# 题目讲解

- 设计一个队列/栈
- 支持：出队/栈，入队/栈，求最大元素
- 要求O(1)
- 均摊分析
- 要点：
  - 队列：新开一个辅助队列，维护其单调性
  - 栈：新开一个辅助栈，维护某个元素之前的最大元素

# 题目讲解

- Leetcode 128. Longest Consecutive Sequence
- 构建hashset，删除当前元素，判断当前元素的-1，+1是否在hashset中，向两端扩展，直到不能扩展
- 关键：Hashset删除，均摊分析

# 图论选讲

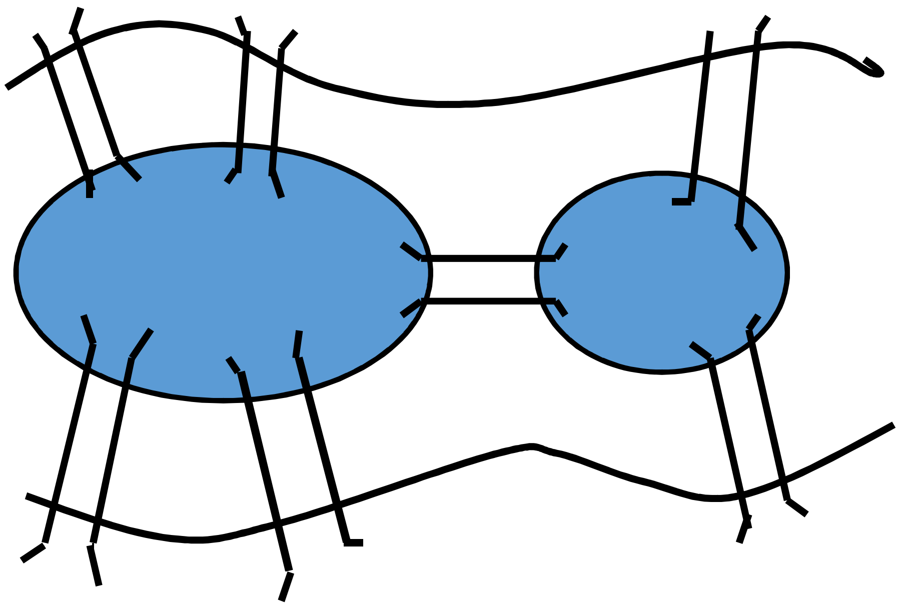
Ben

# Outline

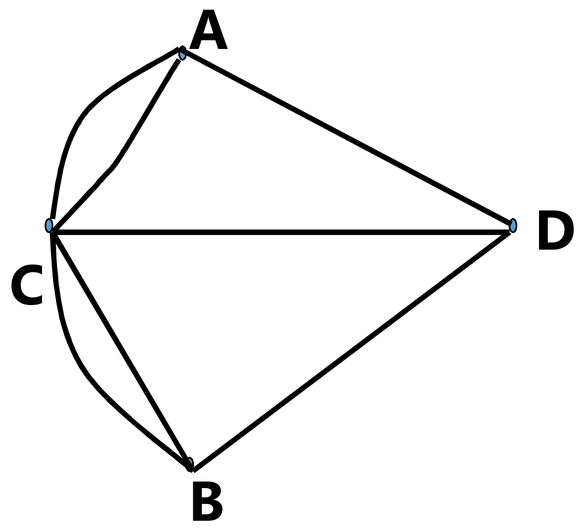
- 图定义
- 拓扑排序
- 最短路（Dijkstra，Floyed)
- 最小生成树

# 图：定义

哥尼斯堡“七桥”问题



一笔画问题



# 图：定义

- 描述事物之间的关系
- 结点集 V = {v1, v2, ..., vn}
- 边集合 E = {e1, e2, ..., em}, 其中 ei = ( vi, vi’ )
- G = <V, E>
- 有向图 ← 无向图
- 空间复杂度：$O(n + m)$ 或 $O(n^2)$
- 邻接矩阵 邻接表

