Matrix Exponentiation:

```
void multiply(int a[3][3], int b[3][3])
{
    // Creating an auxiliary matrix to store elements
    // of the multiplication matrix
    int mul[3][3];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            mul[i][j] = 0;
            for (int k = 0; k < 3; k++)
                mul[i][j] += a[i][k]*b[k][j];
        }
    }

    // storing the muliplication resul in a[][]
    for (int i=0; i<3; i++)
        for (int j=0; j<3; j++)
            a[i][j] = mul[i][j];  // Updating our matrix
}

// Function to compute F raise to power n-2.
int power(int F[3][3], int n)
{
    int M[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}};

    // Multiply it with initial values i.e with
    // F(0) = 0, F(1) = 1, F(2) = 1
    if (n==1)
        return F[0][0] + F[0][1];

    power(F, n/2);

    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);
```

```
    // Multiply it with initial values i.e with
    // F(0) = 0, F(1) = 1, F(2) = 1
    return F[0][0] + F[0][1] ;
}

// Return n'th term of a series defined using below
// recurrence relation.
// f(n) is defined as
//        f(n) = f(n-1) + f(n-2) + f(n-3), n>=3
// Base Cases :
//        f(0) = 0, f(1) = 1, f(2) = 1
int findNthTerm(int n)
{
    int F[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}} ;

    return power(F, n-2);
}
```

Minimum Spanning Tree

```
struct compare
{
    bool operator()(pair<ll,ll> p1,pair<ll,ll> p2) {
        return (p1.first>p2.first);
    }
};
ll minspantree (ll node)
{
        priority_queue <pair <ll,ll>, vector < pair <ll,ll> > , compare > q;
        q.push(make_pair(0,node));
        ll ans=0;
        while (!q.empty())
        {
                pair <ll,ll> p;
                p=q.top();
                q.pop();
                if (visited[p.second])
                        continue;
                ans+=p.first;
```

```
            if (p.first!=0)
                edges.push_back(p.first);
            visited[p.second]=1;
            for (ll i=0;i<v[p.second].size();i++)
            {
                ll ind=v[p.second][i].second;
                if (!visited[ind])
                {
                    q.push(v[p.second][i]);
                }
            }
        }
        return ans;
}
```

Segment Tree (Range minimum query):

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
#define N 100005
ll tree[2*N+1];

// segment tree for range minimum query
// 0- indexed array is used for both segment tree and original array

void build (ll a[],ll st,ll ed,ll i){
if(st==ed) {
tree[i]=a[st];
return;
}
ll mid=st+(ed-st)/2;
build(a,st,mid,2*i+1);
build(a,mid+1,ed,2*i+2);
tree[i]=min(tree[2*i+1],tree[2*i+2]);
}

void update(ll a[],ll st,ll ed, ll k,ll i, ll y){
if(k>ed||k<st) return;          // if index to be updated is out of range
if(st==ed) {
```

```
   if(st==k)
   tree[i]=y ;

   return;
}
ll mid=st+(ed-st)/2;
update(a,st,mid,k,2*i+1,y);
update(a,mid+1,ed,k,2*i+2,y);
tree[i]= min(tree[2*i+1],tree[2*i+2]);

}


ll check(ll a[],ll st,ll ed,ll l,ll r,ll node,ll x){
if(l>ed||r<st) return  INT_MAX  ;

if(st==ed)    return tree[node];


ll mid =st+(ed-st)/2;

return min(check(a,st,mid,l,r,2*node+1,x),check(a,mid+1,ed,l,r,2*node+2,x));


}
```

Bellman Ford

```
for (int i = 1; i <= V-1; i++)
    {
        for (int j = 0; j < E; j++)
        {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight <
dist[v])
                dist[v] = dist[u] + weight;
        }
```

FloydWarshell

```
void floydWarshall (int graph[][V])
{
    /* dist[][] will be the output matrix that will finally have the shortest
      distances between every pair of vertices */
    int dist[V][V], i, j, k;

