Team Note of CU_ChunoPuti

Compiled on May 15, 2025

Contents

1 Have you...

1.1 tried...

- **Reading the problem once more?**
- doubting "obvious" things?
- writing obivous things?
- radical greedy approach?
- thinking in reverse direction?
- a greedy algorithm?
- network flow when your greedy algorithms stuck?
- a dynamic programming?
- checking the range of answer?
- random algorithm?
- graph modeling using states?
- inverting state only on odd indexes?
- square root decomposition?
- calculating error bound on a real number usage?
- pigeonhole principle?.. **Does the long range actually needed?**

1.2 checked...

- **you have read the statement correctly?**
- typo copying the team note?
- initialization on multiple test case problem?
- additional information from the problem?
- undefined behavior?
- overflow?
- function without return value?
- real number error?
- implicit conversion?
- comparison between signed and unsigned integer?

2 Data Structure

Should be **tested**.

2.1 Binary Lifting

Time Complexity: $Q \log n + n \log n$

```
const int mx=2e5+1,LOG=18;
vector<int>adj[mx];
int up[mx][LOG];
int depth[mx];
bool vis[mx];
void dfs(int u)
{
    vis[u]=1;
    for(int j=1;j<LOG;j++){
        up[u][j]=up[up[u][j-1]][j-1];
    }

    for(auto &v: adj[u]){
        if(!vis[v]){
            depth[v]=depth[u]+1;
            up[v][0]=u;
            dfs(v);
        }
    }
}
int get_lca(int u,int v){   //O(log(n))
    if(depth[u]<depth[v])   swap(u,v);
    int k=depth[u]-depth[v];
    for(int j=LOG-1;j>=0;j--){
        if(k&(1<<j)){
            u=up[u][j];
        }
    }
    if(u==v)    return u;
    else{
        for(int j=LOG-1;j>=0;j--){
            if(up[u][j]!=up[v][j]){
                u=up[u][j];
                v=up[v][j];
            }
        }
        return up[u][0];
    }
}
int main()
{
    up[1][0]=1;
    depth[1]=0;
    dfs(1);
    int lca=get_lca(u,v);
}
```

2.2 Segment Tree (Maximum Subsegment Sum)

```
typedef struct data{
    int ans;
    int sm;
    int mxpre;
    int mxsuf;
}data;
data make_data(int val)
{
    data ret;
    ret.ans=ret.sm=ret.mxpre=ret.mxsuf=val;
    return ret;
}
const int mx=50001;
int ar[mx];
data t[mx*4];

data combine(data l,data r){
    if(l.ans==-inf) return r;
    else if(r.ans==-inf)    return l;
    data res;
    res.ans=max({l.ans,r.ans,l.mxsuf+r.mxpre});
    res.mxpre=max({l.mxpre,l.sm+r.mxpre});
    res.mxsuf=max({r.mxsuf,r.sm+l.mxsuf});
    res.sm=l.sm+r.sm;
    return res;
}
void build(int i,int tl,int tr)
{
    if(tl==tr)  t[i]=make_data(ar[tl]);
    else{
        int tm=(tl+tr)/2;
        build(i*2,tl,tm);
        build(i*2+1,tm+1,tr);
        t[i]=combine(t[i*2],t[i*2+1]);
    }
}
data query(int i,int tl,int tr,int l,int r)
{
    if(l>tr || r<tl){
        return make_data(-inf);
    }
    if(tl>=l && tr<=r)  return t[i];

    int tm=(tl+tr)/2;
    data ret1=query(i*2,tl,tm,l,r);
    data ret2=query(i*2+1,tm+1,tr,l,r);
    return combine(ret1,ret2);
}
void update(int v,int tl,int tr,int pos,int new_val)
{
    if(tl==tr){
        ar[pos]=new_val;
        t[v]=make_data(new_val);
        return;
    }else{
        int tm=(tl+tr)/2;
        if(pos<=tm){
            update(v*2,tl,tm,pos,new_val);
        }else update(v*2+1,tm+1,tr,pos,new_val);
        t[v]=combine(t[v*2],t[v*2+1]);
    }
}

///returns the idx first element smaller than h within
//the the given range; min tree
int get_first(int v,int tl,int tr,int l,int r,int h)
{
    if(l>tr || r<tl)    return -1;
    if(tl>=l && tr<=r){
        if(t[v]>=h) return -1;
        while(tl!=tr){
            int tm=(tl+tr)/2;
            if(t[v*2]>=h){
                tl=tm+1;
                v=v*2+1;
            }else{
                tr=tm;
                v=v*2;
            }
        }
        return tl;
    }

    int tm=(tl+tr)/2;
    int rs=get_first(2*v,tl,tm,l,r,h);
    if(rs!=-1) return rs;
    else return get_first(v*2+1,tm+1,tr,l,r,h);
}
```

2.3 Segment Tree (Offline Query)

```
const int mx=1e5+1;
int mnt[mx*4];
int mxt[mx*4];
int r[mx];
typedef struct info{
    int pos;
    int type; ///3->query 1->left start, 2->right start,4->left end ,
    int id;
    info(int pos,int type,int id) : pos(pos) ,type(type) ,id(id){}
}info;
bool cmp(const info &p1,const info &p2)
{
    if(p1.pos<p2.pos)   return true;
    else if(p1.pos==p2.pos){
        if(p1.type<p2.type) return true;
        else if(p1.type==p2.type){
            return (p1.id<=p2.id);
```

```
        }else return false;
    }else return false;
}
void input(vector<info> &v,int n,int m)
{
    int a,b,pos;
    for(int i=1;i<=n;i++){
        cin>>a>>b;
        v.PB({a,1,i});
        v.PB({(a+b)/2,4,i});
        v.PB({(a+b+1)/2,2,i});
        v.PB({b,5,i});
        r[i]=b;
    }

    for(int i=1;i<=m;i++){
        cin>>pos;
        v.PB({pos,3,i});
    }
}
void update1(int v,int tl,int tr,int pos,int val)
{
    if(tl==tr)  mnt[v]=val;
    else {
        int tm=(tl+tr)/2;
        if(pos<=tm) update1(v*2,tl,tm,pos,val);
        else update1(v*2+1,tm+1,tr,pos,val);
        mnt[v]=min(mnt[v*2],mnt[v*2+1]);
    }
}
void update2(int v,int tl,int tr,int pos,int val)
{
    if(tl==tr)  mxt[v]=val;
    else {
        int tm=(tl+tr)/2;
        if(pos<=tm) update2(v*2,tl,tm,pos,val);
        else update2(v*2+1,tm+1,tr,pos,val);
        mxt[v]=max(mxt[v*2],mxt[v*2+1]);
    }
}
void solve(vector<int>&ans,vector<info>&v,int n,int m){
    for(auto &x: v){
        if(x.type==1) update1(1,1,n,x.id,x.pos);
        else if(x.type==4) update1(1,1,n,x.id,inf);
        else if(x.type==2) update2(1,1,n,x.id,r[x.id]);
        else if(x.type==5) update2(1,1,n,x.id,-inf);
        else{
            int req=0;
            int ret=mnt[1];
            if(ret!=inf)    req=max(req,(x.pos-ret));
            ret=mxt[1];
            if(ret!=(-inf))    req=max(req,(ret-x.pos));
            ans[x.id]=req;
        }
    }
}
signed main()
{
    int t,n,m;
    cin>>t;
    for(int tc=1;tc<=t;tc++){
        cin>>n>>m;
        vector<info>v;
        input(v,n,m);
        sort(v.begin(),v.end(),cmp);
        vector<int> ans(m+1);
        for(int i=0;i<=(n*4);i++)   mnt[i]=inf;
        for(int i=0;i<=(n*4);i++)   mxt[i]=-inf;
        solve(ans,v,n,m);
    }
    return 0;
}
```

2.4 BIT/FenwickTree

```
 int BIT[N];
 //it uses one-based indexing internally
void update(int x,int val,int n) {
    //++x;//needed if the actual array is 0 based index
    while(x<=n)
    {
      BIT[x]+=val  x+=(x&-x);
    }
}
int query(int x) {
    //++x; //needed if the actual array is 0 based index
    int res=0;
    while(x>0)
    {
      res+=BIT[x];  x-=(x&-x);
    }
    return res;
}
```

2.5 MO's algo

```
const int maxn =1e6+3;
const int mx=2e5+2;
int K, ans[mx], a[mx], sum;

struct Query{
    int index, L, R;
    bool operator < (const Query &other) const {
int block_own = L/K;
int block_other = other.L/K;
if(block_own == block_other)
            return R < other.R;
return block_own < block_other;
}
};
int cnt[maxn];
//You should overwrite just add and remove function ...
//hence i am using zero based indexing
//after taking L and R input
//they should be decremented
//n,q highest 10^5 otherwise TLE

void add(int idx){
   // sum+=a[index];
     int k=a[idx];
   cnt[k]++;

      if(cnt[k]==1)sum++;

}

void remove(int idx){
  //  sum-=a[index];
  int k=a[idx];

  cnt[k]--;
     if(cnt[k]==0)sum--;
 // if(cnt[a[idx]]==0)sum--;
}

signed  main(){

    int n,q;
    cin>>n;
    K = sqrt(n);
    for(int i=0;i<n;i++)
    {
      cin>>a[i];
     }
     int x,y;
    vector<Query>query;
     cin>>q;
    for(int i=0;i<q;i++){
        // in(x),in(y);
         cin>>x>>y;
         query.push_back({i,--x,--y});

    }
    sort(query.begin(), query.end());
    int L=0, R=-1;
    for(int i=0;i<q;i++){
        while(R<query[i].R) add(++R);
        while(L<query[i].L) remove(L++);
        while(R>query[i].R) remove(R--);
```

```
        while(L>query[i].L) add(--L);
        ans[query[i].index] = sum;
    }
    for(int i=0;i<q;i++){
     cout << ans[i] << "\n";
    }
    return 0;
}
```

2.6 DSU

```
const int mx=2e5+3;
 int parent[mx];
 int depth[mx];
void make_set(int v) {
    parent[v] = v;
    depth[v] = 1;
}
int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}
void union_set(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (depth[a] < depth[b])
            swap(a, b);
        parent[b] = a;
        depth[a] += depth[b];
    }
}
```

2.7 Trie/suffix tree

```
struct Trie {
  static const int B = 31;//nofbits in input number..
  struct node {
    node* nxt[2];
    int sz;
    node() {
      nxt[0] = nxt[1] = NULL;
      sz = 0;
    }
  }*root;
  Trie() {
    root = new node();
  }
  void insert(int val) {
    node* cur = root;
    cur -> sz++;
    for (int i = B - 1; i >= 0; i--) {
      int b = val >> i & 1;
      if (cur -> nxt[b] == NULL) cur -> nxt[b] = new node();
      cur = cur -> nxt[b];
      cur -> sz++;
    }
  }
  int query(int x, int k) { //n(values) s.t. val^x < k
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      if (cur == NULL) break;
      int b1 = x >> i & 1, b2 = k >> i & 1;
      if (b2 == 1) {
        if (cur -> nxt[b1]) ans += cur -> nxt[b1] -> sz;
        cur = cur -> nxt[!b1];
      } else cur = cur -> nxt[b1];
    }
    return ans;
  }
  int get_max(int x) { //returns maximum of val ^ x
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur -> nxt[!k]) cur = cur -> nxt[!k], ans |=
(1LL<<i);
      else cur = cur -> nxt[k];
    }
    return ans;
  }
  int get_min(int x) { // returns minimum of val ^ x
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur -> nxt[k]) cur = cur -> nxt[k];
      else cur = cur -> nxt[!k], ans |= (1LL<<i);
    }
    return ans;
  }
  void del(node* cur) {
    for (int i = 0; i < 2; i++) if (cur -> nxt[i]) del(cur ->
nxt[i]);
    delete(cur);
  }
} t;
```

