CoU Enigmatologist’s Notebook

Comilla University

Mehedi Hasan Arafat

Zyan Tarek

Mestu Paul

Last Updated on: June 2, 2022

1. Contents

[1 Important formulas 3](#_Toc105089730)

[1.1 Prime 3](#_Toc105089731)

[1.2 Properties of phi: 3](#_Toc105089732)

[1.3 Properties of mod: 3](#_Toc105089733)

[1.4 Properties of Digitsum: 3](#_Toc105089734)

[1.5 Properties of FLOOR CEIL: 3](#_Toc105089735)

[1.6 Bit Manipulation Macros 3](#_Toc105089736)

[1.7 Important Series and properties 3](#_Toc105089737)

[2 Number Theory 5](#_Toc105089738)

[2.1 Sieve 5](#_Toc105089739)

[2.2 Sum of divisor 5](#_Toc105089740)

[2.3 Miller Robin 6](#_Toc105089741)

[2.4 Number of divisor 6](#_Toc105089742)

[2.5 Euler Phi 7](#_Toc105089743)

[2.6 Euler phi from 1 to MAX 7](#_Toc105089744)

[3 Data structure 8](#_Toc105089745)

[1. 8](#_Toc105089746)

[3.1 Segment tree 8](#_Toc105089748)

[3.2 Sqrt Decomposition 9](#_Toc105089749)

[3.3 DSU 10](#_Toc105089750)

[3.4 Trie 10](#_Toc105089751)

[3.5 Ordered Set 11](#_Toc105089752)

[3.6 Sparse Table 11](#_Toc105089753)

[4 Geometry 11](#_Toc105089754)

[4.1 Triangle 11](#_Toc105089759)

[4.2 Circle 12](#_Toc105089760)

[4.3 Others 12](#_Toc105089761)

[5 Graph 13](#_Toc105089762)

[5.1 Articulation point 13](#_Toc105089763)

[5.2 Dijkstra 13](#_Toc105089764)

[5.3 Bellmenford 14](#_Toc105089765)

[5.4 Floyed Warshal 14](#_Toc105089766)

[5.5 Ford Fulkerson 15](#_Toc105089767)

[5.6 Prim 16](#_Toc105089768)

[5.7 Kruskal 16](#_Toc105089769)

[5.8 LCA 17](#_Toc105089770)

[5.9 SCC 18](#_Toc105089771)

[5.10 Topsort 18](#_Toc105089772)

[6 String 19](#_Toc105089791)

[6.1 KMP 19](#_Toc105089798)

[6.2 Hashing 20](#_Toc105089799)

[6.3 Z function 21](#_Toc105089800)

[7 DP 21](#_Toc105089801)

[7.1 Meet in the middle 21](#_Toc105089809)

[7.2 Maximum Sum Matrix 22](#_Toc105089810)

# Important formulas

## Prime

* The number of prime numbers less than or equal to n is approximately
* The k-th prime number approximately equals

## Properties of phi:

1. ;where n>= log2m

## Properties of mod:

## Properties of Digitsum:

* 1. =1

## Properties of FLOOR CEIL:

## Bit Manipulation Macros

#define **least\_one\_pos**(x) \_\_builtin\_ffs(x)  
#define **leading\_zeros**(x) \_\_builtin\_clz(x)  
#define **tailing\_zeros**(x) \_\_builtin\_ctz(x)  
#define **num\_of\_one**(x) \_\_builtin\_popcount(x)  
#define **msb**(x) 32-**leading\_zeros**(x)

## Important Series and properties

1. Vandermonde

# Number Theory

## Sieve

vector<**long long**> prime;  
bitset<100000>mark;  
**inline void** sieve( **long long** n)  
{  
 mark[0]=mark[1]=1;  
 **long long** i,j,limit=sqrt(n\*1.0)+2;  
 prime.emplace\_back(2);  
 **for**(i=4; i<=n; i+=2)  
 mark[i]=1;  
 **for**(i=3; i<=n; i+=2)  
 {  
 **if**(!mark[i])  
 {  
 prime.emplace\_back(i);  
 **if**(i<=limit)  
 {  
 **for**(j=i\*i; j<=n; j+=i\*2)  
 mark[j]=1;  
 }  
 }  
 }  
}

## Sum of divisor

**long long** SumOfDivisor(**long long** n)  
{  
 **long long** ans=1;  
 **for**(**long long** i=0; prime[i]\*prime[i]<=n; i++)  
 {  
 **long long** sum=0,p=1;  
 **while**(n%prime[i]==0)  
 {  
 n/=prime[i];  
 p\*=prime[i];  
 sum+=p;  
 }  
 ans\*=(sum+1);  
 }  
 **if**(n>1)  
 ans\*=(n+1);  
 **return** ans;  
}

