return a.y < b.y;
   return a.x < b.x;
}
//the convexhull is anti-clockwise
int graham(P* p, int n, int *ch) {
   if(n < 2) {
      ch[0] = 0;
      return 1;
   }
   sort(p, p + n, g_cmp);
   int len = 0, len0 = 1;
   for (int i = 0; i < n; ++i) {
      while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[
            i])) >= 0) --len;
      ch[len++] = i;
   }
   len0 = len; // notice !!!
   for (int i = n - 2; i >= 0; --i) {
      while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[
            i])) >= 0) --len;
      ch[len++] = i;
   }
   return len - 1;
}
```

7.15 Points in Triangle

```
const double PI = acos(-1.0);
const double eps = 1e-9;
const int N = 510;
struct P {
   int x, y;
} ar[N], br[N];
int n,m;
double ans;
pair<double, int> ap[N];
double bp[N];
int half[N][N];
int below[N][N];
double angle[N][N];
void pre_process() {
   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j)
         angle[i][j] = atan2(ar[j].y - ar[i].y + 0.0, ar[j].x - ar
               [i].x + 0.0);
   }
   for (int i = 0; i < n; ++i) {
      for(int j = 0; j < n; ++j) {
         if (j < i) ap[j] = make_pair(angle[i][j], j);
         if (j > i) ap[j-1] = make_pair(angle[i][j], j);
      }
      sort(ap,ap+n-1);
      for(int j = 0; j < m; ++j) {
         bp[j] = atan2(br[j].y - ar[i].y + 0.0, br[j].x - ar[i].x
               + 0.0);
      }
      sort(bp, bp+m);

      int beg = 0, end = 0;
      for (int j = 0; j < n - 1; ++j) {
         while(end < m && bp[end] + eps < ap[j].first) ++end;
         below[i][ap[j].second] = end;
      }
      beg = 0, end = 0;
      for (int j = 0; j < n - 1; ++j) {
         if (ap[j].first > 0) break;
         double r = ap[j].first + PI;
         while (end < m && bp[end]+ eps < r) ++end;
         while (beg < end && bp[beg]+ eps < ap[j].first) ++beg;
         half[i][ap[j].second] = end - beg;
         half[ap[j].second][i] = m - (end - beg);
      }
   }
}
int cross(P& o, P& a, P& b) {
   return (a.x - o.x)*(b.y - o.y) - (b.x - o.x)*(a.y - o.y);
}
int get(int o,int a,int b) { // oa - > ob
   if (angle[o][a] + eps < angle[o][b]) return below[o][b] - below[o][a];
   return below[o][b] + m - below[o][a];
}
bool get_Ans() {
   pre_process();
   bool has = false;
   ans = 1e100;
   for(int i = 0; i < n; ++i) {
      for (int j = i + 1; j < n; ++j) {
         for (int k = j + 1; k < n; ++k) {
            int a = i, b = j, c = k;
            double area = cross(ar[a],ar[b],ar[c]);
            if (area < 0) swap(b,c);
            int center = get(a,b,c) + get(b,c,a) + get(c,a,b);
            center += half[b][a] + half[c][b] + half[a][c];
            center -= 2*m;
            if(center != 0) {
               has = true;
               ans = min(ans, fabs(area)*0.5/center);
            }
         }
      }
   }
   return has;
}
int main() {
   int cas,tcas = 0;
   for (cin >> cas; cas; --cas) {
      scanf("%d%d",&n,&m);
      for (int i = 0; i < n; ++i) scanf("%d%d", &ar[i].x, &ar[i].y);
      for (int i = 0; i < m; ++i) scanf("%d%d",&br[i].x, &br[i].y);
      if (get_Ans()) printf("Case #%d: %.6lf\n",++tcas,ans);
      else printf("Case #%d: -1\n",++tcas);
   }
}
```

7.16 N Dimension Cut