    /* Initialize the solution matrix same as input graph matrix. Or
       we can say the initial values of shortest distances are based
       on shortest paths considering no intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    /* Add all vertices one by one to the set of intermediate vertices.
      ---> Before start of an iteration, we have shortest distances between all
      pairs of vertices such that the shortest distances consider only the
      vertices in set {0, 1, 2, .. k-1} as intermediate vertices.
      ----> After the end of an iteration, vertex no. k is added to the set of
      intermediate vertices and the set becomes {0, 1, 2, .. k} */
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++)
            {
                // If vertex k is on the shortest path from
                // i to j, then update the value of dist[i][j]
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
```

```
    // Print the shortest distance matrix
    printSolution(dist);
}
```

Finding Point of intersection of two line segments
Given a line passing through (x1,y1) and (x2,y2)

A = y2-y1

B = x1-x2

C = A*x1+B*y1

Find A, B, C for both lines

```
double det = A1*B2 - A2*B1

if(det == 0){

//Lines are parallel

}else{

double x = (B2*C1 - B1*C2)/det

double y = (A1*C2 - A2*C1)/det

}
```

FAST INPUT_OUTPUT

```
ios_base::sync_with_stdio(false);
   cin.tie(NULL);
```

A inverse mod m:

//value of x is the inverse

```
ll gcd(ll a, ll m, ll* x,ll* y)
{
 if(m==0){
          *x=1;
          *y=0;
          return a;
 }

ll x1,y1;
gcd(m,a%m,&x1,&y1);

*y=x1-(a/m)*y1;
*x=y1;
}
```

Big Int

```
#include <boost/multiprecision/cpp_int.hpp>
using boost::multiprecision::cpp_int;
```

Use cpp_int for bigint

Number Theory Codes

```
ll power(ll x, ll y)
{
    if (y == 0)
        return 1;
    else if (y%2 == 0)
        return power(x, y/2)*power(x, y/2);
    else
        return x*power(x, y/2)*power(x, y/2);
}
```

```
ll modular_exp (ll x,ll n,ll M)
	{
		if (n==0)
			return 1;
		else if (n%2==0)
			return (modular_exp((x*x)%M,n/2,M))%M;
		else
			return (x*modular_exp((x*x)%M,(n-1)/2,M)%M);
	}
bool isPrime[100001];
void sieve()
	{
		for(ll i = 0; i <= N;++i) {
		   isPrime[i] = true;
		}
		isPrime[0] = false;
		isPrime[1] = false;
		for(ll i = 2; i * i <= N; ++i) {
		    if(isPrime[i] == true) {
			for(ll j = i * i; j <= N ;j += i)
			    isPrime[j] = false;
		    }
		}
	}

typedef struct
	{
		ll number;
		ll power;
	}point;
vector <point> prime_factorize (ll A)
	{
		vector <point> fact_a;
		for (ll i=2;i*i<=A;i++)
		{
			ll key=0;
			while (A%i==0)
			{
```

```
                        A/=i;
                        key++;
                }
                if (key)
                {
                        point p;
                        p.number=i;
                        p.power=key;
                        fact_a.push_back(p);
                }
        }
        if (A!=1)
                {
                        point p;
                        p.number=A;
                        p.power=1;
                        fact_a.push_back(p);
                }
        return fact_a;
    }
```

Euler Totient function

```
ll phi (ll n)
{
ll result = n;
for (ll i=2; i*i<=n; ++i)
if(n %i==0)
{
while(n %i==0)
n /= i;
result -= result / i;
}
if (n > 1)
result -= result / n;
return result;
}
```

Value of ncr

```
ll ncr (ll n,ll m)
{
// calculates the value of ncm
```

```
ll i,j;
ll bc[1000][1000];
for (i=0; i<=n; i++) bc[i][0] = 1;
for (j=0; j<=n; j++) bc[j][j] = 1;
for (i=1; i<=n; i++)
for (j=1; j<i; j++)
bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
return bc[n][m];
}
```

Geometry Codes

Finding point of intersection of two lines

