Team notebook

October 27, 2016

Contents

1 DP

1.1 digitDP

```
vector < int > digit;
ll lim;
ll dp[3][3][31][31][3];
int vis[3][3][31][31][3];
int tcase, tt;

ll rec( bool start, bool small, int pos, ll value, int prev ) {
    if( pos == lim ) return value;
    ll &ret = dp[start][small][pos][value][prev];
    int &v = vis[start][small][pos][value][prev];
    if( v == tt ) return ret;
    v = tt;
    int sesh = small ? 1 : digit[pos];
    ret = 0;
    if( !start ) {
        for( int i=0; i<=sesh; i++ ) {
            ret += rec( false, small || i < digit[pos], pos+1, ( i & prev
                ) + value, i );
        }
    } else {
        for( int i=1; i<=sesh; i++ ) {
            ret += rec( false, small || i < digit[pos], pos+1, ( i & prev
                ) + value, i );
        }
        ret += rec( true, true, pos + 1, 0, 0 );
    }
    return ret;
}

ll calc( ll n ) {

    digit.clear();

    while( n ) {
        digit.push_back( n&1 );
        n >>= 1;
    }

    lim = digit.size();

    reverse( digit.begin(), digit.end() );

    tt++;
    return rec( true, false, 0, 0, 0 );
}
```

1.2 nthPerm

```
long long fac[26];
long long pos, n;
int freq[26];

void init() {
    fac[0] = 1;
    for( int i=1; i<26; i++ ) fac[i] = fac[i-1] * i;
}

long long koita( int n ) {
    long long ret = fac[n];
    for( int i=0; i<26; i++ ) ret /= fac[ freq[i] ];
    return ret;
```

```
}

void solve( int sz ) {
    long long upto, now;
    bool found;
    while( sz ) {
        upto = 0;
        found = 0;
        for( int i=0; i<26 && !found; i++ ) {
            if( freq[i] == 0 ) continue;
            freq[i]--;
            now = koita( sz-1 );
            if( now + upto >= n ) {
                n -= upto;
                sz--;
                putchar( 'a' + i );
                found = 1;
            } else {
                freq[i]++;
                upto += now;
            }
        }
        if( !found ) break;
    }
    putchar( '\n' );
}
```

2 Data Structure

2.1 BIT

```
using vi = vector < int >;
using vii = vector < vi >;

struct BIT_2D {
    int n;
    vii tree;

    BIT_2D () {}
    BIT_2D ( int _n ) : n( _n ), tree( _n, vi( _n, 0 ) ) {}
    ~BIT_2D () {}
    void update_y( int x, int y, int v ) {
        for( ; y<n; y+=(y&-y) ) {
            tree[x][y] += v;
        }
    }

    void update( int x, int y, int v ) {
        for( ; x<n; x+=(x&-x) ) {
            update_y( x, y, v );
        }
    }

    int query_y( int x, int y ) {
        int ret = 0;
        for( ; y; y-=(y&-y) ) {
            ret += tree[x][y];
        }
        return ret;
    }

    int query( int x, int y ) {
        int ret = 0;
        for( ; x; x-=(x&-x) ) {
            ret += query_y( x, y );
        }
        return ret;
    }

    int query( int x1, int y1, int x2, int y2 ) {
        return ( query( x2, y2 ) - query( x2, y1-1 ) - query( x1-1, y2 ) +
            query( x1-1, y1-1 ) );
    }
}

struct BIT {
    int n;
    vi tree;

    BIT () {}
    BIT ( int _n ) : n( _n ), tree( _n, 0 ) {}
    ~BIT () {}
    void update( int x, int v ) {
        for( ; x<n; y+=(x&-x) ) {
            tree[x] += v;
        }
    }
```

```
    int query( int x ) {
        int ret = 0;
        for( ; x; x-=(x&-x) ) {
            ret += tree[x];
        }
        return ret;
    }

    int query( int x, int y, int x2, int y2 ) {
        return ( query( y ) - query( x-1 ) );
    }

    int getind(int x) {
            int idx = 0, mask = n;
            while( mask && idx < n ) {
                    int t = idx + mask;
                    if( x >= tree[t] ) {
                            idx = t;
                            x -= tree[t];
                    }
                    mask >>= 1;
            }
            return idx;
        }
}
```

2.2 Disjoint Set

```
/**
    Implementation of Disjoint-Set Union Data Structure
    Running time:
        O(nlog(n))
    Usage:
        - call make_set() to reset the set
        - call find_rep() to get the set of the vertex
        - call union_() to merge to sets
    Input:
        - n, number of sets
    Tested Problems:
      UVA:
        10608 - Friends
        11503 - Virtual Friends
        10583 - Ubiquitous Religions
**/

struct Disjoint_Set {
    int n;
    vector < int > par, cnt, rank;

    Disjoint_Set( int n ) : n(n), rank(n), par(n), cnt(n) {}

    void make_set() {
        for(int i=0; i<n; i++) {
            par[i] = i;
            cnt[i] = 1;
            rank[i] = 0;
        }
    }

    int find_rep( int x ) {
        if(x != par[ x ]) {
            par[ x ] = find_rep( par[ x ] );
        }
        return par[ x ];
    }

    int union_( int u, int v ) {
        if( ( u = find_rep( u ) ) != ( v = find_rep( v ) ) ) {
            if( rank[ u ] < rank[ v ] ) {
                cnt[ v ] += cnt[ u ];
                par[ u ] = par[ v ];
                return cnt[v];
            } else {
                rank[ u ] = max( rank[ u ], rank[ v ] + 1 );
                cnt[ u ] += cnt[ v ];
                par[ v ] = par[ u ];
            }
        }
        return cnt[u];
    }
};
```

2.3 HLD

```
/**

Heavy Light Decomposition

Problem (lightoj 1348 - Aladin & return journey)

Description:

    ** graph -> for storing initial graph
    ** parent -> for storing parents
    ** ChainHead -> head of each chain
    ** ChainPos -> position of each node in it's chain
    ** ChainInd -> chain number of each node
    ** chainNumber -> it's the chain count, initialy it's 0

Steps:

    1) Set chainNumber to 0
    2) Run dfs function from the root node to store subtree size
    3) Run HLD function from root to generate HLD

**/



int chainHead[MX];
int chainPos[MX];
int chainInd[MX];
int parent[MX];
int subTreeSize[MX];
int chainNumber;
vector <int> graph[MX];
vector <int> chain[MX];

void dfs(int x) {
    vis[x] = 1;
    int cnt,i,j;
    cnt = 1;
    for (i=0; i<graph[x].size(); i++) {
        if (!vis[graph[x][i]]) {
            dfs(graph[x][i]);
            cnt += subTreeSize[graph[x][i]];
            parent[graph[x][i]] = x;
        }
    }
    subTreeSize[x] = cnt;
}

void hld(int x) {
    if (chainHead[chainNumber] == -1) chainHead[chainNumber] = x;
    chain[chainNumber].PB(vi[x]);
    chainInd[x] = chainNumber;
    chainPos[x] = chain[chainNumber].size()-1;
    int ind = -1, mx = -1, i, j, u, v;
    for (i=0; i<graph[x].size(); i++) {
        u = graph[x][i];
        if (chainInd[u] == -1 && subTreeSize[u] > mx) {
            mx = subTreeSize[u];
            ind = u;
        }
    }
    if (ind != -1) {
        hld(ind);
    }
    for (i=0; i<graph[x].size(); i++) {
        u = graph[x][i];
        if (chainInd[u] == -1) {
            chainNumber++;
            hld(u);
        }
    }
}
```

2.4 LCA

```
//LCA using sparse table
//Complexity: O(NlgN,lgN)
#define mx 100002
int L[mx]; //
int P[mx][22]; //
int T[mx]; //
vector<int>g[mx];
void dfs(int from,int u,int dep)
{
    T[u]=from;
    L[u]=dep;
    for(int i=0;i<(int)g[u].size();i++)
    {
```

```
        int v=g[u][i];
        if(v==from) continue;
        dfs(u,v,dep+1);
    }
}

int lca_query(int N, int p, int q) //N=
  {
      int tmp, log, i;

      if (L[p] < L[q])
          tmp = p, p = q, q = tmp;

        log=1;
      while(1) {
        int next=log+1;
        if((1<<next)>L[p])break;
        log++;

      }

        for (i = log; i >= 0; i--)
          if (L[p] - (1 << i) >= L[q])
              p = P[p][i];

      if (p == q)
          return p;

        for (i = log; i >= 0; i--)
          if (P[p][i] != -1 && P[p][i] != P[q][i])
              p = P[p][i], q = P[q][i];

      return T[p];
  }

void lca_init(int N)
  {
      memset (P,-1,sizeof(P)); //
                    _
      int i, j;
       for (i = 0; i < N; i++)
          P[i][0] = T[i];

      for (j = 1; 1 << j < N; j++)
         for (i = 0; i < N; i++)
             if (P[i][j - 1] != -1)
                 P[i][j] = P[P[i][j - 1]][j - 1];
}
```

2.5 Mo's Algo

```
/**
    Implementation of Mo's Algo with SQRT-Decomposition Data Structure
    Running time:
        O( ( n + q ) * sqrt( n ) * f() )
    Mo's Algo is a algorithm to process queries offline
    For it to work, this condition must be satisified:
        1) There can be no updates in the array
        2) All queries must be known beforehand
    Tested Problems:
      CF:
        220B - Little Elephant and Array
**/
#include <bits/stdc++.h>
using namespace std;

using piii = pair < pair < int, int >, int >;
const int mx = 1e5 + 1;
int BLOCK_SIZE;
int n, m;
int calc;
int ar[mx];
int ans[mx];
unordered_map < int, int > cnt;
piii query[mx];

struct {
    bool operator()( const piii &a, const piii &b ) {
        int block_a = a.first.first / BLOCK_SIZE;
        int block_b = b.first.first / BLOCK_SIZE;
        if( block_a != block_b ) {
            return block_a < block_b;
        }
        return a.first.second < b.first.second;
    }
} cmp;

void add( int x ) {
```

```
    calc -= ( cnt[x] == x ? 1 : 0 );
    cnt[x]++;
    calc += ( cnt[x] == x ? 1 : 0 );
}

void remove( int x ) {
    calc -= ( cnt[x] == x ? 1 : 0 );
    cnt[x]--;
    calc += ( cnt[x] == x ? 1 : 0 );
}

int main() {
    #ifdef LU_SERIOUS
        freopen( "in.txt", "r", stdin );
//      freopen( "out.txt", "w+", stdout );
    #endif // LU_SERIOUS
    while( ~scanf( "%d %d", &n, &m ) ) {

        BLOCK_SIZE = sqrt( n );
        cnt.clear();
        calc = 0;

        for( int i=0; i<n; i++ ) scanf( "%d", ar+i );

        for( int i=0; i<m; i++ ) {
            scanf( "%d %d", &query[i].first.first, &query[i].first.second
                 );
            query[i].second = i;
        }

        sort( query, query+m, cmp );

        int mo_l = 0, mo_r = -1;

        for( int i=0; i<m; i++ ) {
            int left = query[i].first.first - 1;
            int right = query[i].first.second - 1;

            while( mo_r < right ) {
                mo_r++;
                add( ar[mo_r] );
            }

            while( mo_r > right ) {
                remove( ar[mo_r] );
                mo_r--;
            }

            while( mo_l < left ) {
                remove( ar[mo_l] );
                mo_l++;
            }

            while( mo_l > left ) {
                mo_l--;
                add( ar[mo_l] );
            }

            ans[ query[i].second ] = calc;
        }

        for( int i=0; i<m; i++ ) {
            printf( "%d\n", ans[i] );
        }
    }
    return 0;
}
```

