图的存储

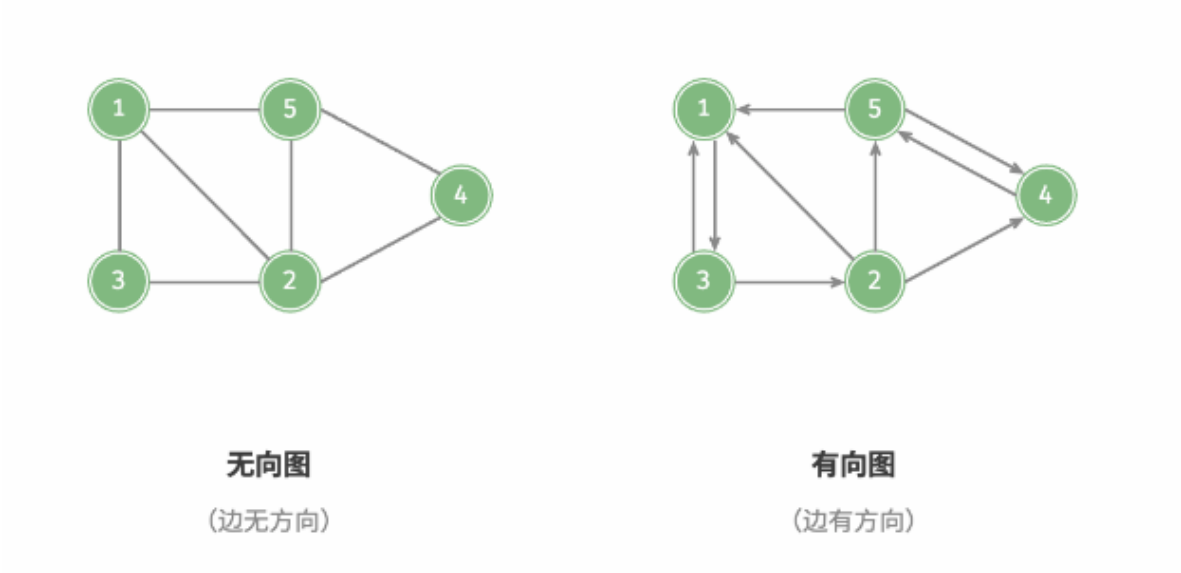
判定简单图：https://hydro.ac/d/shallowdream/p/35 模板

