CODEBOOK RUET CSE 20

TEAM: NeverEndingHope

Template(Sefayet):

```
#include<bits/stdc++.h>

using namespace std;

#define ll              long long

#define scl(n)          scanf("%lld", &n)

#define fr(i,n)         for (ll i=0;i<n;i++)

#define fr1(i,n)        for(ll i=1;i<=n;i++)

#define pfl(x)          printf("%lld\n",x)

#define endl              "\n"

#define pb              push_back

#define asort(a)          sort(a,a+n)

#define dsort(a)          sort(a,a+n,greater<int>())

#define vasort(v)       sort(v.begin(), v.end());

#define vdsort(v)       sort(v.begin(),
v.end(),greater<ll>());

#define pn              printf("\n")

#define md              10000007

#define debug             printf("I am here\n")

#define l(s)               s.size()

#define tcas(i,t)         for(ll i=1;i<=t;i++)

#define pcas(i)            printf("Case %lld: ",i)

#define fast
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(
0);

Const ll maxN=1e17+10;

#define M 10000

void setIO(){

    #ifndef ONLINE_JUDGE

    freopen("input.txt", "r", stdin);

    freopen("output.txt", "w", stdout);

   #endif // ONLINE_JUDGE

}

int main()

{

    fast;

     ll t;

    //setIO();

     //ll tno=1;;

     //t=1;

    cin>>t;

    while(t--){

    }

    return 0;

}
```

Macro :

```
#include<ext/pb_ds/assoc_container.hpp>

#include<ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

#define nn '\n'

#define fo(i,n) for(i=0;i<n;i++)

#define deb(x) cout << #x << "=" << x << endl
```

```
#define deb2(x, y) cout << #x << "=" << x << ","
<< #y << "=" << y << endl

#define Setpre(n) cout<<fixed<<setprecision(n)

#define all(x) x.begin(), x.end()

#define rev(x) reverse(all(x))

#define sortall(x) sort(all(x))

#define mem(a,b) memset(a,b,sizeof(a))

#define fast_IO ios_base::sync_with_stdio(0),
cin.tie(0), cout.tie(0)

#define Set(x, k) (x |= (1LL << k))

#define Unset(x, k) (x &= ~(1LL << k))

#define Check(x, k) (x & (1LL << k))

#define Toggle(x, k) (x ^ (1LL << k))

typedef long long ll;

typedef unsigned long long ull;

typedef pair<ll, ll>    pll;

typedef vector<ll>      vl;

typedef vector<pll>     vpll;

typedef vector<vl>      vvl;

template <typename T> using PQ =
priority_queue<T>;

template <typename T> using QP =
priority_queue<T,vector<T>,greater<T>>;


template <typename T> using ordered_set = tree<T,
null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

template <typename T,typename R> using
ordered_map = tree<T, R , less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

inline ll Ceil(ll p, ll q)  {return p < 0 ? p / q
: p / q + !!(p % q);}
```

```
inline ll Floor(ll p, ll q) {return p > 0 ? p / q
: p / q - !!(p % q);}

inline double logb(ll base,ll num){ return
(double)log(num)/(double)log(base);}

int popcount(ll x){return
__builtin_popcountll(x);};

int poplow(ll x){return __builtin_ctzll(x);};

int pophigh(ll x){return 63 -
__builtin_clzll(x);};

const double EPS = 1e-9;

const int N = 2e5+10;

const int M = 1e9+7;
```

```
//RECURSION

///SUBSET Generation:

int a[6]={1,2,3,4,5};

int subset[6];

int n=5;

void gen_subset(int pos, int cnt){

  if(pos==n){

    // print values of subset[0...cnt]

    for(int i=0;i<cnt;i++) printf("%d ",subset[i]);

    printf("\n");

    return;  }

  gen_subset(pos+1, cnt);

  subset[cnt] = a[pos];

  gen_subset(pos+1, cnt+1);

}

int main(){

    gen_subset(0,0);
}
```

```
///PERMUTATION GEN
int used[20];
int number[20];
int a[] = {1, 2, 3, 4};
void permutation(int at, int n) {
    //Base Case
    if (at == n) {
        for (int i = 0 ; i < n ; i++) {
            printf("%d ", number[i]);
        }
        printf("\n");
        return;
    }
    for (int i = 0; i < n; i++) {
        if (!used[i]) {
            used[i] = 1;
            number[at] = a[i];
            //Recursive work
            permutation(at + 1, n);
            used[i] = 0;
        }
    }
}
int main() {
int n = sizeof(a) / sizeof(a[0]);
    permutation(0, n);
    return 0;
```

```
}
```

Subset Sum:

```
bitset<N> can;
cin>>n>>k;
can[0]=true;
fo(i,n){
    int x;cin>>x;
    can|=(can<<x);
}
cout<<(can[k]?"YES\n":"NO\n");
```

