A Look at Graphs

Theory, Design and Algorithms

Sandesh Bhusal

About the Instructor

Sandesh Bhusal

Avid fan of LOTR, Linux and Open Source

Find me on:

Github: @sandeshbhusal
Facebook: @bhussu
Blog: sandeshbhusal.github.io/blog

What we will be overviewing

- Macros and a Sample Test case
- Re-routing Input output
- C++ Powerhouse
  - Vector, map
  - Auto, lambda and params
- Graph Theory : : Basics, Types
- Graphs: Representation
- Graphs: DFS, BFS, SSSP, LCA and where they are used.

Macros for CP

Useful macros to exploit while doing competitive programming

The compilers in the online judge compile your code and run them to generate an output. Your output will then be compared against standard output to see if you gave the correct answer. In most of the online judges, a macro

ONLINE_JUDGE

Is defined to check if it is an online judge or not

```
#ifdef ONLINE_JUDGE
// Code for online judge
#endif
```

Macros for CP

Useful macros to exploit while doing competitive programming

You can use macros while doing competitive programming. The judge exposes a macro called ONLINE_JUDGE. In conjunction to it, you can use macros such as:

```
#define vector<int> vi

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

#define ll long long
```

Though we do not use many macros. Your code will most likely be -O2 optimized within the compiler, but this might not happen. You might have noticed

```
#include <bits/stdc++.h>
```

Which includes all the STL libraries in your code.

The Structure of a test case

The basic outline how the compiler in most online judges work

Most files begin with the number of test cases, then following them will be lines of input. See the following question for an example:

Input: The input file contains a single line containing an integer “N” representing the number of test cases. “N” lines follow. Each line contains integers “a” and “b”. You have to print the GCD of the integers followed by a newline character

Eg:
4
1 2
4 8
3 9
12 84

Re Routing Input And Output

Changing the default pointers so that we can read inputs more efficiently

Let us take an example of the previous question. How would you go about doing it?

Sample code:

```
int tc;
while(tc--){
     int a, b;
     int c = gcd(a, b);
     printf(“%d \n”, c);
}
```

The problem: What if you want to test multiple cases on your own?

Re Routing Input And Output

Changing the default pointers so that we can read inputs more efficiently

You can re-route the input and output through the freopen command. This will let you test many test cases and check them on your computer. Along with the judge definitions, you can pull the input from an input file and dump them on a output file.

Code Example:

```
#ifndef ONLINE_JUDGE
    freopen(“input.txt”, “r”, stdin);
    freopen(“output.txt”, “w”, stdout);
#endif
```

Part I : C++
PowerHouse

VECTOR

The dynamically allocated arrays in C++. Included in the Standard Template library and will make your life much, much easier.

Vector is a structure that resembles a dynamically allocated array in C++. If you need to make an array that increases with size, you need a vector. It has following structure and functions:

vector<data_type> name;

Eg.

```
vector<int> myVector;
```

Useful methods:

1. size()
2. push_back()
3. resize()
4. pop_back()
5. Referencing using []

MAP

The key-value pair array in C++. Useful for string algorithms and integer counting. Included in the Standard Template library and will make your life much, much easier.

Maps are key-value pairs in C++.

Structure:

std::map<key_data_type, value_data_type> name;

Eg:

```
std::map<int, int> countFactorPower;
std::map<char, int> characterFrequency;
```

The values are initialized as empty, and integers are initialized as 0.

Additional Methods for you to explore:

```
std::find(container.begin(),
container.end(), compare_function)
```

AUTO AND LAMBDA

Additions with C++11 that are cool

You can detect the type of variable using auto keyword if you do not know the type.

```
for(auto x: array){
    std::cout << x << std::endl;
}
```

Auto keyword is also useful for lambda function:

```
auto funcName = [](params){
    // code
}
```

You are requested to explore these further. Bonus: Foreach loop.

Passing Parameters

What you should and should not do.

