# **Algorithm Codes for Practice and Contests**

**01) C++ Habijabi**

**02) Prime Factorization**

**03) String Reverse**

**04) String Division**

**05) String MOD**

**06) Factorial Modulo**

**07) GCD & LCM**

**08) LCM**

**09) Dijkstra with PQ**

**10) Dijkstra with Q**

**11) Max Sub-array**

**12) DFS**

**13) BFS**

**14) LCS**

**15) LIS**

**16) Fibonacci with String [1->5000]**

**17) Euler Function**

**18) Joseph’s algorithm**

**19) Binomial Co-efficient’s**

**20) Stern Brocot Tree**

**21) Big Mod**

**22) Heap Sort**

**23) Median Finding in Linear Time**

**24) DFS [Alternate]**

**25) Bellman Ford**

**26) Coin Change [Limited]**

**27) Coin Change [unlimited]**

**28) Floyd War-shall**

**29) TP Sort**

**30) MST [Kruskal]**

**31) Merge Sort**

**32) Divisors of a Number**

**33) Sum of Divisors**

**34) Goldbach’s Conjecture**

**35) Digits of N!**

**36) Digits of XY**

**37) Trailing Zeros of Factorial**

**38) Generate Number of Divisors**

**39) Last Nonzero Digit of factorial**

**40) Inverse Modulo**

**41) Factorial Factor**

**42) Primitive Roots**

**43) Rod Cutting**

**44) Edit Distance DP**

**45) Extended Euclid Algorithm**

**46) Minimum Cost Path DP**

**47) Rat in a Maze[DP]**

**48) nCr DP**

**49) Naive Pattern Searching**

**50) Articulation Point**

**51) Articulation Bridge**

**\*\*) Geometry**

**C++ Habijabi**

**=============**

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

#include <bitset>

#include <algorithm>

#include <functional>

#include <numeric>

#include <utility>

#include <sstream>

#include <iostream>

#include <iomanip>

#include <cstdio>

#include <cmath>

#include <cstdlib>

#include <ctime>

#include <cstring>

#include <cctype>

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

#define MAX(a,b) ((a)>(b)?(a):(b))

#define MIN(a,b) ((a)<(b)?(a):(b))

#define In freopen("In.txt", "r", stdin);

#define Out freopen("out.txt", "w", stdout);

using namespace std;

**Prime Factorization**

**=================**

bool status[1100002];

void siv()

{

int i,j;

int N=1000000;

int sq=sqrt(N);

for(i=4; i<=N; i+=2) status[i]=1;

for(i=3; i<=sq; i+=2)

{

if(status[i]==0)

{

for(j=i\*i; j<=N; j+=i) status[j]=1;

}

}

status[1]=1;

}

**String Reverse**

**==============**

void string\_reverse(char c[])

{

int len = strlen(c);

int i;

int mid = len/2;

char temp;

for(i=0; i<mid; i++)

{

temp = c[i];

c[i] = c[len-i-1];

c[len-i-1] = temp;

}

c[len] = '\0';

printf("%s\n",c);

}

**String Division**

**===============**

for(i=0; i<len; i++)

{

rem = (rem \* 10 + a[i] - '0')%n;

div = (div \* 10 + a[i] - '0')/n;

b[i] = div+'0';

div = rem;

}

**String Mod**

**==========**

for(i=0; i<len; i++)

{

d = (d\*10+a[i]-'0')%n;

}

**Factorial Modulo**

**================**

int factmod ( int n, int p ) {

int res = 1 ;

while ( n > 1 ) {

res = ( res \* ( ( n / p ) % 2 ? p - 1 : 1 ) ) % p

;

for ( int i = 2 ; i <= n % p ; ++ i )

res = ( res \* i ) % p ;

n /= p ;

}

return res % p ;

}

**GCD**

**===**

int gcd ( int a, int b )

{

if ( b == 0 )

return a ;

else

return gcd ( b, a % b ) ;

}

//

int gcd ( int a, int b )

{

return b ? gcd ( b, a % b ) : a ;

}

//

int gcd ( int a, int b )

{

while ( b ) {

a % = b ;

swap ( a, b ) ;

}

return a ;

}

**LCM:**

**====**

int lcm ( int a, int b )

{

return a / gcd ( a, b ) \* b ;

}

**DijkStra's SP with PQ**

**=====================**

#define XX 10000

#define M 100

vector <int> edge[M],cost[M];

int d[M];

struct data

{

int city, dist;

bool operator < ( const data& p ) const

{

return dist > p.dist;

}

};

int dijkstra(int source, int destination)

{

memset(d,XX,sizeof(d));

priority\_queue<data> q;

data u, v;

u.city = source, u.dist = 0;

q.push( u );

d[ source ] = 0;

while( !q.empty() )

{

u = q.top();

q.pop();

int ucost = d[ u.city ];

for(int i=0; i<edge[u.city].size(); i++)

{

v.city = edge[u.city][i], v.dist = cost[u.city][i] + ucost;

if( d[v.city] > v.dist )

{

d[v.city] = v.dist;

q.push( v );

}

}

}

return d[ destination ];

}

int main()

{

int u,v,c;

int n,e,i,j;

cin >> n >> e;

for(i=0; i<e; i++)

{

cin >> u >> v >> c;

edge[u].push\_back(v);

cost[u].push\_back(c);

}

cout << endl << "List" << endl ;

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

{

cout << "Edge[" << i << "] ";

for(j=0; j<edge[i].size(); j++)

cout << edge[i][j] << " ";

cout << endl;

}

int x = dijkstra(1,n);

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

{

cout << "Distance[" << i << "]: " << d[i] << endl;

}

cout << endl << "X : " << x << endl;

return 0;