```
#define N 11
#define M 101
#define MAX 100001
#define MOD 14121413LL
struct DIM {
   __int64 l, u;
};
struct OBJ {
   DIM cod[N];
};
bool inter(OBJ &a, OBJ &b, int n) {
   for (int i = 0; i < n; i++)
      if ((a.cod[i].l - b.cod[i].u)*(a.cod[i].u - b.cod[i].l) >= 0)
         return false;
   return true;
}
OBJ rec[MAX];
int top;
__int64 cut(OBJ obj[M], int m, int n) {
   top = 0;
   __int64 ans=0,tmp=1;
   for (int i = 0; i < m; i++) {
      tmp=1;
      for(int j=0;j<n;j++)
         tmp=(tmp*(obj[i].cod[j].u-obj[i].cod[j].l))%MOD;
      ans=(ans+tmp)%MOD;
      for (int j = top - 1; j >= 0; j--)
         if (inter(rec[j], obj[i], n)) {
            for(int k=0;k<n;k++) {
               if(rec[j].cod[k].l<obj[i].cod[k].l) {
                  rec[top]=rec[j];
                  rec[top++].cod[k].u=obj[i].cod[k].l;
                  rec[j].cod[k].l=obj[i].cod[k].l;
               }
               if(rec[j].cod[k].u>obj[i].cod[k].u) {
                  rec[top]=rec[j];
                  rec[top++].cod[k].l=obj[i].cod[k].u;
                  rec[j].cod[k].u=obj[i].cod[k].u;
               }
            }
            tmp=1;
            for(int k=0;k<n;k++)
               tmp=(tmp*(rec[j].cod[k].u-rec[j].cod[k].l))%MOD;
            ans=(ans-tmp)%MOD;
            if(ans<0)
               ans+=MOD;
            --top;
            swap(rec[j],rec[top]);
         }
      rec[top++]=obj[i];
   }
   return ans;
}
int main() {
   while (scanf("%d%d", &m, &n) != EOF) {
      OBJ obj[M];
      for (int i = 0; i < m; i++) {
         for (int j = 0; j < n; j++)
            scanf("%I64d", &obj[i].cod[j].l);
         for (int j = 0; j < n; j++) {
            scanf("%I64d", &obj[i].cod[j].u);
            if (obj[i].cod[j].l > obj[i].cod[j].u)
               swap(obj[i].cod[j].l, obj[i].cod[j].u);
         }
      }
      printf("%I64d\n",cut(obj,m,n));
   }
}
```

8 Sevenzero Geometry

```
const double PI = acos(-1.0);
int sgn(double a) {
   return (a > EPS) - (a < -EPS);
}
struct Po {
   double x,y;
   Po(double a=0, double b=0){x=a;y=b;}
   Po operator - (const Po &a) const {
      return Po(x - a.x, y - a.y);
   }
   Po vect(double a) const { // return a vector of length a
      a /= sqrt(x*x + y*y);
      return Po(x * a, y * a);
   }
   Po left() const { // rotate 90 degrees
      return Po(-y, x);
   }
};
struct Seg {
   Po s,e;
   Seg(){}
   Seg(Po a,Po b){s=a;e=b;}
};
struct Line {
   double a,b,c;
   Line(double x=1,double y=-1,double z=0){a=x;b=y;c=z;}
   Line(Po p1,Po p2) {
      int sig=1;
      a=p2.y-p1.y;
      if(a<0) {a=-a;sig=-1;}
      b=sig*(p1.x-p2.x);
      c=sig*(p1.y*p2.x-p1.x*p2.y);
   }
};
double xm(Po a,Po b,Po c) { // (ab)X(ac)

   return (b.x-a.x)*(c.y-a.y)-(c.x-a.x)*(b.y-a.y);
}
double dm(Po a,Po b,Po c) { // (ab)*(ac)