Passing parameters can be done in two ways:

1. Pass by value
2. Pass by reference.

Additionally you should be using the const keyword wherever possible.

Part II : The **Basics** of Graph

Need of a Graph

In the modern world, Graphs are used to model various things, some of which are listed on this slide.

Quick Lookups, Balanced additions

Flow modeling, of information or any other thing

Distributed Systems design

Natural Language Processing

Social Sciences relationship modeling

Mathematical theories, set checks

…

Ever heard of **facebook** ?

What is a Graph ?

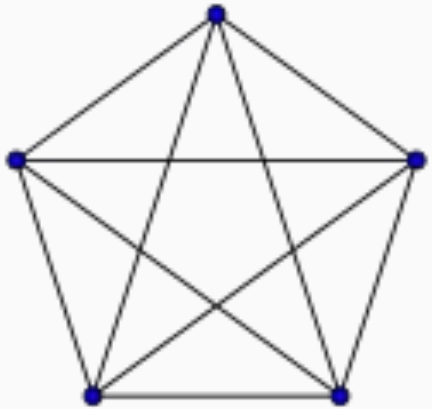
A graph is a representation of relationship between objects, where objects are represented by a node, called **Vertex** and the relationships are modeled by links, called **Edges**. It is a non empty ordered pair, of vertices and edges

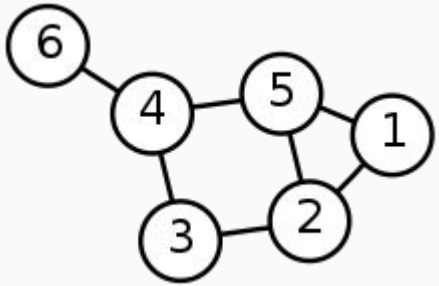
V = { set of vertices }

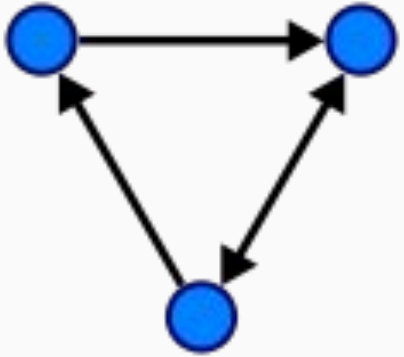
E = { (a, b, {k}) , … }; where a and b belong to set V and { k } is relation between them

So, G = (V, E)

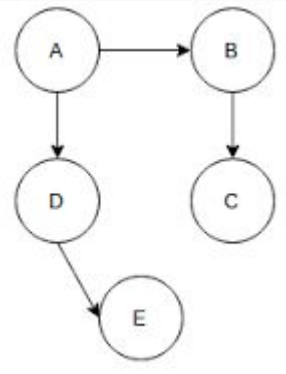
Some Examples:

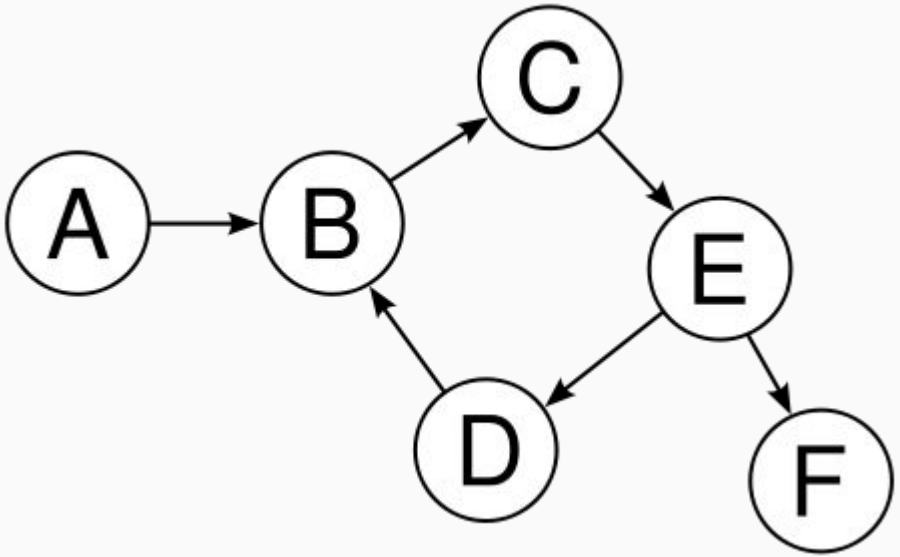




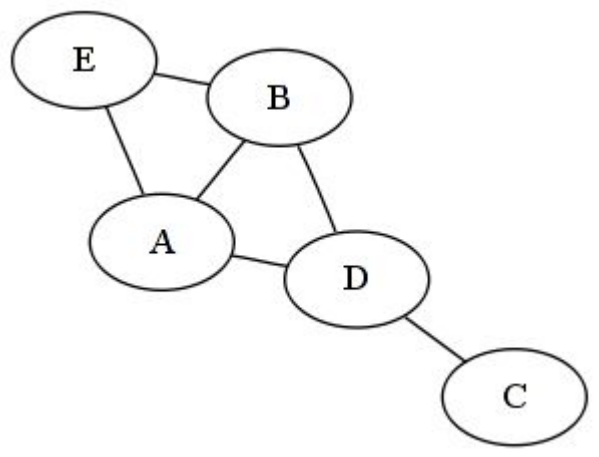


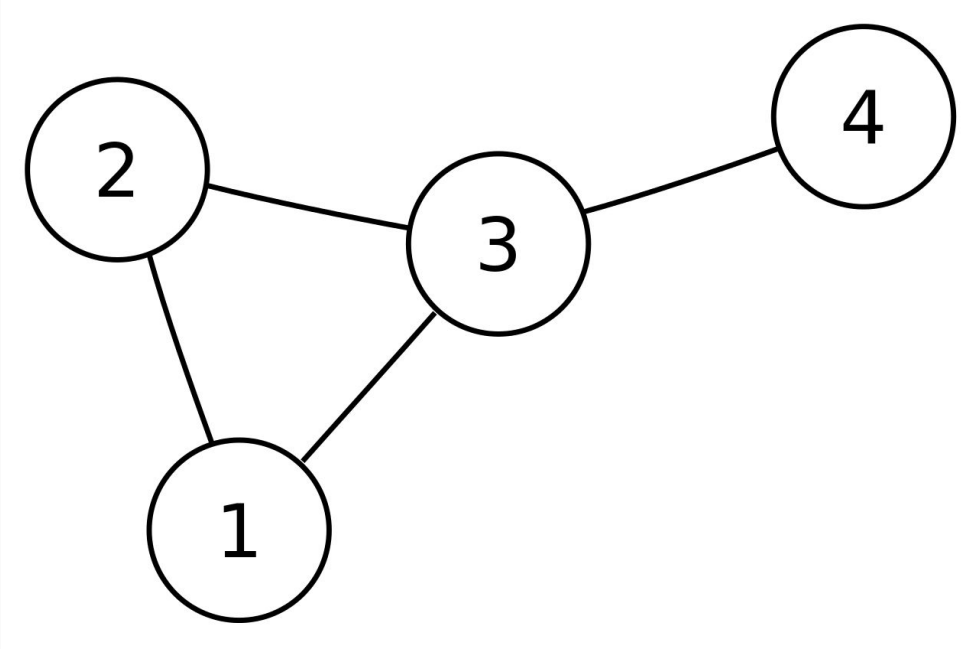
Types of Graphs :: Directed