}

**DijkStra's SP with Q**

**====================**

#define INFINITY -100000

#define M 100

vector <int> edge[M],cost[M];

int dis[M];

int BFS(int source,int Destincation)

{

queue <int> Q;

Q.push(source);

memset(dis,INFINITY,sizeof(dis));

dis[source] = 0;

while(!Q.empty())

{

int u = Q.front();

Q.pop();

int uCost = dis[u];

for(int i=0; i<edge[u].size(); i++)

{

int v = edge[u][i];

int vCost = cost[u][i] + uCost;

if(dis[v] < vCost)

{

dis[v] = vCost;

Q.push(v);

}

}

}

return dis[Destincation];

}

int main()

{

int u,v,c;

int n,e,i,j;

cin >> n >> e;

for(i=0; i<e; i++)

{

cin >> u >> v >> c;

edge[u].push\_back(v);

cost[u].push\_back(c);

}

cout << endl << "List" << endl ;

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

{

cout << "Edge[" << i << "] ";

for(j=0;j<edge[i].size();j++)

cout << edge[i][j] << " ";

cout << endl;

}

int x = BFS(0,n-1);

for(i=0;i<n;i++)

{

cout << "Distance[" << i << "]: " << dis[i] << endl;

}

cout << endl << "X : " << x << endl;

return 0;

}

**Max Sub-array**

**=============**

int maxCrossingSum(int a[],int l,int m,int h)

{

int left\_sum = INT\_MIN;

int sum = 0,i;

for(i=m; i>=l; i--)

{

sum += a[i];

if(sum > left\_sum)

left\_sum = sum;

}

sum = 0;

int right\_sum = INT\_MIN;

for(i=m+1; i<=h; i++)

{

sum += a[i];

if(sum > right\_sum)

right\_sum = sum;

}

return left\_sum+right\_sum;

}

int findMaxSubarray(int a[],int low,int high)

{

if(high == low)

return a[low];

int mid = (low + high)/2;

return max(findMaxSubarray(a,low,mid),findMaxSubarray(a,mid+1,high),maxCrossingSum(a,low,mid,high));

}

**DFS**

**===**

void dfs(int g[100][100])

{

for(u=0;u<n;u++)

{

color[u]='w';

parent[u]=-1;

}

time=0;

for(u=0;u<n;u++)

{

if(color[u]=='w')

dfs\_visit(g,u);

}

}

void dfs\_visit(int g[100][100],int u)

{

int v;

time++;

distance[u]=time;

color[u]='g';

for(v=0;v<n;v++)

{

if(g[u][v]==1)

{

if(color[v]=='w')

{

parent[v]=u;

dfs\_visit(g,v);

}

}

}

color[u]=='b';

time++;

f\_time[u]=time;

}

**BFS**

**===**

vector<int>G[100];

void bfs(int n,int src)

{

queue<int>Q;

Q.push(src);

int visited[100]={0},level[100];

int parent[100];

visited[src]=1;

level[src]=0;

while(!Q.empty())

{

int u=Q.front();

for(int i=0;i<G[u].size();i++)

{

int v=G[u][i];

if(!visited[v])

{

level[v]=level[u]+1;

parent[v]=u;

visited[v]=1;

Q.push(v);

}

}

Q.pop();

}

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

printf("%d to %d distance %d",src,i,level[i]);

}

**LCS**

**===**

int lcs(void)

{

int lena = strlen(a);

int lenb = strlen(b);

int i,j;

for(i=1; i<=lena; i++)

{

for(j=1; j<=lenb; j++)

{

if(a[i-1]==b[j-1])

c[i][j] = c[i-1][j-1]+1;

else

c[i][j] = maximum(c[i][j-1],c[i-1][j]);

}

}

return c[lena][lenb];

}

**LIS**

**===**

while(scanf("%d",&n) && n)

{

for(i=0; i<n; i++)

{

scanf("%d",&arr[i]);

}

for(i=0; i<n; i++)

total[i]= 1;

for(i=0; i<n; i++)

{

l = i;

if(i!=0)

for(j=i-1; j>=0; j--)

{

if(arr[l]>arr[j])

{

l = j;

total[i]++;

}

}

r = i;

if(i!=n-1)

for(j=i+1; j<n; j++)

{

if(arr[r]<arr[j])

{

r = j;

total[i]++;

}

}

}

int num = 0;

for(i=0; i<n; i++)

{

if(total[i]>num)

num = total[i];

}

printf("%d\n",num);

}

**Fibonacci upto 5000 with String**

**===============================**

long int len1,len2,len,i,j;

char c[5001];

typedef struct

{

char bigfact[5001];

} countfactsum;

countfactsum str[5001];

void string\_reverse()

{

int len = strlen(c);

int i;

int mid = len/2;

char temp;

for(i=0; i<mid; i++)

{

temp = c[i];

c[i] = c[len-i-1];

c[len-i-1] = temp;

}

c[len] = '\0';

}

int maxx()

{

if(len1>len2)

return len1;

else

return len2;

}

void fibo()

{

char arra[5001],arrb[5001];

char b[10],a[10];

int extra,sum,k,l;

a[0] = '0';

b[0] = '1';

a[1]='\0';

b[1]='\0';

sscanf(a,"%s",str[0].bigfact);

sscanf(b,"%s",str[1].bigfact);

for(i=2; i<=5000; i++)

{

strcpy(arra,str[i-2].bigfact);

strcpy(arrb,str[i-1].bigfact);

len1 = strlen(arra);

len2 = strlen(arrb);

len = maxx();

extra = 0;

sum = 0;

j = 0;

k = len1-1;

l = len2-1;

while(1)

{

len--;

if(k<0)

arra[k] = '0';

if(l<0)

arrb[l] = '0';

sum = (arra[k]-'0') + (arrb[l]-'0') + extra;

k--;

l--;

if(sum>=10)