PREFIX SUM

2D prefix sum:

```
ll arr[N][N];
ll pfsum[N][N];
void buildPS(){
    for(int i=1;i<N;i++){
        for(int j=1;j<N;j++){
            pfsum[i][j]=arr[i][j]+pfsum[i-1][j]+pfsum[i][j-1]-pfsum[i-1][j-1];
        }
    }
}
ll getSum(ll a,ll b,ll c,ll d){
    return pfsum[c][d]-pfsum[a-1][d]-pfsum[c][b-1]+pfsum[a-1][b-1];
}
```

BASIC MATH

primesieve:

```
ll N=2e5+10;
vector<bool> Primes(N,1);
vector<ll>primenos;
void SieveOfEratosthenes(ll n)
{
 Primes[1]=0;
 for (ll i=2;i*i<=n;i++) {
 if(Primes[i]==1){
 for(ll j=i*i;j<=n;j+=i)
  Primes[j]=0;
  }
 }
 for(ll i=1;i<n;i++){
  if(Primes[i]){
   primenos.push_back(i);
  }
 }
}
void generatePrimeFactors(int N)
{
 int s[N+1];
 sieveOfEratosthenes(N, s);
 printf("Factor Power\n");
 int curr = s[N];
 int cnt = 1;
 while (N > 1)
 {
  N /= s[N];
  if (curr == s[N])
  {
   cnt++;
   continue;
  }
  printf("%d\t%d\n", curr, cnt);
  curr = s[N];
  cnt = 1;
 }
}
```

Prime Factor:

```
vector<bool> isPrime(N,1);
vl lp(N,0),hp(N,0);
void primeSieve(){
    isPrime[0]=isPrime[1]=false;
    for(int i=2;i<N;i++){
        if(isPrime[i]==true){
            lp[i]=hp[i]=i;
            for(int j=2*i;j<N;j+=i) {
                isPrime[j]=false;
                hp[j]=i;
                if(lp[j]==0){
                    lp[j]=i;
                }
            }
        }
    }
}

vl getPrimeFactor(ll n){
    vl prime_factors;
    while(n>1){
        ll prime_factor=hp[n];
        while(n%prime_factor==0){
            n/=prime_factor;

prime_factors.push_back(prime_factor);
        }
    }
    Return prime_factors;
}
```

Divisor Store:

```
vector<int> divisors[N];
void divisor_store(){
    for(int i=2;i<N;i++){
        for(int j=i;j<N;j+=i) {
            divisors[j].push_back(i);   }
    }
}
```

2…

```
bitset<500001> Primes;
void SieveOfEratosthenes(int n)
{
    Primes[0] = 1;
    for (int i = 3; i*i <= n; i += 2) {
 if (Primes[i / 2] == 0) {
for (int j = 3 * i; j <= n; j += 2 * i)
                Primes[j / 2] = 1;
        }
    }
}
```

POWER MOD:

```
ll power(ll a,ll b,ll mod)
{   int res = 1;
   a=a%mod;
   if (a==0) return 0;
   while (b>0)
   {
      if (b&1) res=(res*a)%mod;
      b /=2;
      a=(a*a)%mod;
   }
   return res;
}
```

Euler totient(n):

```
const int MAX = 100001;
bool isPrime[MAX+1];
// Stores prime numbers upto MAX - 1 values
vector<ll> p;
// Finds prime numbers upto MAX-1 and
// stores them in vector p
void sieve(){
    for (ll i = 2; i<= MAX; i++){
        // if prime[i] is not marked before
        if (isPrime[i] == 0){
            // fill vector for every newly
            // encountered prime
            p.push_back(i);
            // run this loop till square root of MAX,
            // mark the index i * j as not prime
            for (ll j = 2; i * j<= MAX; j++)
                isPrime[i * j]= 1;
        }
    }
}
// function to find totient of n
ll phi(ll n){
    ll res = n;
    // this loop runs sqrt(n / ln(n)) times
    for (ll i=0; p[i]*p[i] <= n; i++){



        if (n % p[i]== 0){
            // subtract multiples of p[i] from r
            res -= (res / p[i]);
            // Remove all occurrences of p[i] in n
            while (n % p[i]== 0) n /= p[i];
        }
    }
    // when n has prime factor greater
// than sqrt(n)
    if (n > 1) res -= (res / n);
    return res;
}
```

Euler totient(1-n):

```
// Computes and prints totient of all numbers
// smaller than or equal to n.
#define sz 10000000
ll prime[sz + 9], etf[sz + 9];
void computeTotient(){
    etf[1] = 1;
    for(ll i = 2; i <= sz; i++){
        if(!prime[i]){
            etf[i] = i - 1;
            for(ll j = 1; j * i <= sz; j++)
                if(!prime[j*i])prime[j*i] = i;
        }
        else{
            etf[i] = etf[prime[i]] * etf[ i/prime[i] ];
            ll g = 1;
            if(i % (prime[i]*prime[i]) == 0) g = prime[i];
            etf[i] *= g;
            etf[i] /= etf[g];
        }
    }
}
```