# 图：例子

四个城市：v1、v2、v3、v4，其中v1与v2间，v1与v4间，v2与v3间有直达高速公路**相连（无向图）**

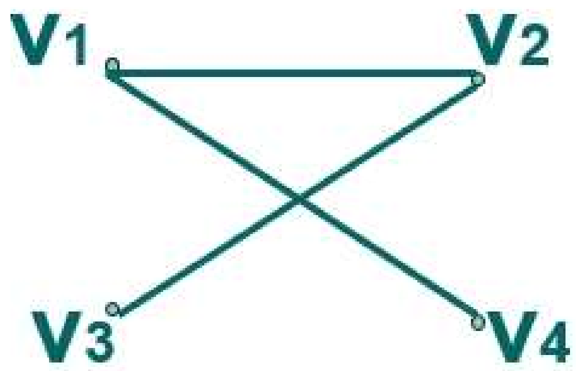
G = <V, E>　　V = (v1, v2, v3, v4)

E = (e1, e2, e3)

e1 = (v1, v2)

e2 = (v1, v4)

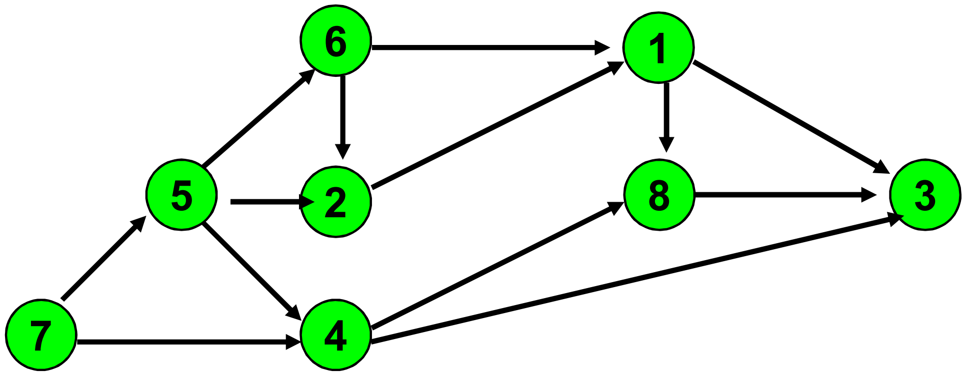
e3 = (v2, v3)



# 拓扑排序：定义

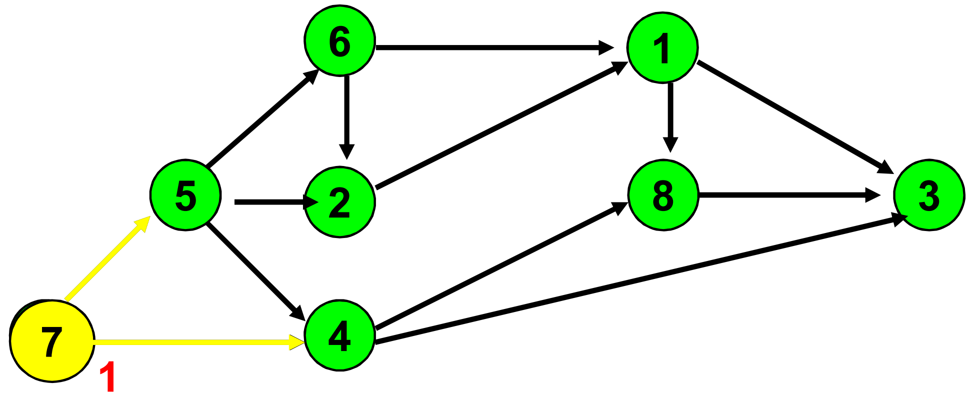
- 有向无环图（DAG）
- 场景：任务依赖
- 时间复杂度 O(n + m)
- 附加空间复杂度 O(n)
- 每次找入度为0的点
- 维护入度