2.8 Segment Tree-Lazy propagation

Usage: Following code is for "adding on segments, querying for maximum". For assainngment on range, a marked boolean array of segment tree size, can be maintained.

```
void build(int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(v*2, tl, tm);
        build(v*2+1, tm+1, tr);
        t[v] = max(t[v*2], t[v*2 + 1]);
    }
}

void push(int v) {
    t[v*2] += lazy[v];
    lazy[v*2] += lazy[v];
    t[v*2+1] += lazy[v];
    lazy[v*2+1] += lazy[v];
    lazy[v] = 0;
}

void update(int v, int tl, int tr, int l, int r, int addend) {
    if (l > r)
        return;
    if (l == tl && tr == r) {
        t[v] += addend;
        lazy[v] += addend;
    } else {
        push(v);
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm), addend);
        update(v*2+1, tm+1, tr, max(l, tm+1), r, addend);
        t[v] = max(t[v*2], t[v*2+1]);
    }
}

int query(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return -INF;
    if (l == tl && tr == r)
        return t[v];
    push(v);
    int tm = (tl + tr) / 2;
    return max(query(v*2, tl, tm, l, min(r, tm)),
               query(v*2+1, tm+1, tr, max(l, tm+1), r));
}
```

2.9 Sparse Talbe(1D)[S]

```
const int mx=1e5+1;
const int LOG=log2(mx)+1;
int table[mx][LOG],ar[mx],log_2[mx];
void buildSparseTable(int n)
{
```

```
    int k=log2(n)+1;
    for(int i=1;i<=n;i++)   table[i][0]=ar[i]; //2^0=1

    for(int j=1;j<k;j++){
        for(int i=1;(i+(1<<j)-1)<=n;i++){
            table[i][j]=min(table[i][j-1],
            table[i+(1<<(j-1))][j-1]);
        }
    }
}
int query(int l,int r)
{
    int len=r-l+1;
    //int k=log2(len);
    int k=log_2[len];
    return min(table[l][k],table[r-(1<<k)+1][k]);
}
int main()
{
    int n;
    cin>>n;
    log_2[1]=0;
    for(int i=2;i<=n;i++){
        log_2[i]=log_2[i/2]+1;
    }
    for(int i=1;i<=n;i++)   cin>>ar[i];
    buildSparseTable(n);
    int q,l,r;
    cin>>q;
    while(q--){
        cin>>l>>r;
        l++ , r++; ///0 based indexing to 1 based indexing
        cout<<query(l,r)<<endl;
    }
    return 0;
}
```

2.10 Sparse Talbe(2D)[S]

```
const int mx=1e3+2;
const int LOG=log2(mx)+1;
int table[mx][mx][LOG][LOG],ar[mx][mx];

void buildSparseTable(int n,int m)
{
    int log2m=log2(m),log2n=log2(n);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++)    table[i][0][j][0]=ar[i][j];

        for(int kc=1;kc<=log2m;kc++){ //kc-> k for column
            for(int j=1;(j+(1<<kc)-1)<=m;j++){//j->col of ith row
                table[i][0][j][kc]=max(table[i][0][j][kc-1],
                table[i][0][j+(1<<(kc-1))][kc-1]);
            }
        }
    }
    //The above step is nothing but computing
    the sparse table for each row.
    //The complexity for one row is O(m*logm) and so for all
    rows O(n*m*logm).

    for(int kr=1;kr<=(log2n);kr++){ //kr-> k for row
        for(int i=1;(i+(1<<kr)-1)<=n;i++){
            for(int kc=0;kc<=log2m;kc++){//kc -> k for column
                for(int j=1;(j+(1<<kc)-1)<=m;j++){  //column of ith row
                    table[i][kr][j][kc]=max(table[i][kr-1][j][kc],
                    table[i+(1<<(kr-1))][kr-1][j][kc]);
                }
            }
        }
    }
    //Clearly,the above step is O(n*m*logn*logm)
}

int query(int x1,int y1,int x2,int y2){ //O(1)
    int lenx=(x2-x1+1);
    int leny=(y2-y1+1);
    int kx=log2(lenx),ky=log2(leny);
    int minR1=max(table[x1][kx][y1][ky],
    table[x1][kx][y2-(1<<ky)+1][ky]);
    int minR2=max(table[x2-(1<<kx)+1][kx][y1][ky],
    table[x2-(1<<kx)+1][kx][y2-(1<<ky)+1][ky]);
    return max(minR1,minR2);
}

int chk(int x1,int y1,int x2,int y2)
{
    int l=0,r=min(x2-x1+1,y2-y1+1),mid,ret;
    while(l<r){
        mid=(l+r+1)/2;
        ret=query(x1,y1,x2-mid+1,y2-mid+1);
        if(ret>=mid){
            l=mid;
        }else r=mid-1;
    }
    return l;
}

signed main()
{
    optimize();
    int n,m,x1,y1,x2,y2,q;
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++)   cin>>ar[i][j];
    }
    for(int i=n;i>=1;i--){
        for(int j=m;j>=1;j--){
            if(ar[i][j]==1){
                int mn=min(ar[i][j+1],ar[i+1][j]);
                if(ar[i+mn][j+mn]!=0){
                    ar[i][j]=mn+1;
                }else ar[i][j]=mn;
            }
        }
    }
    buildSparseTable(n,m);
    cin>>q;
    while(q--){
        cin>>x1>>y1>>x2>>y2;
        int ret=chk(x1,y1,x2,y2);
        cout<<ret<<endl;
    }
    return 0;
}
```

2.11 hld

Usage: If you additionally calculate and store maximums of all prefixes for each heavy path, then you get a $\mathcal{O}(\log n)$ solution because all maximum queries are on prefixes except at most once when we reach the ancestor l.

Time Complexity: $\mathcal{O}(Q \log^2 n)$

```
const int mx=2e5+1;
int par[mx],depth[mx],heavy[mx],t[mx*4]; //heavy child:
intialize it as -1
int pos[mx],head[mx];//position in the segmentTree array and
head of the chain
int cur_pos;
vector<int>adj[mx];
int dfs(int v) //returns sub_tree size
{//also finds heavy child , parent and depth
    int sz=1; //do not use size
    int max_c_size=0;
    for(int c: adj[v]){
        if(c!=par[v]){
            par[c]=v,depth[c]=depth[v]+1;
            int c_size=dfs(c);
            sz+=c_size;
            if(c_size>max_c_size){
                max_c_size=c_size,heavy[v]=c;
            }
        }
    }
    return sz;
}
void decompose(int v,int h) //h->head of chain
```

```
{
    head[v]=h,pos[v]=cur_pos++;
    if(heavy[v]!=-1)    decompose(heavy[v],h);
    for(int c: adj[v]){
        if(c!=par[v] && c!=heavy[v])    decompose(c,c);
    }
}
void init(int n) {
    for(int i=1;i<=n;i++)   heavy[i]=-1;
    cur_pos = 0;
    depth[1]=0; //considering 1 as source node
    dfs(1);
    decompose(1, 1);
}
///add segment tree here
int query(int a,int b,int n)
{
    int res=0;
    for(;head[a]!=head[b];b=par[head[b]]){
        if(depth[head[a]]>depth[head[b]]){
           swap(a,b);}//depth of head of a will be smaller
        int
cur_heavy_path_max=segment_tree_query(1,1,n,pos[head[b]],pos[b]);  // read input data for the next query
        res=max(res,cur_heavy_path_max);
    }
    if(depth[a]>depth[b])//both in same chain now
        swap(a,b);//b is below a in reset to depth
    int last_heavy_path_max =
segment_tree_query(1,1,n,pos[a],pos[b]);
    res = max(res, last_heavy_path_max);
    return res;

}
```

2.12 mo cp-algo

```
void remove(idx);  // TODO: remove value at idx from
//data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of
the data structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect
    the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

2.13 Square Root Decomposition

```
// input data
int n;
vector<int> a (n);

// preprocessing
int len = (int) sqrt (n + .0) + 1; // size of the block and the numbe
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];

// answering the queries
for (;;) {
    int l, r;
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to [l, r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}
```

2.14 unorderdHashing$_s imilarTree$

```
ll bigMOD(ll a,ll n,ll mod){
    if(n==0)    return 1;
    ll ret=1;
    if(n&1){
        ret=(a*(bigMOD(a,n-1,mod)))%mod;
    }else{
        ret=bigMOD(a,n/2,mod);
        ret*=(ret);
        ret%=mod;
    }
    return ret;
}


pair<ll,ll> dfs(int &i,ll &cnt,ll level,ll &maxL,string &st)
{
    ll hasH1=0,hasH2=0;
    ll cnt2=1,maxL2=level;
    pair<ll,ll>p;

    while(st[i]=='1'){
        i++;
        p=dfs(i,cnt2,level+1,maxL2,st);
        hasH1+=p.first;
        hasH2+=p.second;
        hasH1%=MOD;
        hasH2%=MOD;
    }
    i++;
    hasH1+=((cnt2*bigMOD(10007,maxL2-level+1,MOD))%MOD);
    hasH1%=MOD;

    hasH2+=((cnt2*bigMOD(10033,maxL2-level+1,MOD))%MOD);
    hasH2%=MOD;

    maxL=max(maxL,maxL2);
    cnt+=cnt2;

    return {hasH1,hasH2};
}
```

```
int main()
{

    int t;
    cin>>t;
    string st,st2;
    getline(cin,st);
    for(int tc=1;tc<=t;tc++ ){
        getline(cin,st);
        getline(cin,st2);
        int i;
        ll cnt=0,mxL=0;

        cout<<"Case "<<tc<<": ";
        if(dfs(i=0,cnt=0,1,mxL,st)==dfs(i=0,cnt=0,1,mxL,st2)){
            cout<<"Same"<<endl;
        }else cout<<"Different"<<endl;

    }

    return 0;
}
```

2.15 Euler Tour On Tree (Sum root to node else include LCA)

