LEARN DSA WITH C++

WEEK :: 16

[Last Note]

LEARN DSA WITH C++

PDF

COURSE INSTRUCTOR BY
ROHIT NEGI

MADE BY-
PRADUM SINGHA

Check Profile

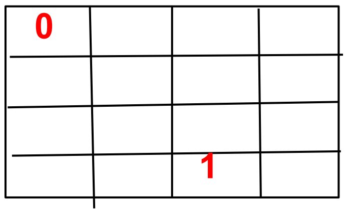
My profile

LEARN DSA WITH C++

WEEK :: 16 **DAY: 01** **DATE: 01-08-2023**

Grid + Stock

Number of Unique Paths << GeeksforGeeks >>



Reach 0 to 1 : how much way

```cpp
class Solution
{
    public:
    //Function to find total number of unique paths.
    int NumberOfPath(int a, int b)
    {
        //code here
        vector<vector<int>>dp(a,vector<int>(b,1));

        for(int i=a-2; i>=0; i--)
        for(int j=b-2; j>=0; j--)
        dp[i][j] = dp[i+1][j]+dp[i][j+1];

         return dp[0][0];
    }
};
```

Optimization code

```cpp
class Solution
{
    public:
    //Function to find total number of unique paths.
    int NumberOfPath(int a, int b)
    {
        //code here
        vector<int>dp(b,1);

        for(int i=a-2; i>=0; i--)
        for(int j=b-2; j>=0; j--)
        dp[j] += dp[j+1];

        return dp[0];
    }
};
```

Unique Paths in a Grid << GeeksforGeeks >>

1	1	1	1
1	0	1	1
1	1	0	1
1	1	1	1

Reach 1 to another 1 : find path
Can't travel 0 block

```
class Solution {
  public:
    int uniquePaths(int n, int m, vector<vector<int>> &grid) {
        // code here
        if(grid[0][0]==0 || grid[n-1][m-1] ==0)
        return 0;

        vector<vector<int>>dp(n,vector<int>(m,0));

        // last row
        for(int j=m-2; j>=0; j--)
        {
            if(grid[n-1][j]==1)
            dp[n-1][j]=1;
            else
            break;
        }
        // last col
        for(int i=n-2; i>=0; i--)
        {
            if(grid[i][m-1]==1)
            dp[i][m-1]=1;
            else
            break;
        }

        for(int i=n-2; i>=0; i--)
        for(int j=m-2; j>=0; j--)
        {
            if(grid[i][j])
            dp[i][j]=(dp[i+1][j]+dp[i][j+1])%1000000007;
        }
        return dp[0][0];
    }
};
```

Minimum Path Sum << LeetCode >>

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        int n=grid.size(), m= grid[0].size();
        // last row
        for(int j=m-2; j>=0; j--)
        grid[n-1][j] += grid[n-1][j+1];

        // last col
        for(int i=n-2; i>=0; i--)
        grid[i][m-1] += grid[i+1][m-1];

        for(int i=n-2; i>=0; i--)
        for(int j=m-2; j>=0; j--)
        grid[i][j] += min(grid[i+1][j] , grid[i][j+1]);

        return grid[0][0];
    }
};
```

Best Time to Buy and Sell Stock << LeetCode >>

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int profit = 0, n = prices.size();
        int stock = prices[0];

        for(int i=1; i<n; i++)
        {
            profit = max(profit, prices[i] - stock);
            stock = min(stock, prices[i]);
        }
        return profit;
    }
};
```

Best Time to Buy and Sell Stock II << LeetCode >>

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int profit =0, n= prices.size();

        for(int i=1; i<n; i++)
        {
            if(prices[i]>prices[i-1])
            profit += prices[i] - prices[i-1];
        }
        return profit;
    }
};
```

LEARN DSA WITH C++

WEEK :: 16 **DAY: 02** **DATE: 01-08-2023**

Stock+How to define row and column in dp

Subset Sum Problem << [GeeksforGeeks](#) >>

// Recursion

