LECTURE 3

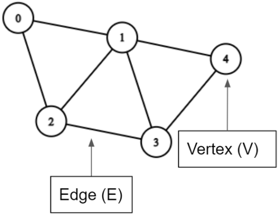
BASIC GRAPH REDUCTION

CS200

Feb 10, 2023

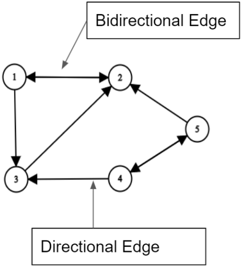
Definition

Our View of A Graph



Vertex (V)

Edge (E)



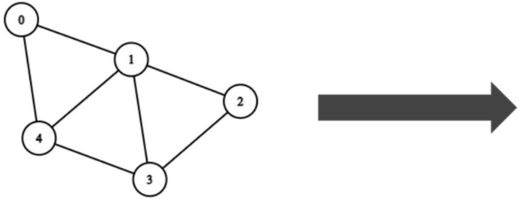
Bidirectional Edge

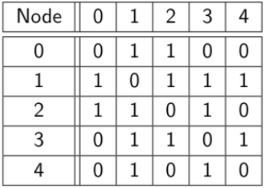
Directional Edge

Code Representation

Can represent graph in two ways.

1. Adjacency Matrix





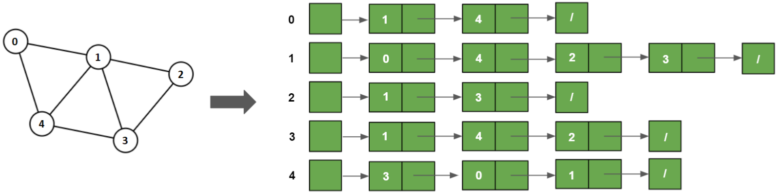
Node	0	1	2	3	4
0	0	1	1	0	0
1	1	0	1	1	1
2	1	1	0	1	0
3	0	1	1	0	1
4	0	1	0	1	0

Adding a node and graph traversal are $\texttt{O(V}^{2}\texttt{)}$.

$$M_{u,v} = 1 \text{ iff u has a directed edge to v}$$

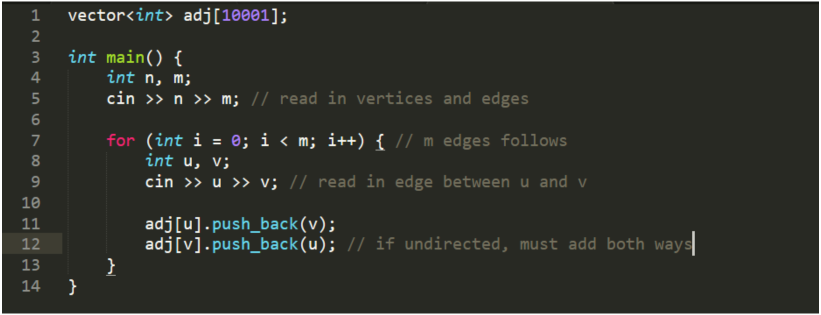
Code Representation

2. Adjacency List (List of LinkedLists)



Complexity: `O(V + E)`

Adjacency List in C++

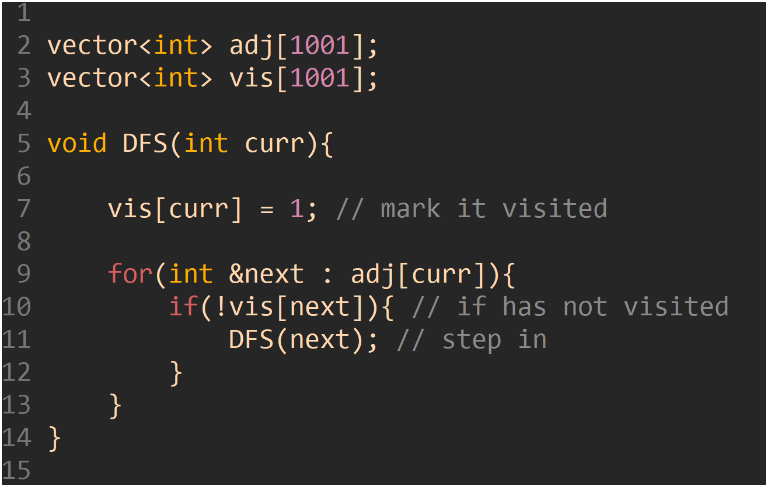


```cpp
vector<int> adj[10001];

int main() {
    int n, m;
    cin >> n >> m; // read in vertices and edges

    for (int i = 0; i < m; i++) { // m edges follows
        int u, v;
        cin >> u >> v; // read in edge between u and v

        adj[u].push_back(v);
        adj[v].push_back(u); // if undirected, must add both ways
    }
}
```

DFS & BFS

Depth-First Search (DFS)