{

sum = sum - 10;

extra = 1;

}

else

extra = 0;

c[j] = sum + '0';

j++;

if(extra == 0 && len<=0)

break;

}

c[j]='\0';

string\_reverse(c);

strcpy(str[i].bigfact,c);

}

}

int main()

{

fibo();

long int n;

while(scanf("%ld",&n)==1)

{

printf("%s\n",str[n].bigfact);

}

return 0;

}

**Euler Function**

**==============**

int phi (int n) {

int result = n;

for (int 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;

}

**Joseph’s Algorithm**

**==================**

int joseph (int n, int k) {

if (n == 1) return 0;

if (k == 1) return n-1;

if (k > n)return (joseph (n-1, k) + k) % n;

int cnt = n/k;

int res = joseph (n - cnt, k);

res -= n % k;

if (res < 0) res += n;

else res += res / (k - 1);

return res;

}

**Binomial coefficients**

**=====================**

int C(int n, int k)

{

double res = 1;

for (int i=1; i<=k; ++i)

{

res = res \* (n-k+i) / i;

}

return (int) (res + 0.01);

}

**Stern - Brocot Tree**

**====================**

**// General Algo**

string find (int x, int y, int a = 0, int b = 1, int

c = 1, int d = 0) {

int m = a+c, n = b+d;

if (x == m && y == n)

return "";

if (x \* n < y \* m)

return 'L' + find (x, y, a, b, m, n);

else

return 'R' + find (x, y, m, n, c, d);

}

**// UVA Problem Algo**

for(i=0; i<n; i++)

{

a = 0,b = 1,c = 1,d = 0;

x = a + b;

y = c + d;

gets(str);

len = strlen(str);

for(j=0; j<len; j++)

{

if(str[j]=='R')

{

a = x;

c = y;

}

else if(str[j]=='L')

{

b = x;

d = y;

}

x = a + b;

y = c + d;

}

printf("%lld/%lld\n",x,y);

}

**Big Mod**

**========**

while(scanf("%lld %lld %lld",&n,&p,&m)==3)

{

long long ans=1;

while(p>0)

{

if(p%2==1)

ans=(ans\*n);

ans=ans%m;

n=(n\*n);

n=n%m;

p=p/2;

}

printf("%lld\n",ans);

}

**Heapsort**

**=========**

#include <stdio.h>

int heapsize,length;

void max\_heapify(int a[],int i)

{

int largest,l,r;

l=2\*i;

r=2\*i+1;

if(l<=heapsize&&a[l]>a[i])

largest=l;

else

largest=i;

if(r<=heapsize&&a[r]>a[largest])

largest=r;

if(largest!=i)

{

int t=a[i];

a[i]=a[largest];

a[largest]=t;

max\_heapify(a,largest);

}

return;

}

void build\_max\_heap(int a[])

{

int i;

heapsize=length;

for(i=length/2;i>=0;i--)

max\_heapify(a,i);

return;

}

void heapsort(int a[])

{

build\_max\_heap(a);

int i;

int l=2\*i,r=(2\*i)+1;

for(i=length;i>=1;i--)

{

int t=a[l];

a[l]=a[i];

a[i]=t;

heapsize--;

max\_heapify(a,0);

}

return;

}

int main()

{

int a[100],i;

scanf("%d",&length);

for(i=0;i<length;i++)

scanf("%d",&a[i]);

heapsort(a);

for(i=0;i<length;i++)

printf("%d ",a[i]);

return 0;

}

**Median Finding Algo in Leaner Time**

**==================================**

int N;

int A[] = {5,7,-1,-8,9,2,13};

int partitions(int low,int high)

{

int p=low,r=high,x=A[r],i=p-1;

for(int j=p; j<=r-1; j++)

{

if (A[j]<=x)

{

i=i+1;

swap(A[i],A[j]);

}

}

swap(A[i+1],A[r]);

return i+1;

}

int selection\_algorithm(int left,int right,int kth)

{

while(true)

{

int pivotIndex=partitions(left,right); //Select the Pivot Between Left and Right

int len=pivotIndex-left+1;

if(kth==len)

return A[pivotIndex];

else if(kth<len)

right=pivotIndex-1;

else

{

kth=kth-len;

left=pivotIndex+1;

}

}

}

int main()

{

int sz = sizeof(A)/sizeof(A[0]);

int loc = selection\_algorithm(0,sz-1,(sz+1)/2);

for(int i=0; i<sz; i++)

cout << A[i] << " " ;

cout << endl << loc << endl;

return 0;

}

**DFS[Alternate]**

**==============**

int node,grid[m][m],distance[m]= {0},times=0,dist[m]= {0};

bool visited[m];

void initialize()

{

memset(grid,0,sizeof(grid));

memset(visited,false,sizeof(visited));

return;

}

void dfs(int u)

{

visited[u] = true;

for(int i=1; i<=node; i++)

{

if(grid[u][i] && !visited[i])

dfs(i);

}

return;

}

int main()

{

int edge,u,v,cnt=0;

cin >> node >> edge;

for(int i=1; i<=edge; i++)

{

cin >> u >> v;

grid[u][v] = 1;

}

for(int i=1; i<=node; i++)

{

if(!visited[i])

{

cnt++;

dfs(i);

}

}

cout << "DFS Visit Total: " << cnt << endl;

return 0;

}

**Bellman Ford Algo**

**=================**

int Distance[100];

int node,vertx;

struct vertices

{

int source,destination,cost;

void sourceDestinationCost(int \_source,int \_destination,int \_cost)

{

source = \_source;

destination = \_destination;

cost = \_cost;

}

} vertxx[100];

void Bellman\_Ford(int src)

