Template

ronaflx@Glare

November 3, 2011

Contents

1 string

1.1 ACAutoMachine

```
#define code(ch) ((ch) - ’a’)
const int KIND = 26;
const int MAXN = 500000;
struct node
{
   node* next[KIND], *fail;
   int count;
} pool[MAXN], *pp, *root, *q[MAXN];
int head, tail;
node *newNode()
{
   pp->fail = NULL;
   pp->count = 0;
   memset(pp->next, 0, sizeof (pp->next));
   return pp++;
}
void initialize()
{
   pp = pool;
   root = newNode();
}
void insert(char *str)
{
   node *now = root;
   while (*str)
   {
      int idx = code(*str);
      if (now->next[idx] == NULL)
         now->next[idx] = newNode();
      now = now->next[idx];
      str++;
   }
   now->count++;
}
void buildFail(node*& now, int ith)//build now’s ith son
{
   if(now == root) now->next[ith]->fail = root;
   node* tmp = now->fail;
   while(tmp)
   {
      if(tmp->next[ith] != NULL)
      {
         now->next[ith]->fail = tmp->next[ith];
         return;
      }
      tmp = tmp->fail;
   }
   if(tmp == NULL) now->next[ith]->fail = root;
}
void build()
{
   head = tail = 0;
   q[tail++] = root;
   while (head != tail)
   {
      node *beg = q[head++];
      for (int i = 0; i < KIND; i++)
      {
         if (beg->next[i] == NULL) continue;
         buildFail(beg, i);
         q[tail++] = beg->next[i];
      }
   }
}
node* goStatus(node* now, int ith)
{
   while(now->next[ith] == NULL && now != root)
      now = now->fail;
   now = now->next[ith];
   return now == NULL ? root : now;
}
int query(char* str)
{
   int cnt = 0;
   node* p = root, *tmp;
   while (*str)
   {
      tmp = p = goStatus(p, code(*str));
      while (tmp != root && tmp->count != -1)
      {
         cnt += tmp->count;
         tmp->count = -1;
         tmp = tmp->fail;
      }
      str++;
   }
   return cnt;
}
```

1.2 KMP

```
//be careful with mod string and main string
void prefix(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(char *main, 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;
}
```

1.3 ELFhash

```
class hash_table
{
public:
   unsigned int ELFhash(char *key)
   {
      unsigned int h = 0, g;
      while (*key)
      {
         h = (h << 4) + *key++;
         g = h & 0xf0000000L;
         if (g) h ^= g >> 24;
         h &= ~g;
      }
      return h % MOD;
   }
   int find(char * str, int judge = 0)
   {
      int t = ELFhash(str);
      for (hashCell * i = g[t]; i != NULL; i = i->next)
         if (!strcmp(i->str, str))
            return i->p = judge ? i->p + 1: i->p;
      return 0;
   }

   void insert(char* str, int p)
   {
      if(find(str, 1)) return;
      unsigned t = ELFhash(str);
      strcpy(pp->str, str);
      pp->p = p;
      pp->next = g[t];
      g[t] = pp++;
   }
private:
   const static int MOD = 387173;
   const static int SIZE = 380001;
   struct hashCell
   {
      char str[20];
      int p;
      hashCell * next;
   } pool[SIZE], *g[MOD], *pp;
};
```

1.4 Minimum Representation

```
int Minimum_Representation(char *s, int len)
{
   int i = 0, j = 1, k = 0, t;
   while(i < len && j < len && k < len)
   {
      t = s[(i + k) % len] - s[(j + k) % len];
      if(t == 0) k++;
      else
      {
         if(t < 0) j += k + 1;
         else i += k + 1;
         if(i == j) j++;
         k = 0;
      }
   }
   return min(i, j);
}
```

1.5 C++replaceAll

```
string& replace_all(string& str, const string& src, const string& dest)
{
   string::size_type pos(0);
   while((pos = str.find(src)) != string::npos)
      str.replace(pos,src.length(), dest);
   return str;
}
string& replace_all_distinct(string& str, const string& src, const string& dest
    )
{
   for(string::size_type pos(0);(pos = str.find(src, pos)) != string::npos;pos
       += dest.length())
      str.replace(pos, src.length(), dest);
   return str;
}
```

1.6 Base64

```
static const std::string base64_chars =
          "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
          "abcdefghijklmnopqrstuvwxyz"
          "0123456789+/";


static inline bool is_base64(unsigned char c) {
  return (isalnum(c) || (c == '+') || (c == '/'));
}

std::string base64_encode(unsigned char const* bytes_to_encode, unsigned int
     in_len) {
  std::string ret;
  int i = 0;
  int j = 0;
  unsigned char char_array_3[3];
  unsigned char char_array_4[4];

  while (in_len--) {
    char_array_3[i++] = *(bytes_to_encode++);
    if (i == 3) {
      char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
      char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0
          xf0) >> 4);
      char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0
          xc0) >> 6);
      char_array_4[3] = char_array_3[2] & 0x3f;

      for(i = 0; (i <4) ; i++)
        ret += base64_chars[char_array_4[i]];
      i = 0;
    }
  }

  if (i)
  {
    for(j = i; j < 3; j++)
      char_array_3[j] = '\0';

    char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
    char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0
        ) >> 4);
    char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0
        ) >> 6);
  char_array_4[3] = char_array_3[2] & 0x3f;

  for (j = 0; (j < i + 1); j++)
   ret += base64_chars[char_array_4[j]];

  while((i++ < 3))
   ret += ’=’;

 }

 return ret;

}

std::string base64_decode(std::string const& encoded_string) {
 int in_len = encoded_string.size();
 int i = 0;
 int j = 0;
 int in_ = 0;
 unsigned char char_array_4[4], char_array_3[3];
 std::string ret;

 while (in_len-- && ( encoded_string[in_] != ’=’) && is_base64(encoded_string[in_])) {
  char_array_4[i++] = encoded_string[in_]; in_++;
  if (i ==4) {
   for (i = 0; i <4; i++)
    char_array_4[i] = base64_chars.find(char_array_4[i]);

   char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1] & 0x30) >> 4);
   char_array_3[1] = ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] & 0x3c) >> 2);
   char_array_3[2] = ((char_array_4[2] & 0x3) << 6) + char_array_4[3];

   for (i = 0; (i < 3); i++)
    ret += char_array_3[i];
   i = 0;
  }
 }

 if (i) {
  for (j = i; j <4; j++)
   char_array_4[j] = 0;

  for (j = 0; j <4; j++)
   char_array_4[j] = base64_chars.find(char_array_4[j]);

  char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1] & 0x30) >> 4);
  char_array_3[1] = ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] & 0x3c) >> 2);
  char_array_3[2] = ((char_array_4[2] & 0x3) << 6) + char_array_4[3];

  for (j = 0; (j < i - 1); j++) ret += char_array_3[j];
 }

 return ret;
}
```

1.7 Trie Tree

```
//trie
const int SIZE = 33 * 100000;
const int KIND = 26;
```

```
struct node
{
   node *child[KIND];
   int final;
} pool[SIZE],*root,*last;
void build()
{
   last = root = pool;
   memset(pool, 0, sizeof(pool));
}
void insert(char *from)
{
   node * p = root;
   for (char *i = from; *i; i++)
   {
      if (p->child[*i - 'a'] == NULL)
         p->child[*i - 'a'] = ++last;
      p = p->child[*i - 'a'];
      p->final++;
   }
}
int query(char *from)
{
   node * p = root;
   for (char *i = from; *i; i++)
   {
      p = p->child[*i - 'a'];
      if (p == NULL) return 0;
   }
   return p->final;
}
```

1.8 Suffix Array

```
const int MAXN = 21000;
struct Sfx
{
   int i;
   int key[2];
   bool operator < (const Sfx& s) const
   {
      return key[0] == s.key[0] ? key[1] < s.key[1] : key[0] < s.key[0];
   }
} sfx[MAXN], temp[MAXN];
int rank[MAXN], bucket[MAXN], height[MAXN];// rank from 0 to n - 1
//锛窑radixSortecond then first
void radixSort(Sfx* in, int n, int idx, Sfx* out)
{
   memset(bucket, 0, sizeof(int) * (n + 1));
   for (int i = 0; i < n; i++)
      bucket[in[i].key[idx]]++;
   for (int i = 1; i <= n; i++)
      bucket[i] += bucket[i - 1];
   for (int i = n - 1; i >= 0; i--)//for down
      out[--bucket[in[i].key[idx]]] = in[i];
}
void buildSA(const char* text, int n)
{
   for (int i = 0; i < n; i++)
   {
      sfx[i].i = sfx[i].key[1] = i;
      sfx[i].key[0] = text[i];
   }
   sort(sfx, sfx + n);
   for (int i = 0; i < n; i++)
      sfx[i].key[1] = 0;
   int wid = 1;
   while (wid < n)
   {
      rank[sfx[0].i] = 0;
      for (int i = 1; i < n; i++)
         rank[sfx[i].i] = rank[sfx[i - 1].i] + (sfx[i - 1] < sfx[i]);
      for (int i = 0; i < n; i++)
      {
         sfx[i].i = i;
         sfx[i].key[0] = rank[i];
         sfx[i].key[1] = i + wid < n ? rank[i + wid]: 0;
      }
      radixSort(sfx, n, 1, temp);
      radixSort(temp, n, 0, sfx);
      wid <<= 1;
   }
}
void calHeight(const char* text, int* rank, int n)
{
   //h[i] = height[rank[i]], h[i] >= h[i - 1] - 1;
   for(int i = 0; i < n; i++)
      rank[sfx[i].i] = i;
   for(int i = 0, k = 0, j; i < n; i++)
   {
      if (rank[i] == 0)
         height[rank[i]] = 0;
      else
      {
         if(k > 0) k-- ;
         for (j = sfx[rank[i] - 1].i; text[i + k] == text[j + k]; k++);
         height[rank[i]] = k;
      }
   }
}
int RMQ[MAXN][20];
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);
}
```

2 data structure

2.1 AVL

```cpp
const int INF = 10000000;
template<typename T>
class AVL
{
public:
   AVL()
   {
      pp = pool;
      TMP = node(0, 0, NULL, NULL);
      MYNULL = &TMP;
      roof = MYNULL;
   }
   void clear()
   {
      roof = MYNULL;
      pp = pool;
   }
   int size(){return roof->size;}
   void insert(T k){insert(roof, k);}
   void erase(T k){erase(roof, k);}
   bool empty(){return roof == MYNULL;}
   int findK(int k)
   {
      if (k <= 0)return -INF;
      return findK(roof, k);
   }
private:
#define max(a,b) ((a) < (b) ? (b) : (a))
   struct node
   {
      node *lchild, *rchild;
      T value;
      int h, size;
      node() {}
      node(int h, int size, node * lchild, node * rchild)
      {
         this->size = size, this->h = h;
         this->lchild = lchild, this->rchild = rchild;
      }
   };
   node* roof;
   static const int N = 100000;
   node* MYNULL, TMP, pool[N], *pp;
   int findK(node* &R, int k)
   {
      if (k == R->lchild->size + 1)
         return R->value;
      else if (k <= R->lchild->size)
         return findK(R->lchild, k);
      else if (k > R->size - R->rchild->size)
         return findK(R->rchild, k + R->rchild->size - R->size);
   }
   void fix(node* &R)
   {
      R->h = max(R->rchild->h, R->lchild->h) + 1;
      R->size = R->rchild->size + R->lchild->size + 1;
   }
   void rightsinglerotate(node* &R)
   {
      node * lc = R->lchild;
      R->lchild = lc->rchild;
      fix(R);
      lc->rchild = R;
      R = lc;
      fix(R);
   }
   void leftsinglerotate(node* &R)
   {
      node * rc = R->rchild;
      R->rchild = rc->lchild;
      fix(R);
      rc->lchild = R;
      R = rc;
      fix(R);
   }
   void leftdoublerotate(node* &R)
   {
      rightsinglerotate(R->rchild);
      leftsinglerotate(R);
   }
   void rightdoublerotate(node* &R)
   {
      leftsinglerotate(R->lchild);
      rightsinglerotate(R);
   }
   void maintain(node* &R)
   {
      if (R->lchild != MYNULL)
      {
         if (R->lchild->lchild->h == R->rchild->h + 1)
            rightsinglerotate(R);
         else if (R->lchild->rchild->h == R->rchild->h + 1)
            rightdoublerotate(R);
      }
      if (R->rchild != MYNULL)
      {
         if (R->rchild->rchild->h == R->lchild->h + 1)
            leftsinglerotate(R);
         else if (R->rchild->lchild->h == R->lchild->h + 1)
            leftdoublerotate(R);
      }
   }
   void insert(node* &R, T value)
   {
      if (R == MYNULL)
      {
         R = mynew(value);
         return;
      }
      else if (value <= R->value)
         insert(R->lchild, value);
      else if (value > R->value)
         insert(R->rchild, value);
      fix(R);
      maintain(R);
   }
   void erase(node* &R, T value)
   {
      if (R == MYNULL)
         return;
      if (R->value == value)
      {
         if (R->rchild == MYNULL)
         {
            node * tmp = R;
            R = tmp->lchild;
         }
```

```
         else
         {
            node *tmp = R->rchild;
            while (tmp->lchild != MYNULL)
               tmp = tmp->lchild;
            R->value = tmp->value;
            erase(R->rchild, tmp->value);
            fix(R);
         }
         return;
      }
      else if (value < R->value)
         erase(R->lchild, value);
      else if (value < R->value)
         erase(R->rchild, value);
      fix(R);
      maintain(R);
   }
   node* mynew(T value)
   {
      pp->lchild = pp->rchild = MYNULL;
      pp->size = pp->h = 1;
      pp->value = value;
      return pp++;
   }
#undef max
};
```