# 拓扑排序：过程



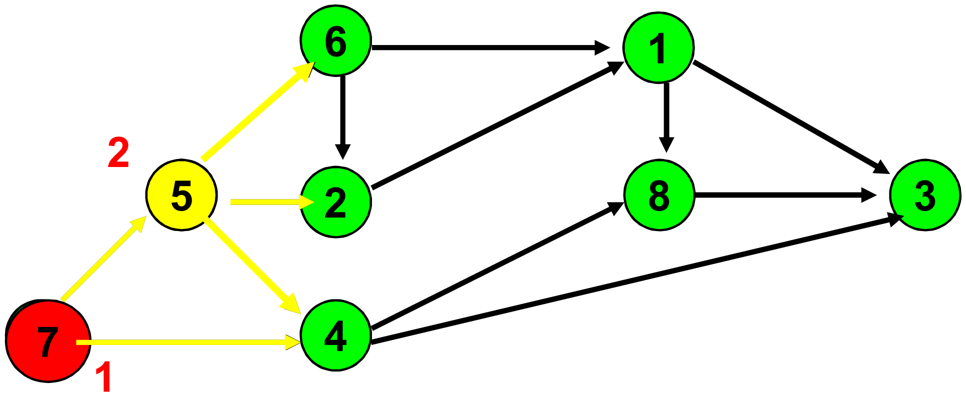

| 结点 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 入度 | 2 | 2 | 3 | 2 | 1 | 1 | 0 | 2 |

# 拓扑排序：过程



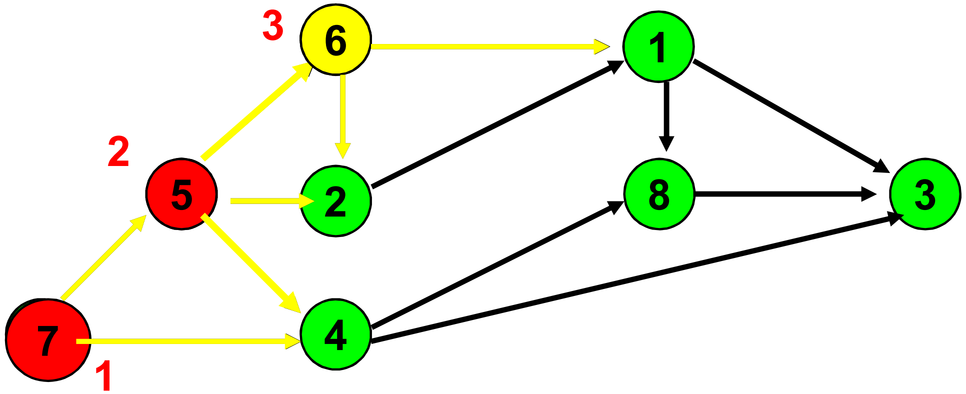

| 结点 | 1 | 2 | 3 | 4 | 5 | 6 | | 8 |
|---|---|---|---|---|---|---|---|---|
| 入度 | 2 | 2 | 3 | 1 | 0 | 1 | | 2 |

# 拓扑排序：过程



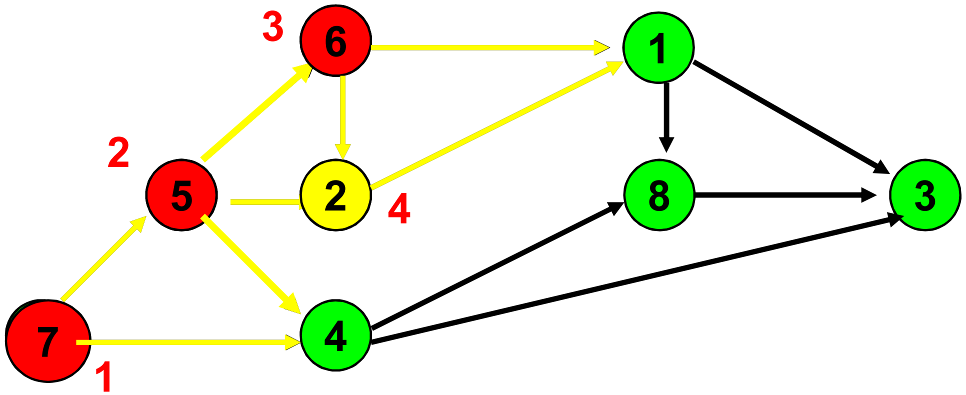

| 结点 | 1 | 2 | 3 | 4 | | 6 | | 8 |
|---|---|---|---|---|---|---|---|---|
| 入度 | 2 | 1 | 3 | 0 | | 0 | | 2 |