{

bool yesOrNo = true;

memset(Distance,int\_\_max,sizeof(Distance));

Distance[src] = 0;

for(int i=1; i<node; i++)

{

for(int j=0; j<vertx; j++)

{

int u = vertxx[j].source;

int v = vertxx[j].destination;

int weight = vertxx[j].cost;

if(Distance[u]+weight < Distance[v])

Distance[v] = Distance[u]+weight;

}

}

for(int j=0; j<vertx; j++)

{

int u = vertxx[j].source;

int v = vertxx[j].destination;

int weight = vertxx[j].cost;

if(Distance[u]+weight < Distance[v])

{

yesOrNo = false;

break;

}

}

if(yesOrNo==true)

{

cout << "No Negative Cycle!!" << endl;

}

else

cout << "Negative Cycle Available!!" << endl;

for(int i=0; i<node; i++)

cout << i << " " << Distance[i] << endl;

}

int main()

{

cin >> node >> vertx;

int i,j;

int u,v,cost;

for(int i=0; i<vertx; i++)

{

cin >> u >> v >> cost;

vertxx[i].sourceDestinationCost(u,v,cost);

}

Bellman\_Ford(0); // Here Zero is Source

return 0;

}

**Coin Change[Limited]**

**====================**

int main()

{

///\*\*\* Credit: KN Roy \*\*\*///

int i,j;

int coinNumber;

int coinNumberStore[10];

int highestGivingTotal = 0;

int coinChange[100];

memset(coinChange,0,sizeof(coinChange));

coinChange[0] = 1;

scanf("%d",&coinNumber);

for(i=1; i<=coinNumber; i++)

{

scanf("%d",&coinNumberStore[i]);

highestGivingTotal += coinNumberStore[i];

}

for(i=1; i<=coinNumber; i++)

{

for(j=highestGivingTotal; j>=0; j--)

{

if(coinChange[j]!=0)

coinChange[j+coinNumberStore[i]] = coinChange[j] + coinChange[j+coinNumberStore[i]];

}

}

for(j=0; j<=highestGivingTotal; j++)

{

printf("coinChange[%d] = %d\n",j,coinChange[j]);

}

int MaxWayToGiveMoney = 0;

for(j=0; j<=highestGivingTotal; j++)

if(coinChange[j]>=MaxWayToGiveMoney)

MaxWayToGiveMoney = coinChange[j];

printf("Max Way to Give Money = %d\n",MaxWayToGiveMoney);

return 0;

}

**Coin Change[Unlimited]**

**======================**

int main()

{

int i,j;

int coinNumber;

int coinNumberStore[10];

int coinChange[102];

memset(coinChange,0,sizeof(coinChange));

coinChange[0] = 1;

scanf("%d",&coinNumber);

for(i=1; i<=coinNumber; i++)

scanf("%d",&coinNumberStore[i]);

for(i=1; i<=coinNumber; i++)

for(j=0; j<=100; j++)

if(coinChange[j]!=0)

coinChange[j+coinNumberStore[i]] = coinChange[j] + coinChange[j+coinNumberStore[i]];

for(j=0; j<=100; j++)

printf("coinChange[%d] = %d\n",j,coinChange[j]);

int MaxWayToGiveMoney = 0;

for(j=0; j<=100; j++)

if(coinChange[j]>=MaxWayToGiveMoney)

MaxWayToGiveMoney = coinChange[j];

printf("Max Way to Give Money = %d\n",MaxWayToGiveMoney);

return 0;

}

**Floyd Warshall**

**==============**

#define sz 20

int node,dis[sz][sz];

void floyd\_warshall()

{

for(int k=1;k<=node;k++)

for(int i=1;i<=node;i++)

for(int j=1;j<=node;j++)

dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);

}

main()

{

int edge,u,v,w;

while(cin>>node>>edge)

{

memset(dis,1,sizeof(dis));

for(int i=0;i<=node;i++) dis[i][i]=0;

for(int i=1;i<=edge;i++)

{

cin>>u>>v>>w;

dis[u][v]=w;

}

floyd\_warshall();

for(int i=1;i<=node;i++)

{

for(int j=1;j<=node;j++)

if(dis[i][j]>9999) cout<<"~~ ";

else printf("%2d ",dis[i][j]);

cout<<endl<<endl;

}

}

}

**Topological Sort**

**================**

#define s 100

using namespace std;

typedef struct tag {

int node,time;

}finishing\_time;

bool comp(finishing\_time a,finishing\_time b)

{

if(a.time>b.time) return true;

return false;

}

finishing\_time f[s];

int t,a[s][s],d[s],node;

bool color[s];

void dfs(int u)

{

color[u]=true;

d[u]=++t;

for(int i=1;i<=node;i++)

if(a[u][i] && !color[i])

dfs(i);

f[u].node=u;

f[u].time=++t;

return ;

}

main()

{

int j,u,v,w,edge;

memset(color,false,sizeof(color));

memset(a,0,sizeof(a));

for(int i=1;i<=node;i++)

f[i].time=0;

t=0;

cin>>node>>edge;

for(int i=1;i<=edge;i++)

{

cin>>u>>v;

a[u][v]=1;

}

for(int i=1;i<=node;i++)

if(!color[i]) dfs(i);

sort(&f[1],&f[node+1],comp);

cout<<endl<<"Topological Order: ";

for(int i=1;i<node;i++)

cout<<f[i].node<<" ";

cout<<f[node].node<<endl;

}

**MST[Krushkal]**

**============**

int rnk[m],parent[m],flag[m];

struct frnds

{

int u,v,cost;

void udp(int \_u,int \_v, int \_cost)

{

u = \_u;

v = \_v;

cost = \_cost;

}

} frnd[m];

bool comp(frnds p,frnds q)

