
# 数据结构与算法

## 第 11 讲：图

韩文弢

清华大学

2024—2025 学年度春季学期

# 本讲内容

# 11.1 问题引入

# 11.1.1 逻辑结构

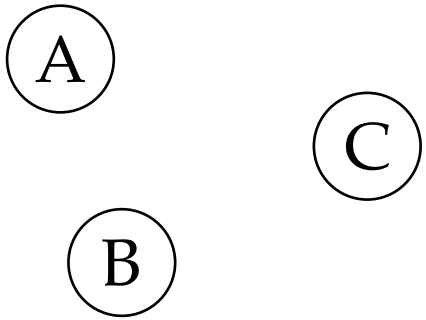



图 11.1　集合结构

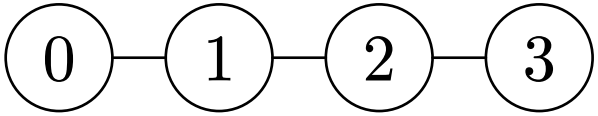



图 11.2　线性结构

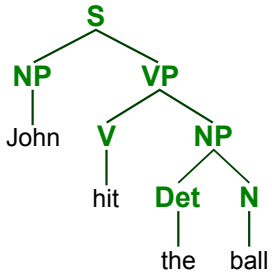



图 11.3　层次结构

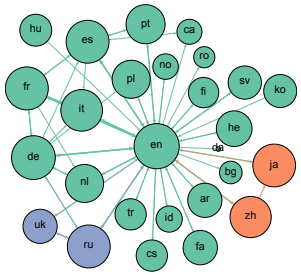
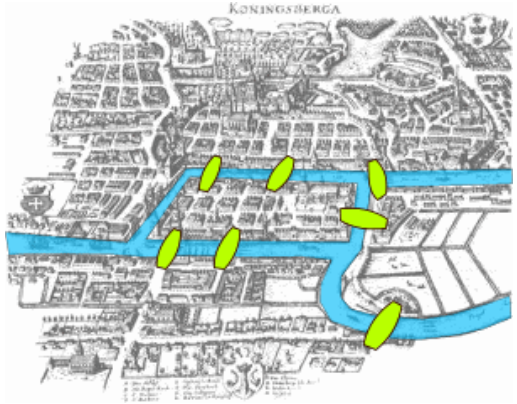



图 11.4　网络结构

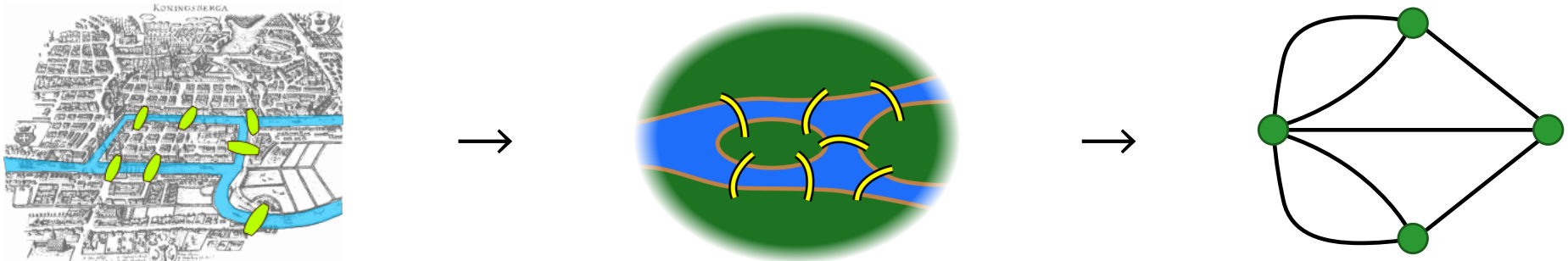
# 11.1.2 七桥问题




**哥尼斯堡七桥问题**：18 世纪东普鲁士的哥尼斯堡，有一条河穿城而过，将城市分为几个部分。河上有七座桥将这些部分连接起来。人们提出问题：能否从某个部分出发，经过全部的七座桥，且每座桥只走一遍，最后还能回到出发点?

# 11.1.3 欧拉的解法

1736 年，29 岁的数学家欧拉（Leonhard Euler）解决了这个问题：将陆地的每个部分用点表示，桥用点之间的边表示，这样七桥问题就可以抽象成一种图模型。问题由此转化为从图中的任意一个点出发，是否存在一条路径，能恰好经过每条边一次后回到原点?