SORTING AND SEARCHING

Binary Search:

```
ll func(ll pos){
}
ll bs(ll low,ll high){
    ll mid;
    while(high-low>=2){
        mid=(high+low)>>1;
        if(func(mid)) low=mid;
        else high=mid-1;
    }
    if(func(high)) return high;
    return low;
}
```

Binary Search(real number):

```
double func(double mid){
}
double bs(double l,double r){
    double eps=1e-9;            //set the error limit here
    while(r-l>eps) {
        double mid=l+(r-l)/2;
        if (func(mid)) l=mid;
        else r=mid;
    }
    return l;
}
```

Ternary Search:

```
double func(double mid){
}
double ts(double l, double r){
    double eps=1e-9;               //set the error limit here
    while (r-l>eps){
        double mid1=l+(r-l)/3;
        double mid2=r-(r-l)/3;
        double f1=func(mid1);      //evaluates the function at mid1
        double f2=func(mid2);      //evaluates the function at mid2
        if (f1<f2) l = mid1;       //change f1>f2 if needed minimum
        else r=mid2;
    }
    return func(l);                //return the maximum of func(x) in [l, r]
}
```

count sort:

```
void countSort(vl &v){
    ll i=0,n=v.size(),mx=*max_element(all(v));
    vl cnt(mx+1,0);
    vl sorted(n);
    fo(i,n) cnt[v[i]]++;
    Fo(i,1,cnt.size()) cnt[i]+=cnt[i-1];
    Fo(i,n-1,-1) sorted[--cnt[v[i]]]=v[i];
    fo(i,v.size()) v[i]=sorted[i];
}
```

Inversion count(number of pair of index i,j where i<j && v[i]>v[j]):

```
// Returns inversion count in v[0..n-1]
ll getInvCount(vl &v,ll n){
    ordered_set<pll> st;
    ll invcount = 0;
    for(ll i = 0; i < n; i++) {
        ll temp=st.order_of_key({v[i], -1});
        temp=(ll)st.size()-temp;
        invcount+=temp;
        st.insert({v[i], i});
    }
    return invcount;
}
```

Kadane:

```
ll kadane(vl &vc,ll n){
    ll sum,currSum,i=0;
    sum=currSum=vc[0];
    Fo(i,1,n){
        currSum=max(currSum+vc[i],vc[i]);
        sum=max(sum,currSum);
        // currSum= (currSum<0? 0:currSum);
    }
    return sum;
}
```

Length of LIS(O(nlogn):

```
ll lis(vl &v){
    if (v.empty()) return 0;
    vl tail(v.size(), 0);
    int length = 1; // always points empty slot in tail
    tail[0] = v[0];
    for(int i=1;i<v.size();i++){
        auto b = tail.begin(), e = tail.begin() + length;
        auto it = lower_bound(b, e, v[i]);
        if (it == tail.begin() + length) tail[length++] = v[i];
        else *it = v[i];
    }
    return length;
}
```

Sparse Table:

```
int t[N][19], b[N], a[N];
void build(int n) {
    for(int i = 1; i <= n; ++i) t[i][0] = b[i];
    for(int k = 1; k < 19; ++k) {
        for(int i = 1; i + (1 << k) - 1 <= n; ++i) {
            t[i][k] = max(t[i][k - 1], t[i + (1 << (k - 1))][k - 1]);
        }
    }
}
int query(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return max(t[l][k], t[r - (1 << k) + 1][k]);
}
```

GRAPH & TREES

dfs:

```
bool vis[N];
void dfs(ll vertex){
    //take action on vertex after entering the
vertex
    vis[vertex]=true;
    for(ll child: g[vertex]){
        //take action on child before entering
the child node
        if(vis[child]) continue;
        dfs(child);
        //take action on child after entering the
child node
    }
    //take action on vertex before exiting the
vertex
}
```

bfs:

```
bool vis[N];
ll level[N];
void bfs(ll source){
    queue<ll> q;
    q.push(source);
    vis[source]=1;
    level[source]=0;
    while(!q.empty()){
        ll cur_v=q.front();
        q.pop();
        for(ll child:g[cur_v]){
            if(!vis[child]){
                q.push(child);
```

```
                vis[child]=1;
                level[child]=1+level[cur_v];
            }
        }
    }
}
```

***LEAF NODES IN A TREE

```
void dfs(list<int> t[], int node, int parent)
{
    int flag = 1;

    for (auto ir : t[node]) {
        if (ir != parent) {
            flag = 0;
            dfs(t, ir, node);
        }
    }

    if (flag == 1)
        cout << node << " ";
}
```

