LEARN DSA WITH C++

WEEK :: 12

LEARN DSA WITH C++

PDF

COURSE INSTRUCTOR BY
ROHIT NEGI

MADE BY-
PRADUM SINGHA

Check Profile

My profile

LEARN DSA WITH C++

WEEK :: 12 DAY: 01 DATE: 10-07-2023

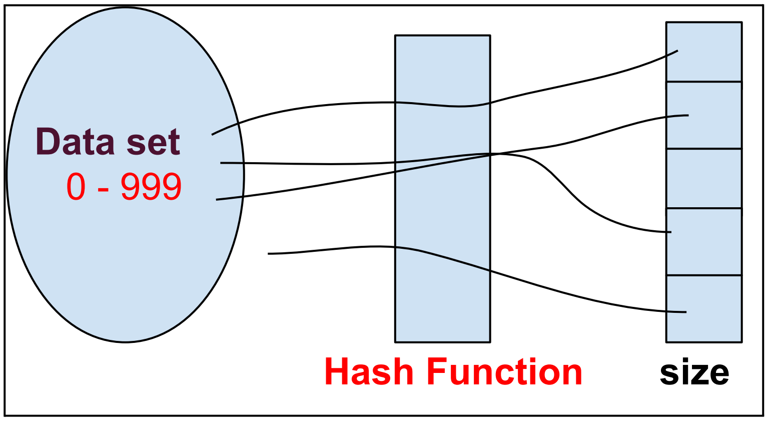
HASHING BASIC

Search :-	
Algorithm	Time Complexity

Array Sorted -> Binary Tree
Unsorted -> Linear Search
	O(log n)
	O(n)

Linked List :	O(N)
Binary Tree :	O(n)
Binary Search Tree :	O(n)
B. Balance S. Tree	O(n)
Priority_Queue max/min heap	O(n)
Stack :	O(n)
Queue :	O(n)

But Hashing : Search -> O(1)



Hashing

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function.

Hash Function : converts a given numeric or alphanumeric key to a small practical integer value.

How to Insert value in array size:-

keys = { 13, 128, 275, 991, 334 };
Every element % 10

	0
991	1
	2
13	3
334	4
275	5

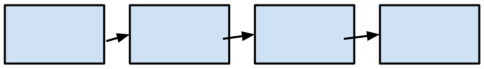
Find element = 991/10 = 1 find index 1

Collision :-

13%10 = 3; 223%10 = 3; 23%10 = 3;
When multiple elements allocate the same index.

To reduce :- Using better hash function.

Channing :-

Insert the value in the same location.  Create Link List

Time Complexity :-
Insert :- O(1)
Search :- O(n)
Delete :- O(n)

Load Factor :-

α = n/m ; **n** = Total no of element; **m** = size of array
=> 20/10 = 2 → every location enters 2 elements.

Open Addressing :-

1. Linear Probing
2. Quad Probing
3. Double hashing

MAP

stores a collection of key-value pairs, where each key is associated with a single value.

MAP Define :-

map<key> = value
M[pradum] = 1999;
map<int, int>m;

Map = Balance Binary Search Tree O(log n)

Create Map :-

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

int main()
{
    map<int, int>m;
    // insert the data
    m[2] = 111;
    m[4] = 222;
    m[50] = 0;
    cout<<m[2]<<endl;
    cout<<m[4]<<endl;
    cout<<m[10]<<endl;
    // 1 --> there have value
    // 0 --> there have no value
    cout<<m.count(50)<<endl;

    // size
    cout<<m.size()<<endl;

    // print all value
    for(auto i= m.begin(); i != m.end(); i++)
    {
        cout<<i->first<<" "<<i->second<<endl;
    }

return 0;
};
```

Unordered_map :-

```
#include<iostream>
#include<map>
#include<unordered_map>
using namespace std;

int main()
{
    unordered_map<int, int>m;
    // insert the data
    m[2] = 111;
    m[4] = 222;
    m[50] = 0;

    // size
    cout<<m.size()<<endl;
    cout<<m[2]<<endl;
    cout<<m[4]<<endl;
    cout<<m[10]<<endl;
    // 1 --> there have value
    // 0 --> there have no value
    cout<<m.count(50)<<endl;

