图搜索初探

Outline

- 何谓图?
  - 图是描述世间万物关系的一种方式
  - 节点+边
- 隐式图
  - 状态（结点）不确定（明显）
  - 关系（边）不确定（明显）
  - 如何确定状态和关系（重点）

Outline

- 图搜索
  - 深度优先遍历（DFS）
  - 广度优先遍历（BFS）
- 隐式图搜索
  - N皇后问题、骑士游历问题、八数码

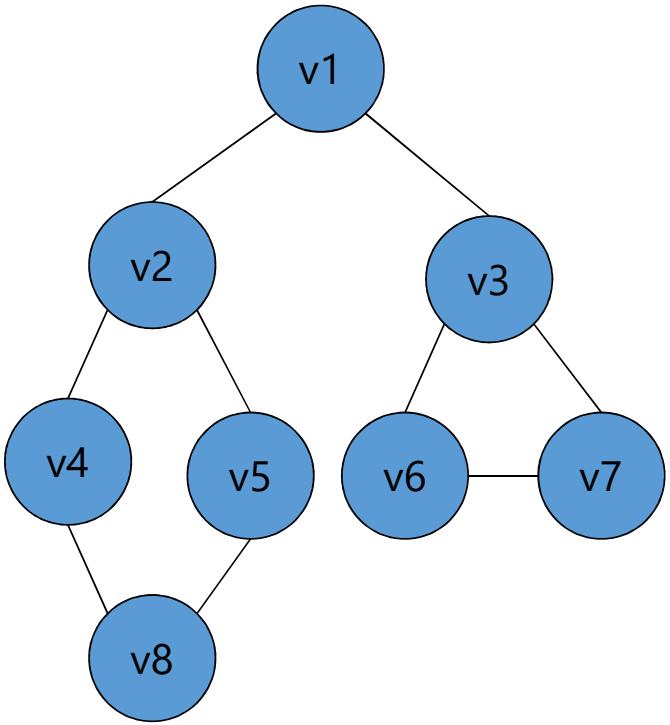
遍历：定义

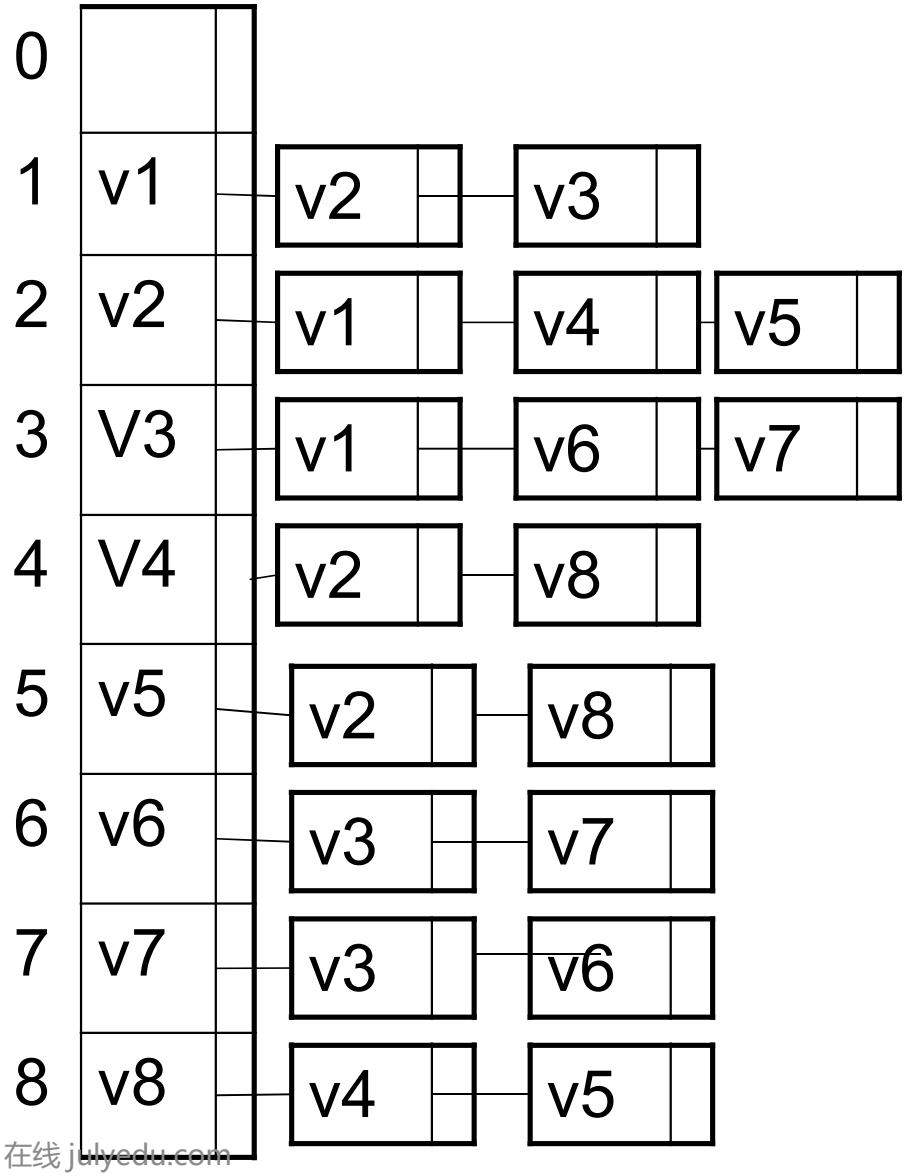
- 按某种顺序访问“图”中所有的节点
- 顺序
  - 深度优先（优先往深处走）
  - 广度优先（优先走最近的）
- 时间复杂度 O(n + m)
- 空间复杂度?

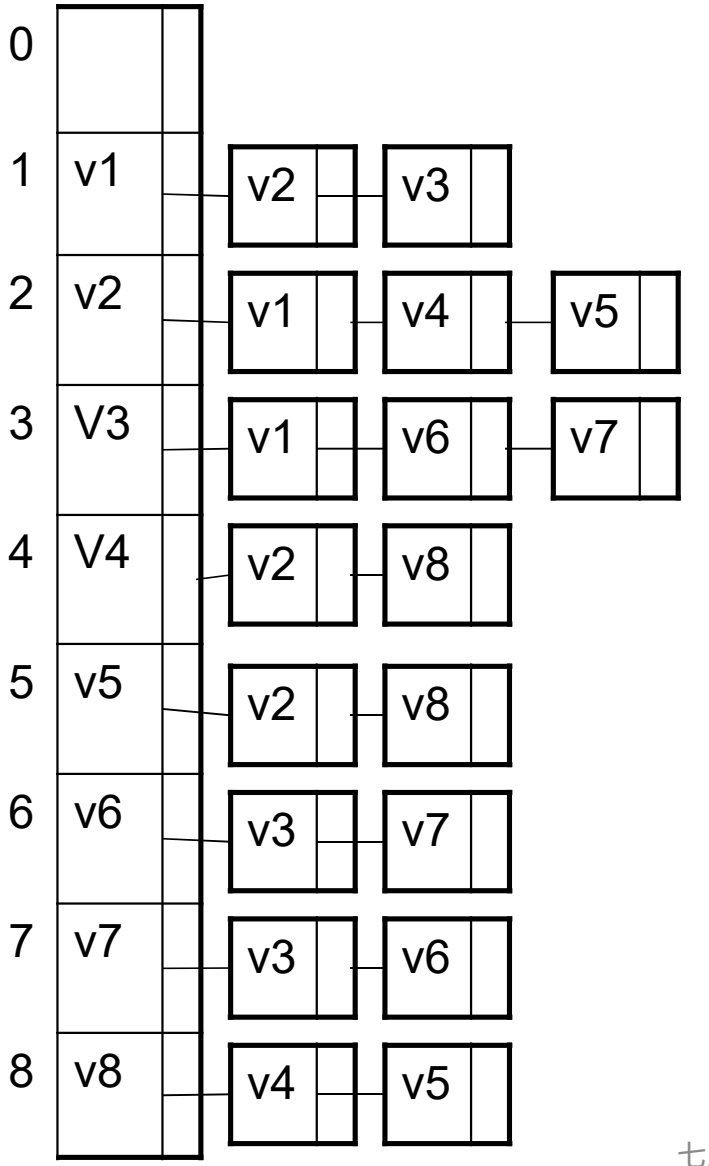
遍历

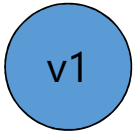
- 给出图G，要求求从入口v1访问到每一个点
- 两种遍历方式的数据结构
  - 栈（递归，深度优先）
  - 队列（广度优先）
- 广度优先找出的路径，经过节点数最少