***finding the minimum way from n to 1 in a tree

```
vector<ll> ans;
void bfs(ll source){
    queue<ll> q;
    q.push(source);
    vis[source]=1;
    level[source]=0;
    while(!q.empty()){
        ll curr_v=q.front();
        q.pop();
        //cout<<curr_v<<" ";
        for(ll child: g[curr_v]){
            if(!vis[child]){
                q.push(child);
                vis[child]=1;
                if(level[child]>level[curr_v]+1){
```

```
                level[child]=level[curr_v]+1;
                par[child]=curr_v;
                }
            }
        }
    }
    //cout<<endl;
}
ll v=n;
  while(v!=1){

    ans.push_back(v);
     v=par[v];
  }
   ans.push_back(1);
   reverse(ans.begin(),ans.end());
   for(auto it:ans){
    cout<<it<<" ";
   }
   cout<<endl;
```

dijkstra:

```
vpll g[N];
vl dist(N,INT_MAX);
void dijkstra(int source){
    QP<pll> pq;
    pq.push(mp(0,source));
    dist[source]=0;
    while(pq.size()){
        ll v=pq.top().second;
        ll v_dist=pq.top().first;
        pq.pop();
        if(v_dist>dist[v]) continue;
        vis[v]=1;
        for(auto &child:g[v]){
            ll child_v=child.first;
            ll wt=child.second;
            if(dist[v]+wt<dist[child_v]){
                dist[child_v]=dist[v]+wt;

pq.push(mp(dist[child_v],child_v));
            }
        }
    }
}
```

Multisource bfs:

```
const ll maxN=1e3+10;//for graph
const ll INF=1e9+10;
#define M 10000


//when edges dont have same weight...0 and 1 weights..use 0-1 bfs
 ll n,m;
 ll val[maxN][maxN];
 ll vis[maxN][maxN];
 ll lev[maxN][maxN];
 void reset(){


    for(ll i=0;i<n;i++){
        for(ll j=0;j<m;j++){
            vis[i][j]=0;
            lev[i][j]=INF;
        }
```

```
   }


}
bool isvalid(ll i,ll j){
   return i>=0 &&  j>=0 && i< n && j<m;


}
 vector<pair<ll,ll> >movements={
{0,1},{0,-1},{1,0},{-1,0},
{1,1},{1,-1},{-1,1},{-1,-1}


};
ll bfs(){
    ll mx=0;
    for(ll i=0;i<n;i++){
       for(ll j=0;j<m;j++){
         mx=max(mx,val[i][j]);
       }
   }
   queue< pair<ll,ll> >q;


   for(ll i=0;i<n;i++){
       for(ll j=0;j<m;j++){
       if(mx==val[i][j]){
           q.push({i,j});
           lev[i][j]=0;
           vis[i][j]=1;
       }
       }
```

```
    }
    ll ans=0;
    while(!q.empty()){
        auto v=q.front();
        ll v_x=v.first;
        ll v_y=v.second;
        q.pop();
        for(auto movement : movements){
            ll child_x=movement.first+v_x;
            ll child_y=movement.second+v_y;
            if(!isvalid(child_x,child_y))
continue;
            if(vis[child_x][child_y]) continue;


            q.push({child_x,child_y});

lev[child_x][child_y]=lev[v_x][v_y]+1;
            vis[child_x][child_y]=1;
            ans=max(ans,lev[child_x][child_y]);
        }


    }
    return ans;
 }
int main()
{
    fast;
     ll t;
     //ll tno=1;;
     //t=1;
    cin>>t;
   while(t--){

    cin>>n>>m;
    reset();
    for(ll i=0;i<n;i++){
        for(ll j=0;j<m;j++){
            cin>>val[i][j];
        }
    }
   cout<< bfs()<<endl;
   }
    return 0;
}
```

dsu:

```
int par[N];
int sz[N];
// multiset<int> sizes;
void make(int v){
    par[v]=v;
    sz[v]=1;
    sizes.insert(1);
}
int find(int v){
    if(v==par[v]) return v;
    return par[v]=find(par[v]);
}
// void merge(int a,int b){
//     sizes.erase(sizes.find(sz[a]));
//     sizes.erase(sizes.find(sz[b]));
//     sizes.insert(sz[a]+sz[b]);
// }



void Union(int a,int b){
    a=find(a);
    b=find(b);
    if(a!=b){
        if(sz[a]<sz[b]) swap(a,b);
        par[b]=a;
        // merge(a,b);
        sz[a]+=sz[b];
    }
}
```

0-1 Bfs:

```
const ll maxN=1e5+10;//for graph
const ll INF=1e9+10;
#define M 10000



vector<pair<ll,ll> >g[maxN];
vector<ll> lev(maxN,INF);
//when edges dont have same weight...0 and 1 weights..use 0-1 bfs
 ll n,m;
ll bfs(){

    deque<ll> q;
    q.push_back(1);

    lev[1]=0;
    while(!q.empty()){
        ll curr_v=q.front();
        q.pop_front();
        for(auto &child : g[curr_v]){
            ll child_v=child.first;
            ll weight=child.second;
            if(lev[curr_v]+weight<lev[child_v]){
                lev[child_v]=lev[curr_v]+weight;
                if(weight==1){
                    q.push_back(child_v);
                }
                else{
                    q.push_front(child_v);
                }
            }
        }
    }
    return lev[n]==INF? -1:lev[n];
}
```