```
const int mx=2e5+1;
const int LOG=20;
int v[mx],t[mx*8],ar[mx*2],in[mx],out[mx];
vector<int>adj[mx];
bool vis[mx];
int timeStamp;

void dfs(int u)
{
    timeStamp++;
    in[u]=timeStamp;
    vis[u]=1;
    ar[timeStamp]=v[u];

    for(auto &v: adj[u]){
        if(!vis[v]){
            dfs(v);
        }
    }
    timeStamp++;
    out[u]=timeStamp;
    ar[timeStamp]=-v[u];
}

void build(int v,int tl,int tr){
    if(tl==tr)  t[v]=ar[tl];
    else{
        int tm=(tl+tr)/2;
        build(v*2,tl,tm);
        build(v*2+1,tm+1,tr);
        t[v]=t[v*2]+t[v*2+1];
    }
}

void update(int v,int tl,int tr,int x,int val)
{
    if(tl==tr){
        ar[x]=val;
        t[v]=val;
    }else{
        int tm=(tl+tr)/2;
        if(x<=tm)   update(v*2,tl,tm,x,val);
        else update(v*2+1,tm+1,tr,x,val);
        t[v]=t[v*2]+t[v*2+1];
    }
}

int query(int v,int tl,int tr,int l,int r){
    if(r<tl || l>tr)    return 0;
    else if(tl>=l && tr<=r) return t[v];
    else{
        int tm=(tl+tr)/2;
        int ret=query(v*2,tl,tm,l,r);
        ret+=query(v*2+1,tm+1,tr,l,r);
        return ret;
    }
}

signed main()
{
    optimize();

    int n,q,a,b,s,x,type;
    cin>>n>>q;

    for(int i=1;i<=n;i++)   cin>>v[i];

    for(int i=1;i<n;i++){
        cin>>a>>b;
        adj[a].PB(b);
        adj[b].PB(a);
    }

    dfs(1);
    build(1,1,timeStamp);

    while(q--){
        cin>>type;
        if(type==1){
            cin>>a>>b;
            update(1,1,timeStamp,in[a],b);
            update(1,1,timeStamp,out[a],-b);
            v[a]=b;
        }else{
            cin>>a;
            int ans=query(1,1,timeStamp,in[1],in[a]);
            cout<<ans<<endl;
        }

    }

    return 0;
}
```

3 String Algorithm

3.1 String Hashing

```
const int mx=1e6+1;
 int  Hash[mx];
 int revHash[mx];
 const int mod=1e9+7;
 const int p=31;
 int InV[mx];
 long long binpow(long long a, long long b, long long m) {
    a %= m;
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
 void Hash_it( string rev)
 {
      int power=1;
      Hash[0]=0;
      revHash[0]=0;
      string s=rev;
      reverse(rev.begin(),rev.end());
        for(int i=0;i<s.size();i++)
        {
            int cur=s[i]-'a'+1;
            int cur1=rev[i]-'a'+1;
             Hash[i+1]=(Hash[i]+(cur*power)%mod)%mod;
            revHash[i+1]=(revHash[i]+(cur1*power)%mod)%mod;
               cur=binpow(power,mod-2,mod);
            InV[i]=cur;
            power=(power*p)%mod;
        }
```

```
}
int get_hash(int L,int R)
{
     //1-based indexing
     int res=Hash[R]  -Hash[L-1] ;
     if(L==1)return res;
     int kk;
     kk=(res*InV[L-1])%mod;
    if(kk<0)kk+=mod;
    return kk;

}

int get_hashrev(int L,int R,int n)
{
   //1-based indexing
     int l=L;
     L=(n-R)+1;
     R=(n-l)+1;
   int res=revHash[R]  -revHash[L-1] ;
     if(L==1)return res;
    int kk=(res*InV[L-1])%mod;
    if(kk<0)kk+=mod;
    return kk;

}
bool is_palin(int l,int r,int n)
{
     if(get_hash(l,r)==get_hashrev(l,r,n))return true;

   return false;
}
```

3.2 Suffix array with LCP-O(n)

```
    #include <bits/stdc++.h>
using namespace std;
const int MAX_N = 1e5;
char str[MAX_N];
int N, m, SA[MAX_N], LCP[MAX_N];
int x[MAX_N], y[MAX_N], w[MAX_N], c[MAX_N];

inline bool cmp(const int a, const int b, const int l)
{
return (y[a] == y[b] && y[a + l] == y[b + l]);
}

void Sort()
{
    for (int i = 0; i < m; ++i)
        w[i] = 0;
    for (int i = 0; i < N; ++i)
        ++w[x[y[i]]];
    for (int i = 0; i < m - 1; ++i)
        w[i + 1] += w[i];
    for (int i = N - 1; i >= 0; --i)
        SA[--w[x[y[i]]]] = y[i];
}

void DA()
{
    ++N;
    for (int i = 0; i < N; ++i)
        x[i] = str[i], y[i] = i;
    Sort();
    for (int i, j = 1, p = 1; p < N; j <<= 1, m = p)
    {
        for (p = 0, i = N - j; i < N; i++)
            y[p++] = i;
        for (int k = 0; k < N; ++k)
            if (SA[k] >= j)
                y[p++] = SA[k] - j;
        Sort();
        for (swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i)
            x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
    }
    for (int i = 1; i < N; ++i)
        SA[i - 1] = SA[i];
    --N;
}

void kasaiLCP()
{
    for (int i = 0; i < N; ++i)
        c[SA[i]] = i;
    LCP[0] = 0;
    for (int i = 0, h = 0; i < N; ++i)
        if (c[i] > 0)
        {
            int j = SA[c[i] - 1];
            while (i + h < N && j + h < N && str[i + h] == str[j+h])
                ++h;
            LCP[c[i]] = h;
            if (h > 0)
                --h;
        }
}

void suffixArray()
{
    m = 256;
    N = strlen(str);
    DA();
    kasaiLCP();
}
int main()
{

    int n;
    cin >> n;

    cin >> str;
    suffixArray();

    for (int i = 0; i < n; i++)
    {
        cout << SA[i] << endl;
    }
    cout << endl;

    for (int i = 1; i < n; i++)
    {
        cout << LCP[i] << endl;
    }

    return 0;
}
```

3.3 Z-array

```
    vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

Should be **tested**.

3.4 Suffix Array with LCP

Usage:
Time Complexity: `s`: string, `upper`: max `s`; e. g. 256 for ascii `string`. `sa`: pass suffix array together
(N+upper)for`SA`,O(N)for`lcp_array`$\mathcal{O}(N + upper)$ for `SA`, $\mathcal{O}(N)$ for `lcp_array`@i(N+upper)for`SA`,O(N)for`lcp_array`$\mathcal{O}(N + upper)$ for `SA`, $\mathcal{O}(N)$ for `lcp_array`@iO(N+upper)for`SA`,O(N)for`lcp_array`

3.5 Z-algorithm

Usage: i-th element is common prefix of S and $S_{i...|S|}$
Time Complexity: $\mathcal{O}(N)$

```
vector<int> Z(const vector<int>& S) {
    int N = S.size();
    vector<int> Z(N);
    int L = 0, R = 0;
    for(int i = 1; i < N; i++) {
        if(i+Z[i-L] < R) Z[i] = Z[i-L];
        else {
            L = i, R = max(R, i);
            while(R<N && S[R] == S[R-i]) ++R;
            Z[i] = R-i;
        }
    }
    return Z;
}
```

3.6 Manacher

Usage: Relation of palindromic sub-string vector, `p` with the actual 0 indexed string, `st`:

- `p[i*2+1]` is the length of the even palindromic substring where the center is between `st[i]` and `st[i-1]`
- `p[i*2+2]` is the length of the odd palindromic sub-string where center is `st[i]`

Time Complexity: $\mathcal{O}(N)$

```
void manacher(string &st)
{
    string s;
    for(auto &ch: st){
        s.PB('#');
        s.PB(ch);
    }
    s.PB('#');
    int n=s.size();
    s="$"+s+"^";
    std::vector<int> p(n+2);
    int l=1,r=1; //s[l+1]...s[r-1] is the rightmost palindrome
    for(int i=1;i<=n;i++){
        if(r>=i)    p[i]=min(r-i,p[l+(r-i)]);
        while(s[i-p[i]]==s[i+p[i]]){
            p[i]++;
        }
        if((i+p[i])>r){
            l=i-p[i],r=i+p[i];
        }
    }
    for(int i=1;i<=n;i++)   p[i]--;
}
```

Support Incremental Aho-corasick

3.7 Aho-Corasick

Usage: `MAXC`: size of alphabet, `F`, `FG`: failure (parent), failure graph, `ftrans`: state transition function.

```
template <int MAXC = 26> struct AhoCorasick {
  vector<array<int, MAXC>> C;
  vector<int> F;
  vector<vector<int>> FG;
  vector<bool> E;

  int node() {
    int r = C.size();
    E.push_back(0);
    F.push_back(-1);
    C.emplace_back();
    fill(C.back().begin(), C.back().end(), -1);
    return r;
  }
  int ctrans(int n, int c) {
    if (C[n][c] == -1) C[n][c] = node();
    return C[n][c];
  }
  int ftrans(int n, int c) const {
    while (n && C[n][c] == -1) n = F[n];
    return C[n][c] != -1 ? C[n][c] : 0;
  }
  AhoCorasick(vector<vector<int>> P) {
    node();
    for (int i = 0; i < (int)P.size(); i++) {
      int n = 0;
      for (int c : P[i]) n = ctrans(n, c);
      E[n] = 1;
    }
    queue<int> Q;
    F[0] = 0;
    for (int c : C[0]) if (c != -1) Q.push(c), F[c] = 0;
    while (!Q.empty()) {
      int n = Q.front(); Q.pop();
      for (int c = 0; c < MAXC; ++c) if (C[n][c] != -1) {
        int f = F[n];
        while (f && C[f][c] == -1) f = F[f];
        F[C[n][c]] = C[f][c] != -1 ? C[f][c] : 0;
        Q.emplace(C[n][c]);
      }
    }
    FG.resize(F.size());
    for (int i = 1; i < (int)F.size(); i++) {
      FG[F[i]].push_back(i);
      if (E[i]) Q.push(i);
    }
    while (!Q.empty()) {
      int n = Q.front();
      Q.pop();
      for (int f : FG[n]) E[f] = 1, Q.push(f);
    }
  }
  bool check(vector<int> V) {
    if (E[0]) return 1;
    int n = 0;
    for (int c : V) {
      n = ftrans(n, c);
      if (E[n]) return 1;
    }
    return 0;
  }
};
```

3.8 Suffix Automata

```
class SAutomata
{    private:
class SNode
{
public:
int link, len;
map<char,int> next;
SNode()
{
link = -1;
len = 0;
}
SNode(const SNode& other)
{
link = other.link;
len = other.len;
next = other.next;
}
};
vector nodes;
int last, cur;


    public:

SAutomata(string S)
: nodes(2*S.size()),
dp(2*S.size(), -1),
last(0),
cur(1)
{
for(int i=0; i<S.size(); i++)
add_char(S[i]);
}
void add_char(char A)
{
trace(A);
int nw = cur++;
nodes[nw].len = nodes[last].len+1;
int p=last;
for(; p!=-1 && nodes[p].next.find(A)==nodes[p].next.end(); p=nodes[p]
{
```

```
nodes[p].next[A]=nw;
}
if(p==-1)
{
nodes[nw].link = 0;
}
else if(nodes[nodes[p].next[A]].len == nodes[p].len + 1)
{
nodes[nw].link = nodes[p].next[A];
}
else
{
int nxt = nodes[p].next[A];
int clone = cur++;
nodes[clone] = nodes[nxt];
nodes[clone].len = nodes[p].len + 1;
nodes[nxt].link = clone;
nodes[nw].link = clone;
for(; p!=-1 && nodes[p].next.find(A)!=nodes[p].next.end() &&  nodes[p].next[A]==nxt; p=nodes[p].link)
{
nodes[p].next[A]=clone;
}
}
last = nw;
}};
```

