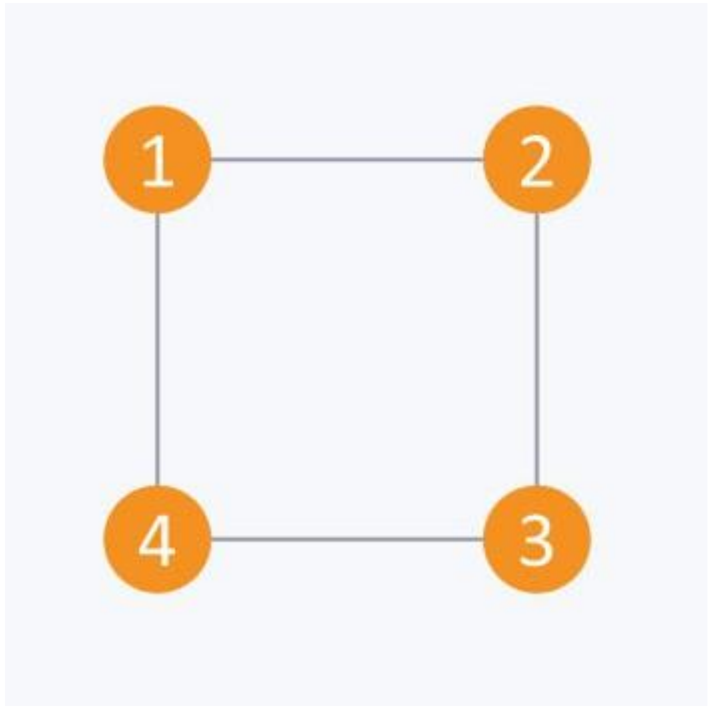
Graph representation using adjacency list

Presented by: Adarsh Singh

What is adjacency list?

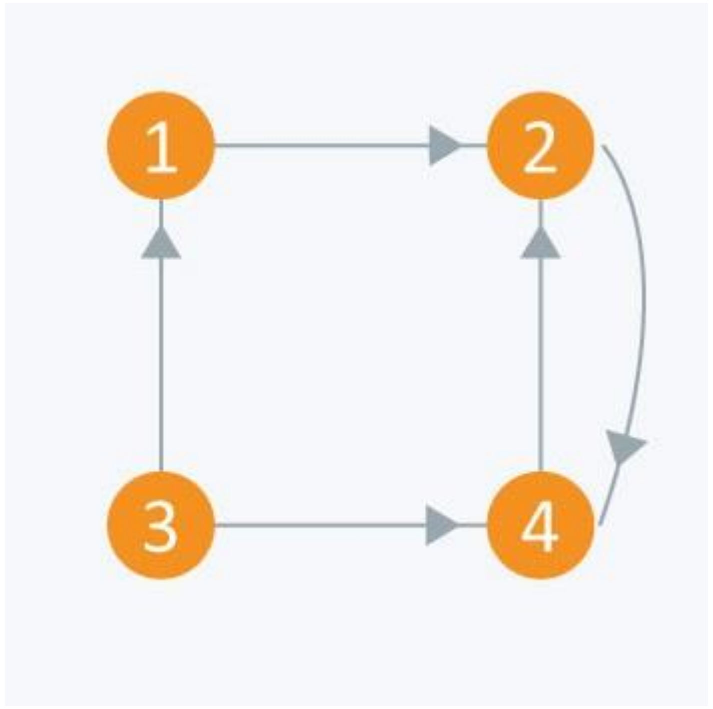
Adjacency list is an array A of separate lists. Each element of the array $\mathbf{A_i}$ is a list which contains all the vertices that are adjacent to vertex i. For weighted graph we can store weight or cost of the edge along with the vertex in the list using pairs. In an undirected graph, if vertex j is in list A_i then vertex i will be in list $\mathbf{A_j}$. Space complexity of adjacency list is **O(V + E)** because in Adjacency list we store information for only those edges that actually exist in the graph. In a lot of cases, where a matrix is sparse (A sparse matrix is a matrix in which most of the elements are zero. By contrast, if most of the elements are nonzero, then the matrix is considered dense.) using an adjacency matrix might not be very useful, since it'll use a lot of space where most of the elements will be 0, anyway. In such cases, using an adjacency list is better.