经过研究，欧拉给出了结论（欧拉定理），并开创了数学的一个新的分支：**图论**。

# 11.2 图的概念

# 11.2.1 图的定义

**定义 1 图**（graph） $G = (V, E)$ 是一个二元组，其中 $V$ 是**顶点**（vertex，复数 vertices）集合，$E$ 是**边**（edge）的集合。

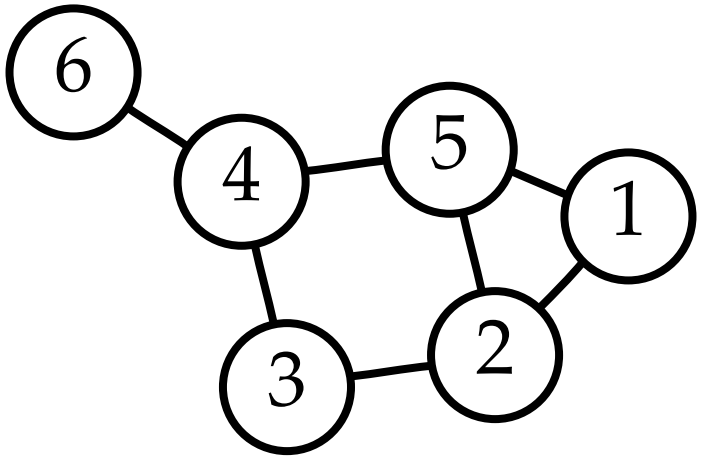



图 11.6　图的示例

# 11.2.2 有向图与无向图

**定义 2** 如果边具有方向性，把这样的图称为**有向图**（directed graph），边称为**有向边**或**弧**。有向边一般记为 $e = (u, v)$。

如果边不具有方向性，把这样的图称为**无向图**（undirected graph），边称为**无向边**。无向边一般记为 $e = \{u, v\}$。

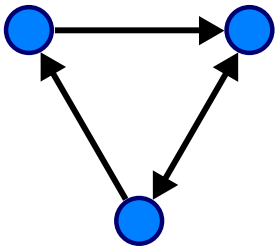


图 11.7　有向图示例

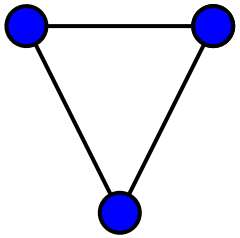


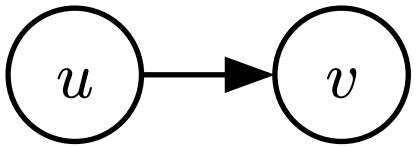
图 11.8　无向图示例

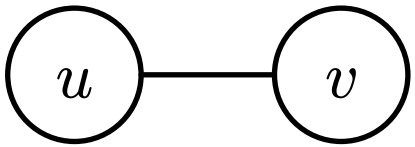
# 11.2.3 邻接关系

**定义 3** 图的顶点之间有边相连，称顶点间有**邻接**（adjacent）关系。

若 $e = (u, v)$ 是一条有向边，则称顶点 $u$ 是边 $e$ 的**起点**，顶点 $v$ 是边 $e$ 的**终点**，顶点 $u$ 邻接到顶点 $v$，顶点 $v$ 邻接自顶点 $u$。