Floyd Warshal:

```
ll dp[N][N];
const int INF=1e9;
void floyd_warshall(int n){
    ll i,j,k;
    fo(i,n+1){
        dp[i][i]=0;
    }
    Fo(k,1,n+1){
        Fo(i,1,n+1){
            Fo(j,1,n+1){
                if(dp[i][k]!=INF && dp[k][j]!=INF){

                    dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j]);
                }
            }
        }
    }
}
```

Solving graph matrix using dfs

```
const ll maxN=1e5+10;//for graph
#define M 10000


class Solution {
public:
void dfs(int i,int j,int initialColor,int newColor,vector<vector<int>>& image){
    int n=image.size();
    int m=image[0].size();
    if(i<0 || j<0) return;
    if(i>=n || j>=m) return;
    if(image[i][j]!=initialColor) return;
    image[i][j]=newColor;
    dfs(i-1,j,initialColor,newColor,image);
    dfs(i+1,j,initialColor,newColor,image);
    dfs(i,j-1,initialColor,newColor,image);
    dfs(i,j+1,initialColor,newColor,image);

}
vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int color) {
        int initialColor=image[sr][sc];

        if(initialColor!=color) dfs(sr,sc,initialColor,color,image);
        return image;
    }
};
```

///Diameter of a tree

```
#define M 10000
const int N=1e5+10;
//maximum possible path between two nodes of a tree=diameter
vector<ll> g[N];
ll depth[N];
void dfs(ll v,ll par=-1){

    for(ll child : g[v]){
        if(child==par) continue;
        depth[child]=depth[v]+1;
        dfs(child,v);
    }
}
int main()
{
    fast;
     ll t;
     //ll tno=1;;
    t=1;
    //cin>>t;
    while(t--){
     ll n;
     cin>>n;
     ll x,y;
    for(ll i=0;i<n-1;i++){
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    dfs(1);

    ll mx_depth=-1;
    ll mx_d_node;
    for(ll i=1;i<=n;i++){
       if(mx_depth<depth[i]){
        mx_depth=depth[i];
        mx_d_node=i;
       }
       depth[i]=0;
    }
    dfs(mx_d_node);
    //consider mx_d_node is the central node and find the maximum depth....u will find the diameter

    mx_depth=-1;
    for(ll i=1;i<=n;i++){
       if(mx_depth<depth[i]){
        mx_depth=depth[i];

       }

    }
    cout<<mx_depth<<endl;
   }
    return 0;
}
```

Segment Tree:

```
ll tre[3*N];
ll merge(ll x,ll y){
    return x+y;//change according to problem
}
```

```
void buildSegTree(vector<ll>& arr, ll treeIndex, ll lo, ll hi){
    if (lo == hi) {                 // leaf node. store value in node.
        tre[treeIndex] = arr[lo];
        return;
    }
    ll mid = lo + (hi - lo) / 2;    // recurse deeper for children.
    buildSegTree(arr, 2 * treeIndex + 1, lo, mid);
    buildSegTree(arr, 2 * treeIndex + 2, mid + 1, hi);
    // merge build results
    tre[treeIndex] = merge(tre[2 * treeIndex + 1], tre[2 * treeIndex + 2]);
}
// call this method as buildSegTree(arr, 0, 0, n-1);
// Here arr[] is input array and n is its size.



ll querySegTree(ll treeIndex, ll lo, ll hi, ll i, ll j){
    // query for arr[i..j]
    if (lo > j || hi < i)               // segment completely outside range
        return 0;                       // represents a null node
    if (i <= lo && j >= hi)             // segment completely inside range
        return tre[treeIndex];
    ll mid = lo + (hi - lo) / 2;       // partial overlap of current segment and queried range. Recurse deeper.
    if (i > mid)
        return querySegTree(2 * treeIndex + 2, mid + 1, hi, i, j);
    else if (j <= mid)
        return querySegTree(2 * treeIndex + 1, lo, mid, i, j);
    ll leftQuery = querySegTree(2 * treeIndex + 1, lo, mid, i, mid);
    ll rightQuery = querySegTree(2 * treeIndex + 2, mid + 1, hi, mid + 1, j);
    // merge query results
    return merge(leftQuery, rightQuery);
}
// call this method as querySegTree(0, 0, n-1, i, j);
// Here [i,j] is the range/interval you are querying.
// This method relies on "null" nodes being equivalent to storing zero.
void updateValSegTree(ll treeIndex, ll lo, ll hi, ll arrIndex, ll val)
{
    if (lo == hi) {                    // leaf node. update element.
        tre[treeIndex] = val;
        return;
    }
    ll mid = lo + (hi - lo) / 2;    // recurse deeper for appropriate child
    if (arrIndex > mid)
        updateValSegTree(2 * treeIndex + 2, mid + 1, hi, arrIndex, val);
else if (arrIndex <= mid)
        updateValSegTree(2 * treeIndex + 1, lo, mid, arrIndex, val);
  // merge updates
    tre[treeIndex] = merge(tre[2 * treeIndex + 1], tre[2 * treeIndex + 2]);
}
// call this method as updateValSegTree(0, 0, n-1, i, val);
// Here you want to update the value at index i with value val.
```