## Miller Robin

**long long** bigmod(**unsigned long long** n, **unsigned long long** p, **unsigned long long** m) {  
  
 **unsigned long long** x = 1;  
 n %= m;  
 **while** (p) {  
 **if** (p & 1)  
 x = (**\_\_uint128\_t**) x \* n % m;  
 n = (**\_\_uint128\_t**) n \* n % m;  
 p >>= 1;  
 }  
 **return** x;  
}  
**bool** isComposite(**unsigned long long** n, **int** p, **unsigned long long** d, **unsigned long long** m) {  
 **unsigned long long** x = bigmod(n, d, m);  
 **if** (x == 1 || x == m - 1)  
 **return false**;  
 **for** (**int** i = 1; i < p; ++i) {  
 x = (**\_\_uint128\_t**) x \* x % m;  
 **if** (x == m - 1)  
 **return false**;  
 }  
 **return true**;  
}  
**bool** isPrime(**unsigned long long** n) {  
 **if** (n < 2)  
 **return false**;  
 **unsigned long long** m = n;  
 n--;  
 **int** p = 0;  
 **while** (n % 2 == 0)  
 n >>= 1, p++;  
 **for** (**auto** &i: **{**2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37**}**) {  
 **if** (m == i)  
 **return true**;  
 **if** (isComposite(i, p, n, m))  
 **return false**;  
 }  
 **return true**;  
}

## Number of divisor

**long long** NumberOfDivisor(**long long** n)  
{  
 **long long** ans=1;  
 **for**(**long long** i=0; prime[i]\*prime[i]<=n; i++)  
 {  
 **long long** counter=0;  
 **while**(n%prime[i]==0)  
 {  
 n/=prime[i];  
 counter++;  
 }  
 ans\*=(counter+1);  
 }  
 **if**(n>1)ans\*=2;  
 **return** ans;  
}

## Euler Phi

**long long** phi(**long long** n)  
{  
 **long long** result = n;  
 **for** (**long long** p = 2; p \* p <= n; ++p)  
 {  
 **if** (n % p == 0)  
 {  
 **while** (n % p == 0)  
 n /= p;  
 result -= result / p;  
 }  
 }  
 **if** (n > 1)  
 result -= result / n;  
 **return** result;  
}

## Euler phi from 1 to MAX

#define **MAX** 100000  
**long long** phi[**MAX** + 7];  
**void** generatePhi()  
{  
 phi[1] = 0;  
 **for** (**long long** i = 2; i <= **MAX**; i++)  
 {  
 **if**(!phi[i])  
 {  
 phi[i] = i-1;  
 **for**(**long long** j = (i << 1); j <= **MAX**; j += i)  
 {  
 **if**(!phi[j])  
 phi[j] = j;  
 phi[j] = phi[j] \* (i-1) / i;  
 }  
 }  
 }  
}

# Data structure



*//p=1 begin=0 end=n-1*

vector<**long long**> tree, arr, lazy;  
**void** build(**int** p, **int** begin, **int** end) {  
 **if** (begin == end) {  
 tree[p] = arr[begin];  
 **return**;  
 }  
 **int** left = p << 1;  
 **int** right = (p << 1) + 1;  
 **int** mid = (begin + end) >> 1;  
 build(left, begin, mid);  
 build(right, mid + 1, end);  
 tree[p] = min(tree[left], tree[right]);  
}  
**void** update\_lazy(**int** p, **int** begin, **int** end) {  
 tree[p] += lazy[p];  
 **if** (begin != end) {  
 **int** left = p << 1;  
 **int** right = (p << 1) + 1;  
 lazy[left] += lazy[p];  
 lazy[right] += lazy[p];  
 }  
 lazy[p] = 0;  
}  
**void** update(**int** p, **int** begin, **int** end, **int** l, **int** r, **long long** value) {  
 **if** (lazy[p] != 0)  
 update\_lazy(p, begin, end);  
 **if** (l > end || r < begin)  
 **return**;  
 **if** (begin >= l && end <= r) {  
 lazy[p] += value;  
 update\_lazy(p, begin, end);  
 **return**;  
 }  
 **int** left = p << 1;  
 **int** right = (p << 1) + 1;  
 **int** mid = (begin + end) >> 1;  
 update(left, begin, mid, l, r, value);  
 update(right, mid + 1, end, l, r, value);  
 tree[p] = min(tree[left], tree[right]);  
}  
**long long** query(**int** p, **int** begin, **int** end, **int** l, **int** r) {  
 **if** (lazy[p] != 0)  
 update\_lazy(p, begin, end);  
 **if** (l > end || r < begin)  
 **return LLONG\_MAX**;  
 **if** (begin >= l && end <= r)  
 **return** tree[p];  
 **int** left = p << 1;  
 **int** right = (p << 1) + 1;  
 **int** mid = (begin + end) >> 1;  
 **long long** a = query(left, begin, mid, l, r);  
 **long long** b = query(right, mid + 1, end, l, r);  
 **return** min(a, b);  
}  
**void** segment\_tree(vector<**long long**> temp) {  
 arr = temp;  
 tree.resize(4 \* arr.size());  
 build(1, 0, arr.size() - 1);  
 lazy.assign(4 \* arr.size(), 0LL);  
}