若 $e = \{u, v\}$ 是一条无向边，则称顶点 $u$ 和 $v$ 都是边 $e$ 的**端点**，顶点 $u$ 和 $v$ 是邻接的。

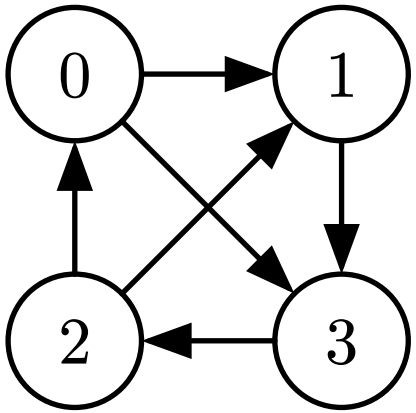


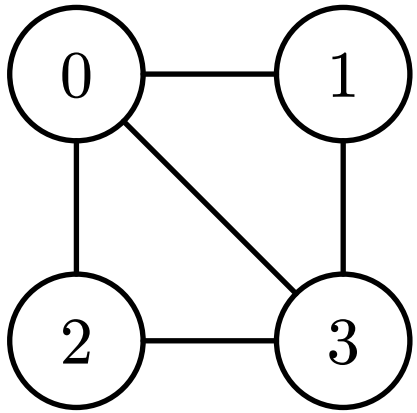



# 11.2.4 顶点的度

**定义 4** 有向图中，一个顶点的**出度**是从这个顶点发出的边的条数，**入度**是指向这个顶点的边的条数。

无向图中，一个顶点的**度**是以这个顶点为端点的边的条数。



# 11.2.5 简单图与多重图

**定义 5** 若一个图中任意两个顶点之间最多只有一条边（有向图可以有正反方向的两条边），也不包含自边（即某个顶点连接它自己的边），则称这样的图为**简单图**，否则称为**多重图**。

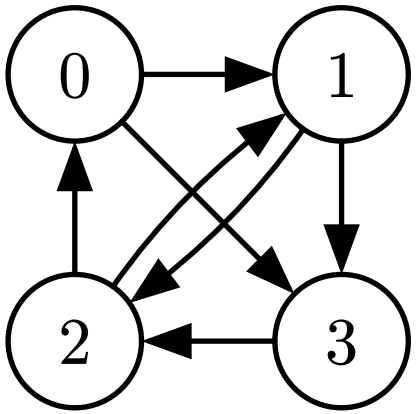



图 11.11　简单图

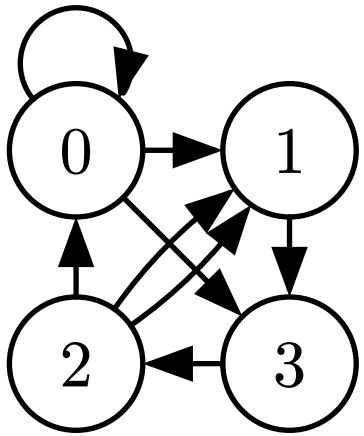



图 11.12　多重图

# 11.2.6 完全图

**定义 6** 有 $n$ 个顶点的有向图，边数最多为 $n(n-1)$，称为**有向完全图**。类似的，有 $n$ 个顶点的无向图，边数最多为 $\binom{n}{2} = \frac{1}{2}n(n-1)$，称为**无向完全图**。