```
class Solution{
public:

bool find(int n, int sum, vector<int> &arr)
{
   // Base condition
   if(sum ==0)
   return 1;

   if(n==0)
   return 0;

   if(sum < arr[n-1])
   return find(n-1, sum, arr);

   else
   return find(n-1, sum-arr[n-1], arr) || find(n-1, sum, arr);
}

   bool isSubsetSum(vector<int>arr, int sum){
      // code here
      int n= arr.size();
      return find(n, sum, arr);
   }
};
```

Top Down Approach

```
class Solution{
public:

bool find(int n, int sum, vector<int> &arr, vector<vector<int>> &dp)
{
    // Base condition
    if(sum ==0)
    return 1;

    if(n==0)
    return 0;

    if(dp[n][sum] != -1)
    return dp[n][sum];

    if(sum < arr[n-1])
    return dp[n][sum] = find(n-1, sum, arr, dp);
    else
    return dp[n][sum] = find(n-1, sum-arr[n-1], arr, dp) || find(n-1, sum, arr, dp);
}

    bool isSubsetSum(vector<int>arr, int sum){
        // code here
        int n= arr.size();
        vector<vector<int>>dp(n+1, vector<int>(sum+1, -1));
        return find(n, sum, arr, dp);
    }
};
```

Bottom Up Approach

```
class Solution{
public:

    bool isSubsetSum(vector<int>arr, int sum){
        // code here
        int n= arr.size();
        vector<vector<int>>dp(n+1, vector<int>(sum+1, 0));

        // Initialize the dp
        for(int j=0; j<=sum; j++)
        dp[0][j]=0;

        for(int i=0; i<=n; i++)
        dp[i][0]=1;

        for(int i=1; i<=n; i++)
        for(int j=1; j<=sum; j++)
        {
            if(j<arr[i-1])
            dp[i][j]= dp[i-1][j];
            else
            dp[i][j] = dp[i-1][j-arr[i-1]] || dp[i-1][j];
        }
        return dp[n][sum];
    }
};
```

Another Approach

```
                                // Recursion
class Solution{
public:
    bool find(int index, int sum, const std::vector<int> &arr, int n) {
        if (sum == 0)
            return true;

        if (index == n)
            return false;

        if (sum < arr[index])
            return find(index + 1, sum, arr, n);

        return find(index + 1, sum - arr[index], arr, n) || find(index + 1, sum,
arr, n);
    }
    bool isSubsetSum(std::vector<int> arr, int sum) {
        int n = arr.size();
        return find(0, sum, arr, n);
    }
};
```

Top Down Approach

```
class Solution{
public:
    bool find(int index, int sum,vector<int> &arr, int n,vector<vector<int>> &dp) {
        if (sum == 0)
            return true;

        if (index == n)
            return false;

        if(dp[index][sum] != -1)
        return dp[index][sum];

        if (sum < arr[index])
            return dp[index][sum]= find(index + 1, sum, arr, n, dp);

        return dp[index][sum]= find(index + 1, sum - arr[index], arr, n, dp) ||
find(index + 1, sum, arr, n, dp);
    }

    bool isSubsetSum(std::vector<int> arr, int sum) {
        int n = arr.size();
        vector<vector<int>>dp(n+1, vector<int>(sum+1, -1));
        return find(0, sum, arr, n,dp);
    }
};
```

Bottom Up Approach

```
class Solution{
public:

    bool isSubsetSum(std::vector<int> arr, int sum) {

        int n = arr.size();
        vector<vector<int>>dp(n+1, vector<int>(sum+1, 0));

        for(int j=0; j<=sum; j++)
        dp[n][j]=0;

        for(int i=0; i<=n; i++)
        dp[i][0]=1;

        for(int i=n-1; i>=0; i--)    // n row
        for(int j=1; j<=sum; j++)    // sum col
        {
            if(j<arr[i])
            dp[i][j] = dp[i+1][j];
            else
            dp[i][j]= dp[i+1][j-arr[i]] || dp[i+1][j];
        }
        return dp[0][sum];
    }
};
```

Best Time to Buy and Sell Stock III **<< LeetCode >>**

// Recursion