2.6 SQRT Decomposition

```
/**
    Implementation of SQRT-Decomposition Data Structure
    Running time:
        O( ( n + q ) * sqrt( n ) * f() )
    Usage:
        - call int() to initialize the array
        - call update() to update the element in a position
        - call query() to get ans from segment [L...R]
    Input:
        - n, number of elements
        - n elements
        - q queries
    Tested Problems:
      lightOJ:
        1082 - Array Queries
**/
#include <bits/stdc++.h>
using namespace std;
```

```
const int mx = 1e5 + 1;
const int sz = 1e3 + 1;
const int inf = 1e9;
int BLOCK_SIZE;
int n, q, t, cs, x, y;
int BLOCKS[sz];
int ar[mx];

int getID( int idx ) {
    return idx / BLOCK_SIZE;
}

void init() {
    for( int i=0; i<sz; i++ ) BLOCKS[i] = inf;
}

void update( int idx, int val ) {
    int id = getID( idx );
    BLOCKS[id] = min( val, BLOCKS[id] );
}

int query( int l, int r ) {
    int le = getID( l );
    int ri = getID( r );
    int ret = inf;

    if( le == ri ) {
        for( int i=l; i<=r; i++ ) {
            ret = min( ret, ar[i] );
        }
        return ret;
    }

    for( int i=l; i<(le+1)*BLOCK_SIZE; i++ ) ret = min( ret, ar[i] );
    for( int i=le+1; i<ri; i++ ) ret = min( ret, BLOCKS[i] );
    for( int i=ri*BLOCK_SIZE; i<=r; i++ ) ret = min( ret, ar[i] );

    return ret;
}

int main() {
    #ifdef LU_SERIOUS
        freopen( "in.txt", "r", stdin );
//      freopen( "out.txt", "w+", stdout );
    #endif // LU_SERIOUS
    scanf( "%d", &t );
    for( cs=1; cs<=t; cs++ ) {
        scanf( "%d %d", &n, &q );
        BLOCK_SIZE = sqrt( n );
        init();
        for( int i=0; i<n; i++ ) {
            scanf( "%d", &ar[i] );
            update( i, ar[i] );
        }
        printf( "Case %d:\n", cs );
        for( int i=0; i<q; i++ ) {
            scanf( "%d %d", &x, &y );
            printf( "%d\n", query( x-1, y-1 ) );
        }
    }
    return 0;
}
```

2.7 STL order statistics tree II

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int,null_type,less<int>,rb_tree_tag,
tree_order_statistics_node_update> order_set;

order_set X;

int get(int y) {
  int l=0,r=1e9+1;
  while(l<r) {
    int m=l+((r-l)>>1);
    if(m-X.order_of_key(m+1)<y)
      l=m+1;
    else
      r=m;
  }
  return l;
```

```
}

main(){
  ios::sync_with_stdio(0);
  cin.tie(0);
  int n,m;
  cin>>n>>m;

  for(int i=0;i<m;i++) {
    char a;
    int b;
    cin>>a>>b;
    if(a=='L')
      cout<<get(b)<<endl;
    else
      X.insert(get(b));
  }
}

/***
Input
20 7
L 5
D 5
L 4
L 5
D 5
L 4
L 5

Output
5
4
6
4
7
***/
```

2.8 STL order statistics tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <bits/stdc++.h>

using namespace __gnu_pbds;
using namespace std;

typedef
tree<
  pair<int,int>,
  null_type,
  less<pair<int,int>>,
  rb_tree_tag,
  tree_order_statistics_node_update>
ordered_set;

main()
 {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n;
    int sz=0;
    cin>>n;
    vector<int> ans(n,0);

    ordered_set t;
    int x,y;
    for(int i=0;i<n;i++)
    {
        cin>>x>>y;
        ans[t.order_of_key({x,++sz})]++;
        t.insert({x,sz});
    }

    for(int i=0;i<n;i++)
        cout<<ans[i]<<'\n';
}

/***
Input
5
1 1
5 1
7 1
3 3
5 5

Output
```

```
1
2
1
1
0
***/
```

2.9 Segment Tree

```
struct info {
    int prop, sum;
} tree[ mx * 3 ];

void update( int node, int b, int e, int i, int j, int x ) {
//    cerr << b << " " << e << " " << i << " " << j << " " << x << "\n";
    if( i > e || j < b ) {
        return;
    }
    if( b >= i && e <= j ) {
        tree[node].sum = ( e - b + 1 ) * x;
        tree[node].prop = x;
        return;
    }

    int left = node << 1;
    int right = left | 1;
    int mid = (b + e) >> 1;

    if( tree[node].prop != -1 ) {
        tree[left].sum = ( mid - b + 1 ) * tree[node].prop;
        tree[right].sum = ( e - mid ) * tree[node].prop;
        tree[node].sum = tree[left].sum + tree[right].sum;
        tree[left].prop = tree[node].prop;
        tree[right].prop = tree[node].prop;
        tree[node].prop = -1;
    }

    update(left, b, mid, i, j, x);
    update(right, mid + 1, e, i, j, x);

    tree[node].sum = tree[left].sum + tree[right].sum;
}
```

```
int query( int node, int b, int e, int i, int j ) {
    if( i > e || j < b ) {
        return 0;
    }
    if( b >= i and e <= j ) {
        return tree[node].sum;
    }

    int left = node << 1;
    int right = left | 1;
    int mid = (b + e) >> 1;

    if( tree[node].prop != -1 ) {
        tree[left].sum = ( mid - b + 1 ) * tree[node].prop;
        tree[right].sum = ( e - mid ) * tree[node].prop;
        tree[node].sum = tree[left].sum + tree[right].sum;
        tree[left].prop = tree[node].prop;
        tree[right].prop = tree[node].prop;
        tree[node].prop = -1;
    }

    int p1 = query( left, b, mid, i, j );
    int p2 = query( right, mid + 1, e, i, j );


    return p1 + p2;
}
```

2.10 hash table

```
/**
 * Micro hash table, can be used as a set.
 *   Very efficient vs std::set
 * */

const int MN = 1001;
struct ht {
  int _s[(MN + 10) >> 5];
  int len;
  void set(int id) {
    len++;
    _s[id >> 5] |= (1LL << (id & 31));
  }
```

```
  bool is_set(int id) {
    return _s[id >> 5] & (1LL << (id & 31));
  }
};
```

2.11 persistent seg tree

```
/**
 * Important:
 * When using lazy propagation remembert to create new
 * versions for each push_down operation!!!
 * */

struct node {
  node *l, *r;
  long long acc;
  int flip;

  node (int x) : l(NULL), r(NULL), acc(x), flip(0) {}
  node () : l(NULL), r(NULL), acc(0), flip(0) {}
};

typedef node* pnode;

pnode create(int l, int r) {
  if (l == r) return new node();
  pnode cur = new node();
  int m = (l + r) >> 1;
  cur-> l = create(l, m);
  cur-> r = create(m + 1, r);
  return cur;
}

pnode copy_node(pnode cur) {
  pnode ans = new node();
  *ans = *cur;
  return ans;
}

void push_down(pnode cur, int l, int r) {
  assert(cur);
  if (cur-> flip) {
    int len = r - l + 1;
    cur-> acc = len - cur-> acc;
    if (cur-> l) {
      cur-> l = copy_node(cur-> l);
      cur-> l -> flip ^= 1;
    }
    if (cur-> r) {
      cur-> r = copy_node(cur-> r);
      cur-> r -> flip ^= 1;
    }
    cur-> flip = 0;
  }
}

int get_val(pnode cur) {
  assert(cur);
  assert((cur-> flip) == 0);
  if (cur) return cur-> acc;
  return 0;
}

pnode update(pnode cur, int l, int r, int at, int what) {
  pnode ans = copy_node(cur);
  if (l == r) {
    assert(l == at);
    ans-> acc = what;
    ans-> flip = 0;
    return ans;
  }
  int m = (l + r) >> 1;
  push_down(ans, l, r);
  if (at <= m) ans-> l = update(ans-> l, l, m, at, what);
  else ans-> r = update(ans-> r, m + 1, r, at, what);

  push_down(ans-> l, l, m);
  push_down(ans-> r, m + 1, r);
  ans-> acc = get_val(ans-> l) + get_val(ans-> r);
  return ans;
}

pnode flip(pnode cur, int l, int r, int a, int b) {
  pnode ans = new node();

  if (cur != NULL) {
    *ans = *cur;
  }
```

```
  if (l > b || r < a)
    return ans;

  if (l >= a && r <= b) {
    ans-> flip ^= 1;
    push_down(ans, l, r);
    return ans;
  }

  int m = (l + r) >> 1;
  ans-> l = flip(ans-> l, l, m, a, b);
  ans-> r = flip(ans-> r, m + 1, r, a, b);
  push_down(ans-> l, l, m);
  push_down(ans-> r, m + 1, r);
  ans-> acc = get_val(ans-> l) + get_val(ans-> r);
  return ans;
}

long long get_all(pnode cur, int l, int r) {
  assert(cur);
  push_down(cur, l, r);
  return cur-> acc;
}

void traverse(pnode cur, int l, int r) {
  if (!cur) return;
  cout << l << " - " << r << " : " << (cur-> acc) << " " << (cur-> flip)
      << endl;
  traverse(cur-> l, l, (l + r) >> 1);
  traverse(cur-> l, 1 + ((l + r) >> 1), r);
}
```

2.12 sliding window

```
/*
 * Given an array ARR and an integer K, the problem boils down to
     computing for each index i: min(ARR[i], ARR[i-1], ..., ARR[i-K+1]).
 * if mx == true, returns the maximun.
 * http://people.cs.uct.ac.za/~ksmith/articles/sliding_window_minimum.html
 * */

vector<int> sliding_window_minmax(vector<int> & ARR, int K, bool mx) {
  deque< pair<int, int> > window;
  vector<int> ans;
  for (int i = 0; i < ARR.size(); i++) {
    if (mx) {
      while (!window.empty() && window.back().first <= ARR[i])
        window.pop_back();
    } else {
      while (!window.empty() && window.back().first >= ARR[i])
        window.pop_back();
    }
    window.push_back(make_pair(ARR[i], i));

    while(window.front().second <= i - K)
      window.pop_front();

    ans.push_back(window.front().first);
  }
  return ans;
}
```