```
typedef struct
{
	double x,y;
}point;
typedef struct
{
	double a,b,c;
}line;
line find_abc (point p1,point p2)
{
	line l1;
	l1.a=p2.y-p1.y;
	l1.b=p1.x-p2.x;
	l1.c=a*p1.x+b*y1;
	return l1;
}
point intersection (line l1,line l2)
{
	point p;
	double det =l1.a*l2.b-l1.b*l2.a;
	if(det == 0)
	{
		p.x=INT_MAX;
		p.y=INT_MAX;
	}
```

```
        else
        {
         p.x = (l2.b*l1.c - l1.b*l2.c)/det;
         p.y = (l1.a*l2.c - l2.a*l1.c)/det;
        }
        return p;
}
```

Compiling in c++ 11

g++ -std=c++11 filename

Reading from a file

```
freopen("input.txt", "r", stdin); // redirects standard input
freopen("output.txt", "w", stdout); // redirects standard output
```

Bitwise operations

Checking odd or even

```
if (num & 1)
   cout << "ODD";
else
   cout << "EVEN";
```

Left shift and right shift

```
1<<n = 2^n
```

Checking power of two

```
bool isPowerOfTwo(ll x)
{
    // x will check if x == 0 and !(x & (x - 1)) will check if x
is a power of 2 or not
    return (x && !(x & (x - 1)));
}
```

Counting number of ones in binary

```
int count_one (int n)
  {
    while( n )
    {
    n = n&(n-1);
```

```
        count++;
    }
    return count;
}
```

Checking ith bit is set

```
bool check (int N)
{
  if( N & (1 << i) )
    return true;
  else
    return false;
}
```

Generating all possible subsets of a given set

```
void printPowerSet(char *set, int set_size)
{
    /*set_size of power set of a set with set_size
      n is (2**n -1)*/
    unsigned int pow_set_size = pow(2, set_size);
    int counter, j;

    /*Run from counter 000..0 to 111..1*/
    for(counter = 0; counter < pow_set_size; counter++)
    {
      for(j = 0; j < set_size; j++)
       {
          /* Check if jth bit in the counter is set
             If set then print jth element from set */
          if(counter & (1<<j))
            printf("%c", set[j]);
       }
       printf("\n");
    }
}
```

Returning the rightmost one in a binary number

x ^ (x & (x-1)) : Returns the rightmost 1 in binary representation of x.

x&(-x)

x | (1 << n) : Returns the number x with the nth bit set.

Number of digits in n

```
Number of digits in N =floor(log10(N)) + 1;
```

String Functions

To take an input string with spaces in it : getline(cin,s);

```
s.push_back('a');
s.pop_back();
// push_back and pop_back insert and delete at the strings end
string::iterator it;
s.begin() pointer to first element of a string
s.end() pointer to last element + 1, to access last element
s.end()-1 as
To access value of element use *it as it is a pointer
To swap elements from one string to another use str1,str2
str1.swap(str2), swaps the value in the two strings.
String Class:
Different types of constructors
s=”sample string”
string s2(s); // initializes s2 with s
string str3(5, '#'); // initializes with #####
string str4(str1, 6, 6); // second parameter is index, starting
from zero, third parameter length of substring
string str5(str2.begin(), str2.begin() + 5); //same as above but
using iterators
str4.clear(); // clears string, deletes all characters
To append to a string, s=s+s1 or s.append(s1) to append s1 to s
s.append(s1,0,6) again second paramter is the index and the
third parameter is the lenght of substring

//  find returns index where pattern is found.
    //  If pattern is not there it returns predefined
    //  constant npos whose value is -1

    if (str6.find(str4) != string::npos)
          The pos os str6.find(str4)
str6.substr(7, 3) substring of str6 starting at index 7 with
length= 3
```

Check cycle in a directed acyclic graph and topological sorting:

```
int check_cycle(int n){
queue<int> q;
for(int i=1;i<=n;i++) if(indegree[i]==0) q.push(i);
set<int> vis;
while(!q.empty()){
 int w=q.front();
cout<<w<<endl;
 vis.insert(w);
 for(int i=0;i<adj[w].size();i++){
          int k=adj[w][i];
          indegree[k]--;
          if(indegree[k]==0) q.push(k);
 }
q.pop();
}

return (vis.size()==n);