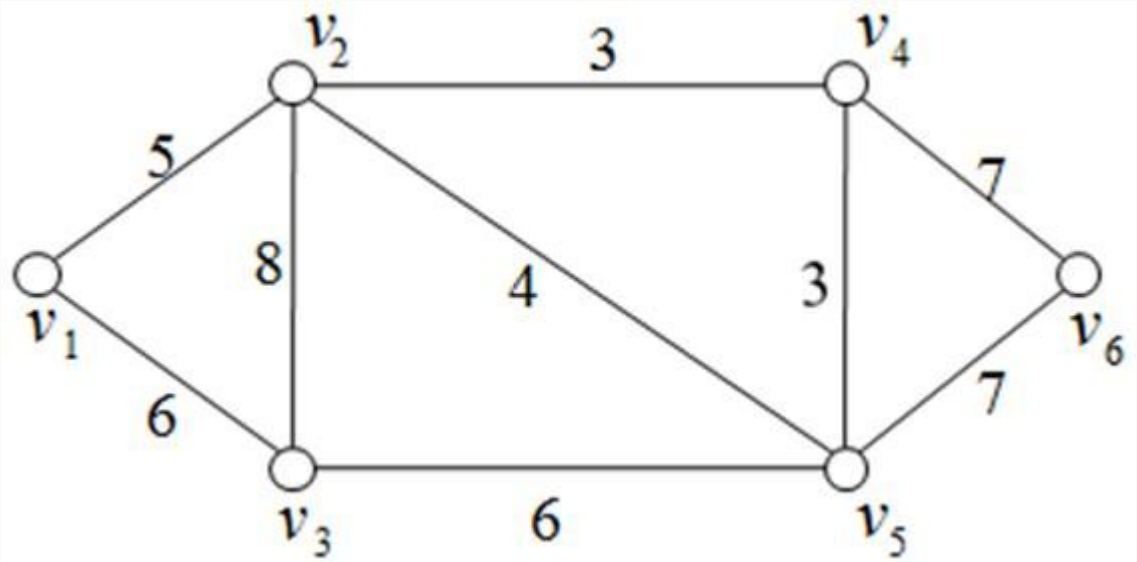


Types of Graphs :: Undirected

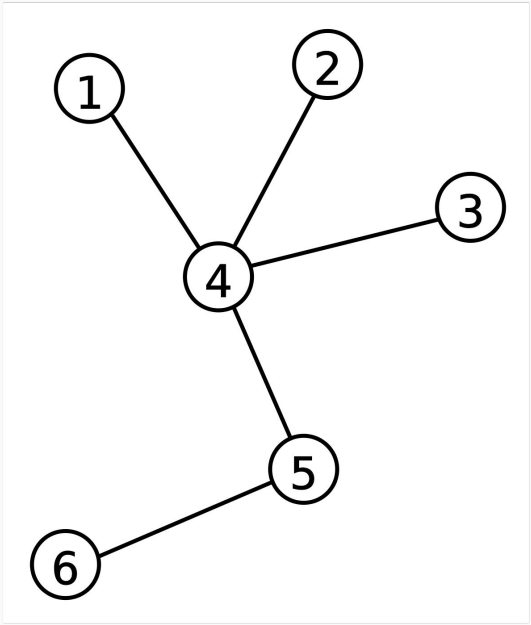


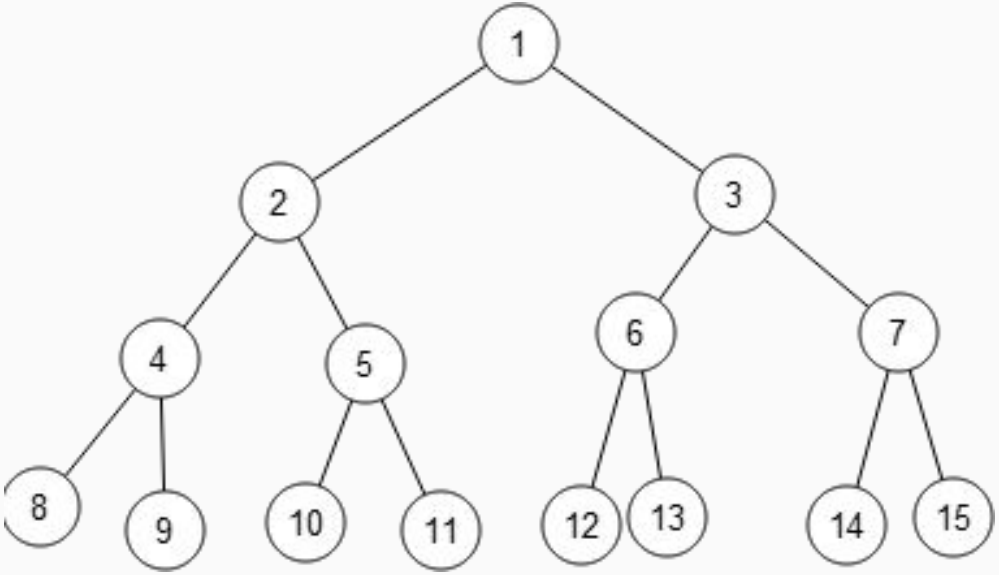


Types of Graphs :: Weighted