2.13 trie xor

```
#define MAX 500001

struct trie
{
    int cand[2];
    trie()
    {
        clrall(cand,-1);
    }
};

trie tree[MAX*32+7];
ll csum;

int tot_node;

void insert_trie(int root,ll val)
{
    int i,j,k;
    int fbit;
    rvp(i,0,32)
```

```
    {
        fbit=(int) ((val>>(ll) i)&1LL);
        if(tree[root].cand[fbit]==-1)
        {
            tree[root].cand[fbit] = ++tot_node;
        }
        root = tree[root].cand[fbit];
    }
    return ;
}

int delete_trie(int root,ll val,int i)
{
    if(i==-1) return 0;
    int fbit;
    fbit=(int) ((val>>(ll) i)&1LL);
    if(tree[root].cand[fbit]==-1) return 0;
    int chld=delete_trie(tree[root].cand[fbit],val,i-1);
    if(chld==0)
    {
        tree[root].cand[fbit]=-1;
    }
    int nchld=0;
    if(tree[root].cand[fbit]!=-1) nchld++;
    if(tree[root].cand[!fbit]!=-1) nchld++;
    return nchld;
}

ll solve(int root,ll cval)
{
    ll res=0;
    int fbit,cbit;
    int i,j,k;
    rvp(i,0,32)
    {
        fbit=(int) ((cval>>(ll) i)&1LL);
        cbit=!(fbit);
        if(tree[root].cand[fbit]!=-1)
        {
            if(fbit) res|=(1LL << (ll) i);
            root=tree[root].cand[fbit];
        }
        else
        {
            if(cbit) res|=(1LL << (ll) i);
            root=tree[root].cand[cbit];
        }
    }
    return res;
}

ll max_val(ll val)
{
    int i,j,k;
    ll ret=0;
    int gbit;
    rvp(i,0,32)
    {
        gbit=(int) ((val>>(ll) i)&1LL);
        if(!gbit) ret|=(1LL << (ll) i);
    }
    return ret;
}
```

2.14 trie

```
struct node
{
        bool endmark;
        node *next[26+1];
        node()
        {
                endmark=false;
                for(int i=0;i<26;i++) next[i]=NULL;
        }
}*root;

void insert(char *str,int len)
{
        node *curr=root;
        for(int i=0;i<len;i++)
        {
                int id=str[i]-'a';
                if(curr->next[id]==NULL)
                        curr->next[id]=new node();
                curr=curr->next[id];
        }
        curr->endmark=true;
```

```
}
bool search(char *str,int len)
{
        node *curr=root;
        for(int i=0;i<len;i++)
        {
                int id=str[i]-'a';
                if(curr->next[id]==NULL) return false;
                curr=curr->next[id];
        }
        return curr->endmark;
}
void del(node *cur)
{
      for(int i=0;i<26;i++)
             if(cur->next[i])
                  del(cur->next[i]) ;


         delete(cur) ;
}
```

3 Geometry

3.1 closest pair problem

```
struct point {
  double x, y;
  int id;
  point() {}
  point (double a, double b) : x(a), y(b) {}
};

double dist(const point &o, const point &p) {
  double a = p.x - o.x, b = p.y - o.y;
  return sqrt(a * a + b * b);
}

double cp(vector<point> &p, vector<point> &x, vector<point> &y) {
  if (p.size() < 4) {
    double best = 1e100;
    for (int i = 0; i < p.size(); ++i)
      for (int j = i + 1; j < p.size(); ++j)
        best = min(best, dist(p[i], p[j]));
    return best;
  }

  int ls = (p.size() + 1) >> 1;
  double l = (p[ls - 1].x + p[ls].x) * 0.5;
  vector<point> xl(ls), xr(p.size() - ls);
  unordered_set<int> left;
  for (int i = 0; i < ls; ++i) {
    xl[i] = x[i];
    left.insert(x[i].id);
  }
  for (int i = ls; i < p.size(); ++i) {
    xr[i - ls] = x[i];
  }

  vector<point> yl, yr;
  vector<point> pl, pr;
  yl.reserve(ls); yr.reserve(p.size() - ls);
  pl.reserve(ls); pr.reserve(p.size() - ls);
  for (int i = 0; i < p.size(); ++i) {
    if (left.count(y[i].id))
      yl.push_back(y[i]);
    else
      yr.push_back(y[i]);

    if (left.count(p[i].id))
      pl.push_back(p[i]);
    else
      pr.push_back(p[i]);
  }

  double dl = cp(pl, xl, yl);
  double dr = cp(pr, xr, yr);
  double d = min(dl, dr);
  vector<point> yp; yp.reserve(p.size());
  for (int i = 0; i < p.size(); ++i) {
    if (fabs(y[i].x - l) < d)
      yp.push_back(y[i]);
  }
  for (int i = 0; i < yp.size(); ++i) {
    for (int j = i + 1; j < yp.size() && j < i + 7; ++j) {
      d = min(d, dist(yp[i], yp[j]));
```

```
    }
  }
  return d;
}

double closest_pair(vector<point> &p) {
  vector<point> x(p.begin(), p.end());
  sort(x.begin(), x.end(), [](const point &a, const point &b) {
    return a.x < b.x;
  });
  vector<point> y(p.begin(), p.end());
  sort(y.begin(), y.end(), [](const point &a, const point &b) {
    return a.y < b.y;
  });
  return cp(p, x, y);
}
```

3.2 convex hull

```
/**

CONVEX HULL

Definition:
    ** Uses vp as an inner temporary vector to store hull
    ** Uses hull vector to store the hull

**/


vector <g_point> vp;
vector <g_point> hull;

g_point pivot;

bool cmp(g_point p, g_point q)
{
    if (g_collinear(p,pivot,q))
    {
        return g_dist(pivot,p) < g_dist(pivot,q);
    }
    double dx1,dx2,dy1,dy2;
    dx1 = p.x - pivot.x;
    dx2 = q.x - pivot.x;
    dy1 = p.y - pivot.y;
    dy2 = q.y - pivot.y;
    return atan2(dy1, dx1) < atan2(dy2, dx2); // atan2 function is used
        for double
}

void buildhull() // works if n>=3
{
    int i,j,id;
    double x,y;
    g_point p,mx;
    hull.clear();
    mx = vp[0];
    id = 0;
    for (i=0;i<vp.size();i++)
    {
        p = vp[i];
        if (p.y < mx.y || (p.y == mx.y && p.x > mx.x))
        {
            mx = p;
            id = i;
        }
    }
    swap(vp[0],vp[id]);
    pivot = mx;
    sort(vp.begin()+1,vp.end(),cmp);
    vector <g_point> stk;
    int sz = vp.size() - 1;
    stk.PB(vp[sz]);
    stk.PB(vp[0]);
    stk.PB(vp[1]);
    i = 2;
    while (i < n)
    {
        p = vp[i];
        sz = stk.size() - 1;
        if (g_ccw(stk[sz],stk[sz-1],p))
        {
            stk.PB(p);
            i++;
        }
        else
        {
            stk.pop_back();
```

```
      }
    }
    swap(hull,stk);
}
```

3.3 squares

```
typedef long double ld;

const ld eps = 1e-12;
int cmp(ld x, ld y = 0, ld tol = eps) {
    return ( x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

struct point{
  ld x, y;
  point(ld a, ld b) : x(a), y(b) {}
  point() {}
};

struct square{
  ld x1, x2, y1, y2,
     a, b, c;
  point edges[4];
  square(ld _a, ld _b, ld _c) {
    a = _a, b = _b, c = _c;
    x1 = a - c * 0.5;
    x2 = a + c * 0.5;
    y1 = b - c * 0.5;
    y2 = b + c * 0.5;
    edges[0] = point(x1, y1);
    edges[1] = point(x2, y1);
    edges[2] = point(x2, y2);
    edges[3] = point(x1, y2);
  }
};

ld min_dist(point &a, point &b) {
  ld x = a.x - b.x,
     y = a.y - b.y;
  return sqrt(x * x + y * y);
}

bool point_in_box(square s1, point p) {
  if (cmp(s1.x1, p.x) != 1 && cmp(s1.x2, p.x) != -1 &&
      cmp(s1.y1, p.y) != 1 && cmp(s1.y2, p.y) != -1)
    return true;
  return false;
}

bool inside(square &s1, square &s2) {
  for (int i = 0; i < 4; ++i)
    if (point_in_box(s2, s1.edges[i]))
      return true;

  return false;
}

bool inside_vert(square &s1, square &s2) {
  if ((cmp(s1.y1, s2.y1) != -1 && cmp(s1.y1, s2.y2) != 1) ||
      (cmp(s1.y2, s2.y1) != -1 && cmp(s1.y2, s2.y2) != 1))
    return true;
 return false;
}

bool inside_hori(square &s1, square &s2) {
  if ((cmp(s1.x1, s2.x1) != -1 && cmp(s1.x1, s2.x2) != 1) ||
      (cmp(s1.x2, s2.x1) != -1 && cmp(s1.x2, s2.x2) != 1))
    return true;
 return false;
}

ld min_dist(square &s1, square &s2) {
  if (inside(s1, s2) || inside(s2, s1))
    return 0;

  ld ans = 1e100;
  for (int i = 0; i < 4; ++i)
    for (int j = 0; j < 4; ++j)
      ans = min(ans, min_dist(s1.edges[i], s2.edges[j]));


  if (inside_hori(s1, s2) || inside_hori(s2, s1)) {
    if (cmp(s1.y1, s2.y2) != -1)
      ans = min(ans, s1.y1 - s2.y2);
    else
    if (cmp(s2.y1, s1.y2) != -1)
```

```
      ans = min(ans, s2.y1 - s1.y2);
  }

  if (inside_vert(s1, s2) || inside_vert(s2, s1)) {
    if (cmp(s1.x1, s2.x2) != -1)
      ans = min(ans, s1.x1 - s2.x2);
    else
    if (cmp(s2.x1, s1.x2) != -1)
      ans = min(ans, s2.x1 - s1.x2);
  }

  return ans;
}
```

