Train VIII

Nov 14. Ye Cao

Things we covered so far :

Week 1 Sort
Search

2 pointer
Binary Search
BIT (Binary Indexed Tree)

3. Fenwick Tree & RSQ (Range Sum Query)
Accumulative list
Square root decomposition
Multidimensional Accumulative Matrix

4. Range Sum Query Underlying number theory

5. DP (Dynamic Programming)
Shortest path problem
Exhaustive Search done in linear time
Fibonacci Number Problem
Parent Pointer technique

6 Perfect information Blackjack (DP)

7 Longest Palindrom Subsequence (DP)

8. Schedual Planning (No session held)

9. Graph

Graph Terminology:

Nodes
Edges

Path (A path is simple if each node appears at most once in the path)
Cycle

Connect Graph: $\exists$ a path $\forall$ node a to $\forall$ node b

Components:
The connected parts of a graph are called its components

Tree: a graph consisted of n nodes and n-1 edges

Directed Graph: edges can traverse in one direction only

Edge Weight:
Each edge is assigned a weight.

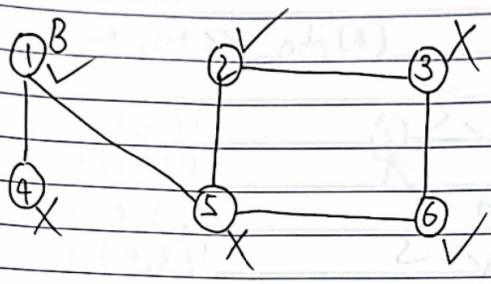
Regular graph: Every node has degree of constant d.
Complete graph: Degree of every node is n-1.

Colorings:

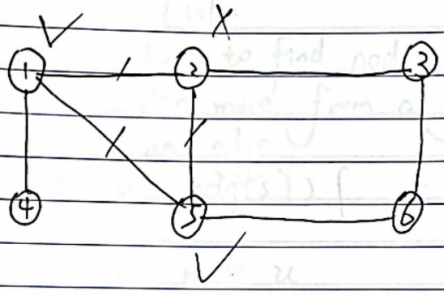
Bipartite: $\Leftrightarrow$ it's possible to color graph using two colors
$\Leftrightarrow$ Exactly when it doesn't contain a cycle with odd number edges

Example :

Bipartite



Non-bipartite



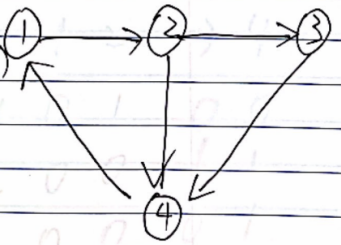
Graph Representation

Adjacency list representation

```
vector<int> adj[N];
```

adj[1].PB(2)
adj[2].PB(3)
adj[3].PB(4)
adj[4].PB(1)

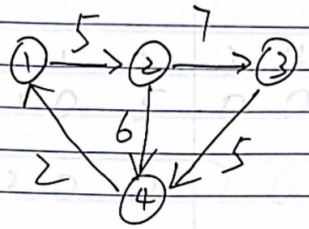
adj[2].PB(4)



weighed graph repesentation

Vector<pair<int, int>> adj[4]

adj[1].PB({2,5})
adj[2].PB({3,7})
adj[2].PB({4,6})
adj[3].PB({4,5})
adj[4].PB({1,2})



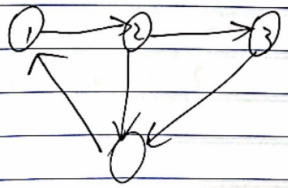
Benefit of Adjacency List:

Easy to find nodes to which we can move from a given node through an edge.

```
for(auto u : adj[s]){
    // process node u
}
```

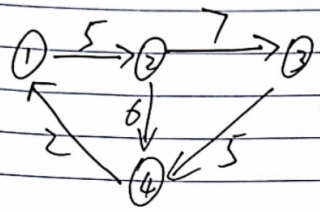
Adjacency Matrix

int adj[A][V]



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

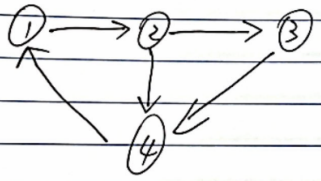
weighted



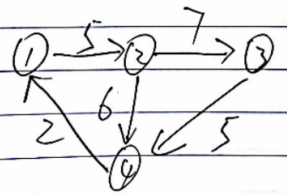
	1	2	3	4
1	0	5	0	0
2	0	0	7	6
3	0	0	0	5
4	2	0	0	0

Edge list

vector<pair<int, int>> edges;



edges.PB({1, 2})
edges.PB({2, 3})
edges.PB({2, 4})
edges.PB({3, 4})
edges.PB({4, 1})



{1, 2, 5}
{2, 3, 7}
{2, 4, 6}
{3, 4, 5}
{4, 1, 2}

Bipartite Check

```
bool isBipartite(vector<vector<int>> & graph) {
    int color[graph.size()];
    for (int idx = 0; idx < graph.size(); idx++) {
        color[idx] = 0;
    }
    queue<int> visits;
    bool isComplete;
    do {
        isComplete = true;
        for (int idx
```

See GitHub for rest of the codes + comments