度的数量：https://hydro.ac/d/shallowdream/p/34 模板

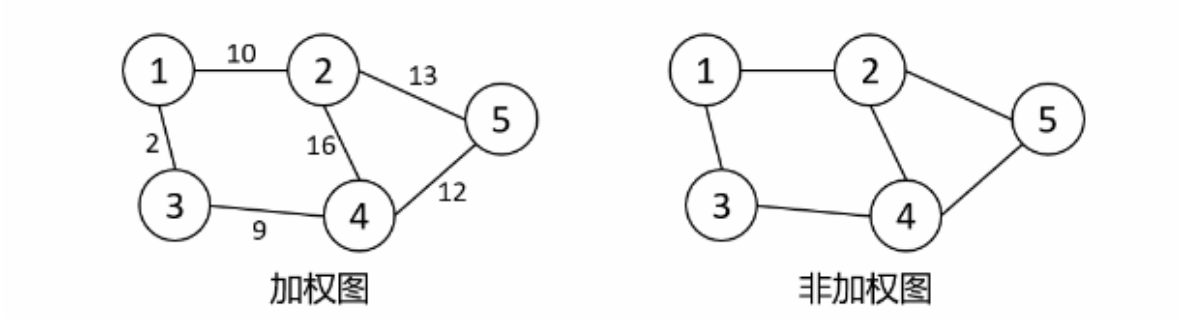
图的基本概念

一个图 G 由 **顶点（Vertex）** 的集合和 **边（Edge）** 的集合组成，记为 $G = (V, E)$，其中：

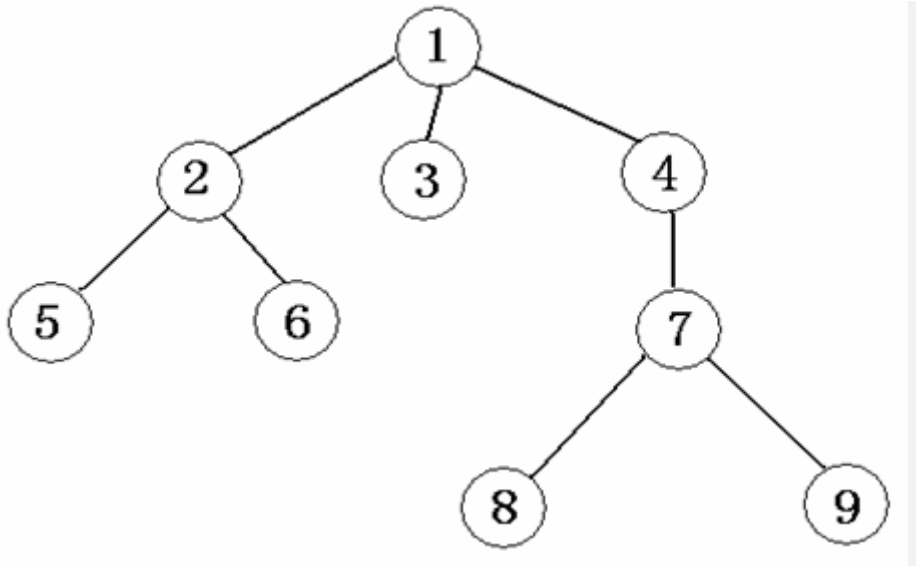
- V **（顶点集合）**：图中所有顶点的集合，通常记为 $V = v_1, v_2, \ldots, v_n$。
- E **（边集合）**：图中所有边的集合，表示顶点之间的连接关系，通常记为 $E = e_1, e_2, \ldots, e_m$。
- 无向图：边没有方向，表示顶点之间的双向关系。如果顶点 A 和顶点 B 之间有一条边，则可以从 A 到 B，也可以从 B 到 A。
- 有向图：边有方向，表示顶点之间的单向关系。顶点 A 到顶点 B 的边表示只能从 A 到 B，而不能从 B 到 A。一般无向图可以看做特殊的有向图，即 A, B 之间有一条边，表示 A 到 B 有一条有向边，B 到 A 有一条有向边。



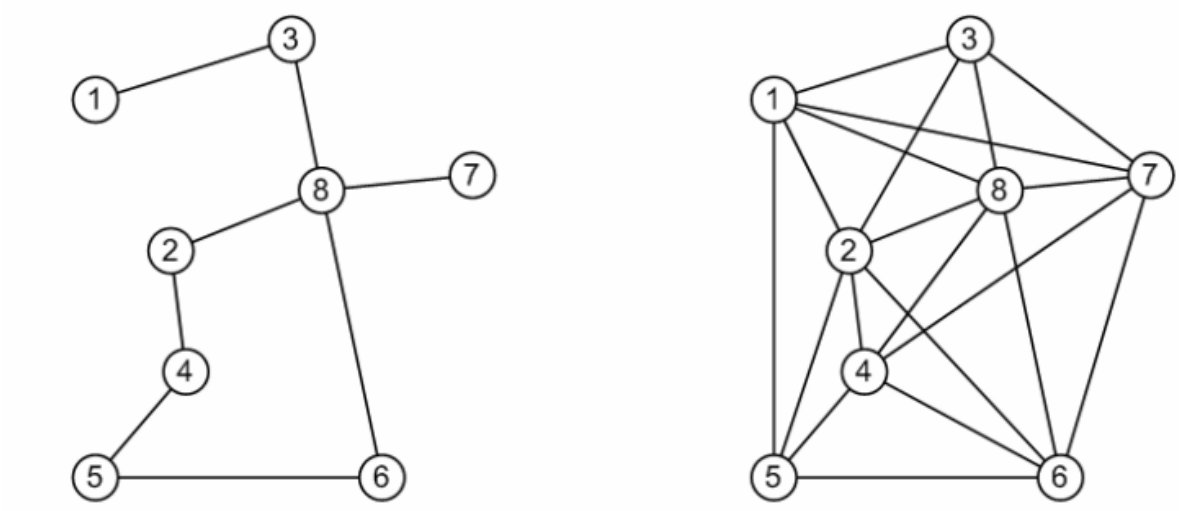
- 加权图：边具有权重（或权值），通常表示距离、时间、费用等。
- 无权图：边没有权重，所有边的权重默认为相同（通常可以认为是 1）。