3.4 template

```
const double EPS = 1e-9;
const double PI = acos(-1.0);

/** GeometryStructures **/

struct g_point
{
    double x,y;
    g_point()
    {
        x = y = 0;
    }
    g_point(double _x,double _y)
    {
        x = _x; y = _y;
    }
    bool operator < (g_point other) const
    {
        if (fabs(x - other.x) > EPS)
        {
            return x < other.x;
        }
        return y < other.y;
    }
    bool operator == (g_point other) const
    {
        return ((fabs(x - other.x) < EPS) && (fabs(y - other.y)));
    }
};

struct g_line
{
    double a,b,c;
    g_line ()
    {
        a = b = c = 0;
    }
    g_line(double _a,double _b, double _c)
    {
        a = _a; b = _b; c = _c;
    }
};

struct g_vector
{
    double x,y;
    g_vector (double _x, double _y)
    {
        x = _x;
        y = _y;
    }
};


/**  Function List **/

//Geometry
double DEG_to_RAD (double theta); ///converts degree to radian
double RAD_to_DEG (double theta); ///converst radian to degree
double g_dist(g_point p1, g_point p2); ///finds euclidian distance
     between two points
g_point g_rotate(g_point p, double theta); ///rotates point 'p' by
     'theta' degrees
void g_pointsToLine (g_point p1, g_point p2, g_line &l); ///initiates
     line 'l'
bool g_areParallel(g_line l1, g_line l2); ///returns true if two lines
     are parallel
bool g_areSame(g_line l1, g_line l2); ///returns true if two lines are
     same or two segments (l1 and l2) are in same line
bool g_areIntersect(g_line l1, g_line l2, g_point &p); ///returns true if
     two lines intersect, sets the point of intersect as 'p'
```

```cpp
g_vector g_toVec(g_point a, g_point b); ///returns a vector from point
    'a' -> 'b'
g_vector g_scale(g_vector v, double s); ///returns a vector scaled or
    multiplied by 's'
g_point g_translate(g_point p, g_vector v); ///returns a point which is a
    translation of 'p'
double g_dot (g_vector a, g_vector b); ///returns dot product of two
    vectors
double g_cross(g_vector a, g_vector b); ///returns cross product of two
    vectors
double g_norm_sq(g_vector v); ///returns v.x * v.x + v.y * v.y, essential
    for finding distance of point to line segment and angle between
    points.
double g_distToLine(g_point p, g_point a, g_point b, g_point &c);
    ///returns shortest distance from point 'p' to line a->b, stores
    closest point to as 'c'
double g_distToLineSegment(g_point p, g_point a, g_point b, g_point &c);
    /// returns shortest distance from point 'p' to lineSegment a->b,
    stores closest point to as 'c'
double g_angle(g_point a, g_point o, g_point b); ///return angle <aob
bool g_ccw(g_point q, g_point p, g_point r); ///returns true if 'r' in
    left side of line p->q (counter clock-wise test)
bool g_cw(g_point q, g_point p, g_point r); ///returns true if 'r' in
    right side of line p->q (clock-wise test)
bool g_collinear (g_point q, g_point p, g_point r); ///returns true if
    three points are collinear;

int GCD (int x, int y){if (x%y==0) return y; else return (GCD(y,x%y));}

int main()
{
    #ifdef O_Amay_Valo_Basheni
        freopen("get.txt","r",stdin);
    #endif // O_Amay_Valo_Basheni
    g_point a(2,2),b(4,3),c(3,2);
    g_vector ab = g_toVec(a,b);
    //ab.x/=2; ab.y/=2;
    c = g_translate(c,ab);
    cout<<c.x<<" "<<c.y<<endl;
    c = g_rotate(c,360);
    cout<<c.x<<" "<<c.y<<endl;
    c = g_rotate(c,180);
    cout<<c.x<<" "<<c.y<<endl;
//    double d = DEG_to_RAD(360);
//    c = g_rotate(c,d);
//    cout<<c.x<<" "<<c.y<<endl;
    return 0;
}




/** GeometryFunctions **/

double DEG_to_RAD (double theta)
{
    return ((theta * PI)/180.0);
}

double RAD_to_DEG (double theta)
{
    return ((theta * 180.0) /PI);
}

double g_dist(g_point p1, g_point p2)
{
    return hypot(p1.x - p2.x, p1.y - p2.y);
}

g_point g_rotate(g_point p, double theta)
{
    double rad = DEG_to_RAD(theta);
    //rad = theta;
    //rad = (theta *(180.0/PI));
    return g_point (p.x * cos(rad) - p.y * sin(rad), p.x * sin(rad) + p.y
        * cos (rad));
}

void g_pointsToLine (g_point p1, g_point p2, g_line &l)
{
    if (fabs(p1.x - p2.x) < EPS)
    {
        l.a = 1.0; l.b = 0.0; l.c = -p1.x;
    }
    else
    {
        l.b = 1.0;
        l.a = -(double) (p1.y - p2.y)/ (p1.x - p2.x);
        l.c = -(double) (l.a * p1.x) - p1.y;
    }
```

```
}

bool g_areParallel(g_line l1, g_line l2)
{
    return ((fabs (l1.a - l2.a) < EPS) && (fabs(l1.b - l2.b) < EPS));
}

bool g_areSame(g_line l1, g_line l2)
{
    return ((g_areParallel(l1,l2)) && (fabs(l1.c - l2.c) < EPS));
}

bool g_areIntersect(g_line l1, g_line l2, g_point &p)
{
    if (g_areParallel(l1,l2)) return false;
    p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
    if (fabs(l1.b) > EPS)
        p.y = -(l1.a * p.x + l1.c);
    else
        p.y = -(l2.a * p.x + l2.c);
    return true;
}

g_vector g_toVec(g_point a, g_point b)
{
    return g_vector (b.x - a.x, b.y - a.y);
}

g_vector g_scale(g_vector v, double s)
{
    return g_vector(v.x * s, v.y * s);
}

g_point g_translate(g_point p, g_vector v)
{
    return g_point(p.x + v.x, p.y + v.y);
}

double g_dot(g_vector a, g_vector b)
{
    return (a.x * b.x + a.y * b.y);
}

double g_cross(g_vector a, g_vector b)
{
```

```
    return (a.x * b.y - a.y * b.x);
}

double g_norm_sq(g_vector v)
{
    return v.x * v.x + v.y * v.y;
}

double g_distToLine(g_point p, g_point a, g_point b, g_point &c)
{
    g_vector ap = g_toVec(a,p);
    g_vector ab = g_toVec(a,b);
    double u = g_dot(ap,ab) / g_norm_sq(ab);
    c = g_translate(a,g_scale(ab,u));
    return g_dist (p,c);
}

double g_distToLineSegment(g_point p, g_point a, g_point b, g_point &c)
{
    g_vector ap = g_toVec(a,p);
    g_vector ab = g_toVec(a,b);
    double u = g_dot(ap,ab) / g_norm_sq(ab);
    if (u < 0.0)
    {
        return g_dist(p,a);
    }
    if (u > 1.0)
    {
        return g_dist(p,b);
    }
    c = g_translate(a,g_scale(ab,u));
    return g_dist(p,c);
}

double g_angle(g_point a, g_point o, g_point b)
{
    g_vector oa = g_toVec(o,a);
    g_vector ob = g_toVec(o,b);
    return acos (g_dot(oa,ob) / sqrt(g_norm_sq(oa) * g_norm_sq(ob)));
}

bool g_ccw(g_point q, g_point p, g_point r)
{
    return g_cross (g_toVec(p,q),g_toVec(p,r)) > 0;
}
```

```cpp
bool g_cw(g_point q, g_point p, g_point r)
{
    return g_cross (g_toVec(p,q),g_toVec(p,r)) < 0;
}

bool g_collinear (g_point q, g_point p, g_point r)
{
    return fabs(g_cross(g_toVec(p,q),g_toVec(p,r))) < EPS;
}
```

3.5 triangles

Let a, b, c be length of the three sides of a triangle.

$$p = (a + b + c) * 0.5$$

The inradius is defined by:

$$iR = \sqrt{\frac{(p - a)(p - b)(p - c)}{p}}$$

The radius of its circumcircle is given by the formula:

$$cR = \frac{abc}{\sqrt{(a + b + c)(a + b - c)(a + c - b)(b + c - a)}}$$

4 Graph

4.1 Articulation Point

```cpp
/**
An O(V+E) approach:
--------------------
- perform a DFS on the graph.
- compute d(i) and low(i) for each vertex 1 ... i
       d(i):  dfs number of i, represents the discovery time.
       low(i): the least dfn reachable from i throguh a path consisting
           of zero or
              more edges follwoing by zero or one back edges.
- vertex u is an AP if and only if:
       - u is the root of the dfs tree and has at least two children.
       - u is not the root and has a child v for which low(v) >= d(u).
 **/

const int mx = 1e4 + 10;
vector < int > G[mx];
int tim, root, n, m, a, b;
int ap[mx], vis[mx], low[mx], d[mx], par[mx];

void ap_dfs(int u) {
    tim++;
    int cnt = 0;
    low[u] = tim;
    d[u] = tim;
    vis[u] = 1;
    int v;
    for(int i=0; i<G[u].size(); i++) {
        v = G[u][i];
        if( v == par[u] ) continue;
        if( !vis[v] ) {
            par[u] = v;
            ap_dfs( v );
            low[u] = min( low[u], low[v] );
            /// d[u] < low[v] if bridge is needed
            if( d[u] <= low[v] && u != root ) {
                ap[u] = 1;
            }
            cnt++;
        } else {
            low[u] = min( low[u], d[v] );
        }
        if( u == root && cnt > 1 ) ap[u] = 1;
    }
}
```

4.2 Bellman Ford

```cpp
struct Edge {
    int u, v, w;
};
vector < Edge > vs;
int n, dis[205];

void bell() {
    dis[1] = 0;
```

```
    for(int i = 1; i < n; i++) {
        for(int j = 0; j < vs.size(); j++) {
            if( dis[ vs[j].v ] > dis[ vs[j].u ] + vs[j].w ) {
                dis[vs[j].v] = dis[vs[j].u] + vs[j].w;
            }
        }
    }
}
```

4.3 DAG Check

```
vector < int > G[mx], tops;
bool vis[mx], bg[mx];

bool dfs( int u ) {
    bool ret = true;
    if( !vis[u] ) {
        vis[u] = 1;
        bg[u] = 1;
        int v;
        for( int i=0; i<G[u].size(); i++ ) {
            v = G[u][i];
            if( !vis[v] ) {
                ret &= dfs( v );
            }
            if( bg[v] ) return false;
        }
    }
    bg[u] = false;
    return true;
}

bool dag( int n) {
    memset( vis, 0, sizeof vis );
    memset( bg, 0, sizeof bg );
    bool ret = true;
    for( int i=1; i<=n; i++ ) {
        if( !vis[i] ) {
            ret &= dfs( i );
        }
    }
```