## Segment tree

## Sqrt Decomposition

vector<**long long**> vcr;  
vector<vector<**long long** >> blocks;  
**long long** N, block\_size;  
**void** initialize() {  
 block\_size = sqrt(N);  
 **long long** block\_no = -1;  
 **for** (**int** i = 0; i < N; ++i) {  
 **if** (i % block\_size == 0) {  
 block\_no++;  
 vector<**long long**> s;  
 blocks.emplace\_back(s);  
 }  
 blocks[block\_no].push\_back(vcr[i]);  
 }  
 **for** (**auto** &i: blocks) {  
 sort(i.begin(), i.end());  
 }  
  
}  
**void** query(**int** l, **int** r, **long long** v, **int** p, **long long** u) {  
 **long long** k = 0;  
 **int** l1 = l;  
 **while** (l % block\_size && l <= r) {  
 **if** (vcr[l] < v)  
 k++;  
 l++;  
 }  
 **while** (l + block\_size <= r) {  
 **int** sz=l / block\_size;  
 k += lower\_bound(blocks[sz].begin(), blocks[sz].end(), v) -blocks[sz].begin();  
 l += block\_size;  
 }  
 **while** (l <= r) {  
 **if** (vcr[l] < v)  
 k++;  
 l++;  
 }  
 **int** sz=p / block\_size;  
 **int** x = lower\_bound(blocks[sz].begin(), blocks[sz].end(), vcr[p])-blocks[sz].begin();  
 blocks[sz][x] = (u \* k) / (r - l1 + 1);  
 vcr[p] = (u \* k) / (r - l1 + 1);  
 sort(blocks[sz].begin(), blocks[sz].end());  
}  
**void** print\_array() {  
 **for** (**auto** &i: vcr) {  
 cout << i << **endl**;  
 }  
}  
**void** sqrt\_Decomposition(vector<**long long**> &vc) {  
 N = vc.size();  
 vcr = vc;  
 initialize();  
}

## DSU

vector<**long long**> parent, siz;  
  
**void** disjointSet(**long long** n) {  
 parent.resize(n), siz.resize(n, 1);  
 iota(parent.begin(), parent.end(), 0);  
}  
  
**long long** find\_root(**long long** i) {  
 **while** (parent[i] != i) {  
 parent[i] = parent[parent[i]];  
 i = parent[i];  
 }  
 **return** i;  
}  
  
**void** weighted\_union(**long long** a, **long long** b) {  
 **long long** root\_a = find\_root(a);  
 **long long** root\_b = find\_root(b);  
 **if** (root\_a == root\_b)  
 **return**;  
 **if** (siz[root\_a] >= siz[root\_b])  
 swap(root\_a, root\_b);  
 parent[root\_a] = parent[root\_b], siz[root\_b] += siz[root\_a];  
}  
  
**bool** is\_connected(**long long** a, **long long** b) {  
 **return** find\_root(a) == find\_root(b);  
}

## Trie

vector<vector<**int**>> trie\_tree;  
**int** min\_val = **'0'**, total\_nodes = 0;  
vector<**int**> newnode;  
**void** Trie(**int** keys) {  
 newnode.resize(keys, -1);  
 trie\_tree.emplace\_back(newnode);  
  