Adjacency list(undirected graph)

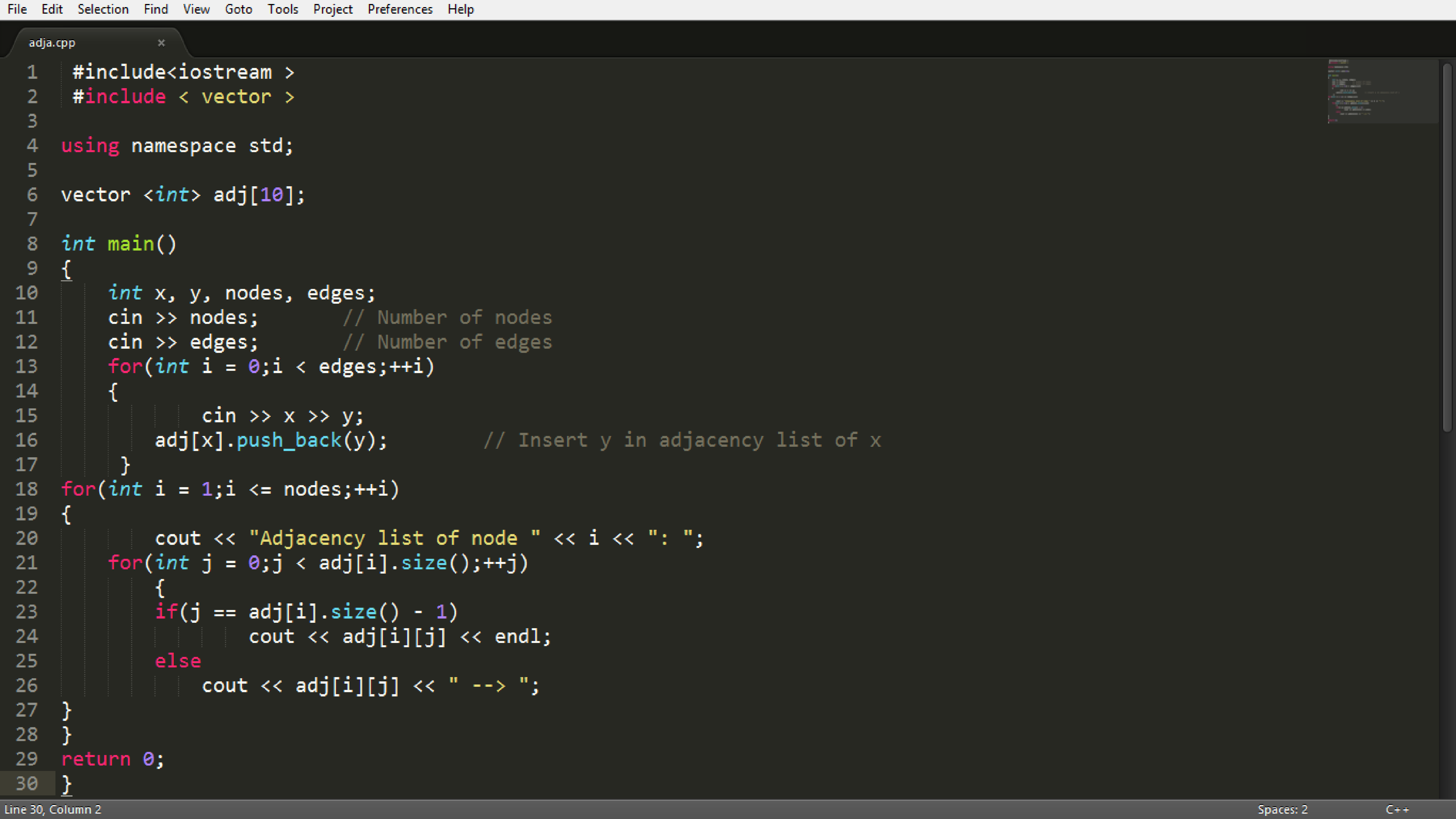


Adjacency list of the graph is:
A1 → 2 → 4
A2 → 1 → 3
A3 → 2 → 4
A4 → 1 → 3

Adjacency list(directed graph)



Adjacency list of the graph is:
A1 → 2
A2 → 4
A3 → 1 → 4
A4 → 2


File Edit Selection Find View Goto Tools Project Preferences Help
adja.cpp

```cpp
 #include<iostream >
 #include < vector >

using namespace std;

vector <int> adj[10];

int main()
{
    int x, y, nodes, edges;
    cin >> nodes;       // Number of nodes
    cin >> edges;       // Number of edges
    for(int i = 0;i < edges;++i)
    {
            cin >> x >> y;
        adj[x].push_back(y);        // Insert y in adjacency list of x
     }
for(int i = 1;i <= nodes;++i)
{
        cout << "Adjacency list of node " << i << ": ";
    for(int j = 0;j < adj[i].size();++j)
        {
        if(j == adj[i].size() - 1)
                cout << adj[i][j] << endl;
        else
            cout << adj[i][j] << " --> ";
}
}
return 0;
}
```

Line 30, Column 2
Spaces: 2
C++

Input:

```
4
5
1 2
2 4
3 1
3 4
4 2
```

Output:

```
Adjacency list of node 1: 2
Adjacency list of node 2: 4
Adjacency list of node 3: 1 --> 4
Adjacency list of node 4: 2
```

Thanks