****TEMPLATE****

```
#include<bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
template<typename T>
using ordered_set=tree<T, null_type,less<T>,
rb_tree_tag,tree_order_statistics_node_update>;
using ll=long long;
using ld=long double;
#define fast ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define read freopen ("in.txt","r",stdin);
#define sortv(k)     sort(k.begin(),k.end())
#define sortg(k)     sort(k.begin(),k.end(),greater<int>())
#define rev(k)       reverse(k.begin(),k.end())
#define pfp(x,y)     cout<<fixed<<setprecision(y)<<x<<endl;
#define ff first
#define pb push_back
#define pi acos(-1.0)
//cin.get();
constexpr ll MOD=998244353;
const int limit=100005;


void run_case(){

    return;
}

int main(){
    fast; read;
    int tc=1;
    cin>>tc;
    while(tc--) run_case();
    return 0;
}
```

****STRING TEMPLATE****

```
void solution(int t){
    string s; getline(cin,s);
}
int main()
{
    int tc;
    cin>>tc;
    cin.get();
    for(int t=1;t<=tc;t++) {
        solution(t);
    }
}
```

***(KMP ALGO (FOR FINDING PATTERN EXIST OR NOT AND HOW MANY TIMES IT EXIST)

```
void kmp(string t,string p)
```

```
{
    string s=p+"?"+t;
    int n=s.size();
    vector<int> pi(n);
    for(int i=1;i<n;i++) {
        int j=pi[i-1];
        while(j>0 && s[i]!=s[j]) j = pi[j-1];

        if (s[i]==s[j]) j++;
        pi[i] = j;
    }
    int cnt=0;
    for(int i=p.size();i<n;i++) {
        if(pi[i]==p.size()) cnt++;
    }
    cout<<cnt<<endl;
}

kmp(t,p);  ///p=pattern we need to search and t from where we need to search


                    ****Diagonal TEMPLATE****

for(int gap=1;gap<=n;gap++) ///Diagonal traversal
{
    for(int r=1,c=gap;r<=n,c<=n;r++,c++) ///main and upper diagonal
    if(gap>1)
    {
        for(int r=gap,c=1;r<=n;r++,c++) ///lower daigonal
    }
}


                    ****MOD AND BINARY EXPO****

ll addmod(ll a,ll b){
    return Mod(Mod(a)+Mod(b));  ///((a%m)+(b%m))%m
}
ll Mod(ll x){
    return ((x%MOD + MOD)%MOD); ///we add mod because x can be positive or negative
}
ll mulmod(ll a,ll b){
    return Mod(Mod(a)*Mod(b)); ///((a%m)*(b%m))%m
}
ll Binary_expo(ll a,ll n){   ///a^n
      ll res=1;
      while(n){
            if(n&1)
            res=mulmod(res,a);
            n/=2;
            a=mulmod(a,a);
      }
      return res%MOD;
}

                    ****NUMBER THEORY****
         ///SEIVE

bool vis[limit];
```

```
vector<int>prime;
void seive()
{
    vis[0]=vis[1]=1;
    for(int i=4; i<limit; i+=2) vis[i] = 1;
    for(int i=3; i*i<limit; i+=2)
    {
        if(vis[i]) continue;
        for(int j=i*i; j<limit; j+=2*i) vis[j] = 1;
    }
    prime.pb(2);
    for(int j=3; j<limit; j+=2)
    if(vis[j]==0) prime.pb(j);
}

               ///segmented seive

vector<int>primes;  ///collecting all the primes uptill sqrt(right) segment

void sqrtprime(int n) ///generating all primes number sqrt(Right) segment
{
    vector<int> isprime(n+1,0);
    isprime[1]=1;

    for(int i=2;i*i<=n;i++)
        if(isprime[i]==0) {
            for(int j=i*i;j<=n;j+=i) isprime[j]=1;
        }
    for(int i=1;i<=n;i++)
        if(isprime[i]==0) primes.push_back(i);
}
void primegenerator(int L,int R) ///taking the left and the right segment
{
    if(L==1) L++;
    int index=R-L+1;  ///creating an array of index right-left+1 size
    vector<int>ans(index,0);
    for(auto p:primes)  {
        if(p*p<=R) {
            int i=(L/p)*p;  ///finding the first divisor of p primes

            if(i<L) i+=p;  ///if it is below left segment.

            for(;i<=R;i+=p) {
                if(i!=p) ans[i-L]=1;  ///updating all the primes by [number-left] as a index
            }
        }
    }
    for(int i=0;i<index;i++) if(ans[i]==0) cout<<L+i<<endl;

}

        ///Count the number of divisors of a number using seive  O(logn)

int countdivisor(int n)   ///first pre-load the prime array using seive
{
    int divisor=1;
    for(int i=0;prime[i]*prime[i]<=n;i++) {
        if(n%prime[i]==0) {
            int cnt=1;
            while(n%prime[i]==0)  { cnt++; n/=prime[i]; }
            divisor*=cnt;
        }
    }
    return divisor;
}
```

********************* Property 1 **

যদি P একটি প্রাইম সংখ্যা হয় তবে p*2*2*…*2 এবং p*3*3*3…*3 এর GCD অবশ্যই হবে p হবে।
এখানে আমরা ২ এর স্থানে অন্যকোন প্রাইম এবং ৩ এর স্থানে অন্যকোন প্রাইম বসাতে পারি।
gcd(p*x*x*x , p*y*y*y*y)==p   ;   where (x ≠ y) and { x and y must be prime }.