{

if(p.cost>q.cost)

return true;

return false;

}

void make\_set(int n)

{

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

{

parent[i] = i;

rnk[i] = 0;

}

return;

}

int find\_set(int x)

{

if( x != parent[x])

parent[x] = find\_set(parent[x]);

return parent[x];

}

void Union(int a,int b)

{

if(rnk[a] > rnk[b])

parent[b] = a;

else

parent[a] = b;

if(rnk[a]==rnk[b])

rnk[a]++;

return;

}

void Kruskal(int N,int E)

{

make\_set(N);

memset(flag,0,sizeof(flag));

int u,v,cost,total\_cost = 0,cnt=0;

for(int i = 1; i<=E && cnt < N-1; i++)

{

u = frnd[i].u;

v = frnd[i].v;

cost = frnd[i].cost;

if(find\_set(u)!=find\_set(v))

{

total\_cost += cost;

Union(find\_set(u),find\_set(v));

flag[i] = true;

cnt++;

}

}

printf("MST COST = %d\n",total\_cost);

printf("frnds Used\n");

for(int i=1; i<=E; i++)

{

if(flag[i]==true)

printf("%d ---> %d\n",frnd[i].u,frnd[i].v);

}

return;

}

int main()

{

int i,j,G[m][m]= {0};

int node,e;

int ua,uv,ucost;

while(cin >> node >> e)

{

for(i=1; i<=e; i++)

{

cin >> ua >> uv >> ucost;

frnd[i].udp(ua,uv,ucost);

}

sort(&frnd[1],&frnd[e],comp);

Kruskal(node,e);

}

return 0;

}

**Mergesort**

**=========**

#define MEX 100

using namespace std;

int cnt;

void merge(int lef[],int rig[],int arr[],int m,int n)

{

int i=0,j=0,k=0;

int nL = m;

int nR = n;

while(i<nL && j<nR)

{

if(lef[i]<=rig[j])

{

arr[k] = lef[i];

i++;

}

else

{

arr[k] = rig[j];

j++;

cnt += m - i;

}

k++;

}

while(i<nL)

{

arr[k] = lef[i];

i++;

k++;

}

while(j<nL)

{

arr[k] = rig[j];

j++;

k++;

}

void mergesort(int arr[],int n)

{

int i;

if(n<2)

return;

int mid = (n+1)/2;

int left[mid],right[n-mid];

for(i=0; i<mid; i++)

left[i] = arr[i];

for(i = mid; i<n; i++)

right[i-mid] = arr[i];

mergesort(left,mid);

mergesort(right,n-mid);

merge(left,right,arr,mid,n-mid);

}

**How many Divisors of a Number**

**=============================**

#include <stdio.h>

#include <math.h>

#include <stdbool.h>

#define SIZE\_N 1000

#define SIZE\_P 1000

bool flag[SIZE\_N+5];

int primes[SIZE\_P+5];

int seive()

{

int i,j,total=0,val;

for(i=2; i<=SIZE\_N; i++) flag[i]=1;

val=sqrt(SIZE\_N)+1;

for(i=2; i<val; i++)

if(flag[i])

for(j=i; j\*i<=SIZE\_N; j++)

flag[i\*j]=0;

for(i=2; i<=SIZE\_N; i++)

if(flag[i])

primes[total++]=i;

return total;

}

int divisor(int N)

{

int i,val,count,sum;

val=sqrt(N)+1;

sum=1;

for(i=0; primes[i]<val; i++)

{

if(N%primes[i]==0)

{

count=0;

while(N%primes[i]==0)

{

N/=primes[i];

count++;

}

sum\*=(count+1);

}

}

if(N>1)

sum=sum\*2;

return sum;

}

**Sum of Divisors**

**===============**

#include <stdio.h>

#include <math.h>

#include <stdbool.h>

#define SIZE\_N 1000

#define SIZE\_P 1000

bool flag[SIZE\_N+5];

int primes[SIZE\_P+5];

int seive()

{

int i,j,total=0,val;

for(i=2; i<=SIZE\_N; i++) flag[i]=1;

val=sqrt(SIZE\_N)+1;

for(i=2; i<val; i++)

if(flag[i])

for(j=i; j\*i<=SIZE\_N; j++)

flag[i\*j]=0;

for(i=2; i<=SIZE\_N; i++)

if(flag[i])

primes[total++]=i;

return total;

}

int Sum\_Of\_Divisor(int N)

{

int i,val,count,sum,p,s;

val=sqrt(N)+1;

sum=1;

for(i=0; primes[i]<val; i++)

{

if(N%primes[i]==0)

{

p=1;

while(N%primes[i]==0)

{

N/=primes[i];

p=p\*primes[i];

}

p=p\*primes[i];

s=(p-1)/(primes[i]-1);

sum=sum\*s;

}

}

if(N>1)

{

p=N\*N;

s=(p-1)/(N-1);

sum=sum\*s;

}

return sum;

}

**Goldbach’s Conjecture**

**=====================**

int FindSol(int n)

{

int i, res=0;

for(i=2; i<=n/2; i++)

if(flag[i] && flag[n-i])

res++;

return res;

}

**How Many Digits of N!**

**=====================**

long long int digit[10000001];

void digitcount()

{

long long int n=10000001,i;

double sum = 0.0;

digit[0] = 1;

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

{

sum += log10((double)i);

digit[i] = ceil(sum);

}

digit[1] = 1;

}

**How many Digit’s of XY**

**======================**

D=floor[log10(N!)]+1

=floor[log10(1x2x3x…………xN)]+1

=floor[log10(1) + log10(2) + ……………+log10(N)]+1

**Trailing Zeros of a Factorial**

**=============================**

#include <stdio.h>

int Trailing\_Zeros(int N)

{

int sum=0;

while(N)