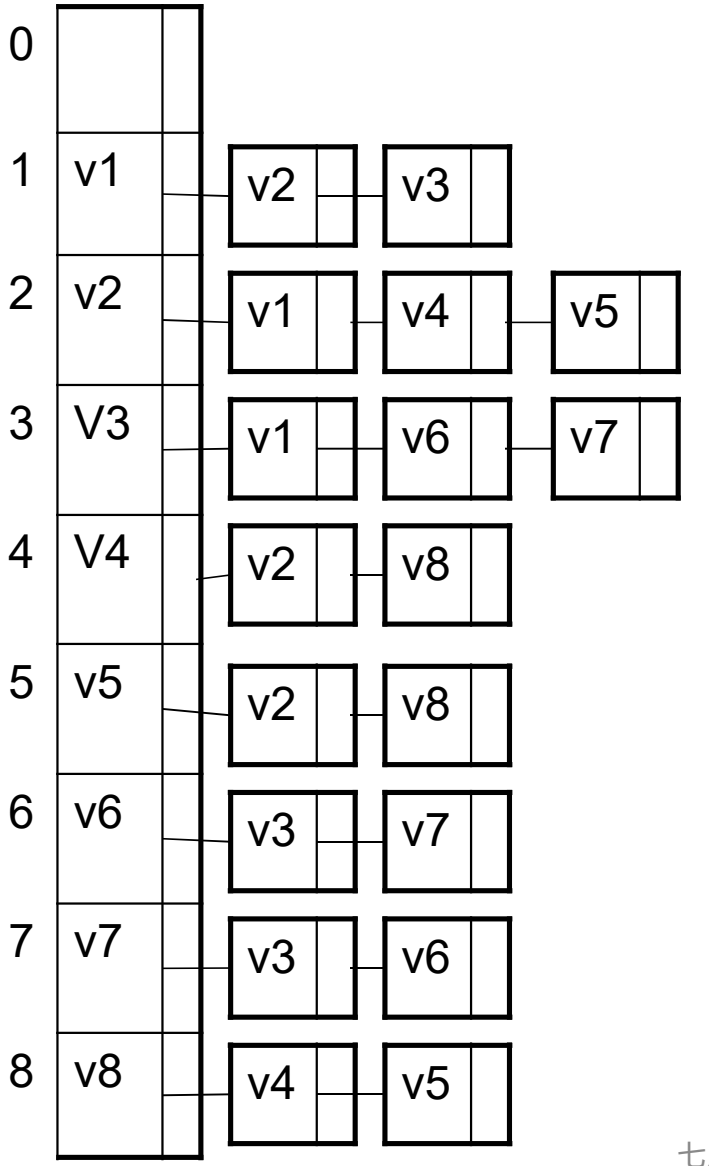
深度优先

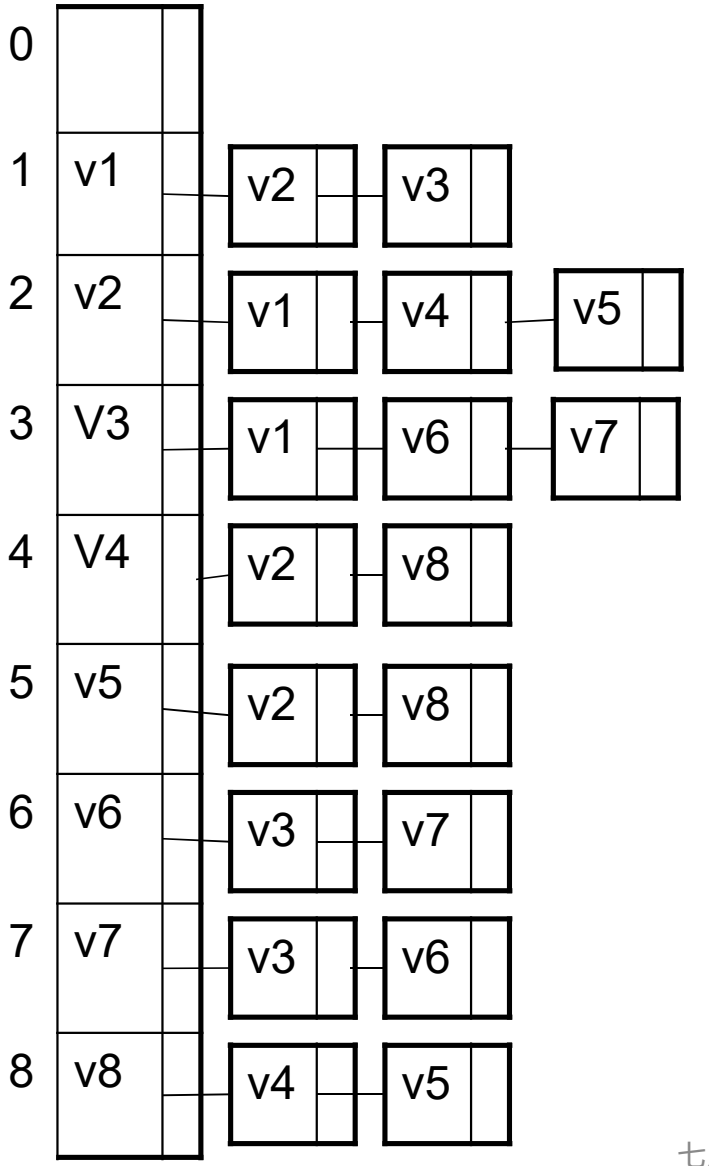


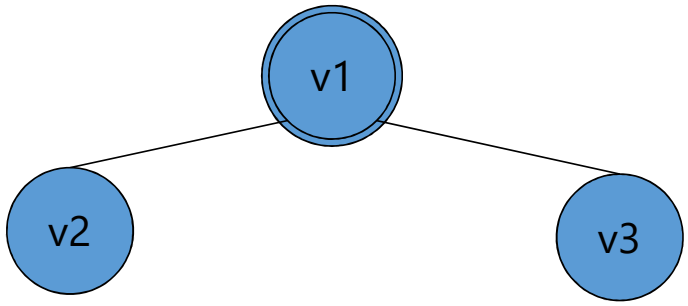


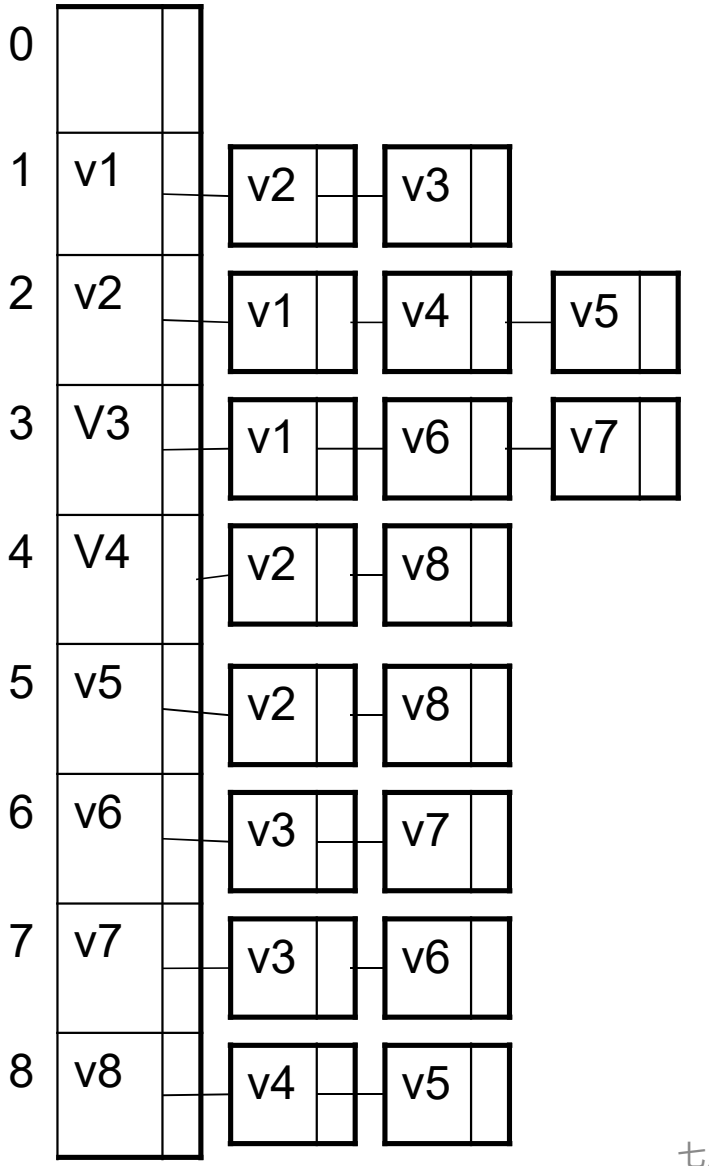


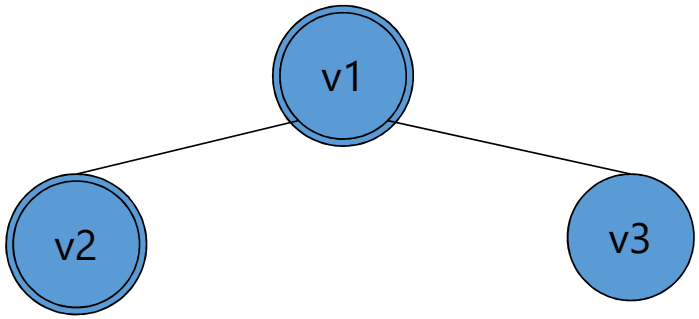


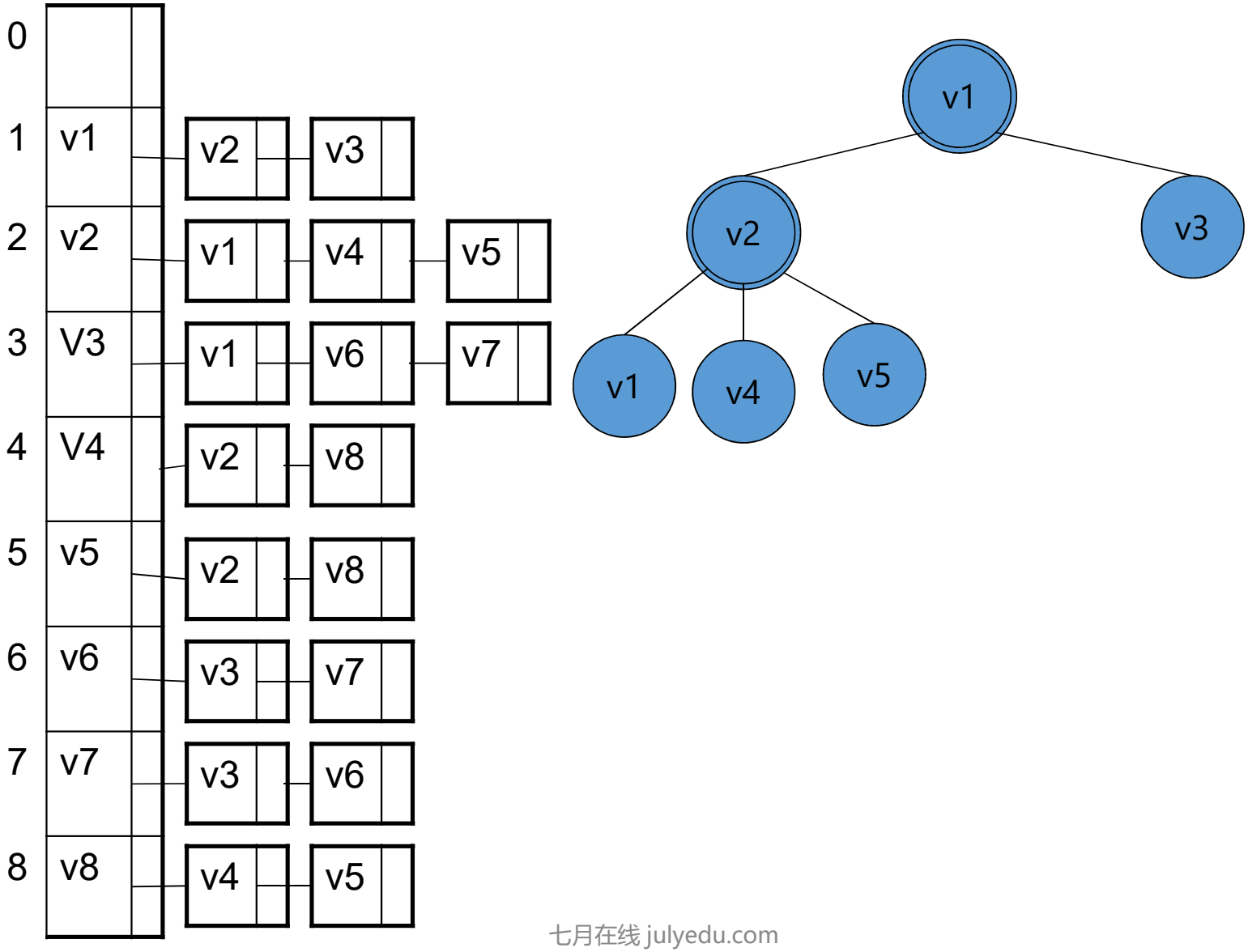


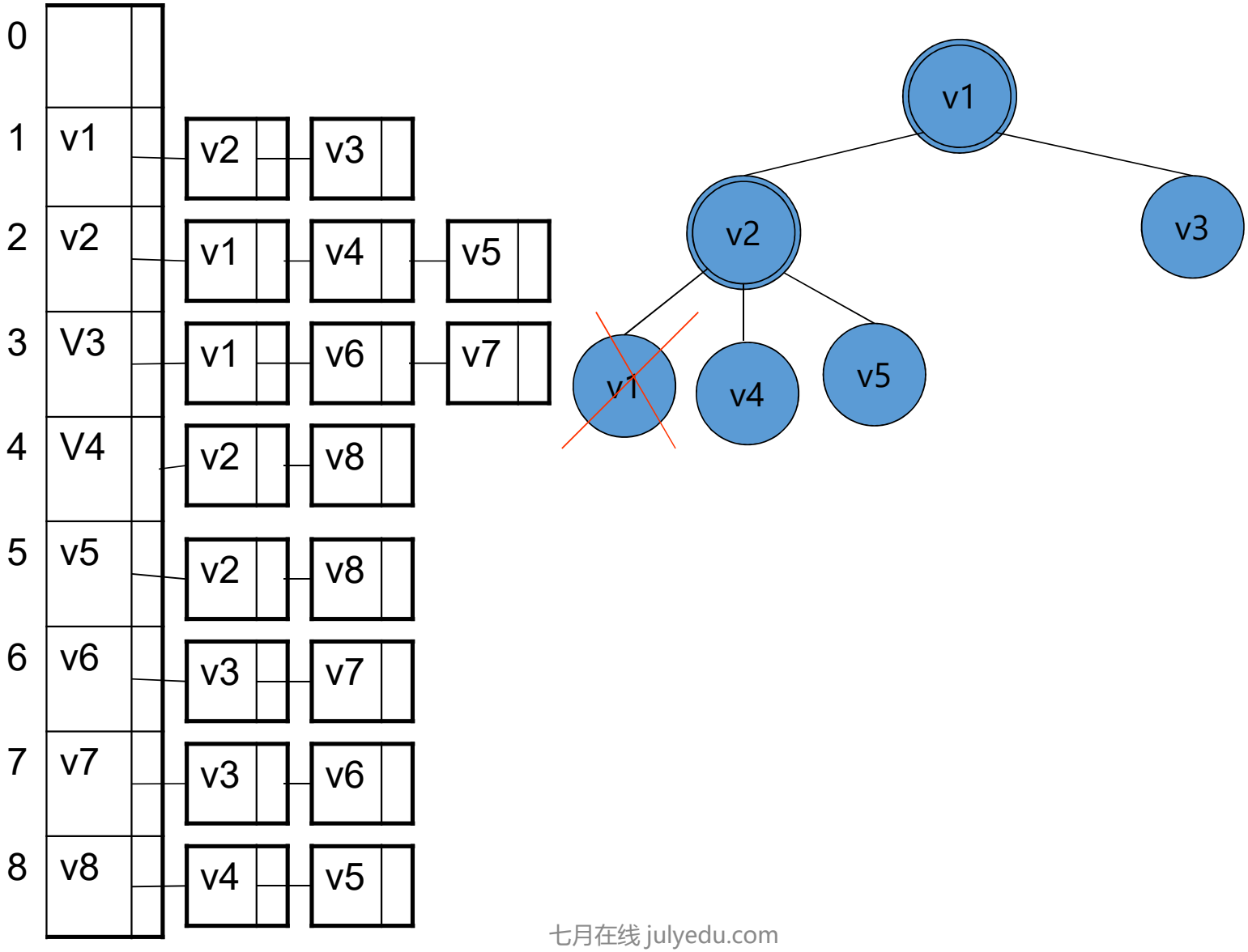


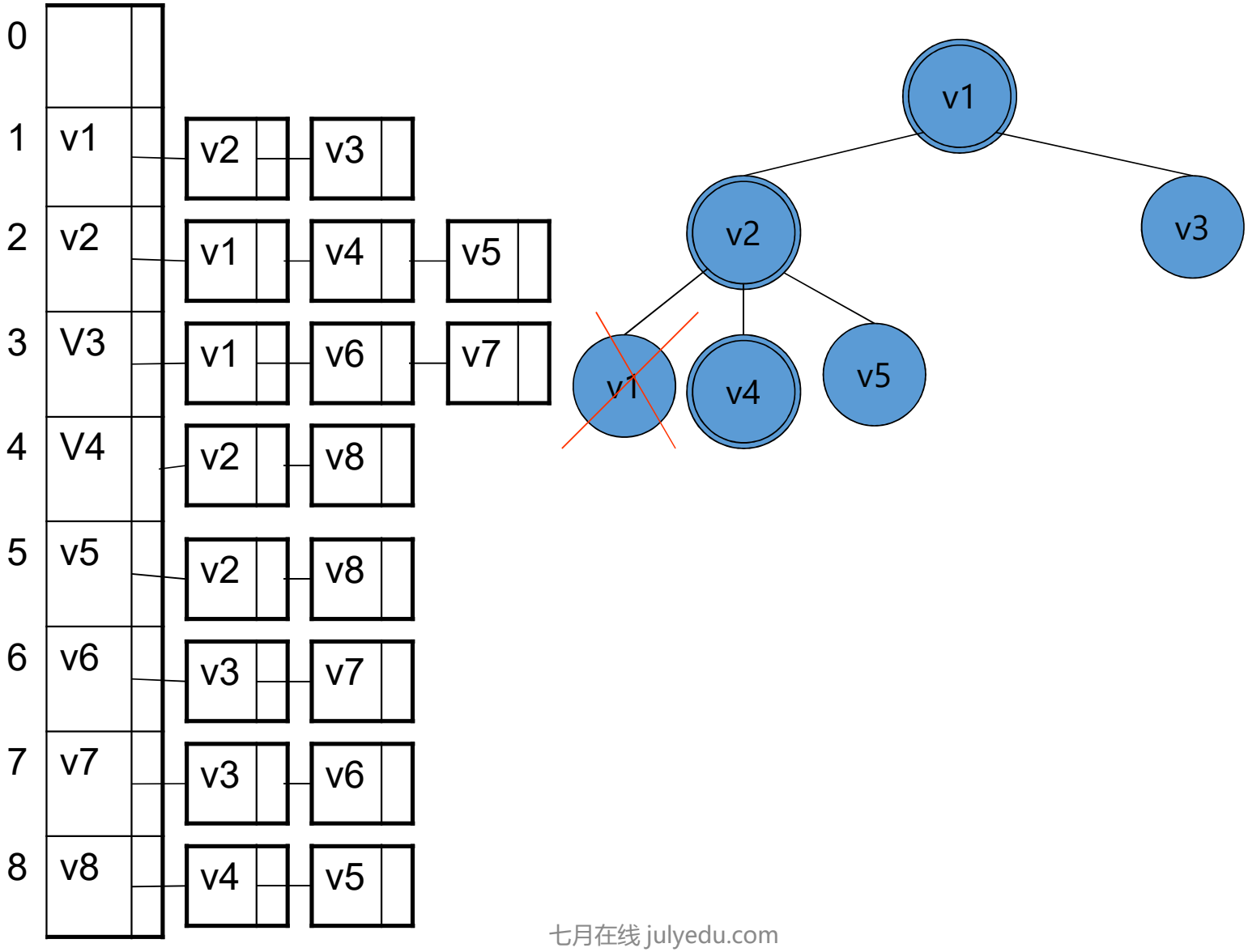


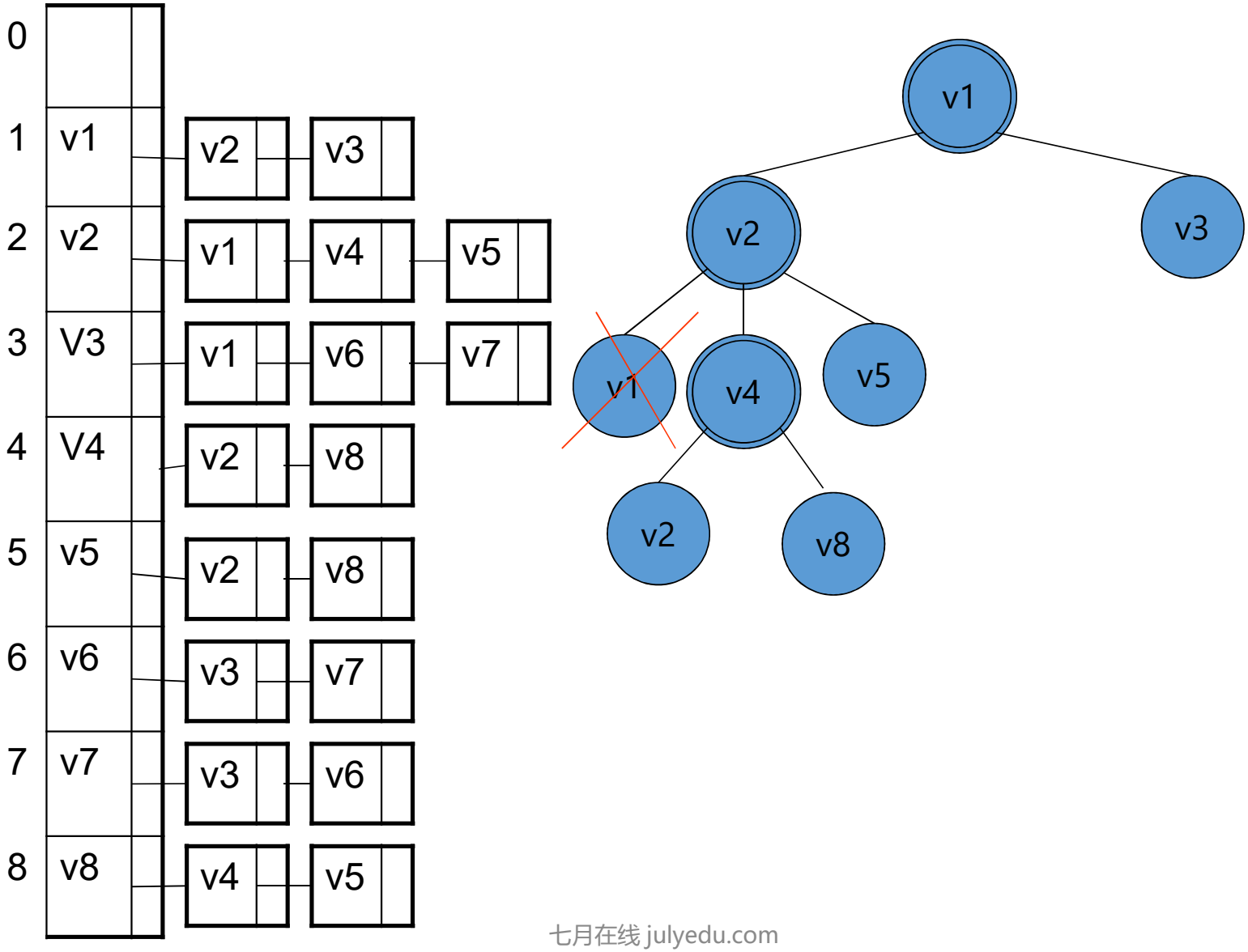


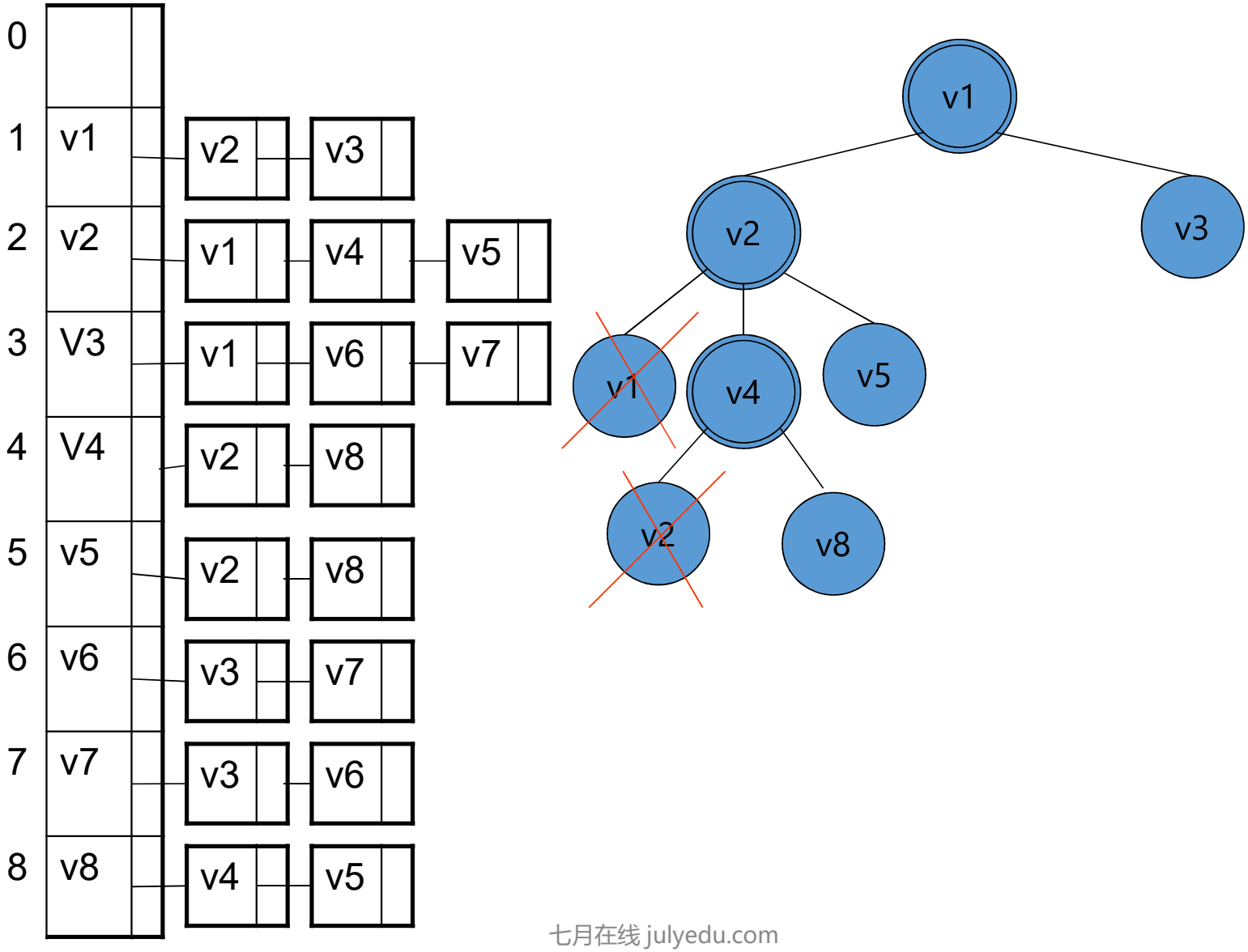


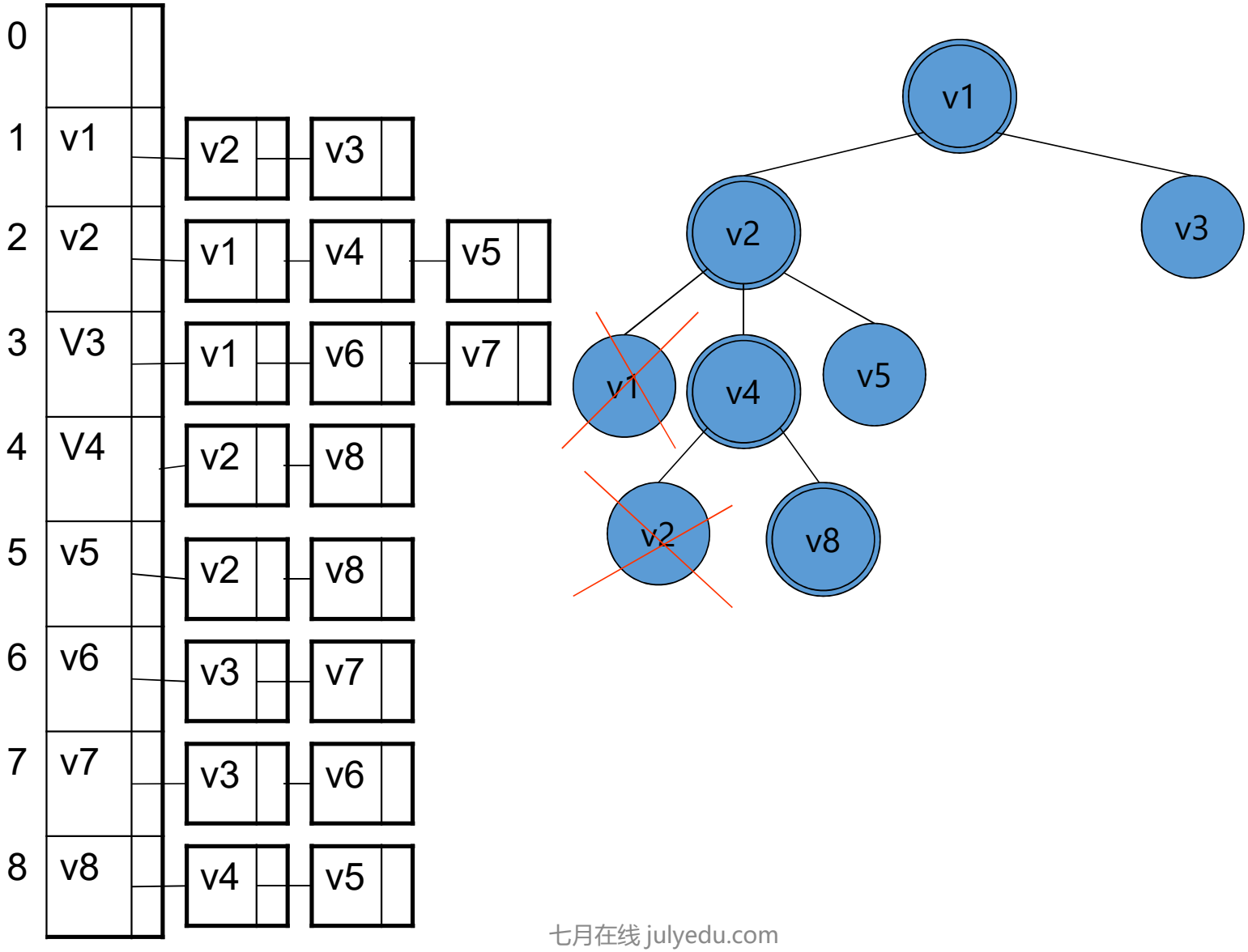


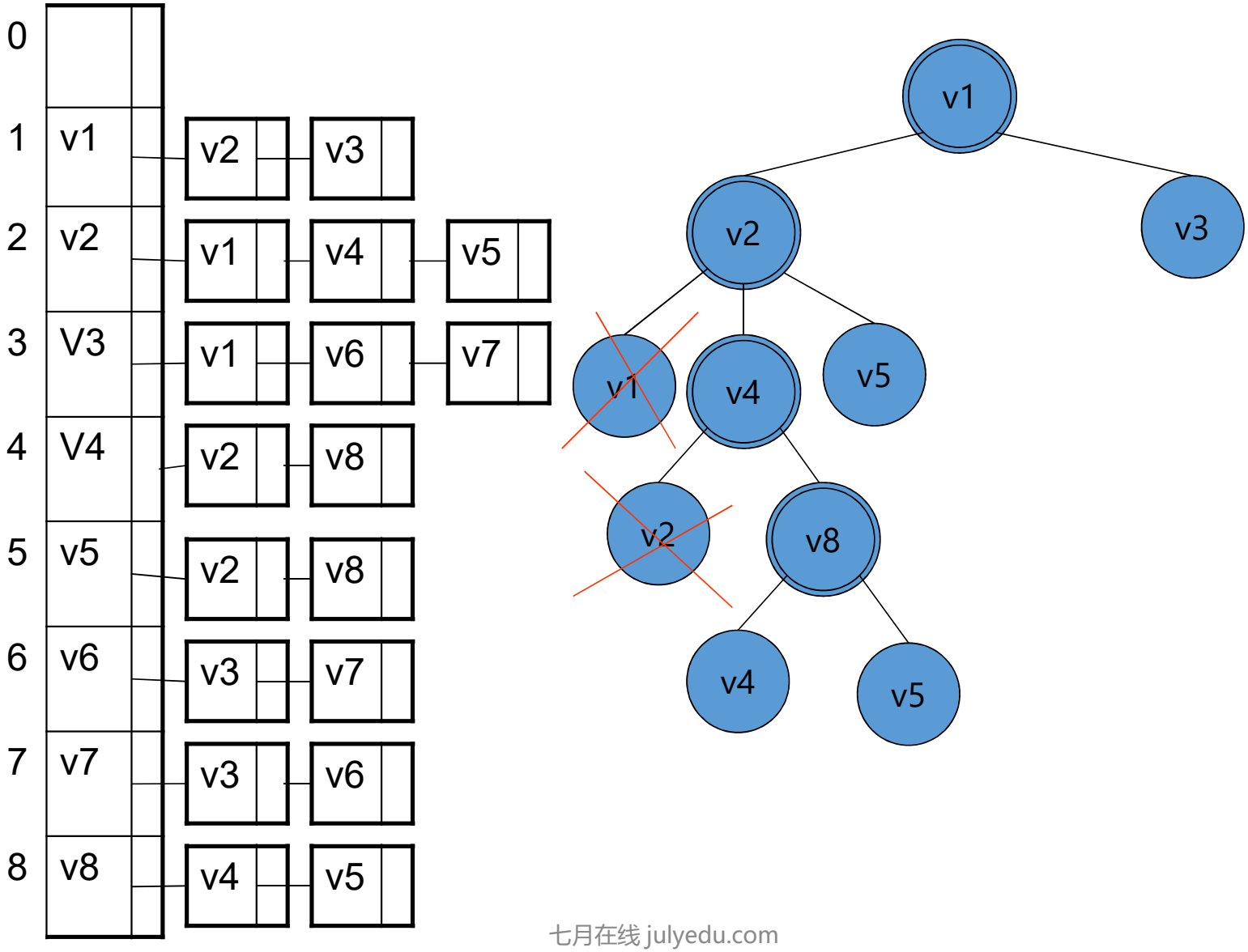


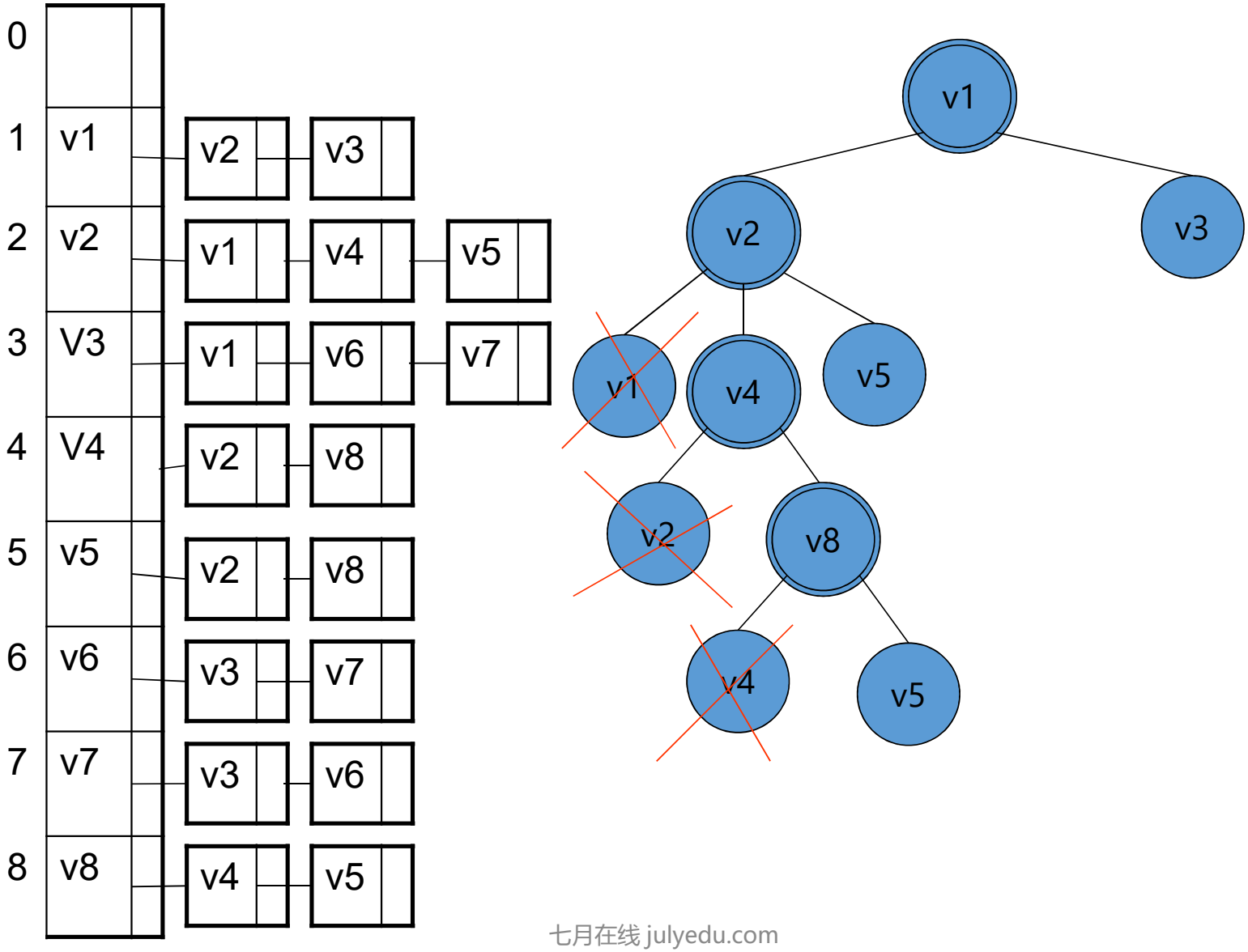


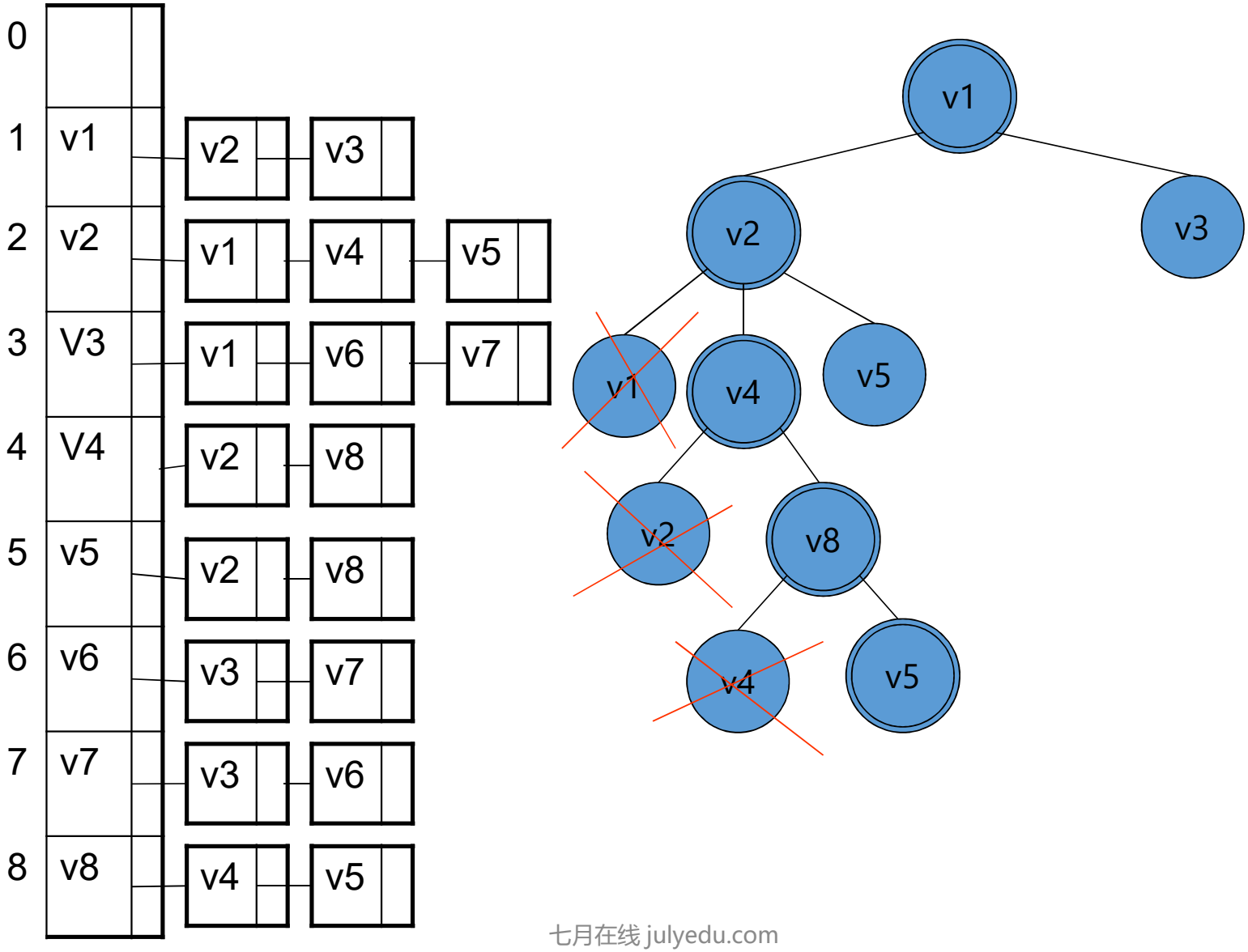


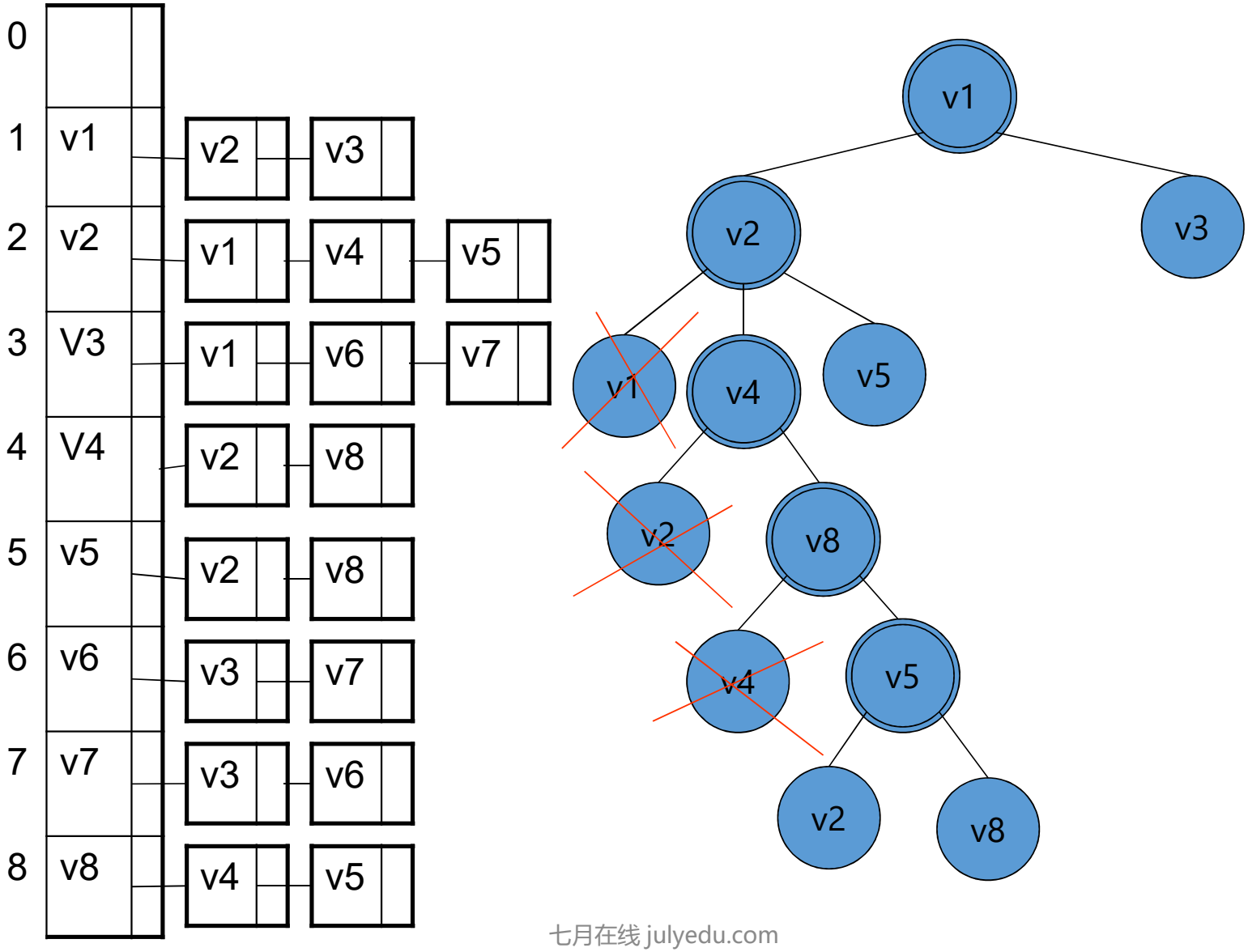