******************** Property 2 ***

Any array’s gcd contains the common divisors of all the array elements.

********************** ১ থেকে n পর্যন্ত সংখ্যাকে মিনিমাম কয়ভাগে ভাগ করা যাবে ,যেখানে প্রত্যেক সংখ্যা পেয়ারওয়াইজ কো-প্রাইম হবে? **********************

--->n/2 group.

********************** Represent n as a sum of K prime number(distinct or not) *******************************

For K=1   n must be prime.
For K=2   If n is Even by Goldbach Conjecture it is possible.

If n is odd then (N-2) must be prime otherwise not.
For K>=3…… prove always yes.

****GEOMETRY****

****************** Coplanar test ************************

Area=x1*(y2-y3) +x2*(y3-y1) +x3*(y1-y2)   If (Area==0) they are coplanar.

****DATA STRUCTURE SORTING****

```
set<char> vow = {'a','e','i','o','u'};
bool isvow(char c)
{
     return vow.find(c) != vow.end();
}

bool compair(pair<ll,ll>a,pair<ll,ll>b)
{
    if(a.ff==b.ff)return a.ss<b.ss;
    return a.ff>b.ff;
}

///EKTA PORTION KE SORT KORA

compair(pair<int,int>p,pair<int,int>q){

    if(p.ff=q.ff) return p.ss>q.ss;
    return p.ff<q.ff;
}

vector<pair<int,int>>a(n);

sort(a.begin()+x,a.begin()+y,compair);
```

****DATA STRUCTURE****

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
namespace __gnu_pbds{
          typedef tree<int,
                       null_type,
                       less_equal<int>,
                       rb_tree_tag,
                       tree_order_statistics_node_update> ordered_set;
}
using namespace __gnu_pbds;

void Insert(ordered_set &s,int x){ //this function inserts one more occurrence of
(x) into the set.
     s.insert(x);
}
bool Exist(ordered_set &s,int x){ //this function checks weather the value (x)
exists in the set or not.
     if((s.upper_bound(x))==s.end()){ return 0; }
     return ((*s.upper_bound(x))==x);
}
void Erase(ordered_set &s,int x){ //this function erases one occurrence of the
value (x).
     if(Exist(s,x)){ s.erase(s.upper_bound(x)); }
}
int FirstIdx(ordered_set &s,int x){ //this function returns the first index of the
value (x)..(0 indexing).
    if(!Exist(s,x)){ return -1; }
    return (s.order_of_key(x));
}
int Value(ordered_set &s,int idx){ //this function returns the value at the index
(idx)..(0 indexing).
   return (*s.find_by_order(idx));
}
int LastIdx(ordered_set &s,int x){ //this function returns the last index of the
value (x)..(0 indexing).
    if(!Exist(s,x)){ return -1; }
    if(Value(s,(int)s.size()-1)==x){ return (int)(s.size())-1;}
    return FirstIdx(s,*s.lower_bound(x))-1;
}
int Count(ordered_set &s,int x){ //this function returns the number of occurrences
of the value (x).

     if(!Exist(s,x)){return 0;}
     return LastIdx(s,x)-FirstIdx(s,x)+1;
}
void Clear(ordered_set &s){ //this function clears all the elements from the set.
     s.clear();
}
int Size(ordered_set &s){ //this function returns the size of the set.
     return (int)(s.size());
}
int main(){

    ordered_set s;
    return 0;
}


      ****DSU***

vector<int>Parent(limit,-1);
vector<int>Rank(limit,1); ///Rank means size of this cc

int Find(int a){
    if(Parent[a]<0) return a;  return Parent[a]=Find(Parent[a]);
}

void Union(int a,int b){
    a=Find(a); b=Find(b);
    if(a!=b){
        if(Rank[b]>Rank[a]) swap(a,b);
        Parent[b]=a;
        Rank[a]+=Rank[b];
    }
}
void solution(){
    int n,m;
    cin>>n>>m;
    while(m--){
        int a,b; cin>>a>>b; Union(a,b);
    }
}

      ***DFS***
vector<int>adj[10001];
bool vis[10001]={false};

void dfs(int p){
    vis[p]=true;
    for(int child:adj[node])
        if(!vis[child]) dfs(child);
}

   ///cycle detection

int col[10001];
bool dfs(int node,int par)
{
    vis[node]=true;
    for(int child:adj[node]){
        if(!vis[child]) if(dfs(child,node)==true) return true;
        else if(child!=par) return true; ///find a back edge
    }
    return false;
}