    // print all value
    for(auto i= m.begin(); i != m.end(); i++)
    {
        cout<<i->first<<" "<<i->second<<endl;
    }
return 0;
};
```

First element to occur k times << GeeksforGeeks >>

```
class Solution{
    public:
    int firstElementKTime(int a[], int n, int k)
    {
        // number, count
        unordered_map<int, int>m;
        for(int i=0; i<n; i++)
        {
            m[a[i]]++;
            // exist nahi kare he to use create karo
            // exist karte he to use 1 increase karo
            if(m[a[i]]== k)
            return a[i];
        }
        return -1;
    };
};
```

LEARN DSA WITH C++

WEEK :: 12 **DAY**: 02 **DATE**: 11-07-2023

HASHING & MAP

Smallest range in K lists << GeeksforGeeks >>

```
class Solution{
    public:
    pair<int,int> findSmallestRange(int arr[][N], int n, int k)
    {
          //code here
          priority_queue<pair<int, pair<int, int>>, vector<pair<int, pair<int,
int>>>,greater<pair<int, pair<int, int>>>>minheap;
          int mini = INT_MAX;
          int maxi = INT_MIN;
          int row, col;

          // Create minheap with k element
          for(int i=0; i<k; i++)
          {
              row = i;
              col = 0;
              minheap.push(make_pair(arr[row][col], make_pair(row, col)));
              mini = min(mini, arr[row][col]);
              maxi = max(maxi, arr[row][col]);
          };
          int start = mini;
          int end = maxi;
          pair<int, pair<int,int>>temp;
          while(!minheap.empty())
          {
              temp = minheap.top();
              minheap.pop();
              row = temp.second.first;
              col = temp.second.second;
              mini = temp.first;

              if(end - start > maxi - mini)
              {
                  end = maxi;
                  start = mini;
              }
              if(col ==n-1)
              break;
              else
              {
                  maxi = max(maxi, arr[row][col +1]);
                  minheap.push(make_pair(arr[row][col+1], make_pair(row, col+1)));
              }
          }
          return {start,end};
    }
};
```

Largest subarray with 0 sum << GeeksforGeeks >>

```
class Solution{
    public:
    int maxLen(vector<int>&A, int n)
    {
        // Your code here
        int len = 0;
        // number, index
        unordered_map<int, int>m;
        int sum = 0;
        for(int i=0; i<n; i++)
        {
            sum+= A[i];
            //Exist
            if(sum==0)
            len = i+1;
            else if(m.count(sum))
            len = max(len, i-m[sum]);
            // not exist
            else
            m[sum] = i;
        }
        if(sum==0)
        return n;

        return len;
    }
};
```

2 Sum << InterviewBit >>

```
vector<int> Solution::twoSum(const vector<int> &A, int B) {
    unordered_map<int, int>m;
    vector<int>ans;
    for(int i=0; i<A.size(); i++)
    {
        if(m.count(B-A[i]))
        {
            ans.push_back(m[B-A[i]]+1);
            ans.push_back(i+1);
            return ans;
        }
        else
        {
            if(m.count(A[i]) ==0)
            m[A[i]]=i;
        }
    }
    return ans;
}
```

Largest subarray of 0's and 1's << GeeksforGeeks >>

```
class Solution{
  public:
    int maxLen(int A[], int N)
    {
        // Your code here
        int len = 0;
        // sum, Index
        unordered_map<int, int>m;
        int sum = 0;
        for(int i=0; i<N; i++)
        {
            if(A[i] ==1)
            sum++;
            else
            sum--;

            // 1 : sum=0
            if(sum == 0)
            len = i+1;

            // 2 if sum does not exist
            else if(m.count(sum))
            {
                len = max(len, i-m[sum]);
            }
            //3 If sum doesn't exist
            else
            m[sum] = i;
        }
        return len;
    }
};
```

Count distinct elements in every window << GeeksforGeeks >>