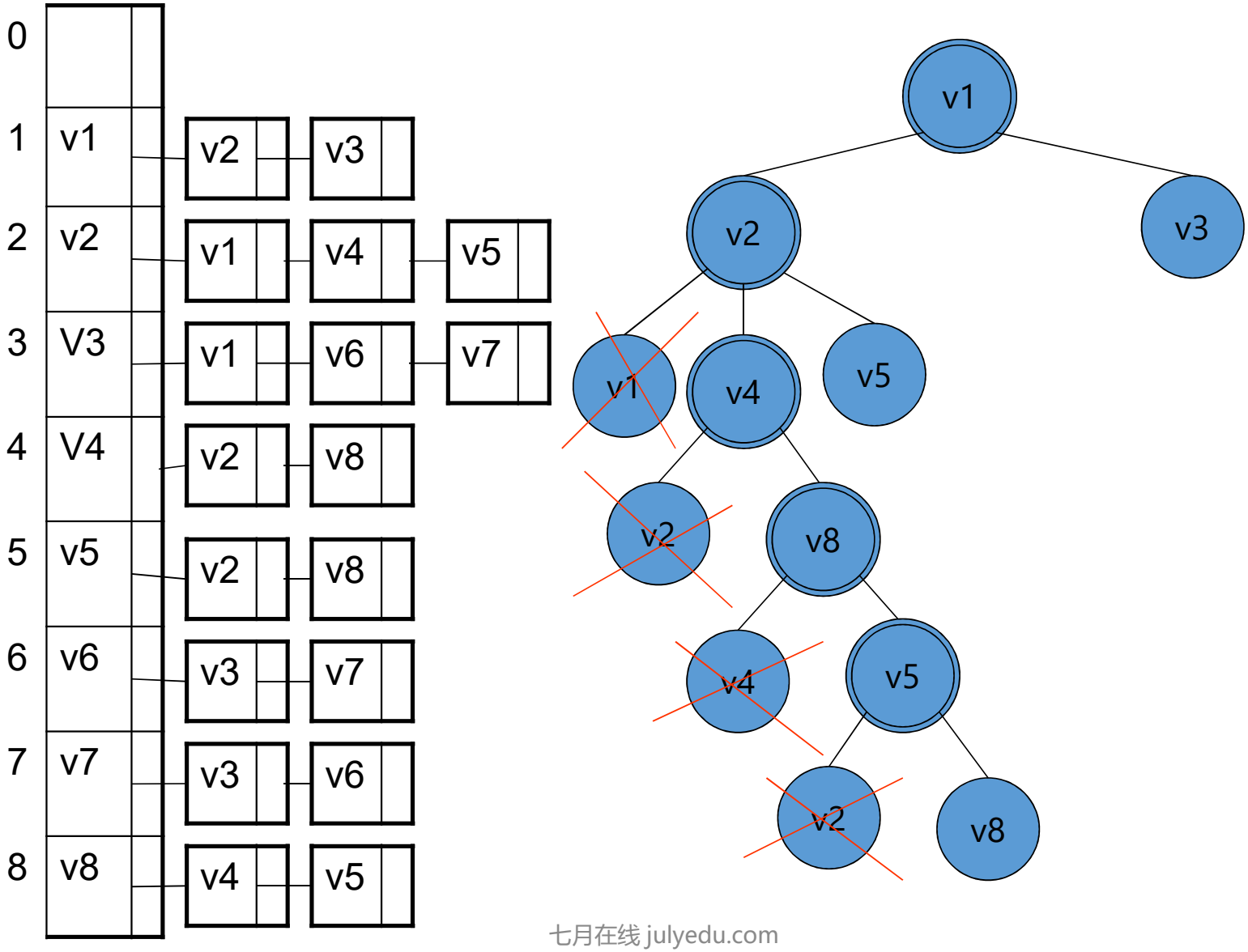


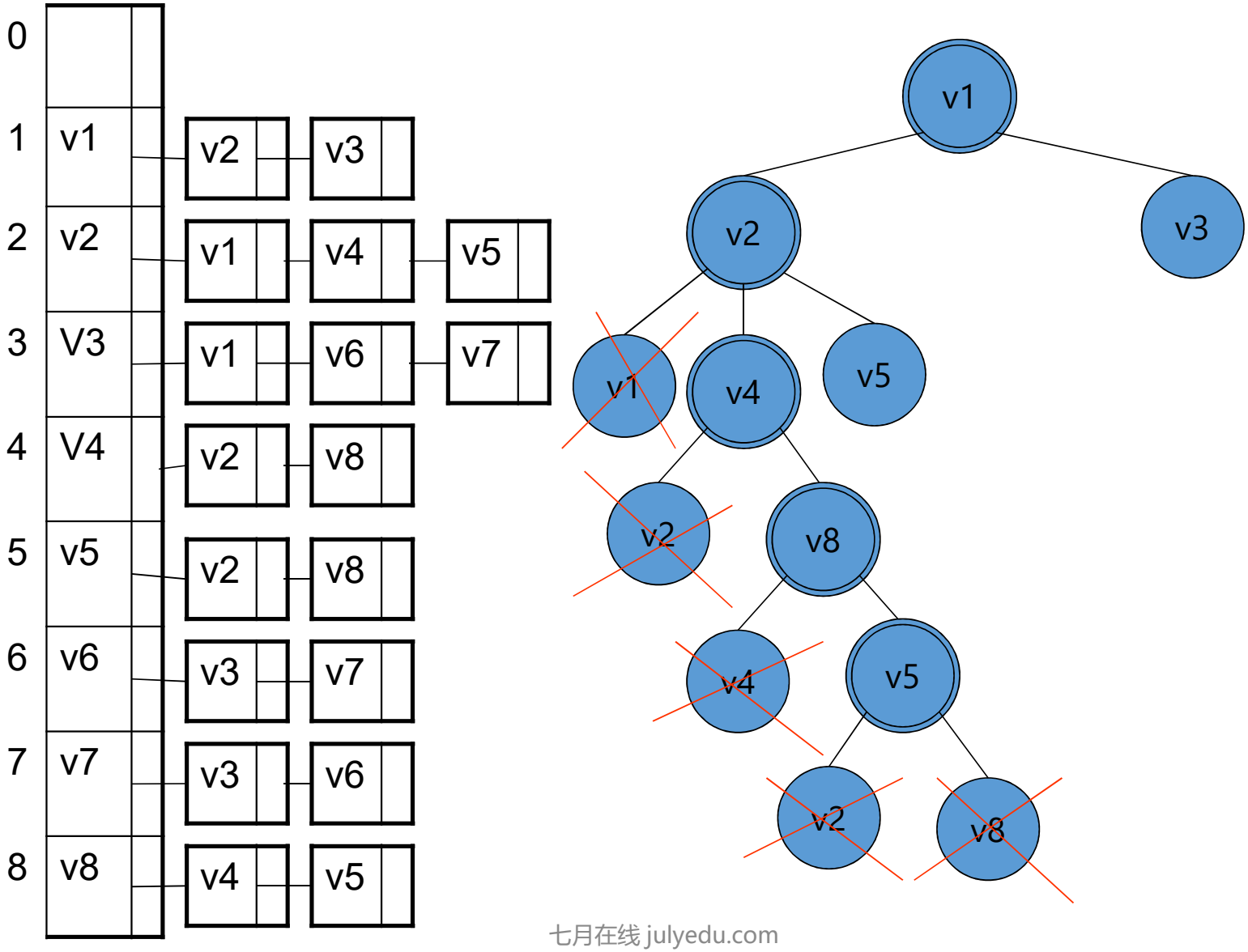


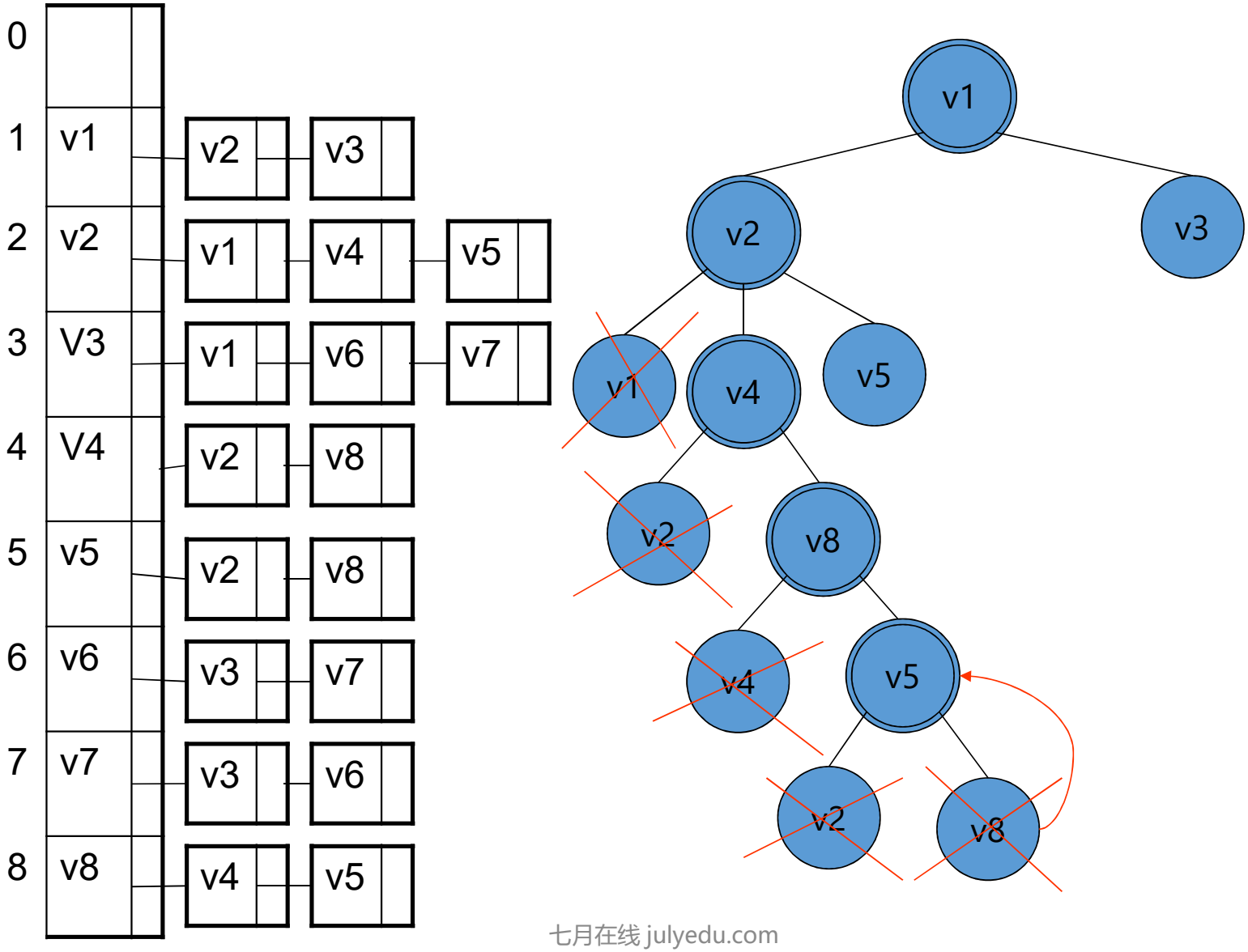


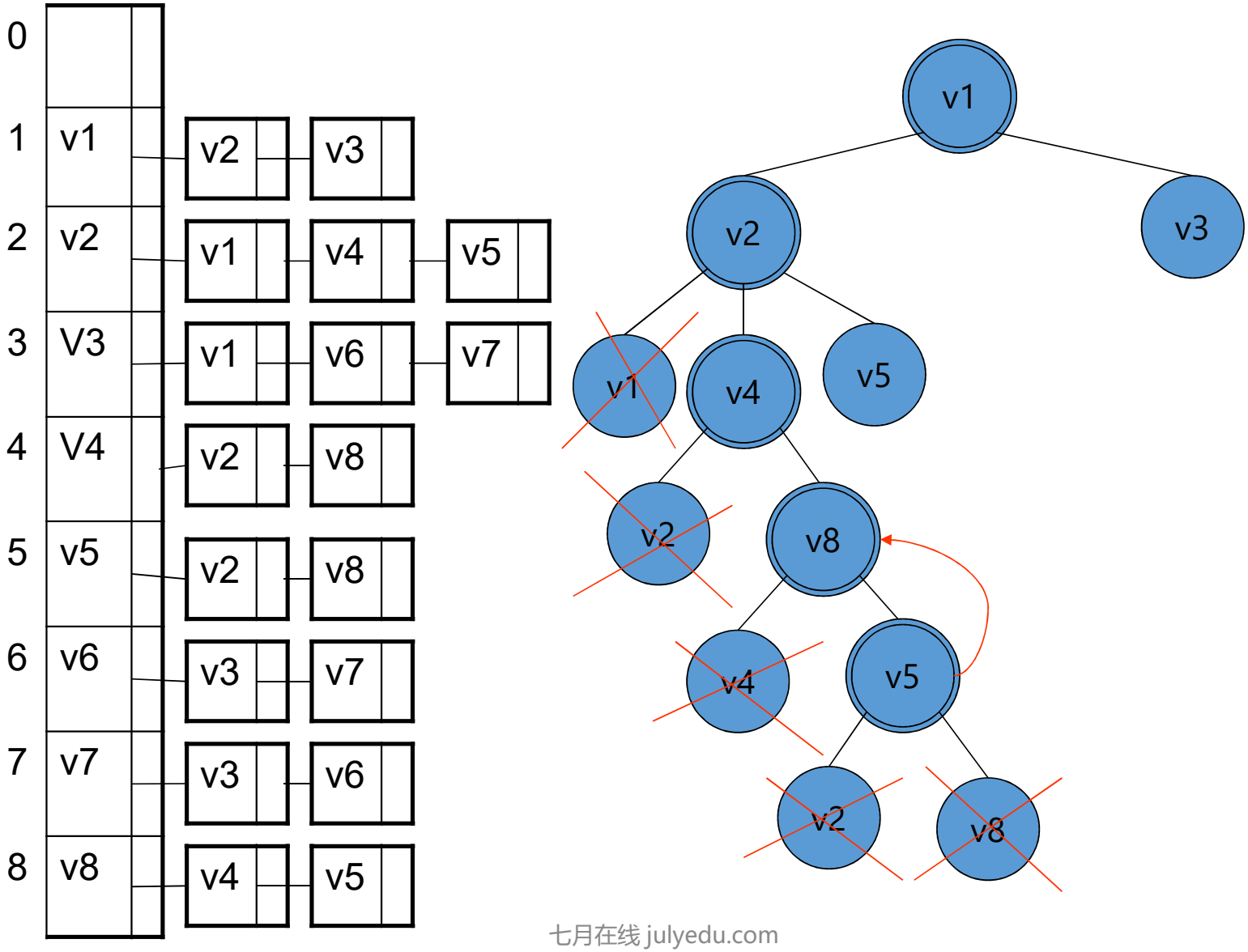


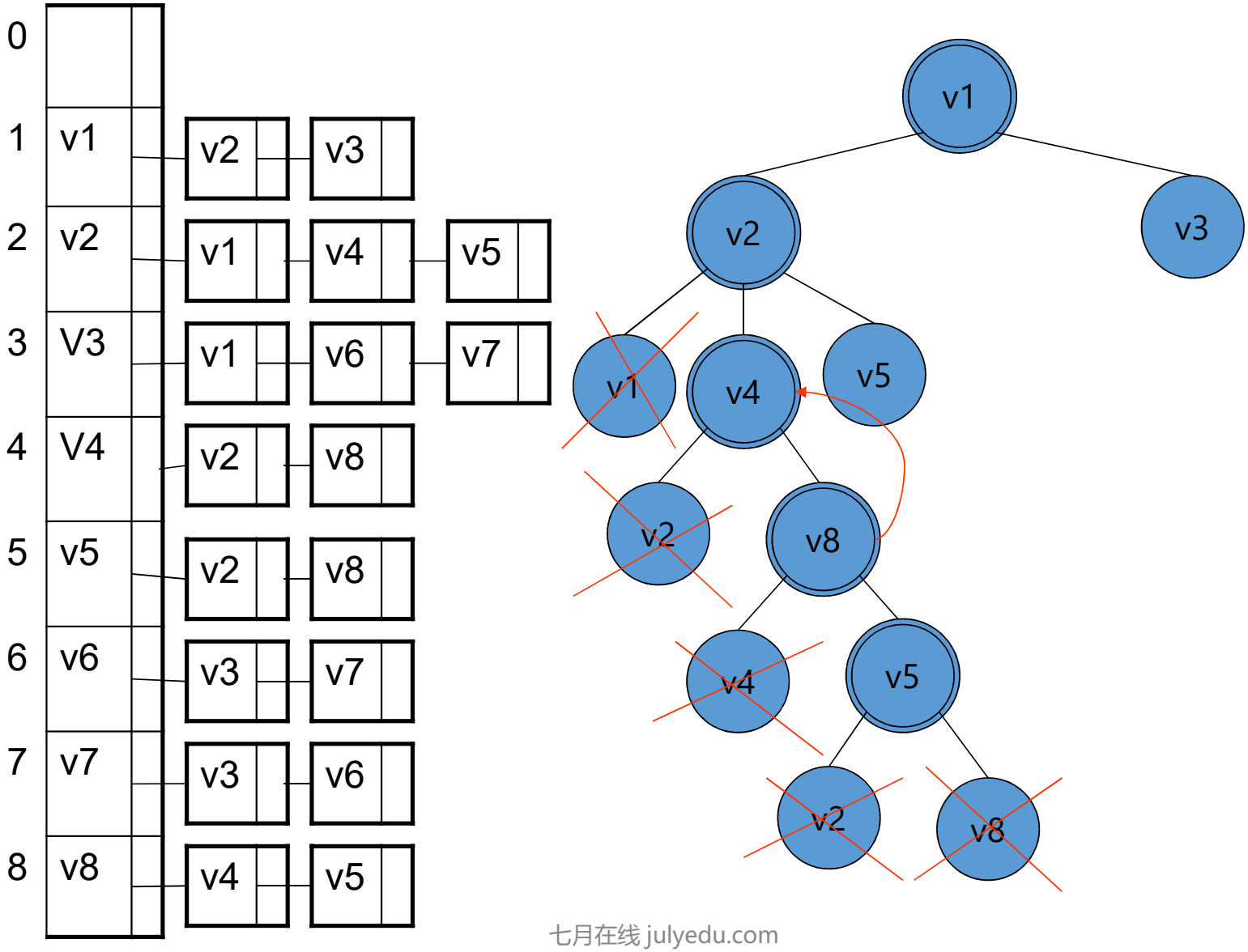


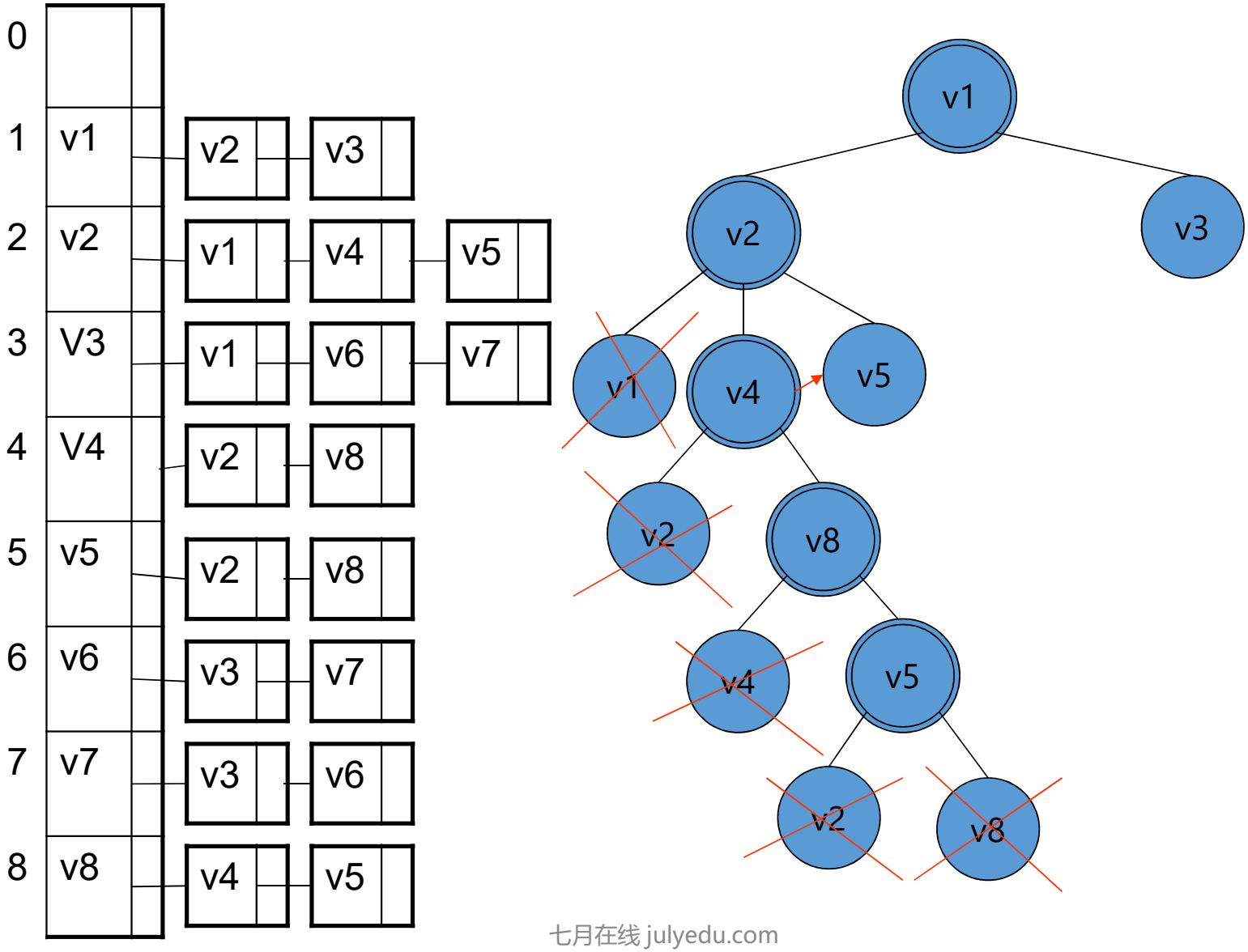


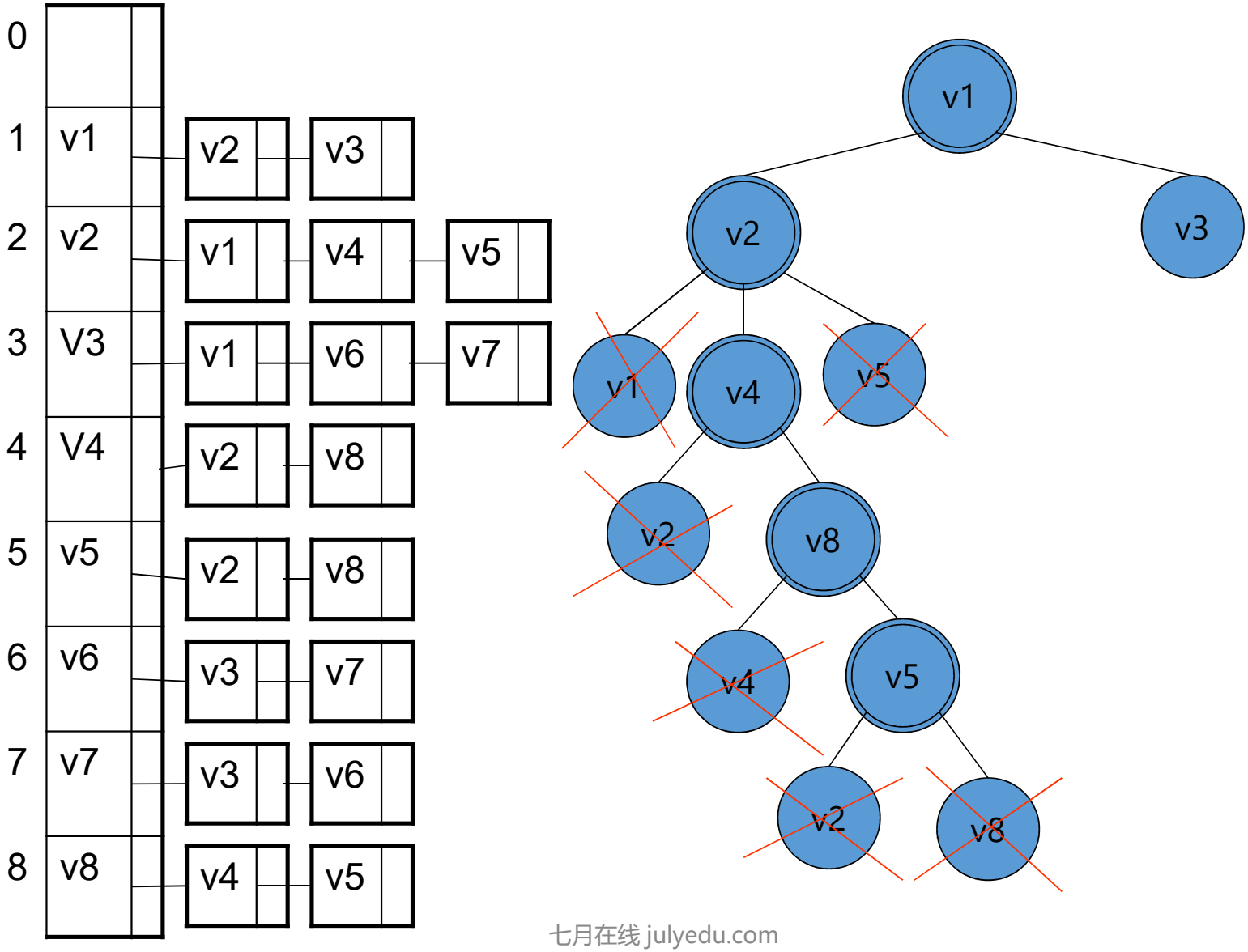


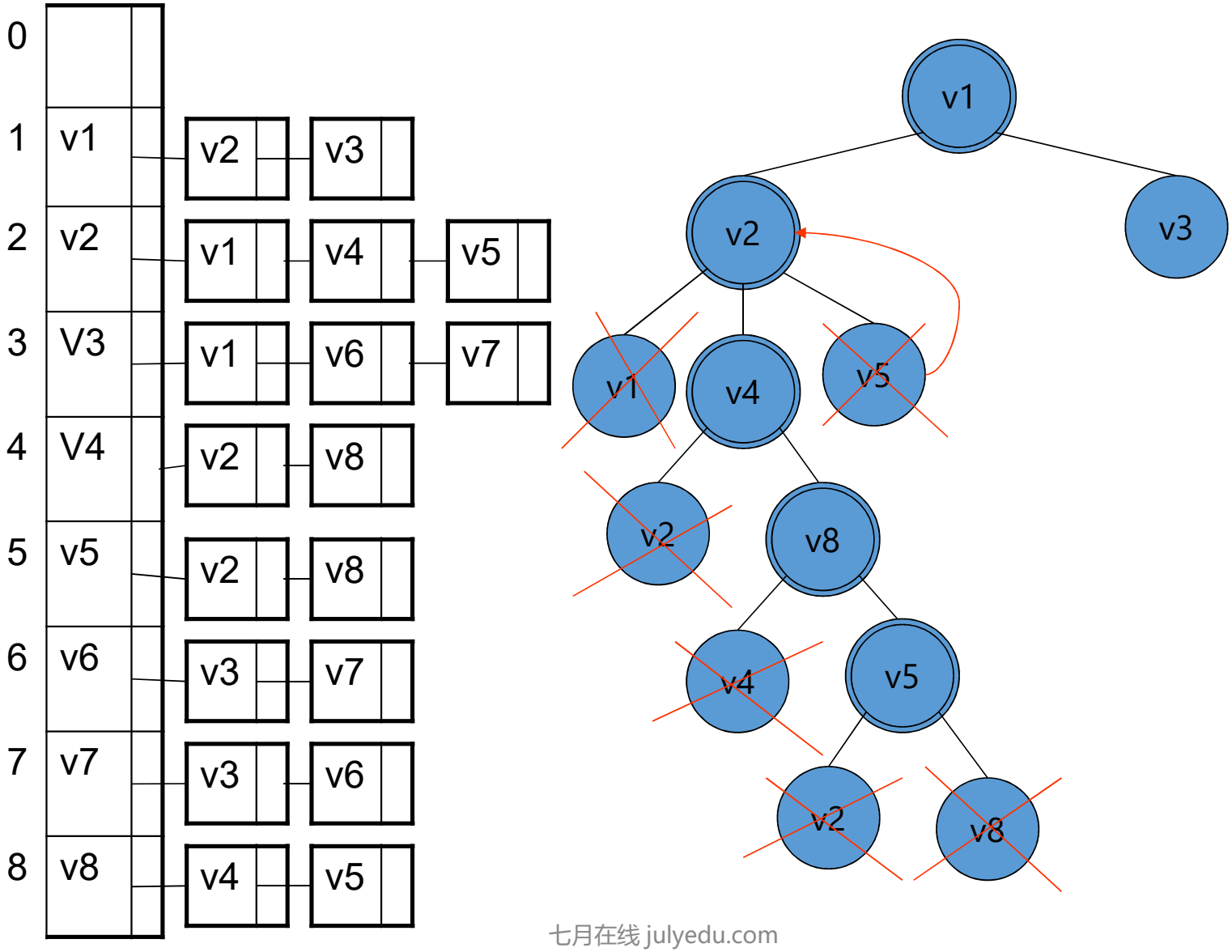


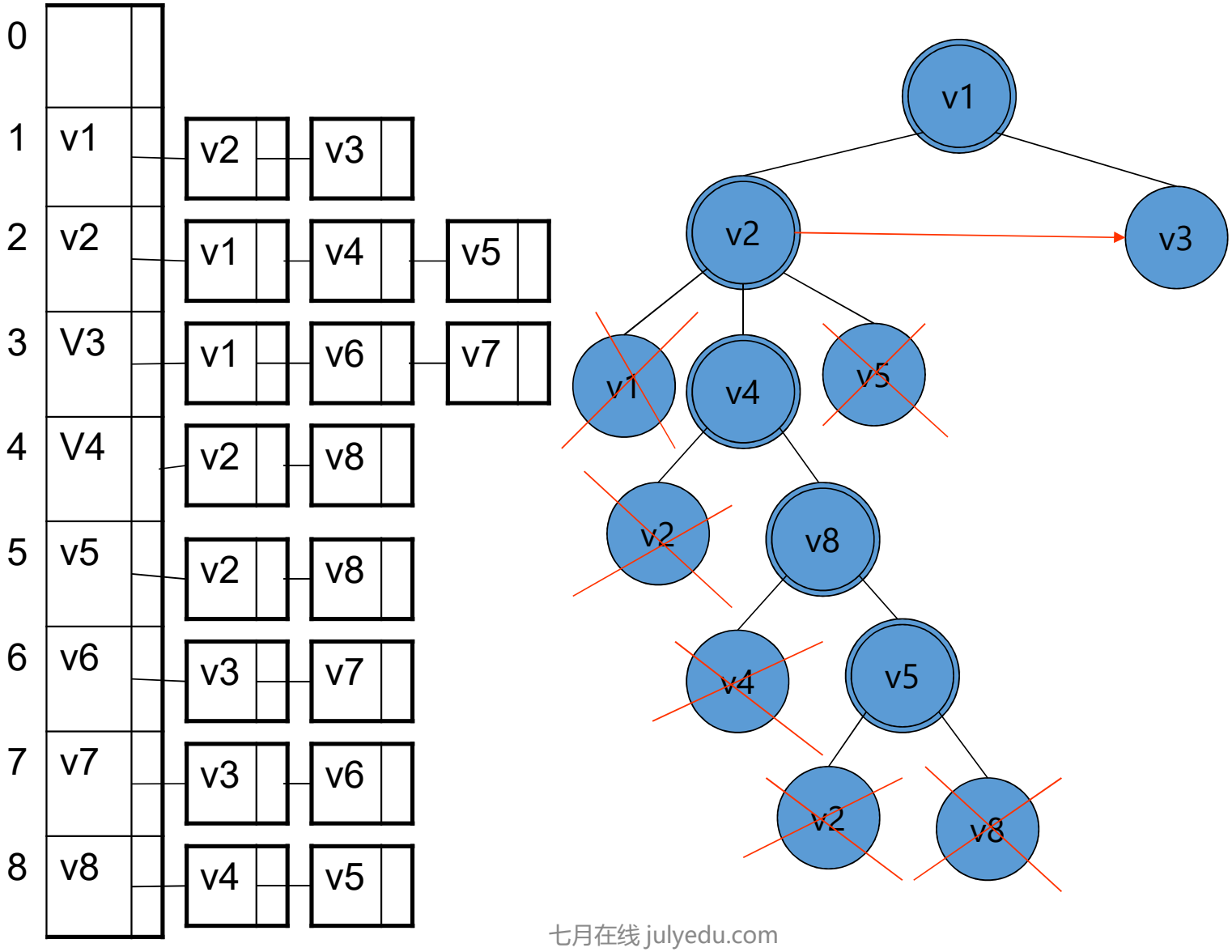


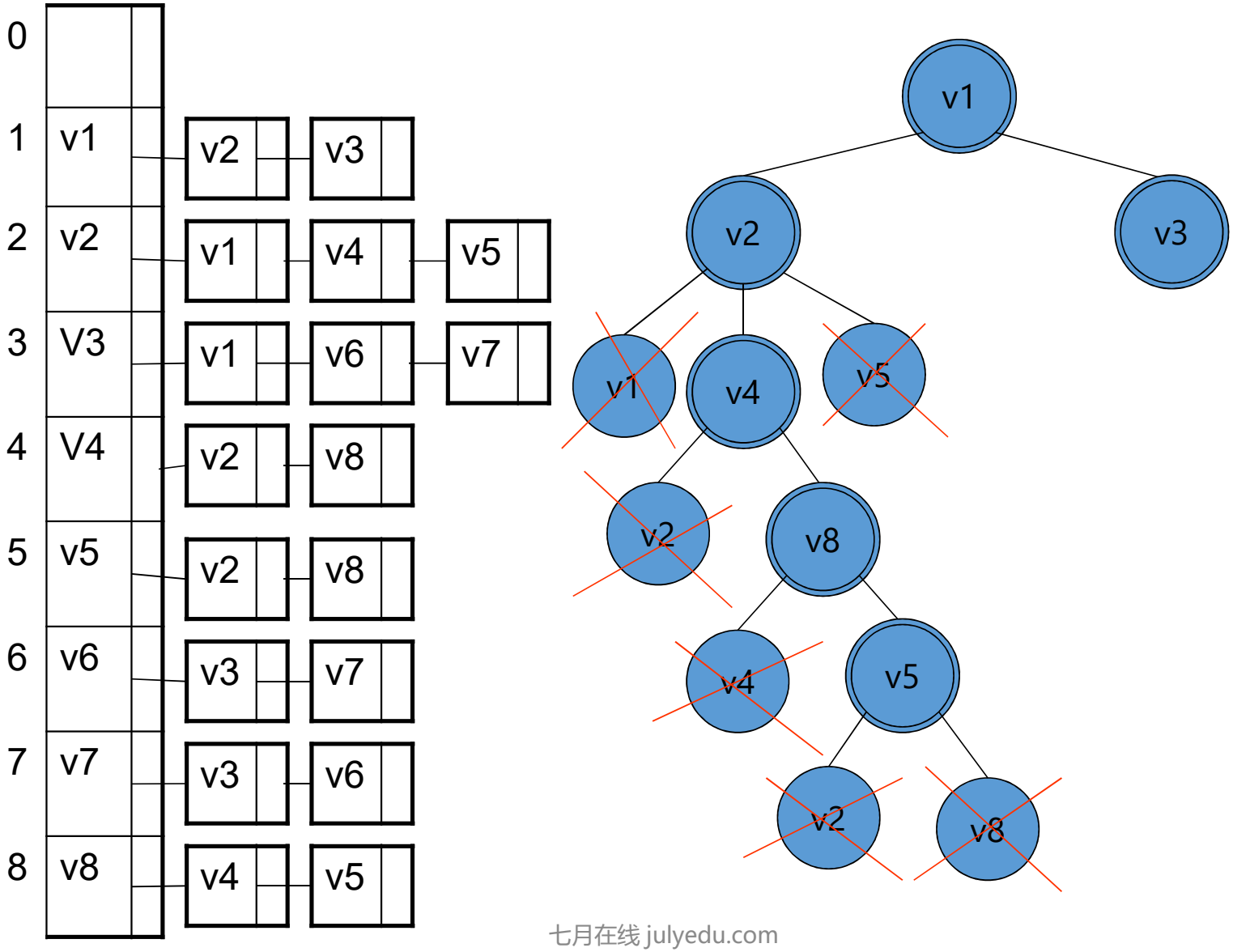


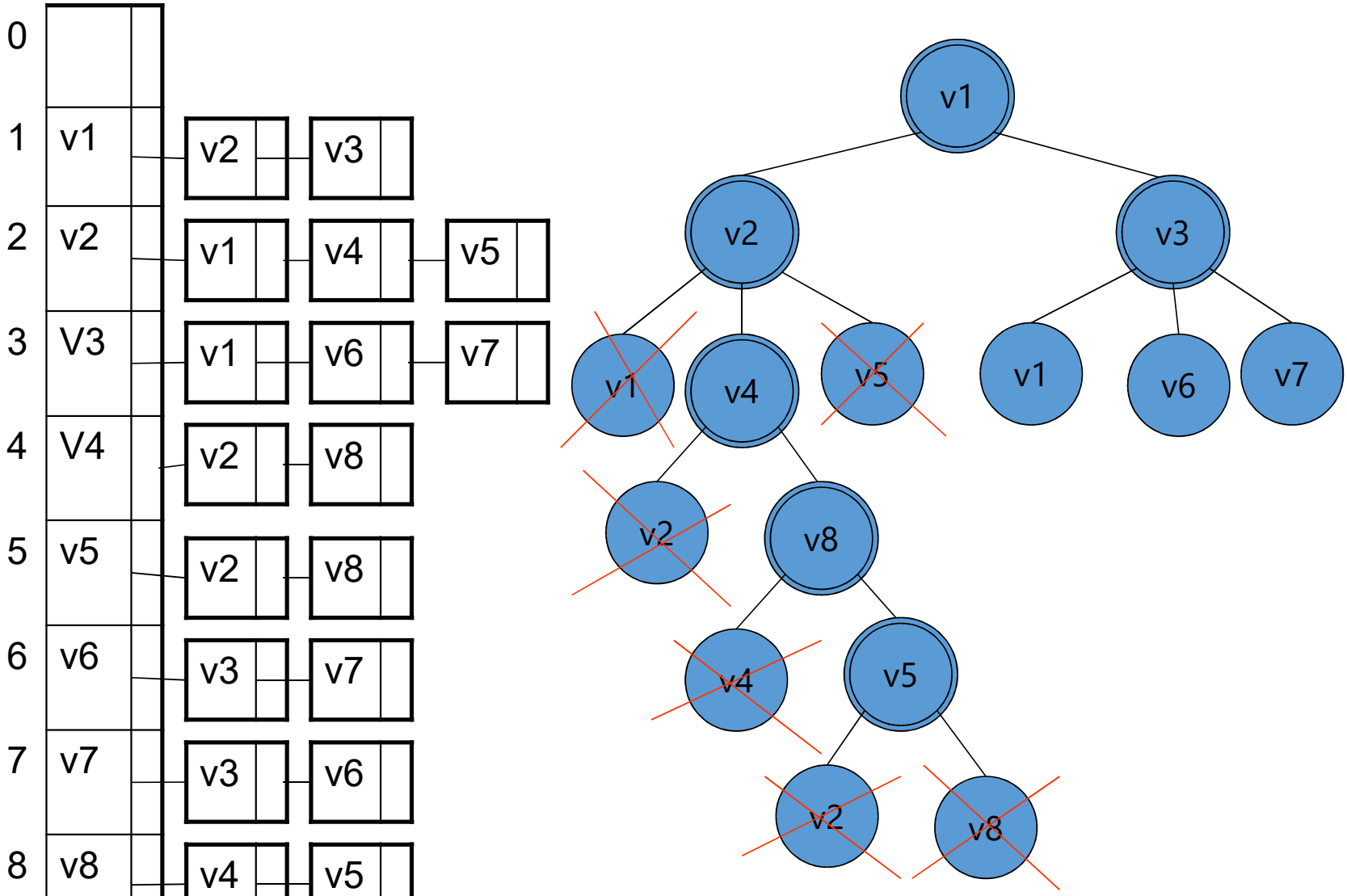


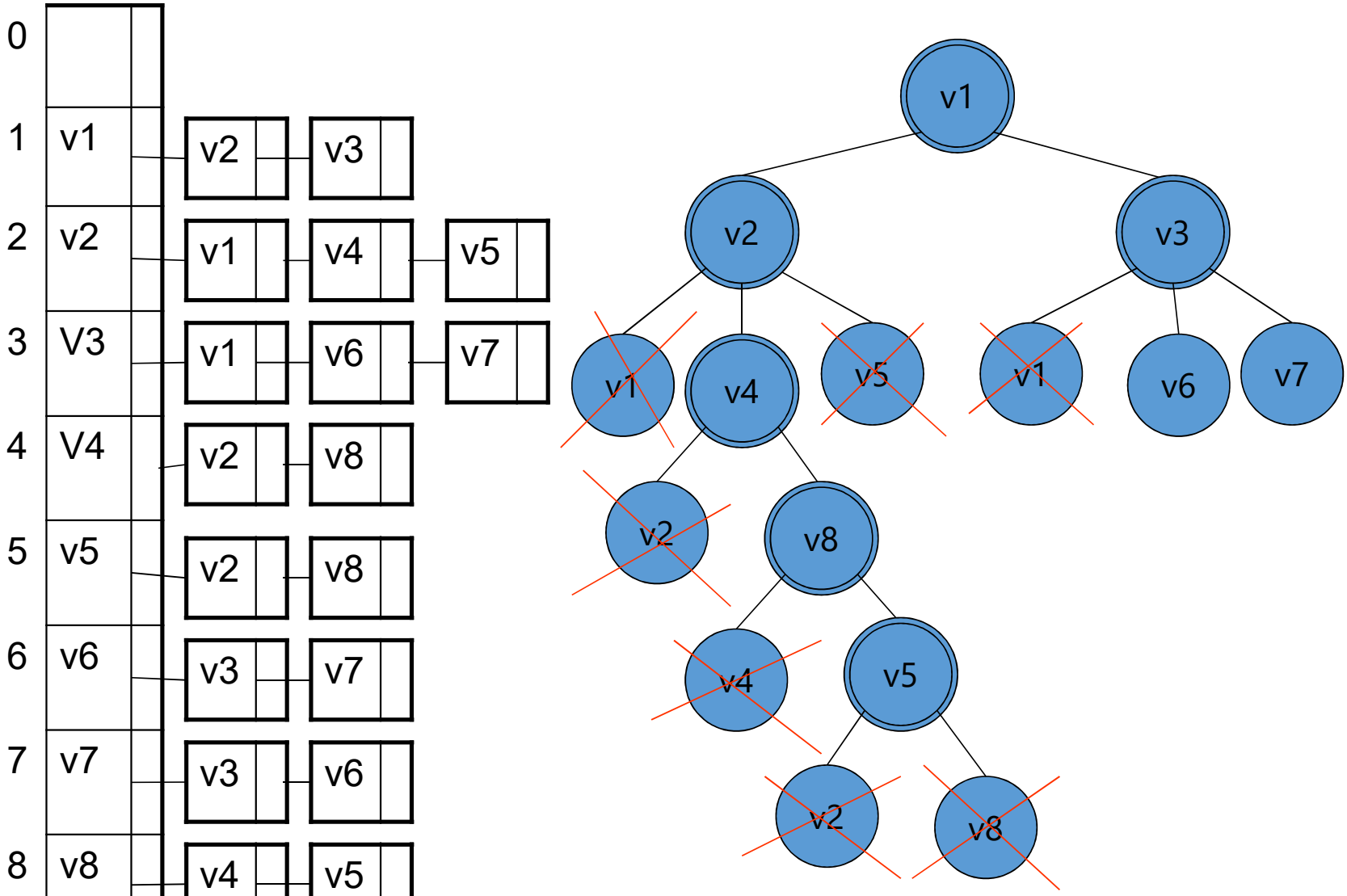


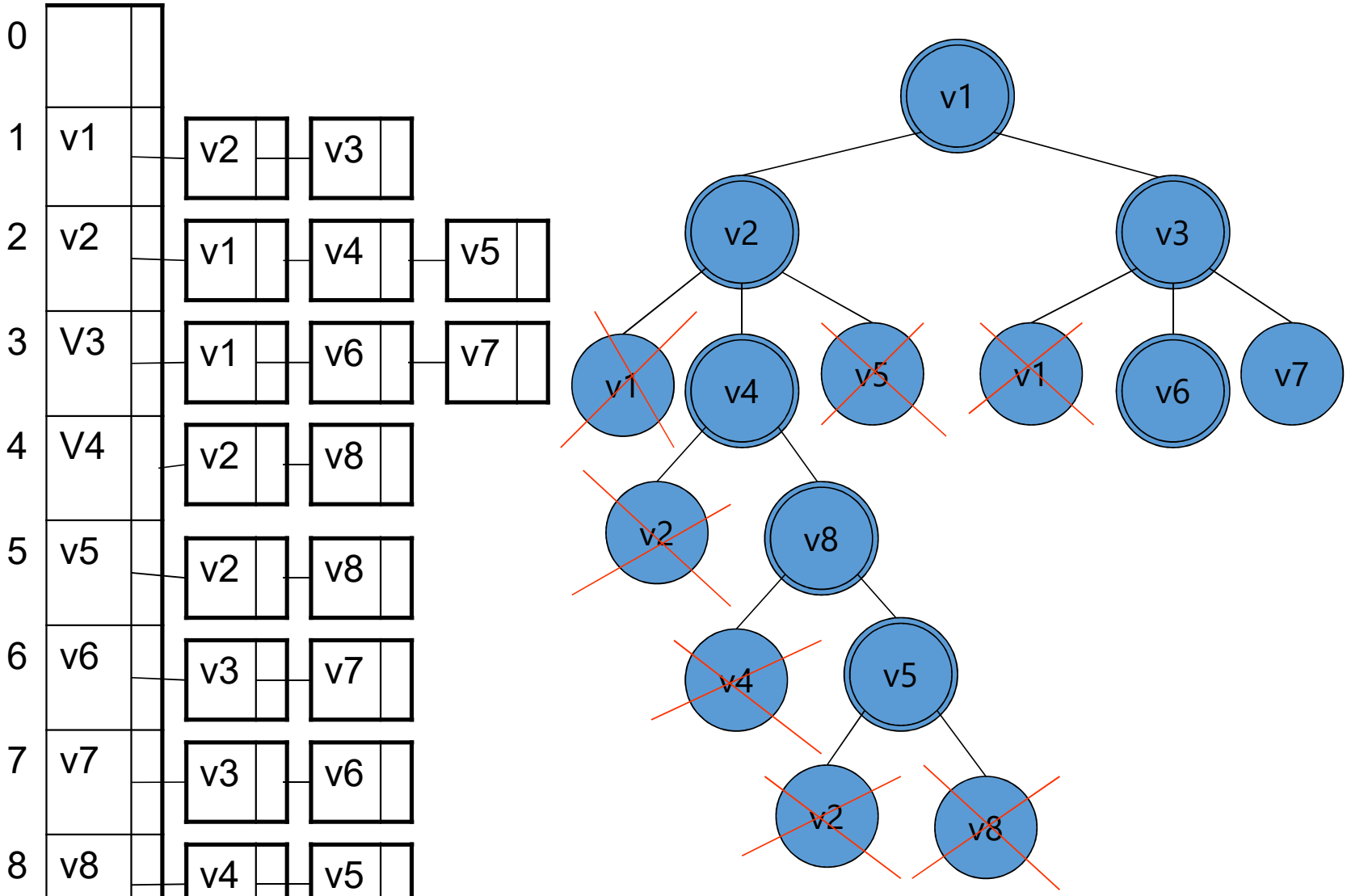


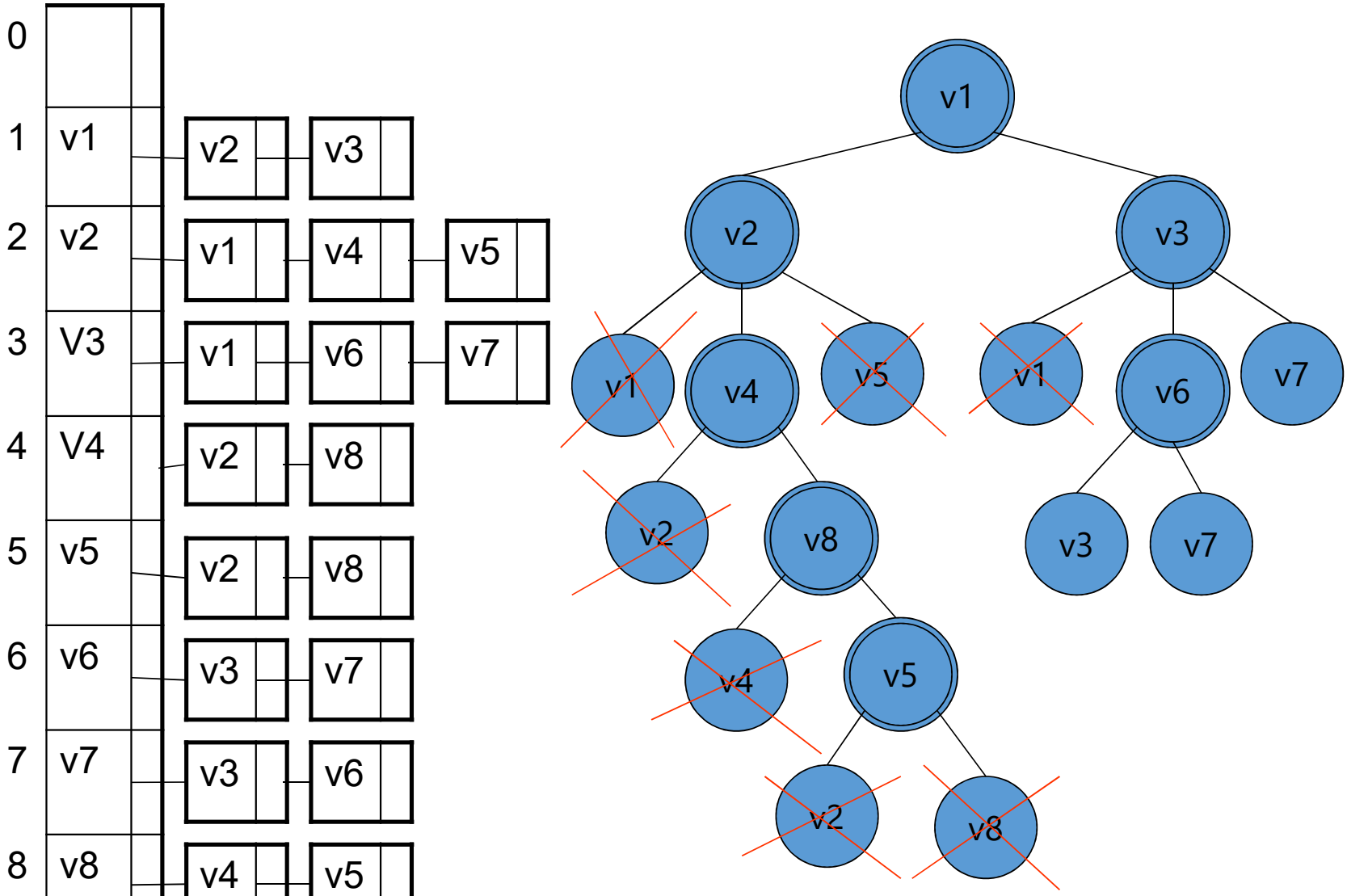


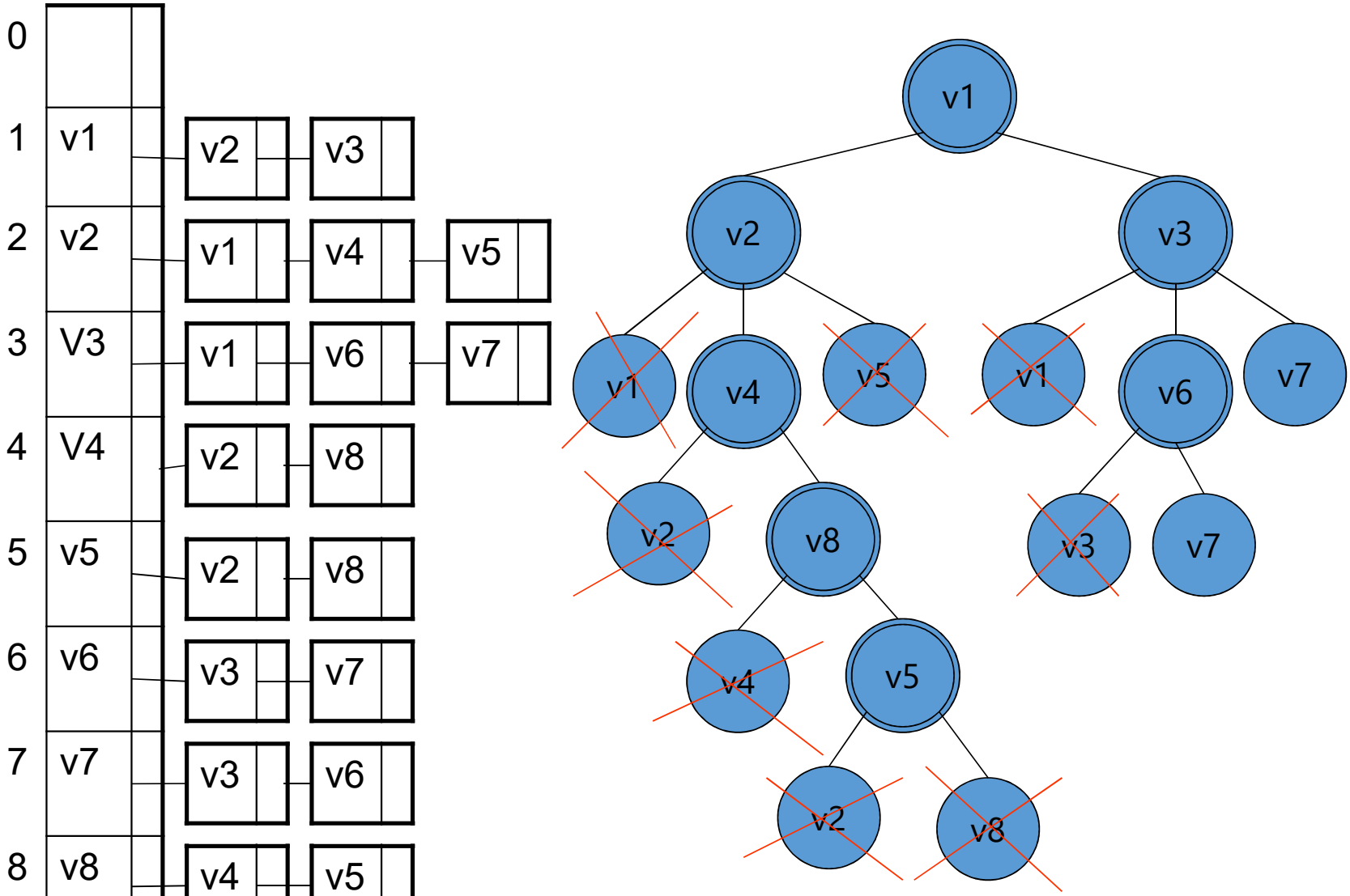


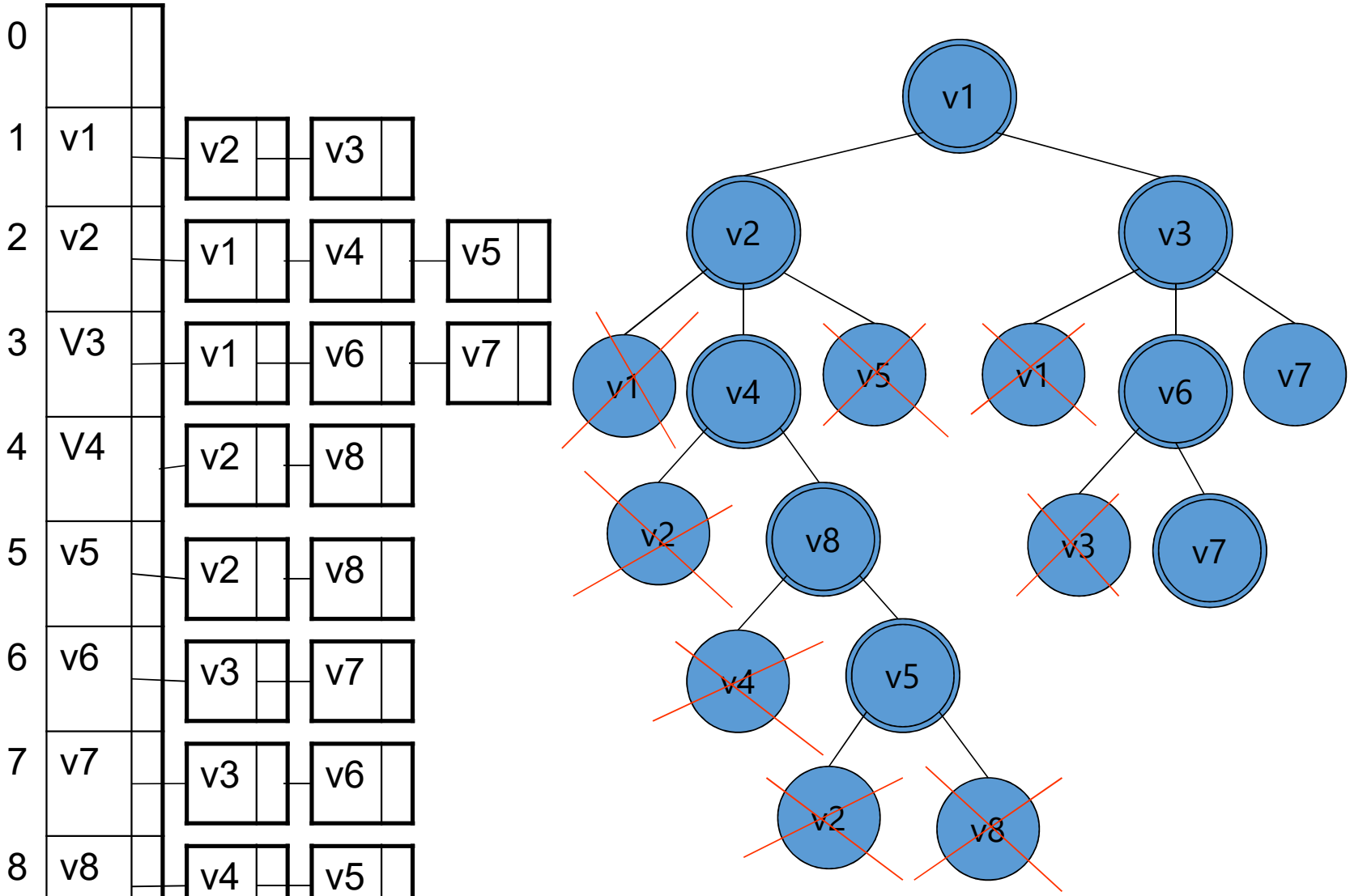