4.4 Dinic

```
// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source and sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain actual flow values, look at edges with capacity > 0
//       (zero capacity edges are residual edges).

#include<cstdio>
#include<vector>
#include<queue>
using namespace std;
typedef long long LL;

struct Edge {
  int u, v;
  LL cap, flow;
  Edge() {}
  Edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
  int N;
  vector<Edge> E;
  vector<vector<int>> g;
  vector<int> d, pt;

  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

  void AddEdge(int u, int v, LL cap) {
    if (u != v) {
      E.emplace_back(Edge(u, v, cap));
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(Edge(v, u, 0));
      g[v].emplace_back(E.size() - 1);
    }
```

```
}

bool BFS(int S, int T) {
  queue<int> q({S});
  fill(d.begin(), d.end(), N + 1);
  d[S] = 0;
  while(!q.empty()) {
    int u = q.front(); q.pop();
    if (u == T) break;
    for (int k: g[u]) {
      Edge &e = E[k];
      if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
        d[e.v] = d[e.u] + 1;
        q.emplace(e.v);
      }
    }
  }
  return d[T] != N + 1;
}

LL DFS(int u, int T, LL flow = -1) {
  if (u == T || flow == 0) return flow;
  for (int &i = pt[u]; i < g[u].size(); ++i) {
    Edge &e = E[g[u][i]];
    Edge &oe = E[g[u][i]^1];
    if (d[e.v] == d[e.u] + 1) {
      LL amt = e.cap - e.flow;
      if (flow != -1 && amt > flow) amt = flow;
      if (LL pushed = DFS(e.v, T, amt)) {
        e.flow += pushed;
        oe.flow -= pushed;
        return pushed;
      }
    }
  }
  return 0;
}

LL MaxFlow(int S, int T) {
  LL total = 0;
  while (BFS(S, T)) {
    fill(pt.begin(), pt.end(), 0);
    while (LL flow = DFS(S, T))
      total += flow;
  }
    return total;
  }
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum Flow
    (FASTFLOW)

int main()
{
  int N, E;
  scanf("%d%d", &N, &E);
  Dinic dinic(N);
  for(int i = 0; i < E; i++)
  {
    int u, v;
    LL cap;
    scanf("%d%d%lld", &u, &v, &cap);
    dinic.AddEdge(u - 1, v - 1, cap);
    dinic.AddEdge(v - 1, u - 1, cap);
  }
  printf("%lld\n", dinic.MaxFlow(0, N - 1));
  return 0;
}

// END CUT
```

4.5 Edmonds Karp

```
/**
    Implementation of Edmonds-Karp max flow algorithm
    Running time:
        O(|V|*|E|^2)
    Usage:
        - add edges by add_edge()
        - call max_flow() to get maximum flow in the graph
    Input:
        - n, number of nodes
        - directed, true if the graph is directed
        - graph, constructed using add_edge()
        - source, sink
    Output:
        - Maximum flow
```

```
    Tested Problems:
      CF:
        653D - Delivery Bears
      UVA:
        820 - Internet Bandwidth
        10330 - Power Transmission
**/

#include <bits/stdc++.h>
using namespace std;

const int INF = 1e9;

struct edmonds_karp {
    int n;
    vector < int > par;
    vector < bool > vis;
    vector < vector < int > > graph;

    edmonds_karp () {}
    edmonds_karp( int _n ) : n( _n ), par( _n ), vis( _n ), graph( _n,
        vector< int > ( _n, 0 ) ) {}
    ~edmonds_karp() {}

    void add_edge( int from, int to, int cap, bool directed ) {
        this->graph[ from ][ to ] += cap;
        this->graph[ to ][ from ] = directed ? graph[ to ][ from ] + cap :
            graph[ to ][ from ] ;
    }

    bool bfs( int src, int sink ) {
        int u;
        fill( vis.begin(), vis.end(), false );
        fill( par.begin(), par.end(), -1 );
        vis[ src ] = true;
        queue < int > q;
        q.push( src );
        while( !q.empty() ) {
            u = q.front();
            q.pop();
            if( u == sink ) return true;
            for(int i=0; i<n; i++) {
                if( graph[u][i] > 0 and not vis[i] ) {
                    q.push( i );
                    vis[ i ] = true;
                    par[ i ] = u;
                }
            }
        }
        return par[ sink ] != -1;
    }

    int min_val( int i ) {
        int ret = INF;
        for( ; par[ i ] != -1; i = par[ i ] ) {
            ret = min( ret, graph[ par[i] ][ i ] );
        }
        return ret;
    }

    void augment_path( int val, int i ) {
        for( ; par[ i ] != -1; i = par[ i ] ) {
            graph[ par[i] ][ i ] -= val;
            graph[ i ][ par[i] ] += val;
        }
    }

    int max_flow( int src, int sink ) {
        int min_cap, ret = 0;
        while( bfs( src, sink ) ) {
            augment_path( min_cap = min_val( sink ), sink );
            ret += min_cap;
        }
        return ret;
    }
};
```

4.6 EulerianPath

```
struct Edge;
typedef list<Edge>::iterator iter;

struct Edge
{
        int next_vertex;
        iter reverse_edge;

        Edge(int next_vertex)
```

```
                :next_vertex(next_vertex)
                { }
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];       // adjacency list

vector<int> path;

void find_path(int v)
{
        while(adj[v].size() > 0)
        {
                int vn = adj[v].front().next_vertex;
                adj[vn].erase(adj[v].front().reverse_edge);
                adj[v].pop_front();
                find_path(vn);
        }
        path.push_back(v);
}

void add_edge(int a, int b)
{
        adj[a].push_front(Edge(b));
        iter ita = adj[a].begin();
        adj[b].push_front(Edge(a));
        iter itb = adj[b].begin();
        ita->reverse_edge = itb;
        itb->reverse_edge = ita;
}
```

4.7 Floyed Warshall

```
/**
    Implementation of Floyd Warshall Alogrithm
    Running time:
        O( |v| ^ 3 )
    Input:
        - n, number vertex
        - graph, inputed as an adjacency matrix
    Tested Problems:
      UVA:
        544 - Heavy Cargo - MaxiMin path
        567 - Risk - APSP
**/

using vi = vector < int >;
using vvi = vector < vi >;

/// mat[i][i] = 0, mat[i][j] = distance from i to j, path[i][j] = i
void APSP( vvi &mat, vvi &path ) {
    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {
        for( int from=0; from<V; from++ ) {
            for( int to=0; to<V; to++ ) {
                if( mat[ from ][ via ] + mat[ via ][ to ] < mat[ from ][
                    to ] ) {
                    mat[ from ][ to ] = mat[ from ][ via ] + mat[ via ][ to
                        ];
                    path[ from ][ to ] = path[ via ][ to ];
                }
            }
        }
    }
}

/// prints the path from i to j
void print( int i, int j ) {
    if( i != j ) {
        print( i, path[i][j] );
    }
    cout << j << "\n";
}

/// check if negative cycle exists
bool negative_cycle( vvi &mat ) {
    APSP( mat );
    return mat[0][0] < 0;
}

void transtitive_closure( vvi &mat ) {
    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {
        for( int from=0; from<V; from++ ) {
            for( int to=0; to<V; to++ ) {
                mat[ from ][ to ] |= ( mat[ from ][ via ] & mat[ via ][ to
                    ] );
```

```
            }
        }
    }
}

/// finding a path between two nodes that maximizes the minimum cost
void mini_max( vvi &mat ) {
    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {
        for( int from=0; from<V; from++ ) {
            for( int to=0; to<V; to++ ) {
                mat[ from ][ to ] = min( mat[ from ][ to ], max( mat[ from
                    ][ via ], mat[ via ][ to ] ) );
            }
        }
    }
}

/// finding a path between two nodes that minimizes the maximum cost
/// eg: max load a truck can carry from one node to another node where
/// the paths have weight limit
void maxi_min( vvi &mat ) {
    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {
        for( int from=0; from<V; from++ ) {
            for( int to=0; to<V; to++ ) {
                mat[ from ][ to ] = max( mat[ from ][ to ], min( mat[ from
                    ][ via ], mat[ via ][ to ] ) );
            }
        }
    }
}
```

4.8 Kruskal

```
/**
    Implementation of Kruskal's minimum spanning tree algorithm
    Running time:
        O(|E|log|V|)
    Usage:
        - initialize by calling init()
        - add edges by add_edge()
        - call kruskal() to generate minimum spanning tree
    Input:
        - n, number of nodes, provided when init() is called
        - graph, constructed using add_edge()
    Output:
        - weight of minimum spanning tree
        - prints the mst
    Tested Problems:
        UVA:
            1208 - Oreon
*/

vector< edge > edges;
vector < int > par, cnt, rank;
int N;


int kruskal() {
    int ret = 0;
    make_set();
    sort( edges.begin(), edges.end() );
    cout << "Case " << ++cs << ":\n";
    for( edge e : edges ) {
        int u = e.u;
        int v = e.v;
        if( ( u = find_rep( u ) ) != ( v = find_rep( v ) ) ) {
            if( rank[ u ] < rank[ v ] ) {
                cnt[ v ] += cnt[ u ];
                par[ u ] = par[ v ];
            } else {
                rank[ u ] = max( rank[ u ], rank[ v ] + 1 );
                cnt[ u ] += cnt[ v ];
                par[ v ] = par[ u ];
            }
            cout << city[ e.u ] << "-" << city[ e.v ] << " " << e.cost <<
                "\n";
            ret += e.cost;
        }
    }
    return ret;
}
```

4.9 Max BPM

```
typedef long long int ll;
typedef vector<int> VI;
typedef vector<VI> VVI;
const int inf = 1e9;
int clr[1001];
VVI g;

bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {
    for (int j = 0; j < w[i].size(); j++) {
        if (w[i][j] && !seen[j]) {
            seen[j] = true;
            if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
                mr[i] = j;
                mc[j] = i;
                return true;
            }
        }
    }
    return false;
}

int BipartiteMatching(const VVI &w ) {
    VI mr = VI(w.size(), -1);
    VI mc = VI(w[0].size(), -1);

    int ct = 0;
    for (int i = 0; i < w.size(); i++) {
        VI seen(w[0].size());
        if (FindMatch(i, w, mr, mc, seen)) ct++;
    }
    return ct;
}
```

4.10 MinCostMatching

```
///////////////////////////////////////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs. In practice, it solves 1000x1000 problems in around 1
// second.
//
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative. To perform
// maximization, simply negate the cost[][] matrix.
///////////////////////////////////////////////////////////////////////

#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }

  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
        Lmate[i] = j;
        Rmate[j] = i;
        mated++;
```

```
        break;
      }
    }
  }

  VD dist(n);
  VI dad(n);
  VI seen(n);

  // repeat until primal solution is feasible
  while (mated < n) {

    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;

    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
      dist[k] = cost[s][k] - u[s] - v[k];

    int j = 0;
    while (true) {

      // find closest
      j = -1;
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;

      // termination condition
      if (Rmate[j] == -1) break;

      // relax neighbors
      const int i = Rmate[j];
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
        if (dist[k] > new_dist) {
          dist[k] = new_dist;
          dad[k] = j;
        }
      }
    }

    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];

    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;

    mated++;
  }

  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];

  return value;
}
```