2.2 Splay

```
//be careful with pushDown
template<typename T = int>
class Splay
{
private:
#define SIZE(x) ((x) ? (x)->size : 0)
#define SUM(x) ((x) ? (x)->sum : 0)
#define VAL(x) ((x) ? (x)->val : 0)
#define CENTRE (root->ch[1]->ch[0])
   struct node
   {
      int size, sum, add;
      node* ch[2], *pre;
      T v;
      node(T v = T(), int size = 1, node* l = NULL,
          node* r = NULL, node* pre = NULL)
      {
         this->v = v;
         sum = add = 0;
         this->size = size;
         this->pre = pre;
         ch[0] = l, ch[1] = r;
      }
   };
   node * root;
   void pushDown(node*& x)
   {
// node* y = x->ch[0];
// if(y)
// {
//  y->add += x->add;
//  y->sum += SIZE(y) * x->add;
// }
// y = x->ch[1];
// if(y)
// {
//  y->add += x->add;
//  y->sum += SIZE(y) * x->add;
// }
// x->add = 0;
   }

   void pushUp(node*& x)
   {
      x->size = SIZE(x->ch[0]) + SIZE(x->ch[1]) + 1;
      x->sum = SUM(x->ch[0]) + SUM(x->ch[1]) + x->add;
   }
   void rotate(node* x, int type)
   {
      node *y = x->pre;
      pushDown(y);
      pushDown(x);
      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[0] == y) y->pre->ch[0] = x;
         else y->pre->ch[1] = x;
      }
      x->ch[type] = y, y->pre = x;
      if (y == root) root = x;
      pushUp(y);
      pushUp(x);
   }
   void insert(node* &R, T v = T(), node* f = NULL)
   {
      if (R == NULL)
      {
         R = new node(v, 1, NULL, NULL, f);
         splay(R, NULL);
         return;
      }
      else if (v <= R->v) insert(R->ch[0], v, R);
      else if (v > R->v) insert(R->ch[1], v, R);
   }
   void clear(node*& root)
   {
      if(root == NULL) return;
      clear(root->ch[0]);
      clear(root->ch[1]);
      delete root;
      root = NULL;
   }
   node* join(node*& x, node*& y)
   {
      if(x == NULL) return y;
      if(y == NULL) return x;
      x->pre = y->pre = NULL;
      node* z = x;
      while(z->ch[1] != NULL) z = z->ch[1];
      splay(z, NULL);
      z->ch[1] = y;
      y->pre = z;
      pushDown(z);
      return z;
   }
   void splay(node* x, node* f)
   {
      pushDown(x);
      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, *z = y->pre;
            if (z->ch[0] == y)
            {
               if (y->ch[0] == x) // l
                  rotate(y, 1), rotate(x, 1);
               else // z
                  rotate(x, 0), rotate(x, 1);
            }
            else
            {
               if (y->ch[1] == x) // l
                  rotate(y, 0), rotate(x, 0);
               else // z
                  rotate(x, 1), rotate(x, 0);
            }
         }
      }
      pushUp(x);
   }
   node* MaxOrMin(node* x, int type)//0 minimum 1 maximum
   {
      if(x == NULL) return x;
      while(x->ch[type] != 0) x = x->ch[type];
      return x;
   }
public:
   Splay(): root(NULL) {}
   void insert(T v)
   {
      insert(root, v);
   }
   int size()
   {
      return SIZE(root);
   }
   void clear()
   {
      clear(root);
   }
   node* MaxOrMin(int type)
   {
      return MaxOrMin(root, type);
   }
   int rank(T v)
   {
      node* x = find(root, v);
      return SIZE(x) + 1;
   }
   node* selectK(int k)
   {
      node* x = root;
      while(x != NULL && SIZE(x->ch[0]) + 1 != k)
      {
         if(k <= SIZE(x->ch[0])) x = x->ch[0];
         else
         {
            k -= SIZE(x->ch[0]) + 1;
            x = x->ch[1];
         }
      }
      if(x != NULL) splay(x, NULL);
      return x;
   }
   node* find(T v)
   {
      node* x = root;
      while(x != NULL && x->v != v)
         x = x->ch[v > x->v];
      return x;
   }
   void erase(T v)
   {
      node* x = find(v);
      if(x == NULL) return;
      splay(x, NULL);
      root = join(x->ch[0], x->ch[1]);
      if(root != NULL)
         root->pre = NULL;
      delete x;
   }
   node* PreOrSuc(T v, int type)// 0 predecessor 1 successor
   {
      node* x = find(v);
      if(x == NULL) return NULL;
      else return MaxOrMin(x->ch[type], !type);
   }
   void update(int l, int r, int c)
   {
      node* ll = find(l - 1);
      splay(ll, NULL);
      node* rr = find(r + 1);
      splay(rr, root);
      if(root->ch[1] == NULL || CENTRE == NULL) return;
      CENTRE->add += c;
      CENTRE->sum += c;
   }
   int query(int l, int r)
   {
      node* ll = find(l - 1);
      splay(ll, NULL);
      node* rr = find(r + 1);
      splay(rr, root);
      if(root->ch[1] == NULL || CENTRE == NULL) 0;
      return CENTRE->sum;
   }
};
Splay<int> spl;
int main()
{
   int n, t;
   int a, b, cases = 1;
   char cmd[100];
   scanf("%d", &t);
   while(t-- && scanf("%d", &n) == 1)
   {
      spl.clear();
      for(int i = 0; i <= n + 1; i++) spl.insert(i);
      for(int i = 1; i <= n; i++)
      {
         scanf("%d", &a);
         spl.update(i, i, a);
      }
      printf("Case %d:\n", cases++);
      while(scanf("%s", cmd) == 1 && strcmp(cmd, "End"))
      {
         scanf("%d %d", &a, &b);
         if(!strcmp(cmd, "Query"))
         {
            printf("%d\n", spl.query(a, b));
         }
         else if(!strcmp(cmd, "Add"))
         {
            spl.update(a, a, b);
         }
         else
         {
            spl.update(a, a, -b);
         }
      }
   }
   return 0;
}
```

2.3 LCA(RMQ)

```
const int SIZE = 10000;
int color[SIZE], interval[SIZE * 2 + 1], col;
int first[SIZE], depth[SIZE], uncolor[SIZE];
//color records the color of every node
vector<int> adj[SIZE];bool check[SIZE];
int n, pt, q, tim, dp[SIZE * 2 + 1][15];
void dfs(int x, int d)
{
   check[x] = true;
   interval[pt++] = col;
   first[col] = pt - 1;
   color[x] = col;
   uncolor[col++] = x;
   depth[x] = d;
   int s = adj[x].size(), w;
   for (int i = 0; i < s; i++)
   {
      w = adj[x][i];
      if (!check[w])
      {
         dfs(w, d + 1);
         interval[pt++] = color[x];
      }
   }
}
void initialize()
{
   scanf("%d %d", &n, &q);
   int a, b;
   for (int i = 0; i < n; i++)
      adj[i].clear();
   for (int i = 1; i < n; i++)
   {
      scanf("%d %d", &a, &b);
      adj[a].push_back(b);
      adj[b].push_back(a);
   }
   tim = pt = col = 0;
   memset(check, false, sizeof (check));
}
int main()
{
   int t;
   scanf("%d", &t);
   while (t--)
   {
      initialize();
      dfs(0, 1);
      for (int i = 0; i < pt; i++)
         dp[i + 1][0] = interval[i];
      for (int j = 1; j <= log(pt) / log(2); j++)
      {
         for (int i = 1; i + (1 << j) - 1 <= pt; i++)
         {
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
         }
      }
      while (q--)
      {
         int a, b, len, ans;
         scanf("%d %d", &a, &b);
         int tempa = a,tempb = b;
         a = first[color[a]] + 1;
         b = first[color[b]] + 1;
         if (a > b)
         {
            int temp = a;
            a = b;
            b = temp;
         }
         len = b - a + 1;
         len = log(len) / log(2) + 0.00001;
         ans = min(dp[a][len], dp[b - (1 << len) + 1][len]);
         printf("%d\n",depth[tempa] + depth[tempb] - 2 * depth[uncolor[ans]]);
      }
   }
   return 0;
}
```

2.4 leftist Tree

```
#define CMP(a, b) ((a) > (b))
#define DIST(v) ((v == NULL) ? -1 : (v->dist))
//must be careful when clear after merge
//because of the pointer could not be NULL
//especially when use new just makeNULL when memory is enough
template<typename T>
class leftist_tree
{
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;
   node* merge(node* &left, node* &right)
   {
      if(left == NULL) return right;
      if(right == NULL) return left;
      if(CMP(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:
   leftist_tree(){root = NULL;s = 0;}
   ~leftist_tree(){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(leftist_tree<T>& tree)
   {
      this->root = merge(root, tree.root);
      s += tree.s;
      tree.root = NULL;
   }
   void makeNULL(){root = NULL;}
};
```

2.5 partition Tree

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

2.6 tree Representation

2.7 UFS

```
int father[N], rank[N];
int find_set(int x)
{
   return father[x] = father[x] == x ? x :find_set(father[x]);
}
bool union_set(int x,int y)
{
   if(x != y)
   {
      if(rank[x] < rank[y]) father[x] = y;
      else
      {
         rank[x] += rank[x] == rank[y];
         father[y] = x;
      }
      return false;
   }
   return true;
}
bool link_set(int x, int y)
{
   return union_set(find_set(x), find_set(y));
}

void initialize(int n)
{
   for(int i = 1;i <= n;i++)
   {
      rank[i] = 0;
      father[i] = i;
   }
}
```

2.8 BIT_findK

```
/* make sure that the sum is not lower than k*/
int findK(int K)
{
   int ans = 0, cnt = 0;
   for (int i = log(MAXN - 1) / log(2); i >= 0; i--)
   {
      ans += (1 << i);
      if (ans >= MAXN || cnt + c[ans] >= K)
         ans -= (1 << i);
      else
         cnt += c[ans];
   }
   return ans + 1;
}
```

2.9 UFS_value

```
/* HOJ cube stacking*/
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
const int SIZE = 300001;
int cnt[SIZE], dis[SIZE], father[SIZE];

void initialize()
{
   for (int i = 0; i < SIZE; i++)
   {
      cnt[i] = 1;
      dis[i] = 0;
      father[i] = i;
   }
}

int set_find(int x)
{
   if (x != father[x])
   {
      int f = father[x];
      father[x] = set_find(father[x]);
      dis[x] += dis[f];
   }
   return father[x];
}
int main()
{
   int t, x, y;
   char ch;
   while (scanf("%d", &t) == 1)
   {
      initialize();
      while (t--)
      {
         scanf(" %c", &ch);
         if (ch == 'M')
         {
            scanf("%d %d", &x, &y);
            int f1 = set_find(x), f2 = set_find(y);
            if (f1 != f2)
            {
               dis[f2] = cnt[f1];
               father[f2] = f1;
               cnt[f1] += cnt[f2];
            }
         }
         else if (ch == 'C')
         {
            scanf("%d", &x);
            int f = set_find(x);
            printf("%d\n", cnt[f] - dis[x] - 1);
         }
      }
   }
   return 0;
}
```