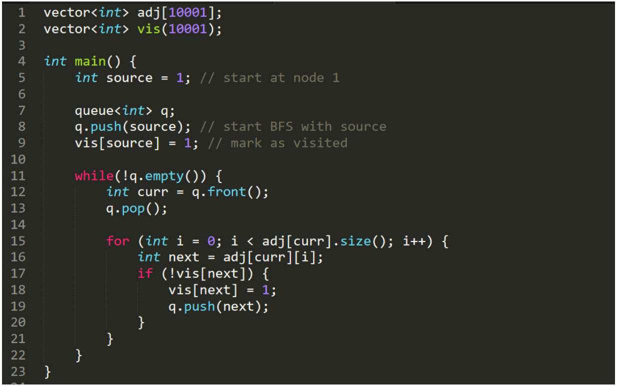
- Explore as deep as possible, backtrack once branch is fully explored.
- Implement recursively or with a stack.



```

vector<int> adj[1001];
vector<int> vis[1001];

void DFS(int curr){

    vis[curr] = 1; // mark it visited

    for(int &next : adj[curr]){
        if(!vis[next]){ // if has not visited
            DFS(next); // step in
        }
    }
}

```

Breadth-First Search (BFS)

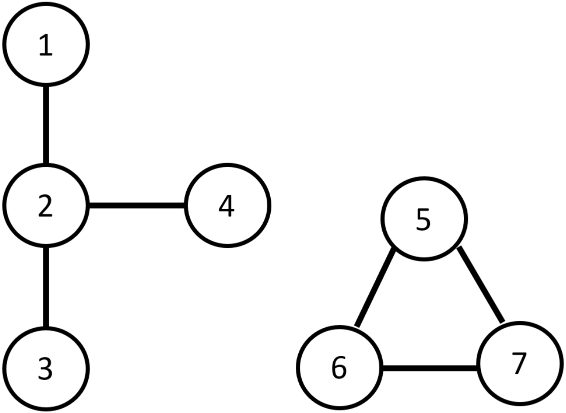
- Explore as wide as possible, look through all neighbors before moving on.
- Implement using a queue.



```cpp
vector<int> adj[10001];
vector<int> vis(10001);

int main() {
    int source = 1; // start at node 1

    queue<int> q;
    q.push(source); // start BFS with source
    vis[source] = 1; // mark as visited

    while(!q.empty()) {
        int curr = q.front();
        q.pop();

        for (int i = 0; i < adj[curr].size(); i++) {
            int next = adj[curr][i];
            if (!vis[next]) {
                vis[next] = 1;
                q.push(next);
            }
        }
    }
}
```

BFS is suited for finding the shortest path in unweighted graph.

Aside - Connected Components



Is this a graph?

Graphs can have multiple components that do not touch!

Graph Reduction

Molecule Interactions

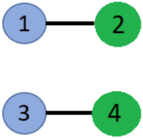
Problem: A pharmaceutical company has two sets of molecules and wants to test a hypothesis: whether or not each molecule only interacts with molecules from the other group. Output "Yes" if the hypothesis holds (there are no intra-interactions) or "No" otherwise.

Input: A single row containing `n`, the # of molecules, and `m`, the # of interactions. `m` rows will follow, each containing two numbers representing an interaction between molecules.

Example

Example 1

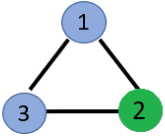
Input:
4 2
1 2
3 4



Output:
Yes

Example 2

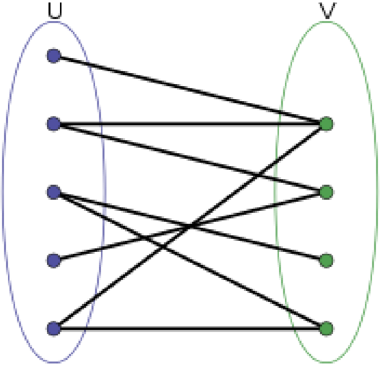
Input:
3 3
1 2
2 3
3 1



Output:
No

Observation

We can separate the two groups, circle them, and map each interaction. If we find that no two molecules of the same group have an interaction, we know the answer! This is a test of bipartiteness.



Theorem: A graph is bipartite iff it is two-colorable.

Prime Path

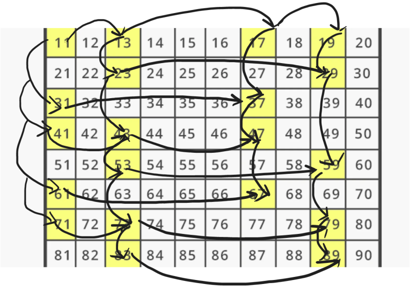
Given two 4-digit prime numbers `x` and `y`, what is the minimum amount of steps needed to change `x` into `y`? You can modify one digit each time to any digit of your choice, but each intermediate number also has to be prime.

Input: `1033 8179`
Output: `6`
Explanation: `1033` → `1733` → `3733` → `3739` → `3779` → `8779` → `8179`

Prime Path

Note: This is an example with 2-digit numbers

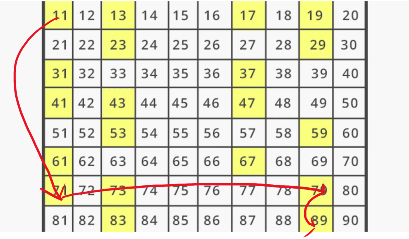
- Need to do two things first:
  - Find all prime numbers
  - Identify all those that are 1 change away



11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

Prime Path

Explore with BFS to find the shortest path



11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

A Little More to Think About

Algorithmic Use of BFS/DFS

What are some of the other ways we could use/modify BFS/DFS?

- Dijkstra
- Bellman-Ford
- Double-Ended Parallel