4 Graph

4.1 Dijkstra's Algorithm - Single Source Shortest Path

Time Complexity: $(V+E)\log V$ :: will go throw geeksforgeeks letter

```
const int mx=1e5+1;
std::vector<pair<int,int> > adj1[mx],adj2[mx];
int d1[mx],d2[mx];
void dijkstra(int s,int n,vector<pair<int,int>>adj[mx],int
dist[mx]){
    for(int i=1;i<=n;i++)   dist[i]=9e18;
    dist[s]=0;
    priority_queue<pair<int,int>,vector<pair<int,int> >,
greater<pair<int,int>> >pq;
    pq.push({0,s});
    while(!pq.empty()){
        int u=pq.top().S;
        int curD=pq.top().F;
        pq.pop();
        if(dist[u]<curD)    continue;
        for(auto &x: adj[u]){
            int v=x.F,w=x.S;
            if((curD+w)<dist[v]){
                dist[v]=curD+w;
                pq.push({dist[v],v});
            }
        }
    }
}
```

4.2 Bellman–Ford - Detect Negative Cycle

```
ll n,m;
ll dis[mx];
struct e{
    ll u,v,w;
};
vector<e>edge;
bool Bellman_Ford(ll source){
    for(ll i=1;i<=n;i++) dis[i]=infLL;
    dis[source]=0;
    bool is_cycle=false;
    for(ll i=1;i<=n;i++){
        is_cycle=false;
        for(ll j=0;j<edge.size();j++){
            ll u=edge[j].u;
            ll v=edge[j].v;
            ll w=edge[j].w;
            if( dis[u]<infLL and dis[u]+w < dis[v] ){
                dis[v]=max( dis[u]+w,-infLL );
                is_cycle=true;
            }
        }
    }
    return is_cycle;
}
void _case_(){
    edge.clear();
    cin >> n >> m ;
    while(m--){
        ll u,v,w;
        cin >> u >> v >> w ;
        edge.pb({u,v,w});
    }
    if(Bellman_Ford(1)) cout << "negative cycle\n";
    else cout << "no negative cycle\n";
}
```

4.3 DSU

4.4 Small to large merging

```
   void dfs_size(int v, int p)
{
    sz[v] = 1;
    for (auto u : adj[v])
    {
        if (u != p)
        {
            dfs_size(u, v);
            sz[v] += sz[u];
        }
    }
}
void dfs(int v, int p, bool keep)
{
    int Max = -1, bigchild = -1;
    for (auto u : adj[v])
    {
        if (u != p && Max < sz[u])
        {
            Max = sz[u];
            bigchild = u;
        }
    }
    for (auto u : adj[v])
    {
        if (u != p && u != bigchild)
        {
            dfs(u, v, 0);
        }
    }
    if (bigchild != -1)
    {
        dfs(bigchild, v, 1);
        swap(vec[v], vec[bigchild]);
    }
    vec[v].push_back(v);
    cnt[color[v]]++;
    for (auto u : adj[v])
    {
        if (u != p && u != bigchild)
        {
            for (auto x : vec[u])
            {
                cnt[color[x]]++;
                vec[v].push_back(x);
            }
        }
    }
    // there are cnt[c] vertex in subtree v color with c
    if (keep == 0)
    {
        for (auto u : vec[v])
        {
            cnt[color[u]]--;
        }
    }
}
```

4.4.1 dsu

```
ll parent[mx];
ll volume[mx];
ll find_parent(ll v){
    if (v == parent[v])
        return v;
    return parent[v] = find_parent(parent[v]);
}
void make_union(ll u, ll v){
    u = find_parent(u);
    v = find_parent(v);
    if (u != v){
        if (volume[u] < volume[v]){
            swap(u, v);
        }
        parent[v] = u;
        volume[u] += volume[v];
    }
}
void solution(){
    for(int i=0;i<mx;i++){
        parent[i] = i ;
        volume[i] = 1 ;
    }
    ll n , q ;
    cin >> n >> q ;
    for(int i=0;i<n;i++){
        ll u , v ;
        cin >> u >> v ;
        make_union(u,v) ;
    }
    for(int i=0;i<q;i++){
        ll u , v ;
        cin >> u >> v ;
        if(find_parent(u)!=find_parent(v)){
            cout << "Disconnected" el;
        }
        else cout << "Connected" el ;
    }
}
```

4.5 Kruskal's algorithm for MST

```
struct node{
    ll u , v , w ;
    node(ll first , ll second , ll weight){
        u = first ;
        v = second ;
        w = weight ;
    }
};
ll volume[mx] , parent[mx] ;
ll find_parent(ll v){
    if(v==parent[v]) return v ;
    return parent[v] = find_parent(parent[v]) ;
}
void set_union(ll u , ll v){
    u = find_parent(u) ;
    v = find_parent(v) ;
    if(u!=v){
        if(volume[u]<volume[v]) {
            swap(u,v) ;
        }
        parent[v] = u ;
        volume[u] += volume[v] ;
    }
}
bool comp(node i, node j){
    return i.w < j.w ;
}
void solution()
{
    ll n , m , cost = 0 ;
    cin >> n >> m ;
    vector<node>edges ;
    vector<pair<ll,ll>> mst ;
    for(int i=0;i<m;i++)
    {
        ll u , v , w ;
        cin >> u >> v >> w ;
        edges.pb(node(u,v,w))  ;
    }
    for(int i=0;i<=n;i++) {
        volume[i] = 1 ;
        parent[i] = i ;
    }
    sort(edges.begin(),edges.end(),comp) ;
    for(auto i:edges)
    {
        if( find_parent(i.u) != find_parent(i.v) ){
            cost += i.w ;
            mst.pb({i.u,i.v}) ;
            set_union(i.u,i.v) ;
        }
    }
    cout << cost el;
    for(auto i:mst) {
        cout << i.first << " " << i.second el;
    }
}
```

4.6 Strongly Connected Components

```
vl adj[mx] , tadj[mx] ;
ll vis[mx] , cnt = 0 ;
stack<ll>stk ;
void dfs1(ll node) {
    vis[node] = 1 ;
    for(auto it:adj[node]) {
        if(!vis[it]) {
            dfs1(it) ;
        }
    }
    stk.push(node) ;
}
void dfs2(ll node) {
    cnt ++ ;
    vis[node] = 0 ;
    for(auto it:tadj[node]) {
        if(vis[it]) {
            dfs2(it) ;
        }
    }
}
void solution() {
    ll n , m ;
    cin >> n >> m ;
    stack<ll> emptyStack;
    swap(stk, emptyStack);
    for(ll i=0;i<=n;i++) {
        adj[i].clear() ;
        tadj[i].clear() ;
        vis[i] = 0 ;
    }
    for(ll i=0;i<m;i++) {
        ll u , v ;
        cin >> u >> v ;
        adj[u].push_back(v) ;
        tadj[v].push_back(u) ;
    }
    for(ll i=1;i<=n;i++) {
        if(!vis[i]) {
            dfs1(i) ;
        }
    }
    vector<ll> scc ;
    while(!stk.empty()) {
        ll topnode = stk.top() ;
        stk.pop() ;
        if(vis[topnode]) {
            cnt = 0 ;
            dfs2(topnode) ;
            scc.push_back(topnode) ;
        }
    }
}
```

4.7 Topological Sort

```
ll n , m ;
bool possible ;
vl topological ;
ll vis[mx] , dis[mx] ;
vector<pair<int,int>>adj[mx] ;
void dfs(int u){
    vis[u] = 1 ;
    for(auto i:adj[u])
    {
        int v = i.first ;
        int w = i.second ;
        if(!vis[v]){
            dfs(v) ;
        }
    }
    topological.pb(u) ;
}
void topological_sort(){
    topological.clear() ;
    for(int i=1;i<=n;i++){
        if(!vis[i]){
            dfs(i) ;
        }
    }
    reverse(topological.begin(),topological.end()) ;
    possible = (topological.size()==n) ;
}
void solution(){
    cin >> n >> m ;
    for(int i=1;i<=n;i++){
        adj[i].clear() ;
        vis[i]=0 ;
        dis[i] = infLL ;
    }
    for(int i=0;i<m;i++){
        int u , v , w ;
        cin >> u >> v >> w ;
        adj[u].pb(make_pair(v,w)) ;
    }
    possible = false ;
    topological_sort() ;
    if(possible) {
        for(auto i:topological){
            cout << i << " " ; cout el;
        }
    }
    else cout << "Not Possible" el;
}
```

4.8 Bridges In a graph

```
ll n , tin[mx] , low[mx] , vis[mx] , timer = 1 ;
vl adj[mx]  ;
vector<pair<ll,ll>>bridges ;
void dfs(ll node , ll parent ) {
    vis[node] = 1 ;
    tin[node] = low[node] = timer ;
    timer ++ ;
    for(auto it:adj[node]){
        if(it==parent) continue ;
        if(!vis[it]) {
            dfs(it,node) ;
            low[node] = min(low[node],low[it]) ;
            if(low[it] > tin[node]){
                bridges.push_back({it,node}) ;
            }
        }
        else {
            low[node] = min(low[node],low[it]) ;
        }
    }
}
void solution() {
    cin >> n ;
    for(int i=1;i<=n;i++) {
        vis[i] = 0 ;
        tin[i] = n+1 ;
        low[i] = n+1 ;
        adj[i].clear() ;
    }
    for(int i=0;i<n-1;i++) {
        ll u , v ;
        cin >> u >> v ;
        adj[u].push_back(v) ;
        adj[v].push_back(u) ;
    }
    dfs(1,-1) ;
    for(auto it:bridges) {
        cout << it.first << " " << it.second  el ;
    }
}
```