}  
**void** push(string &s) {  
 **int** level = 0;  
 **for** (**int** i = 0; i < s.size(); ++i) {  
 **if** (trie\_tree[level][s[i] - min\_val] == -1) {  
 trie\_tree[level][s[i] - min\_val] = ++total\_nodes;  
 trie\_tree.emplace\_back(newnode);  
 }  
 level = trie\_tree[level][s[i] - min\_val];  
 }  
}  
**long long** search(string &s) {  
 **long long** level = 0,value = 0, j = s.size() - 1;  
 **for** (**int** i = 0; i < s.size(); ++i, --j) {  
 **if** (trie\_tree[level][(s[i] - min\_val) ^ 1] == -1)  
 level = trie\_tree[level][(s[i] - min\_val)];  
 **else** {  
 value = value | (1LL << j);  
 level = trie\_tree[level][(s[i] - min\_val) ^ 1];  
 }  
 }  
 **return** value;  
}

## Ordered Set

#include**<ext/pb\_ds/assoc\_container.hpp>  
using namespace** \_\_gnu\_pbds;  
**template**<**typename** T> **using** ordered\_set = tree<T,null\_type,less<T>,rb\_tree\_tag,tree\_order\_statistics\_node\_update>;

gp\_hash\_table<**int**, **int**> table;

## Sparse Table

vector<**long long** >ara;  
vector<vector<**long long** >>BiT;  
**long long** lim,N;  
**void** compute\_ST()  
{  
 **for**(**int** i=0;i<N;i++)BiT[0][i]=i;  
 **for**(**long long** k=1;(1<<k)<N;k++){  
 **for**(**long long** i=0;i+(1<<k)<=N;i++){  
 **long long** x=BiT[k-1][i];  
 **long long** y=BiT[k-1][i+(1<<k-1)];  
 BiT[k][i]=ara[x]<=ara[y] ? x : y;  
 }  
 }  
}  
**void** Sparse\_table(**long long** N,vector<**long long** >&ara)  
{  
 ara=ara;  
 N=N;  
 lim=64-\_\_builtin\_clz(N);  
 BiT.resize(lim,vector<**long long** >(N));  
 compute\_ST();  
}  
**long long** query(**long long** i,**long long** j)  
{  
 **long long** k=log2(j-i);  
 **long long** x=BiT[k][i];  
 **long long** y=BiT[k][j-(1<<k)+1];  
 **return** ara[x]<=ara[y] ? x: y;  
}

# Geometry



## Triangle

|  |  |
| --- | --- |
| Circumcircle |  |
| Incircle Radius |  |
| Excircle Radius (If the circle is tangent to side a of the triangle) |  |
| Heron’s Formula |  |
| Sine & Cosine rule | 𝑎^2 = 𝑏^2 + 𝑐^2 − 2𝑏𝑐𝐶𝑜𝑠A |

## Circle

|  |  |
| --- | --- |
| Arc Length |  |
| Sector Area |  |
| Chord length |  |
| Outside one another | 𝐶1𝐶2 > 𝑟1 + 𝑟2 |
| Touching externally | 𝐶1𝐶2 = 𝑟1 + 𝑟2 |
| Intersecting at 2 points | |𝑟1 + 𝑟2 | < 𝐶1𝐶2 < 𝑟1 + 𝑟2 |
| Touching internally | 𝐶1𝐶2 = |𝑟1 − 𝑟2| |
| One inside the other | 𝐶1𝐶2 < |𝑟1 − 𝑟2 | |

## Others

|  |  |
| --- | --- |
| Cube |  |
| Cylinder | 𝑣𝑜𝑙𝑢𝑚𝑒 = 𝜋ℎ |
| Cone | 𝑎𝑟𝑒𝑎 = 𝜋𝑟𝑙 |
| sphere | 𝑎𝑟𝑒𝑎 = 4𝜋𝑟 2  𝑣𝑜𝑙𝑢𝑚𝑒 = 4 3 𝜋𝑟3 |

# Graph

## Articulation point

bitset<10017> is\_visited;  
vector<**long long**> low, dtime;  
set<**long long**>artipoint;  
vector<vector<**long long**>> adjlist;  
**int** minutes;  
**void** articulationpoints(**long long** u, **long long** p = -1) {  
 ++minutes;  
 is\_visited[u] = **true**;  
 low[u] = dtime[u] = minutes;  
 **int** child = 0;  
 **for** (**auto** i:adjlist[u]) {  
 **if** (i == p)  
 **continue**;  
 **if** (is\_visited[i]) {  
 low[u] = min(low[u], dtime[i]);  
 } **else** {  
 articulationpoints(i, u);  
 low[u] = min(low[u], low[i]);  
 **if** (dtime[u] <= low[i] && p != -1)  
 artipoint.insert(u);  
 child++;  
 }  
 }  
 **if** (p == -1 && child > 1)  
 artipoint.insert(u);  
}