Lazy Propagation:

```
void updateLazySegTree(int treeIndex, int lo, int hi, int i, int j, int val){
    if (lazy[treeIndex] != 0) {                // this node is lazy
        tre[treeIndex] += (hi - lo + 1) * lazy[treeIndex]; // normalize current node by removing laziness
        if (lo != hi) {                        // update lazy[] for children nodes
            lazy[2 * treeIndex + 1] += lazy[treeIndex];
            lazy[2 * treeIndex + 2] += lazy[treeIndex];
        }
        lazy[treeIndex] = 0;                   // current node processed. No longer lazy
    }
    if (lo > hi || lo > j || hi < i)
        return;
// out of range. escape.
    if (i <= lo && hi <= j) {
// segment is fully within update range
        tre[treeIndex] += (hi - lo + 1) * val;
// update segment
        if (lo != hi) {
// update lazy[] for children
            lazy[2 * treeIndex + 1] += val;
            lazy[2 * treeIndex + 2] += val;
        }
        return;
    }
    int mid = lo + (hi - lo) / 2;
// recurse deeper for appropriate child
```

```
 updateLazySegTree(2 * treeIndex + 1, lo, mid, i,
j, val);
    updateLazySegTree(2 * treeIndex + 2, mid + 1,
hi, i, j, val);
    // merge updates
    tre[treeIndex] = tre[2 * treeIndex + 1] +
tre[2 * treeIndex + 2];
}
// call this method as updateLazySegTree(0, 0, n-
1, i, j, val);
// Here you want to update the range [i, j] with
value val.
int queryLazySegTree(int treeIndex, int lo, int
hi, int i, int j){
```

```
    // query for arr[i..j]
    if (lo > j || hi < i)
// segment completely outside range
        return 0;
// represents a null node
    if (lazy[treeIndex] != 0) {
// this node is lazy
        tre[treeIndex] += (hi - lo + 1) *
lazy[treeIndex]; // normalize current node by
removing laziness
        if (lo != hi) {
// update lazy[] for children nodes
            lazy[2 * treeIndex + 1] +=
lazy[treeIndex];
            lazy[2 * treeIndex + 2] +=
lazy[treeIndex];
        }
        lazy[treeIndex] = 0;
// current node processed. No longer lazy
    }
 if (i <= lo && j >= hi)
// segment completely inside range
        return tre[treeIndex];
    int mid = lo + (hi - lo) / 2;
// partial overlap of current segment and queried
range. Recurse deeper.
    if (i > mid)
        return queryLazySegTree(2 * treeIndex +
2, mid + 1, hi, i, j);
    else if (j <= mid)
        return queryLazySegTree(2 * treeIndex +
1, lo, mid, i, j);
    int leftQuery = queryLazySegTree(2 *
treeIndex + 1, lo, mid, i, mid);
```

```
int rightQuery = queryLazySegTree(2 * treeIndex + 2, mid + 1, hi, mid + 1, j);



// merge query results
    return leftQuery + rightQuery;
}
// call this method as queryLazySegTree(0, 0, n-1, i, j);
// Here [i,j] is the range/interval you are querying.
// This method relies on "null" nodes being equivalent to storing zero.
```

BIT:

```
ll BITree[100009];
///do this for range: getSum(r) - getSum(l - 1)
ll getSum(ll index){
    ll sum = 0; // Iniialize result
    // Traverse ancestors of BITree[index]
    while (index>0){
        sum += BITree[index];  // Add current element of BITree to sum
        index -= index & (-index);  // Move index to parent node in getSum View
    }
    return sum;
}
void updateBIT(ll n, ll index, ll val){
    // Traverse all ancestors and add 'val'
    while (index <= n){
        // Add 'val' to current node of BI Tree
        BITree[index] += val;
        // Update index to that of parent in update View
        index += index & (-index);
    }
}
```

BITWISE & BINARY OPERATIONS

Binary Exponentiation:

```
int binexp(int a, int b){
    int result=1;
    while(b>0){
        if(b&1){
            result=(result * 1LL * a) % M;
        }
        a = (a * 1LL *a) % M;
        b>>=1;
    }
    return result;
}
```

Binary Multiply:

```
ll binMultiply(ll a,ll b){
    ll ans=0;
    while(b>0){
        if(b&1) ans=(ans+a)%M;
        a=(a+a)%M;
        b>>=1;
    }
    return ans;
}
```

HASHING

Custom hash for unordered map:

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

Hashing:

```
#define MAXLEN 1000010
constexpr uint64_t mod = (1ULL << 61) - 1;
const uint64_t seed = chrono::system_clock::now().time_since_epoch().count();
const uint64_t base = mt19937_64(seed)() % (mod / 3) + (mod / 3);
uint64_t base_pow[MAXLEN];
int64_t modmul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a >> 32, l2 = (uint32_t)b, h2 = b >> 32;
    uint64_t l = l1 * l2, m = l1 * h2 + l2 * h1, h = h1 * h2;
    uint64_t ret = (l & mod) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >> 3) + 1;
    ret = (ret & mod) + (ret >> 61);
    ret = (ret & mod) + (ret >> 61);
    return ret - 1;
}
void init(){
    base_pow[0] = 1;
    for (int i = 1; i < MAXLEN; i++){
        base_pow[i] = modmul(base_pow[i - 1], base);
    }
}
struct PolyHash{
    /// Remove suff vector and usage if reverse hash is not required for more speed
    vector<int64_t> pref, suff;
    PolyHash() {}
    template <typename T>
    PolyHash(const vector<T>& ar){
        if (!base_pow[0]) init();

        int n = ar.size();

        assert(n < MAXLEN);

        pref.resize(n + 3, 0), suff.resize(n + 3, 0);

        for (int i = 1; i <= n; i++){

            pref[i] = modmul(pref[i - 1], base) + ar[i - 1] + 997;

            if (pref[i] >= mod) pref[i] -= mod;

        }



        for (int i = n; i >= 1; i--){

            suff[i] = modmul(suff[i + 1], base) + ar[i - 1] + 997;

            if (suff[i] >= mod) suff[i] -= mod;

        }

    }

    PolyHash(const char* str)

        : PolyHash(vector<char> (str, str + strlen(str))) {}

    uint64_t get_hash(int l, int r){

        int64_t h = pref[r + 1] - modmul(base_pow[r - l + 1], pref[l]);

        return h < 0 ? h + mod : h;

    }

    uint64_t rev_hash(int l, int r){

        int64_t h = suff[l + 1] - modmul(base_pow[r - l + 1], suff[r + 2]);

        return h < 0 ? h + mod : h;

    }

};
```

DYNAMIC PROGRAMMING

///top to down:

```
 const ll maxN=1e5+10;

/// 0 1 1 2 3 5 8

ll dp[maxN];

//TOP DOWN approach

ll fib(ll n)

{

    if(n==0) return 0;

    if(n==1) return 1;

    if(dp[n]!=-1) return dp[n];

    //memoisation

    return dp[n]=fib(n-1)+fib(n-2);

}




int main()

{    fast;

    ll t;

    //t=1;

    cin>>t;

    while(t--)

    {

        memset(dp,-1,sizeof(dp));

        ll n;

        cin>>n;

        cout<<fib(n)<<endl;
    }


    return 0;
}
```

///knapsack

```
#define M 10000
const ll maxN=1e5+10;
ll wt[105],val[105];



ll dp[105][100005];
/*In knapsack problem, we are given n items we have to choose some items and to choose those there should be a condition and we must choose optimally
*/
/*bounded(general bag and stealing items prooblem or 0-1 knpsk) and unbounded knapsack(rod cutting)*/
/*fractional knapsack-similar to 0-1 knapsack but we can pick 0.1 part of an item)...not a prob of dp...rather a prob of greedy*/
///BOUNDED KNAPSACK
ll func(ll ind,ll wt_left)
{
    if(wt_left==0) return 0;
    if(ind<0) return 0;
    if(dp[ind][wt_left]!=-1) return dp[ind][wt_left];
    ll ans=func(ind-1,wt_left);
    if(wt_left-wt[ind] >=0) ans=max(ans,func(ind-1,wt_left-wt[ind])+val[ind]);
    return dp[ind][wt_left]=ans;
}
int main()
{
    fast;
    ll t;
    t=1;
    //cin>>t;
    while(t--)
    {
        memset(dp,-1,sizeof(dp));
        ll n,w;
        cin>>n>>w;
        for(ll i=0; i<n; i++)
        {
            cin>>wt[i]>>val[i];
        }
        cout<<func(n-1,w)<<endl;
    }
    return 0;
}
```

BOTTOM UP approach:

```
#define M 10000
const ll maxN=1e5+10;

/// 0 1 1 2 3 5 8
ll dp[maxN];
//Bottom Up approach
```

```
int main()
{
    fast;
     ll t;


     //t=1;
    cin>>t;
   while(t--){
    memset(dp,-1,sizeof(dp));
     ll n;
    cin>>n;
    //Bottom up approach
    dp[0]=0;
    dp[1]=1;
    for(ll i=2;i<=n;i++){
        dp[i]=dp[i-1]+dp[i-2];
    }

    cout<<dp[n]<<endl;

    }

    return 0;
}
ll NcR(ll n,ll r)
{
    long long p = 1, k = 1;

    if (n - r < r)
        r = n - r;
    if (r != 0) {
        while (r) {
            p *= n;
            k *= r;
            long long m = __gcd(p, k);
            p /= m;
            k /= m;
            n--;
            r--;
        }
    }
    else
        p = 1;
  return p;
}
ll factorial(ll n)
{
     if(n == 0)
          return 1;
    ll i = n, fact = 1;
    while (n / i != n) {
        fact = fact * i;
        i--;
    }
    return fact;
}
```