2.10 3D_BIT

```
/*3D BIT escape index 0*/
const int size = 201;
int c[size][size][size], n;
int lowbit(int k)
{
   return k & (k ^(k - 1));
}
int sum(const int x1, const int y1, const int z1)
{
   int s = 0;
   for(int i = x1;i > 0;i -=lowbit(i))
      for(int j = y1;j > 0;j -= lowbit(j))
         for(int k = z1;k > 0;k -= lowbit(k))
            s += c[i][j][k];
   return s;
}

void modify(const int x1, const int y1, const int z1, int val)
{
   for(int i = x1;i < size;i +=lowbit(i))
      for(int j = y1;j < size;j += lowbit(j))
         for(int k = z1;k < size;k += lowbit(k))
            c[i][j][k] += val;
}
/* scanf("%d %d %d %d %d %d %d", &a, &b, &d, &x, &y, &z,&val);
 * modify(a, b, d, val); modify(x + 1, b, d, -val);
 * modify(a, y + 1, d, -val); modify(x + 1, y + 1, d, val);
 * modify(a, b, z + 1, -val); modify(a, y + 1, z + 1, val);
 * modify(x + 1, b, z + 1, val); modify(x + 1, y + 1, z + 1, -val);*/
```

2.11 DLX(exact)

```
const int SIZE = 16;//here
const int 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];
int 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;
}
```

2.12 DLX(indistinct)

```
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;
         // row[size]=i;
         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 graph

3.1 BBC

```
/* 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;
}
```

3.2 EBBC

```
/*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();
   }
}
```

3.3 scc

```
/* tarjan-scc, new graph is a dag from 0 to sccCnt - 1
tms is the topo-order*/
const int V = 50001, E = 150000;
struct edge//graph
{
   int v;
   edge *nxt;
} pool[E * 3], *g[V], *pp, *gscc[V];
int st[V], top, tms[V], pt;//toposort
bool reach[V];//reach is used to label is reached or not
int dfn[V], low[V], idx[V], sccCnt, depth;
void addedge(int u, int v, edge* g[])
{
   pp->v = v;
   pp->nxt = g[u];
   g[u] = pp++;
}
void initialize(int n)
{
   memset(g, 0, sizeof (g));
   memset(reach, false, sizeof (reach));
   memset(dfn, 0, sizeof (dfn));
   pp = pool, depth = pt = top = sccCnt = 0;
}
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 x)
{
   reach[x] = true;
   for (edge *i = gscc[x]; i != NULL; i = i->nxt)
      if (!reach[i->v]) toposort(i->v);
   tms[pt++] = x;
}

void build_newgraph(int n)
{
   memset(gscc, 0, sizeof (gscc));
   for (int i = 0; i < n; i++)
      for (edge *j = g[i]; j != NULL; j = j->nxt)
         if (idx[i] != idx[j->v])addedge(idx[i],idx[j->v], gscc);
}
void solve(int n)
{
   for (int i = 0; i < n; i++)
      if (!reach[i]) dfs(i);
   build_newgraph(n);
   memset(reach, false, sizeof (reach));//reuse reach
   for (int i = 0; i < sccCnt; i++)
      if (!reach[i]) toposort(i);
   reverse(tms, tms + pt);//Topological Sort
}
```

3.4 2-sat

```
#include <iostream>
using namespace std;
/* 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;
}
```

3.5 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;
}
```

3.6 hungary

```
/*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 slove(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;
}
```

3.7 KM

```
/* val must be positive
 * min match use INF - val
 * must build a matrix[V][V]*/
const int V = 100;
const int INF = 100000;
int val[V][V], lx[V], ly[V], my[V];
bool visx[V], visy[V];
void initialize(int n)
{
   CC(val, 0), CC(ly, 0), CC(my, -1);
   fill(lx, lx + n, -INF);
}
bool find_path(int x, const int n)
{
   visx[x] = true;
   for(int i = 0; i < n; i++)
   {
      if(!visy[i] && lx[x] + ly[i] == val[x][i])
      {
         visy[i] = true;
         if(my[i] == -1 || find_path(my[i], n))
         {
            my[i] = x;
            return true;
         }
      }
   }
   return false;
}
int solve(int n)
{
   for(int i = 0; i < n; i++)
      lx[i] = *max_element(val[i], val[i] + n);
   int dx, sum = 0;
   for(int i = 0; i < n; i++)
   {
      while(true)
      {
         CC(visx, 0), CC(visy, 0);
         if(find_path(i, n)) break;
         dx = INF;
         for(int j = 0; j < n; j++)
         {
            if(!visx[j]) continue;
            for(int k = 0; k < n; k++)
            {
               if(visy[k]) continue;
               dx = min(dx, lx[j] + ly[k] - val[j][k]);
            }
         }
         for(int j = 0; j < n; j++)
         {
            if(visx[j]) lx[j] -= dx;
            if(visy[j]) ly[j] += dx;
         }
      }
   }
   for(int i = 0; i < n; i++)
      sum += INF - val[my[i]][i];
   return sum;
}
```

3.8 stableMarriage

```
/* 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));
}
```

3.9 maximal Clique

```
/* 求无向图极大团的个数
 * 极大团就一个被不被其他的完全子图包含的完全子图
 * 最大团一定是一个极大团，但是极大团不一定是最大团
*/
class Bron_Kerbosch
{
private:
   const static int N = 130;
   int n, maps[N][N], cnt;
   void countClique(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;
         }
         countClique(tmpp, tmpps, tmpx, tmpxs);
         if(cnt > 1000)
            return;
         x[xs++] = v;
      }
   }
public:
   void initialize(int n,int m)
   {
      memset(maps, 0, sizeof(maps));
      this->n = n;
      for(int i = 0;i < m;i++)
      {
         int a, b;
         scanf("%d %d", &a, &b);
         a--, b--;
         maps[a][b] = true;
         maps[b][a] = true;
      }
   }
   int countClique()
   {
      cnt = 0;
      int p[N], x[N];
      for(int i = 0;i < n;i++)
         p[i] = i;
      countClique(p, n, x, 0);
      return cnt;
   }
}one ;
```

3.10 MaxClique

```
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;
}
```

3.11 minimum Cut

```
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_Wgner(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;
}
```

3.12 LCA

```
锘縖
onst 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));
}
```

3.13 bellman-ford

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <vector>
using namespace std;
const int SIZE = 10110;
const int INF = 100000000;
vector<pair<pair<int, int>,int> >edge;
int n, a0, a1, b0, b1, l0, l1;
int d[SIZE];

void Bellman_ford(int n,int p)
{
   fill(d,d + n, INF);
   d[n - 1] = 0;
   for(int i = 1;i < n;i++)
   {
      bool unfind = true;
      FOREACH(edge, i)
      {
         int v = i->first.first, u = i->first.second, val = i->second;
         if(d[u] > d[v] + val)
         {
            unfind = false;
            d[u] = d[v] + val;
         }
      }
      if(unfind)
         break;
   }
   FOREACH(edge, i)
   {
      int v = i->first.first, u = i->first.second, val = i->second;
      if(d[u] > d[v] + val)
      {
         puts("-1");
         return;
      }
   }
   for(int i = 1;i <= p;i++)
      printf("%d", d[i] - d[i - 1]);
   puts("");
}
int main()
{
   while(scanf("%d%d%d%d%d%d%d", &n, &a0, &b0, &l0, &a1, &b1, &l1) == 7)
   {
      edge.clear();
      for(int i = 1;i + l0 - 1 <= n;i++)
      {
         edge.push_back(make_pair(make_pair(i-1,i+l0-1),l0-a0));
         edge.push_back(make_pair(make_pair(i+l0-1,i-1),b0 - l0));
      }
      for(int i = 1;i + l1 - 1 <= n;i++)
      {
         edge.push_back(make_pair(make_pair(i-1,i+l1-1),b1));
         edge.push_back(make_pair(make_pair(i+l1-1,i-1),-a1));
      }
      for(int i = 1;i <= n;i++)
      {
         edge.push_back(make_pair(make_pair(i - 1, i), 1));
         edge.push_back(make_pair(make_pair(i, i - 1), 0));
      }
      for(int i = 0;i <= n;i++)
         edge.push_back(make_pair(make_pair(n + 1, i), 0));
      Bellman_ford(n + 2, n);
   }
   return 0;
}
```

3.14 Eular-fleury

```
锘綍
onst int SIZE = 2 * 2000;
const int 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];
int 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()
{
   /*if (!check())
      printf("Round trip does not exist.\n");
   else*/
   {
      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]);
}
```

3.15 k-shortest-path(with cycle)

```
/* 估价函数f(x)=g(x)+h(x);h(x)=h*(x)所以符合;A条件*,短路含环k
 * 中为statusfA的估价函数。*f(x2)=f(x1)-h(x1)+h(x2)+c(x1)(x2);*/
const int V = 1000, E = 100000, INF = 100000000;
struct status
{
   int v, f;
   status() {}
   status(int _v, int _f)
   {v = _v, f = _f;}
   bool operator <(const status a)const
   {return f > a.f;}
};

struct edge
{
   int v, dist;
   edge *nxt;
}*pp, *g[V], *rg[V], pool[E * 2];
int d[V], c[V];
bool chk[V];
void initialize()
{
   pp = pool;
   memset(g, 0, sizeof(g));
   memset(rg, 0, sizeof(rg));
}

void addedge(int u, int v,int dist, edge *g[])
{
   pp->v = v;
   pp->dist = dist;
   pp->nxt = g[u];
   g[u] = pp++;
}

void dijkstra(int n, int t)
{
   fill(d, d + n, INF);
   memset(chk, false, sizeof(chk));
   priority_queue<status> pq;
   pq.push(status(t, 0));
   while(!pq.empty())
   {
      status now = pq.top();
      pq.pop();
      if(chk[now.v]) continue;
      chk[now.v] = true;
      d[now.v] = now.f;
      for(edge *i = rg[now.v]; i != NULL; i = i->nxt)
         pq.push(status(i->v, now.f + i->dist));
   }
}

int Astar(int s, int t, int k)
{
   if(d[s] == INF) return -1;
   memset(c, 0, sizeof(c));
   priority_queue<status> pq;
   pq.push(status(s, d[s]));
   while(!pq.empty())
   {
      status now = pq.top();
      pq.pop();
      c[now.v]++;
      if(c[t] == k) return now.f;
      if(c[now.v] > k) continue;
      for(edge *i = g[now.v]; i!= NULL; i = i->nxt)
         pq.push(status(i->v, now.f - d[now.v] + d[i->v] + i->dist));
   }
   return -1;
}

int main()
{
   int n, m, s, t, k;
   int u, v, dist;
   while(scanf("%d %d", &n, &m) == 2)
   {
      initialize();
      for(int i = 0; i < m; i++)
      {
         scanf("%d %d %d", &u, &v, &dist);
         u--, v--;
         addedge(u, v, dist, g);
         addedge(v, u, dist, rg);
      }
      scanf("%d %d %d", &s, &t, &k);
      s--, t--;
      if(s == t) k++;
      dijkstra(n, t);
      printf("%d\n", Astar(s, t, k));
   }
   return 0;
}
```

3.16 secondSP

```
/*由最短路推出答案,算法变形dijkstra
* 本题求了最短路以及比最短路大的路的条数。1
* poj 3463 Sightseeing
* 本模板允许向回走，再次走到原来经过的点
* 如果要求不课重复走的次短路，记录最短路前驱
* 然后枚举删掉最短路上的边，再求最短路取最小值便是次短路*/
const int SIZE = 1010;
const int INF = 1100000000;
class Edge
```

```
{
public:
   int v, cost, pt;

   bool operator<(const Edge a) const
   {
      return cost > a.cost;
   }
} tmp, nxt, beg;
vector <Edge> adj[SIZE];
int dist[SIZE][2], cnt[SIZE][2];
int n, m;
bool check[SIZE][2];

int dijkstra(int start, int end)
{
   memset(check, false, sizeof (check));
   memset(cnt, 0, sizeof (cnt));
   for (int i = 1; i <= n; i++)
      dist[i][0] = dist[i][1] = INF;
   priority_queue<Edge> pq;
   beg.v = start, beg.pt = 0, beg.cost = 0;
   pq.push(beg);
   cnt[start][0] = 1;
   dist[start][0] = 0;
   while (!pq.empty())
   {
      beg = pq.top();
      pq.pop();
      if (check[beg.v][beg.pt])
         continue;
      check[beg.v][beg.pt] = true;
      int s = adj[beg.v].size(), w, cst;
      for (int i = 0; i < s; i++)
      {
         tmp = adj[beg.v][i];
         w = tmp.v, cst = tmp.cost;
         if (dist[w][0] > dist[beg.v][beg.pt] + cst && !check[w][0])
         {
            dist[w][1] = dist[w][0];
            cnt[w][1] = cnt[w][0];
            dist[w][0] = dist[beg.v][beg.pt] + cst;
            cnt[w][0] = cnt[beg.v][beg.pt];
            nxt.v = w, nxt.pt = 0, nxt.cost = dist[w][0];
            pq.push(nxt);
      nxt.v = w,nxt.pt = 1,nxt.cost = dist[w][1];
      pq.push(nxt);
         }
         else if (dist[w][0] == dist[beg.v][beg.pt] + cst && !check[w][0])
         {
            cnt[w][0] += cnt[beg.v][beg.pt];
            nxt.v = w, nxt.pt = 0, nxt.cost = dist[w][0];
            pq.push(nxt);
         }
         else if (dist[w][1] > dist[beg.v][beg.pt] + cst && !check[w][1])
         {
            dist[w][1] = dist[beg.v][beg.pt] + cst;
            cnt[w][1] = cnt[beg.v][beg.pt];
            nxt.pt = 1, nxt.v = w, nxt.cost = dist[w][1];
            pq.push(nxt);
         }
         else if (dist[w][1] == dist[beg.v][beg.pt] + cst && !check[w][1])
         {
            cnt[w][1] += cnt[beg.v][beg.pt];
            nxt.pt = 1, nxt.v = w, nxt.cost = dist[w][1];
            pq.push(nxt);
         }
      }
   }
   int num = cnt[end][0];
   if (dist[end][0] + 1 == dist[end][1])
      num += cnt[end][1];
   return num;
}

int main()
{
   int T;
   scanf("%d", &T);
   while (T--)
   {
      scanf("%d %d", &n, &m);
      for (int i = 1; i <= n; ++i)
         adj[i].clear();
      for (int i = 0; i < m; ++i)
      {
         int a, b, c;
         scanf("%d %d %d", &a, &b, &c);
         beg.v = b, beg.cost = c;
         adj[a].push_back(beg);
      }
      int start, end;
      scanf("%d %d", &start, &end);
      printf("%d\n", dijkstra(start, end));
   }
   return 0;
}
```

3.17 SPFA