## Dijkstra

vector<**long long**> dis;  
vector<**int**> parent;  
vector<vector<pair<**int**, **int**>>> adjlist;  
**void** Dijkstra(**int** node, **int** source = 0) {  
 dis.assign(node, **LLONG\_MAX**);  
 parent.assign(node, -1);  
 dis[source] = 0;  
 priority\_queue<pair<**long long**, **int**>> pq;  
 pq.push({0, source});  
 bitset<100007> processed;  
 **while** (!pq.empty()) {  
 **int** cur\_node = pq.top().second;  
 pq.pop();  
 **if** (processed[cur\_node])  
 **continue**;  
 processed[cur\_node] = 1;  
 **for** (**auto** &i : adjlist[cur\_node]) {  
 **int** x = i.first;  
 **long long** w = i.second;  
 **if** (dis[cur\_node] + w < dis[x]) {  
 dis[x] = dis[cur\_node] + w;  
 parent[x] = cur\_node;  
 pq.push({-dis[x], x});  
 }  
 }  
 }  
}

## Bellmenford

vector<**long long**>Node[100005],cost[100005];  
**long long** n,m,i,j,cc=0,k;  
**long long** dis[100005],parent[100005];  
**long long** inf=10e9;  
**void** bellmenford(**long long** s,**long long** f)  
{  
 **for**(i=1;i<=n;i++){  
 **if**(i==s)dis[i]=0;**else** dis[i]=inf;  
 parent[i]=-1;  
 }  
 **for**(i=1;i<n;i++){  
 **bool** done=**true**;  
 **for**(j=1;j<=n;j++){  
 **for**(k=0;k<Node[j].size();k++){  
 **long long** u=j,v=Node[j][k],uv=cost[j][k];  
 **if**(dis[u]+uv<dis[v]){  
 dis[v]=dis[u]+uv;  
 parent[v]=u;  
 done=**false**;  
 }  
 }  
 }  
 **if**(done)**break**;*/// there was nothing to update ;* }  
 */// Looking for Cycle ;* **bool** found=**true**;  
 **for**(i=1;i<=n;i++){  
 **for**(j=0;j<Node[i].size();j++){  
 **long long** u=i,v=Node[i][j],uv=cost[i][j];  
 **if**(dis[u]+uv<dis[v]){  
 cout<<**"Found Negative Cycle"**<<**endl**;  
 found=**false**;  
 **return**;  
 }  
 }  
 **if**(!found)**break**;  
 }  
 **for**(i=1;i<=n;i++)  
 cout<<**"NODE : "**<<i<<**" distance : "**<<dis[i]<<**endl**;  
}

## Floyed Warshal

**long long** n,i,j,cc=0,m,k;  
**long long** adj[100][100];  
**long long** path[100][100];  
**void** floyed\_Warshal()  
{  
 **for**(k=1;k<=n;k++){  
 **for**(i=1;i<=n;i++){  
 **for**(j=1;j<=n;j++)  
 **if**(adj[i][k]+adj[k][j]<adj[i][j]){  
 adj[i][j]=adj[i][k]+adj[k][j];  
 path[i][j]=path[i][k];  
 }  
 }  
 }  
}

## Ford Fulkerson

**const int** maX=1e5+5;  
**typedef** vector<vector<**long long**>>v1;  
v1 Graph;  
**long long** capacity[1000][1000];  
**long long** n,m;  
**void** init(**int** N)  
{  
 Graph=v1(N+1);  
}  
**long long** bfs(**long long** s,**long long** t,vector<**long long**>&parent)  
{  
 fill(parent.begin(),parent.end(),-1);  
 parent[s]=-2;  
 queue<pair<**long long**,**long long**>>q;  
 q.push({s,**INT\_MAX**});  
 **while**(!q.empty()){  
 **long long** u=q.front().first;  
 **long long** flow=q.front().second;  
 q.pop();  
 **for**(**long long** i=0;i<Graph[u].size();i++){  
 **long long** v=Graph[u][i];  
 **if**(parent[v]==-1 && capacity[u][v]){  
 parent[v]=u;  
 **long long** new\_flow=min(flow,capacity[u][v]);  
 cout<<v<<**" "**;  
 **if**(v==t)**return** new\_flow;  
 q.push({v,new\_flow});  
 }  
 }  
 }  
 **return** 0;  
}  
**long long** max\_flow(**long long** s,**long long** t)  
{  
 vector<**long long**>parent(n+1);  
 **long long** flow=0;  
 **long long** new\_flow;  
 **while**( new\_flow=bfs(s,t,parent)){  
 cout<<**endl**;  
 cout<<new\_flow<<**endl**;  
 flow+=new\_flow;  
 **long long** u=t;  
 **while**(s != u){  
 **long long** prev=parent[u];  
 capacity[prev][u]-=new\_flow;  
 capacity[u][prev]+=new\_flow;  
 u=prev;  
 }  
 }  
 **return** flow;  
}