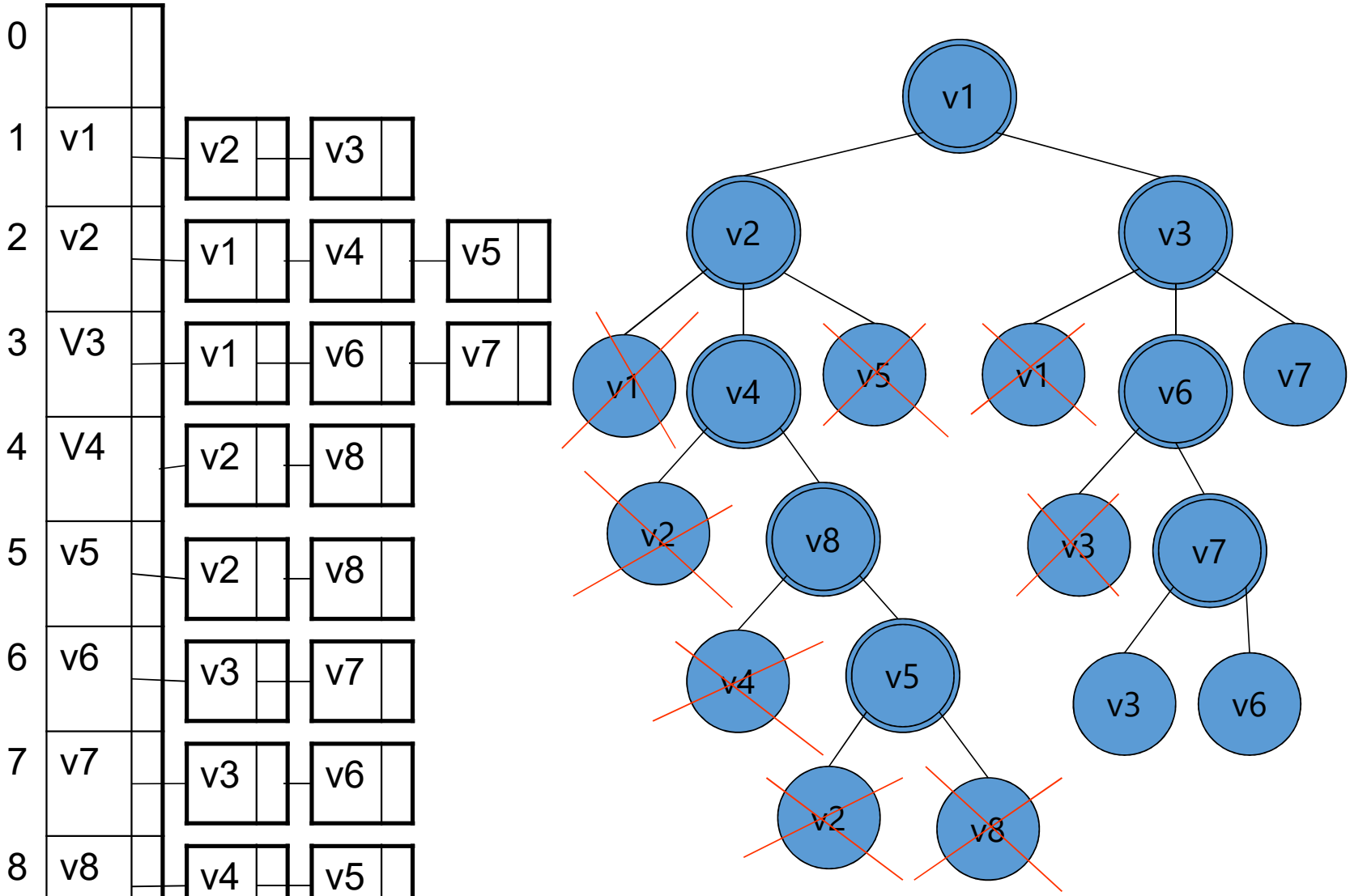


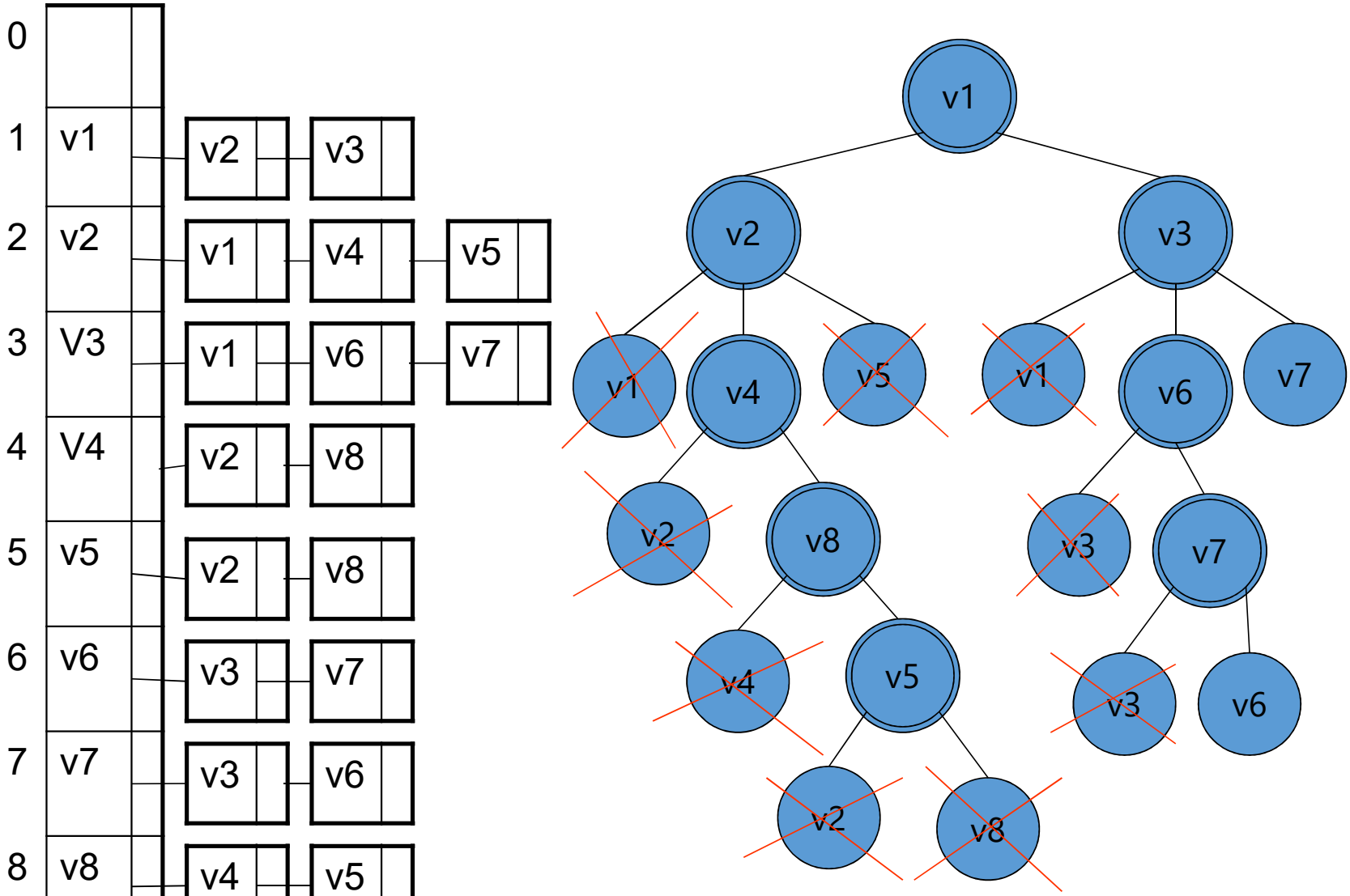


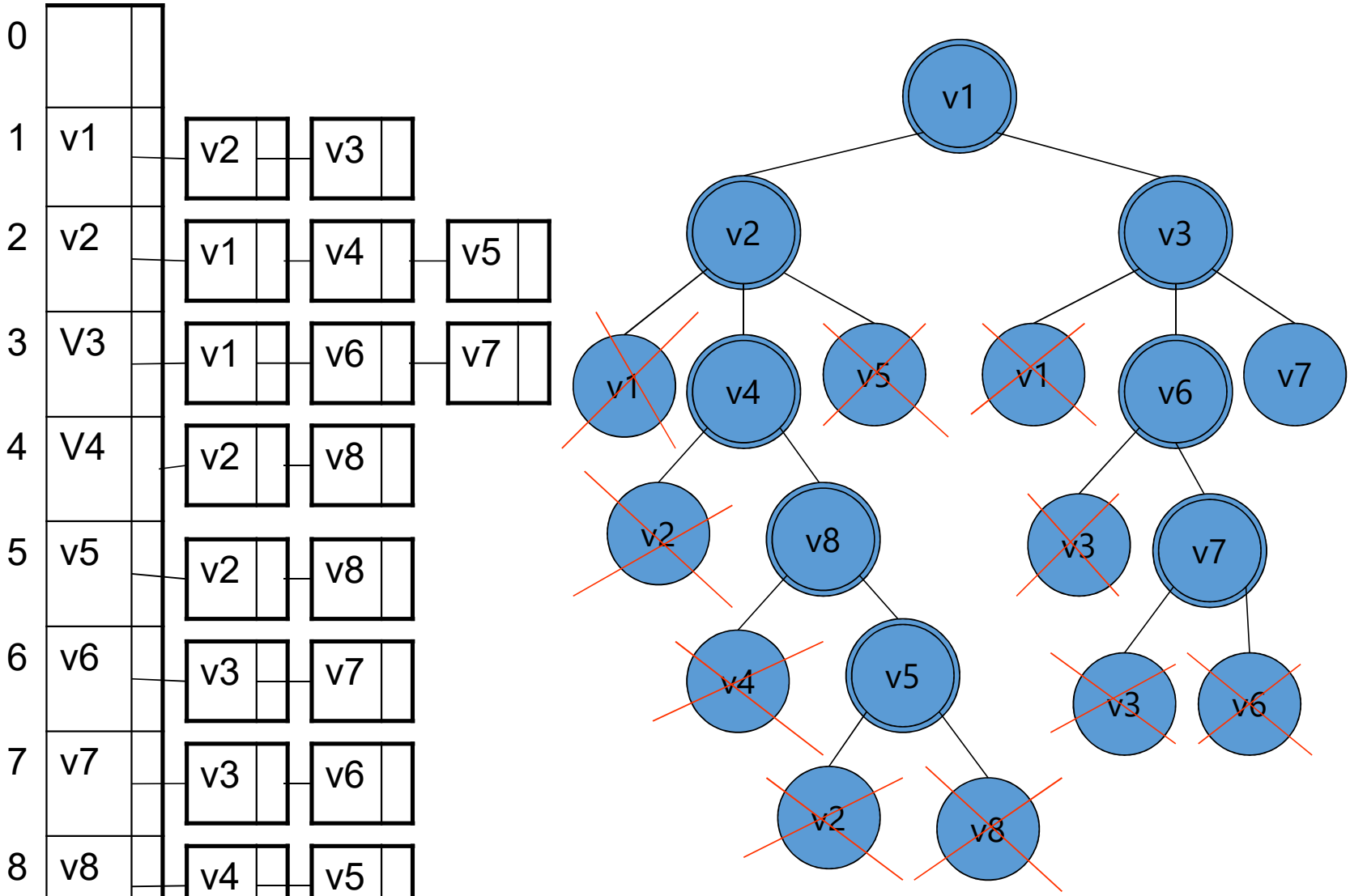


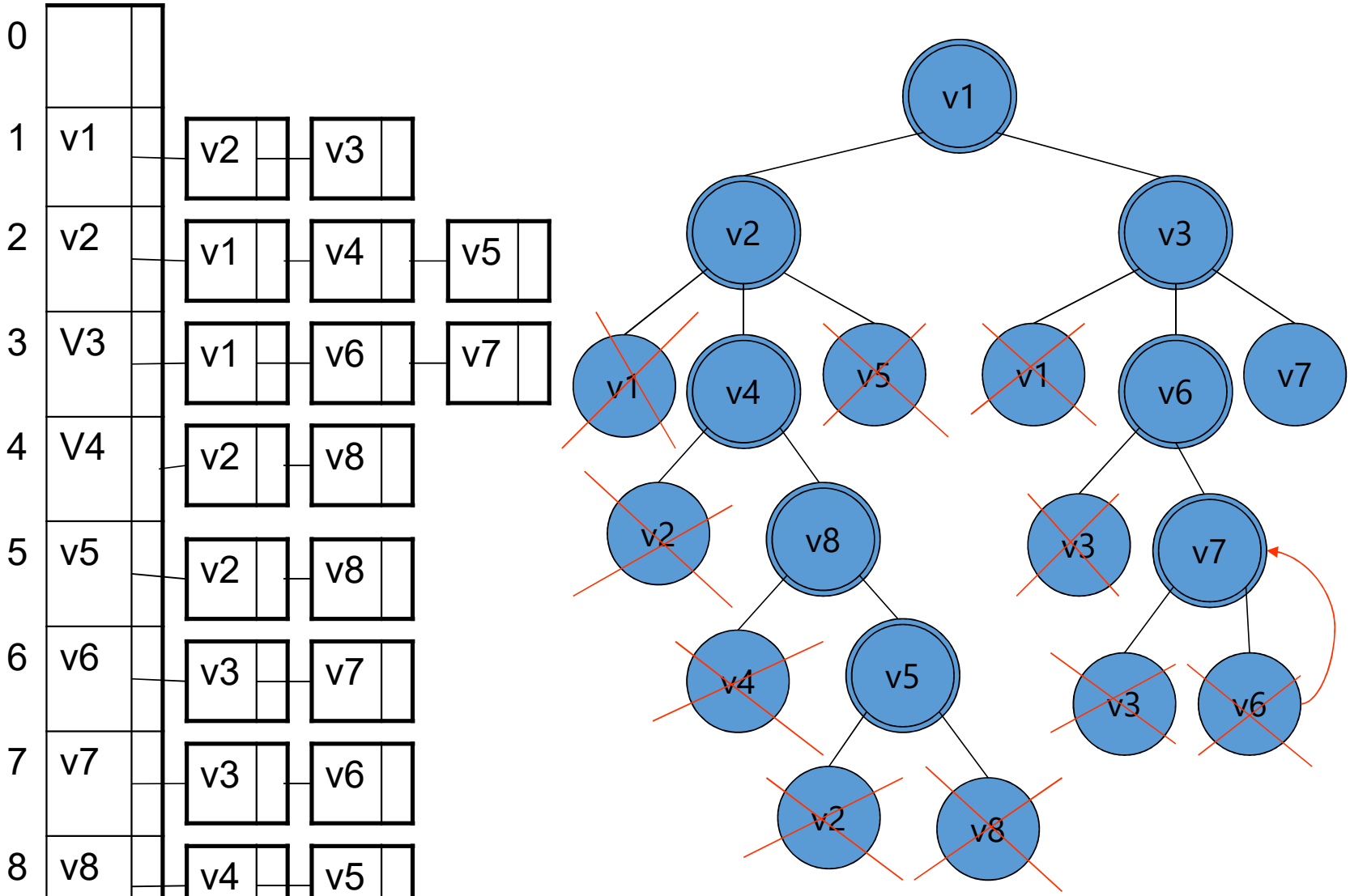


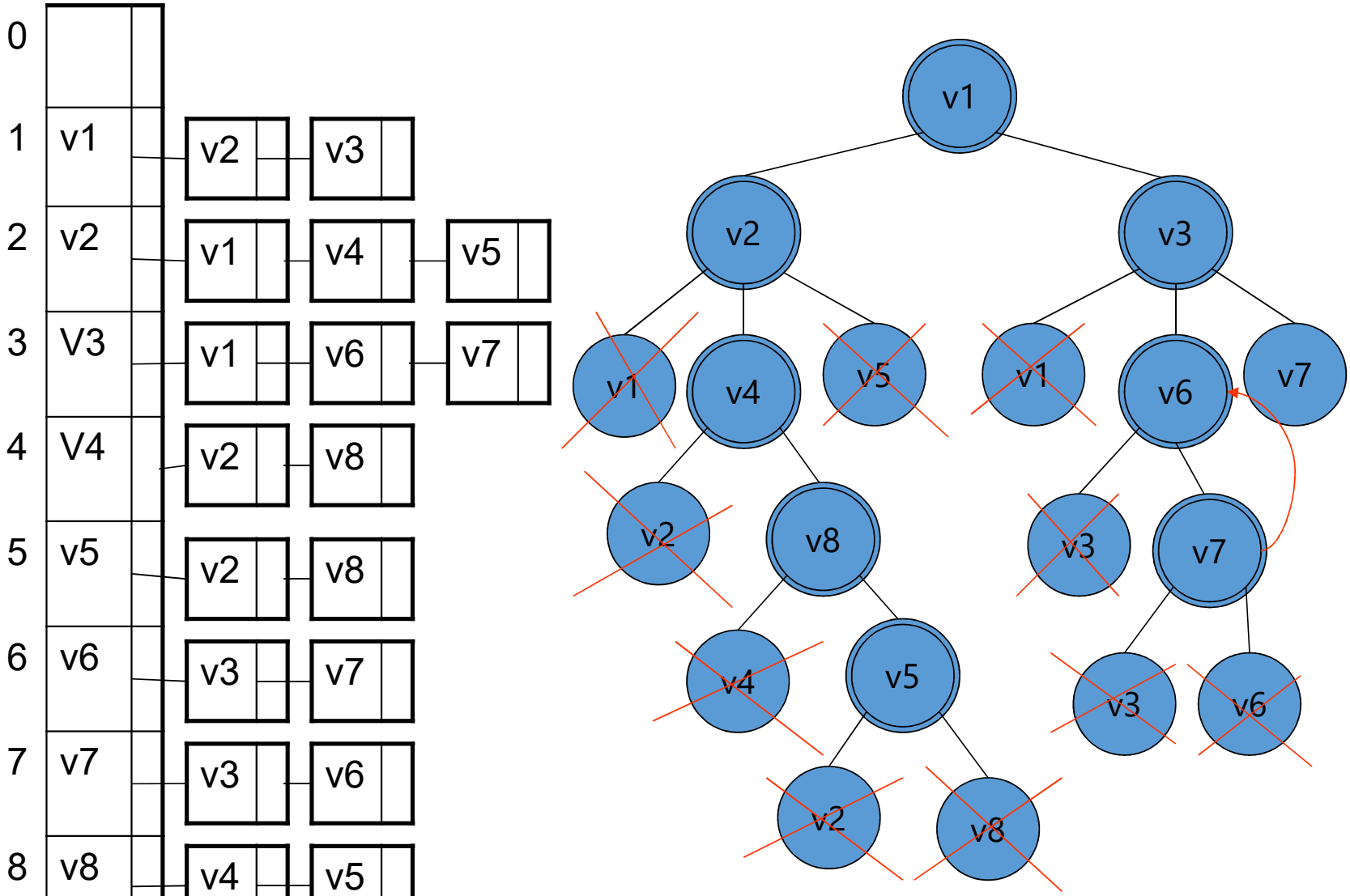


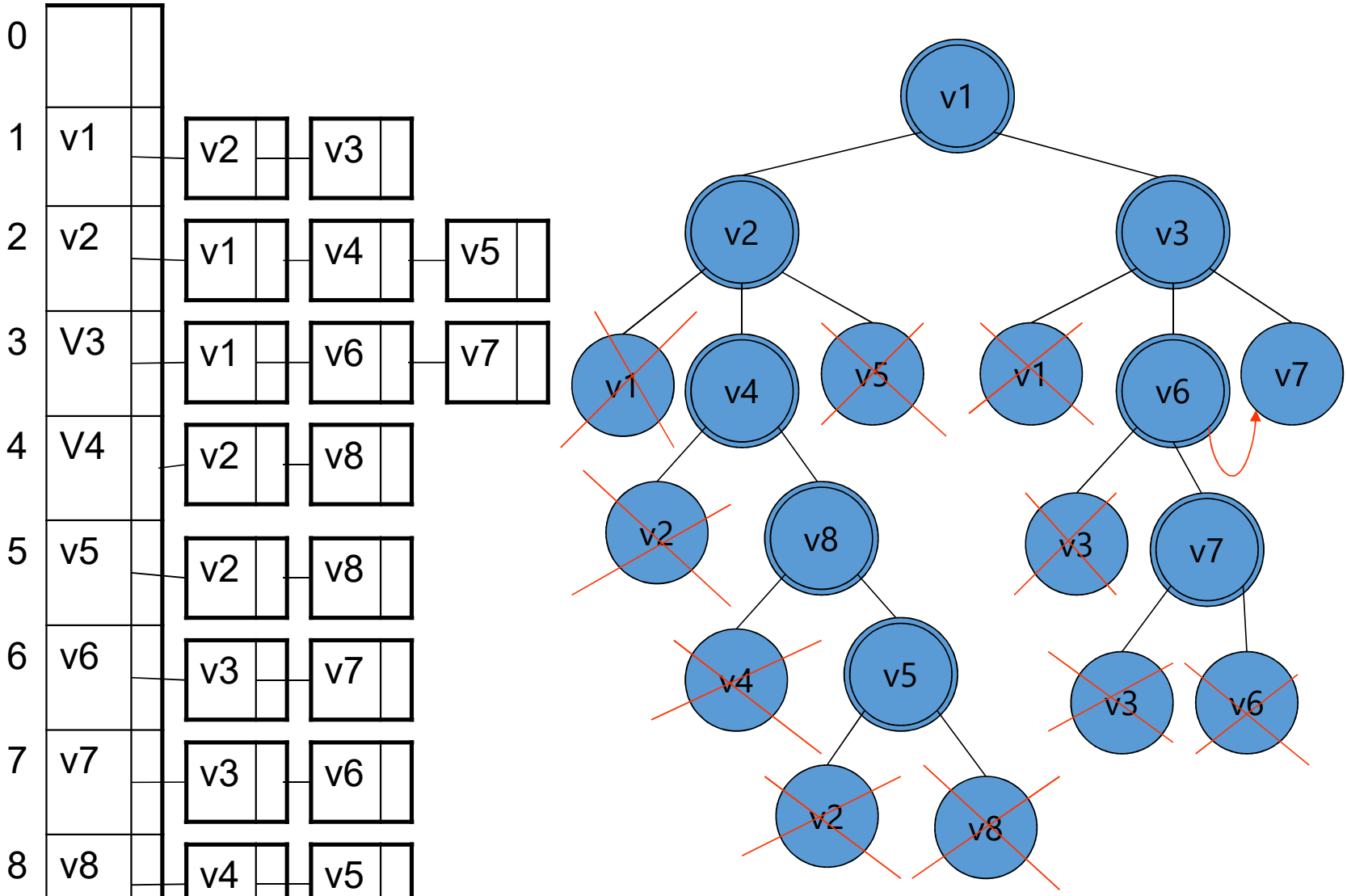


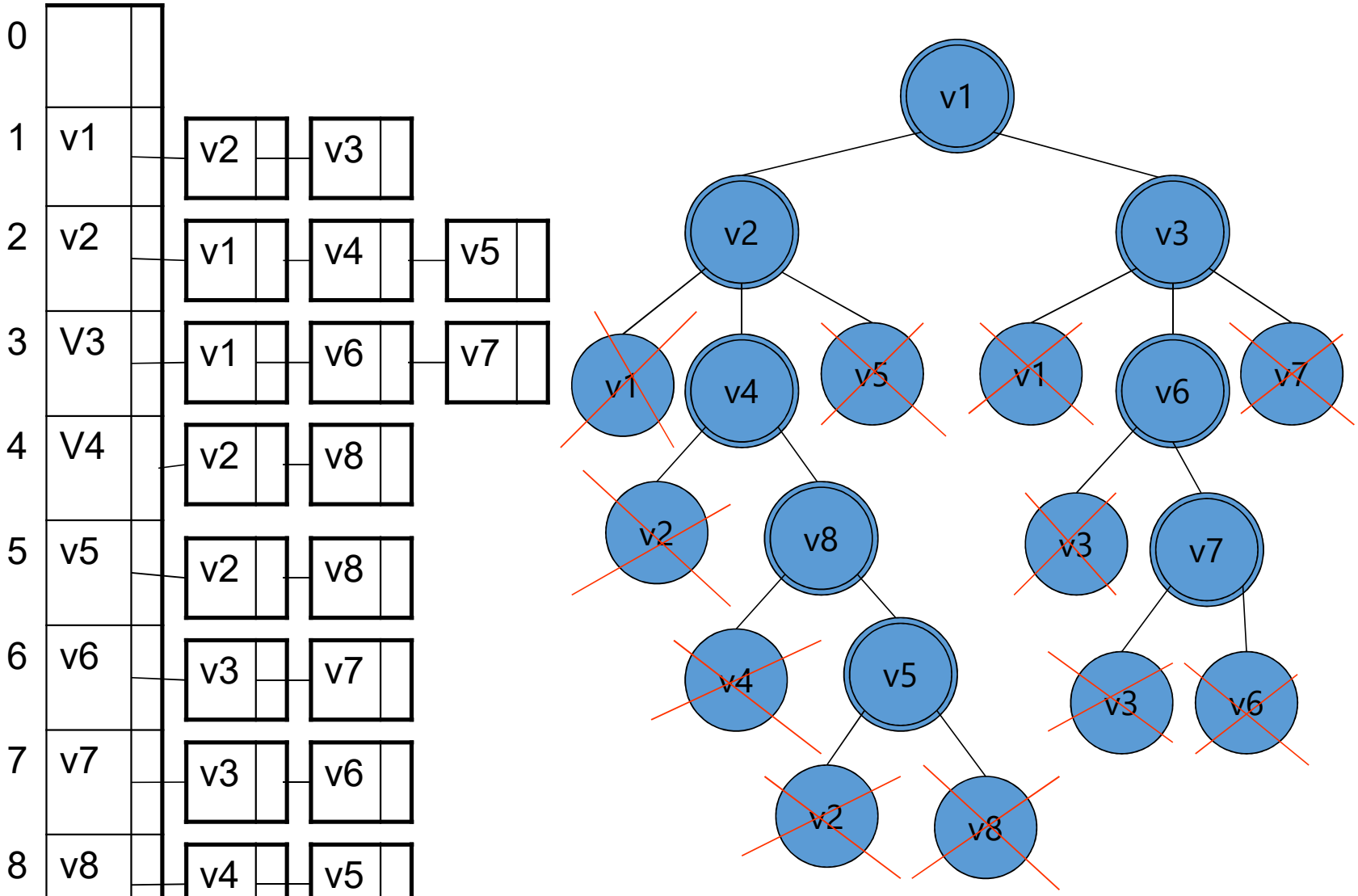


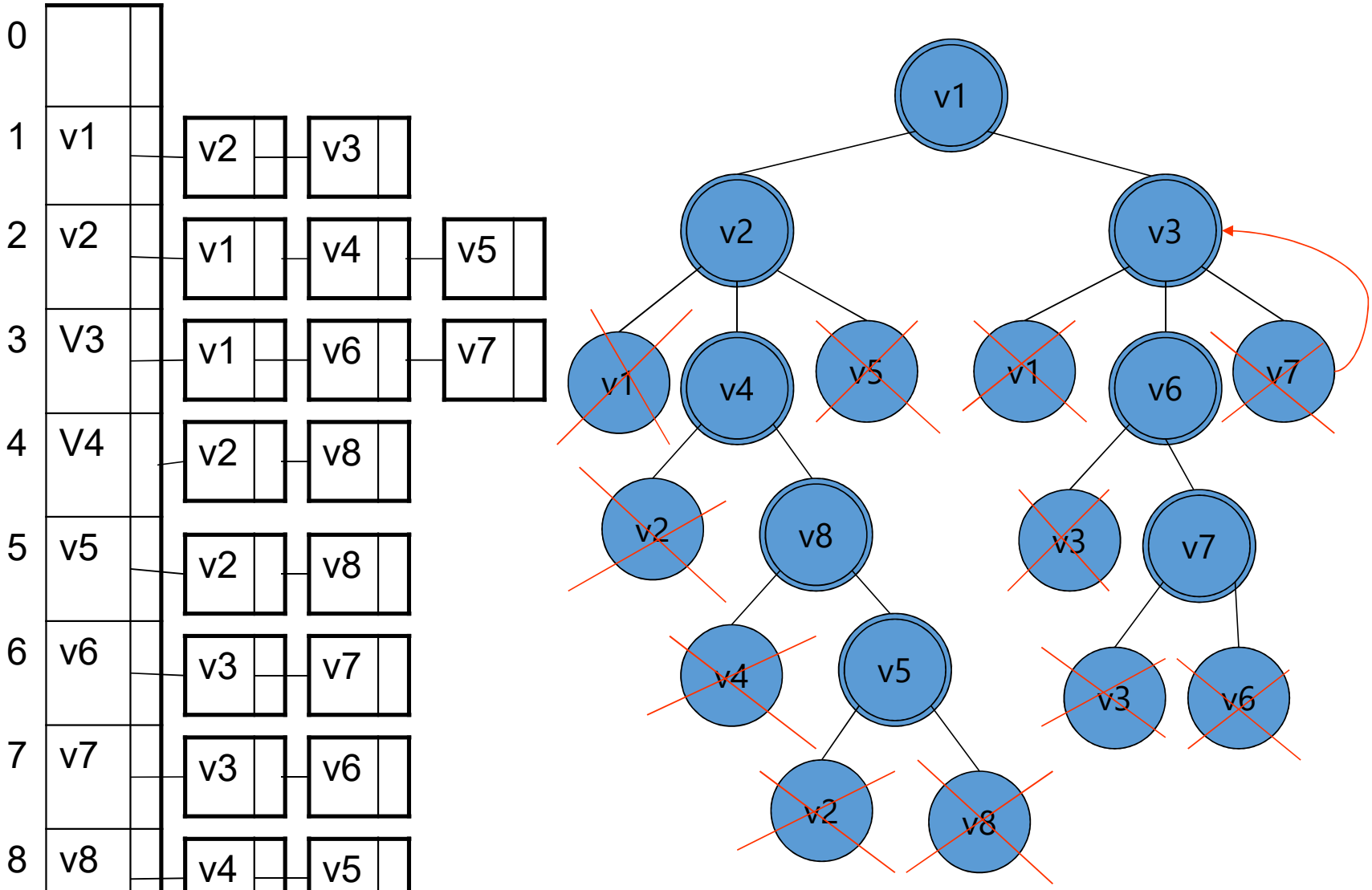


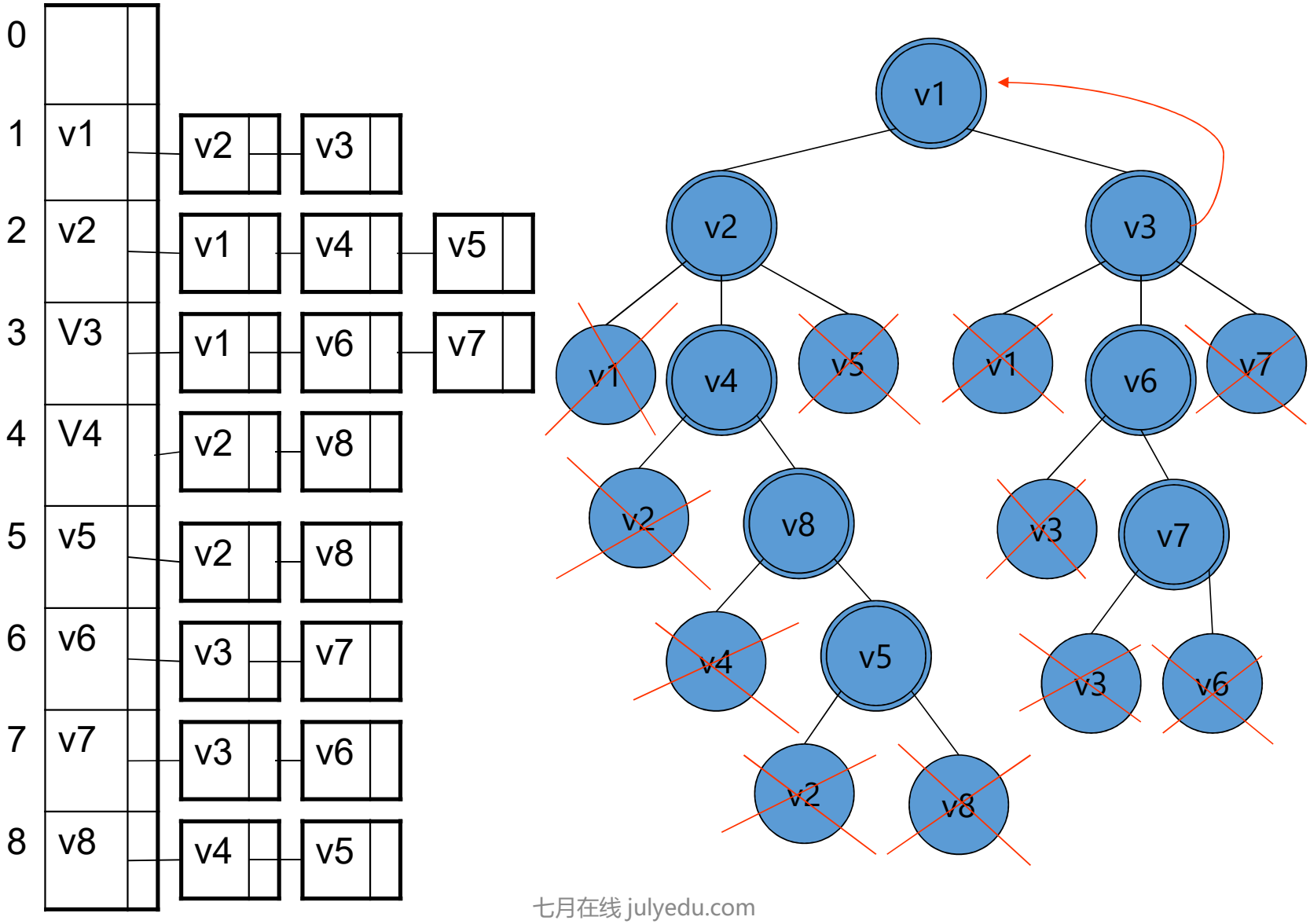




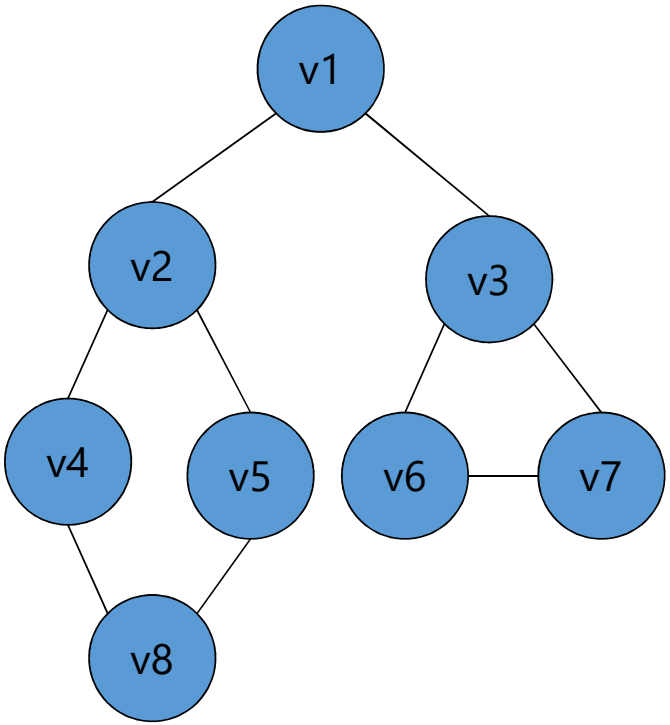


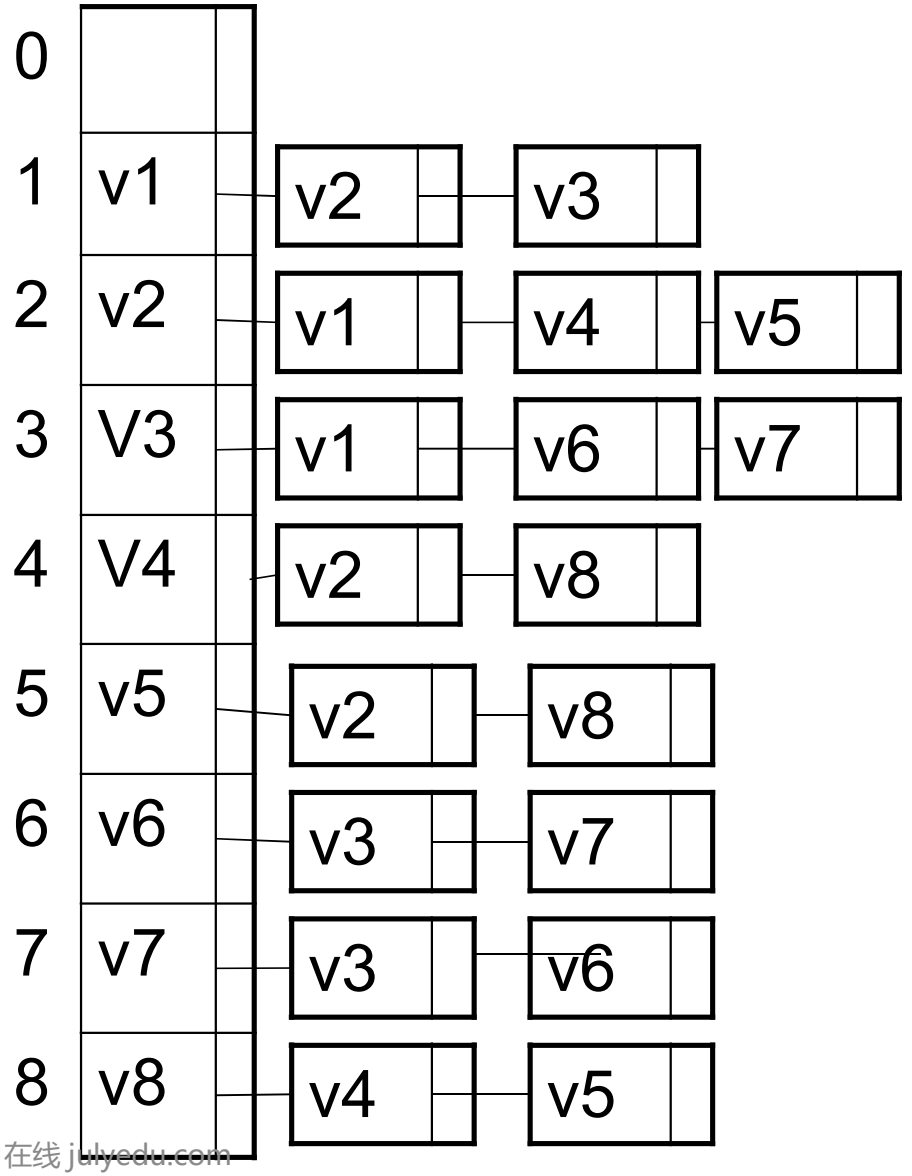


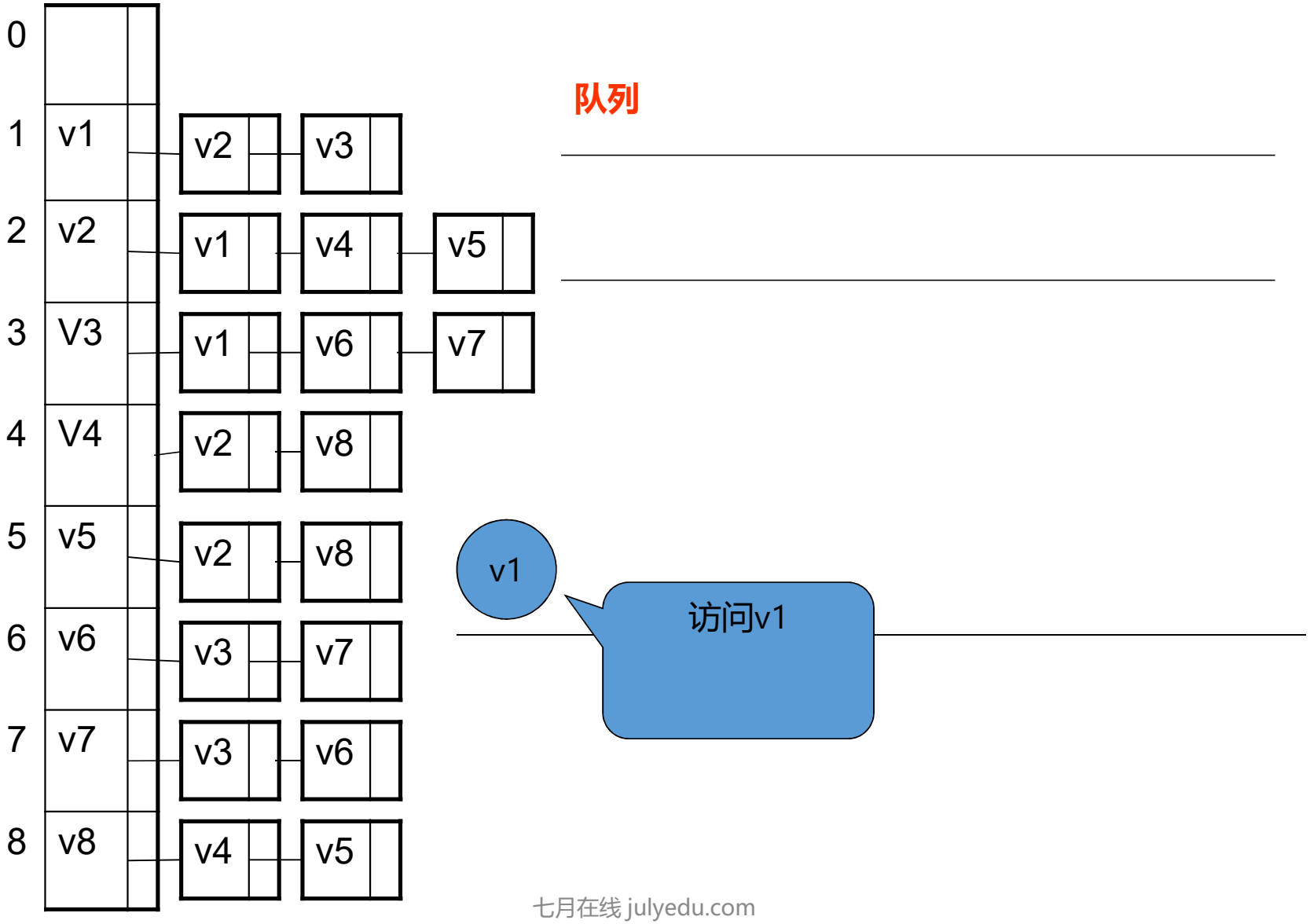


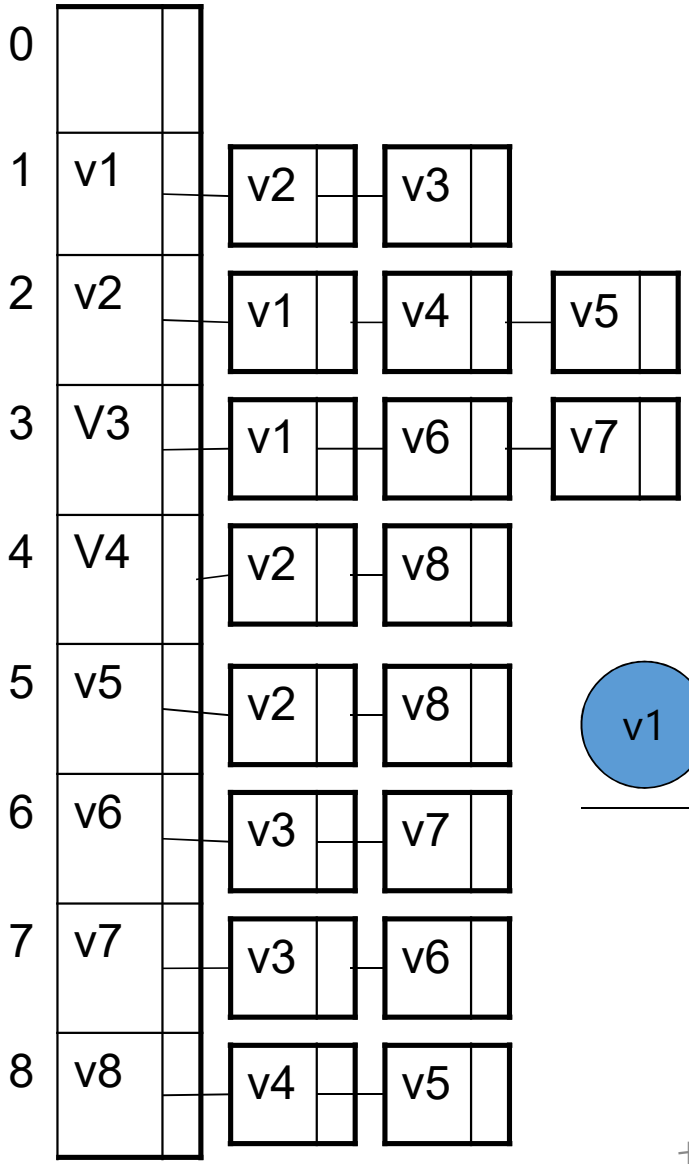


广度优先






0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v1
访问v1