```
class Solution {
public:
    vector<int> countDistinct(int A[], int n, int k) {
        // A[i],  count
        unordered_map<int, int> m;
        int distinct_count = 0;
        vector<int> ans;

        for (int i = 0; i < k; i++)
        {
            m[A[i]]++;
            if (m[A[i]] == 1)
                distinct_count++;
        }

        ans.push_back(distinct_count);

        for (int i = k; i < n; i++)
        {
            // Old element exclude hoga
            m[A[i - k]]--;
            if (m[A[i - k]] == 0)
                distinct_count--;

            // new element add hoga
            m[A[i]]++;
            if (m[A[i]] == 1)
                distinct_count++;

            ans.push_back(distinct_count);
        }

        return ans;
    }
};
```

LEARN DSA WITH C++

WEEK :: 12 **DAY**: 03 **DATE**: 12-07-2023

HASHING & MAPs

First Repeating element << InterviewBit >>

```
int Solution::solve(vector<int> &A)
{
    int n = A.size();
    unordered_map<int, int> m;

    int ans = -1;
    int smallest_index = n; // Initialize the smallest index to a value larger than the array size.

    for (int i = 0; i < n; i++) {
        if (m.count(A[i]) > 0) {
            // A[i] is a repeating element.
            if (m[A[i]] < smallest_index) {
                smallest_index = m[A[i]];
                ans = A[i];
            }
        } else {
            // First occurrence of A[i], store its index.
            m[A[i]] = i;
        }
    }
    return ans;
}
```

Subarrays with equal 1s and 0s << GeeksforGeeks >>

```
class Solution{
  public:
    //Function to count subarrays with 1s and 0s.
    long long int countSubarrWithEqualZeroAndOne(int arr[], int n)
    {
        //Your code here
        int sum = 0;
        long long int final =0;
        unordered_map<int,int>m;
        m[0] =1;

        for(int i=0; i<n; i++)
        {
            if(arr[i])
            sum++;
            else
            sum--;

            final += m[sum];
            m[sum]++;
        }

        return final;
    }
};
```

Subarray with B odd numbers << InterviewBit >>

```
int Solution::solve(vector<int> &A, int B) {
    unordered_map<int, int>m;
    // total = count of odd, sub = no of subarray
    int total =0, sum =0;
    m[0] =1;

    for(int i=0; i<A.size(); i++)
    {
        // Count number of odd
        if(A[i]%2)
        total++;
        // Map is increase
        m[total]++;
        // B==0
        if(B==0)
        sum +=(m[total -B] -1);
        // sum mein include
        else if(m.count(total -B))
        sum += m[total - B];
    }
    return sum;
}
```

Equal 0, 1 and 2 << GeeksforGeeks >>

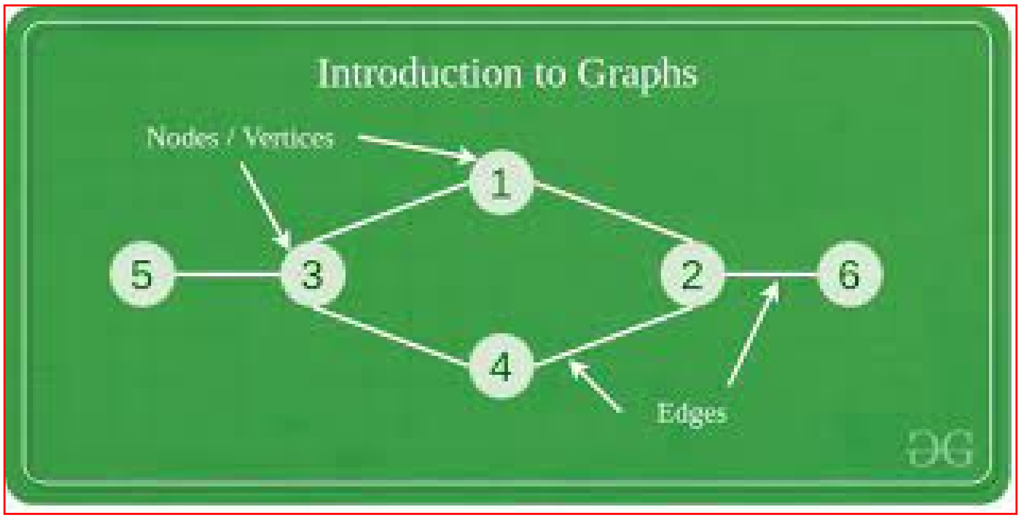
```
class Solution {
  public:
    long long getSubstringWithEqual012(string str) {
        // code here