## Prim

**const int** maX=1e5+5;  
**long long** nodes,edges;  
**bool** visit[maX];  
vector<pair<**long long**,**long long**>>adj[maX];  
**long long** prim(**long long** x)  
{**long long** i,j,minimumcost=0,cost;  
 priority\_queue<pair<**long long**,**long long**>,vector<pair<**long  
 long**,**long long**>>,greater<pair<**long long**,**long long**>>> Q;  
 pair<**long long**,**long long**> p;  
 Q.push({0,x});  
 **while**(! Q.empty()){  
 p=Q.top();  
 Q.pop();  
 x=p.second;  
 **if**(visit[x]==**true**)**continue**;  
 visit[x]=**true**;  
 minimumcost+=p.first;  
 **for**(i=0;i<adj[x].size();i++){  
 **long long** y=adj[x][i].second;  
 **if**(visit[y]==**false**)Q.push(adj[x][i]);  
 }  
 }  
 **return** minimumcost;  
}

## Kruskal

**const int** maX=1e5+5;  
**long long** id[maX],nodes,edges;  
pair<**long long**,pair<**long long**,**long long**>>p[maX];  
**void** initialize()  
{  
 **for**(**int** i=1;i<maX;i++)id[i]=i;  
}  
**long long** root(**long long** x)  
{  
 **while**(x != id[x])id[x]=id[id[x]],x=id[x];  
 **return** x;  
}  
**void** union1(**long long** x,**long long** y)  
{  
 **long long** p=root(x);**long long** q=root(y);id[p]=id[q];  
}  
**long long** kruskal(pair<**long long**,pair<**long long**,**long long**>>p[])  
{  
 **long long** x,y,cost,minimumcost=0,i;  
 **for**(i=0;i<edges;i++){  
 x=p[i].second.first; y=p[i].second.second;cost=p[i].first;  
 **if**(root(x) != root(y)){  
 minimumcost+=cost;  
 union1(x,y);  
 }  
 }  
 **return** minimumcost;  
}

## LCA

**const int** maX=1e5+5;  
**int** parent[maX];  
**int** L[maX];  
vector<**int**> arr[maX];  
**int** BiT[32][maX];  
**int** visit[maX];  
**void** dfs(**int** x,**int** dep)  
{  
 **int** i,j,k;  
 visit[x]=1;  
 L[x]=dep;  
 **for**(i=0;i<arr[x].size();i++){  
 **if**(visit[arr[x][i]]==0){  
 parent[arr[x][i]]=x;  
 dfs(arr[x][i],dep+1);  
 }  
 }  
}  
**void** compute\_ST(**int** n)  
{  
 **int** i,j;  
 memset(BiT,-1,**sizeof**(BiT));  
 **for**(i=0;i<n;i++){  
 BiT[i][0]=parent[i];  
 }  
 **for**(i=1;(1<<i)<n;i++){  
 **for**(j=0;j<n;j++){  
 BiT[j][i]=BiT[BiT[j][i-1]][i-1];  
 }  
 }  
}  
**int** lca\_query(**int** p,**int** q)  
{  
 **int** temp,i,j,cc=0;  
 **if**(L[q]>L[p])swap(p,q);  
 **int** log=1;  
 **while**(1){  
 temp=log+1;  
 **if**((1<<temp)>L[p])**break**;  
 log++;  
 }  
 **for**(i=log;i>=0;i--){  
 **if**(L[p]-(1<<i)>=L[q])  
 p=BiT[p][i];  
 }  
 **if**(p==q)**return** p;  
 **for**(i=log;i>=0;i--){  
 **if**(BiT[p][i] != -1 && BiT[p][i] != BiT[q][i]){  
 p=BiT[p][i],q=BiT[q][i];  
 }  
 }  
 **return** parent[p];  
}