- 树：是一种特殊的无向图，是一个无环连通图。树具有 n 个顶点 $n-1$ 条边。

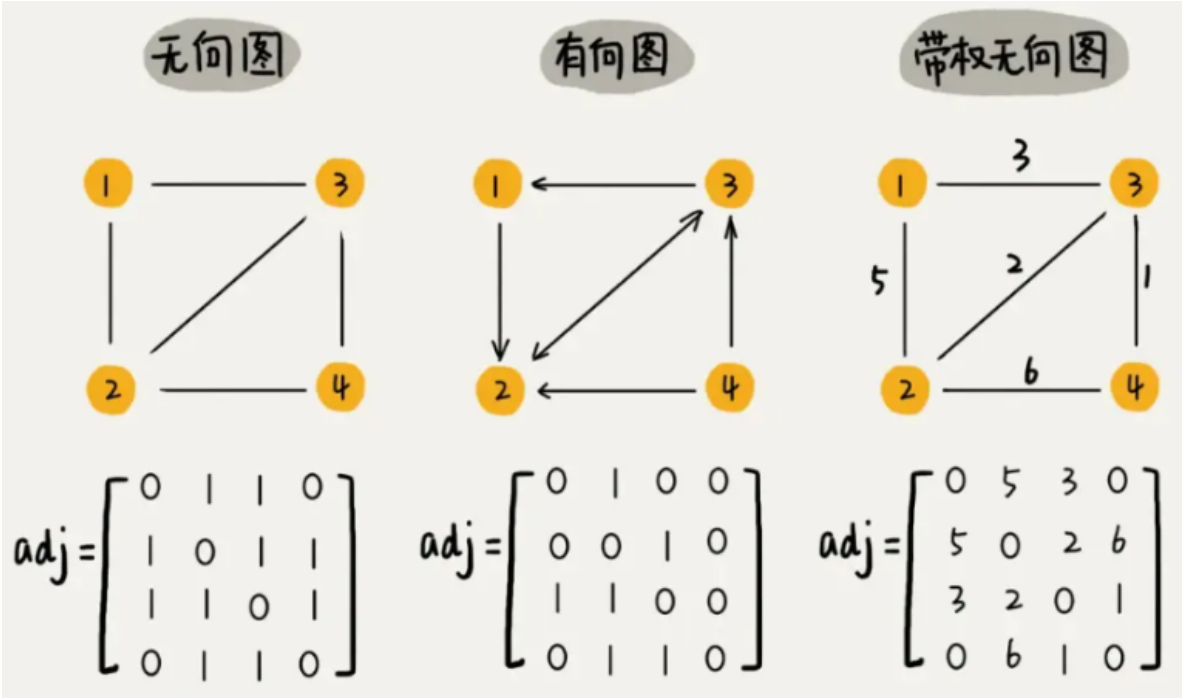


- **稀疏图**：边数接近点数。
- **稠密图**：边数远大于点数。



邻接矩阵存图

邻接矩阵，可以理解为是二维数组，一般用于存稠密图，初始时数组元素均为一个 (INF/0)，表示均互不可达。我们用 `g[a][b]=c`，代表点 a 到 b 有一条权值为 c 的有向边，若是无权图则用 `g[a][b]=1` 表示。




```cpp
#include <iostream>
#include <vector>
using namespace std;

const int N = 100; // 最大顶点数
const int INF = 1e9; // 设为无穷大，表示不可达
int g[N][N]; // 邻接矩阵

void initGraph(int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            g[i][j] = (i == j ? 0 : INF); // 自环设为 0，其他初始化为 INF
}

void addEdge(int a, int b, int c) {
    g[a][b] = c; // 有向带权图
    // g[b][a] = c; // 如果是无向图，则取消注释
}

void printGraph(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (g[i][j] == INF) cout << "INF ";
            else cout << g[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    int n = 4;
    initGraph(n);

    addEdge(0, 1, 4);
    addEdge(0, 2, 2);
    addEdge(1, 2, 5);
    addEdge(1, 3, 7);
```

```

    printGraph(n);
    return 0;
}