        unordered_map<int, unordered_map<int, int>>m;
        m[0][0] = 1;
        int count_0 = 0, count_1 =0, count_2 = 0;
        long long sum =0;
        int first, second;

        for(int i=0; i<str.size(); i++)
        {
            if(str[i] =='0')
            count_0++;
            else if(str[i] == '1')
            count_1++;
            else
            count_2++;

            first = count_0 - count_1, second = count_0 - count_2;
            sum += m[first][second];
            m[first][second]++;
        }
        return sum;
    }
};
```

LEARN DSA WITH C++

WEEK :: 12 **DAY**: 04 **DATE**: 13-07-2023

GRAPH BASIC

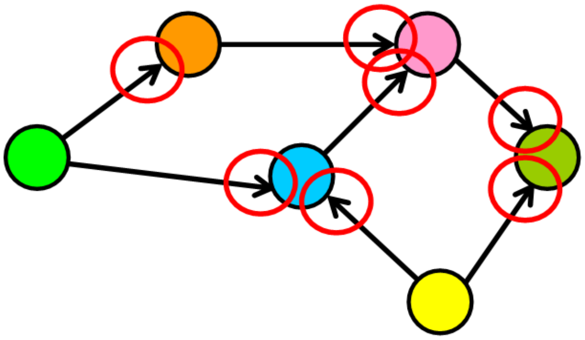
A Graph is a non-linear data structure consisting of vertices and edges.

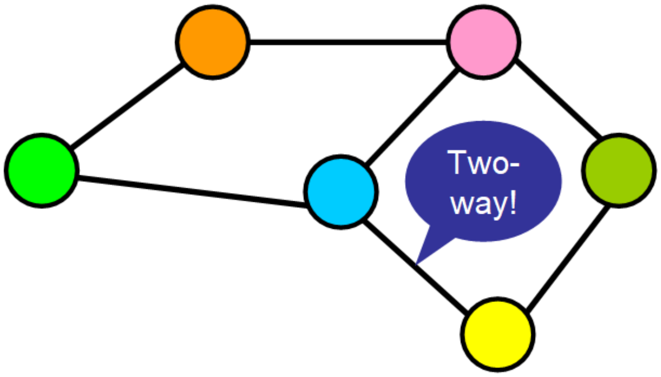


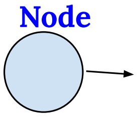
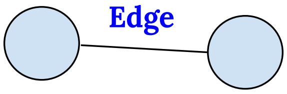
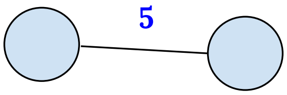
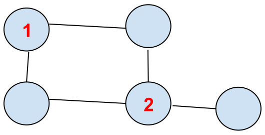
Graph Use :-

GPS systems and Google Maps
Social Networks
The Google Search
Operations Research
Even Chemistry

Directed Graph :-
[Define Direction]



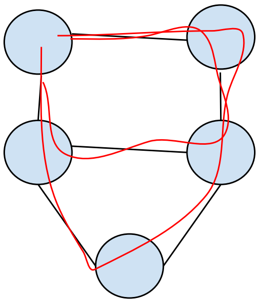
Undirected Graph :- [No Direction]	

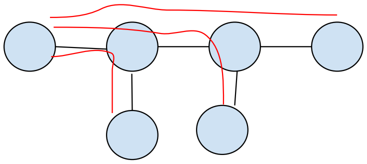
Vertex :- (Node)			
Edge : Which connects 2 Node			
Weighted/graph			
Degree :		For 1	For 2
		2	3

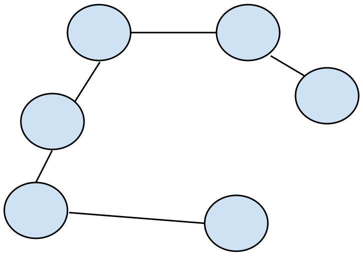
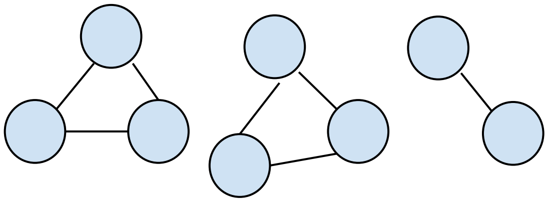
Directed Graph :-

In-Degree :- edge is incoming

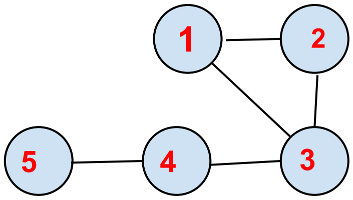
Out-Degree:- edge is outcome

Cycle :-	No Edge or No Node should repeat
	

Path :- No repeat same edge & Node	

Connected Graph	Disconnected Graph
	

Undirected Graph - Representation



Connected Node define in Table

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	0	0
3	1	1	0	1	0
4	0	0	1	0	1
5	0	0	0	1	0

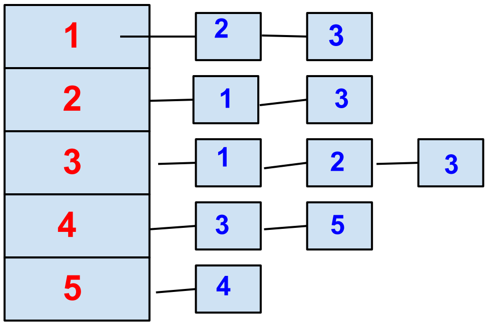
Adjacent Matrix

Time Complexity :- O(n^2)

Space Complexity :- O(n^2)

Adjacency :-

1. [2,3]
2. [1,3]
3. [3,4]
4. [4,5]
5. [5]



Time Complexity :- O(2*E + V)	Space Complexity :- O(2*E + V)

	Adjacent Matrix	Adjacency List
Insertion	O(1)	O(1)
Delete	O(1)	O(V)
Find / Search	O(1)	O(V)

Using array help of Vector & STL :-

vector<int>arr[5]

0	2 , 3
1	1 , 3
2	1 , 2 , 3
3	3 , 5
4	4

Which Graph use :-

Dense Graph :- high density [most node connected]	Adj Matrix
Sparse Graph :- low edges	Adjacency

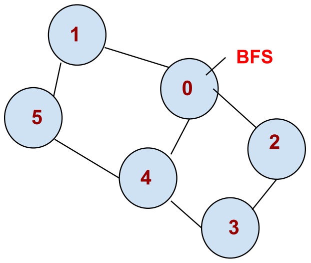
Adjacent Matrix :-

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

int main()
{
    int v, e;
    cin>>v>>e;
    int A[v][v];
    int a, b;
    for(int i=0; i<e; i++)
    {
        cin>>a>>b;
        A[a][b] =1;
        A[b][a] =1;
    }

return 0;
};
```