4.11 MinCostMaxFlow

```
// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972). This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]). For a regular max flow, set all edge costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:           O(|V|^3) augmentations
//     min cost max flow:  O(|V|^4 * MAX_EDGE_COST) augmentations
//
```

```cpp
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive values only.

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
  int N;
  VVL cap, flow, cost;
  VI found;
  VL dist, pi, width;
  VPII dad;

  MinCostMaxFlow(int N) :
    N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}

  void AddEdge(int from, int to, L cap, L cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }

  void Relax(int s, int k, L cap, L cost, int dir) {
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++)
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
```

```
};

// BEGIN CUT
// The following code solves UVA problem #10594: Data Flow

int main() {
  int N, M;

  while (scanf("%d%d", &N, &M) == 2) {
    VVL v(M, VL(3));
    for (int i = 0; i < M; i++)
      scanf("%Ld%Ld%Ld", &v[i][0], &v[i][1], &v[i][2]);
    L D, K;
    scanf("%Ld%Ld", &D, &K);

    MinCostMaxFlow mcmf(N+1);
    for (int i = 0; i < M; i++) {
      mcmf.AddEdge(int(v[i][0]), int(v[i][1]), K, v[i][2]);
      mcmf.AddEdge(int(v[i][1]), int(v[i][0]), K, v[i][2]);
    }
    mcmf.AddEdge(0, 1, D, 0);

    pair<L, L> res = mcmf.GetMaxFlow(0, N);

    if (res.first == D) {
      printf("%Ld\n", res.second);
    } else {
      printf("Impossible.\n");
    }
  }

  return 0;
}

// END CUT
```

4.12 SCC Kosaraju

```
#include <bits/stdc++.h>
using namespace std;
int p, t;
bool vis[1001];
vector<int> G[1001], gT[1001];
map<string,int> mp;
stack < int > top_sorted;

void dfs_top_sort(int u) {
	vis[u] = true;
	for(int v: G[u]) {
		if(!vis[v]) {
			dfs_top_sort( v );
		}
	}
	top_sorted.push( u );
}

void top_sort() {
	for(int i=1; i<=p; i++) {
		if(!vis[i]) {
			dfs_top_sort(i);
		}
	}
}

void dfs_kosaraju(int u) {
	vis[u] = true;
	for(int v: gT[u]) {
		if(!vis[v]) {
			dfs_kosaraju( v );
		}
	}
}

int kosaraju() {
	memset( vis, false, sizeof(vis) );
	top_sort();
	int u, ret = 0;
	memset( vis, false, sizeof(vis) );
	while(!top_sorted.empty()) {
		u = top_sorted.top();
		top_sorted.pop();
		if(!vis[u])
			dfs_kosaraju( u ), ret++;
	}
	return ret;
}
```

4.13 directed mst

```
const int inf = 1000000 + 10;

struct edge {
  int u, v, w;
  edge() {}
  edge(int a,int b,int c) : u(a), v(b), w(c) {}
};

/**
 * Computes the minimum spanning tree for a directed graph
 * - edges : Graph description in the form of list of edges.
 *    each edge is: From node u to node v with cost w
 * - root : Id of the node to start the DMST.
 * - n    : Number of nodes in the graph.
 * */

int dmst(vector<edge> &edges, int root, int n) {
  int ans = 0;
  int cur_nodes = n;
  while (true) {
    vector<int> lo(cur_nodes, inf), pi(cur_nodes, inf);
    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      if (w < lo[v] and u != v) {
        lo[v] = w;
        pi[v] = u;
      }
    }

    lo[root] = 0;
    for (int i = 0; i < lo.size(); ++i) {
      if (i == root) continue;
      if (lo[i] == inf) return -1;
    }
    int cur_id = 0;
    vector<int> id(cur_nodes, -1), mark(cur_nodes, -1);
    for (int i = 0; i < cur_nodes; ++i) {
      ans += lo[i];
      int u = i;
      while (u != root and id[u] < 0 and mark[u] != i) {
        mark[u] = i;
        u = pi[u];
      }
      if (u != root and id[u] < 0) { // Cycle
        for (int v = pi[u]; v != u; v = pi[v])
          id[v] = cur_id;
        id[u] = cur_id++;
      }
    }

    if (cur_id == 0)
      break;

    for (int i = 0; i < cur_nodes; ++i)
      if (id[i] < 0) id[i] = cur_id++;

    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      edges[i].u = id[u];
      edges[i].v = id[v];
      if (id[u] != id[v])
        edges[i].w -= lo[v];
    }
    cur_nodes = cur_id;
    root = id[root];
  }

  return ans;
}
```

4.14 konig's theorem

In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

4.15 minimum path cover in DAG

Given a directed acyclic graph $G = (V, E)$, we are to find the minimum number of vertex-disjoint paths to cover each vertex in V.

We can construct a bipartite graph $G' = (Vout \cup Vin, E')$ from G, where :

$$Vout = \{v \in V : v \ has \ positive \ out - degree\}$$

$$Vin = \{v \in V : v \ has \ positive \ in - degree\}$$

$$E' = \{(u, v) \in Vout \times Vin : (u, v) \in E\}$$

Then it can be shown, via König's theorem, that G' has a matching of size m if and only if there exists $n - m$ vertex-disjoint paths that cover each vertex in G, where n is the number of vertices in G and m is the maximum cardinality bipartite mathching in G'.

Therefore, the problem can be solved by finding the maximum cardinality matching in G' instead.
NOTE: If the paths are note necesarily disjoints, find the transitive closure and solve the problem for disjoint paths.

4.16 planar graph (euler)

Euler's formula states that if a finite, connected, planar graph is drawn in the plane without any edge intersections, and v is the number of vertices, e is the number of edges and f is the number of faces (regions bounded by edges, including the outer, infinitely large region), then:

$$f + v = e + 2$$

It can be extended to non connected planar graphs with c connected components:

$$f + v = e + c + 1$$

4.17 stable marriage

```
int t, n;
int ar[201][101];
int partners[101];
int fl[101];

bool check( int w, int m, int m1 ) {
    for( int i=0; i<n; i++ ) {
        if( ar[w][i] == m1 ) return true;
        if( ar[w][i] == m ) return false;
    }
    return true;
}

void stable_match() {
    memset( partners, -1, sizeof partners );
    memset( fl, 0, sizeof fl );
    int cnt = n;
    int m, w, m1;
    while( cnt ) {
        for( m=0; m<n; m++ ) if( !fl[m] ) break;
        for( int i=0; i<n && !fl[m]; i++ ) {
            w = ar[m][i];
            if( partners[w-n] == - 1 ) {
                partners[w-n] = m;
                fl[m] = 1;
                cnt--;
            } else {
                m1 = partners[w-n];
                if( !check( w, m, m1 ) ) {
                    partners[w-n] = m;
                    fl[m] = 1;
                    fl[m1] = 0;
                }
            }
        }
    }
    for( int i=0; i<n; i++ ) {
        printf( " (%d %d)", partners[i]+1, i+n+1 );
    }
    printf( "\n" );
}
```

4.18 two sat (with kosaraju)

```
/**
 * Given a set of clauses (a1 v a2)^(a2 v a3)....
 * this algorithm find a solution to it set of clauses.
 * test: http://lightoj.com/volume_showproblem.php?problem=1251
 **/

#include<bits/stdc++.h>
using namespace std;
#define MAX 100000
#define endl '\n'

vector<int> G[MAX];
vector<int> GT[MAX];
vector<int> Ftime;
vector<vector<int> > SCC;
bool visited[MAX];
```

```cpp
int n;


void dfs1(int n){
  visited[n] = 1;

  for (int i = 0; i < G[n].size(); ++i) {
    int curr = G[n][i];
    if (visited[curr]) continue;
    dfs1(curr);
  }

  Ftime.push_back(n);
}

void dfs2(int n, vector<int> &scc) {
  visited[n] = 1;
  scc.push_back(n);

  for (int i = 0;i < GT[n].size(); ++i) {
    int curr = GT[n][i];
    if (visited[curr]) continue;
    dfs2(curr, scc);
  }
}


void kosaraju() {
  memset(visited, 0, sizeof visited);

  for (int i = 0; i < 2 * n ; ++i) {
    if (!visited[i]) dfs1(i);
  }

  memset(visited, 0, sizeof visited);
  for (int i = Ftime.size() - 1; i >= 0; i--) {
    if (visited[Ftime[i]]) continue;
    vector<int> _scc;
    dfs2(Ftime[i],_scc);
    SCC.push_back(_scc);
  }
}

/**
 * After having the SCC, we must traverse each scc, if in one SCC are -b
     y b, there is not a solution.
 * Otherwise we build a solution, making the first "node" that we find
     truth and its complement false.
 **/


bool two_sat(vector<int> &val) {
  kosaraju();
  for (int i = 0; i < SCC.size(); ++i) {
    vector<bool> tmpvisited(2 * n, false);
    for (int j = 0; j < SCC[i].size(); ++j) {
      if (tmpvisited[SCC[i][j] ^ 1]) return 0;
      if (val[SCC[i][j]] != -1) continue;
      else {
        val[SCC[i][j]] = 0;
        val[SCC[i][j] ^ 1] = 1;
      }
      tmpvisited[SCC[i][j]] = 1;
    }
  }
  return 1;
}

// Example of use

int main() {

  int m, u, v, nc = 0, t; cin >> t;
  // n = "nodes" number, m = clauses number

  while (t--) {
    cin >> m >> n;
    Ftime.clear();
    SCC.clear();
    for (int i = 0; i < 2 * n; ++i) {
      G[i].clear();
      GT[i].clear();
    }

    // (a1 v a2) = (a1 -> a2) = (a2 -> a1)
    for (int i = 0; i < m ; ++i) {
      cin >> u >> v;
      int t1 = abs(u) - 1;
      int t2 = abs(v) - 1;
```

```
    int p = t1 * 2 + ((u < 0)? 1 : 0);
    int q = t2 * 2 + ((v < 0)? 1 : 0);
    G[p ^ 1].push_back(q);
    G[q ^ 1].push_back(p);
    GT[p].push_back(q ^ 1);
    GT[q].push_back(p ^ 1);
  }

  vector<int> val(2 * n, -1);
  cout << "Case " << ++nc <<": ";
  if (two_sat(val)) {
    cout << "Yes" << endl;
    vector<int> sol;
    for (int i = 0; i < 2 * n; ++i)
      if (i % 2 == 0 and val[i] == 1)
        sol.push_back(i / 2 + 1);
    cout << sol.size() ;

    for (int i = 0; i < sol.size(); ++i) {
      cout << " " << sol[i];
    }
    cout << endl;
  } else {
    cout << "No" << endl;
  }
 }
 return 0;
}
```

5 Math

5.1 Lucas theorem

For non-negative integers m and n and a prime p, the following congruence relation holds: :

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p},$$

where :

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0,$$

and :

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$$

are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$ if $m \leq n$.