```

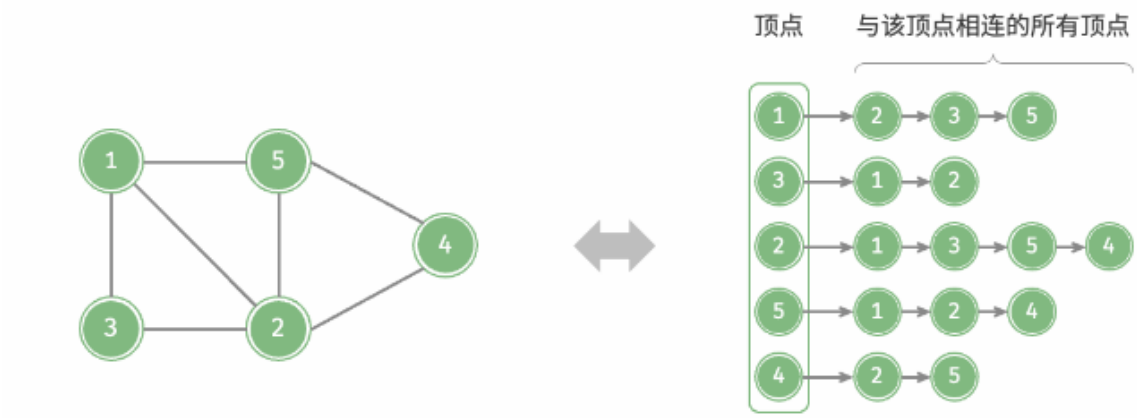
```java
import java.util.Arrays;

public class Main {
    static final int N = 100; // 最大顶点数
    static final int INF = (int) 1e9; // 设为无穷大，表示不可达
    static int[][] g = new int[N][N]; // 邻接矩阵

    public static void main(String[] args) {
        int n = 4;
        initGraph(n);

        addEdge(0, 1, 4);
        addEdge(0, 2, 2);
        addEdge(1, 2, 5);
        addEdge(1, 3, 7);

        printGraph(n);
    }

    static void initGraph(int n) {
        for (int i = 0; i < n; i++) {
            Arrays.fill(g[i], INF);
            g[i][i] = 0; // 自环设为 0
        }
    }

    static void addEdge(int a, int b, int c) {
        g[a][b] = c; // 有向带权图
        // g[b][a] = c; // 如果是无向图，则取消注释
    }

    static void printGraph(int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(g[i][j] == INF ? "INF " : g[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

```python
N = 100  # 最大顶点数
INF = float('inf')  # 设为无穷大，表示不可达
g = [[INF] * N for _ in range(N)]  # 邻接矩阵初始化

def init_graph(n):
    for i in range(n):
        g[i][i] = 0  # 自环设为 0

```

```python
def add_edge(a, b, c):
    g[a][b] = c  # 有向带权图
    # g[b][a] = c  # 如果是无向图，则取消注释

def print_graph(n):
    for i in range(n):
        print(" ".join("INF" if g[i][j] == INF else str(g[i][j]) for j in range(n)))

n = 4
init_graph(n)

add_edge(0, 1, 4)
add_edge(0, 2, 2)
add_edge(1, 2, 5)
add_edge(1, 3, 7)

print_graph(n)
```

邻接表存图

邻接表存图：定义一个数组，数组中每个元素都是一个扩容数组。




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

using namespace std;

const int N = 100; // 最大顶点数
vector<pair<int, int>> g[N]; // 邻接表，每个元素是 (目标顶点, 权重)

void addEdge(int u, int v, int w) {
    g[u].emplace_back(v, w);
    g[v].emplace_back(u, w); // 无向图
}

void printGraph(int n) {
    for (int i = 0; i < n; i++) {
        cout << "Vertex " << i << ": ";
        for (auto [v, w] : g[i]) {
            cout << "(" << v << ", " << w << ") ";
        }
```

```
        cout << endl;
    }
}

int main() {
    addEdge(0, 1, 4);
    addEdge(0, 2, 2);
    addEdge(1, 2, 5);
    addEdge(1, 3, 7);

    printGraph(4);
    return 0;
}

