图的存储

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.16

目录

01

邻接矩阵

TEACHING
COURSEWARE

TEACH

概念引入

邻接矩阵是表达图中顶点与边关系的基础数据结构。
设图中有 n 个顶点，邻接矩阵是一个 n×n 的二维数组（记为 g），其元素 g[u][v] 的取值规则严格对应边的存在性与属性：

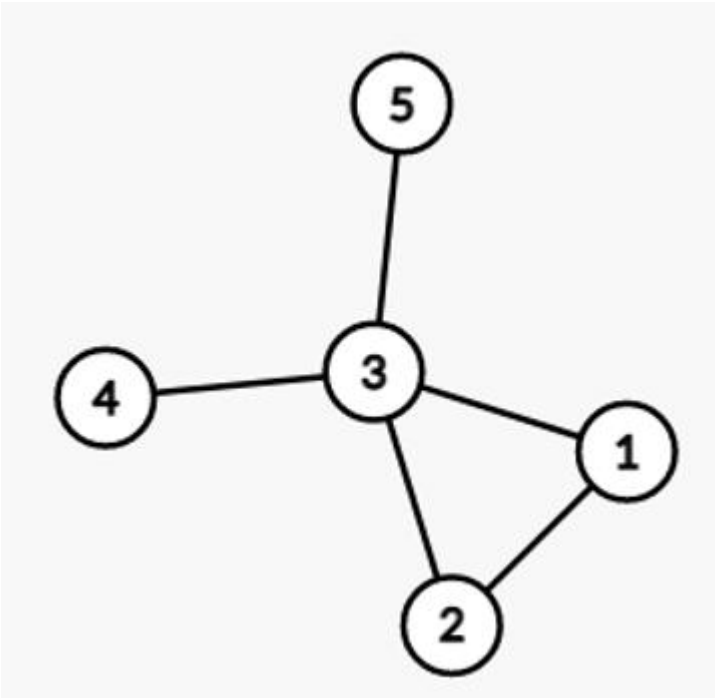
无向图：若顶点 u 和 v 之间有边，则 g[u][v] = g[v][u] = 1（或边的权重，如距离、成本）；若无边，则为 0（或无穷大 ∞，表示不可达）。

有向图：若存在从顶点 u 指向 v 的边，则 g[u][v] = 1（或边的权重）；若无该方向的边，则为 0（或 ∞），此时 g[u][v] 与 g[v][u] 无必然关联。

空间复杂度：$O(n^2)$，仅适合顶点数 n 较小的图（如 n ≤ 1000），若 n 达 1e4，矩阵会占用 1e8 个元素，超出内存限制。

查询效率：判断两点间是否有边、获取边的权重，仅需 $O(1)$ 时间（直接访问 g[u][v]），比邻接表更高效。

概念引入




```
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 0, 0, 1]
[0, 1, 1, 0]
```

```
[0, 1, 0]
[0, 0, 1]
[1, 0, 0]
```

02

实现

TEACHING
COURSEWARE

TEACH

实现

```cpp
#include<bits/stdc++.h>
using namespace std;
int g1[1005][1005]; // g1[u][v]=1表示u和v相连
int main() {
    int n, m;
    cin >> n >> m;  // 读取顶点数n和边数m
    // 初始化邻接矩阵为0（表示初始状态下所有顶点间无连接）
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            g1[i][j] = 0;
        }
    }
    // 读取m条边，更新邻接矩阵和邻接表
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;  // 读取一条边的两个顶点
        // 无向图：更新邻接矩阵，u和v、v和u均标记为相连
        g1[u][v] = 1;
        g1[v][u] = 1;
    }
```