5.2 big mod

```
ll big_mod( ll b, ll p ) {
    ll ret = 1;
    for( ; p; p >>= 1 ) {
        if( p&1 ) ret = ( ret * b ) % mod;
        b = ( b * b ) % mod;
    }
    return ret % mod;
}

ll mod_inv( ll b ) {
        return big_mod( b, mod - 2 );
}
```

5.3 catalan

```
unsigned long int catalan(unsigned int n){
    // Base case
    if (n <= 1) return 1;

    // catalan(n) is sum of catalan(i)*catalan(n-i-1)
    unsigned long int res = 0;
    for (int i=0; i<n; i++)
        res += catalan(i)*catalan(n-i-1);

    return res;
}
```

5.4 convolution

```
typedef long long int LL;
typedef pair<LL, LL> PLL;

inline bool is_pow2(LL x) {
  return (x & (x-1)) == 0;
}
```

```
inline int ceil_log2(LL x) {
  int ans = 0;
  --x;
  while (x != 0) {
    x >>= 1;
    ans++;
  }
  return ans;
}

/* Returns the convolution of the two given vectors in time proportional
    to n*log(n).
 * The number of roots of unity to use nroots_unity must be set so that
     the product of the first
 * nroots_unity primes of the vector nth_roots_unity is greater than the
     maximum value of the
 * convolution. Never use sizes of vectors bigger than 2^24, if you need
     to change the values of
 * the nth roots of unity to appropriate primes for those sizes.
 */
vector<LL> convolve(const vector<LL> &a, const vector<LL> &b, int
    nroots_unity = 2) {
  int N = 1 << ceil_log2(a.size() + b.size());
  vector<LL> ans(N,0), fA(N), fB(N), fC(N);
  LL modulo = 1;
  for (int times = 0; times < nroots_unity; times++) {
    fill(fA.begin(), fA.end(), 0);
    fill(fB.begin(), fB.end(), 0);
    for (int i = 0; i < a.size(); i++) fA[i] = a[i];
    for (int i = 0; i < b.size(); i++) fB[i] = b[i];
    LL prime = nth_roots_unity[times].first;
    LL inv_modulo = mod_inv(modulo % prime, prime);
    LL normalize = mod_inv(N, prime);
    ntfft(fA, 1, nth_roots_unity[times]);
    ntfft(fB, 1, nth_roots_unity[times]);
    for (int i = 0; i < N; i++) fC[i] = (fA[i] * fB[i]) % prime;
    ntfft(fC, -1, nth_roots_unity[times]);
    for (int i = 0; i < N; i++) {
      LL curr = (fC[i] * normalize) % prime;
      LL k = (curr - (ans[i] % prime) + prime) % prime;
      k = (k * inv_modulo) % prime;
      ans[i] += modulo * k;
    }
    modulo *= prime;
  }
  return ans;
}
```

5.5 crt

```
/**
 * Chinese remainder theorem.
 * Find z such that z % x[i] = a[i] for all i.
 * */
long long crt(vector<long long> &a, vector<long long> &x) {
  long long z = 0;
  long long n = 1;
  for (int i = 0; i < x.size(); ++i)
    n *= x[i];

  for (int i = 0; i < a.size(); ++i) {
    long long tmp = (a[i] * (n / x[i])) % n;
    tmp = (tmp * mod_inv(n / x[i], x[i])) % n;
    z = (z + tmp) % n;
  }

  return (z + n) % n;
}
```

5.6 cumulative sum of divisors

```
/**
The function SOD(n) (sum of divisors) is defined
as the summation of all the actual divisors of
an integer number n. For example,

  SOD(24) = 2+3+4+6+8+12 = 35.

The function CSOD(n) (cumulative SOD) of an integer n, is defined as
    below:

  csod(n) = \sum_{i = 1}^{n} sod(i)

It can be computed in O(sqrt(n)):
*/
```

```
long long csod(long long n) {
  long long ans = 0;
  for (long long i = 2; i * i <= n; ++i) {
    long long j = n / i;
    ans += (i + j) * (j - i + 1) / 2;
    ans += i * (j - i);
  }
  return ans;
}
```

5.7 discrete logarithm

```
// Computes x which a ^ x = b mod n.

long long d_log(long long a, long long b, long long n) {
  long long m = ceil(sqrt(n));
  long long aj = 1;
  map<long long, long long> M;
  for (int i = 0; i < m; ++i) {
    if (!M.count(aj))
      M[aj] = i;
    aj = (aj * a) % n;
  }

  long long coef = mod_pow(a, n - 2, n);
  coef = mod_pow(coef, m, n);
  // coef = a ^ (-m)
  long long gamma = b;
  for (int i = 0; i < m; ++i) {
    if (M.count(gamma)) {
      return i * m + M[gamma];
    } else {
      gamma = (gamma * coef) % n;
    }
  }
  return -1;
}
```

5.8 ext euclidean

```
void ext_euclid(long long a, long long b, long long &x, long long &y,
    long long &g) {
  x = 0, y = 1, g = b;
  long long m, n, q, r;
  for (long long u = 1, v = 0; a != 0; g = a, a = r) {
    q = g / a, r = g % a;
    m = x - u * q, n = y - v * q;
    x = u, y = v, u = m, v = n;
  }
}
```

5.9 fibonacci properties

Let A, B and n be integer numbers.

$$k = A - B \tag{1}$$

$$F_A F_B = F_{k+1} F_A^2 + F_k F_A F_{A-1} \tag{2}$$

$$\sum_{i=0}^{n} F_i^2 = F_{n+1} F_n \tag{3}$$

$ev(n)$ = returns 1 if n is even.

$$\sum_{i=0}^{n} F_i F_{i+1} = F_{n+1}^2 - ev(n) \tag{4}$$

$$\sum_{i=0}^{n} F_i F_{i-1} = \sum_{i=0}^{n-1} F_i F_{i+1} \tag{5}$$

5.10 highest exponent factorial

```
int highest_exponent(int p, const int &n){
  int ans = 0;
  int t = p;
  while(t <= n){
    ans += n/t;
    t*=p;
  }
  return ans;
```

```
}
```

5.11 miller rabin

```
const int rounds = 20;

// checks whether a is a witness that n is not prime, 1 < a < n
bool witness(long long a, long long n) {
  // check as in Miller Rabin Primality Test described
  long long u = n - 1;
  int t = 0;
  while (u % 2 == 0) {
    t++;
    u >>= 1;
  }
  long long next = mod_pow(a, u, n);
  if (next == 1) return false;
  long long last;
  for (int i = 0; i < t; ++i) {
    last = next;
    next = mod_mul(last, last, n);
    if (next == 1) {
      return last != n - 1;
    }
  }
  return next != 1;
}


// Checks if a number is prime with prob 1 - 1 / (2 ^ it)
//  D(miller_rabin(99999999999999997LL) == 1);
//  D(miller_rabin(9999999999971LL) == 1);
//  D(miller_rabin(7907) == 1);
bool miller_rabin(long long n, int it = rounds) {
  if (n <= 1) return false;
  if (n == 2) return true;
  if (n % 2 == 0) return false;
  for (int i = 0; i < it; ++i) {
    long long a = rand() % (n - 1) + 1;
    if (witness(a, n)) {
      return false;
    }
  }
  return true;
}
```

5.12 mod inv

```
long long mod_inv(long long n, long long m) {
  long long x, y, gcd;
  ext_euclid(n, m, x, y, gcd);
  if (gcd != 1)
    return 0;
  return (x + m) % m;
}
```

5.13 mod mul

```
// Computes (a * b) % mod
long long mod_mul(long long a, long long b, long long mod) {
  long long x = 0, y = a % mod;
  while (b > 0) {
    if (b & 1)
      x = (x + y) % mod;
    y = (y * 2) % mod;
    b /= 2;
  }
  return x % mod;
}
```

5.14 mod pow

```
// Computes ( a ^ exp ) % mod.
long long mod_pow(long long a, long long exp, long long mod) {
  long long ans = 1;
  while (exp > 0) {
    if (exp & 1)
      ans = mod_mul(ans, a, mod);
    a = mod_mul(a, a, mod);
    exp >>= 1;
  }
  return ans;
```

```
}
```

5.15 number theoretic transform

```
typedef long long int LL;
typedef pair<LL, LL> PLL;

/* The following vector of pairs contains pairs (prime, generator)
 * where the prime has an Nth root of unity for N being a power of two.
 * The generator is a number g s.t g^(p-1)=1 (mod p)
 * but is different from 1 for all smaller powers */
vector<PLL> nth_roots_unity {
  {1224736769,330732430},{1711276033,927759239},{167772161,167489322},
   {469762049,343261969},{754974721,643797295},{1107296257,883865065}};

PLL ext_euclid(LL a, LL b) {
  if (b == 0)
    return make_pair(1,0);
  pair<LL,LL> rc = ext_euclid(b, a % b);
  return make_pair(rc.second, rc.first - (a / b) * rc.second);
}

//returns -1 if there is no unique modular inverse
LL mod_inv(LL x, LL modulo) {
  PLL p = ext_euclid(x, modulo);
  if ( (p.first * x + p.second * modulo) != 1 )
    return -1;
  return (p.first+modulo) % modulo;
}


//Number theory fft. The size of a must be a power of 2
void ntfft(vector<LL> &a, int dir, const PLL &root_unity) {
  int n = a.size();
  LL prime = root_unity.first;
  LL basew = mod_pow(root_unity.second, (prime-1) / n, prime);
  if (dir < 0) basew = mod_inv(basew, prime);
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    LL w = 1;
    for (int i = 0; i < mh; i++) {
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        LL x = (a[j] - a[k] + prime) % prime;
        a[j] = (a[j] + a[k]) % prime;
        a[k] = (w * x) % prime;
      }
      w = (w * basew) % prime;
    }
    basew = (basew * basew) % prime;
  }
  int i = 0;
  for (int j = 1; j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
}
```

5.16 pollard rho factorize

```
long long pollard_rho(long long n) {
  long long x, y, i = 1, k = 2, d;
  x = y = rand() % n;
  while (1) {
    ++i;
    x = mod_mul(x, x, n);
    x += 2;
    if (x >= n) x -= n;
    if (x == y) return 1;
    d = __gcd(abs(x - y), n);
    if (d != 1) return d;
    if (i == k) {
      y = x;
      k *= 2;
    }
  }
  return 1;
}


// Returns a list with the prime divisors of n
vector<long long> factorize(long long n) {
  vector<long long> ans;
  if (n == 1)
    return ans;
  if (miller_rabin(n)) {
```

```
    ans.push_back(n);
  } else {
    long long d = 1;
    while (d == 1)
      d = pollard_rho(n);
    vector<long long> dd = factorize(d);
    ans = factorize(n / d);
    for (int i = 0; i < dd.size(); ++i)
      ans.push_back(dd[i]);
  }
  return ans;
}
```

5.17 sievephi

```
void sievephi()
{
    mark[1] = 1;
    for(ll i = 1; i < Mx; i++)
    {
        phi[i] = i;
        if(!(i & 1)) mark[i] = 1, phi[i] /= 2;
    }
    mark[2] = 0;

    for(ll i = 3; i < Mx; i+=2)
    {
        if(!mark[i])
        {
            phi[i] = phi[i] - 1;
            for(ll j = 2 * i; j < Mx; j += i)
            {
                mark[j] = 1;
                phi[j] /= i;
                phi[j] *= i - 1;
            }
        }
    }
}
```

5.18 sigma function

the sigma function is defined as:

$$\sigma_x(n) = \sum_{d|n} d^x$$

when $x = 0$ is called the divisor function, that counts the number of positive divisors of n.