{

sum+=N/5;

N=N/5;

}

return sum;

}

**Generate Number of Divisors [1 to N]**

**====================================**

int D[1000010];

void DivisorGenerate()

{

int i,j,val,N,M,count;

D[1]=1;

for(i=2; i<=1000000; i++)

{

N=M=i;

val=sqrt(N)+1;

for(j=0; primes[j]<val; j++)

{

if(M%primes[j]==0)

{

count=0;

while(M%primes[j]==0)

{

M/=primes[j];

count++;

}

D[N]=(count+1)\*D[M];

break;

}

}

if(M==N) //Special Case if N equal prime

{

D[N]=2;

}

}

}

**Last Non Zero Digit of Factorial**

**================================**

int PTwo(int N)

{

int T[]= {6,2,4,8};

if(N==0) return 1;

return T[N%4];

}

int LastNZDigit(int N)

{

int A[]= {1,1,2,6,4};

if(N<5) return A[N];

return (PTwo(N/5)\*LastNZDigit(N/5)\*LastNZDigit(N%5))%10;

}

**Inverse Modulo (Using Fermat Little Theorem)**

**============================================**

int BigMod(long long B,long long P,long long M)

{

long long R=1;

while(P>0)

{

if(P%2==1)

{

R=(R\*B)%M;

}

P/=2;

B=(B\*B)%M;

}

return R;

}

int Inverse\_Modulo(int a,int m)

{

return BigMod(a,m-2,m);

}

**Factorial Factor**

**================**

int Factor(int N,int P)

{

if(N==0) return 0;

return Factor(N/P,P)+N/P;

}

**Primitive Roots**

**===============**

int powmod (int a, int b, int p)

{

int res = 1;

while (b)

if (b & 1)

res = int (res \* 1ll \* a % p), --b;

else

a = int (a \* 1ll \* a % p), b >>= 1;

return res;

}

int generator (int p)

{

vector<int> fact;

int phi = p-1, n = phi;

for (int i=2; i\*i<=n; ++i)

if (n % i == 0)

{

fact.push\_back (i);

while (n % i == 0)

n /= i;

}

if (n > 1)

fact.push\_back (n);

for (int res=2; res<=p; ++res)

{

bool ok = true;

for (size\_t i=0; i<fact.size() && ok; ++i)

ok &= powmod (res, phi / fact[i], p) != 1;

if (ok) return res;

}

return -1;

}

**Rod Cutting**

**===========**

int RodCut(int n)

{

if(n<=0)

return 0;

int Max\_value = INT\_MIN;

for(int i=0; i<n; i++)

{

Max\_value = max(Max\_value,arr[i] + RodCut(n-i-1));

}

return Max\_value;

}

**Edit Distance DP**

**================**

int editDistance()

{

int i,j;

int l1 = strlen(str1);

int l2 = strlen(str2);

for(i=0;i<=l1;i++)

DP[0][i] = i;

for(i=0;i<=l2;i++)

DP[i][0] = i;

for(i=1;i<=l1;i++)

{

for(j=1;j<=l2;j++)

{

if(l1==0)

return l2;

else if(l2==0)

return l1;

else if(str1[i-1]==str2[j-1])

DP[i][j] = min(min(1+DP[i-1][j],1+DP[i][j-1]),DP[i-1][j-1]);

else

DP[i][j] = min(DP[i-1][j-1],min(DP[i][j-1],DP[i-1][j])) + 1;

}

}

return DP[l1][l2];

}

**Extended Euclid Algorithm**

**=========================**

void extended\_euclid(int A,int B)

{

int m,n,p,q,a,b;

int x=1,y=0;

int u=0,v=1;

a = A,b = B;

while(b!=0)

{

m = a%b;

q = a/b;

x = x - (u\*q);

y = y - (v\*q);

swap(x,u);

swap(y,v);

a = b;

b = m;

}

cout << A << "(" << x << ") + " << B << "(" << y << ") = " << a << endl;

}

**Minimum Cost Path DP**

**====================**

int minCostDP(int m, int n) //With memoization O(mn)

{

int i, j;

int tc[R][C];

tc[0][0] = cost[0][0];

for (i = 1; i <= m; i++)

tc[i][0] = tc[i-1][0] + cost[i][0];

for (j = 1; j <= n; j++)

tc[0][j] = tc[0][j-1] + cost[0][j];

for (i = 1; i <= m; i++)

for (j = 1; j <= n; j++)

tc[i][j] = min(tc[i-1][j-1], tc[i-1][j], tc[i][j-1]) + cost[i][j];

return tc[m][n];

}

**Rat in a Maze [Backtrack]**

**=========================**

bool isSafe(int i,int j)

{

if(Arr[i][j]==-1 && (i>0 && j>0) && (i<=m && j<=n) && V[i][j]==0)

{

track.push\_back(i);

track.push\_back(j);

return 1;

}

return 0;

}

void backtrack(int i,int j,int r,int s)

{

if(i==r && j==s)

{

//memset(Arr,-1,sizeof(Arr));

for(int x = 0; x<track.size(); x++)

{

cout << track[x] << " ";

}

cout << endl;

track.clear();

}

else

{

if(isSafe(i,j+1)==1)

{

Arr[i][j+1] = 1;

backtrack(i,j+1,r,s);

}

if(isSafe(i+1,j)==1)

{

Arr[i+1][j] = 1;

backtrack(i+1,j,r,s);

}

}

}

**nCr DP**

**======**

unsigned long long nCr(int n, int r)

{

if(n==r) return dp[n][r] = 1;

if(r==0) return dp[n][r] = 1;

if(r==1) return dp[n][r] = (i64)n;

if(dp[n][r]) return dp[n][r];

return dp[n][r] = nCr(n-1,r) + nCr(n-1,r-1);