实现


```
    // 输出邻接矩阵：n行n列，每行元素用空格分隔
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            cout << g1[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

03

邻接表

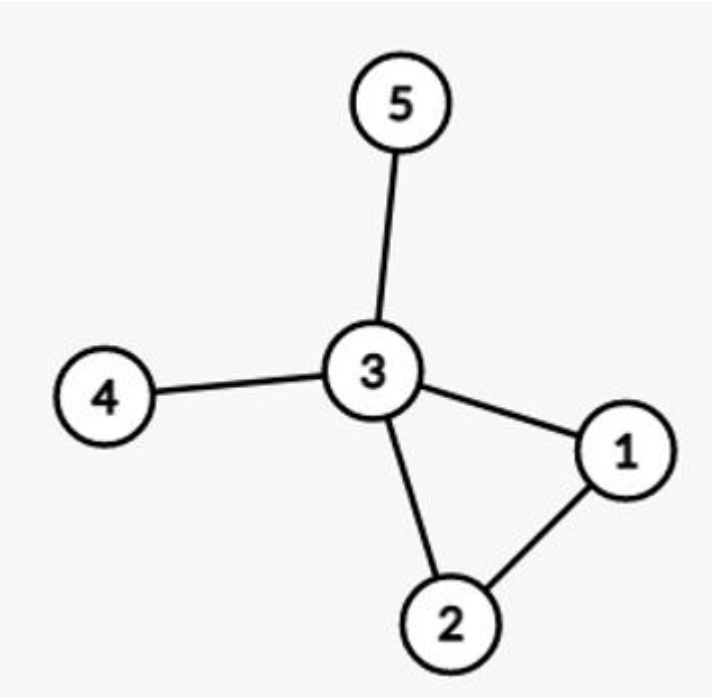
TEACHING
COURSEWARE

TEACH

邻接表

邻接表通常用vector 容器嵌套实现，核心结构为：
外层是一个vector（数组），长度等于顶点数n，下标u对应图中的顶点u
内层每个元素也是vector<int>，直接存储与顶点u相邻的所有顶点编号
对于不同类型的图，构建规则如下：
无向图：若存在边u-v，则在 g[u]中push_back(v)，同时在 g[v]中push_back(u)
有向图：若存在边u→v，则仅在 g[u]中push_back(v)（体现方向特性）
1. 关键特性
空间复杂度：O(n + m)，n为顶点数，m为边数。对于n=1e5级别的大规模图，远优于邻接矩阵的$O(n^2)$
时间效率：
遍历顶点u的所有邻接点：O(g[u].size())
判断u和v是否直接相连：需遍历 g[u]查找v，时间复杂度O(g[u].size())
实现优势：
无需预处理大小，vector自动动态扩容

邻接表



邻接表


```
vector<int> g[3];
g[0].push_back(1);  // 0连接1
g[1].push_back(0);  // 1连接0（无向图双向添加）
g[1].push_back(2);  // 1连接2
g[2].push_back(1);  // 2连接1（无向图双向添加）

vector<int> g2[3];
g2[0].push_back(1);     // 0指向1
g2[1].push_back(2);     // 1指向2
g2[2].push_back(0);     // 2指向0
```

04

实现

TEACHING
COURSEWARE

TEACH

实现


```cpp
#include<bits/stdc++.h>
using namespace std;
vector<int> g2[1005]; // 邻接表，，g2[u]存储与u相连的所有顶点
int main() {
    int n, m;
    cin >> n >> m;  // 读取顶点数n和边数m
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;  // 读取一条边的两个顶点
        // 无向图：更新邻接表，在u的列表中加入v，在v的列表中加入u
        g2[u].push_back(v);
        g2[v].push_back(u);
    }
```

实现


```
    // 输出邻接表：先排序，再输出顶点度数和所有邻接顶点
    for (int i = 1; i <= n; ++i) {
        // 对当前顶点的邻接列表按升序排序
        sort(g2[i].begin(), g2[i].end());
        // 输出顶点的度数（邻接列表的大小）
        cout << g2[i].size();
        // 依次输出所有邻接顶点
        for (int j = 0 ; j < g2[i].size() ; j++) {
            int v = g2[i][j];
            cout << " " << v;
        }
        cout << "\n";
    }
    return 0;
}
```

实现

以下哪个结构可以用来存储图（）

- A. 栈
- B. 二叉树
- C. 队列
- D. 邻接矩阵

实现

考虑由 N 个顶点构成的有向连通图，采用邻接矩阵的数据结构表示时，该矩阵中至少存在（）个非零元素。

- A. $N-1$
- B. N
- C. $N+1$
- D. N^2