```

```
import java.util.*;

public class Main {
    static final int N = 100; // 最大顶点数
    static List<int[]>[] g = new ArrayList[N]; // 每个 List 里的元素是 (目标顶点, 权重)

    public static void main(String[] args) {
        for (int i = 0; i < N; i++) g[i] = new ArrayList<>();
        addEdge(0, 1, 4);
        addEdge(0, 2, 2);
        addEdge(1, 2, 5);
        addEdge(1, 3, 7);

        printGraph(4);
    }

    static void addEdge(int u, int v, int w) {
        g[u].add(new int[]{v, w});
        g[v].add(new int[]{u, w}); // 无向图
    }

    static void printGraph(int n) {
        for (int i = 0; i < n; i++) {
            System.out.print("Vertex " + i + ": ");
            for (int[] edge : g[i]) {
                System.out.print("(" + edge[0] + ", " + edge[1] + ") ");
            }
            System.out.println();
        }
    }
}

```

```
N = 100  # 最大顶点数
g = [[] for _ in range(N)]  # 邻接表，存 (目标顶点, 权重)

def add_edge(u, v, w):
    g[u].append((v, w))
    g[v].append((u, w))  # 无向图

```

```python
def print_graph(n):
    for i in range(n):
        print(f"Vertex {i}: {[(v, w) for v, w in g[i]]}")

add_edge(0, 1, 4)
add_edge(0, 2, 2)
add_edge(1, 2, 5)
add_edge(1, 3, 7)

print_graph(4)

```

判定简单图

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=5e2+10;
int g[N][N];
int n,m;
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
        int a,b;
        scanf("%d%d",&a,&b);
        if(a==b){
            printf("No\n");
            return 0;
        }
        if(g[a][b]==1) {
            printf("No\n");
            return 0;
        }
        g[a][b]=g[b][a]=1;
    }
    printf("Yes");
}
```

```java
import java.util.Scanner;

public class Main {
    static final int N = 502;
    static int[][] g = new int[N][N];

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int m = scanner.nextInt();

        for (int i = 0; i < m; i++) {
            int a = scanner.nextInt();
            int b = scanner.nextInt();

            if (a == b || g[a][b] == 1) {
                System.out.println("No");
                return;
            }
```

```
            g[a][b] = g[b][a] = 1;
        }

        System.out.println("Yes");
        scanner.close();
    }
}

```

```
import sys

N = 502
g = [[0] * N for _ in range(N)]

n, m = map(int, sys.stdin.readline().split())

for _ in range(m):
    a, b = map(int, sys.stdin.readline().split())

    if a == b or g[a][b] == 1:
        print("No")
        sys.exit()

    g[a][b] = g[b][a] = 1

print("Yes")

```

度的数量

```
#include<bits/stdc++.h>
using namespace std;
int n,m;
vector<int> g[1000005];
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    for(int i=1;i<=n;i++){
        cout<<g[i].size()<<" ";
    }
    return 0;
}
```

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    static int n, m;
    static ArrayList<Integer>[] g;

```

```java

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        m = scanner.nextInt();

        g = new ArrayList[n + 1]; // 由于节点编号从 1 开始，因此 n+1

        for (int i = 1; i <= n; i++) {
            g[i] = new ArrayList<>();
        }

        for (int i = 0; i < m; i++) {
            int x = scanner.nextInt();
            int y = scanner.nextInt();
            g[x].add(y);
            g[y].add(x);
        }

        for (int i = 1; i <= n; i++) {
            System.out.print(g[i].size() + " ");
        }

        scanner.close();
    }
}

```

```python
import sys

n, m = map(int, sys.stdin.readline().split())
g = [[] for _ in range(n + 1)]

for _ in range(m):
    x, y = map(int, sys.stdin.readline().split())
    g[x].append(y)
    g[y].append(x)

print(" ".join(str(len(g[i])) for i in range(1, n + 1)))

```