# 拓扑排序：过程



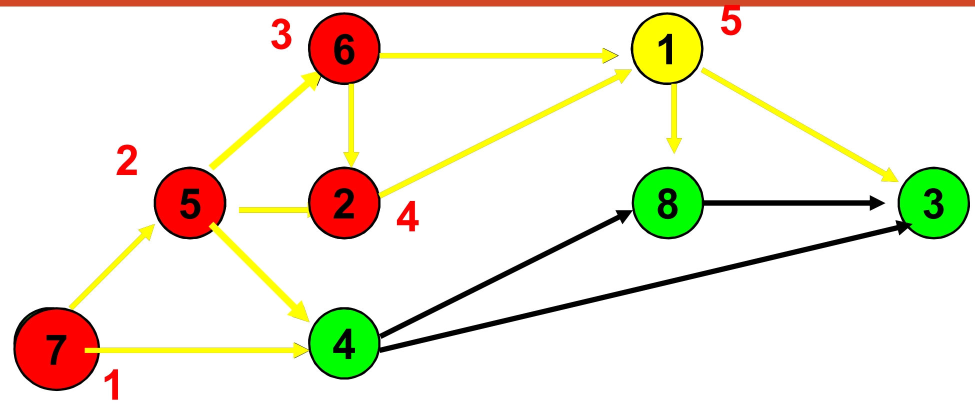

| 结点 | 1 | 2 | 3 | 4 | 8 |
|---|---|---|---|---|---|
| 入度 | 1 | 0 | 3 | 0 | 2 |

# 拓扑排序：过程



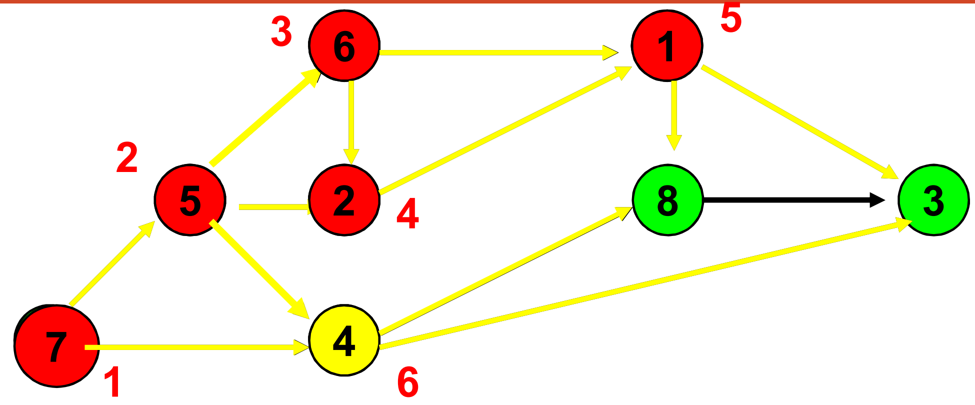

| 结点 | 1 | 3 | 4 | 8 |
|---|---|---|---|---|
| 入度 | 0 | 3 | 0 | 2 |

# 拓扑排序：过程



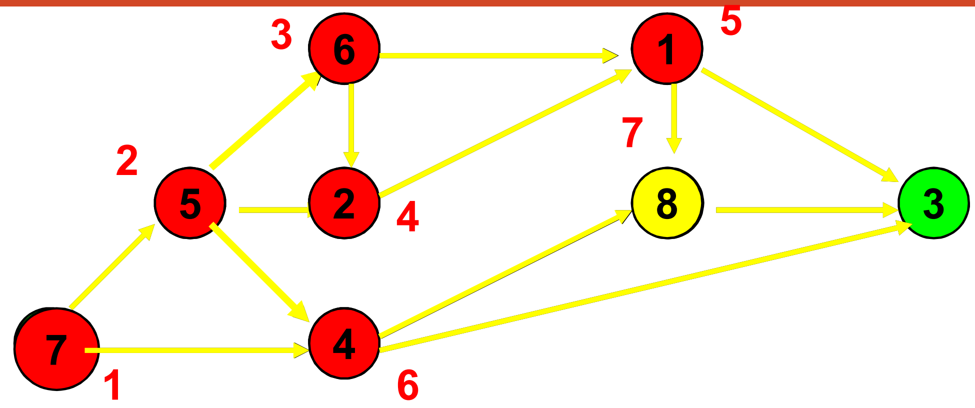

| | | | | |
|---|---|---|---|---|
| 结点 | 3 | 4 | | 8 |
| 入度 | 2 | 0 | | 1 |

# 拓扑排序：过程



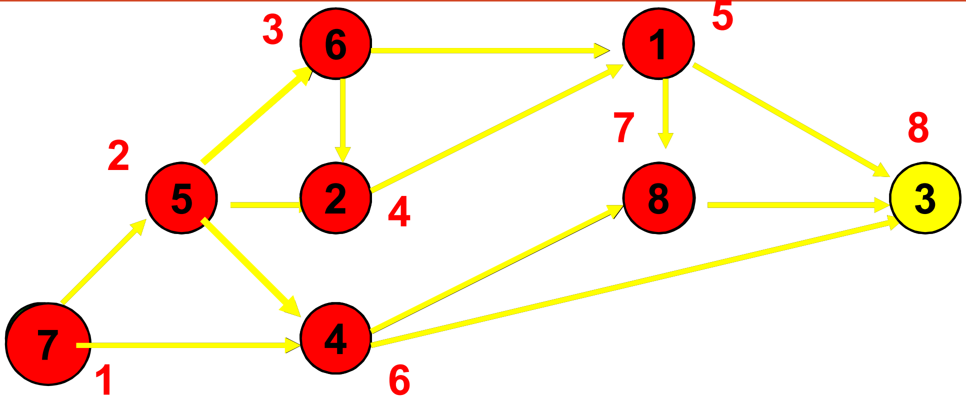

| | | |
|---|---|---|
| 结点 | 3 | 8 |
| 入度 | 1 | 0 |

# 拓扑排序：过程




| | |
|---|---|
| 结点 | 3 |
| 入度 | 0 |

# 拓扑排序：过程




结点

入度

# 拓扑排序：应用

- 假设你有一些任务，以及这些任务之间的依赖关系
- 每个任务有一个完成时间Ti
- 假设可以无限并行，最少要多少时间才能完成？

# 拓扑排序：作业

- Leetcode 207. Course Schedule

# 最短路：定义

- 假设E集合（边集）是有权重的
- 具象
  - V集合代表城市
  - E集合代表城市间高速路，权重为高速路长度
- 两点间存在若干条通路
- 长度最短的通路 → 最短路

# 最短路：单源最短路

- 给定起点s，求到任意点的最短路（Dijkstra）
- 贪心：每次找最近的点
- 维护s到每个点的距离
- 局部最优等于全局最优
- 时间复杂度 $O(n^2)$
- 附加空间复杂度 O(n)

# 最短路：单源最短路

```
Q = {}
d[s] = 0, 其余值为正无穷大
while (|Q| < |V|)
        取出不在Q中的最小的d[i]
        for (i相邻的点j，j不属于Q)