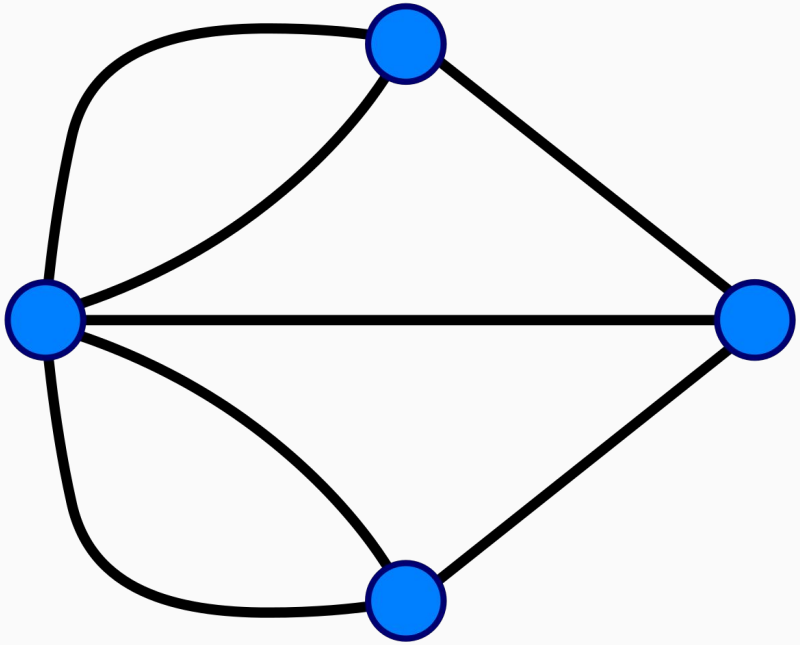


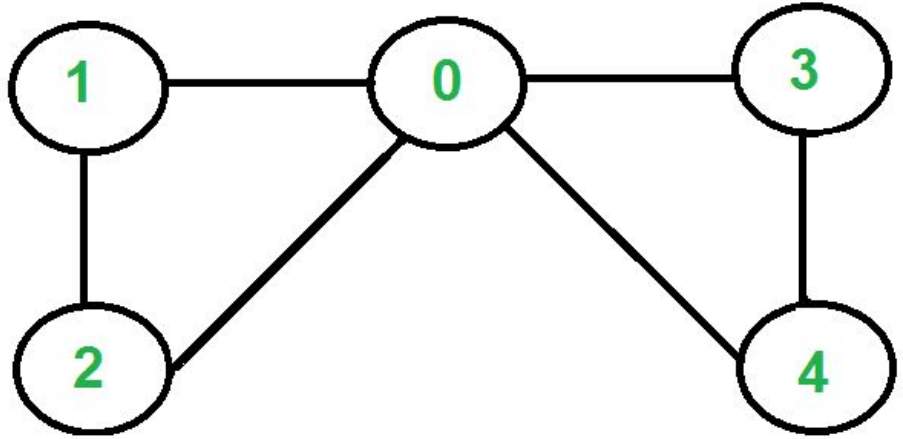
Types of Graphs :: Trees





Euler Path (Seven Bridges of Königsberg)