Now, we are interested in find

$$\sum_{d|n} \sigma_0(d)$$

if n is written as prime factorization:

$$n = \prod_{i=1}^{k} P_i^{e_k}$$

we can demonstrate that:

$$\sum_{d|n} \sigma_0(d) = \prod_{i=1}^{k} g(e_k + 1)$$

where $g(x)$ is the sum of the first x positive numbers:

$$g(x) = (x * (x + 1))/2$$

5.19 sigma

```
/// for i to n
///  ( pi^(xi+1) - 1 ) / ( pi - 1 ) )

ll ans() {
    ll ret = 1, cnt, h;
    for( int i=0; i < prime.size() && prime[i]*prime[i] <= n; i++ ) {
        if( n % prime[i] == 0 ) {
            cnt = 0;
            while( n % prime[i] == 0 ) {
                n /= prime[i];
                cnt++;
            }
            cnt *= m;
            cnt++;
```

```
            h = ( ( ( big_mod( prime[i], cnt ) - 1LL + mod ) % mod ) *
                big_mod( prime[i] - 1LL, mod - 2LL ) ) % mod;
            ret = ( ret * h ) % mod;
//          cerr << ( big_mod( prime[i], cnt + 1 ) - 1 ) << " - " << (
    powl( prime[i], cnt + 1 ) - 1 ) << "\n";
        }
    }
    if( n > 1 ) {
        h = ( ( ( big_mod( n, m + 1LL ) - 1LL + mod ) % mod ) * big_mod( n
            - 1LL, mod - 2LL ) ) % mod;
        ret = ( ret * h ) % mod;
    }
    return ret % mod;
}
```

5.20 totient sieve

```
  for (int i = 1; i < MN; i++)
    phi[i] = i;

  for (int i = 1; i < MN; i++)
    if (!sieve[i]) // is prime
      for (int j = i; j < MN; j += i)
        phi[j] -= phi[j] / i;
```

5.21 totient

```
long long totient(long long n) {
  if (n == 1) return 0;
  long long ans = n;
  for (int i = 0; primes[i] * primes[i] <= n; ++i) {
    if ((n % primes[i]) == 0) {
      while ((n % primes[i]) == 0) n /= primes[i];
      ans -= ans / primes[i];
    }
  }
  if (n > 1) {
    ans -= ans / n;
  }
  return ans;
}
```

6 Matrix

6.1 Gaussin Elimation

```
/**

GAUSSIAN ELIMINATION

Definition:
    ** for equation -> a1X + b1Y + c1Z = d1
    ** for equation -> a2X + b2Y + c2Z = d2
    ** for equation -> a3X + b3Y + c3Z = d3
    ** for equation -> a4X + b4Y + c4Z = d4
    we store the d in the B array
    we store the a, b, c in the A array

**/



double A[101][101], B[101];

void gausian(int n) {
    int i,j,k,l,e,mxIdx;
    double x,y,cns,cnt,ans,mxV;
    for (i=0,e=0; i<n; i++) {
        mxV = A[e][e];
        mxIdx = e;
        for (j=e; j<n; j++) {
            if (fabs(A[j][i]) > fabs(mxV)) {
                mxV = A[j][i];
                mxIdx = j;
            }
        }
        if (mxV < EPS) {
            e++;
            continue;
        }
        for (j=0; j<n; j++) {
            swap(A[mxIdx][j],A[e][j]);
        }
        swap(B[mxIdx],B[e]);
        for (k=0; k<n; k++) A[e][k] /= mxV;
        B[e] /= mxV;
```

```
        for (j=0; j<n; j++) {
            if (j != e) {
                x = A[j][e];
                if (fabs(x) >= EPS) {
                    for (k=0; k<n; k++) {
                        A[j][k] -= (A[e][k] * x);
                    }
                    B[j] -= (B[e] * x);
                }
            }
        }
        e++;
    }
    for (i=99; i>=0; i--) {
        for (j=99; j>=0; j--) {
            if (fabs(A[i][j]) >= EPS && i != j) {
                B[i] -= (A[i][j] * B[j]);
            }
            if (i == j && fabs(A[i][j]) >= EPS) {
                B[i] /= (A[i][j]);
            }
        }
    }
}
```

6.2 Matrix Expo

```
struct Matrix {
    const int mat_sz = 2;
    int a[mat_sz][mat_sz];
    void clear() {
        memset(a, 0, sizeof(a));
    }
    void one() {
        for( int i=0; i<mat_sz; i++ ) {
            for( int j=0; j<mat_sz; j++ ) {
                mat[i][j] = i == j;
            }
        }
    }
    Matrix operator + (const Matrix &b) const {
        Matrix tmp;
        tmp.clear();
        for (int i = 0; i < mat_sz; i++) {
            for (int j = 0; j < mat_sz; j++) {
                tmp.a[i][j] = a[i][j] + b.a[i][j];
                if (tmp.a[i][j] >= mod) {
                    tmp.a[i][j] -= mod;
                }
            }
        }
        return tmp;
    }
    Matrix operator * (const Matrix &b) const {
        Matrix tmp;
        tmp.clear();
        for (int i = 0; i < mat_sz; i++) {
            for (int j = 0; j < mat_sz; j++) {
                for (int k = 0; k < mat_sz; k++) {
                    tmp.a[i][k] += (long long)a[i][j] * b.a[j][k] % mod;
                    if (tmp.a[i][k] >= mod) {
                        tmp.a[i][k] -= mod;
                    }
                }
            }
        }
        return tmp;
    }
    Matrix pw(int x) {
        Matrix ans, num = *this;
        ans.one();
        while (x > 0) {
            if (x & 1) {
                ans = ans * num;
            }
            num = num * num;
            x >>= 1;
        }
        return ans;
    }
};
```

7 Misc

7.1 IO

```
inline int RI() {
    int  ret = 0, flag = 1,ip = getchar();
    for(; ip < 48 || ip > 57; ip = getchar()) {
        if(ip == 45) {
            flag = -1;
            ip = getchar();
            break;
        }
    }
    for(; ip > 47 && ip < 58; ip = getchar())
        ret = ret * 10 + ip - 48 ;
    return flag * ret;
}
```

8 String

8.1 KMP

```
/// complexity : o( n + m )
///solution reference loj 1255 Substring Frequency
#include <bits/stdc++.h>
using namespace std;

int t;
const int mx = 1e6 + 10;
char a[mx], b[mx];
int table[mx], lenA, lenB;

void hash_table( char *s ) {
        table[ 0 ] = 0;
        int i = 1, j = 0;
        while( i < lenB ) {
                if( s[i] == s[j] ) {
                        j++;
                        table[ i ] = j;
                        i++;
                } else {
                        if( j ) {
                                j = table[ j - 1 ];
                        } else {
                                table[ i ] = 0;
                                i++;
                        }
                }
        }
}

int kmp( char *s, char *m ) {
        hash_table( m );
        int i = 0, j = 0;
        int ans = 0;
        while( i < lenA ) {
                while( i < lenA && j < lenB && s[i] == m[j] ) {
                        i++;
                        j++;
                }
                if( j == lenB ) {
                        j = table[ j - 1 ];
                        ans++;
                } else if( i < lenA && s[i] != m[j] ) {
                        if( j ) {
                                j = table[ j - 1 ];
                        } else {
                                i++;
                        }
                }
        }
        return ans;
}

int main() {
#ifdef LU_SERIOUS
        freopen("in.txt", "r", stdin);
#endif // LU_SERIOUS
        scanf( "%d", &t );
        for(int cs=1; cs<=t; cs++) {
                lenA = 0; lenB = 0;
                scanf("%s", &a);
                scanf("%s", &b);
                lenA = strlen( a );
                lenB = strlen( b );
                printf( "Case %d: %d\n", cs, kmp( a, b ) );
        }
        return 0;
}
```

8.2 Manacher

```
int call(char *inp,char *str,int *F,vector< pair<int,int> > &vec){
    //inp is the actual string
    //str is the modified string with double size of inp
    //F[i] contains the length of the palindrome centered at index i
    //Every element of vec cointains starting and ending positions of
        palindromes
    int len=0;
    str[len++]='*';
    for(int i=0; inp[i]; i++){
        str[len++]=inp[i];
        str[len++]='*';
    }
    str[len]='\0';
    int c=0,r=0,ans=0;
    for(int i=1; i < len-1; i++){
        int _i=c-(i-c);
        if(r > i) F[i]=min(F[_i],r-i);
        else F[i]=0;
        while(i-1-F[i]>=0 && str[i-1-F[i]]==str[i+1+F[i]]) {
            F[i]++;
        }
        if(i+F[i] > r) r=i+F[i],c=i;
        ans=max(ans,F[i]);
        vec.push_back(make_pair(i-F[i],i+F[i]));
    }
    return ans;
}
```

8.3 Z algo

```
int L = 0, R = 0;
for( int i = 1; i < n; i++ ) {
       if ( i > R ) {
              L = R = i;
              while ( R < n && s[R-L] == s[R] ) R++;
              z[i] = R-L; R--;
       } else {
       int k = i-L;
              if ( z[k] < R-i+1 ) z[i] = z[k];
              else {
                     L = i;
                     while ( R < n && s[R-L] == s[R] ) R++;
                     z[i] = R-L; R--;
              }
       }
}
int maxz = 0, res = 0;
for ( int i = 1; i < n; i++ ) {
       if ( z[i] == n-i && maxz >= n-i ) { res = n-i; break; }
       maxz = max( maxz, z[i]) ;
}
```