}

**Naïve Pattern Searching**

**=======================**

void searchPattern(string x,string p)

{

int l1 = x.length();

int l2 = p.length();

int i,j;

for(i=0; i<l1-l2; i++)

{

string tmp = x.substr(i,l2);

if(p.compare(tmp)==0)

cout << i << endl;

}

return;

}

**Articulation Point**

**==================**

void dfs\_artpoint(int u)

{

int i,j,c,child=0;

dis[u] = low[u] = ++tme;

art[u] = false;

for(i=0;i<graph[u].size();i++)

{

v = graph[u][i];

if(dis[v]==0)

{

child++;

dfs\_artpoint(v);

low[u] = min(low[u],low[v]);

if(low[v] >= dis[u] && u!=root)

art[u] = true;

}

else if(dis[v] < dis[u])

low[u] = min(low[u],dis[v]);

}

if(u==root && child==2)

art[u] = true;

return;

}

**Articulation Bridge**

**===================**

void BPM(int n,int m)

{

int ret = 0;

for(i=1;i<=n;i++) parent[i] = -1;

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

{

memset(color,0,sizeof(color));

if(dfs(i))

ret++;

}

return;

}

int dfs(int u)

{

if(color[u]) return false;

color[u] = true;

for(int i=0;i<SZ(graph[u]);i++)

{

int v = graph[u][i];

if(parent[v]==-1 || dfs(parent[v]))

{

parent[v] = u;

return true;

}

}

return false;

}

==============================================================================================================

**Geometry**

**========**

//Compute the dot product AB ⋅ BC

int dot(int[] A, int[] B, int[] C){

AB = new int[2];

BC = new int[2];

AB[0] = B[0]-A[0];

AB[1] = B[1]-A[1];

BC[0] = C[0]-B[0];

BC[1] = C[1]-B[1];

int dot = AB[0] \* BC[0] + AB[1] \* BC[1];

return dot;

}

//Compute the cross product AB x AC

int cross(int[] A, int[] B, int[] C){

AB = new int[2];

AC = new int[2];

AB[0] = B[0]-A[0];

AB[1] = B[1]-A[1];

AC[0] = C[0]-A[0];

AC[1] = C[1]-A[1];

int cross = AB[0] \* AC[1] - AB[1] \* AC[0];

return cross;

}

//Compute the distance from A to B

double distance(int[] A, int[] B){

int d1 = A[0] - B[0];

int d2 = A[1] - B[1];

return sqrt(d1\*d1+d2\*d2);

}

//Compute the distance from AB to C

//if isSegment is true, AB is a segment, not a line.

double linePointDist(int[] A, int[] B, int[] C, boolean isSegment){

double dist = cross(A,B,C) / distance(A,B);

if(isSegment){

int dot1 = dot(A,B,C);

if(dot1 > 0)return distance(B,C);

int dot2 = dot(B,A,C);

if(dot2 > 0)return distance(A,C);

}

return abs(dist);

}

//Compute the distance from AB to C

//if isSegment is true, AB is a segment, not a line.

double linePointDist(point A, point B, point C, bool isSegment){

double dist = ((B-A)^(C-A)) / sqrt((B-A)\*(B-A));

if(isSegment){

int dot1 = (C-B)\*(B-A);

if(dot1 > 0)return sqrt((B-C)\*(B-C));

int dot2 = (C-A)\*(A-B);

if(dot2 > 0)return sqrt((A-C)\*(A-C));

}

return abs(dist);

}

**Polygon Area**

**==========**

int area = 0;

int N = lengthof(p);

//We will triangulate the polygon

//into triangles with points p[0],p[i],p[i+1]

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

int x1 = p[i][0] - p[0][0];

int y1 = p[i][1] - p[0][1];

int x2 = p[i+1][0] - p[0][0];

int y2 = p[i+1][1] - p[0][1];

int cross = x1\*y2 - x2\*y1;

area += cross;

}

return abs(cross/2.0);

**Line-Line Intersection**

**=================**

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

if(det == 0){

//Lines are parallel

}else{

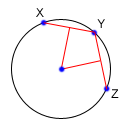
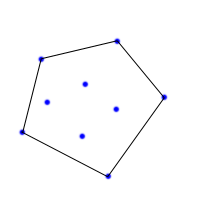
double x = (B2\*C1 - B1\*C2)/det

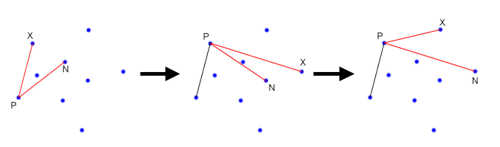
double y = (A1\*C2 - A2\*C1)/det

}

**Convex Hull**

**==========**





convexHull(point[] X){

int N = lengthof(X);

int p = 0;

//First find the leftmost point

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

if(X[i] < X[p])

p = i;

}

int start = p;

do{

int n = -1;

for(int i = 0; i<N; i++){

//Don't go back to the same point you came from

if(i == p)continue;

//If there is no N yet, set it to i

if(n == -1)n = i;

int cross = (X[i] - X[p]) x (X[n] - X[p]);

if(cross < 0){

//As described above, set N=X

n = i;

}

}

p = n;

}while(start!=p);

}

[Alternate:]

//If onEdge is true, use as many points as possible for

//the convex hull, otherwise as few as possible.

convexHull(point[] X, boolean onEdge){

int N = lengthof(X);

int p = 0;

boolean[] used = new boolean[N];

//First find the leftmost point

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

if(X[i] < X[p])

p = i;

}

int start = p;

do{

int n = -1;

int dist = onEdge?INF:0;

for(int i = 0; i<N; i++){

//X[i] is the X in the discussion

//Don't go back to the same point you came from

if(i==p)continue;