                d[j] = min(d[j], d[i] + c[i][j])    //维护距离
        Q = Q + {i}
```

# 最短路：单源最短路作业

- 边权可以为负数么?
- 可以存在负权回路么?
- 实现一个Dijkstra
- 比较与Prim算法（最小生成树）差别
- 思考当$m << n^2$（稀疏图）时，如何优化
- 堆优化

# 最短路：单源次短路

- 给定起点s，求到任意点的次短路（距离大于最短路的最短的路）
- v的次短路：
    - 顶点u的最短路再加上u->v的边
    - 顶点u的次短路再加上u->v的边
- 在原来的代码上加入次短路即可

# 最短路：任意两点最短路

- 求到任两点间的最短路（Floyed）
- 类似动态规划：每次加入一个点
- 维护任意两点间的距离
- 时间复杂度 $O(n^3)$
- 附加空间复杂度 $O(n^2)$

# 最短路：任意两点最短路

```
for i := 1 to n do
    for j := 1 to n do
        read(f[i, j]);
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            f[i, j]= min(f[i, j], f[i, k] + f[k, j]);
```

# 最短路：思考

- 边权可以为负数么?
- 可以存在负权回路么?
- N次dijkstra = Floyed ?

# 最小生成树

- 无向图
- 树 -> 无环（圈）
- 破圈法，避圈法
- Prime
  - 时间复杂度O(n^2)

# SPFA（Bellman-Ford）

```
Q = { s }
While not empty(Q)
        u = dequeue(Q)
        for v ∈ adj[u]
                if d[u] + g[u, v] < d[v]
                        d[v] = d[u] + g[u, v]
                        if not v in Q
                                enqueue(v);
```