```
/* 模板，队列实现SPFA
 * 如果要求负环，设置人工顶点
 * 与已知点连边权值为0
 * 人工顶点先入队列
 * 记录每个点入队列的次数，如果大于则有负环n
 */
const int V = 1010, E = 300001, INF = 10000000;
struct edges
{
   int v, val;
   edges *next;
} pool[E], *g[V], *pp;
int cnt[V], dist[V];
bool vst[V];
void initialize()
{
   memset(g, 0, sizeof (g));
   pp = pool;
}
void addedge(int a, int b, int v)
{
   pp->v = b;
   pp->val = v;
   pp->next = g[a];
   g[a] = pp++;
}
void SPFA(int n)
{
   memset(cnt, 0, sizeof (cnt));
   memset(vst, false, sizeof (vst));
```

```
    fill(dist, dist + n, INF);
    queue<int> q;
    q.push(n - 1);
    dist[n - 1] = 0;
    vst[n - 1] = true;
    while (!q.empty())
    {
       int beg = q.front();
       q.pop();
       vst[beg] = false;
       for (edges * i = g[beg]; i != NULL; i = i->next)
       {
          int tmp = i->v, val = i->val;
          if (dist[tmp] > dist[beg] + val)
          {
             dist[tmp] = dist[beg] + val;
             if (!vst[tmp])
             {
                vst[tmp] = true;
                cnt[tmp]++;
                q.push(tmp);
                if (cnt[tmp] > n)
                {
                   puts("-1");
                   return;
                }
             }
          }
       }
    }
    for(int i = 1;i < n - 1;i++)
       printf("%d", dist[i] - dist[i - 1]);
    puts("");
}
int main()
{
    int n ,a0, b0, l0, a1, b1, l1;
    while(scanf("%d %d %d %d %d %d %d", &n, &a0, &b0, &l0, &a1, &b1, &l1) == 7)
    {
       initialize();
       for(int i = 1;i + l0 - 1 <= n;i++)
       {
          addedge(i - 1, i + l0 - 1, l0 - a0);
          addedge(i + l0 - 1, i - 1, b0 - l0);
       }
       for(int i = 1;i + l1 - 1 <= n;i++)
       {
          addedge(i - 1, i + l1 - 1, b1);
          addedge(i + l1 - 1, i - 1, -a1);
       }
       for(int i = 1;i <= n;i++)
       {
          addedge(i - 1, i, 1);
          addedge(i, i - 1, 0);
       }
       for(int i = 0;i <= n;i++)
          addedge(n + 1, i, 0);
       SPFA(n + 2);
    }
    return 0;
}
```

3.18 Best Radio Spanning Tree

```
const int SIZE = 1001;
const double EXP = 1e-5;
const double INF = 1e10;
double mat[SIZE][SIZE], cost[SIZE][SIZE], val[SIZE][SIZE];
int n;
double x[SIZE], y[SIZE], height[SIZE];
double prim(double low)
{
   bool check[SIZE];
   memset(check, false, sizeof (check));
   check[0] = true;
   double totdis = 0, totcost = 0, dist[SIZE], minn = INF;
   dist[0] = 0;
   for (int i = 1; i < n; i++)
      dist[i] = INF;
   int idx = 0, pree[SIZE], tmp;
   memset(pree, 0, sizeof (pree));
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         val[i][j] = cost[i][j] - low * mat[i][j];
   for (int i = 1; i < n; i++)
   {
      minn = INF;
      for (int j = 1; j < n; j++)
      {
         if (!check[j])
         {
            if (dist[j] > val[idx][j])
            {
               dist[j] = val[idx][j];
               pree[j] = idx;
            }
            if (dist[j] < minn)
            {
               minn = dist[j];
               tmp = j;
            }
         }
      }
      totdis += mat[pree[tmp]][tmp];
      totcost += cost[pree[tmp]][tmp];
      check[tmp] = true;
      idx = tmp;
   }
   return totcost / totdis;
}
double distance(int i, int j)
{
   return sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
}
int main()
{
   while (scanf("%d", &n) == 1 && n)
   {
      for (int i = 0; i < n; i++)
         scanf("%lf %lf %lf", &x[i], &y[i], &height[i]);
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
         {
            mat[i][j] = distance(i, j);
            cost[i][j] = fabs(height[i] - height[j]);
         }
      double low = 0, tmp;
      while (true)
      {
         tmp = prim(low);
```

```
            if (fabs(low - tmp) < EXP)
                break;
            low = tmp;
        }
        printf("%.3lf\n", tmp);
    }
    return 0;
}
```

3.19 Count Spanning Tree

```
const int SIZE = 12;
double guass(int n, double mat[][SIZE])
{
    for (int i = 1; i < n; i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (mat[i][j] == 0)
                continue;
            double kk = mat[i][j] / mat[j][j];
            for (int k = 0; k < n; k++)
                mat[i][k] -= kk * mat[j][k];
        }
    }
    double res = 1.00;
    for(int i = 0;i < n;i++)
        res *= mat[i][i];
    return fabs(res) + 0.005;
}
double maps[SIZE][SIZE];
int main()
{
    int n, t;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
        {
            int cnt = 0;
            for (int j = 0; j < n; j++)
            {
                scanf("%lf", &maps[i][j]);
                if(i == j)
                    maps[i][j] = 0;
                if (maps[i][j] == 1)
                    cnt++;
                maps[i][j] = -maps[i][j];
            }
            maps[i][i] = cnt;
        }
        printf("%.0lf\n",guass(n - 1,maps));
    }
    return 0;
}
```

3.20 Degree Limited Spanning Tree

```
/* 度限制最小生成树为根0 限制度最大k */
const int N = 25;
const int LEN = 15;
const int 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)//node 0 is root
{
   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)//除了点的最小生成森林0
      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)//和点相连0
   {
      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)//保证有解不可能森林大于,个，通过增大度减少树的边权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;
}
```

3.21 second MST

```
/*算法，并查集
kruskal统计了每个点在中的出现次数
mst求次小生成树，是最小生成树的邻集

n求每个点之间的最小边，然后替代之^22
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;
const int SIZE = 101;
const int INF = 0x7fffff;

struct Edge
{
   int v, w, val;

   bool operator<(const Edge a) const
   {
      return val < a.val;
   }
} edge[SIZE * SIZE];
bool check[SIZE],used[SIZE * SIZE];
int father[SIZE], rank[SIZE];
int max_val[SIZE][SIZE],mst;
vector<Edge> adj[SIZE];
int set_find(int x)
{
   return father[x] = father[x] == x ? father[x] : set_find(father[x]);
}

bool set_link(int x,int y)
{
   if (x == y)
      return false;
   if (rank[x] < rank[y])
      father[x] = y;
   else
   {
      father[y] = x;
      rank[x] += rank[x] == rank[y];
   }
   return true;
}

bool set_union(int x, int y)
{
   return set_link(set_find(x), set_find(y));
}
int t, n, m;

void initialize()
{
   scanf("%d %d",&n,&m);
   for (int i = 1; i <= n; i++)
   {
      adj[i].clear();
      father[i] = i;
      rank[i] = 0;
   }
   mst = 0;
   int v, w, val;
   for (int i = 0; i < m; i++)
   {
      scanf("%d %d %d", &v, &w, &val);
      edge[i].v = v;
      edge[i].w = w;
      edge[i].val = val;
   }
   memset(check, false, sizeof (check));
   memset(used,false,sizeof(used));
}
void kruskal()
{
   for (int i = 0; i < m; i++)
   {
      if(set_union(edge[i].v,edge[i].w))
      {
         Edge temp = edge[i];
         adj[edge[i].v].push_back(temp);
         temp.w = temp.v;
         adj[edge[i].w].push_back(temp);
         mst += edge[i].val;
         used[i] = true;
      }
   }
}
void dfs(int now,int pre,int maxn,int val)
{
   check[now] = true;
   max_val[pre][now] = max(maxn,val);
   int s = adj[now].size(),w;
   for(int i = 0;i < s;i++)
   {
      w = adj[now][i].w;
      if(!check[w])
      {
         dfs(w,pre,max_val[pre][now],adj[now][i].val);
      }
   }
}
bool sst()
{
   for(int i = 0;i < m;i++)
   {
      if(!used[i])
      {
         if(edge[i].val == max_val[edge[i].v][edge[i].w])
            return false;
      }
   }
   return true;
}
int main()
{
   scanf("%d", &t);
   while (t--)
   {
      initialize();
      sort(edge, edge + m);
      kruskal();
      for(int i = 1;i <= n;i++)
      {
         int s = adj[i].size(),w;
         memset(check,false,sizeof(check));
         check[i] = true;
         for(int j = 0; j < s;j++)
         {
            w = adj[i][j].w;
            dfs(w,i,0,adj[i][j].val);
         }
      }
      if(sst())
         printf("%d\n",mst);
      else
         printf("Not Unique!\n");
   }
   return 0;
}
```

3.22 minimum direct tree(matrix)

```
锘縞
onst int SIZE = 1001;
const int INF = 1000000000;
double g[SIZE][SIZE], x[SIZE], y[SIZE];
bool visit[SIZE], circle[SIZE];
int pre[SIZE];
double dist(int i, int j)
{
   return sqrt(pow((x[i]-x[j]),2)+pow((y[i]-y[j]),2));
}
void dfs(int t, int n)
{
   if (visit[t]) return;
   visit[t] = 1;
   REP(i, 0, n) if (g[t][i] < INF) dfs(i, n);
}
bool connect(int root, int n)
{//judge contection
   CC(visit, 0);dfs(root, n);
   return accumulate(visit, visit + n, 0) == n;
}
void findMinimumEdge(const int root, int n)
{//find min edge of every edge
   REP(i, 0, n)
   {
      if (circle[i] || i == root) continue;
      pre[i] = i;
      double tmp = INF + 1;
      REP(j, 0, n)
      {
         if (circle[j]) continue;
         if (g[j][i] < tmp && i != j)
         {
            tmp = g[j][i];
            pre[i] = j;
         }
      }
   }
}
int findCrile(const int root, int n)
{
   REP(i, 0, n)
   {
      if(circle[i]) continue;
      int now = i;
      CC(visit, 0);
      while(!visit[now] && now != root)
      {
         visit[now] = true;
         now = pre[now];
      }
      if(now != root) return now;
   }
   return -1;
}
void update(int now, int n)
{
   REP(j, 0, n)
   {
      if (circle[j]) continue;
      if (g[j][now] < INF)
         g[j][now] -= g[pre[now]][now]; //update inEdge
   }
   for (int j = pre[now]; j != now; j = pre[j])
   {
      REP(k, 0, n)
      {
         if (circle[k]) continue;
         if (g[j][k] < INF)//update outEdge
            g[now][k] = min(g[now][k], g[j][k]);
         if (g[k][j] < INF)//update inEdge
            g[k][now] = min(g[k][now], g[k][j] - g[pre[j]][j]);
      }
   }
}
double solve(const int root, int n)
{
   double ans = 0;
   int now;
   memset(circle, 0, sizeof (circle));
   do
   {
      findMinimumEdge(root, n);
      if((now = findCrile(root, n)) != -1)
      {
         ans += g[pre[now]][now];
         for (int j = pre[now]; j != now; j = pre[j])
         {
            ans += g[pre[j]][j];
            circle[j] = 1;
         }
         update(now, n);
      }
      else
      {
         REP(j, 0, n)
         {
            if (circle[j] || j == root) continue;
            ans += g[pre[j]][j];
         }
      }
   }while(now != -1);
   return ans;
}
int main()
{
   int n, m, a, b;
   while (scanf("%d%d", &n, &m) != EOF)
   {
      REP(i, 0, n) scanf("%lf%lf", &x[i], &y[i]);
```