The graph has Eulerian Cycles, for example "2 1 0 3 4 0 2"
Note that all vertices have even degree

Representation in computers

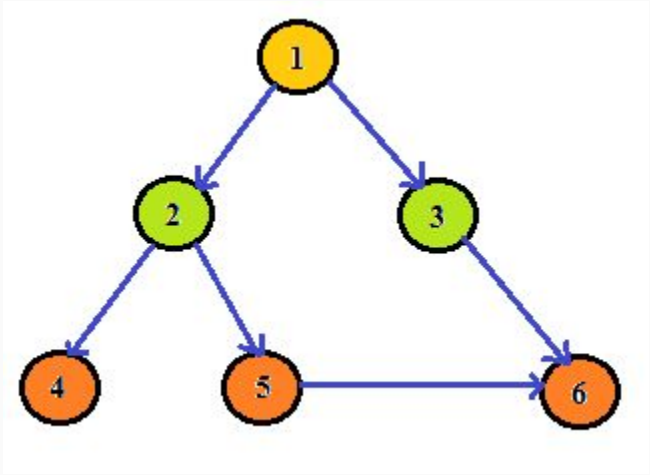
1. Adjacency Lists
2. Adjacency Matrices
3. Standard Conventions.

Code Examples

1. Adjacency List
2. Adjacency matrix
3. Comparison in memory

Part III : Graph Algorithms

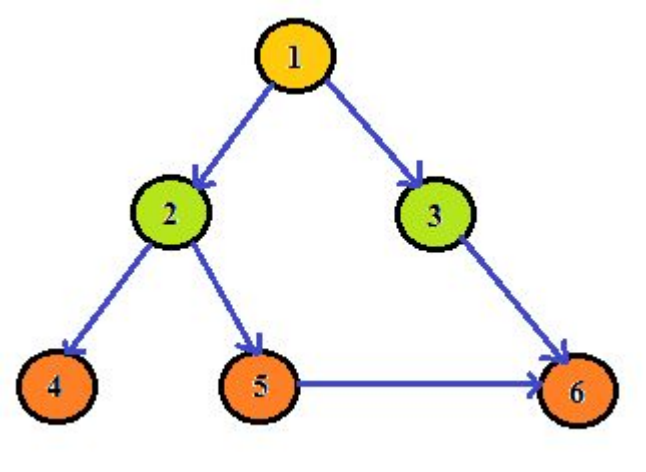
Depth First Search (DFS)




```
DFS-iterative (G, s):                          //Where G is graph and s is source vertex
    let S be stack
    S.push( s )            //Inserting s in stack
    mark s as visited.
    while ( S is not empty):
        //Pop a vertex from stack to visit next
        v  =  S.top( )
        S.pop( )
        //Push all the neighbours of v in stack that are not visited
        for all neighbours w of v in Graph G:
            if w is not visited :
                    S.push( w )
                    mark w as visited
```