```
class Solution {
public:

    int find(int day, int trans, int buy, int n, vector<int>&prices)
    {
        if(trans==0 || day==n)
        return 0;

        if(buy)
        {
            return max(-prices[day] + find(day+1, trans, 0,n,prices),
            find(day+1, trans,1,n,prices));
        }
        else
        {
            return max(prices[day] + find(day+1, trans-1,1,n,prices),
            find(day+1, trans,0,n,prices));
        }
    }
    int maxProfit(vector<int>& prices)
    {
        int n= prices.size();
        return find(0,2,1,n, prices);
    }
};
```

DP

```
class Solution {
public:

    int find(int day, int trans, int buy, int n, vector<int>&prices,
vector<vector<vector<int>>> &dp)
    {
        if(trans==0 || day==n)
        return 0;

        if(dp[day][trans][buy] != -1)
        return dp[day][trans][buy];

        if(buy)
        {
            return dp[day][trans][buy] = max(-prices[day] + find(day+1, trans,
0,n,prices,dp),
            find(day+1, trans,1,n,prices,dp));
        }
        else
        {
            return dp[day][trans][buy] = max(prices[day] + find(day+1,
trans-1,1,n,prices,dp),
            find(day+1, trans,0,n,prices,dp));
        }
    }
    int maxProfit(vector<int>& prices)
    {
        int n= prices.size();
        vector<vector<vector<int>>>dp(n+1, vector<vector<int>>(3, vector<int>(2,-1)));
        return find(0,2,1,n, prices, dp);
    }
};
```

LEARN DSA WITH C++

WEEK :: 16 **DAY: 02** **DATE: 01-08-2023**

Stock+How to define row and column in dp

Best Time to Buy and Sell Stock III << LeetCode >>

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n=prices.size();
        vector<vector<int>>dp(3,vector<int>(n+1,0));

        int total;

        for(int i=1; i<=2; i++)
        {
            total = -prices[0];
            for(int j=2; j<=n; j++)
            {
                dp[i][j]=max(dp[i][j-1], prices[j-1] + total);
                total = max(total, -prices[j-1]+dp[i-1][j]);
            }
        };
        return dp[2][n];
    }
};
```

Player with max score :- << GeeksforGeeks >> {Little bit error find it }

```
class Solution{
public:
    pair<int, int> find(int start, int end, int player, int arr[]) {
        if(start == end) {
            if(player == 0)
                return {arr[start], 0};
            else
                return {0, arr[start]};
        }
        pair<int, int> score1, score2;

        if(player == 0) {
            score1 = find(start+1, end, 1, arr);
            score2 = find(start, end-1, 1, arr);
            score1.first += arr[start];
            score2.first += arr[end];
            if(score1.first > score2.first)
                return score1;
            else
                return score2;
        }
        else {
            score1 = find(start+1, end, 0, arr);
            score2 = find(start, end-1, 0, arr);
            score1.second += arr[start];
            score2.second += arr[end];
            if(score1.second > score2.second)
                return score1;
            else
                return score2;
        }
    }

    bool is1winner(int N, int arr[]) {
        pair<int, int> result = find(0, N-1, 0, arr);
        return result.first > result.second;
    }
};
```

Egg Dropping Puzzle :- << GeeksforGeeks >>

```
                    // Recursion

class Solution
{
  public:
  //Function to find minimum number of attempts needed in
  //order to find the critical floor.
  int find(int eggs, int floors)
  {
    if(eggs==1)
    return floors;

    if(floors==0)
    return 0;

    int ans = INT_MAX;
    int temp;
    for(int i=1; i<=floors; i++)
    {
      // Break or not break
      temp = 1+max(find(eggs-1, i-1), find(eggs, floors-i));
      ans = min(ans, temp);
    }
    return ans;
  }


  int eggDrop(int n, int k)
  {
    // your code here
    return find(n,k);
  }
};
```

```
                              // Dp

class Solution
{
  public:
  //Function to find minimum number of attempts needed in
  //order to find the critical floor.
  int find(int eggs, int floors, vector<vector<int>> &dp)
  {
    if(eggs==1)
    return floors;
```

```
      if(floors==0)
      return 0;

      if(dp[eggs][floors] != -1)
      return dp[eggs][floors];

      int ans = INT_MAX;
      int temp;
      for(int i=1; i<=floors; i++)
      {
         // Break or not break
         temp = 1+max(find(eggs-1, i-1, dp), find(eggs, floors-i, dp));
         ans = min(ans, temp);
      }
      return dp[eggs][floors];
   }


   int eggDrop(int n, int k)
   {
      // your code here
      vector<vector<int>>dp(n+1, vector<int>(k+1, -1));
      return find(n,k,dp);
   }
};
```