}
```

Union- find :

```
ll dsu[N];
ll size[N];



void initialise(){
   for(ll i=0;i<N;i++) dsu[i]=i;
          for(ll i=0;i<N;i++) size[i]=1;
}


ll root(ll n){
while(dsu[n]!=n)
```

```
    n=dsu[n];
return n;
}


void uni(ll p,ll q){
ll i=root(p);
ll j=root(q);

if(size[i]>size[j]) {
dsu[j]=i;
size[i]+=size[j];
}
else
{
    dsu[i]=j;
    size[j]+=size[i];
}

}

bool find(ll p, ll q){
return (root(p)==root(q)) ;

}
```

Common Dp Questions

KnapSack problem

```
int knapSack(int W, int wt[], int val[], int n)
```

```
{
   int i, w;
   int K[n+1][W+1];

   // Build table K[][] in bottom up manner
   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],
K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }

   return K[n][W];
}
```

Longest Common Subsequence

```
int LCSLength(string X, string Y)
{
     int m = X.length(), n = Y.length();

     // lookup table stores solution to already computed
sub-problems
     // i.e. lookup[i][j] stores the length of LCS of substring
     // X[0..i-1] and Y[0..j-1]
     int lookup[m + 1][n + 1];

     // first column of the lookup table will be all 0
     for (int i = 0; i <= m; i++)
          lookup[i][0] = 0;

     // first row of the lookup table will be all 0
     for (int j = 0; j <= n; j++)
```

```
          lookup[0][j] = 0;

     // fill the lookup table in bottom-up manner
     for (int i = 1; i <= m; i++)
     {
          for (int j = 1; j <= n; j++)
          {
               // if current character of X and Y matches
               if (X[i - 1] == Y[j - 1])
                    lookup[i][j] = lookup[i - 1][j - 1] + 1;

               // else if current character of X and Y don't
match
               else
                    lookup[i][j] = max(lookup[i - 1][j],
lookup[i][j - 1]);
          }
     }

     // LCS will be last entry in the lookup table
     return lookup[m][n];
}
```

Longest Increasing Subsequence

```
int LIS(int arr[], int n)
{
     // array to store sub-problem solution. L[i] stores the
length
     // of the longest increasing subsequence ends with arr[i]
     int L[n] = { 0 };

     // longest increasing subsequence ending with arr[0] has
length 1
     L[0] = 1;

     // start from second element in the array
     for (int i = 1; i < n; i++)
     {
```

```
          // do for each element in subarray arr[0..i-1]
          for (int j = 0; j < i; j++)
          {
               // find longest increasing subsequence that ends
with arr[j]
               // where arr[j] is less than the current element
arr[i]

               if (arr[j] < arr[i] && L[j] > L[i])
                    L[i] = L[j];
          }

          // include arr[i] in LIS
          L[i]++;
     }
Find max L[i] after this.

Checking if any subset can equal a sum
#include <iostream>
using namespace std;

// Return true if there exists a subarray of array[0..n) with
given sum
bool subsetSum(int arr[], int n, int sum)
{
     // T[i][j] stores true if subset with sum j can be attained
with
     // using items up to first i items
     bool T[n + 1][sum + 1];

     // if 0 items in the list and sum is non-zero
     for (int j = 1; j <= sum; j++)
          T[0][j] = false;

     // if sum is zero
     for (int i = 0; i <= n; i++)
          T[i][0] = true;

     // do for ith item
```

```
     for (int i = 1; i <= n; i++)
     {
          // consider all sum from 1 to sum
          for (int j = 1; j <= sum; j++)
          {
               // don't include ith element if j-arr[i-1] is
negative
               if (arr[i - 1] > j)
                    T[i][j] = T[i - 1][j];
               else
               // find subset with sum j by excluding or
including the ith item
                    T[i][j] = T[i - 1][j] || T[i - 1][j - arr[i
- 1]];
          }
     }

     // return maximum value
     return T[n][sum];
}
```