   return (b.x-a.x)*(c.x-a.x)+(b.y-a.y)*(c.y-a.y);
}
bool posy(Po &a) { // angle sort
   if(a.y>0||(a.y==0&&a.x>0))
      return 1;
   return 0;
}
bool cmp(Po a,Po b) { // sgn recommended
   if(posy(a)!=posy(b))
      return posy(a)>posy(b);
   return xm(Po(0,0),a,b)>0;
}
Po rotate(Po p,Po p0,double ang) {
   Po vec=p-p0,ret;
   ret.x = vec.x * cos(ang) - vec.y * sin(ang);
   ret.y = vec.x * sin(ang) + vec.y * cos(ang);
   return ret+p0;
}
int segcross(Seg a,Seg b) { // 1 normal 2 abnormal
   double xm1,xm2,xm3,xm4;
   xm1=xm(a.s,a.e,b.s);
   xm2=xm(a.s,a.e,b.e);
   xm3=xm(b.s,b.e,a.s);
   xm4=xm(b.s,b.e,a.e);
   if(xm1*xm2<-EPS&&xm3*xm4<-EPS)
      return 1;
   if(eq(xm1,0)&&dm(b.s,a.s,a.e)<EPS) return 2;
   if(eq(xm2,0)&&dm(b.e,a.s,a.e)<EPS) return 2;
   if(eq(xm3,0)&&dm(a.s,b.s,b.e)<EPS) return 2;
   if(eq(xm4,0)&&dm(a.e,b.s,b.e)<EPS) return 2;
   return 0;
}
bool Linecross(Line l1,Line l2,Po &p) {
   double d=l1.a*l2.b-l2.a*l1.b;
   if(fabs(d)<EPS)
      return false;
   p.x=(l2.c*l1.b-l1.c*l2.b)/d;
   p.y=(l2.a*l1.c-l1.a*l2.c)/d;
   return true;
}
double caliper() { // convex caliper
   if(top<=3)
      return dis(conv[0],conv[1]);
   int p=0;
   double ans=0;
   for(int i=0;i<top-1;i++) {
      while(xm(conv[p],conv[i],conv[i+1])<xm(conv[(p+1)%top],conv[i],conv[i+1])+EPS)
         p=(p+1)%top;
      ans=max(ans,max(dis(conv[p],conv[i]),dis(conv[p],conv[i+1])));
   }
   return ans;
}
#define N 1510 // ------half plane-------
```

```
struct PS {
   int size;
   Po p[N];
   PS(){size=0;}
   void ins(Po po)
   {
      if(size&&eq(p[size-1].x,po.x)&&eq(p[size-1].y,po.y))
         return;
      p[size++]=po;
   }
};
struct HP {
   double a,b,c;
   HP(double x=1,double y=-1,double z=0){a=x;b=y;c=z;}
   double ptol(Po p){return a*p.x+b*p.y+c;}
};
PS cut(Po p1,Po p2,PS ps) { // counterclockwise
   PS ret;
   for(int i=0;i<ps.size-1;i++) {
      double xm1=xm(p1,p2,ps.p[i]),xm2=xm(p1,p2,ps.p[i+1]);
      Po crp;
      if(xm1>EPS&&xm2<-EPS) {
         ret.ins(ps.p[i]);
         linecross(Line(p1,p2),Line(ps.p[i],ps.p[i+1]),crp);
         ret.ins(crp);
         continue;
      }
      if(xm1<-EPS&&xm2>EPS) {
         linecross(Line(p1,p2),Line(ps.p[i],ps.p[i+1]),crp);
         ret.ins(crp);
         ret.ins(ps.p[i+1]);
         continue;
      }
      if(xm1>-EPS) ret.ins(ps.p[i]);
      if(xm2>-EPS) ret.ins(ps.p[i+1]);
   }
   if(!ret.size) return ret;
   if(ret.size==1) ret.p[ret.size++]=ret.p[0];
   ret.ins(ret.p[0]);
   return ret;
}
PS hpi(PS ps) {
   PS ret=ps;
   for(int i=0;i<ps.size-1;i++)
      ret=cut(ps.p[i],ps.p[i+1],ret);
   return ret;
}
{
      PS ps;
      scanf("%d",&n);
      for(int i=0;i<n;i++)
         scanf("%lf%lf",&ps.p[i].x,&ps.p[i].y);
      ps.size=n;
      ps.p[ps.size++]=ps.p[0];
      for(int i=0;i<ps.size/2;i++)
         swap(ps.p[i],ps.p[ps.size-i-1]);
      ps=hpi(ps);
      double ans=0;
      for(int i=0;i<ps.size-1;i++)
         ans+=xm(Po(0,0),ps.p[i],ps.p[i+1]);
      printf("%.2lf\n",ans/2);
}