## SCC

**const int** maX=1e5+5;  
vector<**long long**>Graph[maX],Re\_Graph[maX],check[maX];  
**long long** visit[maX];  
stack<**long long**>ans;  
**void** dfs(**long long** u)  
{  
 visit[u]=1;  
 **for**(**long long** i=0;i<Graph[u].size();i++){  
 **long long** v=Graph[u][i];  
 **if**(!visit[v]){  
 dfs(v);  
 }  
 }  
 ans.push(u);  
}  
**void** dfs2(**long long** u,**long long** mark)  
{  
 check[mark].emplace\_back(u);  
 visit[u]=1;  
 **for**(**long long** i=0;i<Re\_Graph[u].size();i++){  
 **long long** v=Re\_Graph[u][i];  
 **if**(visit[v]==0){  
 dfs2(v,mark);  
 }  
 }  
}

## Topsort

**const int** maX=1e5+5;  
vector<**long long**>ara[maX],cost[maX];  
bitset<maX> visit;  
**long long** start[maX],finish[maX];  
**long long** cnt=0;  
vector<**long long**>ans;  
**void** dfs(**long long** source)  
{  
 cnt++;  
 visit[source]=1;  
 start[source]=cnt;  
 **for**(**int** i=0;i<ara[source].size();i++){  
 **long long** y=ara[source][i];  
 **if**(visit[y]==0){  
 dfs(y);  
 }  
 }  
 cnt++;  
 ans.emplace\_back(source);  
 finish[source]=cnt;  
 **return**;  
}



# String

*// Fills lps[] for given patttern pat[0..M-1]***void** computeLPSArray(string &pat, **int** M, **int**\* lps)  
{*// length of the previous longest prefix suffix* **int** len = 0;  
 lps[0] = 0; *// lps[0] is always 0  
 // the loop calculates lps[i] for i = 1 to M-1* **int** i = 1;  
 **while** (i < M) {  
 **if** (pat[i] == pat[len]) {  
 len++;lps[i] = len;i++;  
 }  
 **else** *// (pat[i] != pat[len])* { *// This is tricky. Consider the example. AAACAAAA and i =7. The idea is similar to search step.* **if** (len != 0) {  
 len = lps[len - 1];*// Also, note that we do not increment i here* }  
 **else** *// if (len == 0)* {  
 lps[i] = 0;  
 i++;  
 }  
 }  
 }  
}  
*// Prints occurrences of txt in pat***void** KMPSearch(string &pat, string &txt)  
{  
 **int** M = pat.size(),N = txt.size();  
 *// create lps[] that will hold the longest prefix suffix values for pattern* **int** lps[M];  
 *// Preprocess the pattern (calculate lps[] array)* computeLPSArray(pat, M, lps);  
 **int** i = 0; *// index for txt[]* **int** j = 0; *// index for pat[]* **while** (i < N) {  
 **if** (pat[j] == txt[i]) {  
 j++;i++;  
 }  
 **if** (j == M) {  
 printf(**"Found pattern at index %d "**, i - j);  
 j = lps[j - 1];  
 } *// mismatch after j matches* **else if** (i < N && pat[j] != txt[i]) {  
 *// Do not match lps[0..lps[j-1]] characters, they will match anyway* **if** (j != 0)  
 j = lps[j - 1];  
 **else** i = i + 1;  
 }  
 }  
}



## KMP

## Hashing

#define **mod1** 1000003889  
#define **mod2** 1000003133  
#define **base1** 181  
#define **base2** 271  
vector<**long long**> pw11, pw12, invpw11, invpw12, hash1, hash2;  
  
**long long** bigmod(**long long** n, **long long** p, **long long** m) {  
 **if** (p == 0)  
 **return** 1;  
 **long long** x = bigmod(n, p >> 1, m);  
 x = (x \* x) % m;  
 **if** (p & 1)  
 x = (x \* n) % m;  
 **return** x;  
}  
  
**inline void** pre(**long long** n) {  
 pw11.resize(n + 2);  
 pw12.resize(n + 2);  
 invpw11.resize(n + 2);  
 invpw12.resize(n + 2);  
 pw11[0] = pw12[0] = 1;  
 **for** (**int** i = 1; i < n + 1; ++i) {  
 pw11[i] = (pw11[i - 1] \* **base1**) % **mod1**;  
 pw12[i] = (pw12[i - 1] \* **base2**) % **mod2**;  
 }  
 invpw12[0] = invpw11[0] = 1;  
 invpw11[1] = bigmod(**base1**, **mod1** - 2, **mod1**);  
 invpw12[1] = bigmod(**base2**, **mod2** - 2, **mod2**);  
 **for** (**int** i = 2; i < n + 1; ++i) {  
 invpw11[i] = (invpw11[i - 1] \* invpw11[1]) % **mod1**;  
 invpw12[i] = (invpw12[i - 1] \* invpw12[1]) % **mod2**;  
 }  
}  
  