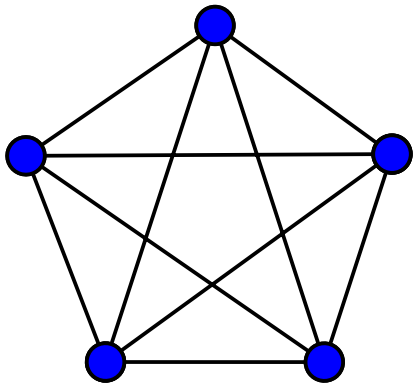


图 11.13　无向完全图示例（$n = 5$），记作 $K_5$

# 11.2.7 加权图

**定义 7** 图的边上可以带有一定的**权重**（weight），称为**加权图**（weighted graph），也称为**网络**（network）。

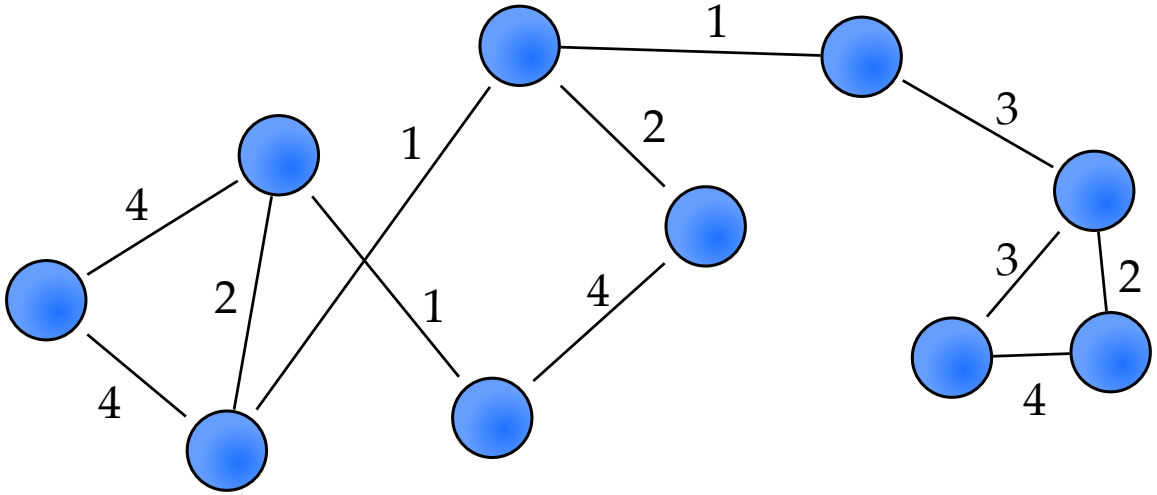



图 11.14　加权图示例

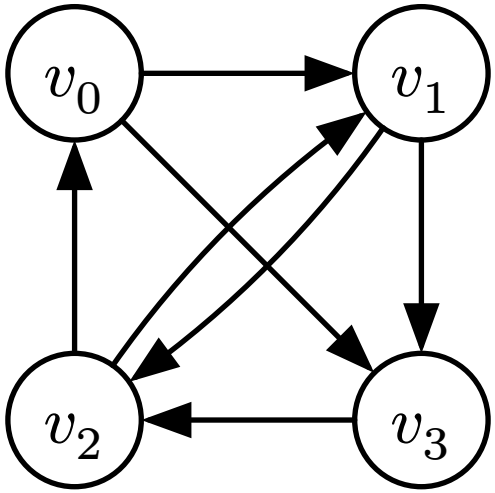
# 11.2.8 路径与环

**定义 8** 对于图中任意两个顶点 $u$ 和 $v$，如果可以从 $u$ 出发，经过若干条边（有向边需要考虑方向性）到达 $v$，则称 $u$ 到 $v$ 之间存在一条**路径**（path），经过的边数称为这条路径的长度。对于加权图，路径的长度也可以是这些边的权重之和。

如果一条路径除了第一个顶点和最后一个顶点可能相同之外，其余的顶点都不相同，称这样的路径为**简单路径**。

第一个顶点和最后一个顶点相同的路径称为**环**（cycle），也称为**回路**。

# 11.2.9 路径与环示例




例. 上图中，$v_0, v_1, v_2$ 是一条路径，也是简单路径；$v_0, v_1, v_2, v_1$ 是一条路径，但不是简单路径；$v_0, v_1, v_2, v_0$ 是一个环（回路）。