4.9 Bridge Finding in a Graph with Articulation Points

```
ll n , m , timer =0 ;
vl disc(mx,-1) , low(mx,-1) , par(mx,0) , vis(mx,0) ;
vi adj[mx] ;
set<int> articulation ;
vector<pair<int,int>> bridge ;
void dfs(int node,int parent){
    par[node]=parent ;
    disc[node] = low[node] = timer++ ;
    vis[node]=1 ;
    int child = 0 ;
    for(auto u:adj[node]){
        if(u==parent) continue ;
        if(!vis[u]){
            dfs(u,node) ;
            low[node] = min(low[node],low[u]) ;
            if(disc[node]<low[u]){
                int x = min(node,u) ;
                int y = max(node,u)  ;
                bridge.push_back({x,y}) ;
            }
            if(disc[node]<=low[u] and parent!=-1){
                articulation.insert(node) ;
            }
            child ++ ;
        }
        else{
            low[node]= min(low[node],disc[u]) ;
        }
    }
    if(parent==-1 and child>1){    // corner case
                articulation.insert(node) ;
    }
}
void solution(){
    cin >> n >> m ;
    for(int i=0;i<=n;i++){
        disc[i]=-1;
        adj[i].clear() ;
        low[i]=-1 ;
        vis[i]=0 ;
        par[i]=-1 ;
    }
    timer = 0 ;
    bridge.clear() ;
    for(int i=0;i<m;i++){
        int u , v ;
        cin >> u >> v ;
        adj[u].pb(v) ;
        adj[v].pb(u) ;
    }
    for(int i=0;i<=n;i++){
        if(!vis[i]) dfs(i,-1) ;
    }kmbng
    sort(bridge.begin(),bridge.end()) ;
    cout << articulation.size() << endl ;
    for(auto i:articulation) {
        cout << i << " " ; cout el;
```

```
    }
    cout << bridge.size() << endl ;
    for(auto i:bridge) {
        cout << i.first << " " << i.second << endl ;
    }
}
```

4.10 Maximum flow - Ford-Fulkerson

```
ll bfs(ll source, ll sink, vector<ll>& parent,
    vector<vector<ll>>& residualGraph) {
    fill(parent.begin(), parent.end(), -1);
    ll V = residualGraph.size();
    parent[source] = -2;
    queue<pair<ll, ll>> q;
    q.push({source, infLL});
    while (!q.empty()) {
        ll u = q.front().first ;
        ll capacity = q.front().second ;
        q.pop();
        for (ll av=1; av <= V; av++) {
            if (u != av && parent[av] == -1 &&
            residualGraph[u][av] != 0) {
                parent[av] = u;
                ll min_cap=min(capacity,residualGraph[u][av]);
                if (av == sink)
                    return min_cap;
                q.push({av, min_cap});
            }
        }
    }
    return 0;
}
ll ford_fulkerson(vector<vector<ll>>& graph,ll source,ll sink){
    vector<ll> parent(graph.size(), -1);
    vector<vector<ll>> residualGraph = graph;
    ll min_cap = 0, max_flow = 0;
    while (min_cap = bfs(source, sink,parent,residualGraph)){
        max_flow += min_cap;
        ll u = sink;
        while (u != source) {
            ll v = parent[u];
            residualGraph[u][v] += min_cap;
            residualGraph[v][u] -= min_cap;
            u = v;
        }
    }
    return max_flow;
}
void solution() {
    ll n , m ;
    cin >> n >> m ;
    vector<vector<ll>> graph(n+1, vector<ll> (n+1, 0));
    for(ll i=0;i<m;i++) {
        ll u , v , w ;
        cin >> u >> v >> w ;
        graph[u][v] += w ;
    }
    cout << ford_fulkerson(graph, 1, n) << endl;
}
```

4.11 Minimum-cost flow - Given K as Total Flow Quantity

```
struct Edge
{
    int from, to, capacity, cost;
};
vector<vector<int>> adj, cost, capacity;
const int INF = 1e9;
void shortest_paths(int n,int v0,vector<int>& d,vector<int>&p){
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if(capacity[u][v]>0 && d[v]>d[u]+cost[u][v]){
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}
int min_cost_flow(int N,vector<Edge> edges,int K,int s,int t){
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }
    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }
        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }
    if (flow < K)
        return -1;
    else
        return cost;
}
```

5 Number Theory

5.1 CRT

//solve linear congruences //x=a1(mod m1) //x=a2(mod m2) //x=a_n(mod m_n) //m1,m2,..m_n must be parwise co-prime //there exist a unique solution lesser than LCM(m1..m_n) mod Lcm(m1..m_n) //if congruences are not pariwise co-prime,reduce to pairwise co-prime

```
    struct Congruence {
    long long a, m;
};

long long chinese_remainder_theorem
(
vector<Congruence> const& congruences
)
{
    long long M = 1;
    for (auto const& congruence : congruences) {
        M *= congruence.m;
    }

    long long solution = 0;
    for (auto const& congruence : congruences) {
```

```
        long long a_i = congruence.a;
        long long M_i = M / congruence.m;
        long long N_i = mod_inv(M_i, congruence.m);
        solution = (solution + a_i * M_i % M * N_i) % M;
    }
    return solution;
}
```

5.2 Inclusion exclusion principle

//Task:Task: given two numbers n and r // count the number of integers in the interval [1,r] // that are relatively prime to n (their greatest common divisor is 1)

```
int solve (int n, int r) {
    vector<int> p;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            p.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        p.push_back (n);

    int sum = 0;
    for (int msk=1; msk<(1<<p.size()); ++msk) {
        int mult = 1,
            bits = 0;
        for (int i=0; i<(int)p.size(); ++i)
            if (msk & (1<<i)) {
                ++bits;
                mult *= p[i];
            }

        int cur = r / mult;
        if (bits % 2 == 1)
            sum += cur;
        else
            sum -= cur;
    }

    return r - sum;
}
```

5.3 Notes

1. If N = (p1^a1) * (p2^a2) * ... * (pN^aN)then NOD (N) = (a1+1) * (a2+1) * ... * (aN+1)
2. If N = (p1^a1) * (p2^a2) * ... * (pN^aN)then, SOD(N) = ((p1^(a1+1)) - 1) / (p1-1) *...* ((pN^(aN+1)) - 1) / (pN-1)
3. If M & N are co-prime then the formula holds :: [Phi (M) * Phi (N) = Phi (M*N)]
4. The Numbers(a) less than or Equal to N who all have [gcd(a,N) = d] will be Phi(N/d)
5. (sum of all the [Phi(d)) = N] ; where d represents all the divisors of N
6. For N>2 Phi(N) is always Even
7. Sum of all the Numbers less than or Equal to N that are Co Prime with N is [N * Phi(N) / 2]
8. [Lcm(1 , n) + Lcm(2 , n) + . . . + Lcm (1 , n) =(n / 2) * ((sum of all Phi(d) * d) + 1)]; d is the divisors of n
9. Mod Inverse can be solved recursively with the following formula :: inv[a] = (-Floor(Mod / A) * inv[Mod%A] + Mod) %Mod
10. logB(x) = (logC(x) / logC(B)) , [Here , logB(x) means log(x) based B] ;
11. What does log10(X) means? 10^(fractional part of the result) means the leading digit!!!
12. lgamma(x) = log(1)+...log(x-1) // this is like magic!!

5.4 Sum of NOD of numbers from 1 to N

```
///*** Complexity: O( sqrt(N) )***
long long SNOD(long long n=10){
    long long res=0;
    long long len=sqrtl(n);
    for(int i=1;i<=len;i++)res+=(n/i)-i; //finding the ordered
                                         //pair like a<b and a*b<n
    res*=2LL; //converting pair count to single value
    res+=len; //adding the equal values like (1,1),(2,2)..
    return res;
}
```

5.5 SOD And NOD

```
bitset<mx>vis ;
vector<ll> prime ;
void sieve()
{
    vis[1]=1;
    for(int i=3;i*i<mx;i+=2){
        if(!vis[i]){
            for(int j=i*i;j<mx;j+=2*i){
                vis[j] = 1 ;
            }
        }
    }
    prime.push_back(2) ;
    for(int i=3;i<mx;i+=2) {
        if(!vis[i]) prime.push_back(i) ;
    }
}
vector<pair<ll,ll>> prime_factorization(ll n)
{
    vector<pair<ll,ll>>factors;
    ll val = n ;
    for(auto i:prime){
        if(i*i>val) break ;
        if(n%i==0){
            int cnt = 0 ;
            while(n%i==0){
                cnt ++ ;
                n/=i ;
            }
            factors.push_back({i,cnt}) ;
        }
    }
    if(n!=1){
        factors.push_back({n,1}) ;
    }
    return factors ;
}
ll sumofDivisors(ll n){
    ll SOD = 1 ;
    vector<pair<ll,ll>> pf = prime_factorization(n) ;
    for(auto it:pf){
        ll up = bin_expo(it.first,it.second+1,infLL) -1 ;
        ll down = it.first - 1;
            SOD *= up/down ;
    }
    return SOD ;
}
ll numberofDivisors(ll n){
    ll NOD = 1 ;
    vector<pair<ll,ll>> pf = prime_factorization(n) ;
    for(auto it:pf){
        NOD *= it.second+1 ;
    }
    return NOD ;
}
```

5.6 ncr mod M where M is Prime

```
ll fact[mx] , inv[mx] ;
ll ncr_modmPrime(ll n, ll r, ll m){
    fact[0] = inv[0] = 1;
    for (ll i = 1; i < mx; i++){
        fact[i] = mod_mul(fact[i - 1], i, m);
    }
    inv[mx - 1] = bin_expo(fact[mx - 1], m - 2, m);
    for (ll i = mx - 2; i >= 1; i--){
        inv[i] = mod_mul(inv[i + 1], (i + 1), m);
    }
    ll up = fact[n];
    ll down = mod_mul(inv[r], inv[n - r], m);
    ll ans1 = mod_mul(up, down, m);
```

```
    return ans1;
}
```

5.7 Linear Diophantine Equation

```
int ext_gcd ( int A, int B, int *X, int *Y ){
    int x2, y2, x1, y1, x, y, r2, r1, q, r;
    x2 = 1; y2 = 0;
    x1 = 0; y1 = 1;
    for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 = x1,
    y2 = y1,x1 = x, y1 = y ) {
        q = r2 / r1;
        r = r2 % r1;
        x = x2 - (q * x1);
        y = y2 - (q * y1);
    }
    *X = x2; *Y = y2;
    return r2;
}
bool linearDiophantine ( int A, int B, int C, int *x, int *y ) {
    int g = gcd ( A, B );
    if ( C % g != 0 ) return false; //No Solution

    int a = A / g, b = B / g, c = C / g;
    ext_gcd( a, b, x, y ); //Solve ax + by = 1

    if ( g < 0 ) { //Make Sure gcd(a,b) = 1
        a *= -1; b *= -1; c *= -1;
    }

    *x *= c; *y *= c; //ax + by = c
    return true; //Solution Exists
}

int main () {
    int x, y, A = 2, B = 3, C = 5;
    bool res = linearDiophantine ( A, B, C, &x, &y );

    if ( res == false ) printf ( "No Solution\n" );
    else {
        printf ( "One Possible Solution (%d %d)\n", x, y );

        int g = gcd ( A, B );

        int k = 1; //Use different value of k to get different
        solutions
        printf ( "Another Possible Solution (%d %d)\n",
        x + k * ( B / g ), y - k * ( A / g ) );
    }

 return 0;
}
```