```
        REP(i, 0, n) REP(j, 0, n) g[i][j] = INF;
        REP(i, 0, m)
        {
           scanf("%d%d", &a, &b);
           a--, b--;
           g[a][b] = dist(a, b);
        }
        if (!connect(0, n)) printf("poor snoopy\n");
        else printf("%.2lf\n", solve(0, n));
     }
     return 0;
}
```

3.23 minimum direct tree(pool)

```
const int N = 1010;
const int 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;
         int 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;
}
```

3.24 maxflow

```
#define OP(i) (((i) - (pool))^1)
class sap
{
private:
   const static int V = 20010, E = 1000000, INF = 100000000;
   int dis[V], numdis[V], pre[V], maxflow;;
   bool reachS[V], reachT[V];
   struct edge
   {
      int v, cap;
      edge *nxt;//保存当前弧，存可行流中的边epree
   } pool[E], *g[V], *pp, *e[V], *pree[V];
   void bfs(int v, int n)//从汇点开始按照反向边流量走
   {
      int que[V], tail = 0;
      bool vst[V] = {0};
      memset(numdis, 0, sizeof(numdis));
      fill(dis, dis + n, n);
      dis[v] = 0, vst[v] = 1, que[0] = v;
      for (int j = 0; j <= tail; j++)
      {
         int tmp = que[j % n];
         for (edge *i = g[tmp]; i != NULL; i = i->nxt)
         {
            if (pool[OP(i)].cap > 0 && !vst[i->v])
            {
               tail++;
               vst[i->v] = 1;
               que[tail % n] = i->v;
               dis[i->v] = dis[tmp] + 1;
               numdis[dis[i->v]]++;
            }
         }
      }
   }
   int findArgumentPath(int &v, int s, int t)
   {
      while (e[v] != NULL)
      {
         if (e[v]->cap > 0 && dis[v] == dis[e[v]->v] + 1)
         {
            pre[e[v]->v] = v, pree[e[v]->v] = e[v], v = e[v]->v;
            if (v == t)
            {
               int minf = INF;
               for (int i = t; i != s; i = pre[i])
                  minf = min(minf,pree[i]->cap);
               for (int i = t; i != s; i = pre[i])
               {
                  pree[i]->cap -= minf;
                  pool[OP(pree[i])].cap += minf;
               }
               v = s;
               return minf;
            }
         }
         else e[v] = e[v]->nxt;
      }
      return 0;
   }
public:
   int maxflowsap(int n, int s, int t)
   {
      bfs(t, n);
      int v = s;
      copy(g, g + n, e);
      while (dis[s] < n)//标号为n 表示无可行流
      {
         int add = findArgumentPath(v, s, t);
         maxflow += add;
         if (add == 0)//发现某个点没有允许弧，维护其距离标号v
         {
            int mindis = n;
            numdis[dis[v]]--;
            if (!numdis[dis[v]]) break;//GAP 优化，发现断层直接退出
            for (edge *i = g[v]; i != NULL; i = i->nxt)
               if (i->cap > 0) mindis = min(mindis,dis[i->v] + 1);
            dis[v] = mindis;
            numdis[dis[v]]++;
            e[v] = g[v];//改变距离标号以后从新维护当前弧，回前驱
            if (v != s) v = pre[v];
         }
      }
      return maxflow;
   }
   void firststart()
   {
      pp = pool;
      maxflow = 0;
      memset(g,0,sizeof(g));
      //memset(reachS, 0, sizeof(reachS));
      //memset(reachT, 0, sizeof(reachT));
   }//后两个用于求割等问题
   void addedge(int i, int j, int cap)
   {
      pp->v = j;
      pp->cap = cap;
      pp->nxt = g[i];
      g[i] = pp++;
   }//不自动加反向边，如果i to j j to 都有容量且相邻加入i
   void dfss(int x)
   {
      reachS[x] = true;
      for(edge *i = g[x]; i != NULL ; i = i->nxt)
         if(i->cap && !reachS[i->v]) dfss(i->v);
   }//网络流割S-割出来的集合是从源点正向边遍历到的点集合TS
   void dfst(int x)
   {
      reachT[x] = true;
      for(edge *i = g[x]; i != NULL; i = i->nxt)
         if(pool[OP(i)].cap && !reachT[i->v]) dfst(i->v);
   }//用于求关键边等问题
   //关键边是一端reachS 一端reachT 且无流量的边
}maxflow;
```

3.25 mincostflow

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

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

4 computational geometry

4.1 geometry

```cpp
const double EPS = 1e-8;
const double PI = acos(-1.0);
const double INF = 1e100;
struct Point
{
      double x, y;
      Point(double xx = 0, double yy = 0)
      {x = xx, y = yy;}
      bool operator <(const Point a) const
      {return y == a.y ? x < a.x : y < a.y;}
      friend ostream& operator << (ostream& out, Point a)
      {
            out << "(" << a.x << " " << a.y << ")";
            return out;
      }
};
/* *********************************************
 * 距离公式
 * ********************************************/
double dist(double x1, double y1, double x2, double y2)
{
      return sqrt(pow(x1 - x2, 2.0) + pow(y1 - y2, 2.0));
}
double sphereDist(double x1, double y1, double x2, double y2, double R = 1)
{
      //longitude x and latitude y
      //z[i] = sin(lat[i]*PI/180);
      //x[i] = cos(lng[i]*PI/180) * cos(lat[i]*PI/180);
      //y[i] = sin(lng[i]*PI/180) * cos(lat[i]*PI/180);
      //dist = x[i]*x[j] + y[i]*y[j] + z[i]*z[j]
      x1 /= 180;y1 /= 180;x2 /= 180;y2 /= 180;
      x1 *= PI;y1 *= PI;x2 *= PI;y2 *= PI;
      return R * acos(sin(y1) * sin(y2) + cos(y1) * cos(y2) * cos(x1 - x2));
}

/* *********************************************
 * 基础应用
 * ********************************************/
int dblcmp(double x)
{
      if(fabs(x) < EPS) return 0;
      return x < 0 ? -1: 1;
}
double det(double x1, double y1, double x2, double y2)
{
      return x1 * y2 - x2 * y1;
}
double cross(Point a, Point b, Point c)//ab x ac
{
      return det(b.x - a.x, b.y - a.y, c.x - a.x, c.y - a.y);
}
double dotbet(double x1, double y1, double x2, double y2)
{
      return x1 * x2 + y1 * y2;
}
double dot(Point a, Point b, Point c)
{
      return dotbet(b.x - a.x, b.y - a.y, c.x - a.x, c.y - a.y);
}
int betweencmp(Point a, Point b, Point c)
{
      return dblcmp(dot(a, b, c));
}
/* *********************************************
 * 直线线段相交模板
 * *********************************************/
bool segcrosssimple(Point a, Point b, Point c, Point d)
{//ab 与是否规范相交cd
      return (dblcmp(cross(a, c, d)) ^ dblcmp(cross(b, c, d))) == -2 &&
            (dblcmp(cross(c, a, b)) ^ dblcmp(cross(d, a, b))) == -2;
}
int segcross(Point a, Point b, Point c, Point d, Point& p)
{//ab 是否相交，规范相交返回交点cd
      double s1, s2, s3, s4;
      int d1 = dblcmp(s1 = cross(a, b, c));
      int d2 = dblcmp(s2 = cross(a, b, d));
      int d3 = dblcmp(s3 = cross(c, d, a));
      int d4 = dblcmp(s4 = cross(c, d, b));
      if((d1 ^ d2) == -2 && (d3 ^ d4) == -2)
      {
            p.x = (c.x * s2 - d.x * s1) / (s2 - s1);
            p.y = (c.y * s2 - d.y * s1) / (s2 - s1);
            return 1;
      }
      if(d1 == 0 && betweencmp(c, a, b) <= 0 ||
        d2 == 0 && betweencmp(d, a, b) <= 0 ||
        d3 == 0 && betweencmp(a, c, d) <= 0 ||
        d4 == 0 && betweencmp(b, c, d) <= 0) return 2;
      return 0;
}
int linecrossseg(Point a, Point b, Point c, Point d, Point& temp)
{//直线于线段相交，返回相交交点abcd
      double s1, s2;
      int d1, d2;
      d1 = dblcmp(s1 = cross(a, b, c));
      d2 = dblcmp(s2 = cross(a, b, d));
      if (d1 * d2 < 0) {
            temp.x = (c.x * s2 - d.x * s1) / (s2 - s1);
            temp.y = (c.y * s2 - d.y * s1) / (s2 - s1);
            return 1;
      }
      if (d1 * d2 == 0)//交于端点
      {
            if(d2 == 0) temp = d;
            else temp = c;
            return 2;
      }
      return 0;
}
bool linecross(Point a, Point b, Point c, Point d, Point& temp)
{//直线与直线是否相交，相交返回交点abcd
      if((b.x - a.x) * (d.y - c.y) == (d.x - c.x) * (b.y - a.y)) return false;
      double s1, s2;
      int d1, d2;
      d1 = dblcmp(s1 = cross(a, b, c));
      d2 = dblcmp(s2 = cross(a, b, d));
      temp.x = (c.x * s2 - d.x * s1) / (s2 - s1);
      temp.y = (c.y * s2 - d.y * s1) / (s2 - s1);
      return true;
}

/* *********************************************
 * 凸包模板
 * *********************************************/
void ConvexHull(Point* pts, Point* stk, int n, int &top)//p[0] != [n - 1]
{
      sort(pts, pts + n);
      top = -1;
      stk[++top] = pts[0];
      stk[++top] = pts[1];
      for(int i = 2;i < n;i++)
      {
            while(top >= 1 && dblcmp(cross(stk[top - 1], stk[top], pts[i])) <= 0)
                  top--;
            stk[++top] = pts[i];
      }
      int now = top;
      for(int i = n - 2;i >= 0;i--)
      {
            while(top >= now + 1 && dblcmp(cross(stk[top - 1], stk[top], pts[i])) <= 0)
                  top--;
            stk[++top] = pts[i];
      }
}
/* ********************************************
 * 旋转卡壳专用
 * ********************************************/
double rotating_calipers_longest(Point* p, int n) //卡壳
{
      double res = 0;
      p[n] = p[0];
      for(int i = 0, j = 1; i < n; i ++)
      {
            while(dblcmp(cross(p[i], p[i + 1], p[j]) - cross(p[i], p[i + 1], p[(j + 1) %n])) < 0)
                  j = (j + 1) % n;
            res = max(res, fabs(cross(p[i], p[i + 1], p[j])));
      }
      return res;
}
double rotating_calipers_triangle(Point p[],int n)
{
      int i, j = 1, q = 2;
      p[n] = p[0];
      p[n+1] = p[1];
      p[n+2] = p[2];
      double temp, ans = 0;
      for (i = 0; i < n; i++)
      {
            while (cross(p[i],p[j],p[q+1]) - (temp = cross(p[i],p[j],p[q])) > EPS)
                  q = (q + 1) % n;
            ans = max(ans, temp);
            while (cross(p[i],p[j+1],p[q]) - (temp = cross(p[i],p[j],p[q])) > EPS)
                  j = (j + 1) % n;
            ans = max(ans, temp);
      }
      return ans;
}

/* ********************************************
 * 多边形重心
 * ********************************************/
Point barycenter(Point a, Point b, Point c)
{
      Point tmp;
      linecross(Point((a.x + b.x) / 2, (a.y + b.y) / 2), c,
                  Point((a.x + c.x) / 2, (a.y + c.y) / 2), b, tmp);
      return tmp;
}
```