# 11.2.10 子图

**定义 9** 设有两个图 $G = (V, E)$ 和 $G' = (V', E')$，若 $V' \subseteq V$ 且 $E' \subseteq E$，则称 $G'$ 为 $G$ 的**子图**（subgraph）。

根据定义，图 $G$ 是其自身的子图。

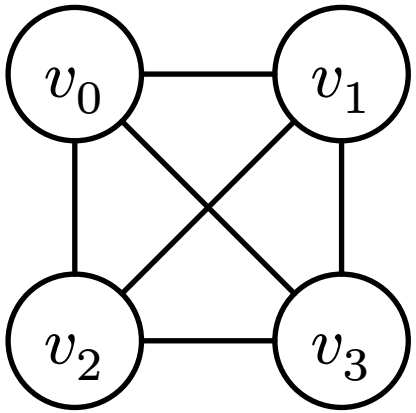



图 11.16　图 $G$

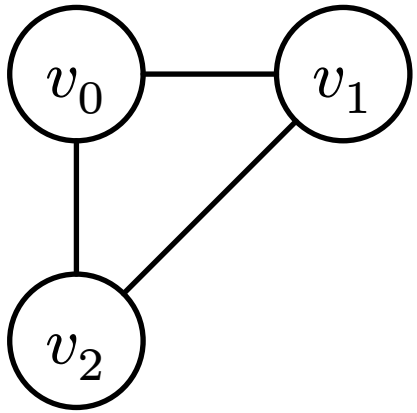



图 11.17　图 $G$ 的一个子图 $G'$

# 11.2.11 连通性

**定义 10** 在一个图中，如果顶点 $u$ 到 $v$ 之间存在路径，称顶点 $u$ 到 $v$ 是**连通**的（connected）。

在一个无向图 $G$ 中，如果任意两个顶点之间都是连通的，称 $G$ 为**连通图**。

如果在一个无向图 $G$ 中存在一个子图 $G'$，向 $G'$ 中再添加一个 $G$ 中的顶点（非 $G'$ 中的顶点）会造成 $G'$ 不连通，且子图 $G'$ 包含了其中顶点之间所有的边，则称 $G'$ 为 $G$ 的一个**极大连通子图**。无向图的极大连通子图称为**连通分量**。

# 11.2.12 连通分量示例

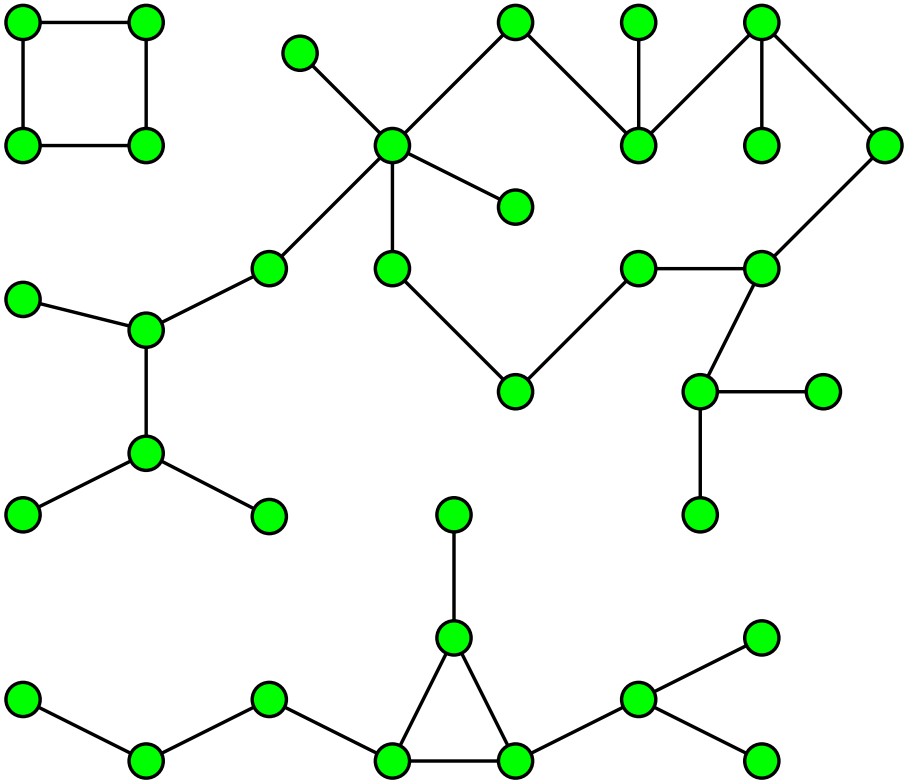


图 11.18　有三个连通分量的图

# 11.3 图的表示

# 11.3.1 图的存储需要考虑的因素

图中既有顶点也有边，图的存储既要考虑顶点的存储，也要考虑边的存储。以下内容以有向图为例，无向图可将无向边视为正反两条有向边。

图中顶点相关数据可以用线性表来存储。对于边而言，

- 可以看做每两个顶点之间的状态，用矩阵来表示，称为**邻接矩阵**。
- 可以看做每个顶点发出的边的集合，用两层的线性表来表示，称为**邻接表**。
- 也可以看做边的集合，用一层的线性表来表示，称为**边表**（不常用）。