Adjacency List :-

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

int main()
{
    int v, e;
    cin>>v>>e;
    vector<int>A[v];
    int a, b;
    for(int i=0; i<e; i++)
    {
        cin>>a>>b;
        A[a].push_back(b);
        A[b].push_back(a);
    }

return 0;
};
```

Breadth First Search or BFS for a Graph :-



1st print 1 distance : 1 , 2 , 4;
2nd print 2 distance : 3 , 5;

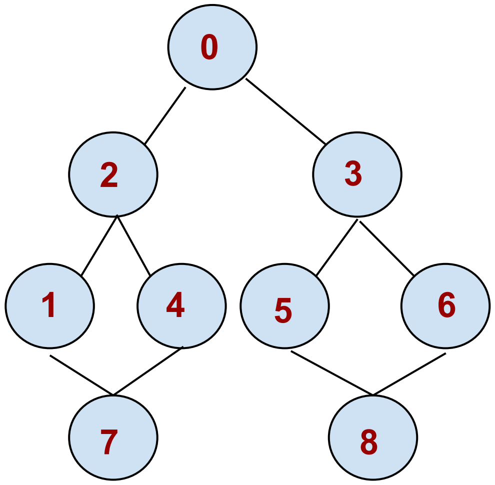
Increase the distance 1 edge

0, 2, 4, 1, 5, 3

BFS of graph << GeeksforGeeks >>