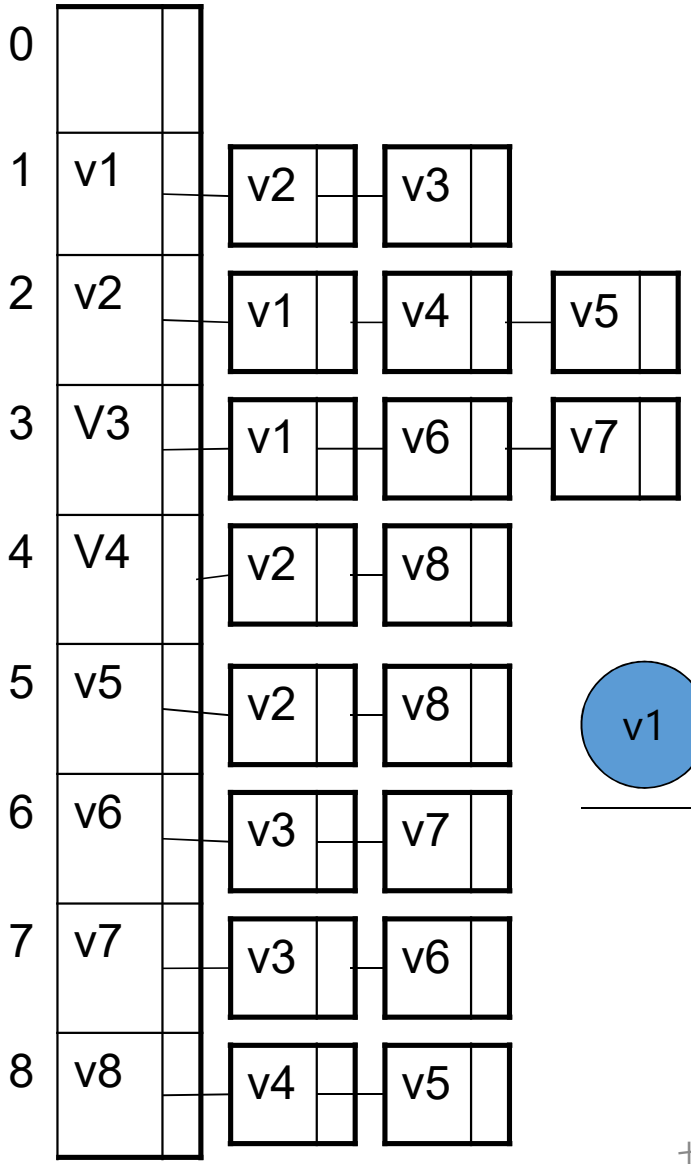


队列

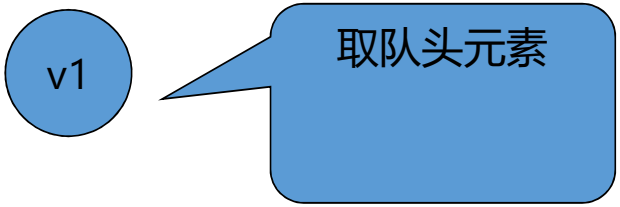
v1

V1入队列

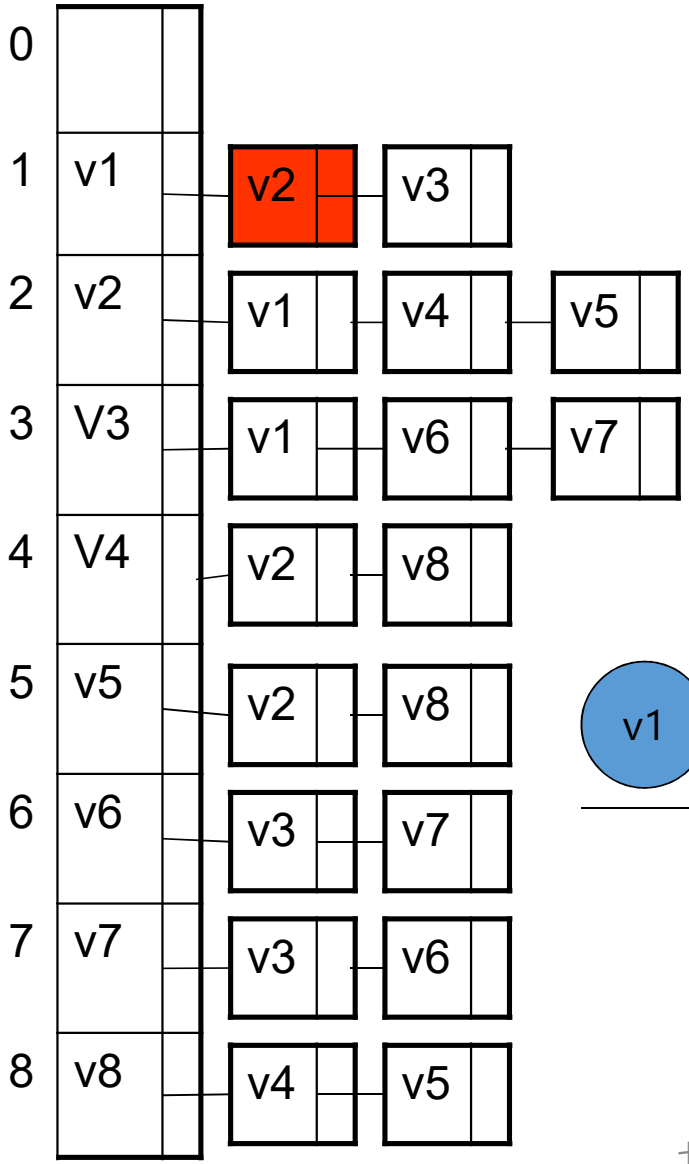
v1


0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5

队列


v1
取队头元素

v1

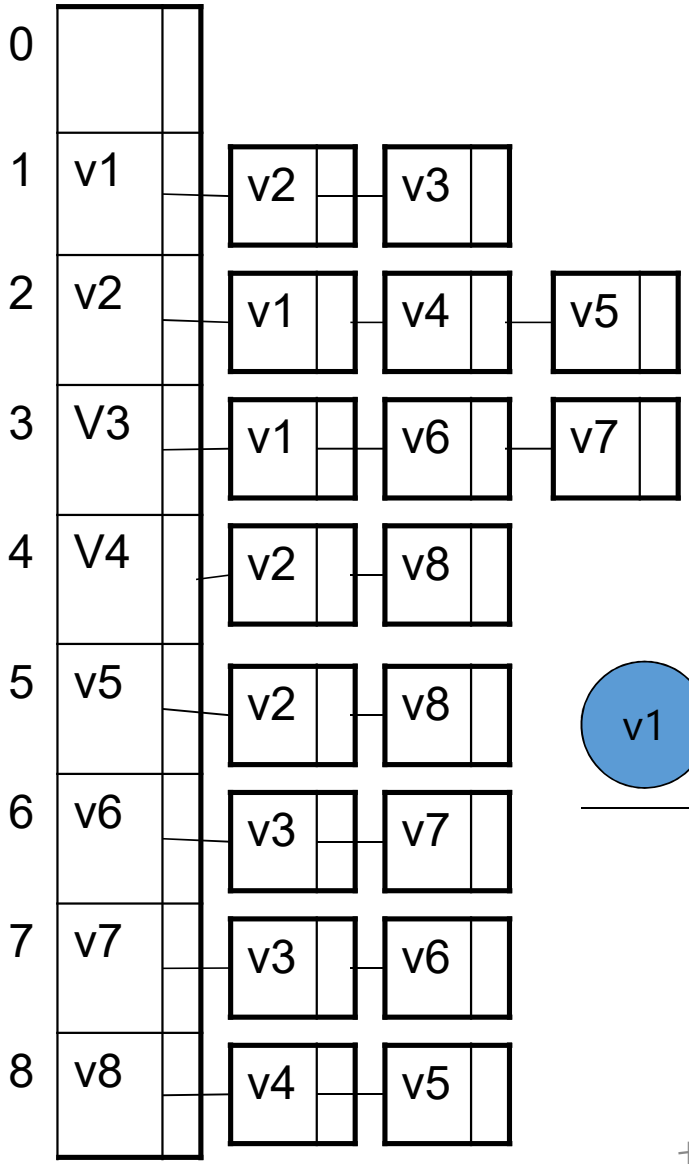


队列

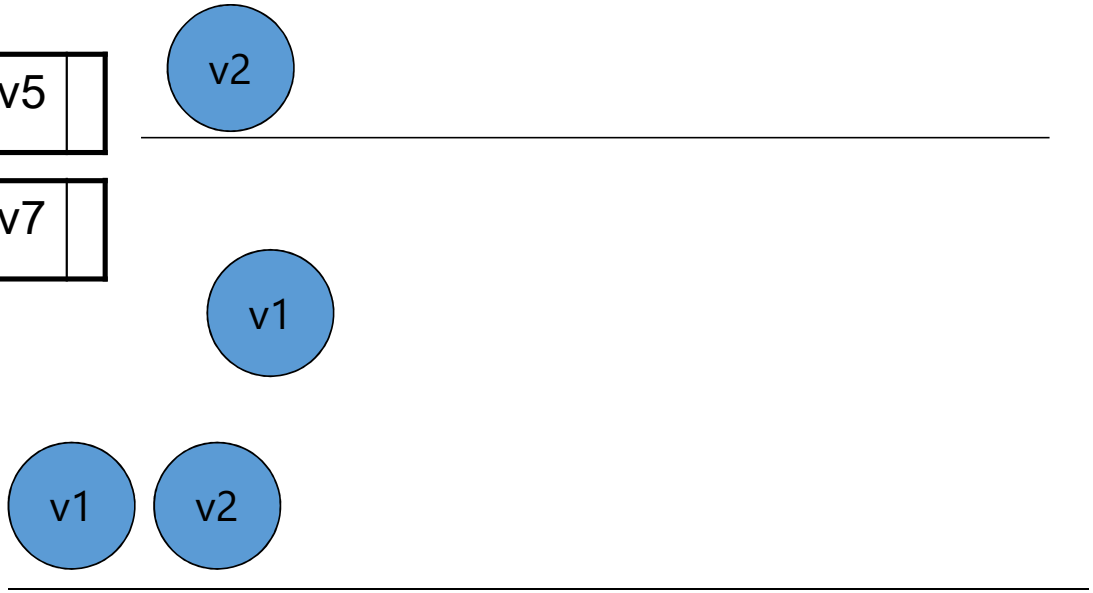
v1

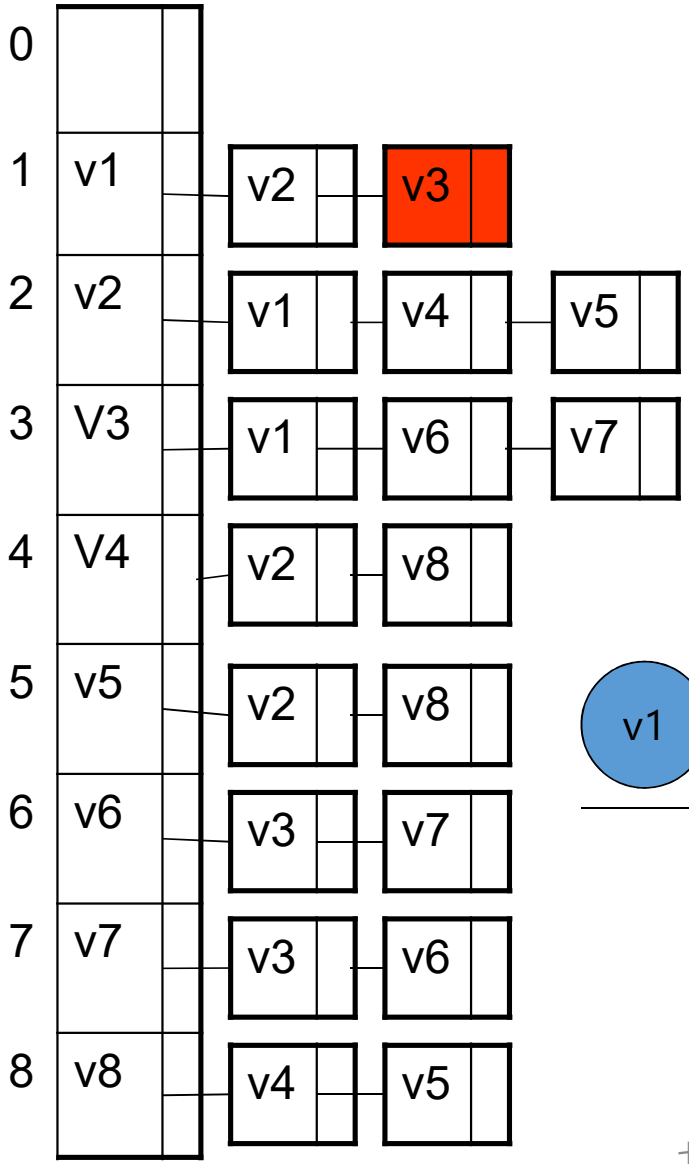
V1的邻接点v2没有被访问过，访问之，且入队列

v1 v2

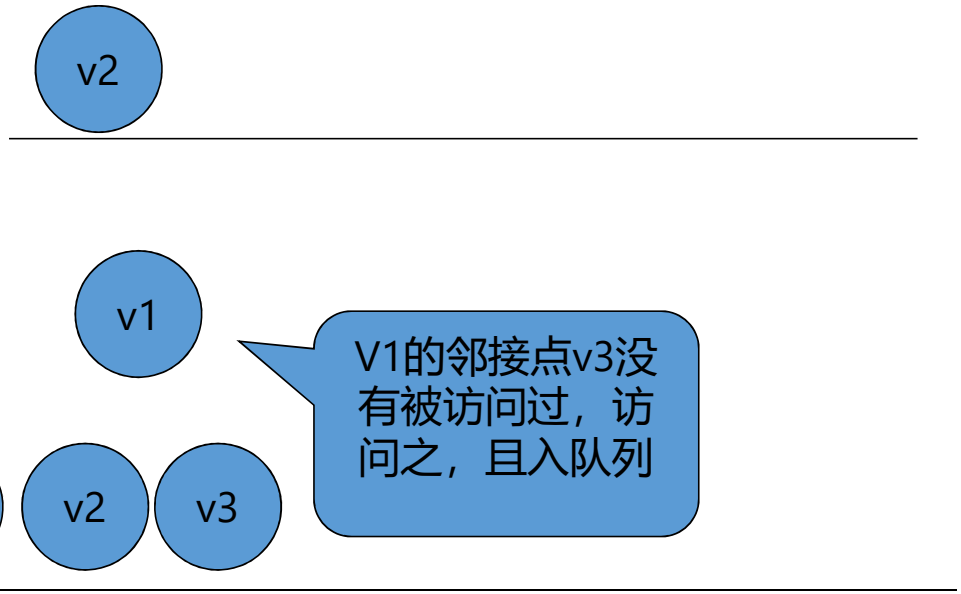

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5

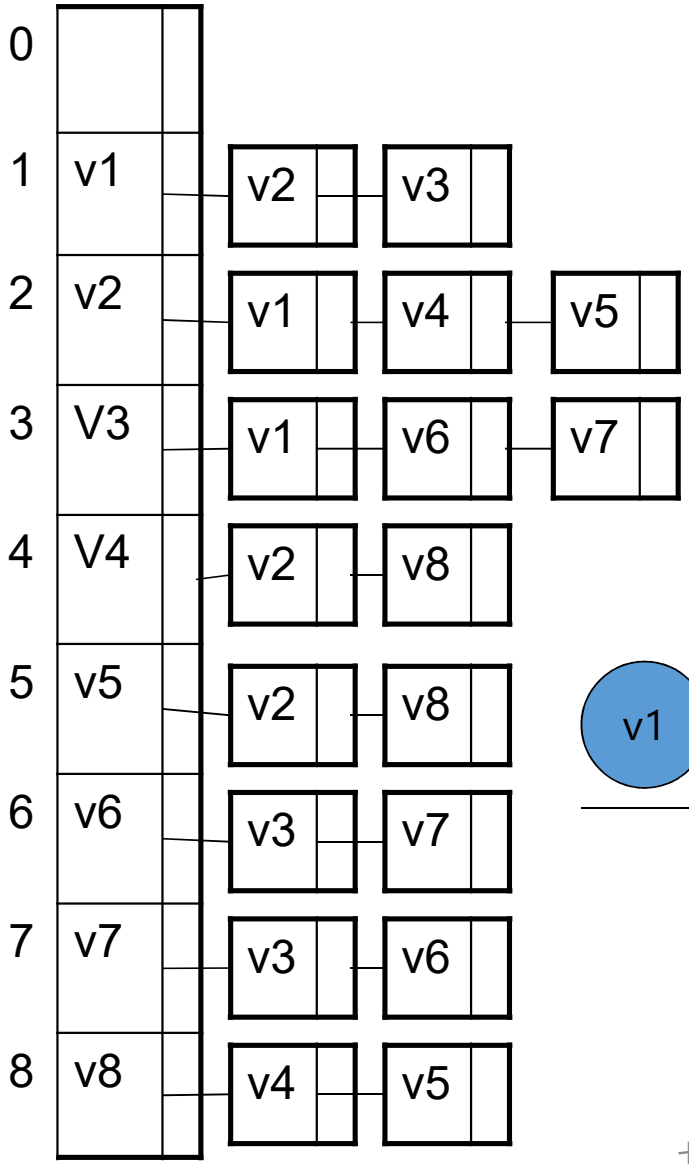
队列


v2
v1
v1
v2

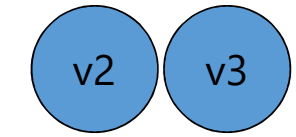

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5

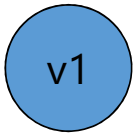