## 11.3.2 邻接矩阵

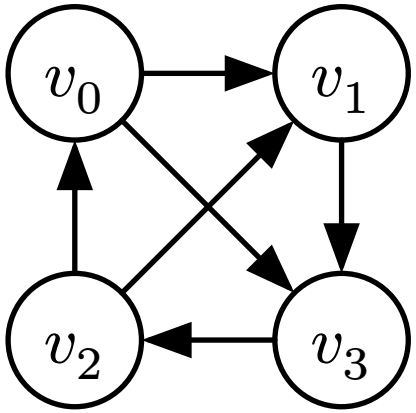
对于一个图 $G=(V,E)$，记顶点数 $|V|=n$，边数 $|E|=m$。设 $V=\{v_0,v_1,\ldots,v_{n-1}\}$。将图 $G$ 中的边看做是每两个顶点之间的状态，可以用矩阵 $A$ 来表示，

$$a_{ij}=\begin{cases}0\text{，当 }(v_i,v_j)\notin E\\1\text{，当 }(v_i,v_j)\in E\end{cases}$$

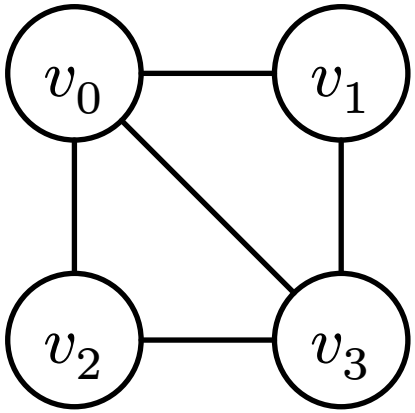
矩阵 $A$ 称为图 $G$ 的邻接矩阵。

当 $G$ 为加权图时，可以令 $a_{ij}=w_{ij}$，其中 $w_{ij}$ 为边 $(v_i,v_j)$ 的权重。如果边 $(v_i,v_j)$ 不存在，可根据情况令 $w_{ij}$ 为 0 或 $\infty$ 等值。

# 11.3.3 邻接矩阵示例




$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

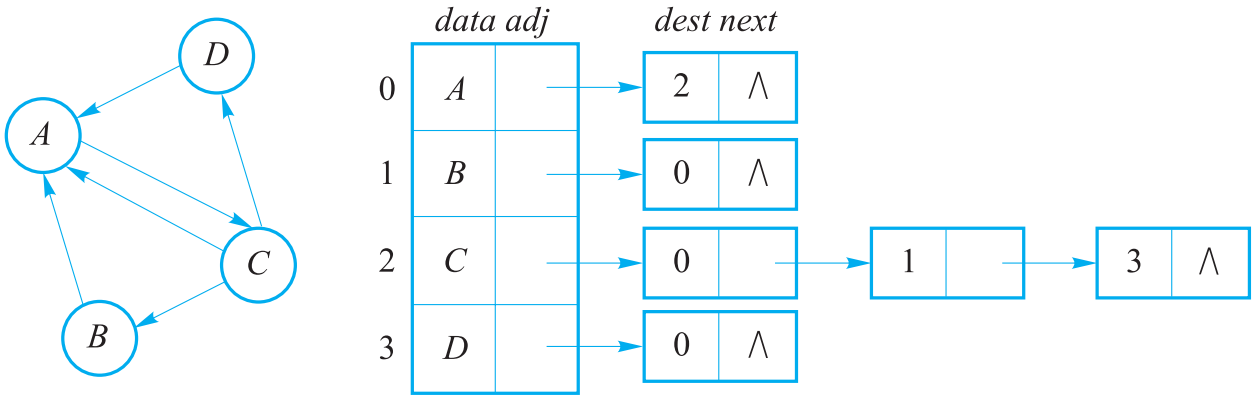



$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

无向图的邻接矩阵是**对称矩阵**。

# 11.3.4 邻接表

邻接矩阵需要使用 $O(n^2)$ 的空间，即便边数很少也是如此。此时，邻接矩阵有大量的零元素，造成空间浪费。为了节省空间，对每个顶点可以只存储发出的边，称为邻接表。邻接表需要的空间为 $O(n+m)$。




有向图$G$　　$G$的邻接表

# 11.3.5 两种表示的对比

| 性质 | 邻接矩阵 | 邻接表 |
| --- | --- | --- |
| 空间复杂度 | $O(n^2)$ | $O(n+m)$ |
| 判断边是否存在 | $O(1)$ | $O(d)$ |
| 访问某个顶点的所有边 | $O(n)$ | $O(d)$ |