5.8 phi(m) CF power tower

```
#define int long long
const int mx=1e5+1;
int ar[mx];

map<ll,ll>mp;
ll phi(ll n)
{
    if(mp.count(n)) return mp[n];
    ll ans=n;
    ll m=n;
    for(ll i=2;i*i<=m;i++){
        if((m%i)==0){
            while((m%i)==0){
                m/=i;
            }
            ans=ans/i *(i-1);
        }
    }
    if(m>1) ans=ans/m *(m-1);
    return mp[n]=ans;
}
int expo_safe_mod(int x,int m)
{
    if(x<m) return x; //either take the full exponent : else if
    //only if x>=log2(m) you can go for phi(m)+(n%phi(m))
    else return m+x%m; //x>=log2(m) : the input m here is basically
    //phi(m) : unless x is zero
}

int bigMOD(int a,int n,int m)
{
    if(n==0)    return 1;
    else{
        int ret;
        if(n&1){
            ret=a*bigMOD(a,n-1,m);
            ret=expo_safe_mod(ret,m);
        }else{
            ret=bigMOD(a,n/2,m);
            ret*=ret;
            ret=expo_safe_mod(ret,m);
        }
        return ret;
    }
}

int solve(int l,int r,int m)
{
    if(m==1){
        if(ar[l]==0)    return 0;
        else return 1;
    }
    if(l==r)    return expo_safe_mod(ar[l],m);
    else{
        int pow=solve(l+1,r,phi(m));
        return bigMOD(ar[l],pow,m);
    }
}
signed main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)   cin>>ar[i];
    int q,l,r;
    cin>>q;
    while(q--){
        cin>>l>>r;
        cout<<solve(l,r,m)%m<<endl;
    }
    return 0;
}
```

5.9 divisor Property of Phi

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
              phi[j] -= phi[i];
}
```

5.10 Generating Highly Composite Numbers

Problem: Given an integer N, find the largest HCN which is smaller than or equal to N.

5.10.1 code (2×10^3) opeation

```
// prime[] is a list of prime.
int prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23 };

int resNum, resDiv, n;
void recur ( int pos, int limit, long long num, int div ) {
    if ( div > resDiv ) { // Get the number with highest NOD
        resNum = num;
        resDiv = div;
    }
    else if ( div == resDiv && num < resNum ) {
and  //In case of tie, take smaller number
        resNum = num;
```

```
    }

    if ( pos == 9 ) return; //End of prime list

    long long p = prime[pos];

    for ( int i = 1; i <= limit; i++ ) {
        if ( num * p > n ) break;
        recur ( pos + 1, i, num * p, div * ( i + 1 ) );
        p *= prime[pos];
    }
}
int main () {
    n = 1000000000;
    resNum = 0;
    resDiv = 0;
    recur ( 0, 30, 1, 1 );
    printf ( "%d %d\n", resNum, resDiv );
}
```

5.11 lucas

Given integers N, K, and a prime P, if we write N and K in base-P as:

$$N = n_0 + n_1 P + n_2 P^2 + \cdots + n_x P^x$$
$$K = k_0 + k_1 P + k_2 P^2 + \cdots + k_x P^x$$

where x is some non-negative integer and $0 \leq n_i, k_i < P$ for all $0 \leq i \leq x$, then

$$\binom{N}{K} \equiv \prod_{i=0}^{x} \binom{n_i}{k_i} \pmod{P}$$

```
ll power(ll a, ll b) { // a^b % mod
  ll ans = 1LL % mod;
  while (b) {
    if (b & 1) ans = mul(ans, a);
    a = mul(a, a);
    b >>= 1;
  }
  return ans;
}
```

5.12 Divisor Summatory Function

In number theory, the divisor summatory function is a function that is a sum over the divisor function.

Mathematically, Given an integer N, find the sum of the number of divisors from 1 to N. That is, find

$$\sum_{i=1}^{N} \mathrm{NOD}(i)$$

5.12.1 Code : $O(\sqrt{N})$ Using Divisor Pairs

1. Find the number of divisor pairs (A, B) where $A < B$.
2. Multiply the result with 2
3. Add the number of pairs (A,B) where A=B

```
int SNOD( int n ) {
    int res = 0;
    int u = sqrt(n);
    for ( int i = 1; i <= u; i++ ) {
        res += ( n / i ) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
    return res;
}
```

5.13 Notes

1. If N = (p1^a1) * (p2^a2) * ... * (pN^aN)then NOD (N) = (a1+1) * (a2+1) * ... * (aN+1)
2. If N = (p1^a1) * (p2^a2) * ... * (pN^aN)then, SOD(N) = ((p1^(a1+1)) - 1) / (p1-1) *...* ((pN^(aN+1)) - 1) / (pN-1)
3. If M & N are co-prime then the formula holds :: [Phi (M) * Phi (N) = Phi (M*N)]
4. The Numbers(a) less than or Equal to N who all have [gcd(a,N) = d] will be Phi(N/d)
5. (sum of all the [Phi(d)) = N] ; where d represents all the divisors of N
6. For N>2 Phi(N) is always Even
7. Sum of all the Numbers less than or Equal to N that are Co Prime with N is [N * Phi(N) / 2]
8. [Lcm(1 , n) + Lcm(2 , n) + . . . + Lcm (1 , n) =(n / 2) * ((sum of all Phi(d) * d) + 1)]; d is the divisors of n
9. Mod Inverse can be solved recursively with the following formula :: inv[a] = (-Floor(Mod / A) * inv[Mod%A] + Mod) %Mod
10. logB(x) = (logC(x) / logC(B)) , [Here , logB(x) means log(x) based B] ;
11. What does log10(X) means? 10^(fractional part of the result) means the leading digit!!!
12. lgamma(x) = log(1)+...log(x-1) // this is like magic!!

5.14 Linear Diophantine Equation

Usage: recursive formula for extended euclid Ω

$$g = a \cdot y_1 + b \cdot \left(x_1 - y_1 \cdot \left\lfloor \frac{a}{b} \right\rfloor \right)$$

Time Complexity: $\mathcal{O}(N)$

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0,
int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx, int
maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
```

```
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

5.15 calculate phi to 1 to n from sieve

In general, for not coprime a and b, the equation

$$\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}$$

For arbitrary x, m and $n \geq \log_2 m$:

$$x^n \equiv x^{\phi(m)+[n \bmod \phi(m)]} \mod m$$

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

6 DP

6.1 Edit Distance

```
int editDistanceUtil(string& S1, string& S2, int i, int j,
vector<vector<int>>& dp){
    if(i<0)
        return j+1;
    if(j<0)
        return i+1;
    if(dp[i][j]!=-1) return dp[i][j];
    if(S1[i]==S2[j])
      return dp[i][j] =  0+editDistanceUtil(S1,S2,i-1,j-1,dp);
    // Minimum of three choices
    else return dp[i][j] =
    1+min(editDistanceUtil(S1,S2,i-1,j-1,dp),
    min(editDistanceUtil(S1,S2,i-1,j,dp),
    editDistanceUtil(S1,S2,i,j-1,dp)));
}
```

6.2 Maximum Product Subarray in an Array O(N)

```
    int maxProductSubArray(vector<int>& nums) {
    int prod1 = nums[0],prod2 = nums[0],result = nums[0];
    for(int i=1;i<nums.size();i++) {
        int temp = max({nums[i],prod1*nums[i],prod2*nums[i]});
        prod2 = min({nums[i],prod1*nums[i],prod2*nums[i]});
        prod1 = temp;
        result = max(result,prod1);
    }
    return result;
}
```

6.3 Longest Common Subsequence

```
int lcs(string s1, string s2) {
    int n=s1.size();
    int m=s2.size() ;
    vector<int> prev(m+1,0), cur(m+1,0);
    // Base Case is covered as we have
    //initialized the prev and cur to 0.
    for(int ind1=1;ind1<=n;ind1++){
        for(int ind2=1;ind2<=m;ind2++){
            if(s1[ind1-1]==s2[ind2-1])
                cur[ind2] = 1 + prev[ind2-1];
            else
                cur[ind2] = 0 + max(prev[ind2],cur[ind2-1]);
        }
        prev= cur;
    }
    return prev[m];
}
```

6.4 Minimum Path Sum In a Grid

```
int minSumPath(int n, int m, vector<vector<int>> &matrix) {
    vector<int> prev(m, 0);
    for (int i = 0; i < n; i++) {
        vector<int> temp(m, 0);
        for (int j = 0; j < m; j++) {
            if (i == 0 && j == 0)
                temp[j] = matrix[i][j];
            else {
                int up = matrix[i][j];
                if (i > 0)
                    up += prev[j];
                else
                    up += 1e9;
                int left = matrix[i][j];
                if (j > 0)
                    left += temp[j - 1];
                else
                    left += 1e9;
                temp[j] = min(up, left);
            }
        }
        prev = temp;
    }
    return prev[m - 1];
}
```

6.5 LIS

```
/*
https://leetcode.com/problems/longest-increasing-subsequence/
*/
class Solution {
    static const int mx=2503;
    int n,ar[mx],dp[mx];
    int solve(){
        for(int i=0;i<n;i++){
            for(int j=1;j<=n;j++){
                if(ar[i]>dp[j-1] && ar[i]<dp[j]){
                    dp[j]=ar[i];
                    break;
                }
            }
        }
        int ans=1;
        for(int i=1;i<=n;i++){
            if(dp[i]!=1e9) ans=i;
            else break;
        }
        return ans;
    }
public:
    int lengthOfLIS(vector<int>& nums) {
        n=nums.size();
        for(int i=0;i<n;i++)    ar[i]=nums[i];
        for(int i=1;i<=n;i++)   dp[i]=1e9;
```

```
        dp[0]=-1e9;

        return solve();
    }
};
```

6.6 Divide and conquer DP

6.7 DP solution print

```
int n,W;
const int mx=1e2+1;
int w[mx];
int val[mx];
ll dp[mx][100001];
struct info{
    int i,c,ans;
    info(){}
    info(int i,int c,int ans) : i(i) ,c(c) ,ans(ans){}
}dir[mx][100001];

ll solve(int i,int rem)
{
    if(i>n) return 0;
    if(dp[i][rem]!=-1)  return dp[i][rem];
    ll ret=0,ret2;
    ret=solve(i+1,rem);
    dir[i][rem]=info(i+1,rem,0);
    if(rem>=w[i]){
        ret2=val[i]+solve(i+1,rem-w[i]);
        if(ret2>ret){
            ret=ret2;
            dir[i][rem]=info(i+1,rem-w[i],1);
        }
    }
    return  dp[i][rem]=ret;
}

void print(int i,int rem)
{
    if(i>n) return;
    if(dir[i][rem].ans) cout<<i<<" ";
    print(dir[i][rem].i,dir[i][rem].c);
}
int main()
{
    optimize();

    cin>>n>>W;
    for(int i=1;i<=n;i++)   cin>>w[i]>>val[i];
    memset(dp,-1,sizeof dp);
    cout<<solve(1,W)<<endl;

    print(1,W);
    cout<<endl;

    return 0;
}
```

6.8 BitmaskDP

- technic should be remembered when and why you should try bitmask dp state
- when there are too many cases and options
- you first probably thinking to hash those things anyway
- but you can;t track state and edit state in this way
- In most cases if elements are or options are 10-15 you should try bistmask dp

below a very standard problem is given and
one of my favorite problem also

here are 100 different types of caps each having a unique id from 1 to 100. Also, there are 'n' persons each having a collection of a variable number of caps. One day all of these persons decide to go in a party wearing a cap but to look unique they decided that none of them will wear the same type of cap. So, count the total number of arrangements or ways such that none of them is wearing the same type of cap.

Constraints: 1 <= n <= 10

