// delete element

```
int pop() {
        nextIndex--;
        return data[nextIndex];

    if(isempty())
        {
        cout<<"Stack is empty"<<endl;
        return INT_MIN;
        }
    nextIndex--
    return data[nextIndex];
    }

int Top()
    {
    if(isEmpty())
        {
        cout<<"Stack is empty"<<endl;
        return INT_MIN;
        }
    nextIndex--;
    return data[nextInd-1]
    }
```

Dynamic Stacks.

Changes.

```
Stacks using array (){
    data = new Int[4]
    nextIndex = 0
    capacity = 4;
}

void push (int element)
{
    if (nextIndex == capacity)
    {
        int *newdata = new Int[2 * capacity];
        for(int i=0; i<capacity; i++)
        {
            newdata[i] = data[i];
        }
        delete [] data;
        data = newdata;
    }
    data[nextIndex] = element;
    nextIndex++;
}
```

Template

Create pair class with two data members.

```
class pair {
    int x;
    int y;

    Public:
        void setX(int x)
            {
                this -> x = x;
            }
        int getX()
            { return x;
            }
        void setY(int y)
            {
                this -> y = y;
            }
        int getY()
            {
                return y;
            }
    };
```

When different datatype

```
templete <typename T>
class Pair {
    T x;
    T y;
public:
    void setX(T x)
    {
        this -> x = x;
    }
    T getx()
    {
        return x;
    }
    void setY(T y)
    {
        This -> y = y;
    }
    T getY()
    {
        return y
    }
};
```

```
int main()
{
    Pair<int> P1;
    Pair<double> P2;
    P1.setx(10)
    P1.sety(20);
    cout << P1.getx() << " " << P2.getY()
```

P2. P2.

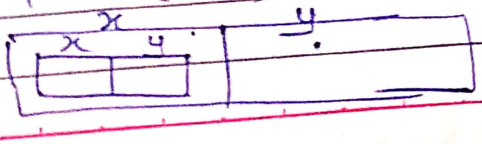
multible varible temp

```
templete <typname T,
          Typename V>
void setY(V Y)
{
}
V getY()
{
    return y
}
```

```
int main()
{
    Pair<int, double> P1.
```

More than two varible

Pair < Pair <int, int>, int>.



```
int main(){
    pair<pair<int,int>,int>p2;
    p2.setY(50);
    pair<int,int> p4;
    p4.setx(15);
    p4.setY(10);
    p2.setx(p4);
    cout<<p2.getx().getx()<<p2.getx().gety()
        <<p2.gety()<<endl;
}
```

Stack using linked list

```
Template<typename T>
Class Node{
    Public:
    T data;                          // int data;
    Node *next;                      // Node<T> *next;
    Node(int data){                  // Node(T data){
        this->data = data;
        next = NULL;
    }
};
Template<typename T>
Class stack{
    Node *head;                      // Node<T> *head;
    T size;                          // int size;
```

```
public:
Stack () {
        head = Null;
        size = 0;
        }

Int get size () {
        return size;
        }

bool isEmpty () {
        Return size == 0;
        }

Void Push (T element)
  {
    Node<T> *newnode = new Node<T>(elem);
    Newnode → next = head;
    head = newnode;
     size ++;
  }

T pop () {  if (!isEmpty())
              { return 0;
              }
    T ans = head → data;
    delete head;        Node<T> *temp = head;
                        head = head → next;
                        delete temp;
                        size --;
                        Return ans;
    }

T Top()
  { if (isEmpty())
      {return 0;
      }
    return head;
```

Inbuilt stack

```
#include<stack>

int main()
{
    stack<int> s;
    s.push(10)
    ——— ;——— (20)
    ——— ;——— (30)

    cout << s.top() << endl;
    cout << s.push();
    cout << s.size();
    cout << s.empty();
```

(fifo) Queue (Abstract d.T)

In How order the element same.

insert → enqueue.(10)

Access → front()
the first

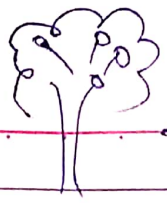
Delete → dequeue()

Size → size()