其中，$d$ 为相关顶点的度数。由上表可知，

- 邻接矩阵适合**稠密图**，以及需要经常判定某两个顶点之间是否有边的场景。
- 邻接表适合**稀疏图**，以及对每个顶点的所有边进行访问的场景。

# 11.4 图的遍历

# 11.4.1 图的遍历的定义

图的遍历是指按一定顺序访问图中的每个顶点。

与二叉树的遍历类似，图的遍历一般也要考虑边的关系。

图的遍历与二叉树不同之处在于图中可能有环，会导致重复访问。

考虑游戏中走迷宫的过程：

1. 来到一个路口（顶点）
2. 选择这个路口还没走过的下一条路（边）
   - 如果新的路口未曾到达，则将其作为当前路口，转 1
   - 否则该路口已经到过，退回上一个路口，转 2
3. 如果当前路口连接的路都走过了，也退回上一个路口，转 2

# 11.4.2 深度优先遍历算法

上述思路称为**深度优先遍历**（depth-first search，DFS）。

**伪码 11.1:** 图的深度优先遍历

```
DFS(u):
    if visited[u]   # 若已经访问过则退回
        return
    visited[u] ← true
    输出访问顶点 u 的信息
    for v ∈ N(u)   # N(u) 是与 u 邻接的顶点集合
        DFS(v)
```

# 11.4.3 深度优先遍历的外层算法

对于可能不连通的图，要在 DFS 外面套一层循环。

**伪码 11.2:** 图的遍历外层算法

```
DFSTraverse(G):
1   for v ∈ V
2       visited[v] ← false   # 设置未访问标志
3   for v ∈ V
4       if ¬ visited[v]   # 若顶点 v 未访问
5           DFS(v)   # 以顶点 v 为起始顶点进行遍历
```

# 11.4.4 深度优先遍历的时间复杂度分析

由于递归调用，直接分析 DFS 的时间复杂度有困难。假设图是连通的，采用邻接表表示。尝试分析伪码每行一共的执行次数。

**伪码 11.3:** 图的深度优先遍历

```
DFS(u):
    if visited[u]   # 一共执行 m 次
        return
    visited[u] ← true   # 一共执行 n 次
    输出访问顶点 u 的信息
    for v ∈ N(u)   # 一共执行 m 次
        DFS(v)
```

# 11.4.5 深度优先遍历的时间复杂度

在邻接表表示的图上进行深度优先遍历的时间复杂度为 $O(n+m)$。

如果使用邻接矩阵，则时间复杂度为 $O(n^2)$。

深度优先遍历可用于判定两个顶点之间是否连通，此外还有很多应用，例如：寻找连通分支、拓扑排序、平面图测试。

# 11.4.6 深度优先遍历的实现

C++

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

class Graph {
 public:
  // 输入一个图
  Graph() {
    int n, m;
    std::cin >> n >> m;
    adj_.resize(n);
    for (int i = 0; i < m; i++) {
      int u, v;
      std::cin >> u >> v;
      adj_[u].push_back(v);
```

# 11.4.6 深度优先遍历的实现

```
15      }
16    }
17    // 对全图进行深度优先遍历
18    void DFSTraverse() {
19      int n = adj_.size();
20      visited_.assign(n, 0);
21      for (int v = 0; v < n; v++) {
22        if (!visited_[v]) {
23          DFS(v);
24          std::cout << '\n';
25        }
26      }
27    }
28    // 从顶点 u 出发进行一趟深度优先遍历
```

# 11.4.6 深度优先遍历的实现

```
29    void DFS(int u) {
30      if (visited_[u]) {
31        return;
32      }
33      visited_[u] = 1;
34      std::cout << u << ' ';
35      for (int v : adj_[u]) {
36        DFS(v);
37      }
38    }
39
40  private:
41    std::vector<std::vector<int>> adj_;  // 邻接表
42    std::vector<int> visited_;           // 访问标志
```

# 11.4.6 深度优先遍历的实现

```
43    // 注意 std::vector<bool> 是按位压缩存储的，有性能问题
44  };
45
46  int main() {
47    Graph g;
48    g.DFSTraverse();
49  }
```

# 11.5 小结

# 11.5.1 本讲小结

- 图的概念：顶点、边、路径、子图、连通性
- 图的表示：邻接矩阵和邻接表
- 图的遍历：深度优先遍历