队列

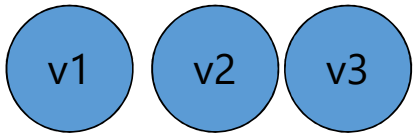

v2
v1
v1
v2
v3
V1的邻接点v3没有被访问过，访问之，且入队列

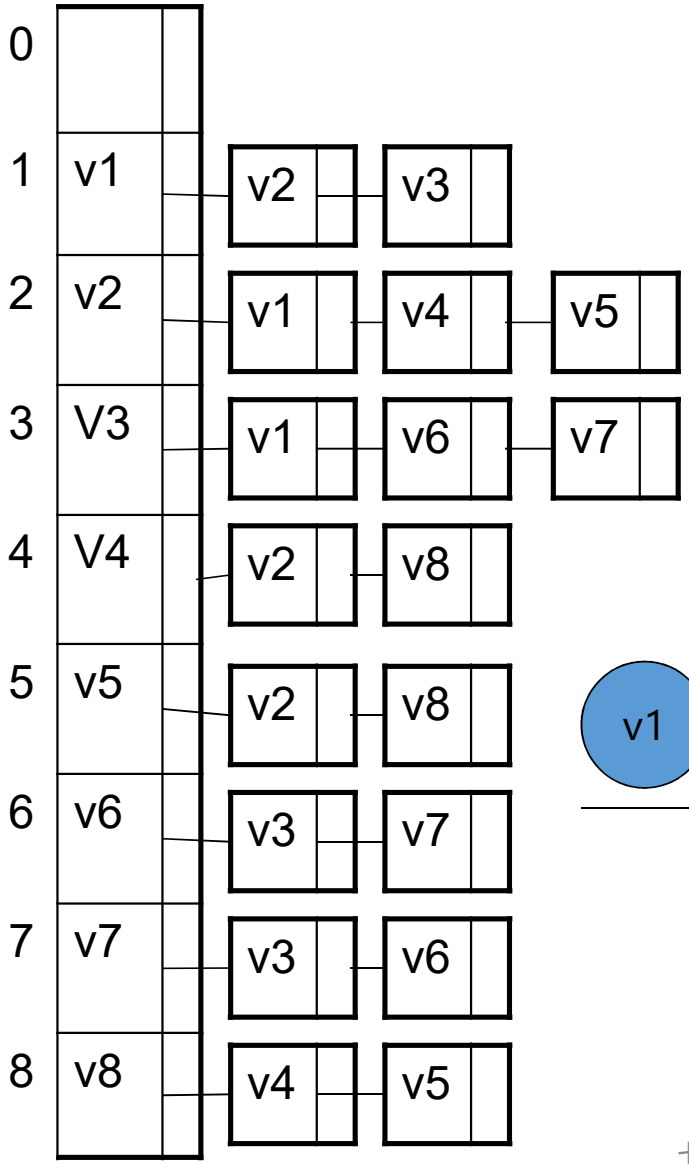


队列

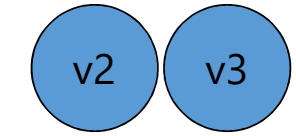


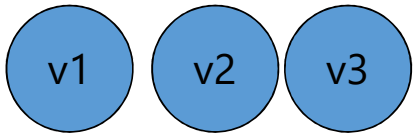


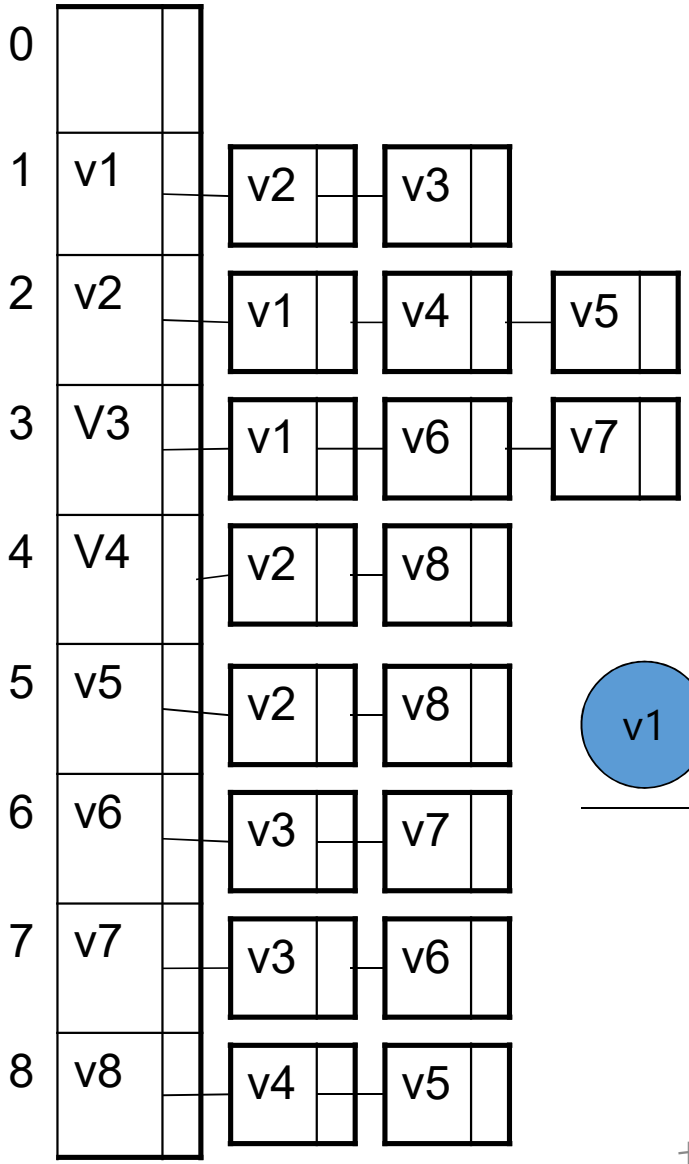


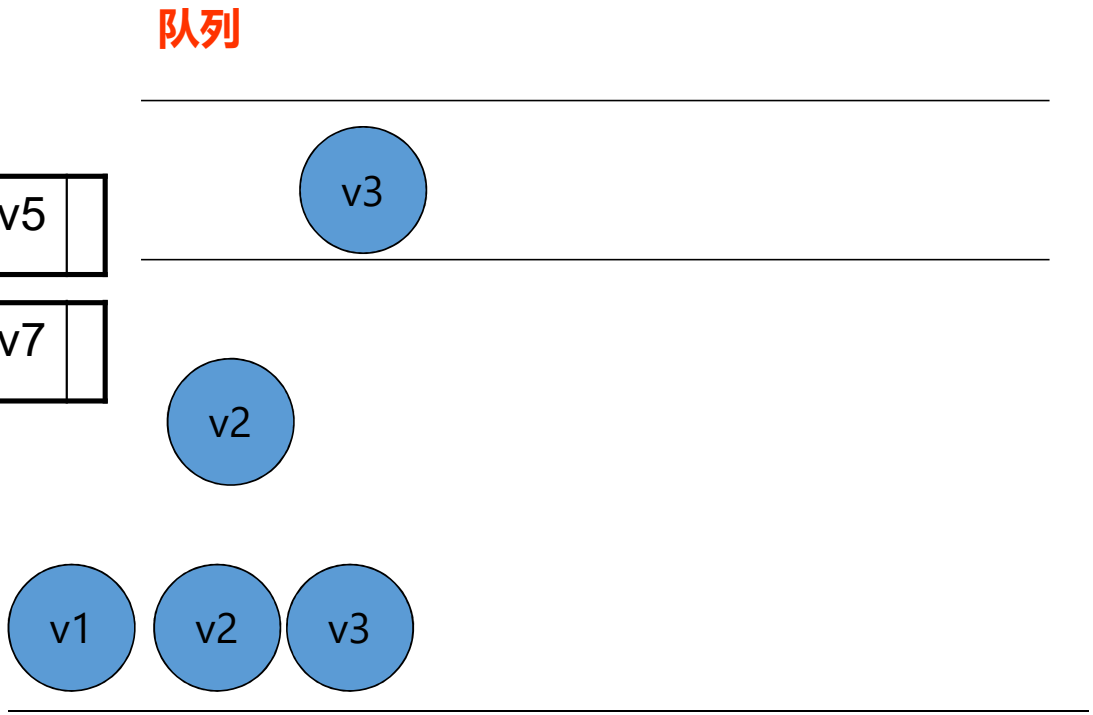


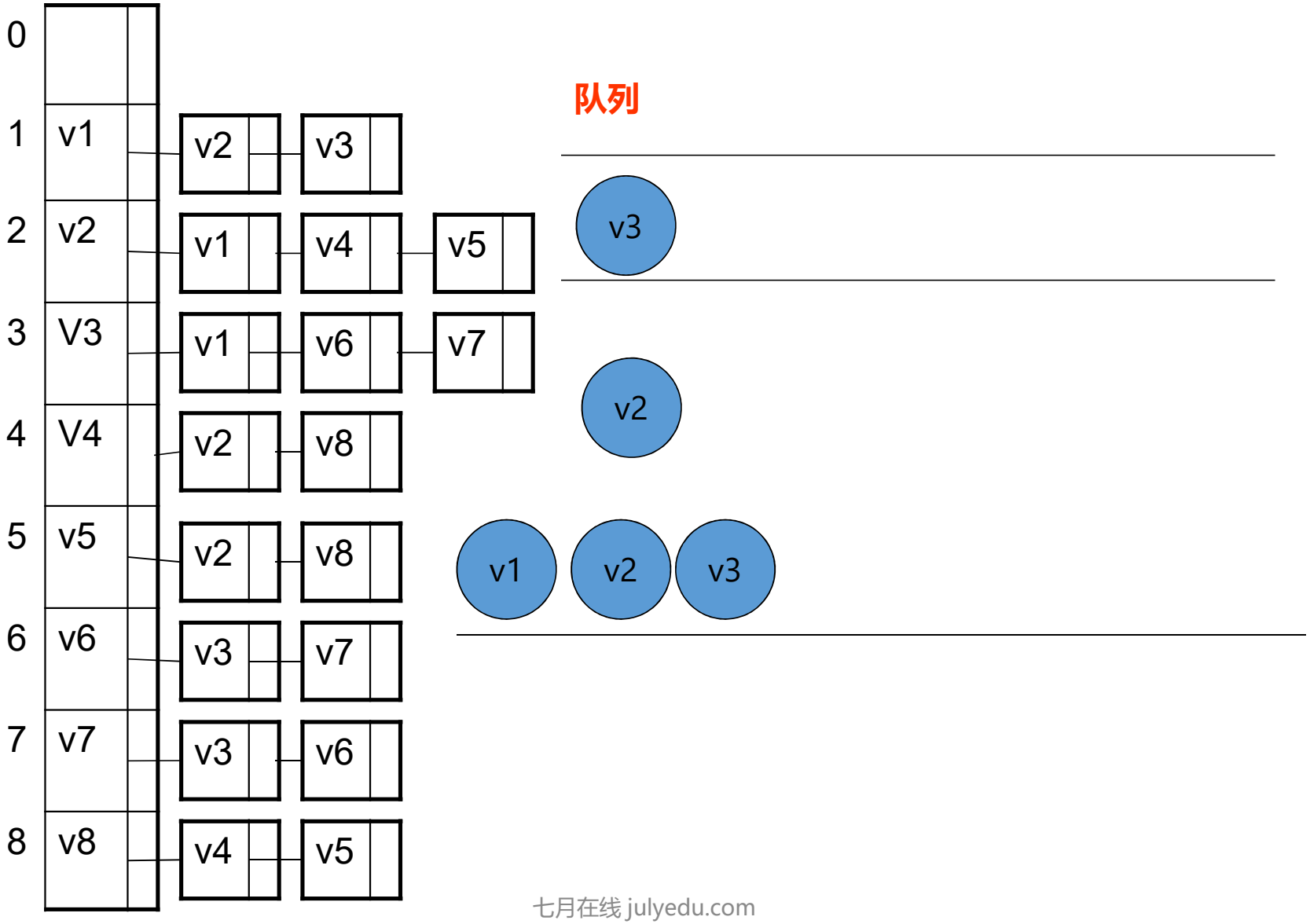
队列

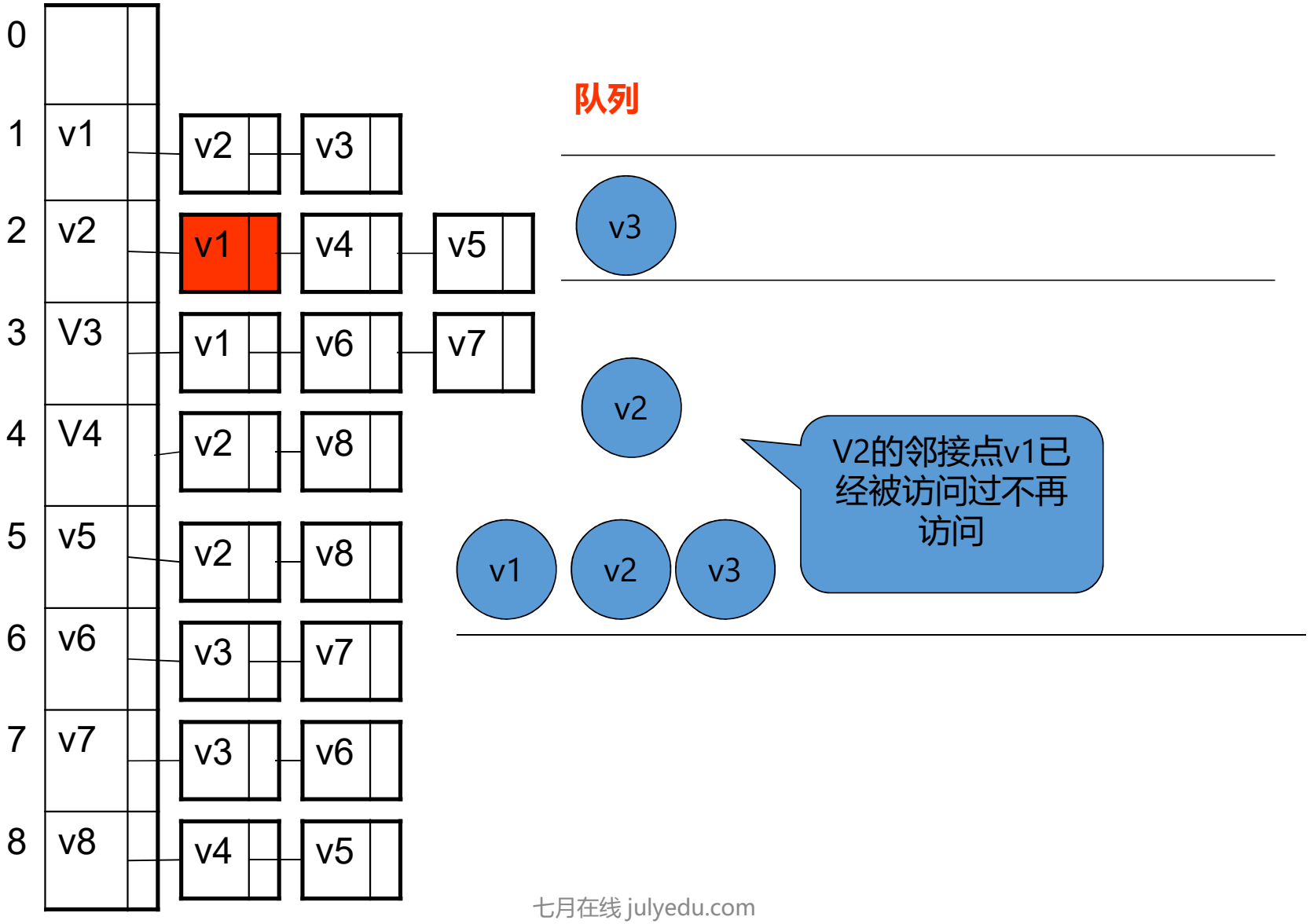


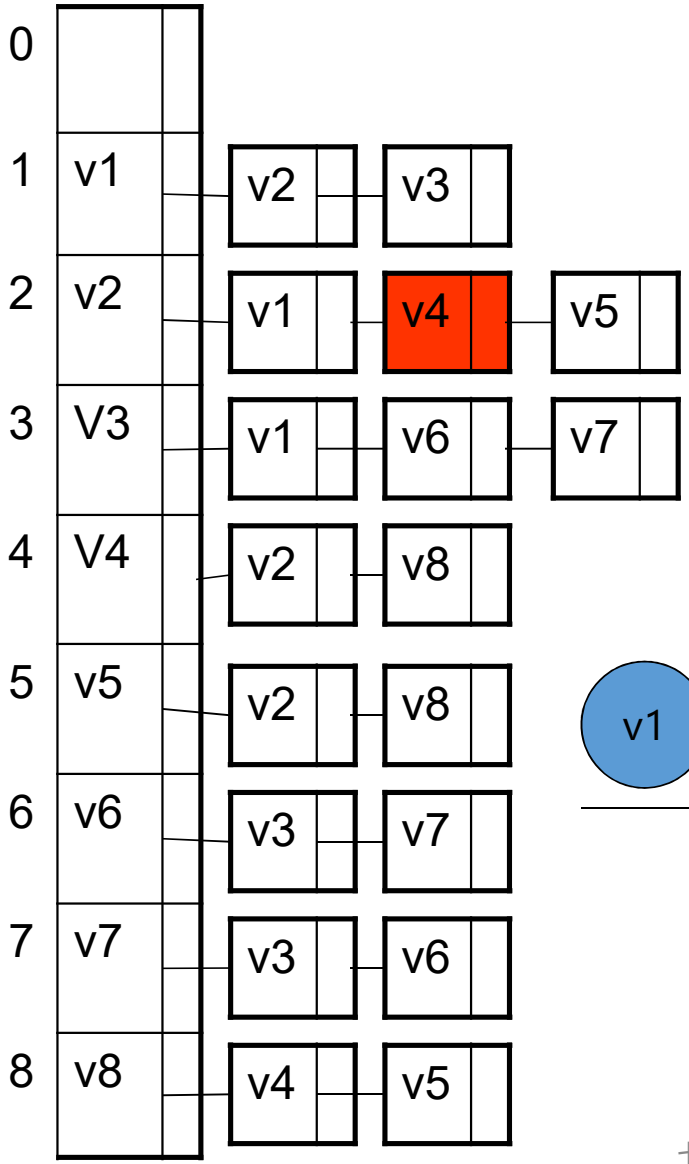




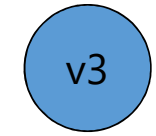


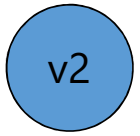


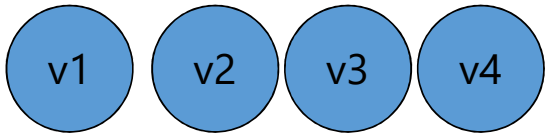

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v3
v2
V2的邻接点v1已经被访问过不再访问
v1
v2
v3