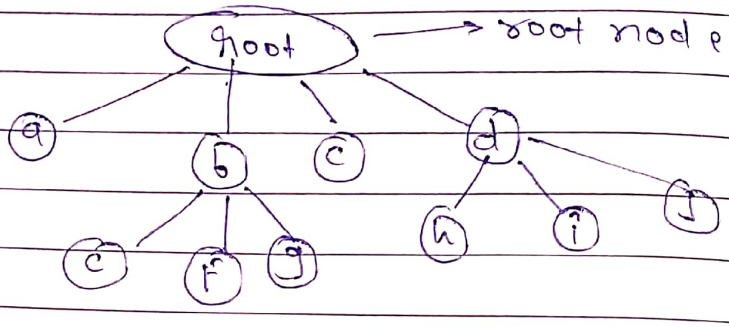
IsEmpty() →

Creating Queue

```
Template <Typename T>

T *data,
int nextindex;
int firstindex;
int size;

Public:
  Queue Using Array (int s)
      {
        data = new T[s];
        nextindex = 0;
        firstindex = -1;
        size = 0;
      }

int getsize ()
      {
        return size;
      }

bool is empty ()
      {
        Return size == 0;
      }

void enqueue (T element)
  {
    data[nextindex] = element;
    nextindex++;
  }
```



Trees



Root → a, b, c, d

a → no one

b → e, f, g.

Terms
1. Root
2. Parent - child
3. leaf (A node with no child)

Tree Node classes

```
#include<vector>
Using namespace std;

template<typename T>
class TreeNode
    {
        Public:
            T data;
            Vetor<TreeNode<T>*> children;
```

```
#include<iostream>
#include "TreeNode.h"
using namespace std

template

int main() {

    TreeNode<int>* root = new TreeNode<int>(1);
    ------------> node1 = new ------- (2);
    ------------> node2 = new ------- (3);

    root -> children.Push_back(node1);
    root -> children.Push_back(node2);


Void PrintTree(TreeNode<int>* root)
    {
        cout << root -> data << endl;

    for(int i = 0; i < root -> children.Size(); i++)
        {
            PrintTree(root -> children[i]);
        }
}
```

By Recursion.

```
void PrintTree(TreeNode<int>* root)
{
    cout << root->data << endl;
    for(int i=0; i<root->children.size(); i++)
    {
        PrintTree(root->children[i]);
    }
}

void PrintTree(TreeNode<int>* root)
{
    if(root == NULL)
    {
        return;
    }
    cout << root->data << " ";
    for(int i=0; i<root->children.size(); i++)
    {
        cout << root->children[i]->data << " ";
    }
}

TreeNode<int>* takeInput()
{
    int rootData;
    cin >> rootData;
    TreeNode<int>* root = new TreeNode<int>(rootData);
```

```
int n;
cout << rootdata << endl;
cin >> n;
for(int i=0; i<n; i++)
{
    TreeNode<int>* children = takeInput();
    root -> children.push_back(child);
}
return root;
}
```

:- Using queue :-

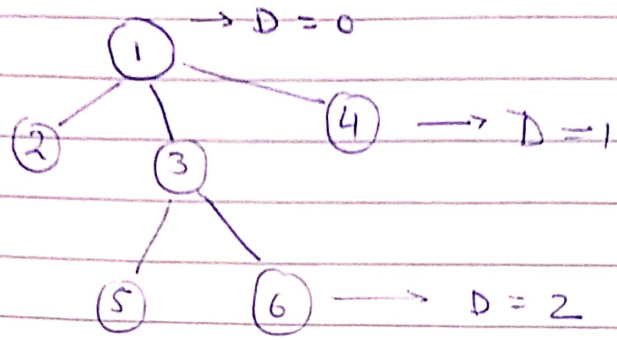
```
TreeNode<int>* takeInputLevelWise()
{
    int rootdata;
    cout<<"Enter rootdata" <<endl;
    cin >> rootdata;

    TreeNode<int>* root = new TreeNode<int>
                          (rootdata);

    queue<TreeNode<int>*> PendingNodes;

    PendingNodes.push(root);
    while(PendingNode.size() != 0)
    {
        TreeNode<int>* front = PendingNote.front();
        PendingNodes.pop();
        cout<<"Enter no of children"<<front->data<<
                                        endl;
        int numchild;
```

```
for (int i = 0; i < numChild; i++)
{
    int childdata;
    cout << "Enter " << i << "th child of " << front->data << endl;
    cin >> childdata;
    TreeNode<int>* child = new TreeNode<int>(childdata);
    front->children.push_back(child);
    pendingNodes.push(child);
}
}
}

int main() {
    TreeNode<int>* root = takeInputLevelWise();
    takeInput();
    printTree(root);
}
```

or

```
int numNodes(TreeNode<int>* root)
{
    int ans = 1;
    for(int i = 0; i < root->children.size(); i++)
    {
        ans += numNodes(root->children[i]);
    }
    return ans;
}
```