// **Final Optimization**

```
class Solution
{
    public:
    //Function to find minimum number of attempts needed in
    //order to find the critical floor.

    int eggDrop(int n, int k)
    {
        // your code here
        vector<vector<int>>dp(n+1, vector<int>(k+1, 0));

        for(int j=0; j<=k; j++)
        dp[1][j]=j;

        for(int i=2; i<=n; i++)
        for(int j=1; j<=k; j++)
        {
            int ans= INT_MAX;
            int temp;
            for(int a=1; a<=j; a++)
            {
                temp = 1+max(dp[i-1][a-1], dp[i][j-a]);
                ans = min(ans,temp);
            }
            dp[i][j] = ans;
        }
        return dp[n][k];

    }
};
```

Longest Increasing Path in a Matrix << GeeksforGeeks >>

```
class Solution {
  public:

    int row[4]={-1,1,0,0};
    int col[4]={0,0,-1,1};

    bool check(int i, int j, int n, int m)
    {
        return i>-1 && i<n && j>-1 && j<m;
    }

    void DFS(int i, int j, vector<vector<int>> &matrix, vector<vector<int>> &path,
int n, int m)
    {
        path[i][j]=1;

        for(int k=0; k<4; k++)
        {
            if(check(i+row[k], j+col[k],n,m) &&
matrix[i][j]<matrix[i+row[k]][j+col[k]])
            {
                if(path[i+row[k]][j+col[k]]==0)
                DFS(i+row[k], j+col[k], matrix, path,n,m);
                path[i][j] = max(path[i][j], 1+path[i+row[k]][j+col[k]]);
            }
        }
    }
    int longestIncreasingPath(vector<vector<int>>& matrix, int n, int m) {

        // Code here
        vector<vector<int>>path(n,vector<int>(m,0));
        int total =0;
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<m; j++)
            {
                if(path[i][j]==0)
                DFS(i, j, matrix, path, n, m);
                total = max(total, path[i][j]);
            }
        }
        return total;

    }
};
```

LEARN DSA WITH C++

WEEK :: 16 **DAY: 00** **DATE: 12-09-2023**

Trie Datastructer

Trie (Insert & Search)

```cpp
#include<iostream>
using namespace std;

class TrieNode
    {
        public:
        TrieNode *child[26];
        bool isEndofword;

        TrieNode()
        {
            isEndofword = false;
            for(int i=0; i<26; i++)
            child[i]=NULL;
        }
    };

    class Trie
    {
        TrieNode *root;
        public:
        Trie()
        {
            root = new TrieNode();
        };
        // insert
        void insert(string word)
        {
            TrieNode *node = root;
            for(char c: word)
            {
                int index = c-'a';
                if(node ->child[index] == NULL)
                {
                    node->child[index] = new TrieNode();
                    node = node->child[index];
                }
                else
                {
                    node = node->child[index];
                }
            }
            node -> isEndofword= true;
        }
        // search
        bool search (string word)
        {
            TrieNode *node = root;
            for(char c: word)
            {
                int index = c-'a';
                if(node->child[index] == NULL)
                return 0;
                node = node -> child[index];
            }
            return node -> isEndofword;
        }

        bool isEmpty(TrieNode *node)
        {
            for(int i=0; i<26; i++)
            {
                if(node->child[i])
                return false;
            }
            return true;
        }

        // Deletion
        bool Delete(TrieNode *node, string word, int depth)
        {
            // Base condition
            if(depth==word.size())
            {
                // It is not the end of word
```

```
                if(node->isEndofword==0)
                {
                    return false;
                };
                // It is the end of word
                node ->isEndofword=0;
                // child exist or not
                return isEmpty(node);
            }

            int index = word[depth]-'a';
            // char doesn't exit
            if(node->child[index] == NULL)
            return false;
            // char exist

            // Recursive call to delete the char int tree
            bool ShouldDeleteChild = Delete(node ->child[index], word,depth+1);

            if(ShouldDeleteChild)
            {
                delete node->child[index];
                node->child[index]=NULL;

                return !node->isEndofword && isEmpty(node);
            };

            return false;
        }

        void Deleteword(string word)
        {
            Delete(root, word, 0);
        }
    };
int main()
{
    Trie *tree = new Trie();
    tree-> insert("apple");
    tree-> insert("appex");
    tree->insert("almond");
    tree->Deleteword("apple");
    cout<<"Does it exist "<<tree->search("appex")<<endl;


return 0;
};
```