```
#include <bits/stdc++.h>
#define MOD 1000000007
using namespace std;
// capList[i]'th vector contains the list of persons having
//a cap with id i
// id is between 1 to 100 so we declared an array of
//101 vectors as indexing
// starts from 0.
vector<int> capList[101];

// dp[2^10][101] .. in dp[i][j], i denotes the mask
//i.e., it tells that
// how many and which persons are wearing
//cap. j denotes the first j caps
// used. So, dp[i][j] tells the number
//ways we assign j caps to mask i
// such that none of them wears the same cap
int dp[1025][101];

// This is used for base case, it has all the N bits set
// so, it tells whether all N persons are wearing a cap.
int allmask;

// Mask is the set of persons, i is cap-id (OR the
// number of caps processed starting from first cap).
long long int countWaysUtil(int mask, int i)
{
    // If all persons are wearing a cap so we
    // are done and this is one way so return 1
    if (mask == allmask)
        return 1;

    // If not everyone is wearing a cap and also there are no more
    // caps left to process, so there is no way, thus return 0;
    if (i > 100)
        return 0;

    // If we already have solved this subproblem, return the answer.
    if (dp[mask][i] != -1)
        return dp[mask][i];

    // Ways, when we don't include this cap in our arrangement
    // or solution set.
    long long int ways = countWaysUtil(mask, i + 1);

    // size is the total number of persons having cap with id i.
    int size = capList[i].size();

    // So, assign one by one ith cap to all the possible persons
    // and recur for remaining caps.
    for (int j = 0; j < size; j++)
    {
        // if person capList[i][j] is already wearing a cap so contin
        // we cannot assign him this cap
        if (mask & (1 << capList[i][j]))
            continue;

        // Else assign him this cap and recur for remaining caps with
        // new updated mask vector
        else
            ways += countWaysUtil(mask | (1 << capList[i][j]), i + 1)
        ways %= MOD;
    }

    // Save the result and return it.
    return dp[mask][i] = ways;
}

// Reads n lines from standard input for current test case
void countWays(int n)
{
    //----------- READ INPUT -------------------------
    string temp, str;
    int x;
    getline(cin, str); // to get rid of newline character
    for (int i = 0; i < n; i++)
    {
        getline(cin, str);
        stringstream ss(str);

        // while there are words in the streamobject ss
        while (ss >> temp)
```

```
        {
            stringstream s;
            s << temp;
            s >> x;

            // add the ith person in the list of cap if with id
            capList[x].push_back(i);
        }
    }
    //-----------------------------------------------------

    // All mask is used to check whether all persons
    // are included or not, set all n bits as 1
    allmask = (1 << n) - 1;

    // Initialize all entries in dp as -1
    memset(dp, -1, sizeof dp);

    // Call recursive function count ways
    cout << countWaysUtil(0, 1) << endl;
}

int main()
{
    int n; // number of persons in every test case
    cin >> n;
    countWays(n);
    return 0;
}
```

6.9 Divide and Conquer DP

```
const int mx=2e5+1;
double t[mx];
double dp[2][mx],sum[mx],rev[mx];
double pre[mx]; //summation of result when a group is
1,2,3,.....n
double cost(int l,int r)
{
    double ans= pre[r]-pre[l-1]-(rev[r]-rev[l-1])*sum[l-1];
    return ans;
}
void d_and_con(int par,int l,int r,int optL,int optR)
{
    if(l>r) return;
    int mid=(l+r)>>1;
    dp[par&1][mid]=inf;
    int opt=optL;
    for(int j=optL;j<=min(mid,optR);j++){
        double res=dp[(par-1)&1][j-1]+cost(j,mid);

        if(res<dp[par&1][mid]){
            dp[par&1][mid]=res;
            opt=j;
        }
    }
    d_and_con(par,l,mid-1,optL,opt);
    d_and_con(par,mid+1,r,opt,optR);
}
signed main()
{
    int n,k;
    cin>>n>>k;
    for(int i=1;i<=n;i++)   cin>>t[i];
    for(int i=1;i<=n;i++){
        sum[i]=t[i];
        sum[i]+=sum[i-1];
        pre[i]=sum[i]/t[i];
        pre[i]+=pre[i-1];
        rev[i]=1.0/t[i];
        rev[i]+=rev[i-1];
    }
    for(int i=1;i<=n;i++)   dp[0][i]=inf;
    for(int par=1;par<=k;par++){
        d_and_con(par,1,n,1,n);
    }
    cout<<fixed<<setprecision(4)<< dp[k&1][n]<<endl;
    return 0;
}
```

6.10 Polar rho

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

using ll = long long;
namespace PollardRho {
  mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
  const int P = 1e6 + 9;
  ll seq[P];
  int primes[P], spf[P];
  inline ll add_mod(ll x, ll y, ll m) {
    return (x += y) < m ? x : x - m;
  }
  inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
    // ll res = x * y - (ll)((long double)x * y / m + 0.5) * m;
    // return res < 0 ? res + m : res;
  }
  inline ll pow_mod(ll x, ll n, ll m) {
    ll res = 1 % m;
    for (; n; n >>= 1) {
      if (n & 1) res = mul_mod(res, x, m);
      x = mul_mod(x, x, m);
    }
    return res;
  }
  // O(it * (logn)^3), it = number of rounds performed
  inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
    for (int i = 0; primes[i] < n && primes[i] < 32; i++) {
      c = pow_mod(primes[i], r, n);
      for (int j = 0; j < s; j++) {
        d = mul_mod(c, c, n);
        if (d == 1 && c != 1 && c != (n - 1)) return false;
        c = d;
      }
      if (c != 1) return false;
    }
    return true;
  }
  void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
      if (!spf[i]) primes[cnt++] = spf[i] = i;
      for (int j = 0, k; (k = i * primes[j]) < P; j++) {
        spf[k] = primes[j];
        if (spf[i] == spf[k]) break;
      }
    }
  }
  // returns O(n^(1/4))
  ll pollard_rho(ll n) {
    while (1) {
      ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v, t = 0;
      ll *px = seq, *py = seq;
      while (1) {
        *py++ = y = add_mod(mul_mod(y, y, n), c, n);
        *py++ = y = add_mod(mul_mod(y, y, n), c, n);
        if ((x = *px++) == y) break;
        v = u;
        u = mul_mod(u, abs(y - x), n);
        if (!u) return __gcd(v, n);
        if (++t == 32) {
          t = 0;
          if ((u = __gcd(u, n)) > 1 && u < n) return u;
        }
      }
      if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
    }
  }
  vector<ll> factorize(ll n) {
    if (n == 1) return vector <ll>();
    if (miller_rabin(n)) return vector<ll> {n};
    vector <ll> v, w;
```

```
    while (n > 1 && n < P) {
      v.push_back(spf[n]);
      n /= spf[n];
    }
    if (n >= P) {
      ll x = pollard_rho(n);
      v = factorize(x);
      w = factorize(n / x);
      v.insert(v.end(), w.begin(), w.end());
    }
    return v;
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  PollardRho::init();
  int t; cin >> t;
  while (t--) {
    ll n; cin >> n;
    auto f = PollardRho::factorize(n);
    sort(f.begin(), f.end());
    cout << f.size() << ' ';
    for (auto x: f) cout << x << ' '; cout << '\n';
  }
  return 0;
}
// https://judge.yosupo.jp/problem/factorize
```

6.11 dbg

Usage:

```
template < typename T >
ostream &operator << ( ostream & os, const vector< T > &v ) {
            os << "{";
                for(auto it = v.begin(); it != v.end(); ++it)
{
                                if( it != v.begin() ) os << ",
";
                                        os << *it;
                                            }
                    return os << "}";
}

template < typename T >
ostream &operator << ( ostream & os, const set< T > &v ) {
            os << "[";
                for(auto it = v.begin(); it != v.end(); ++it)
{
                                if( it != v.begin() ) os << ",
";
                                        os << *it;
                                            }
                    return os << "]";
}

template < typename T >
ostream &operator << ( ostream & os, const multiset< T > &v )
{
            os << "[";
                for(auto it = v.begin(); it != v.end(); ++it)
{
                                if( it != v.begin() ) os << ",
";
                                        os << *it;
                                            }
                    return os << "]";
}

template < typename F, typename S >
ostream &operator << ( ostream & os, const map< F, S > &v ) {
            os << "[";
                for(auto it = v.begin(); it != v.end(); ++it)
{
                                if( it != v.begin() ) os << ",
";
                                        os << it -> first << "
= " << it -> second ;
                                            }
                    return os << "]";
}

#define dbg(args...) do {cerr << #args << " : "; faltu(args);
} while(0)

void faltu () {
            cerr << endl;
}

template <typename T>
void faltu( T a[], int n ) {
            for(int i = 0; i < n; ++i) cerr << a[i] << ' ';
                cerr << endl;
}

template <typename T, typename ... hello>
void faltu( T arg, const hello &... rest) {
            cerr << arg << ' ';
                faltu(rest...);
}
```

6.12 Linear Diophantine equation

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g)
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

6.13 lis2

```
const int mx=1e5+1;
int x[mx],y[mx];

map<int,int >d[mx];

bool find(int x,int y,int idx) i
{

    auto it=d[idx].lower_bound(x);
    if(it==d[idx].begin()){
        return false;
    }
    it--;
    if(it->S<y) return true;
    else return false;
}

void add(int x,int y,int idx)
{
    auto it=d[idx].lower_bound(x);
    auto it2=it;
    {   //not adding conditon
        if(it2!=d[idx].begin()){
            it2--;
            if(it2->S<=y)   return;
        }
        if(d[idx].end()!=it){
```

```
            if(it->F==x && it->S<=y)    return;
        }
    }

    {   //deleting for maintaining the downward staircase
        it2=it;
        while(it2!=d[idx].end() && it2->S>=y)   it2++;
        d[idx].erase(it,it2);
        d[idx].insert({x,y});
    }

}

int chk(int l,int r,int x,int y)
{
    while(l<r){
        int mid=(l+r+1)/2;
        if(find(x,y,mid))   l=mid;
        else r=mid-1;
    }
    return l;
}

int lis(int n)
{
    int ans=0;
    for(int i=1;i<=n;i++){
        int retIdx=chk(0,i,x[i],y[i]);

        add(x[i],y[i],retIdx+1);

        ans=max(ans,retIdx+1);
    }
    return ans;
}

signed main()
{
    optimize();
    #ifndef ONLINE_JUDGE
    file();
    #endif

    int n;
    cin>>n;

    for(int i=1;i<=n;i++)   cin>>x[i]>>y[i];

    cout<<lis(n)<<endl;

    return 0;
}
```