```
Point barycenter(Point p[], int n)
{
      Point ret, t;
      double t1 = 0, t2;
      ret.x = ret.y = 0;
      for (int i = 1; i < n - 1; i++)
            if (fabs(t2 = cross(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;
}
Point verticalfoot(Point a, Point b, Point c)
{//在上都的垂足cab
      Point tmp(c.x - a.y + b.y, c.y + a.x - b.x), ans;
      linecross(a, b, c, tmp, ans);
      return ans;
}
```

4.2 3D-Convex

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <vector>
using namespace std;
int faces;
int sig(double x)
{
   return (x > 1E-6) - (x < -1E-6);
}
#define N 505
struct Point
{
   double x, y, z;
   Point() {}
   Point(double x, double y, double z) : x(x), y(y), z(z) {}
   Point operator +(Point b)
   {
      return Point(x + b.x, y + b.y, z + b.z);
   }
   Point operator -(Point b)
   {
      return Point(x - b.x, y - b.y, z - b.z);
   }
   Point operator /(double t)
   {
      return Point(x / t, y / t, z / t);
   }
   double len()
   {
      return sqrt(x * x + y * y + z * z);
   }
};
double dot(Point a, Point b)
{
   return a.x * b.x + a.y * b.y + a.z * b.z;
```

```
}
Point cross(Point a, Point b)
{
   return Point(a.y * b.z - a.z * b.y,
             -(a.x * b.z - a.z * b.x),
             a.x * b.y - a.y * b.x);
}
Point ps[N];
struct Face
{
   int a,b,c;
   Face(int a ,int b , int c ): a (a), b(b) , c(c) {}
   double area ()
   {
      return cross( ps[b]-ps[a] , ps[c]-ps[a] ).len();
   }
   Point fa() const
   {
      return cross( ps[b]-ps[a] , ps[c]-ps[a] );
   }
   bool same_side(Point q , Point p)
   {
      return sig ( dot(ps[a] - q, cross(ps[b] - q, ps[c] - q))
                * dot(ps[a] - p , cross( ps[b] - p , ps[c] - p)) ) > 0 ;
   }
   bool inFace(Point q) const
   {
      return sig(dot(ps[a] - q, cross(ps[b] - q, ps[c] - q)))==0;
   }
   bool operator == (const Face & face) const
   {
      Point fa1 = fa();
      Point fa2 = face.fa();
      if(sig(cross(fa1,fa2).len())!=0) return false;
      return inFace(ps[face.a]);
   }
};
struct line
{
   int a, b;
   line(int a, int b) : a(a),b(b) {}
};
double convexHull(Point *ps, int n)
{
#define judge(S, T) \
map[C[j].S][C[j].T]=map[C[j].T][C[j].S]= map[C[j].S][C[j].T]==0;\
LT.push_back(line(C[j].S, C[j].T))
   static bool map[N][N];
   static vector <Face> C , FT;
   static vector <line> LT;
   int i, j;
   if(n <= 2) return 0.0;
   if(n == 3) return cross(ps[1]-ps[0] , ps[2]-ps[0]).len()*0.5;
   C.clear();
   memset(map, 0 , sizeof(map));
   for(i = 0; i < 4; i ++)
      C.push_back(Face(i, (i+1)%4, (i+2)%4));
   Point center = (ps[0] + ps[1] + ps[2] + ps[3]) / 4;
   for(i = 4 ; i < n ; i ++ )
   {
      FT.clear();
      LT.clear();
      for (j = 0 ; j < C.size() ; j ++ )
         if ( ! (C[j].same_side( center , ps[i] )) )
         {
            judge(a, b);
            judge(c, b);
            judge(c, a);
         }
         else FT.push_back(C[j]);
      C.clear();
      for(j = 0 ; j < FT.size() ; j ++ )
         C.push_back( FT[j] );
      for(j = 0 ; j < LT.size() ; j ++ )
         if (map [ LT[j].a ][ LT[j].b ])
         {
            C.push_back( Face ( LT[j].a , LT[j].b , i ) );
            map[ LT[j].a ][ LT[j].b ] = map[ LT[j].b ][ LT[j].a ] = 0;
         }
   }
   double area = 0 ;
   for ( i = 0 ; i < C.size() ; i ++ )
      area += C[i].area();
   area /= 2.0;

   faces = 0;
   for(int i = 0; i < C.size(); i ++)
   {
      bool ok = true;
      for(int j = i+1; j < C.size(); j ++)
      {
         if(C[i]==C[j])
         {
            ok = false;
            break;
         }
      }
      faces += ok;
   }
   return area;
}
int main()
{
   int n;
   double x, y, z;
   while(scanf("%d", &n) != EOF)
   {
      for(int i = 0; i < n; i ++)
      {
         scanf("%lf%lf%lf", &x, &y, &z);
         ps[i] = Point(x, y, z);
      }
      while(sig(convexHull(ps, n)) == 0)
      {
         for(int j = n-1; j > 0; j --)
      {
            swap(ps[j], ps[rand()%j]);
         }
      }
      printf("%d\n", faces);
   }
   return 0;
}
```

5 math

5.1 cantor extend

```
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
const int INF = 0XFFFFFF;
const int FAC_N = 10; // 0! to (n - 1)!
int fac[FAC_N] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};
int cantor(int *a, int n)
{
      int ans = 0, i, j, r;
      char p[10] = {0};
      for (i = 0; i < n; i++)
      {
            for (j = 1, r = 0; j <= a[i]; j++)
                  if (p[j] == 0) r++;
            ans += (r - 1) * fac[8 - i];
            p[a[i]] = 1;
      }
      return ans;
}

void uncantor(int s, int *a, int n)// 0 to n - 1
{
      int i, j, r, t;
      char p[FAC_N] = {0};
      for (i = 0; i < n; i++)
      {
            t = s / fac[n - 1 - i] + 1;
            s %= fac[n - 1 - i];
            r = 0, j = 1;
            while (1)
            {
                  if (p[j] == 0) r++;
                  if (r == t) break;
                  j++;
            }
            a[i] = j;
            p[j] = 1;
      }
      return;
}
```

5.2 eular function

```
//f(n * m) = f(n) * f(m)
//f(n) = n * (1 - 1 / p1) *(1 - 1 / p2) (piis n's prim factor)
int euler(int n)
{
   int e;
   int i, j;
   e = n;
   for (i = 2; i * i <= n; i++)
   {
      if (n % i == 0)
      {
         e = e / i * (i - 1);
         while (n % i == 0)
            n = n / i;
      }
   }
   if (n > 1)
      e = e / n * (n - 1);
   return e;
}
```

5.3 Matrix

```
#define MOD 10000000
using namespace std;
const int maxn = 5;
struct Matrix
{
   long long A[maxn][maxn];
   int size;

   Matrix()
   {
      memset(this, 0, sizeof (*this));
   }
};
long long mymod(long long x)
{
   return (x % MOD + MOD) % MOD;
}
Matrix operator+(Matrix m1, Matrix m2)
{
   Matrix ret;
   ret.size = m1.size;
   for (int i = 0; i < ret.size; ++i)
      for (int j = 0; j < ret.size; ++j)
         ret.A[i][j] = mymod(m1.A[i][j] + m2.A[i][j]);
   return ret;
}
Matrix operator-(Matrix m1, Matrix m2)
{
   Matrix ret;
   ret.size = m1.size;
   for (int i = 0; i < ret.size; ++i)
      for (int j = 0; j < ret.size; ++j)
         ret.A[i][j] = mymod(m1.A[i][j] - m2.A[i][j]);
   return ret;
}
Matrix operator*(Matrix m1, Matrix m2)
{
   Matrix ret;
   ret.size = m1.size;
   for (int i = 0; i < ret.size; ++i)
      for (int j = 0; j < ret.size; ++j)
      {
         ret.A[i][j] = 0;
         for (int k = 0; k < ret.size; ++k)
            ret.A[i][j] += m1.A[i][k] * m2.A[k][j];
         ret.A[i][j] = mymod(ret.A[i][j]);
      }
   return ret;
}
Matrix mypower(Matrix m, int n)
{
   Matrix ret, tmp;
   ret.size = m.size;
   if (n == 0)
   {
      for (int i = 0; i < ret.size; ++i)
         ret.A[i][i] = 1;
      return ret;
   }
   tmp = mypower(m, n / 2);
   if (n & 1)
      return tmp * tmp * m;
   else return tmp * tmp;
}
Matrix sumpower(Matrix m, int n)
{
   Matrix tmp;
   if (n == 1) return m;
   tmp = sumpower(m, n / 2);
   if (n & 1)
      return mypower(m, n / 2) * tmp + tmp + mypower(m, n);
   return mypower(m, n / 2) * tmp + tmp;
}
```

5.4 miller rabin

```
/*miller-rabin algorithm do it 5 times or more*/
long long qmod(long long a,long long b,long long c)
{
   long long res = 1,temp = a % c;
   while(b)
   {
      if(b & 1)
         res = (res * temp) % c;
      b>>= 1;
      temp = (temp * temp) % c;
   }
   return res;
}

bool miller(long long n,int t)
{
   if(n == 1)
      return false;
   else if(n == 2)
      return true;
   for(int i = 0;i < t;i++)
   {
      srand(time(NULL));
      long long a = rand() % (n - 2) + 1;
      int b = qmod(a,n - 1,n);
      if(b != 1 && b != n - 1)
         return false;
   }
   return true;
}

int main()
{
   long long a;
   while(scanf("%lld",&a) == 1)
   {
      if(miller(a,4))
         printf("YES\n");
      else
         printf("NO\n");
   }
   return 0;
}
```

5.5 pollard rho

```
typedef long long LL;
LL min;
LL multi(LL a, LL b, LL n)
{
   LL tmp = a % n, s = 0;
   while(b)
   {
      if(b & 1) s = (s + tmp) % n;
      tmp = (tmp + tmp) % n;
      b >>= 1;
   }
   return s;
}
LL gcd(LL a, LL b)
{
   return b ? gcd(b, a % b) : a;
}
LL pollard_rho(LL n, LL c)
{
   LL x, y, d, i = 1, k = 2;
   srand((LL)(0));
   x = ((LL) rand()) % (n - 1) + 1;
   y = x;
   while(1)
   {
      i ++;
      x = (multi(x, x, n) + c) % n;
      d = gcd(y - x + n, n);
      if(d != 1 && d != n) return d;
      if(y == x) return n;
      if(i == k) y = x, k <<= 1;
   }
}
void find(LL n, LL c)
{
   LL r;
   if(n <= 1) return;
   if(test(n))
   {
      if(min > n) min = n;
      return;
   }
   r = pollard_rho(n, c--);
   find(n / r, c);
   find(r, c);
}
LL MaxPrimeFactor(LL n)
{
   if(test(n)) return n;
   LL k = -1, g;
   min = n;
   find(n, C);
   g = MaxPrimeFactor(min);
   k = g > k ? g : k;
   g = MaxPrimeFactor(n / min);
   k = g > k ? g : k;
   return k;
}
int main()
{
   LL n;
   while(scanf("%lld", &n) == 1)
   {
      if(test(n)) //test(n) is miller robin
         printf("Yes\n");
      else
      {
         min = n; //min is the min factor of n
         find(n, C);
         printf("No %lld\n",min);
         //printf("%lld\n",MaxPrimeFactor(n));
      }
   }
   return 0;
}
```

5.6 linearModularSystem

```
typedef long long LL;
LL gcd(LL a, LL b)
{
   if(b == 0) return a;
   return gcd(b, a % b);
}
LL extended_euclid(LL a,LL b,LL &x,LL &y)
{
   if (b == 0)
   {
      x = 1, y = 0;
      return a;
   }
   LL ret = extended_euclid(b ,a % b, x, y), t = x;
   x = y;
   y = t - a / b * y;
   return ret;
}
bool modular_linear(LL a,LL b,LL c)
{
   LL x, y;
   LL d = extended_euclid(a,b,x,y);
   if (c%d) return 0;
   return true;
}
LL linearModularSystem(LL* m, LL* r, int n)//保证互质m且有节,
{
   LL M = accumulate(m, m + n, 1, multiplies<LL>());
   LL ans = 0;
   for(int i = 0; i < n; i++)
   {
      LL Mi = M / m[i], pi, qi;
      LL gcd = extended_euclid(Mi, m[i], pi, qi);
      if(Mi % gcd) return -1;
      ans = (ans + Mi * pi * r[i]) % M;
   }
   return ans <= 0 ? ans + M : ans;
}//minimum non-negative answer
LL linearModularSystemP(LL* m, LL* r, int n)//不互质
{
   LL m0 = m[0], r0 = r[0];//前一方程
   LL m1, r1;//当前方程
   LL x, y, t;
   for(int i = 1; i < n; i++)
   {
      r1 = r[i], m1 = m[i];
      long long gcd = extended_euclid(m0, m1, x, y);
      LL c = r1 - r0;
      if(c % gcd != 0) return -1;
```

```
        //m0 * x - m1 * y = r1 - r0
        t = m1 / gcd;//倍数
        x = (c / gcd * x % t + t) % t;//最小正整数解
        r0 = r0 + x * m0;
        m0 = m0 * m1 / gcd;
    }
    return r0;
}
```