Trie | (Insert and Search) << [GeeksforGeeks](https://www.geeksforgeeks.org) >>

```
void insert(struct TrieNode *root, string key)
{
    // code here
    TrieNode *node = root;

    for(char c: key)
    {
        int index = c-'a';
        if(node->children[index]==NULL)
        {
            node->children[index] = new TrieNode();
        }
        node = node->children[index];
    };
        node->isLeaf = 1;
}

//Function to use TRIE data structure and search the given string.
bool search(struct TrieNode *root, string key)
{
    // code here
    TrieNode *node = root;
    for(char c: key)
    {
        int index = c-'a';
        if(node->children[index] == NULL)
        return 0;

        node = node->children[index];
    };
    return node->isLeaf;
}
```

Phone directory << [GeeksforGeeks](https://www.geeksforgeeks.org) >>

```
class Solution{
public:

    class TrieNode{
        public:
        bool isEndofword;
        TrieNode *child[26];

        TrieNode()
        {
            isEndofword=0;
            for(int i=0; i<26; i++)
            child[i]=NULL;
        }
    };
    class Trie{
        TrieNode *root;
        public:

        Trie()
        {
            root = new TrieNode();
        };

        void insert(string word)
        {
            TrieNode *node = root;
            for(char c: word)
            {
                int index = c-'a';
                if(node->child[index]==NULL)
                {
                    node ->child[index] = new TrieNode();
                    node = node ->child[index];
                }
                else
                {
                    node = node->child[index];
                }
            }
            node -> isEndofword=true;
        }

        void findContact(string prefix, TrieNode *node, vector<string> &contact)
        {
            if(node->isEndofword)
            contact.push_back(prefix);

            for(char c='a'; c<='z'; c++)
            {
                int index = c-'a';
                if(node->child[index])
                findContact(prefix+c, node->child[index], contact);
            }
        }

        vector<string> searchContact(string prefix)
        {
            TrieNode *node = root;

            for(char c: prefix)
            {
                int index = c-'a';
                // prefix doesn't exit
                if(node->child[index]==NULL)
                return {"0"};
                node = node->child[index];
            }
            vector<string>contact;
            findContact(prefix,node,contact);
            return contact;
        }
    };

    vector<vector<string>> displayContacts(int n, string contact[], string s)
    {
        // code here
        Trie tree;
        for(int i=0; i<n; i++)
        tree.insert(contact[i]);

        vector<vector<string>> result;
        string prefix = "";

        for(int i=0; i<s.size(); i++)
        {
            prefix+=s[i];
            vector<string>contact = tree.searchContact(prefix);

            result.push_back(contact);
        }
        return result;

    }
};
```