```
class Solution {
  public:
    // Function to return Breadth First Traversal of given graph.
    vector<int> bfsOfGraph(int V, vector<int> adj[]) {
        // Code here
        vector<int>ans;
        bool visited[V] = {0};
        queue<int>q;
        q.push(0);
        visited[0] =1;
        int node;
        while(!q.empty())
        {
            node = q.front();
            q.pop();
            ans.push_back(node);

            for(int i=0; i<adj[node].size(); i++)
            {
                if(!visited[adj[node][i]])
                {
                    q.push(adj[node][i]);
                    visited[adj[node][i]] =1;
                }
            }
        }
        return ans;
    }
};
```

Depth First Search or DFS for a Graph :-



First print one side : 0, 2, 1, 7, 4,
Then another side : 3, 5, 8, 6

[Go Depth]

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

DFS of Graph << GeeksforGeeks >>

```
class Solution {
  public:
    // Function to return a list containing the DFS traversal of the graph.

    void DFS(int node, vector<int>adj[], vector<int> &ans, vector<bool> &visited)
    {
        if(visited[node])
        return;

        visited[node] = 1;
        ans.push_back(node);

        for(int i=0; i<adj[node].size(); i++)
        DFS(adj[node][i], adj, ans, visited);
    };

    vector<int> dfsOfGraph(int V, vector<int> adj[]) {
        // Code here
        vector<bool>visited(V, 0);
        vector<int>ans;

        DFS(0, adj, ans, visited);

        return ans;
    }
};
```

LEARN DSA WITH C++

WEEK :: 12 **DAY: 05** **DATE: 14-07-2023**

GRAPH DEEP

Shortest path from 1 to n << GeeksforGeeks >>

```
class Solution{
public:
    int minimumStep(int n){
        //complete the function here
        int count_edge =0;
        while(n>=3)
        {
            count_edge += n%3;
            n /= 3;
            count_edge++;
        };
        count_edge += n;
        count_edge--;
        return count_edge;
    }
};
```

Find Center of Star Graph << LeetCode >>

```
class Solution {
public:
    int findCenter(vector<vector<int>>& edges) {
        if(edges[0][0] == edges[1][0] || edges[0][0] == edges[1][1])
        return edges[0][0];
        else
        return edges[0][1];
    }
};
```

Number of Provinces << GeeksforGeeks >>

```
class Solution {
  public:
  
  void DFS(vector<vector<int>> &adj, vector<bool> &visit, int node)
  {
      if(visit[node])
      return;
      
      visit[node] = 1;
      
      for(int i=0; i<adj[node].size(); i++)
      {
          if(adj[node][i])
          DFS(adj, visit, i);
      };
      return;
  }
    int numProvinces(vector<vector<int>> adj, int V) {
        // code here
        int count = 0;
        vector<bool>visit(V, 0);
        
        for(int i=0; i<V; i++)
        {
            if(!visit[i])
            {
                count++;
                DFS(adj, visit, i);
            }
        }
        return count;
    }
};
```

Detect cycle in an undirected graph << GeeksforGeeks >>

```
class Solution {
  public:
    // Function to detect cycle in an undirected graph.
    bool DetectCycle(vector<int>adj[], int node, int parent, vector<bool>visited)
    {
        visited[node] =1;
        for(int i=0; i<adj[node].size(); i++)
        {
            // If its adjacent are not visited
            if(!visited[adj[node][i]])
            {
                if(DetectCycle(adj, adj[node][i], node, visited))
                return 1;
            }
            else if(parent != adj[node][i])
            return 1;
        }
    }
    bool isCycle(int V, vector<int> adj[]) {
        // Code here
        vector<bool>visited(V, 0);
        for(int i=0; i<V; i++)
        {
            if(!visited[i])
            {
                if(DetectCycle(adj, i, -1, visited))
                return 1;
            }
        }
        return 0;
    }
};
```