5.7 Eratosthenes

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 100000;
int tag[N], p[N];
void get_prime()
{
    int cnt = 0;
    for (int i = 2; i < N; i++)
    {
        if (!tag[i]) p[cnt++] = i;
        for (int j = 0; j < cnt && p[j] * i < N; j++)
        {
            tag[i*p[j]] = 1;
            if (i % p[j] == 0) break;
        }
    }
}
```

6 dynamic Programming

6.1 DFA DP

```
LL DP(LL a, int k)
{
   char str[N * 2];
   sprintf(str, "%lld", a);
   LL dp[2][2][N][N][N], ans = 0;//section, 0\1, length, mod
   int len = strlen(str);
   memset(dp, 0, sizeof(dp));//deal with prefix 0
   dp[0][1][0][(str[0] - '0') % k][0]++;
   for(int now = 1; now < str[0] - '0'; now++)
      dp[0][0][0][now % k][0]++;
   REP(nxt, 1, len)
   {
      REP(now, 1, 10)//deal with prefix 0
         dp[0][0][nxt][now % k][0]++;
      REP(a, 0, k)
      {
         REP(b, 0, k)
         {
            for(int now = 0; now < 10; now++)
            {
               if(now == str[nxt] - '0')
               {
                  dp[0][1][nxt][(a * 10 + now) % k][b]+=dp[0][1][nxt-1][a][b];
                  dp[1][1][nxt][a][(b * 10 + now) % k]+=dp[0][1][nxt-1][a][b];
                  dp[1][1][nxt][a][(b * 10 + now) % k]+=dp[1][1][nxt-1][a][b];
               }
               else if(now < str[nxt] - '0')
               {
                  dp[0][0][nxt][(a * 10 + now) % k][b]+=dp[0][1][nxt-1][a][b];
                  dp[1][0][nxt][a][(b * 10 + now) % k]+=dp[0][1][nxt-1][a][b];
                  dp[1][0][nxt][a][(b * 10 + now) % k]+=dp[1][1][nxt-1][a][b];
               }
               dp[0][0][nxt][(a * 10 + now) % k][b] += dp[0][0][nxt - 1][a][b];
               dp[1][0][nxt][a][(b * 10 + now) % k] += dp[0][0][nxt - 1][a][b];
               dp[1][0][nxt][a][(b * 10 + now) % k] += dp[1][0][nxt - 1][a][b];
            }
         }
      }
   }
   for(int a = 0;a < k;a++)
      for(int b = 0;b < k;b++)
         if((a + b) % k == 0)
            ans += dp[1][0][len - 1][a][b] + dp[1][1][len - 1][a][b];
   return ans;
}
int main ()
{
   LL a, b;
   int k;
   while (scanf ("%lld%lld%d", &a, &b, &k) == 3)
      printf("%lld\n", DP(b, k) - DP(a - 1, k));
   return 0;
}
```

6.2 mask & connection

```
typedef long long LL;
const int N = 14;
const int TOT = 50000;
const int MAXN = 1594323;// 3^13
```

```
char maps[N][N];
int bit3[N] = {1}, status[TOT];
int 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);
}
void preprocess()
{
   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;
}
int getbit(int s, int i)
{
   while(i-- > 0) s /= 3;
   return s % 3;
}
void transfer(LL& dest, LL add)
{
   dest == -1 ? (dest = add) : (dest += add);
}
LL DP(int n, int m, int px, int py)
{
   LL ans = 0;
   int now = 0, pre;
   CC(dp, -1);
   dp[0][0] = 1;
   for(int i = 0; i < n; i++)
   {
      for(int j = 0; j < m; j++)
      {
         pre = now;now ˆ= 1;
         CC(dp[now], -1);
         for(int k = 0, s; s = status[k], s < bit3[m + 1]; k++)
         {
            if(dp[pre][k] == -1) 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)//both down and right build 2 plugin
            {
               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 (( make )) to ()
        {
          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]);
              if(Hash[nows - bit3[b]] == -1)
                cout << nows - bit3[b] << endl;
              break;
            }
          }
        }
        else if(l == 2 && u == 2)// merge )) make (( to ()
        {
          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]);
              if(Hash[nows + bit3[b]] == -1)
                cout << nows + bit3[b] << endl;
              break;
            }
          }
        }
        else if(l == 1 && u == 2)//merge () at last grid
        {
          if(px == i && py == j)
            ans += dp[pre][k];
        }
        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]);
        }
      }
    }
    pre = now;now ^= 1;
    CC(dp[now], -1);//must CC -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 ans;
}
```

```
int main()
{
   int n, m, px, py;
   preprocess();
   while(scanf("%d %d", &n, &m) == 2)
   {
      CC(maps, 0);
      REP(i, 0, n) scanf("%s", maps[i]);
      REP(i, 0, n) REP(j, 0, m) if(maps[i][j] == '.') px = i, py = j;
      printf("%lld\n", DP(n, m, px, py));
   }
   return 0;
}
```

6.3 RMQ

```
/*RMQ-ST be careful with log2*/
const int SIZE = 500001;
int dp[SIZE][20];
int n, q, a, b, len, ans;
int main()
{
   while (scanf("%d %d", &n, &q) == 2)
   {
      for (int i = 1; i <= n; i++)
         scanf("%d", &dp[i][0]);
      for (int j = 1; j <= log(n) / log(2); j++)
         for (int i = 1; i + (1 << (j - 1)) <= n; i++)
            dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
      while (q--)
      {
         scanf("%d %d", &a, &b);
         len = log(b - a + 1) / log(2) + 0.001;
         ans = max(dp[a][len], dp[b - (1 << len) + 1][len]);
         printf("%d\n", ans);
      }
   }
   return 0;
}
```

6.4 HOJ2973

```
char from[201], to[201];
int dp[201][201][3];

int getmin(int start, int end, int state)
{
   if (start > end)
      return 0;
   if (dp[start][end][state] < 100000)
      return dp[start][end][state];
   if ((from[start] == to[start] && state == 0) || state == to[start]-'A' + 1)
      dp[start][end][state] = getmin(start + 1, end, state);
   else
   {
      for (int i = start; i <= end; ++i)
         dp[start][end][state] = min(dp[start][end][state],
      getmin(start, i, to[start] - 'A' + 1) + getmin(i + 1, end, state) + 1);
   }
   return dp[start][end][state];
}

```

```
int main()
{
   int test;
   scanf("%d", &test);
   while (test--)
   {
      scanf("%s %s", from, to);
      int len = strlen(from);
      for (int i = 0; i < len; ++i)
         for (int j = i; j < len; ++j)
            dp[i][j][0] = dp[i][j][1] = dp[i][j][2] = 100000;
      printf("%d\n", getmin(0, len - 1, 0));
   }
   return 0;
}
```

6.5 slope

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

const int SIZE = 20002;
const LL INF = 2000000000LL;
long long w[SIZE], d[SIZE];//input data
long long ds[SIZE], dp[SIZE];
long long sw[SIZE], sp[SIZE], bp[SIZE];
int n, q[SIZE], head, tail;

void preprocess()
{
   ds[1] = 0;
   for(int i = 2; i <= n + 1; i++)
      ds[i] = ds[i - 1] + d[i - 1];
   sw[0] = sp[0] = 0;
   for(int i = 1; i <= n + 1; i++)
   {
      sw[i] = sw[i - 1] + w[i];
      sp[i] = sp[i - 1] + sw[i - 1] * d[i - 1];
   }
   bp[n + 1] = 0;
   for(int i = n; i >= 1; i--)
      bp[i] = bp[i + 1] + w[i] * (ds[n + 1] - ds[i]);
}

void DP()
{
   fill(dp + 1,dp + n + 1, INF);
   long long ans = INF;
   head = tail = 0;
   q[tail++] = 1;
   ans = min(ans,sp[n + 1] - sw[0] * (ds[1] - ds[0]) - sw[1] * (ds[n + 1] - ds
       [1]));
   for(int i = 2; i <= n; i++)
   {
      /*
     * bp[i + 1] = sp[n + 1] - sp[i] - sw[i] * (ds[n + 1] - ds[i]);
     * dp[i] = min(dp[i],bp[i + 1] + sp[j] + sp[i] - sp[j] - sw[j] * (ds[i] -
         ds[j]));
     * dp[i] = min(dp[i],sp[n + 1] - sw[j] * (ds[i] - ds[j]) - sw[i] * (ds[n +
          1] - ds[i]));
       * dp[i]k is the minimum index in 0-k dp[i]j - dp[i]k >= 0 (j < k)
     * (sw[j] * ds[j] - sw[k] * ds[k]) / (sw[j] - sw[k]) <= ds[i] <= ds[i +
         1];
       * so i + 1 decision is more than k
     * y = sw[j] * ds[j] x = sw[j];
       */
      while(head + 1 < tail)
      {
         long long y1 = sw[q[head]] * ds[q[head]],y2 = sw[q[head + 1]] * ds[q[
             head + 1]];
         long long x1 = sw[q[head]],x2 = sw[q[head + 1]];
         if((y2 - y1) <= ds[i] * (x2 - x1)) head++;
         else break;
      }
      int k = q[head];
      dp[i] = sp[n + 1] - sw[k] * (ds[i] - ds[k]) - sw[i] * (ds[n + 1] - ds[i])
          ;
      ans = min(ans,dp[i]);
      while(head + 1 < tail)
      {
         long long y1 = sw[q[tail - 2]] * ds[q[tail - 2]];
         long long y2 = sw[q[tail - 1]] * ds[q[tail - 1]],y3 = sw[i] * ds[i];
         long long x1 = sw[q[tail - 2]],x2 = sw[q[tail - 1]],x3 = sw[i];
         if((y2 - y1) * (x3 - x2) >= (y3 - y2) * (x2 - x1)) tail--;
         else break;
      }
      q[tail++] = i;
   }
   printf("%lld\n",ans);
}

int main()
{
   while(scanf("%d", &n) == 1)
   {
      for(int i = 1; i <= n; i++)
         scanf("%lld %lld", &w[i], &d[i]);
      preprocess();
      DP();
   }
   return 0;
}
```

7 other

7.1 连通性DP

```
class HashTable
{
private:
   const static int SIZE = 1000000;
   const static int MOD = 10007;
   struct HashCell
   {
      int value, idx;
      HashCell *nxt;
   } 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->nxt)
      {
         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->nxt = g[hash];
      g[hash] = pp++;
   }
} hashTable;
const int N = 10;
const int STATE_CNT = 1000000;
const int INF = 10000000;
const int HEX = 10;
const int BIT[] = {1, 10, 100, 1000, 10000, 100000,
      1000000, 10000000, 100000000, 1000000000};
int state[2][STATE_CNT], dp[2][STATE_CNT];
int newState(int s[], const int M)
{
   int lab[N], cnt = 0, newS = 0;
   memset(lab, -1, sizeof(lab));
   lab[0] = cnt++;
   for(int i = M - 1; i >= 0; i--)
   {
      newS *= 10;
      if(lab[s[i]] == -1)
         lab[s[i]] = cnt++;
      newS += lab[s[i]];
   }
   return newS;
}
int cnt[N];
void change(int src, int* dest, const int M)
{
   memset(cnt, 0, sizeof(cnt));
   REP(i, 0, M)
   {
      cnt[src % HEX]++;
      dest[i] = src % HEX;
      src /= HEX;
   }
}
bool isOneBlock(int s)
{
   int last = -1;
   while(s)
   {
      int now = s % HEX;
      if(now != 0 && now != last && last != -1) return false;
      if(now != 0) last = now;
      s /= HEX;
   }
   return true;
}
void transfer(int now, int newS, int val, int& newCnt)
{
   int idx = hashTable.find(newS);
   if(idx != -1)
      dp[now][idx] = max(dp[now][idx], val);
   else
   {
      idx = newCnt;
      hashTable.insert(newS, newCnt++);
      state[now][idx] = newS;
      dp[now][idx] = val;
   }
}
int DP(int n, int m, int maps[N][N])
{
   int ans = -INF;
   state[0][0] = dp[0][0] = 0;
   int newS[N], now = 0, pre = 1;
   int oldCnt, newCnt = 1;
   REP(i, 0, n)
   {
      REP(j, 0, m)
      {
         now ^= 1, pre ^= 1;
         oldCnt = newCnt;
         newCnt = 0;
         hashTable.clear();
         REP(k, 0, oldCnt)
         {
            /*注意处理不选则该块的情况，如果不选择该块会导致一个联通块在表示法中消失。则改选法将导致多个联通块

             */
            change(state[pre][k], newS, m + 1);
            if(newS[j] == 0 && newS[j + 1] == 0)
            {
               transfer(now, state[pre][k], dp[pre][k], newCnt);
               int minn = *max_element(newS, newS + m + 1) + 1;
               newS[j] = newS[j + 1] = minn;
               int nxt = newState(newS, m + 1);
               transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);
            }
            else if(newS[j] == 0 && newS[j + 1])
            {
               newS[j] = newS[j + 1];
               int nxt = newState(newS, m + 1);
               transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);
               if(cnt[newS[j + 1]] > 1)
               {
                  newS[j] = newS[j + 1] = 0;
                  nxt = newState(newS, m + 1);
                  transfer(now, nxt, dp[pre][k], newCnt);
               }
            }
            else if(newS[j] && newS[j + 1] == 0)
            {
               newS[j + 1] = newS[j];
               int nxt = newState(newS, m + 1);
               transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);
               if(cnt[newS[j]] > 1)
               {
                  newS[j] = newS[j + 1] = 0;
                  nxt = newState(newS, m + 1);
                  transfer(now, nxt, dp[pre][k], newCnt);
               }
            }
            else if(newS[j] && newS[j + 1])
            {
               if(newS[j] == newS[j + 1])
               {
                  if(cnt[newS[j]] > 2)
                  {
                     int a = newS[j], b = newS[j + 1];
                     newS[j] = newS[j + 1] = 0;
                     int nxt = newState(newS, m + 1);
                     transfer(now, nxt, dp[pre][k], newCnt);
                     newS[j] = a, newS[j + 1] = b;
                  }
               }
               else
               {
                  if(cnt[newS[j]] > 1 && cnt[newS[j + 1]] > 1)
                  {
                     int a = newS[j], b = newS[j + 1];
                     newS[j] = newS[j + 1] = 0;
                     int nxt = newState(newS, m + 1);
                     transfer(now, nxt, dp[pre][k], newCnt);
                     newS[j] = a, newS[j + 1] = b;
                  }
               }
               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;
               int nxt = newState(newS, m + 1);
               transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);
            }
         }
      }
      now ^= 1, pre ^= 1;
      oldCnt = newCnt;
      newCnt = 0;
      hashTable.clear();
      REP(k, 0, oldCnt)
      {
         if(isOneBlock(state[pre][k]))
            ans = max(ans, dp[pre][k]);
         if(state[pre][k] - BIT[m] > 0)
         {
            change((state[pre][k] - BIT[m]) * 10, newS, m + 1);
            int nxt = newState(newS, m + 1);
            if(nxt != 0)
               transfer(now, nxt, dp[pre][k], newCnt);
         }
         else if(state[pre][k] != 0)
         {
            change(state[pre][k] * 10, newS, m + 1);
            int nxt = newState(newS, m + 1);
            if(nxt != 0)
               transfer(now, nxt, dp[pre][k], newCnt);
         }
      }
      transfer(now, 0, 0, newCnt);
   }
   return ans;
}

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

7.2 input stream

