Graph

LinkedList Concept, 内存中不一定连续的数据，由各个节点的reference串起来

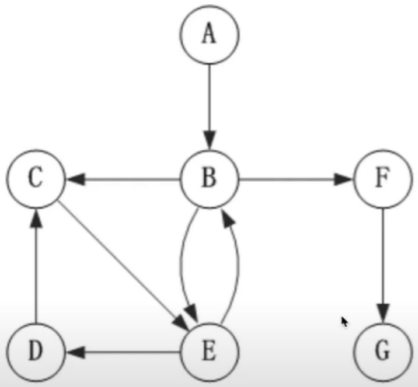
- Adjacency Matrix
  - O(V^2) allocate the space is large
  1. List<T>[n]
  2. Map<T, List<T>>
- Adjacency List

LinkedList

```
public class ListGraph {

    ArrayList<ArrayList<Integer>> graphs;

    public ListGraph(int v) {
        graphs = new ArrayList<>(v);
        for (int i = 0; i < v; i++) {
            graphs.add(new ArrayList<>());
        }
    }

    public void addEdge(int start, int end) {
        graphs.get(start).add(end);
    }

    public void removeEdge(int start, int end) {
        graphs.get(start).remove((Integer)end);
    }
}
```

DFS

深度优先遍历 (Depth-first Search)



- 我们假设初始状态所有顶点都没被访问，然后从每一顶点v出发，先访问该顶点
- 然后依次从它的各个未被访问的邻接点出发，深度优先遍历图，直到图中所有和v相通的顶点都被访问到。
- 遍历完后，还有其他顶点没被访问到，则另选一个未被访问的顶点作为起始点
- 重复上述过程，直到所有顶点都被访问完为止。

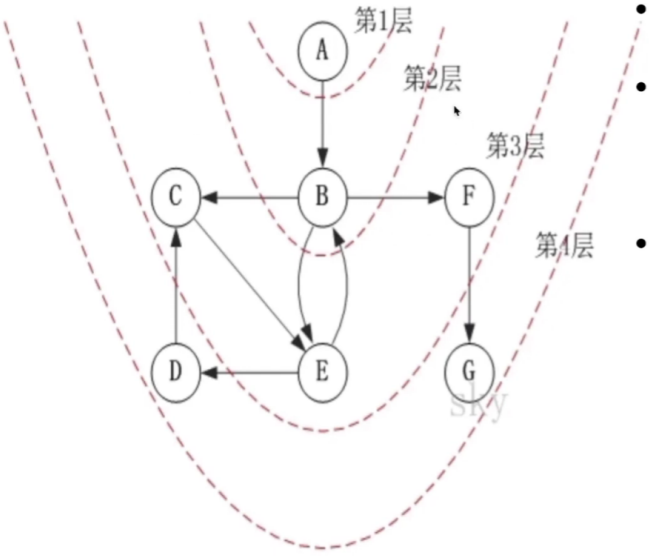
```
public class GraphTraversal {

    ListGraph graph;
    boolean[] visited;

    public GraphTraversal(ListGraph listGraph) {
        this.graph = listGraph;
        visited = new boolean[listGraph.graphs.size()];
    }

    public void DFSTraversal(int v) {
        if(visited[v]) return;
        visited[v] = true;
        System.out.print(v + " -> ");
        Iterator<Integer> neighbors = graph.graphs.get(v).listIterator();
        while (neighbors.hasNext()) {
            int nextNode = neighbors.next();
            if (!visited[nextNode]) {
                DFSTraversal(nextNode);
            }
        }
    }

    public void DFS() {
        for (int i = 0; i < graph.graphs.size(); i++) {
            if (!visited[i]) {
                DFSTraversal(i);
            }
        }
    }
}
```

访问顺序是：A -> B -> C -> E -> D -> F -> G。

BFS

```
public void BFSTraversal(int v) {
    Deque<Integer> queue = new ArrayDeque<>();
    visited[v] = true;
    queue.offerFirst(v);
    while (queue.size() != 0) {
        Integer cur = queue.pollFirst();
        System.out.print(cur + " -> ");
        Iterator<Integer> neighbors = graph.graphs.get(cur).listIterator();
        while (neighbors.hasNext()) {
            int nextNode = neighbors.next();
            if (!visited[nextNode]) {
                visited[nextNode] = true;
                queue.offerLast(nextNode);
            }
        }
    }
}

public void BFS() {
    for (int i = 0; i < graph.graphs.size(); i++) {
        if (!visited[i]) {
            BFSTraversal(i);
        }
    }
}
```

广度优先遍历 (Breadth-First Search)



- 从图中的某一顶点v出发，在访问了v之后依次访问v的各个没有访问到的邻接点
- 然后分别从这些邻接点出发依次访问他们的邻接点，使得先被访问的顶点的邻接点先与后被访问顶点的邻接点被访问，直到图中所有已被访问的顶点的邻接点都被访问到。
- 如果此时图中尚有顶点未被访问，则需要另选一个未曾被访问到的顶点作为新的起始点，重复上述过程

访问顺序是：A -> B -> C -> E -> F -> D -> G。