    ///DFS ON GRID

int adj[100005][100005];
```

```
bool vis[100005][100005]={false};

int dx[]={-1,0,1,0,-1,1,-1,1};
int dy[]={0,-1,0,1,-1,-1,1,1};
int timer;
bool isvalid(int x,int y,int n,int m){
    if(x<1||x>n||y<1||y>m||vis[x][y]==true) return false;
    return true;
}
void dfs(int x,int y,int n,int m){
    vis[x][y]=true;
    for(int i=0;i<4;i++){
        if(isvalid(x+dx[i],y+dy[i],n,m)){
            dfs(x+dx[i],y+dy[i],n,m);
        }
    }
}


dfs(1,1,n,m);
```

BFS

```
void bfs(int srt)
{
    queue<int>q;
    q.push(srt);
    vis[srt]=true;
    while(!q.empty())
    {
        int cur=q.front();
        q.pop();
        for(int child:adj[cur]){
            if(vis[child]==false){
                q.push(child);
                vis[child]=true;
            }
        }
    }
}
int main()
{
    for(int i=1;i<=e;i++){
        int x,y;
        cin>>x>>y;
        adj[x].push_back(y);
    }
    bfs(1,adj,vis);
}
```

***DFS OF GRID

```
bool isvalid(int x,int y,int n,int m)
{
    if(x<1||x>n||y<1||y>m||vis[x][y]==true) return false;
    return true;
}
void dfs(int x,int y,int n,int m){
    vis[x][y]=true;
    for(int i=0;i<4;i++){
        if(isvalid(x+dx[i],y+dy[i],n,m)){
            dfs(x+dx[i],y+dy[i],n,m);
        }
    }
}

dfs(1,1,n,m);
```

dijkstra

```
void solution(){

    int n,e;
    cin>>n>>e;
    vector<pair<int,int>>adj[n+1];
    vector<int>dis(n+1,INFINITY);

    for(int i=1;i<=e;i++){
        int x,y,w;
        cin>>x>>y>>w;
        adj[x].push_back({y,w});
        adj[y].push_back({x,w});
    }
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>pq;
    dis[1]=0;
    pq.push({0,1});
    while(!pq.empty()){
        int w=pq.top().first;
        int src=pq.top().second;
        pq.pop();
        for(pair<int,int>child:adj[src]){
            if(w+child.second<dis[child.first]) ///(dis of src + adjecent dis src
to child) < dis of child stored before
            {
                dis[child.first]=w+child.second;
                pq.push({dis[child.first],child.first});
            }
        }
    }
    for(int i=1;i<=n;i++) cout<<dis[i]<<" ";
    cout<<endl;
    return;
}
```

****TREE DIAMETER***

```
vector<int>adj[200005];
int d[200005];
int dd[200005];
int dia=0;

void diameter(int node,int parent){
    vector<int>child_dis;
    for(int child:adj[node]){
        if(child!=parent){
            diameter(child,node);
            d[node]=max(d[node],1+d[child]);
            child_dis.push_back(d[child]);
        }
    }
    dia=max(dia,d[node]);
    sort(child_dis.begin(),child_dis.end());
    if(child_dis.size()>1){
        int x=child_dis.size()-1;  ///largest distance 1
        int y=x-1;         ///largest distance 2
        dd[node]=2+child_dis[x]+child_dis[y];
    }

    dia=max(dia,dd[node]);
}
void solve(){
    cin>>n;
    for(int i=1;i<n;i++){
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    diameter(1,-1);

    cout<<dia<<endl;
    return;
}
```

SUBTREE OR SUBORDINATE

```
vector<int>adj[300005];
int subsize[300005];

void dfs(int node,int parent){
    subsize[node]=1;
    for(int child:adj[node]){
        if(child!=parent){
            dfs(child,node);
            subsize[node]+=subsize[child];
        }
    }
}
void solve(){
    cin>>n;
    for(int i=1;i<n;i++){
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    dfs(1,0);
    for(int i=1;i<=n;i++) cout<<subsize[i]<<" ";
}
```

DP LIS

```
for(int i=0;i<n;i++) cin>>a[i];
vector<int>lis;
lis.push_back(a[0]);
for(int i=1;i<n;i++)
{
      if(lis.back()<a[i]) lis.push_back(a[i]);
      else{
          int indx=lower_bound(lis.begin(),lis.end(),a[i])-lis.begin();
         lis[indx]=a[i];
      }
}
cout<<lis.size()<<endl;

      ***DP LPS***

int dp[1005][1005];
int solution(int B,int E)
{
    if(B>E) return 0;
    if(B==E) return 1;

    if(dp[B][E]!=-1) return dp[B][E];
    if(s[B]==s[E]) return dp[B][E]=2+solution(B+1,E-1);
    else{
        return dp[B][E]=max(solution(B,E-1),solution(B+1,E));
    }
}
solution(0,n-1);
```