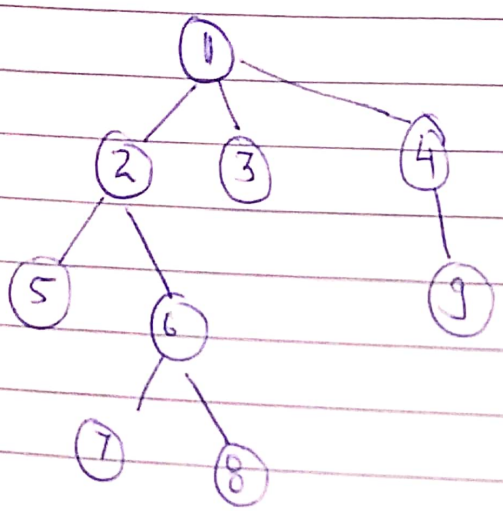
Depth of Node



→ Task Print all nodes at depth d

```
Void PrintAtLevelK( TreeNode<int>* root, int k)
{   if(k==NULL) Return;
    if(k==0)
    {
        cout<<root->data<<endl;
        return;
    }
for(int i=0; i<root->children.size(); i++)
    {
        PrintAtLevel(root->children[i], k-1);
    }
}
```

Print Preorder Tree




```
Void Preorder (TreeNode<int>* root)
{
    Cout<<root->data<<" ";
    for(int i=0; i<root->children.size(); i++)
    {
        preorder (root->children[i]);
    }
}
```

Camlin Page
Date / /

How To delete tree ?

```
Void deleteTreeNode ( TreeNode<int>* root)
{
    for(int i=0; i<root->children.size(); i++)
    {
        delete tree (root->children[i]);
    }
    delete root;
}
```

Dynamic Programming

Better form for D.P.

→ Solved by Memo

```
int fibo_Better(int n), int *ans)
    {
        if(n<=1)
                { ans[n] = n;
                  return;
                }
        int *ans = new int[n+1];

        if (ans[n-1] == 0)
                {
          ans[n-1] = fibo_Better(n-1, ans);
                }

        if(ans[n-2] == 0)
                {
                    ans[n-2] = fibo_Better(n-2, ans);
                }
        ans[n] = ans[n-1] + ans[n-2];
        return ans[n];
    }

    int fibo_Better(int n)
    {
        int *ans = new int[n+1];
        for(int i=0; i<=n; i++)
            {
                a[i] = 0;
            }
    } return fibo_Better(n, ans);
```

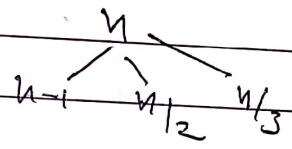
Memorization V/s dynamic

↳ Top down approach

Bottom up approach

```
int fibo_DP(int n)
{
    int * ans = new int[n+1];
    ans[0] = 0;
    ans[1] = 1;
    for(int i = 2; i<=n; i++)
    {
        ans[i] = ans[i-1] + ans[i-2];
    }
    return ans[n];
}
```

Least step to find min of number.




```
int minsteps(int n).
{ // Base case
    if(n==1)
    return 0;
}

int x = minstep(n-1);
int y = Int_Max, z = INT_Max;

if(n%2==0)
{   y = minstep(n/2);
}
if(n%3==0)
{   z = minstep(n/3);
```

```
return min(x, min(y,z)) +1;
}

int main()
{
    int n;
    cin >> n;
    cout << minStep(n) << endl;
}
```

Memorisation

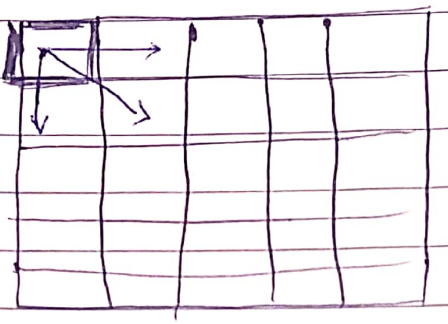
```
// ans --> -1
int minSteps_Better(int n, int *ans)
{
    if(n == 1)
    {
        ans[n] = 0;
        return 0;
    }
    if(ans[n-1] == -1)
    {
        ans[n-1] = minstep_Better(n-1, ans);
    }
    int y = INT_MAX, z = INT_MAX;
    if(n%2 == 0)
    {
        if(ans[n/2] == -1)
        {
            ans[n/2] = minSteps_Better(n/2, ans
        }
        y = ans[n/2];
    }
```

```
if (n%3 == 0)
    {
      if (ans[n/3] == -1)
          {
            ans[n/3] = minStep_Better (n/3, ans);
          }
      x = ans[n/3];
    }
ans[n] = 1 + min ( ans[n-1], min (y, z));
return ans[n];

}

int minSteps_Better (int n)
    {
      int * ans = new int[n+1];
      for (int i = 0; i <= n; i++)
      {
          a[i] = -1;
      }
      return minstep_Better (n, ans);
```