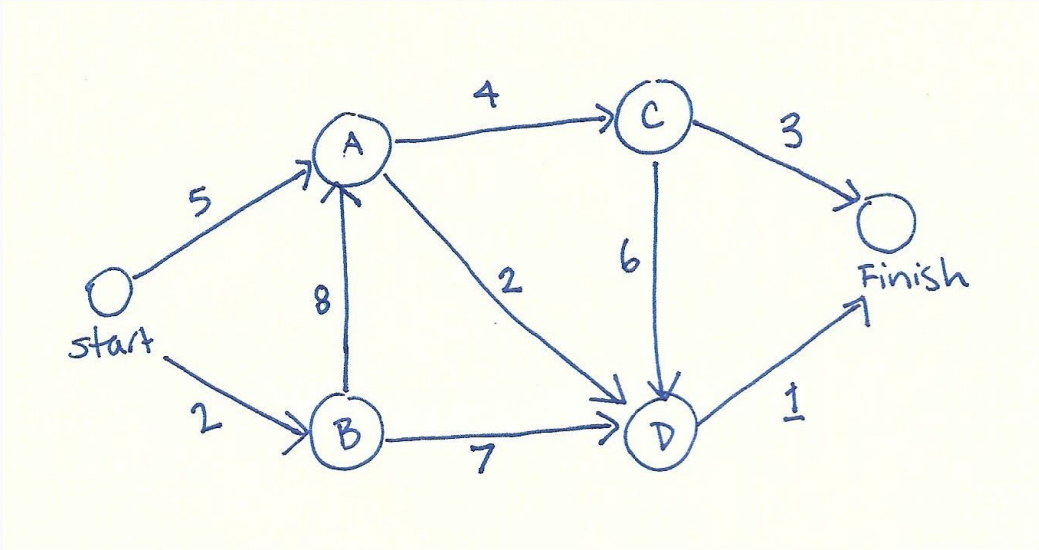
```
DFS-recursive(G, s):
    mark s as visited
    for all neighbours w of s in Graph G:
        if w is not visited:
            DFS-recursive(G, w)
```

Breadth First Search (BFS)



The theory is left as an exercise to the reader, as the source code has been included here.

SSSP (Using Dijkstra's Algorithm)



1. Assign all vertices weight of infinity and source as 0.
2. Scan the graph for min. Weight vertex, and update its children as weight(child) = min(weight(child), weight(parent)+edge)
3. Add the vertex to visited nodes array
4. Repeat steps 2 to 3 until there is a min weight vertex or the destination is reached.

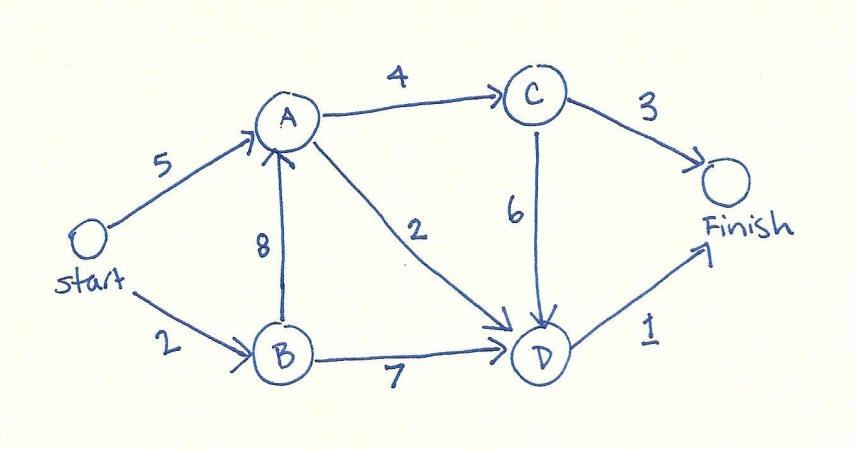
Exercise: Find the difference between Prim's algorithm and Dijkstra's algorithm

Spanning Trees

Spanning Trees:

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.