//---------circle operator-----------
// c1’s arc is [rp1,rp2] c2’s arc is [rp2,rp1]
bool inter(Po p1, double r1, Po p2, double r2, Po &rp1, Po &rp2) {
   double cd = dis(p1, p2);
   if (sgn(cd - r1 - r2) >= 0) return false; // no inter area // ’=’ is tangency
   if (sgn(cd - fabs(r2 - r1)) <= 0) return false; // c2 in c1 or c1 in c2
   double l = (cd + (r1 * r1 - r2 * r2) / cd) / 2;
   double h = sqrt(r1 * r1 - l * l);
   rp1 = p1 + (p2 - p1).vect(l)-(p2 - p1).vect(h).left();
   rp2 = p1 + (p2 - p1).vect(l)+(p2 - p1).vect(h).left();
   return true;
}
//p1 X p2 > 0 return intersec-points’number
int circleLineIntersection(Po cp,double r, Po l1,Po l2,Po& p1,Po& p2) {
   Po p=cp+(l2-l1).left(),rp;
   p.x+=l1.y-l2.y;
   p.y+=l2.x-l1.x;
   linecross(Line(p,cp),Line(l1,l2),rp);
   double d=dis(rp,cp);
   if(sgn(d-r)>0) return 0;
   if(sgn(d-r)==0)
   {
      p1=p2=rp;
      return 1;
   }
   double t=sqrt(r*r-d*d);
   p1=rp-(rp-cp).left().vect(t);
   p2=rp+(rp-cp).left().vect(t);
   return 2;
}
{
      inter(p1, r1, p2, r2, rp1, rp2);
      double d=dis(p1, p2);
      if (sgn(d - r1 - r2) >= 0)
      {
         puts("0.000");
         continue;
      }
      if (sgn(d+r2 - r1)<=0)
      {
         printf("%.3lf\n",PI*r2*r2);
         continue;
      }
      if (sgn(d+r1 - r2)<=0)
      {
         printf("%.3lf\n",PI*r1*r1);
         continue;
      }// Heron’s formula is recommended
      double bow1=r1*r1*acos((r1*r1+d*d-r2*r2)/(2*r1*d))-xm(p1,rp1,rp2)*0.5;
      double bow2=r2*r2*acos((r2*r2+d*d-r1*r1)/(2*r2*d))-xm(p2,rp2,rp1)*0.5;
      printf("%.3lf\n",bow1+bow2);
}
//---------circle union-----------
struct Ang {
   double deg;
   int dt;
   Po p;
   Ang(double d = 0, int t = 0, Po po = Po(0, 0)) {
      deg = d;
      dt = t;
      p = po;
   }
   bool operator<(const Ang & a) const {
      return deg < a.deg;
   }
};
double ans[N];//init ans
//n*n*logn
void mcu(Po p[N], double r[N], int &n) { // you’d better remove the same circle
   int rem[N] = {0}, cnt = 0, clude[N] = {0};
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         if (!rem[i] && i != j) {
            double d = dis(p[i], p[j]);
            //if (sgn(d - r[i] + r[j]) <= 0) rem[j] = 1; // cj in ci union optimization
            //if (sgn(d + r[i] - r[j]) <= 0) rem[j] = 1; // ci in cj inter optimization
            if (sgn(d) == 0 && sgn(r[i] - r[j]) == 0) // remove same
               rem[j] = 1;
            if (sgn(d + r[i] - r[j]) <= 0 && !rem[j])
               clude[i]++;
         }
   for (int i = 0; i < n; i++)
      if (!rem[i]) {
         p[cnt] = p[i];
         clude[cnt] = clude[i] + 1;
         r[cnt++] = r[i];
      }
   n = cnt;
   for (int i = 0; i < n; i++) {
      Po rp1, rp2;
      Ang ang[4 * N];
      cnt = 0;
      for (int j = 0; j < n; j++)
         if (i != j) {
            if (!inter(p[i], r[i], p[j], r[j], rp1, rp2)) continue;
            ang[cnt++] = Ang(atan2(rp1.y - p[i].y, rp1.x - p[i].x), 1, rp1);
            ang[cnt++] = Ang(atan2(rp2.y - p[i].y, rp2.x - p[i].x), -1, rp2);
            if (ang[cnt - 2].deg > ang[cnt - 1].deg) { // tangency attention
               ang[cnt++] = Ang(PI, -1, p[i] - Po(r[i], 0));
               ang[cnt++] = Ang(-PI, 1, p[i] - Po(r[i], 0));
            }
         }
      ang[cnt++] = Ang(-PI, clude[i], p[i] - Po(r[i], 0));
      ang[cnt++] = Ang(PI, -clude[i], p[i] - Po(r[i], 0));
      sort(ang, ang + cnt);
      int sum = 0;
      for (int j = 0; j < cnt; j++) {
         if (sum) { //sum = 1 union; sum = n intersec
            ans[sum]+=(ang[j].deg-ang[j-1].deg)*r[i]*r[i]-xm(p[i],ang[j-1].p,ang[j].p);// bow
            ans[sum]+=xm(Po(0, 0), ang[j - 1].p, ang[j].p);
         }
         sum += ang[j].dt;
      }
   }
}// 0.5 * ans
```