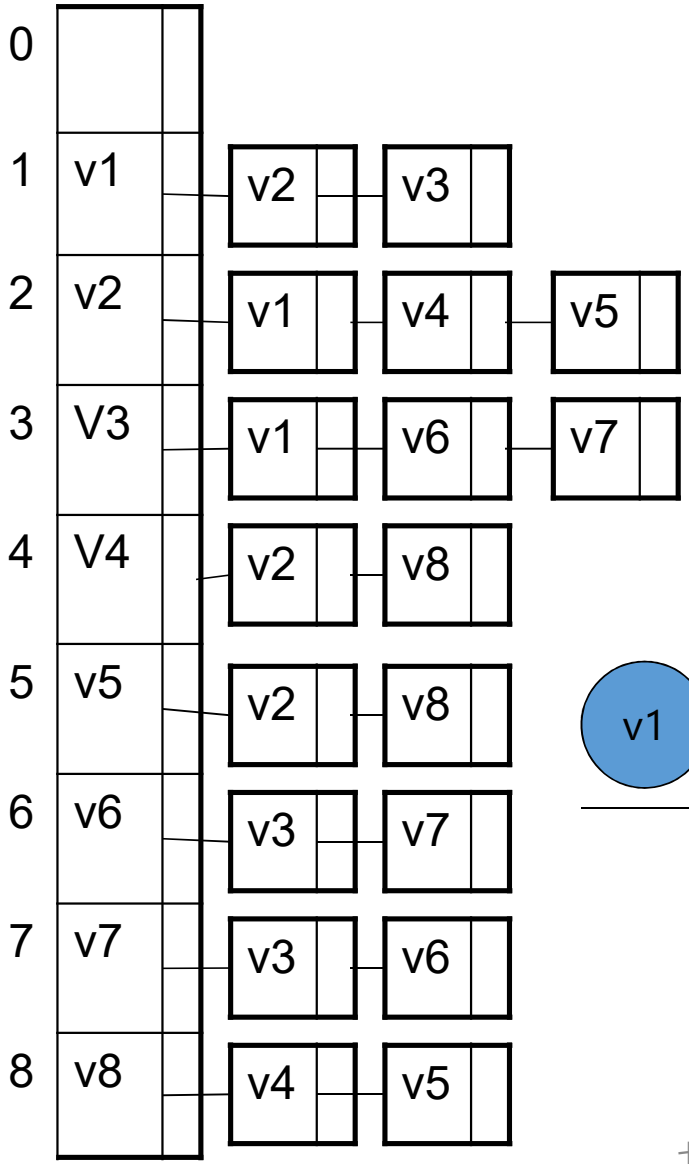
队列



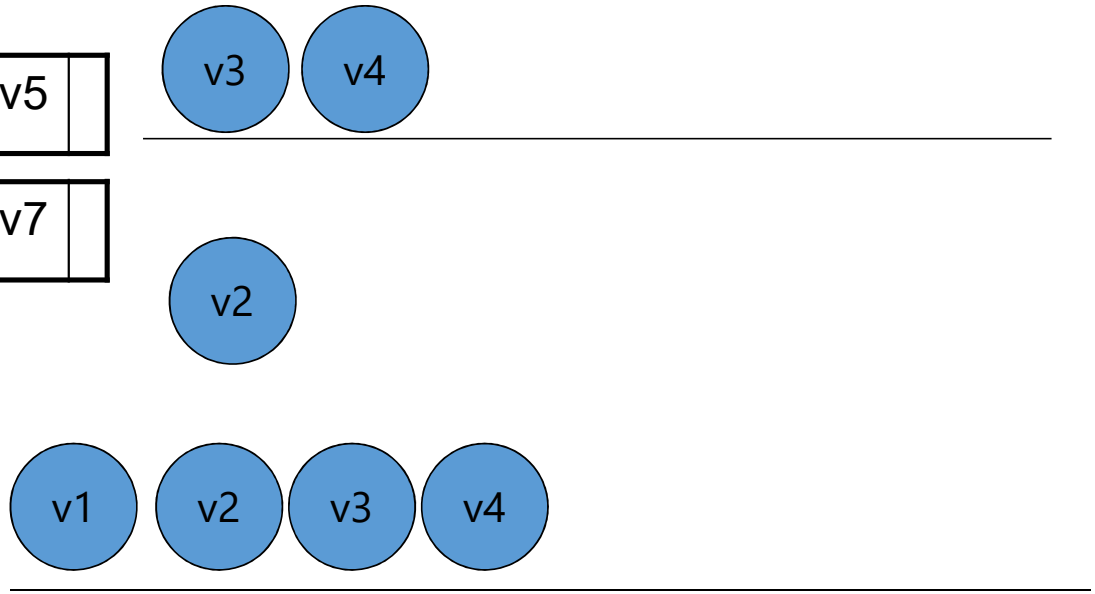


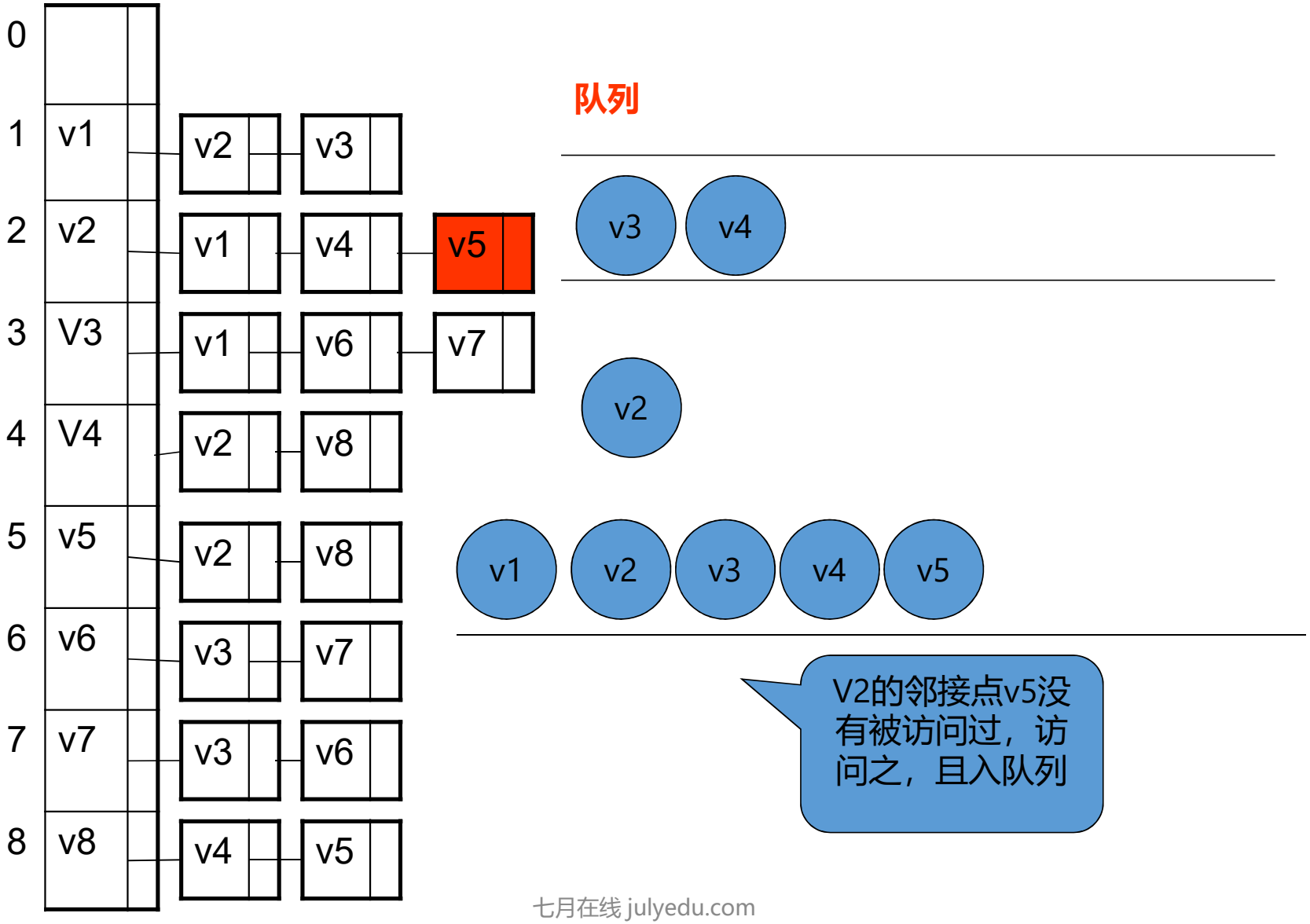


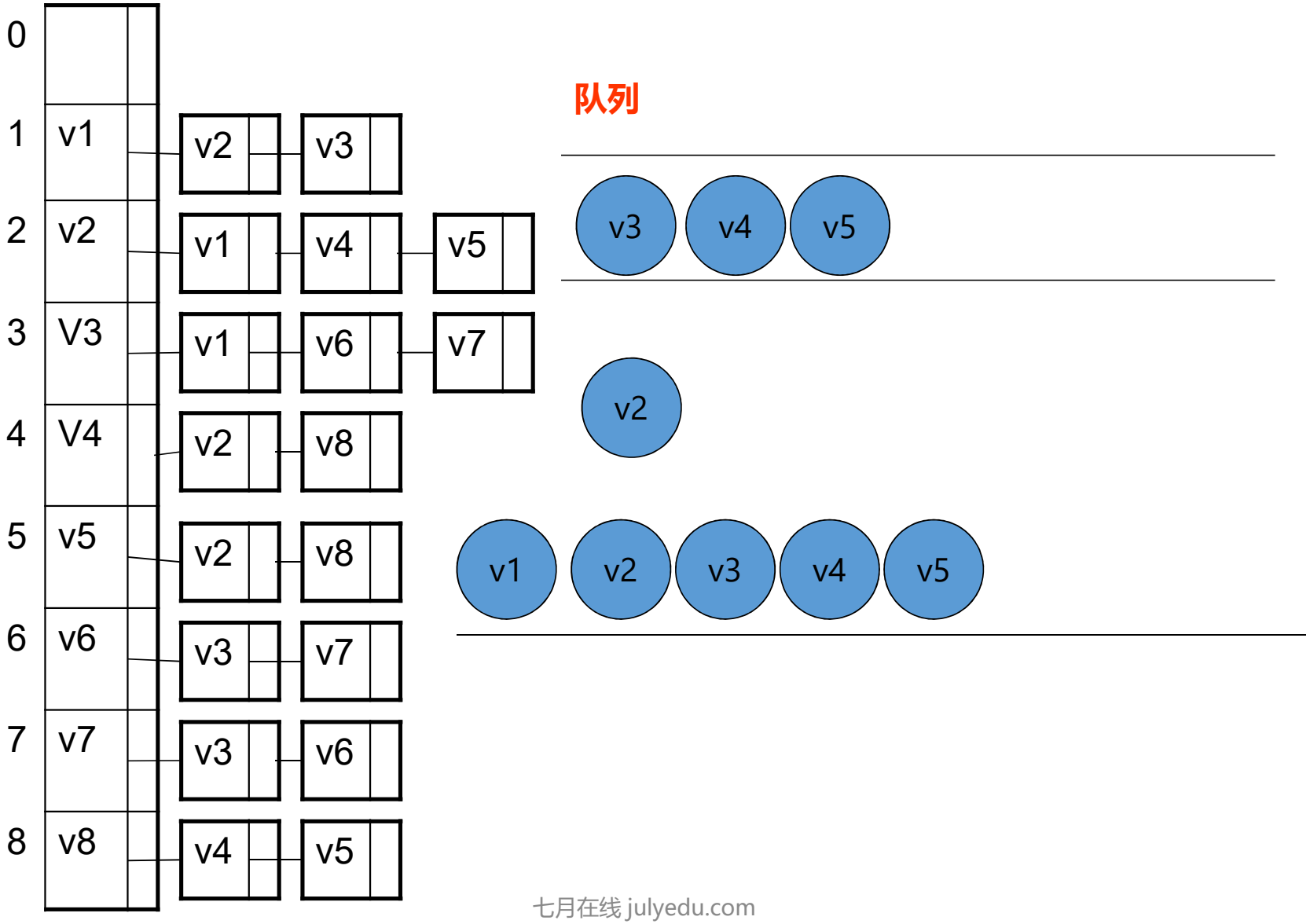
V2的邻接点v4没有被访问过，访问之，且入队列

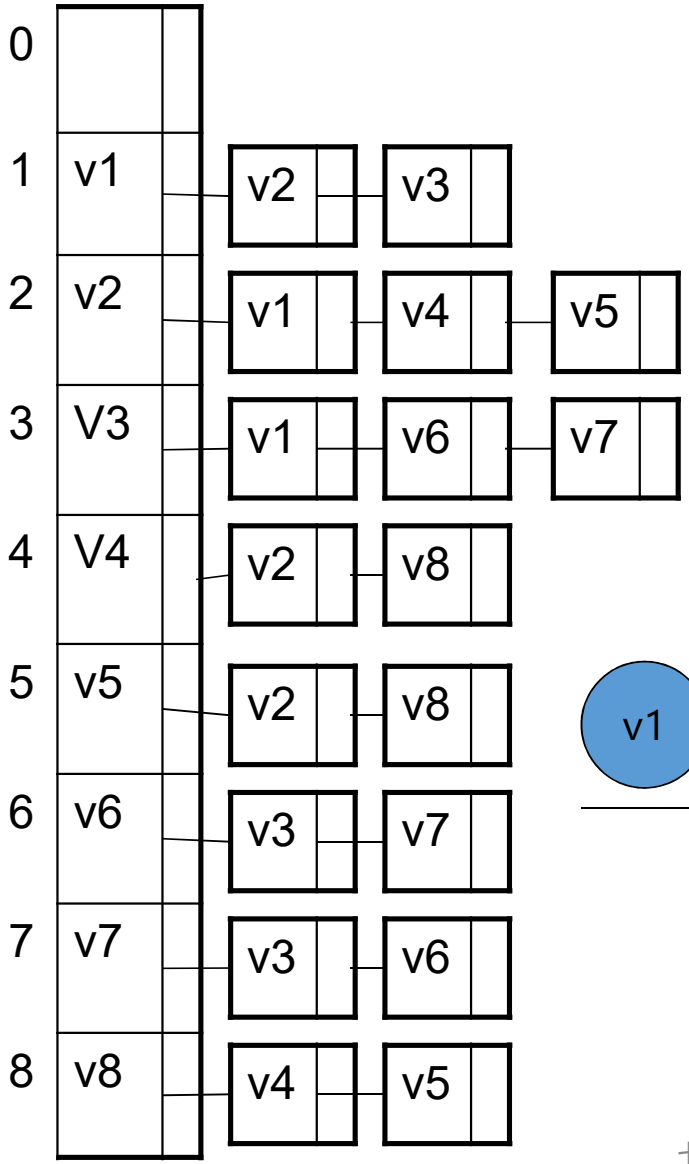


队列

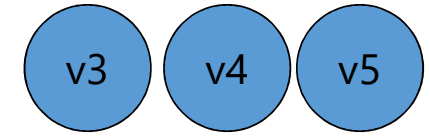


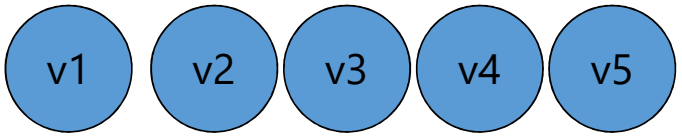

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v3
v4
v2
v1
v2
v3
v4
v5
V2的邻接点v5没有被访问过，访问之，且入队列

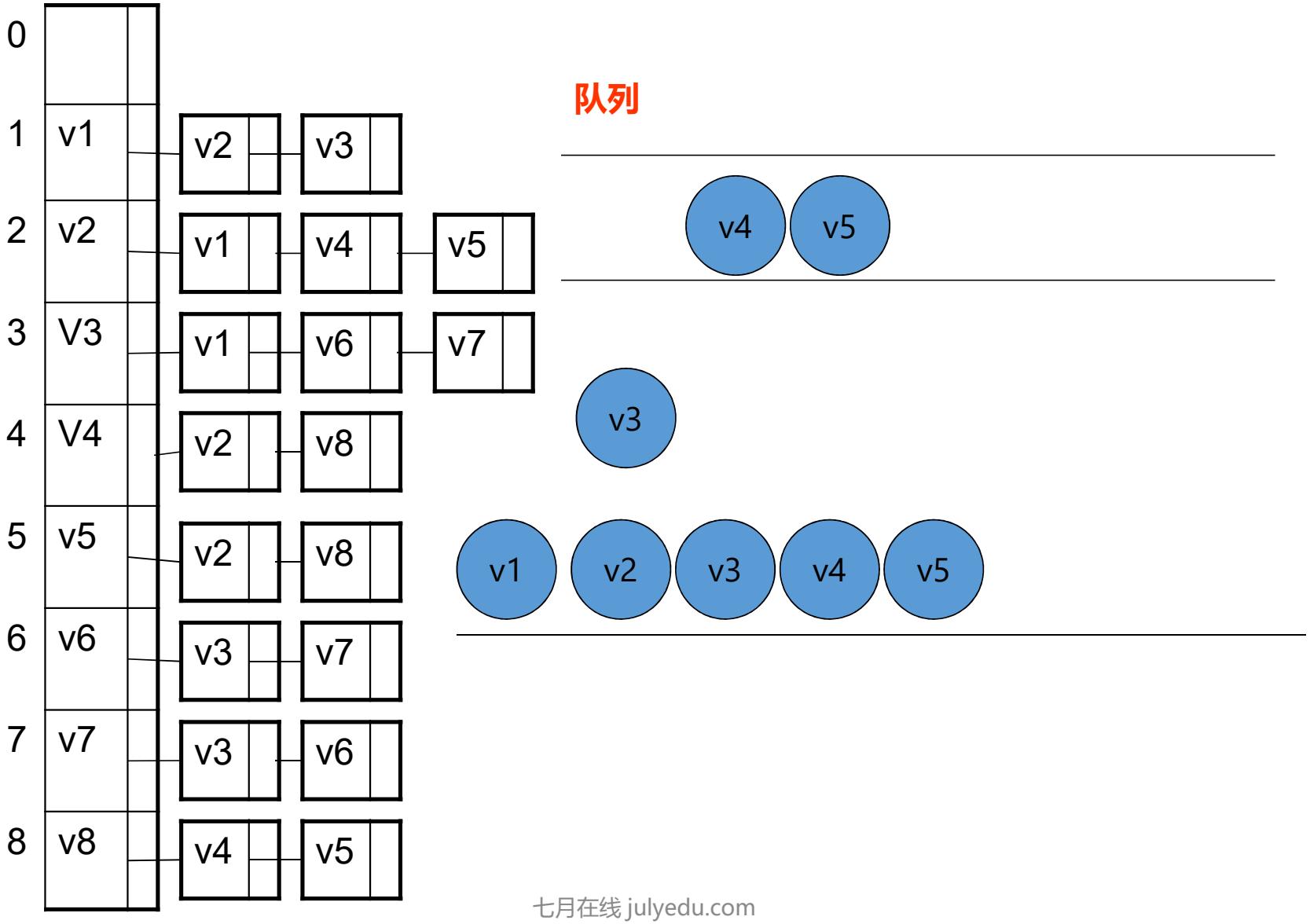


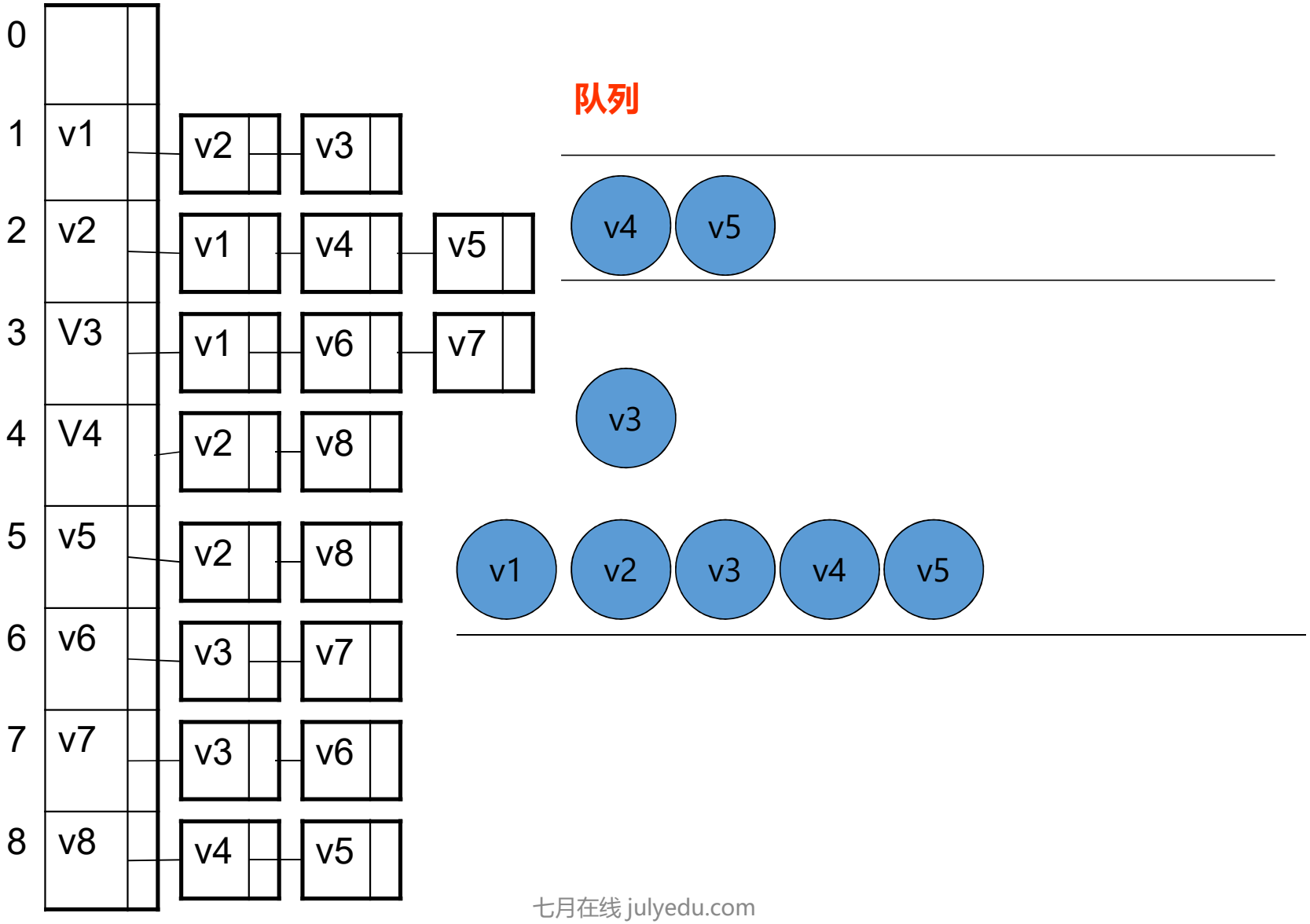


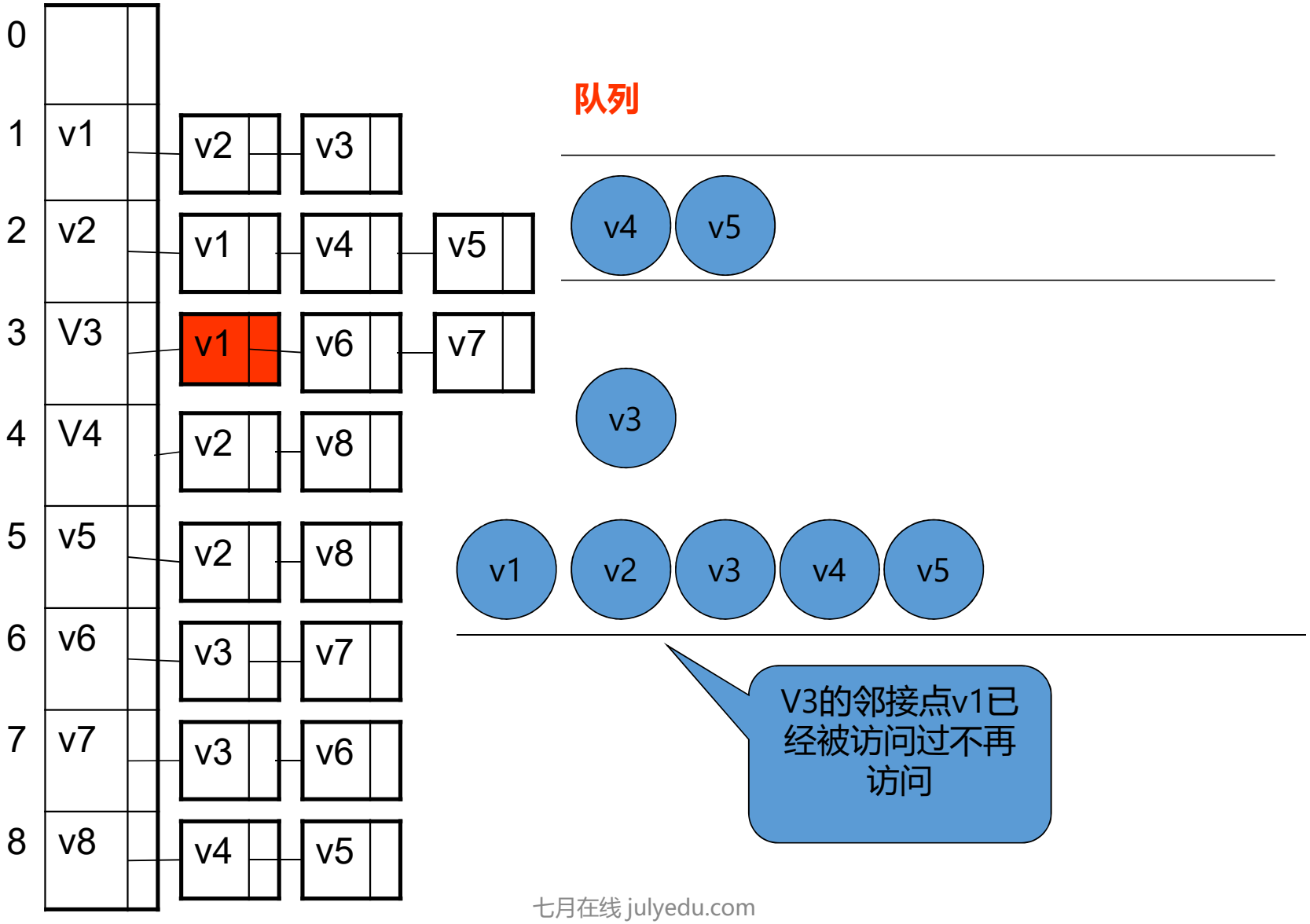
队列

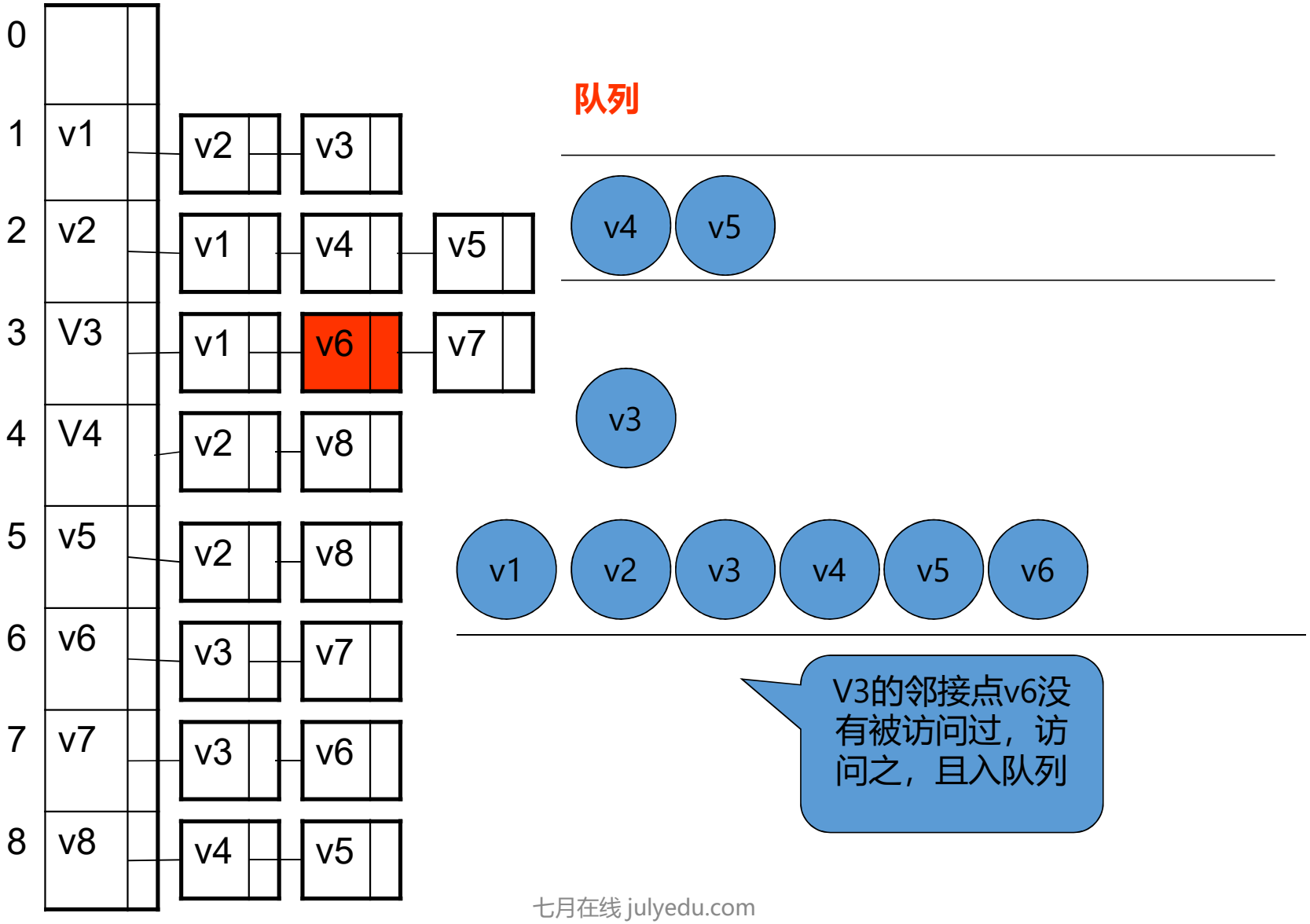


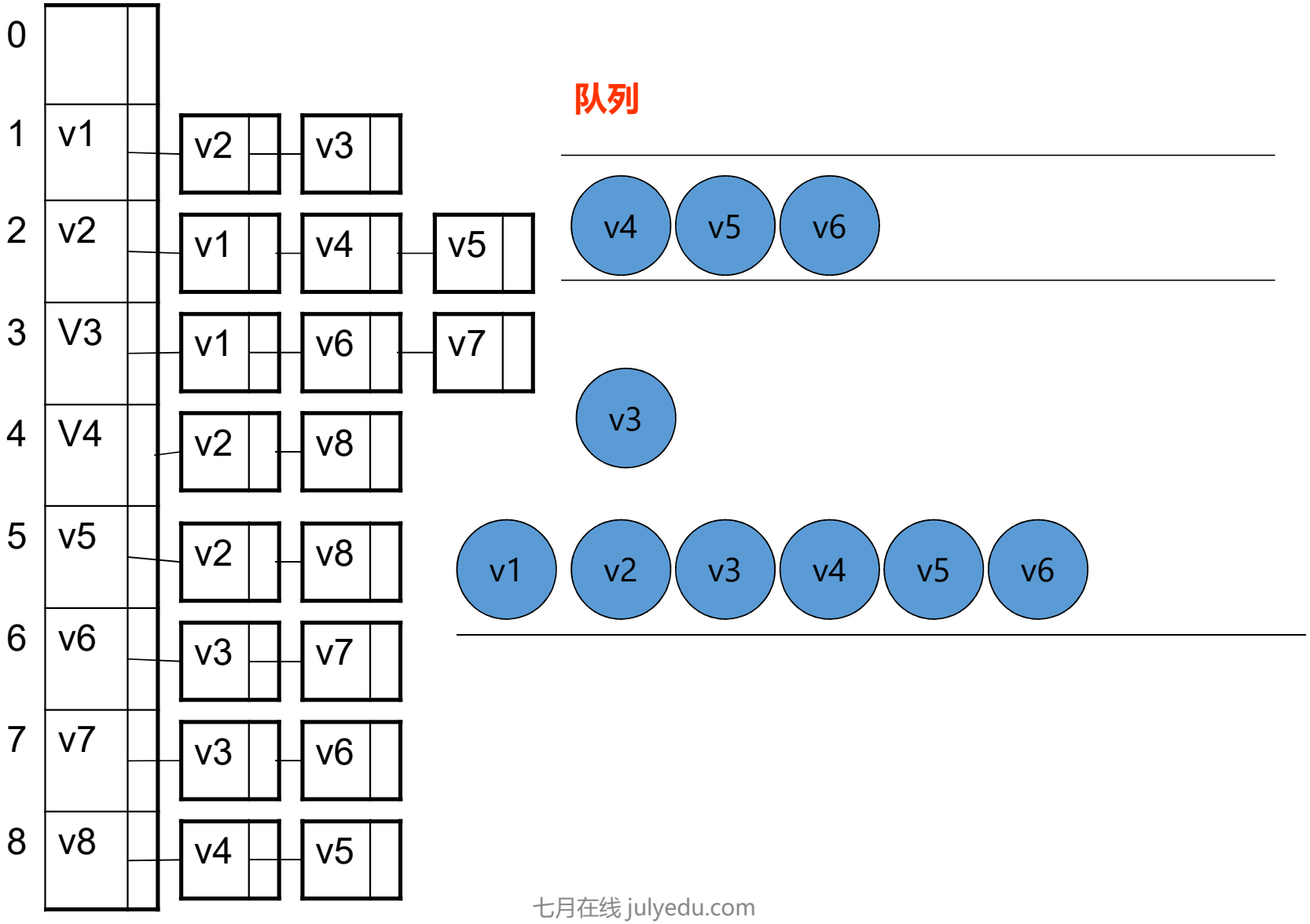


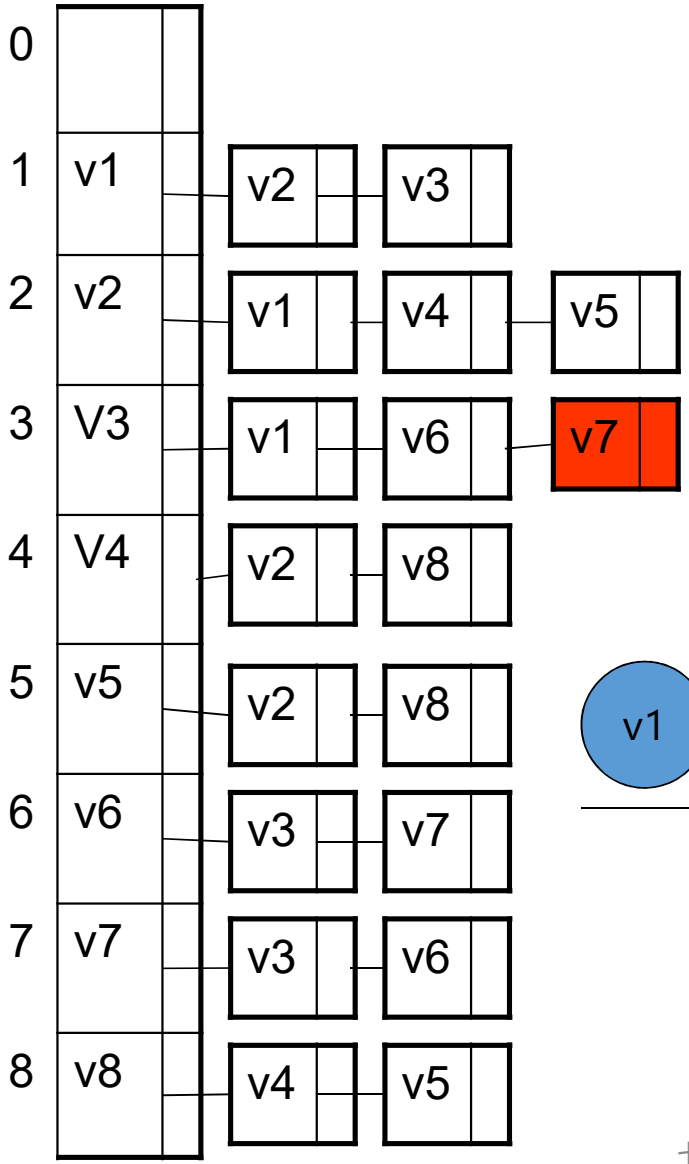




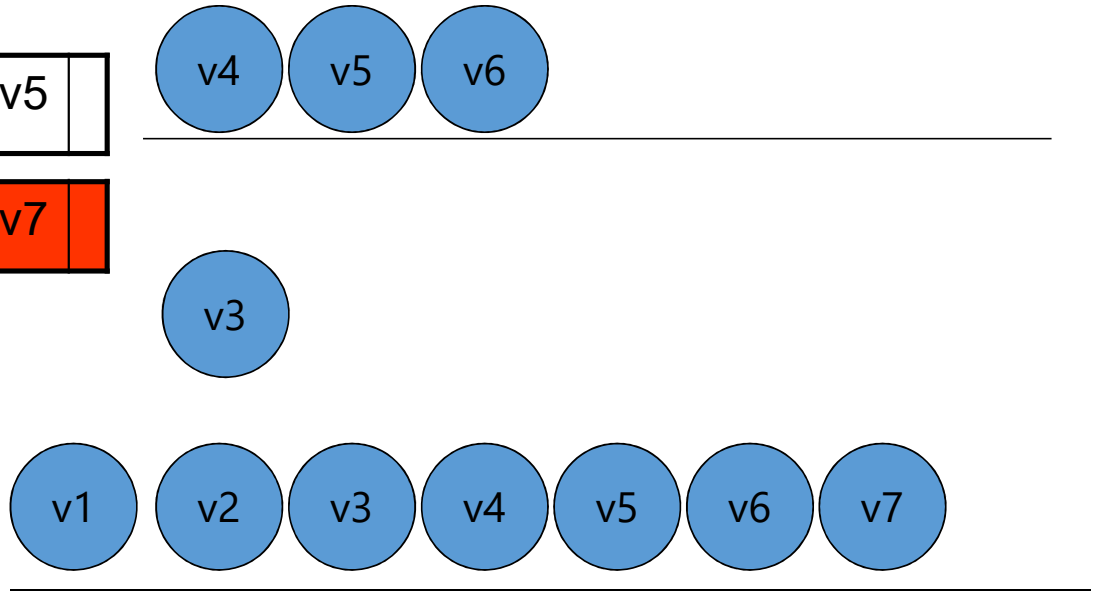

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v4
v5
v3
v1
v2
v3
v4
v5
V3的邻接点v1已经被访问过不再访问