Spanning Trees (Kruskal)

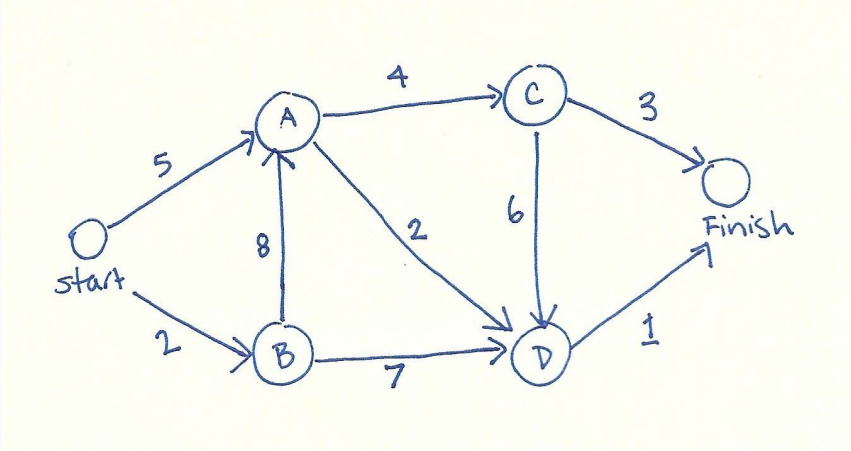


Kruskal’s Algorithm

Kruskal’s Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.
Algorithm Steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

Spanning Trees (Prim)



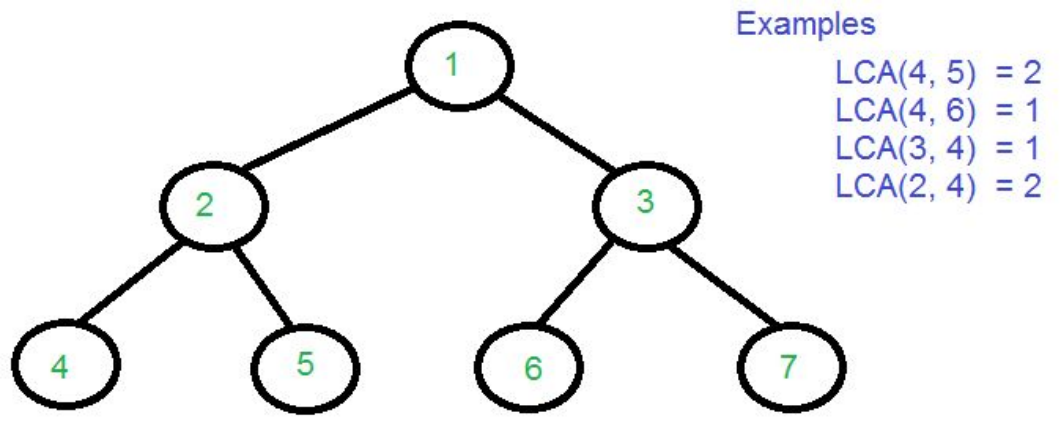
- Maintain two disjoint sets of vertices. One containing vertices that are in the growing spanning tree and other that are not in the growing spanning tree.
- Select the cheapest vertex that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using Priority Queues. Insert the vertices, that are connected to growing spanning tree, into the Priority Queue.
- Check for cycles. To do that, mark the nodes which have been already selected and insert only those nodes in the Priority Queue that are not marked.

Lowest Common ancestor.

LCA is a very interesting algorithm, which helps us to find the common parent between any two nodes in a graph. The steps to do this are:

1. Pick one of the two nodes, and traverse to the root. Make a list of all the nodes along the way. In order to preserve time, the nodes can be stored as a boolean array.
2. Select the other node, and keep traversing upwards until the current parent node is found in the visited array.

This approach is not an optimized one. You can use RMQ for the optimized approach.



BONUS Slide :D

1. Sliding Window Algorithm
2. Two Pointer Algorithm
3. Bracket Matching Problem
4. Some questions for you to solve:
    a. Given some ordered pairs of integers denoting edges in a graph, find the number of connected components (Hackerrank Roads and Libraries, Moon Problem)
5. BogoSort, TimeSort and Some funny Algorithms

Where to from here?

1. Practice makes a man perfect
2. Hackerrank, Codeforces, Leetcode
3. VJudge
4. College Group
5. Nepali Samaj
6. ACM ICPC Selection Talk