**void** hash(string &str) {  
 **long long** n = str.size();  
 hash1.resize(n + 1);  
 hash2.resize(n + 1);  
 hash1[0] = hash2[0] = 0;  
 **for** (**int** i = 1; i <= n; ++i) {  
 hash1[i] = (hash1[i - 1] + ((str[i - 1] - **'a'** + 1) \* 1LL \* pw11[i - 1]) % **mod1**) % **mod1**;  
 hash2[i] = (hash2[i - 1] + ((str[i - 1] - **'a'** + 1) \* 1LL \* pw12[i - 1]) % **mod2**) % **mod2**;  
 }  
}  
  
pair<**long long**, **long long**> hashvalue(**long long** l, **long long** r) {  
 *//hash(l,r) = b^−l [hash\_r+1 −hash\_l]* **long long** x = (((hash1[r] - hash1[l - 1] + **mod1**) % **mod1**) \* invpw11[l - 1]) % **mod1**;  
 **long long** y = (((hash2[r] - hash2[l - 1] + **mod2**) % **mod2**) \* invpw12[l - 1]) % **mod2**;  
 **return** {x, y};  
}

## Z function

**const long long** maX =1e6+10;  
vector<**long long**>Z(maX);  
**void** Z\_Algo(string str){  
 **long long** l=0,r=0;  
 **long long** n=str.size();  
 **for**(**long long** i=1;i<str.size();i++){  
 **if**(i<=r)Z[i]=min(r-i+1,Z[i-l]);  
 **while**(i+Z[i]<n && str[Z[i]]==str[i+Z[i]]) ++Z[i];  
 **if**(i+Z[i]-1>r)r=i+Z[i]-1,l=i;  
 }  
}

# DP



#include **<bits/stdc++.h>  
using namespace** std;  
**int** main() {  
 ios\_base::sync\_with\_stdio(**false**), cin.tie(**nullptr**);  
 **long long** test;  
 cin >> test;  
 **long long** cs = 1;  
 **while** (test--) {  
 **long long** n, m;  
 cin >> n >> m;  
 **long long** ara[n];  
 **for** (**long long** i = 0; i < n; i++)cin >> ara[i];  
 **long long** c1 = n / 2;  
 **long long** c2 = n - c1;  
 vector<**long long**> sum1(1 << c1), sum2(1 << c2);  
 **for** (**long long** i = 0; i < (1 << c1); i++) {  
 **for** (**long long** j = 0; j < c1; j++) {  
 **if** (i&(1ll<<j))  
 sum1[i] += ara[j];  
 }  
 }  
 **for** (**long long** i = 0; i < (1 << c2); i++) {  
 **for** (**long long** j = 0; j < n; j++) {  
 **if** (i&(1ll<<j))  
 sum2[i] += ara[j + c1];  
 }  
 }  
 **long long** cc = 0;  
 sort(sum2.begin(),sum2.end());  
 **for** (**long long** i = 0; i < (1 << c1); i++) {  
 **long long** x = m - sum1[i];  
 **if** (x >= 0) {  
 cc += upper\_bound(sum2.begin(),sum2.end(), x)-  
 sum2.begin();  
 }  
 }  
 cout << **"Case "** << cs++ << **": "** << cc << endl;  
 }  
 **return** 0;  
}



## Meet in the middle

## Maximum Sum Matrix

**long long** kadane(vector<**long long**> &vc)  
{  
 **long long** maxsum=0,current\_sum=0;  
 **for**(**int** i=0; i<vc.size(); i++)  
 {  
 current\_sum+=vc[i];  
 maxsum=max(current\_sum,maxsum);  
 **if**(current\_sum<0)  
 current\_sum=0;  
 }  
 **return** maxsum;  
}  
**long long** max\_sum(vector< vector<**long long**> > &matrix)  
{  
 **long long** sum=**INT\_MIN**;  
 **for**(**int** i=0; i<matrix.size(); i++)  
 {  
 vector<**long long**> vc(matrix.size(),0);  
 **for**(**int** j=i; j<matrix.size(); j++)  
 {  
 **for**(**int** k=0; k<matrix.size(); k++)  
 vc[k]+=matrix[k][j];  
 sum=max(sum,kadane(vc));  
 }  
 }  
 **return** sum;  
}