//Don't go to a visited point

if(used[i])continue;

//If there is no N yet, set it to X

if(n == -1)n = i;

int cross = (X[i] - X[p]) x (X[n] - X[p]);

//d is the distance from P to X

int d = (X[i] - X[p]) ⋅ (X[i] - X[p]);

if(cross < 0){

//As described above, set N=X

n = i;

dist = d;

}else if(cross == 0){

//In this case, both N and X are in the

//same direction. If onEdge is true, pick the

//closest one, otherwise pick the farthest one.

if(onEdge && d < dist){

dist = d;

n = i;

}else if(!onEdge && d > dist){

dist = d;

n = i;

}

}

}

p = n;

used[p] = true;

}while(start!=p);

}

**Intersect of Circle and St. line**

**================================**

int main()

{

double a,b,c;

double p,q,r,s,t,m,n;

cout << "Enter Straight Line Equation: ";

scanf("%lf %lf",&m,&n);

cout << "Enter Circle Equation: ";

scanf("%lf %lf %lf",&p,&q,&r);

a = m\*m + 1;

b = -2\*p + 2 \* m \* (n-q);

c = p\*p + (n-q)\*(n-q) - r;

cout << a << " " << b << " " << c << endl;

double x1,x2,y1,y2,ab;

ab = b\*b - 4 \* a \* c;

cout << ab << endl;

if(ab<0)

cout << "AB < 0" << endl;

else

{

ab = sqrt(ab);

x1 = (-b + ab)/(2\*a);

x2 = (-b - ab)/(2\*a);

y1 = m \* x1 + n;

y2 = m \* x2 + n;

cout << "(" << x1 << "," << y1 << ")" << "(" << x2 << "," << y2 << ")" << endl;

}

return 0;

}

**New Added Codes**

**=================**

**0-1 Knapsack**

**============**

**int CAP;**

**int weight[sz+1];**

**int cost[sz+1];**

**int dp[sz+1][sz\_w+1];**

**int n;**

**void init()**

**{**

**MEM(dp,-1);**

**}**

**int func(int i,int w)**

**{**

**if(i == n)**

**return 0;**

**if(dp[i][w]!=-1)**

**return dp[i][w];**

**int prof1 = 0,prof2 = 0;**

**if(w+weight[i]<=CAP)**

**prof1 = cost[i] + func(i+1,w+weight[i]);**

**prof2 = func(i+1,w);**

**dp[i][w] = MAX(prof1,prof2);**

**return dp[i][w];**

**}**

**int main()**

**{**

**init();**

**cin >> n >> CAP;**

**for(int i=0; i<n; i++)**

**{**

**cin >> weight[i] >> cost[i];**

**}**

**cout << func(0,0) << endl;**

**return 0;**

**}**

**Longest Palindromic Subsequence**

**================================**

class myString

{

public:

static string rev(string str)

{

int ln = str.size();

for(int i=0; i<ln/2; i++)

{

char tmp = str[i];

str[i] = str[ln-i-1];

str[ln-i-1] = tmp;

}

return str;

}

};

int LCS(string a,string b)

{

int c[100][100];

MEM(c,0);

int lena = a.size();

int lenb = b.size();

int i,j;

for(i=1; i<=lena; i++)

{

for(j=1; j<=lenb; j++)

{

if(a[i-1]==b[j-1])

c[i][j] = c[i-1][j-1]+1;

else

c[i][j] = MAX(c[i][j-1],c[i-1][j]);

}

}

return c[lena][lenb];

}

int LPS(string str)

{

int len = str.size();

int i,j,k;

int dp[100][100];

for(i=0;i<len;i++)

dp[i][i] = 1;

for(k=2;k<=len;k++)

{

for(i=0;i<len-k+1;i++)

{

j = i+k-1;

if(str[i] == str[j] && k==2)

dp[i][j] = 2;

else if(str[i] == str[j])

dp[i][j] = dp[i+1][j-1]+2;

else

dp[i][j] = MAX(dp[i][j-1],dp[i+1][j]);

}

}

return dp[0][len-1];

}

int main()

{

// This method is working with LCS(string a, revese(string a))

myString s;

string ss;

cin >> ss;

int n = LCS(ss,s.rev(ss));

cout << "Using LCS(s,rev(s)): " << n << endl;

cout << "Using LPS: " << LPS(ss) << endl;

}

**Rock Climing**

**============**

#define inf 1<<28

int mat[][10]=

{

{-1, 2, 5},

{4, -2, 3},

{1 , 2 ,10,}

};

int dp[10][10];

int r=3,c=3;

int rockClimb(int i,int j)

{

if((i>=0 && i<r) && (j>=0 && j<c))

{

if(dp[i][j]!=-1)

return dp[i][j];

int ret = -inf;

ret = MAX(ret,rockClimb(i+1,j)+mat[i][j]);

ret = MAX(ret,rockClimb(i+1,j-1)+mat[i][j]);

ret = MAX(ret,rockClimb(i+1,j+1)+mat[i][j]);

return dp[i][j] = ret;

}

else return 0;

}

int main()

{

MEM(dp,-1);

cout << rockClimb(0,0) << endl;

return 0;

}

**Sum of digits upto N**

**=====================**

#define limit 10000000

int dp[limit+5];

int totalSum(int n)

{

int total = 0;

while(n!=0)

{

int tmp = n % 10;

total += tmp;

n /= 10;

}

return total;

}

void sumDigitsUptoN()

{

dp[0] = 0;

for(int i=1; i<limit; i++)

{

dp[i] = dp[i-1] + totalSum(i);

}

}

int main()

{

sumDigitsUptoN();

int n;

while(cin >> n)

{

cout << dp[n] << endl;

}

return 0;

}