///Lexicographically compare of two strings

```
string compare(string s1,string s2){
    ll n=s1.size();
    ll a=0,b=0;
    for(ll i=0;i<n;i++){
        if(s1[i]=='1' && s2[i]=='0') return s1;
        if(s2[i]=='1' && s1[i]=='0') return s2;
    }
    return s2;
}
```

Custom comparator sort numeric strings

```
bool myCmp(string s1, string s2)
{
    if (s1.size() == s2.size()) {
        return s1 < s2;
    }
     else {
        return s1.size() < s2.size();
    }
}

// in main
//sort(v.begin(), v.end(), myCmp);
```

handling big numbers

```
// /* 128 bit integer reading */


// __int128 read()
// {
//     __int128 x = 0, f = 1;
//     char ch = getchar();
//     while (ch < '0' || ch > '9')
//     {
//         if (ch == '-')
//             f = -1;
//         ch = getchar();
//     }
//     while (ch >= '0' && ch <= '9')
//     {
//         x = x * 10 + ch - '0';
//         ch = getchar();
//     }
//     return x * f;
// }
// void print(__int128 x)
// {
//     if (x < 0)
//     {
//         putchar('-');
//         x = -x;
//     }
//     if (x > 9)
//         print(x / 10);
//     putchar(x % 10 + '0');
// }
// bool cmp(__int128 x, __int128 y) { return x > y; }


// /* a = read() for reading the integer and print(a) to print that integer.  */
```

Finding the maximum power of 2 in a number in ites prime factorization:

```
int highestPowerOf2(int n)
{
    return (n & (~(n - 1)));
}
//let the number be x;
 ll k=highestPowerOf2(x);
 powo2=log2(k);
///KMP
```

String matching:

Is string p present in string s?

```
 string s,p;
    ll n=p.size();
    ll m=s.size();
    vector<ll>lps(n);
    ll ln=0;
    bool f=0;
    for(ll i=1;i<n;){
      if(p[ln]==p[i]){
        lps[i]=ln+1;
        ln++;
//length of the longest suffix=prefix
        i++;
      }
      else{
        if(ln!=0){
//non matching character found
//length decreases to prev
//counted length
          ln=lps[ln-1];
        }
        else{
//prev counted length=0 so lps[i]=0
          lps[i]=0;
          i++;
        }
      }
    }
    // for(auto it:lps){
    //    cout<<it<<" ";
// }
cout<<endl;
ll l=0,r=0;

//vector<pair<ll,ll>> ans;
vector<ll>ans;
for(l=0;l<m;){
  // cout<<l<<" "<<r<<" "<<cnt<<endl;

  if(s[l]==p[r]){
    l++;
    r++;
  }
  else{
    if(r!=0){r=lps[r-1];
   }
    else{
      l++;
    }
  }

  if(r==n){
    f=1;
    ans.push_back(l-n+1);
  }
}
if(f) {
 cout<<ans.size()<<endl;
 for(auto it:ans){
  cout<<it<<" ";
}
```

```
    cout<<endl;
    }
    else{
      cout<<"Not Found"<<endl;
    }
    cout<<endl;
```

*cnt is the number of matched chars in the range

Check if there exists a point that all ranges cover:

```
bool sortby(const pair<ll, ll>& a,
      const pair<ll, ll>& b)
{
  if (a.first != b.first)
    return a.first < b.first;
  return (a.second < b.second);
}

// Function that returns true if any k
// segments overlap at any point
bool kOverlap(vector<pair<ll, ll> > pairs, ll k)
{
  // Vector to store the starting point
  // and the ending point
  vector<pair<ll, ll> > vec;
  for (ll i = 0; i < pairs.size(); i++) {
    // Starting points are marked by -1
    // and ending points by +1
    vec.push_back({ pairs[i].first, -1 });
    vec.push_back({ pairs[i].second, +1 });
  }
  // Sort the vector by first element
  sort(vec.begin(), vec.end());
  // Stack to store the overlaps
  stack<pair<ll, ll> > st;
  for (int i = 0; i < vec.size(); i++) {
    // Get the current element
    pair<ll, ll> cur = vec[i];

    // If it is the starting point
    if (cur.second == -1) {
      // Push it in the stack
      st.push(cur);
    }
    // It is the ending point
    else {
      // Pop an element from stack
      st.pop();
    }
    // If more than k ranges overlap
    if (st.size() >= k) {
      return true;
    }
  }
  return false;
}
```