Min Cost Path

A → m x n

(0,0) → (m-1, n-1)



Using Brute force

```
int minCostPath(int **input, int m, int n, int i, int j)
{
    // Base case
    if(i == m-1 && j == n-1)
    {
        return input[i][j];
    }
    if(i >= m || j >= n)
    {
        return INT_MAX;
    }
    // Recursive calls
    int x = minCostPath(input, m, n, i, j+1);
    int y = minCostPath(input, m, n, i+1, j+1);
    int z = minCostPath(input, m, n, i+1, j);

    // Small calculation
    int ans = min(x, min(y, z)) + input[i][j];
    return ans;
}

int minCostPath(int *input, int m, int n)
{
    return minCostPath(input, m, n, 0, 0);
}
```

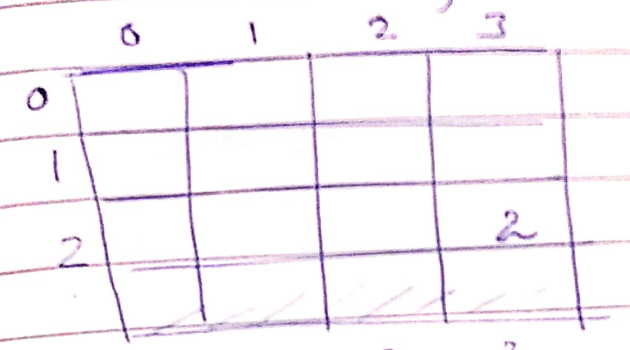
```
int main()
{ int m, n;
cin >> m >> n;
int **input = new int *[m];
for(int i = 0; i<m; ; i++)
        {
          input[i] = new int [n],
          for(int j=0; j<n ; j++)
            {
              cin >> input[i][j];
            }
        }
        cout << minCostPath(input, m, n) << endl;

}
```

Using Memoris.

```
int minCostPath_Better(int **input, int m, int n, int i,
                       int j, int **ans)

if(i==m-1 && j==n-1)
  {
    return input[i][j];
  }
if(i>=m || j>=n)
    {
      return INT_MAX;
    }
if(ans[i][j] != -1)
    {
      return ans[i][j];
    }
```

```
int x = minCostPath_Better(input, m, n, i+1, j, dp);
if(x < Int_Max)
        {
          ans[i+1][j] = x;
        }

Int y = minCostPath_Better(input, m, n, i+1, j+1, dp);
if(y < Int_Max)
        {
          ans[i+1][j+1] = y;
        }

Int z = minCostPath_Better(input, m, n, i, j+1);
if(z < Int_Max)
        {
          ans[i][j+1] = z;
        }

ans[i][j] = min(x, min(y, z)) + input[i][j];
return ans[i][j];
}
```

Bottom up Approach. (Using dp)



	0	1	2	3
0	1	5	2	0
1	8	3	1	4
2	3	7	6	2

DP

```
int minCostPath_Better(int **input, int m, int n)
{
    int ** ans = new int*[m];

    for(int i=0 ; i<m ;i++)
    {
      ans[i] = new int[n];
    }

ans[m-1][n-1] = input[m-1][n-1];

// Last row
for(int j = n-2 ; j>=0; j--)
  {
  ans[m-1][j] = input[m-1][j] + ans[m-1][j+1];

// Last col
for(int i = m-2; i>=0; i--)
      { ans[i][n-1] = input[i][n-1] + ans[i+1][n-1];
      }
```

```
for(int i = m-2 , i>=0; i--)
        {
            for(int j = n-2 , j>=0 ; j--)
        {
        a[i][j] = input[i][j] + min(ans[i][j+1],
                              min(ans[i+1][j+1], ans[i+1][j]))
        }
    }
    return ans[0][0],
}
```

Q Watsy CP2 (0/1 knapsack)

Coin change Problem

```
Public:
        long long int dp[1000][1000],
        long long int solve (int a[], int m, int n)
        {
            if(m == -1 && n>0)
                    return 0,
            if(n == 0)
                    return 1;
            if(n<0)
                    return 0;
            if(dp[m][n] != -1) return dp[m][n];

    return dp[m][n] = solve(a, m, n-a[m])
                    + solve(a, m-1, n);
        }
```

```
long long int count(int a[], int m, int n)
{
    dp[m][n+1];
    for(int i=0; i<m; i++)
    {
        for(int j=0; j<=n; j++)
        {
            dp[i][j] = -1;
        }
    }
    return solve(a, m-1, n);
}
};
```