Segment Tree

Segment Tree :-

```
#include <iostream>
#include <vector>
using namespace std;

class SegmentTree
{
    vector<int> tree;
    vector<int> arr;
    int n;

public:
    SegmentTree(vector<int> input)
    {
        n = input.size();
        arr = input;
        tree.resize(4 * n);
        build(0, 0, n - 1);
    }
    void build(int node, int start, int end)
    {
        if (start == end)
        {
            tree[node] = arr[start];
            return;
        }
        int mid = start + (end - start) / 2;
        build(2 * node + 1, start, mid);
        build(2 * node + 2, mid + 1, end);
        tree[node] = tree[2 * node + 1] + tree[2 * node + 2];
    }

    int range(int node, int start, int end, int left, int right)
    {
        if (end < left || start > right)
            return 0;
        if (start >= left && end <= right)
            return tree[node];
        int mid = start + (end - start) / 2;
        return range(2 * node + 1, start, mid, left, right) + range(2 * node +
2, mid + 1, end, left, right);
    }

    void updating(int node, int start, int end, int idx, int val)
    {
        // Base condition
        if(idx>end || idx<start)
        return;

        if(start==end)
        {
            tree[node]=val;
            return;
        }

        tree[node]+= val-arr[idx];
        int mid = start+(end-start)/2;
        //Left child
        updating(2*node+1,start,mid,idx,val);
        //right child
        updating(2*node+2, mid+1, end, idx, val);
    }


    int query(int left, int right)
    {
        return range(0, 0, n - 1, left, right);
    }

    void update(int idx, int val)
    {
        return updating(0,0,n-1, idx, val);
        arr[idx]=val;
    }
};

int main()
{
    vector<int> arr;
    arr.push_back(3);
    arr.push_back(5);
    arr.push_back(23);
    arr.push_back(31);
    arr.push_back(23);
    arr.push_back(33);

    SegmentTree *Tree = new SegmentTree(arr);
    Tree->update(3,10);
    cout << Tree->query(1, 5) << endl;

    delete Tree; //  free the memory
    return 0;
}
```

Segment Tree find Max:-

```
#include <iostream>
#include <vector>
#include <climits> // Include for INT_MIN
using namespace std;

class SegmentTree
{
    vector<int> tree;
    vector<int> arr;
    int n;

public:
    SegmentTree(vector<int> input)
    {
        n = input.size();
        arr = input;
        tree.resize(4 * n);
        build(0, 0, n - 1);
    }

    void build(int node, int start, int end)
    {
        if (start == end)
        {
            tree[node] = arr[start];
            return;
        }

        int mid = start + (end - start) / 2;
        //left child
        build(2 * node + 1, start, mid);
        //right child
        build(2 * node + 2, mid + 1, end);
        // Sum of my left and right child
        tree[node] = max(tree[2 * node + 1], tree[2 * node + 2]);
    }

    int range(int node, int start, int end, int left, int right)
    {
          if (end < left || start > right)
            return 0; // Change this from INT_MIN to 0
        if (start >= left && end <= right)
            return tree[node];
        int mid = start + (end - start) / 2;
        return max(range(2 * node + 1, start, mid, left, right), range(2 * node + 2, mid + 1, end, left, right));
    }

    void updating(int node, int start, int end, int idx, int val)
    {
        // Base condition
        if (idx > end || idx < start)
            return;
        if (start == end)
        {
            tree[node] = val;
            return;
        }

        int mid = start + (end - start) / 2;
        //Left child
        updating(2 * node + 1, start, mid, idx, val);
        //right child
        updating(2 * node + 2, mid + 1, end, idx, val);

        tree[node] = max(tree[2*node+1],tree[2*node+2]);
    }

    int query(int left, int right)
    {
        return range(0, 0, n - 1, left, right);
    }
    void update(int idx, int val)
    {
        return updating(0, 0, n - 1, idx, val);
        arr[idx]=val;
    }
};

int main()
{
    vector<int> arr;
    arr.push_back(3);
    arr.push_back(5);
    arr.push_back(23);
    arr.push_back(31);
    arr.push_back(23);
    arr.push_back(33);

    SegmentTree *Tree = new SegmentTree(arr);
    Tree->update(3, 10);
    cout << Tree->query(3, 5) << endl;

    delete Tree; // Free the memory
    return 0;
}
```