6.14 Discrete log

```
    // Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

6.15 Centroid Decompostion

```
const int mx=1e5+1,LOG=22;
set<int>adj[mx];
int up[mx][LOG],depth[mx],subSz[mx],par[mx],ans[mx];

int dfs(int u,int p)
{
    subSz[u]=1;
    for(auto &v: adj[u]){
        if(v!=p){
            subSz[u]+=dfs(v,u);
        }
    }

    return subSz[u];
}


void genLCA(int node,int p)
{
    for(auto &v: adj[node]){
        if(v!=p){
            depth[v]=depth[node]+1;
            up[v][0]=node;
            for(auto j=1;j<LOG;j++){
                up[v][j]=up[up[v][j-1]][j-1];
            }

            genLCA(v,node);
        }
    }
}

int findLCA(int u,int v)
{
    if(depth[u] < depth[v])
    swap(u , v);

    if(depth[u]!=depth[v]){ ///get same depth
        int k=depth[u]-depth[v];
        for(int j=LOG-1;j>=0;j--){
            if(k&(1<<j)){
                u=up[u][j];
            }
        }
    }

    if(u==v){ //if v was the ancestor of a then a==b
        return u;
    }

    for(int j=LOG-1;j>=0;j--){
        if(up[u][j]!=up[v][j]){
            u=up[u][j];
            v=up[v][j];
        }



    }

    return up[u][0];
}

int find_centroid(int u,int p,int n)
{
    for(auto &v: adj[u]){
        if(v!=p && subSz[v]>n/2){
            return find_centroid(v,u,n);
        }
```

```
    }
    return u;
}

void decompose(int node,int p)
{
    int subTreeSz=dfs(node,p);
    int centroid=find_centroid(node,p,subTreeSz);


    par[centroid]=p;
    for(auto &v: adj[centroid]){
        adj[v].erase(centroid);
        //adj[centroid].erase(v);
        decompose(v,centroid);
    }
}

int get_distance(int a,int b)
{
    return depth[a] + depth[b] - 2*depth[findLCA(a , b)];
}

void update(int x){
    int k=x;
    //ans[k]=0;
    while(k!=-1){
        ans[k]=min(ans[k],get_distance(x,k));
        k=par[k];
    }
}

int query(int x){
    int k=x;
    int res=infLL;
    while(k!=-1){
        res=min(res,ans[k]+get_distance(k,x));
        k=par[k];
    }

    return res;
}

signed main()
{
    optimize();
    #ifndef ONLINE_JUDGE
    file();
    #endif

    int t,n,m,u,v;
    t=1;
    while(t--){
        cin>>n>>m;
        for(int i=1;i<n;i++){
            cin>>u>>v;
            adj[u].insert(v);
            adj[v].insert(u);
        }

        depth[1]=0;
        up[1][0]=1;
        for(int j=1;j<LOG;j++)  up[1][j]=1;
        genLCA(1,1);


        decompose(1,-1);

        for(int i=1;i<=n;i++)   ans[i]=inf;

        update(1);



        int type,x;
        while(m--){
            cin>>type>>x;
            if(type==1) update(x);
            else cout<<query(x)<<endl;
        }
    }

    return 0;
}
```

6.16 Sublime setup for c++ 14

```
{
 "cmd":["bash", "-c", "g++ -std=c++14 -Wall '${file}' -o
 '${file_path}/${file_base_name}' &&
 '${file_path}/${file_base_name}'"],
 "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
 "working_dir": "${file_path}",
 "selector": "source.c, source.c++",
 "variants":
 [
   {
     "name": "Run",
     "cmd":["bash", "-c", "g++ -std=c++14 '${file}' -o
     '${file_path}/${file_base_name}' &&
     '${file_path}/${file_base_name}'"]
   }
 ]
}
```

6.17 Code running command

remome file first

```
g++ -o hello hello.cpp
./hello
```

7 Geometry

7.1 Trigonometric Formulae

- Area of a triangle using coordinates: $A = \frac{1}{2}|x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)|$
- **Heron's formula** for finding area of triangle is is $\Delta = \sqrt{s(s-a)(s-b)(s-c)}$
- **Law of Sines:**

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

  where, R is the radius of the Circumcircle.
- **Law of Cosines:**

$$\cos A = \frac{b^2 + c^2 - a^2}{2bc}$$

- **Area of a Triangle using Sine:**

$$\Delta = \frac{1}{2}bc\sin A$$

- **Sine**: `sin(x)`
- **Cosine**: `cos(x)`
- **Tangent**: `tan(x)`
- **Arcsine (Inverse sine)**: `asin(x)`
- **Arccosine (Inverse cosine)**: `acos(x)`
- **Arctangent (Inverse tangent)**: `atan(x)`
- **Arctangent of two variables**: `atan2(y, x)`
  //tan(90)=undefined: then, so atan(undefined)=90 ; but, there is no way to give undefined, here you can can use atan2(sth,0); besides, it is preferable to use atan2(dy,dx) (Hasnain Hiakel vai)
- M_PI

7.2 Trianlge and various circles

- The **circumradius** R of a triangle is given by: (Note: **Circumcenter** is the point where the three perpendicular bisectors of its sides intersect)

$$R = \frac{a \cdot b \cdot c}{4 \cdot \Delta}$$

- The radius of the **incircle** of a triangle is given by:

$$r = \frac{\Delta}{s}$$

- The **center** of the **incircle** of a triangle is given by: (Note: it is the point where the three internal angle bisectors of a triangle intersect)

$$x = \frac{ax_1 + bx_2 + cx_3}{a+b+c}$$

$$y = \frac{ay_1 + by_2 + cy_3}{a+b+c}$$

where $A(x_1, y_1)$, $B(x_2, y_2)$, and $C(x_3, y_3)$ are the triangle's vertices.

- The radius of the **excircle** opposite to side a is given by:

$$r_a = \frac{\Delta}{s-a}, \quad r_b = \frac{\Delta}{s-b}, \quad r_c = \frac{\Delta}{s-c}$$

also,

$$r_a = \frac{incircle\_radius * (a+b+c)}{b+c-a}, ...$$

Here, r_a, r_b, r_c are the radii of the excircles opposite sides a, b, and c, respectively. And, Δ is the area of triangle.

- The **centers of the excircles** lie at the intersection of one internal and two external angle bisectors of the triangle.

8 Probability & Excpected value

- The binomial distribution models the number of successes in n independent Bernoulli trials, each with success probability p. Its probability mass function is:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0, 1, \ldots, n$$

$$(p+q)^n = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p.$$

- When we repeat something with probability, p to succeed, then the expected number of tries is $1/p$, till we succeed. (**geometric distribution**.)

9 Misc

9.1 stress testiing snippet

// Bash Script for Stress Testing: (checker.sh)

```
/*----------------------------------------------
for((i = 1; ; ++i)); do
    echo $i
    ./gen $i > in.txt
    diff -w <(./a < in.txt) <(./b < in.txt) || break
done
----------------------------------------------*/
// Random Integer Number Generator:
#define ll long long
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

inline ll gen_random(ll l, ll r) {
    return uniform_int_distribution<ll>(l, r)(rng);
}
// Random Real Number Generator:

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

inline double gen_random(double l, double r) {
    return uniform_real_distribution<double>(l, r)(rng);
}
```

Should be **revised**.
Working in progress.

9.2 (WIP) Magical Polynomial 3-SAT Algorithm

Usage: Use this to solve all problems!
Time Complexity: $\mathcal{O}(n)$

9.3 Policy-based Data structure

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
             tree_order_statistics_node_update>
    ordered_set;
using namespace std;
template <typename T>
using ordered_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
///set in ascending order (the typical one)
template <typename T>
using ordered_set_of_pairs = tree<pair<T, size_t>,
null_type, less<pair<T, size_t>>, rb_tree_tag,
tree_order_statistics_node_update>; ///set of
pairs in ascending order (the typical one)
template <typename T>
using ordered_set_desc = tree<T, null_type, greater<T>,
rb_tree_tag, tree_order_statistics_node_update>;
///set in descending order
template <typename T>
using ordered_set_of_pairs_desc = tree<pair<T, size_t>,
null_type, greater<pair<T, size_t>>, rb_tree_tag,
tree_order_statistics_node_update>; ///set of
pairs in descending order
#define optimize()
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'
#define MP make_pair
#define F first
#define S second
#define PB push_back
ordered_set<int> st1;
ordered_set_of_pairs<int> st2;
ordered_set_desc<int> st3;
ordered_set_of_pairs_desc<int> st4;
int main() {
optimize();
for( int i = 0; i < 10; ++i ) st1.insert(i);
cout << st1.order_of_key(2) << endl; ///how many
elements in st1 less than 2? (output: 2)
cout << *st1.find_by_order(5) << endl << endl;
///what is the 5th minimum element in st1?(0th
based indexing) (output: 5)
for( int i = 0; i < 10; ++i ) st2.insert(MP(i, i));
cout << st2.order_of_key(MP(2, 3)) << endl;
///output: 3
cout << st2.order_of_key(MP(2, 2)) << endl;
///output: 2
cout << st2.order_of_key(MP(3, 2)) << endl;
//output : 3
cout << st2.order_of_key(MP(3, -1)) << endl;
///output: 4 (i know, you were expecting 3. but
giving negative numbers as second element gives
unexpected results.)
cout << (*st2.find_by_order(5)).F << " " <<
(*st2.find_by_order(5)).S << endl << endl;
///output: 5 5
}
```

9.4 Fast I/O and Tiny Template

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
typedef long long ll;
#define PB push_back
#define F first
#define S second
#define endl '\n'
#define all(a) (a).begin(),(a).end()
#define sz(x) (int)x.size()
#define mx_int_prime 999999937
const double PI = acos(-1);
const double eps = 1e-9;
const int inf = 1e9+100;
const ll infLL = 4e18;
```

```
#define MOD 1000000007
#define optimize()
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define file()
freopen("input.txt","r",stdin);freopen("output.txt","w",stdout);

    optimize();
    #ifndef ONLINE_JUDGE
    file();
    #endif
```

9.5 2D Vector And Resize Syntex

```
vector<vector<int>> vec(3, vector<int>(4));
// First resize the 2D vector to 5 rows
vec.resize(5);
// Resize each row (or vector) to 6 columns
for (int i = 0; i < vec.size(); i++)
    vec[i].resize(6);//don't forget to resize vec[0]
```

9.6 Random Number Generator (Uniform Distribution)

Time Complexity: $O(1)$

```
//integer generator
mt19937
rng(chrono::steady_clock::now().time_since_epoch().count());
inline int gen_random(int l, int r)
{
   return uniform_int_distribution<int>(l, r) (rng);
}
// Long long Generator:
mt19937_64
rng(chrono::steady_clock::now().time_since_epoch().count());
inline ll gen_random(ll l, ll r) {
    return uniform_int_distribution<ll>(l, r)(rng);
}
// Random Real Number Generator:
mt19937_64
rng(chrono::steady_clock::now().time_since_epoch().count());
inline double gen_random(double l, double r) {
    return uniform_real_distribution<double>(l, r)(rng);
}

//array randomized shuffling O(n)
shuffle(v.begin(),v.end(), rng);
```