```
class BufferedReader
{
public:
   BufferedReader& operator >> (int& number)
   {
      number = getInt();
      return *this;
   }
private:
   int getInt()
   {
      char ch;
      while((ch = getchar()) && !isdigit(ch));
      int ret = ch - ’0’;
      while((ch = getchar()) && isdigit(ch))
      {
         ret *= 10;
         ret += ch -’0’;
      }
      return ret;
   }
} buf;
```

7.3 splay

```
struct node
{
   static const int INF = 100000000;
   node* ch[2], *pre;
   int v, flip, minn, delta, tot;
   node(int v, int tot, node* l, node* r, node* pre)
      : v(v), minn (v), tot(tot), pre(pre)
   {
      this->delta = 0;
      this->flip = 0;
      ch[0] = l, ch[1] = r;
   }
   inline int minValue()
   {
      return minn;
   }
   inline int size()
   {
      return tot;
   }
   void reverse()
   {
      if(tot == 0) return;
      flip ˆ= 1;
   }
   void add(int d)
   {
      if(tot == 0) return;
      this->minn += d;
      this->delta += d;
      this->v += d;
   }
   void pushDown()
   {
      if(tot == 0) return;
      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 pushUp()
   {
      if(tot == 0) return;
      tot = ch[0]->size() + ch[1]->size() + 1;
      minn = min(v, min(ch[0]->minValue(), ch[1]->minValue()));
   }
};
class splayTree
{
private:
   node* root, * null;
   void clear(node*& now)
   {
      if(now == null) return;
      clear(now->ch[0]);
      clear(now->ch[1]);
      delete now;
      now = null;
   }
   void rotate(node* x, int type)
   {
      node *y = x->pre;
      y->pushDown();
      x->pushDown();
      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[0] == y) y->pre->ch[0] = x;
         else y->pre->ch[1] = x;
      }
      x->ch[type] = y, y->pre = x;
      if (y == root) root = x; // root 表示整棵树的根结点
      y->pushUp();
      x->pushUp();
   }
   void splay(node* x, node* f)
   {
      x->pushDown();
      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) // 一字形旋转
                  rotate(y, 1), rotate(x, 1);
               else // 之字形旋转
                  rotate(x, 0), rotate(x, 1);
            }
            else
            {
               if (y->ch[1] == x) // 一字形旋转
                  rotate(y, 0), rotate(x, 0);
               else // 之字形旋转
                  rotate(x, 1), rotate(x, 0);
            }
         }
      }
      x->pushUp();
   }
   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->pushUp();
   }
   // the flag node is !not! included, be careful when make interval
   void findK(int k, node* pre)
   {
      node* now = root;
      while(true)
      {
         now->pushDown();
         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);
   }
public:
   splayTree()
   {
      null = new node(node::INF, 0, 0, 0, 0);
      root = null;
   }
   ~splayTree()
   {
      clear(root);
      delete null;
   }
   // make a sequence from 1 to n do build(0, n + 1, val)
   // and make sure val[0] = va[1] = INF;
   void build(int l, int r, int* val)
   {
      if(l > r) return;
      build(l, r, root, null, val);
   }
#define centre (root->ch[1]->ch[0])
   int minElement(int a, int b)
   {
      makeInterval(a, b);
      return centre->minValue();
   }
   void addValue(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, revolve right
      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]->pushUp();
      root->pushUp();
      splay(centre, null);
   }
   void erase(int a)
   {
      makeInterval(a, a);
      delete centre;
      centre = null;
      root->ch[1]->pushUp();
      root->ch[0]->pushUp();
   }
   void clear()
   {
      clear(root);
   }
} tree;
const int N = 300000;
int val[N];

int main()
{
   int n, m;
   int a, b, c;
   char cmd[100];
   while(scanf("%d", &n) == 1)
   {
      for(int i = 1; i <= n; i++)
         scanf("%d", &val[i]);
      val[0] = val[n + 1] = node::INF;
      tree.clear();
      tree.build(0, n + 1, val);
      scanf("%d", &m);
      REP(i, 0, m)
      {
         scanf("%s", cmd);
         if(!strcmp(cmd, "ADD"))
         {
            scanf("%d %d %d", &a, &b, &c);
            tree.addValue(a, b, c);
         }
         else if(!strcmp(cmd, "REVERSE"))
         {
            scanf("%d %d", &a, &b);
            tree.reverse(a, b);
         }
         else if(!strcmp(cmd, "REVOLVE"))
         {
            scanf("%d %d %d", &a, &b, &c);
            int tot = b - a + 1;
            c = (c % tot + tot) % tot;
            tree.revolve(a, b, c);
         }
         else if(!strcmp(cmd, "INSERT"))
         {
            scanf("%d %d", &a, &c);
            tree.insert(a, c);
         }
         else if(!strcmp(cmd, "DELETE"))
         {
            scanf("%d", &a);
            tree.erase(a);
         }
         else if(!strcmp(cmd, "MIN"))
         {
            scanf("%d %d", &a, &b);
            printf("%d\n", tree.minElement(a, b));
         }
      }
   }
   return 0;
}
```

7.4 network with low bound

```
#define OP(i) (((i) - (pool))^1)
class sap
{
private:
   const static int V = 2010, E = 100000, INF = 100000000;
   int dis[V], numdis[V], pre[V], b[V];
   struct edge
   {
      int v, cap, low;
      edge *nxt;
   } pool[E], *g[V], *pp, *e[V], *pree[V];
   void bfs(int v, int n)
   {
      int que[V], tail = 0;
      bool vst[V] = {0};
      memset(numdis, 0, sizeof(numdis));
      fill(dis, dis + n, n);
      dis[v] = 0, vst[v] = 1, que[0] = v;
      for (int j = 0; j <= tail; j++)
      {
         int tmp = que[j % n];
         for (edge *i = g[tmp]; i != NULL; i = i->nxt)
         {
            if (pool[OP(i)].cap > 0 && !vst[i->v])
            {
               tail++;
               vst[i->v] = 1;
               que[tail % n] = i->v;
               dis[i->v] = dis[tmp] + 1;
               numdis[dis[i->v]]++;
            }
         }
      }
   }
   int findArgumentPath(int &v, int s, int t)
   {
      while (e[v] != NULL)
      {
         if (e[v]->cap > 0 && dis[v] == dis[e[v]->v] + 1)
         {
            pre[e[v]->v] = v, pree[e[v]->v] = e[v], v = e[v]->v;
            if (v == t)
            {
               int minf = INF;
               for (int i = t; i != s; i = pre[i])
                  minf = min(minf,pree[i]->cap);
               for (int i = t; i != s; i = pre[i])
               {
                  pree[i]->cap -= minf;
                  pool[OP(pree[i])].cap += minf;
               }
               v = s;
               return minf;
            }
         }
         else e[v] = e[v]->nxt;
      }
      return 0;
   }
   void createEdge(int i, int j, int cap, int low)
   {
      pp->v = j, pp->low = low;
      pp->cap = cap, pp->nxt = g[i];
      g[i] = pp++;
   }
public:
   void addedge(int i, int j, int cap, int low)
   {
      createEdge(i, j, cap - low, low);
      createEdge(j, i, 0, low);
      b[j] += low, b[i] -= low;
   }
   int getLimit(int n, int s, int t)
   {
      int tmpans = 0;
      for(int i = 0; i < n; i++)
      {
         if(i == s || i == t) continue;
         if(b[i] > 0) tmpans += b[i];
      }
      return tmpans;
   }
   int maxflowsap(int n, int s, int t)
   {//n points is from 1 to n, src is 0, tar is n + 1
      for(int i = 0; i < n; i++)
      {
         if(i == s || i == t) continue;
         if(b[i] < 0) addedge(i, t, -b[i], 0);
         if(b[i] > 0) addedge(s, i, b[i], 0);
      }
      bfs(t, n);
      int v = s, maxflow = 0;
      copy(g, g + n, e);
      while (dis[s] < n)
      {
         int add = findArgumentPath(v, s, t);
         maxflow += add;
         if (add == 0)
         {
            int mindis = n;
            numdis[dis[v]]--;
            if (!numdis[dis[v]]) break;
            for (edge *i = g[v]; i != NULL; i = i->nxt)
               if (i->cap > 0) mindis = min(mindis,dis[i->v] + 1);
            dis[v] = mindis;
            numdis[dis[v]]++;
            e[v] = g[v];
            if (v != s) v = pre[v];
         }
      }
      return maxflow;
   }
   void firststart()
   {
      pp = pool;
      memset(g, 0, sizeof(g));
      memset(b, 0, sizeof(b));
   }
   void getAnswer(int n, int m, int k)
   {
      vector<int> ans[V];
      edge * j;
      int i = 0;
      for(i = 0,j = &pool[1]; i < m; i++, j += 2)
      {
         if(j->cap == 1)
            ans[pool[OP(j)].v - n].push_back(j->v);
      }
      for(int i = 1;i <= k;i++)
      {
         printf("%d", ans[i].size());
         FOREACH(ans[i], j)
            printf(" %d", *j);
         puts("");
      }
   }
}flow;

int main()
{
   //freopen("data.in", "r", stdin);
   int n, k;
   while(scanf("%d %d", &n, &k) == 2)
   {
      int s = 0, t = n + k + 1;
      int ss = t + 1, tt = t + 2;
      int tot = 0;
      flow.firststart();
      for(int i = 1;i <= n;i++)
      {
         int a, c;
         scanf("%d", &c);
         tot += c;
         while(c--)
         {
            scanf("%d", &a);
            flow.addedge(i, a + n, 1, 0);
         }
      }
      for(int i = 1;i <= n;i++)
         flow.addedge(s, i, 1, 0);
      for(int i = 1;i <= k;i++)
         flow.addedge(i + n, t, n, 2);
      flow.addedge(t, s, n, k * 2);

      int tmpans = flow.getLimit(tt + 1, ss, tt);
      int ff = flow.maxflowsap(tt + 1, ss, tt);
      if(tmpans == ff)
      {
         printf("YES\n");
         flow.getAnswer(n, tot, k);
      }
      else
         printf("NO\n");
   }
   return 0;
}
```