0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v4
v5
v3
v1
v2
v3
v4
v5
v6
V3的邻接点v6没有被访问过，访问之，且入队列

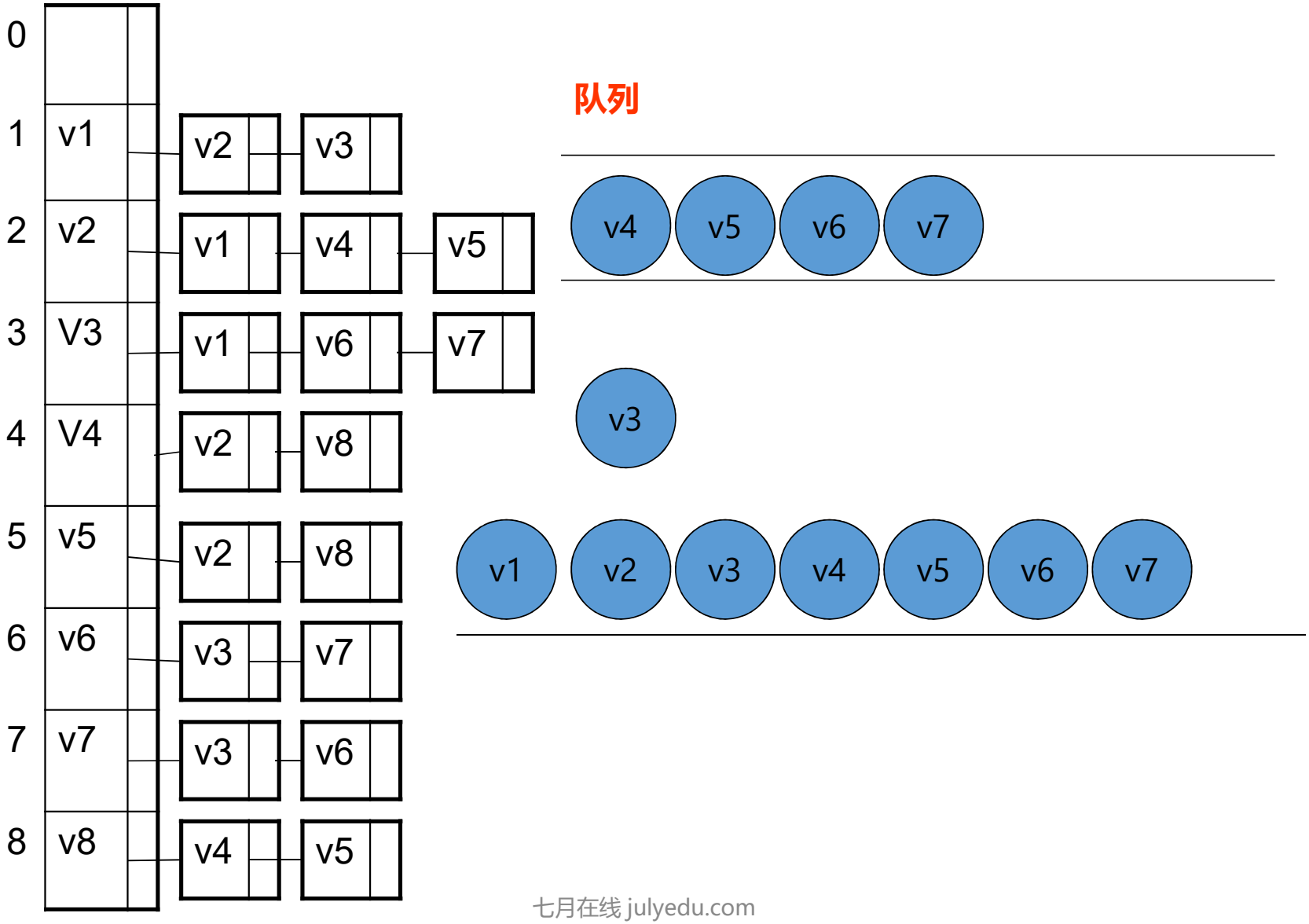


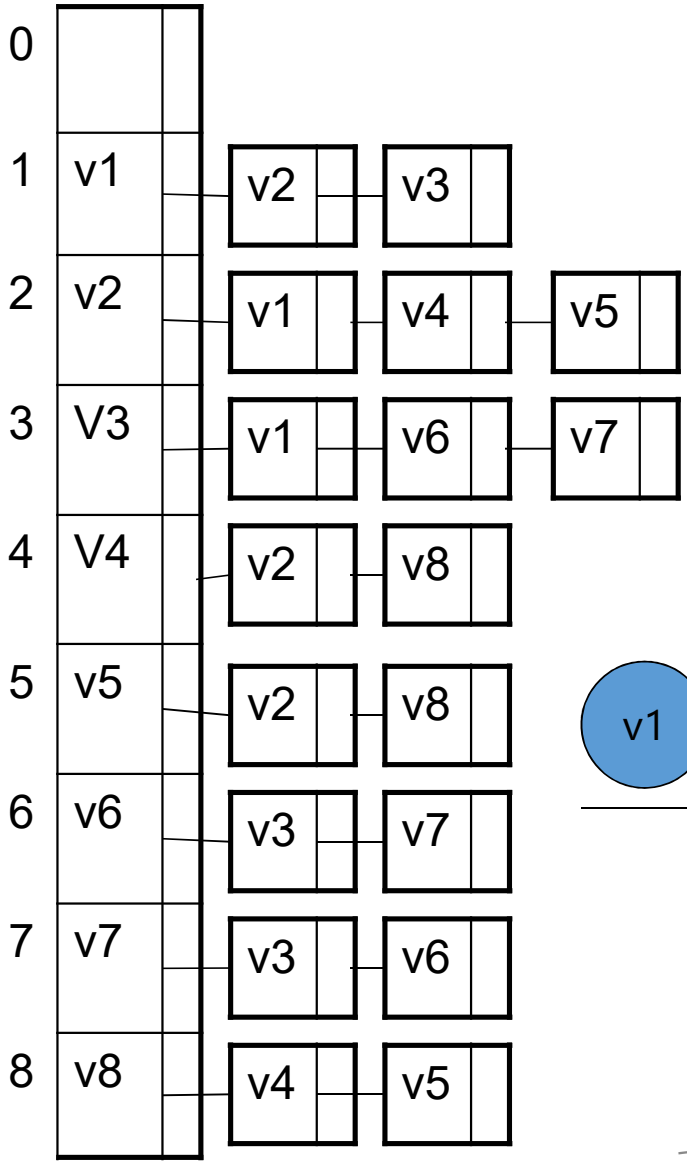


队列

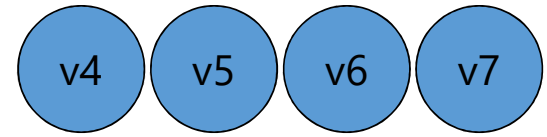


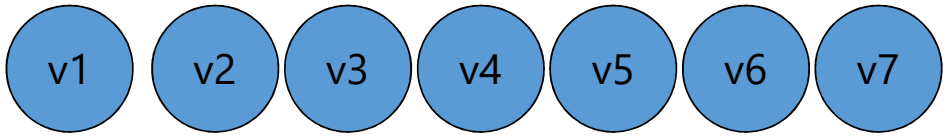
V3的邻接点v7没有被访问过，访问之，且入队列

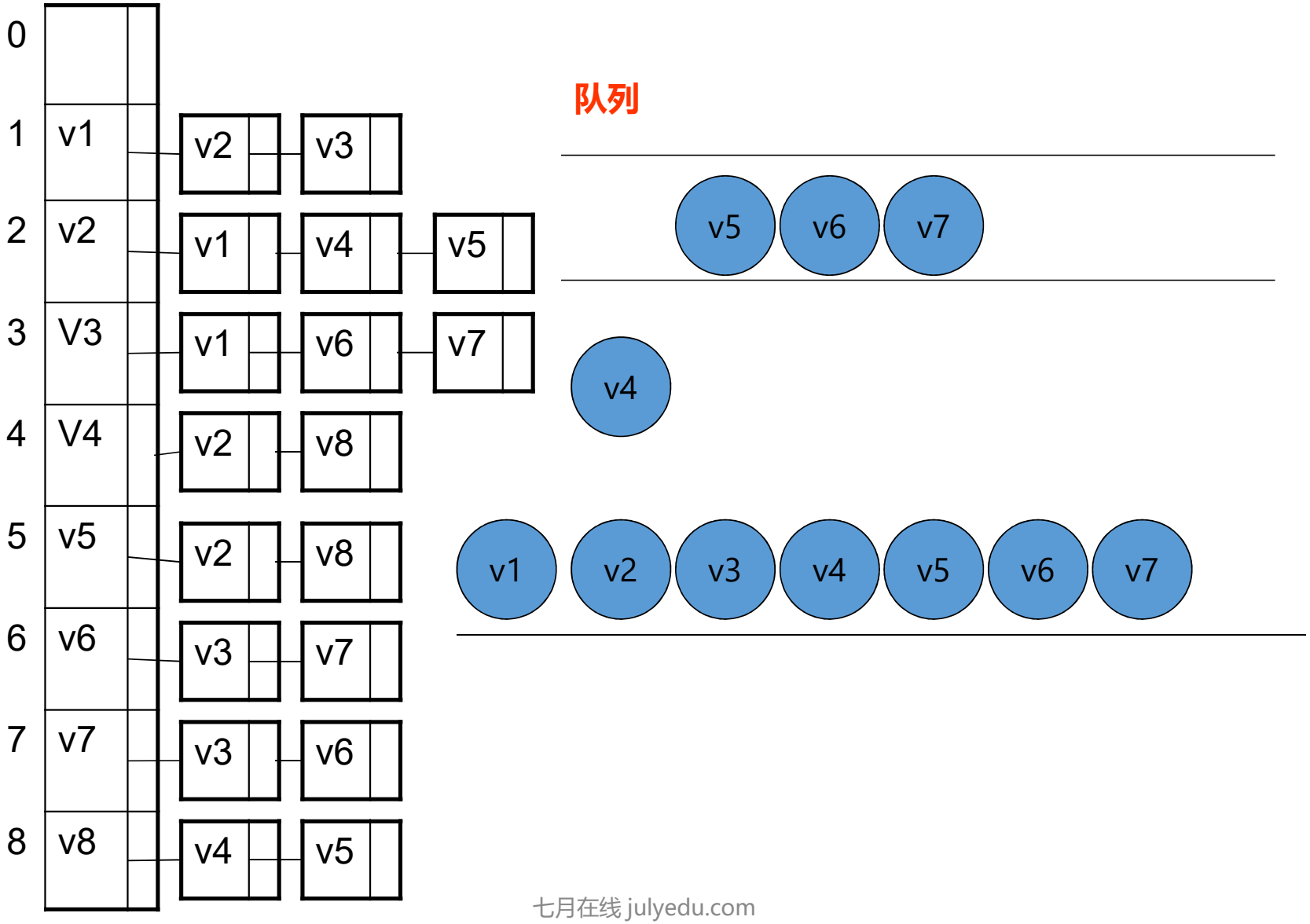

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v4
v5
v6
v7
v3
v1
v2
v3
v4
v5
v6
v7

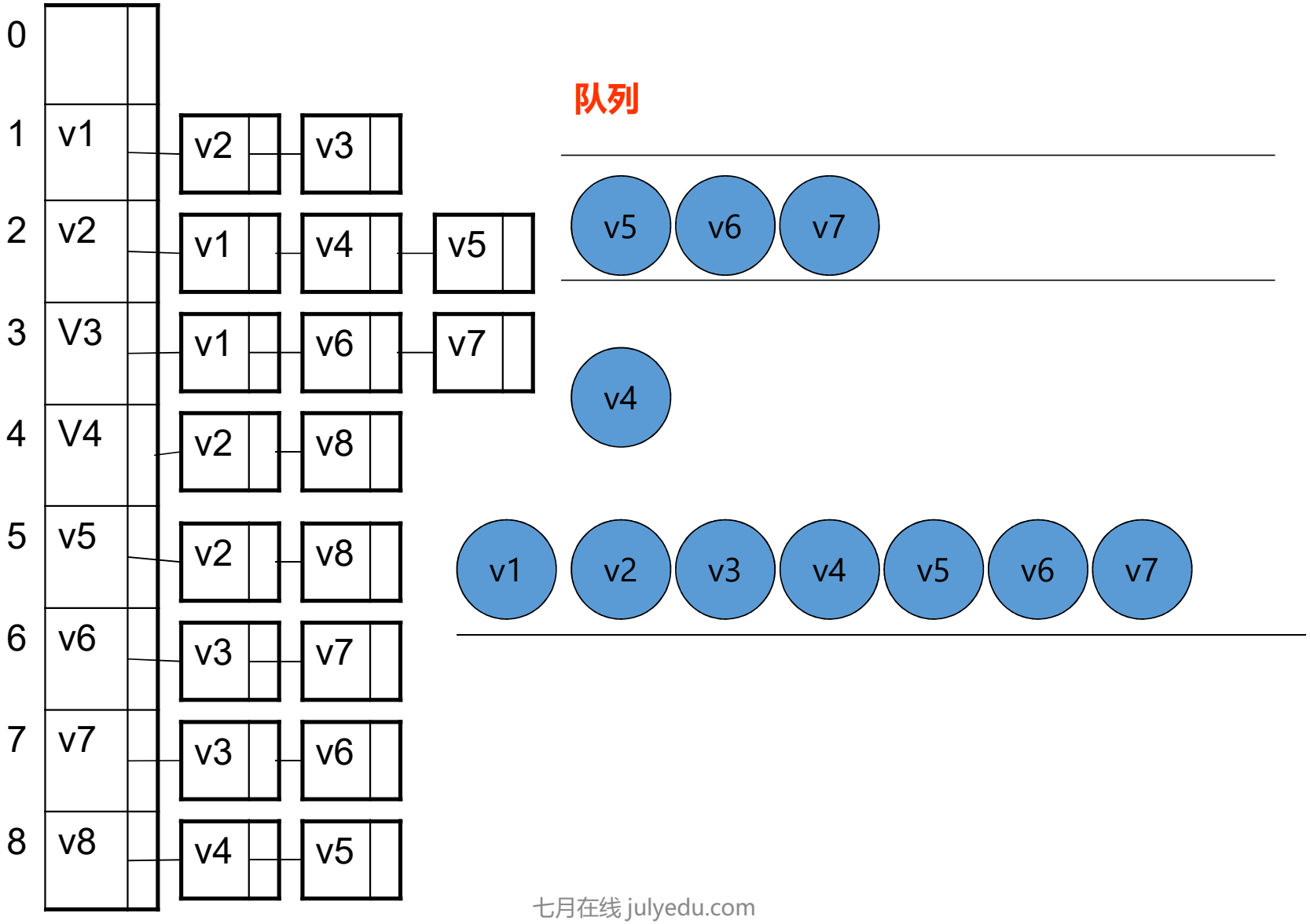


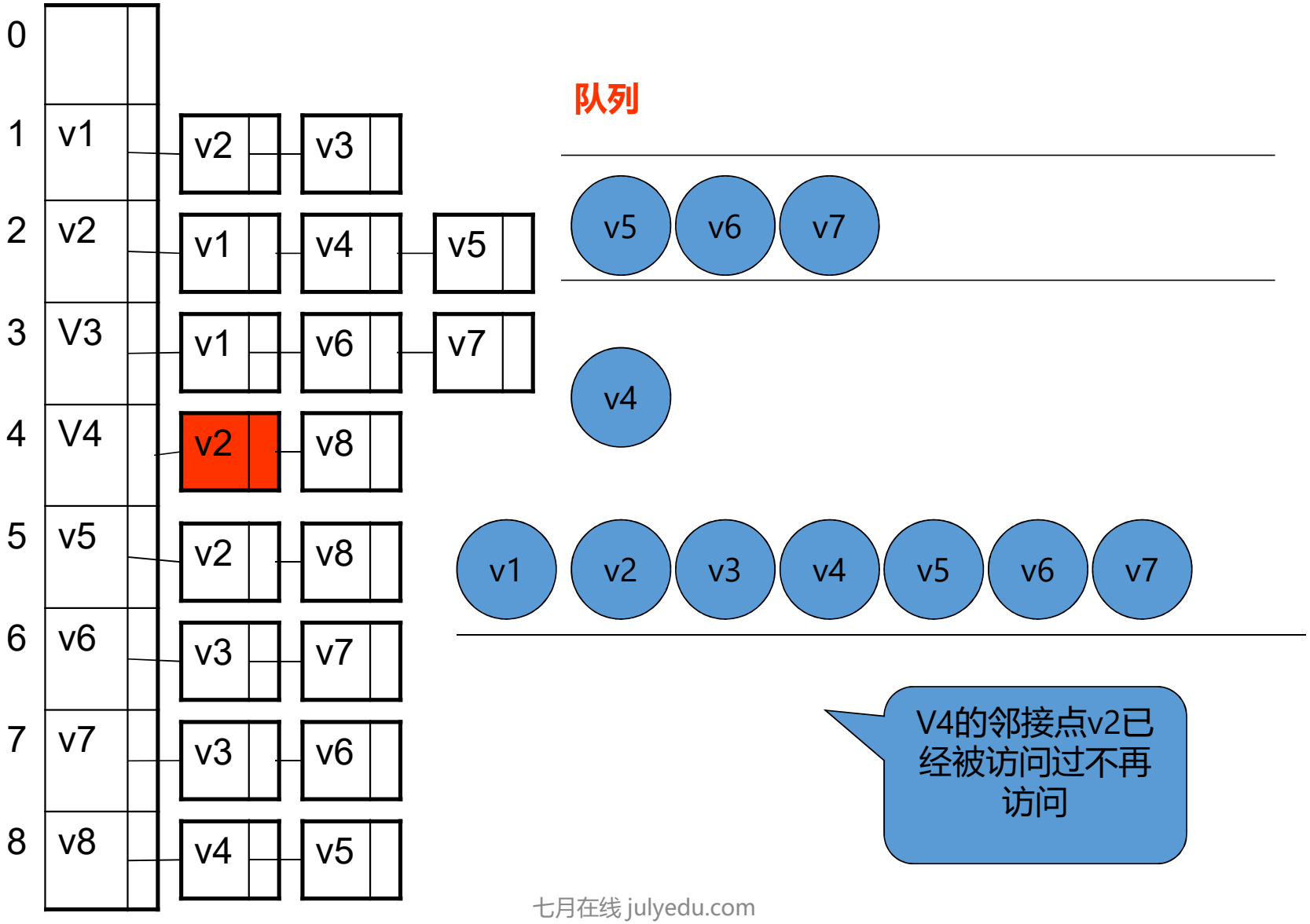
队列

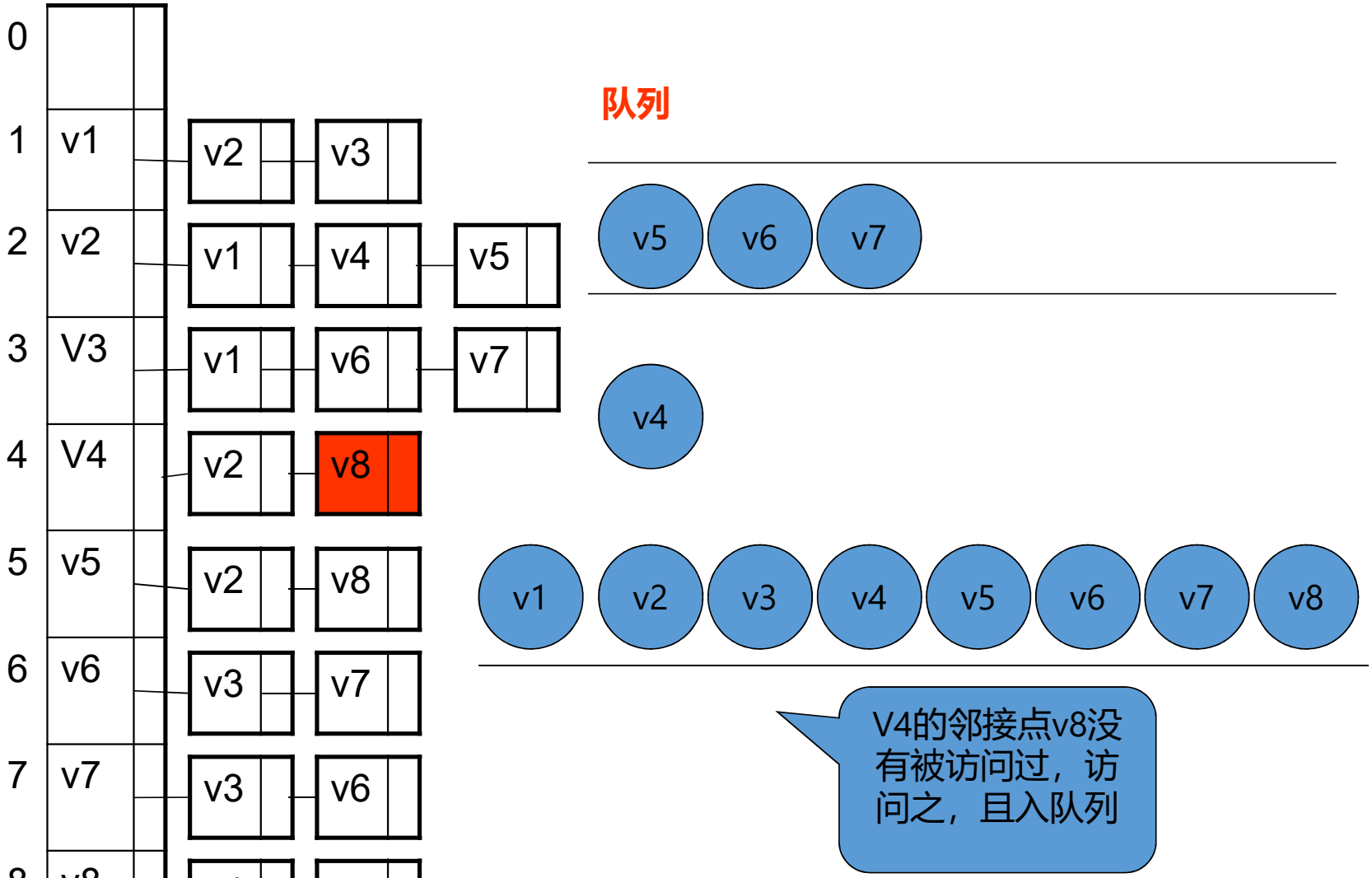


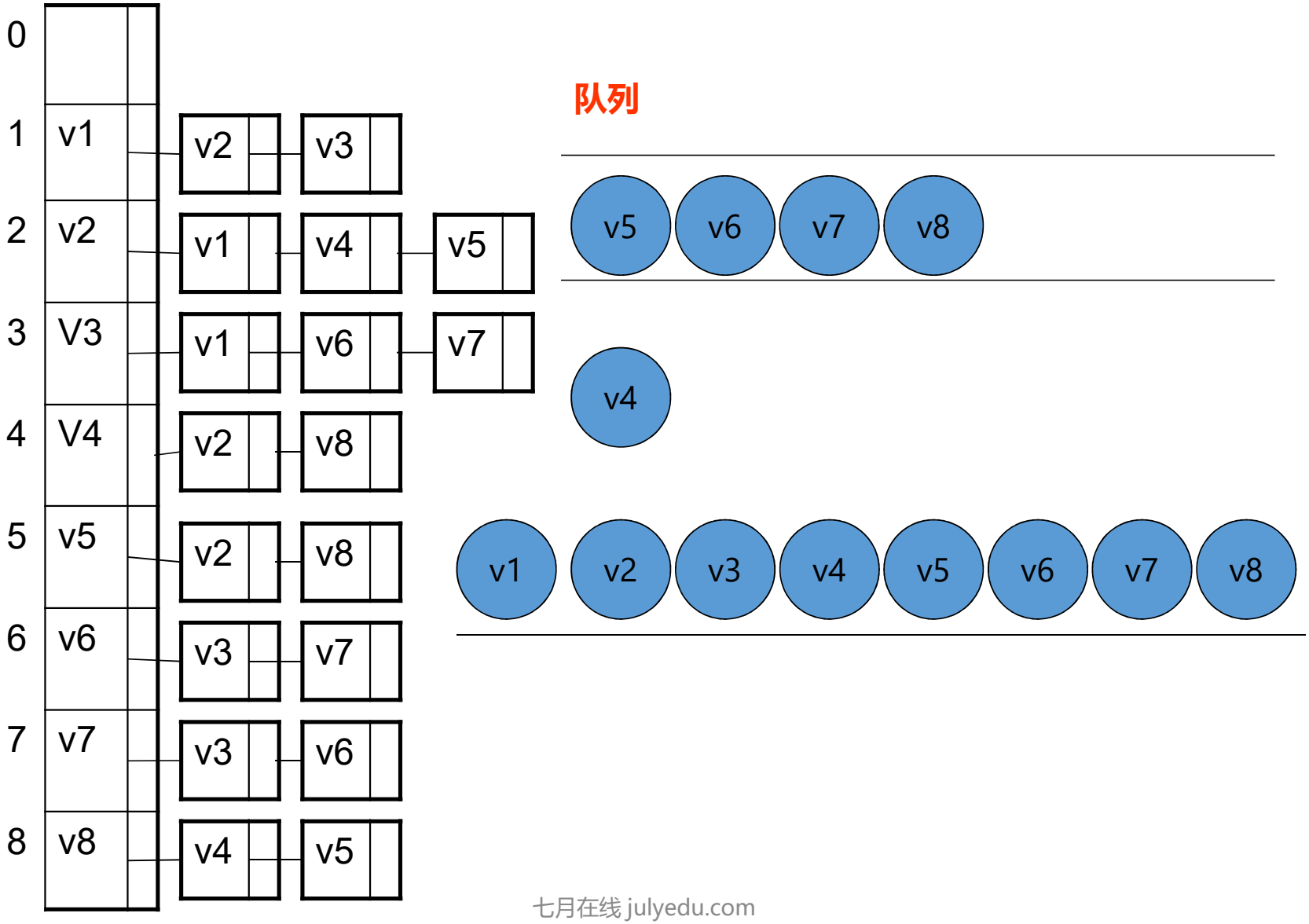


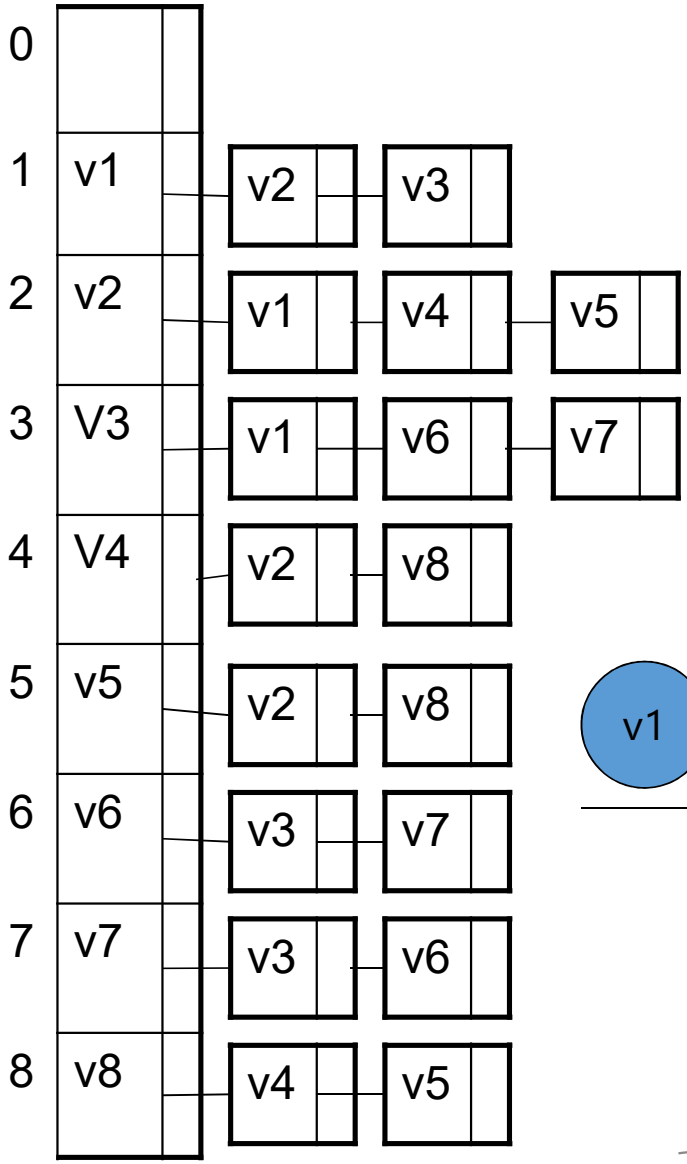

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v5
v6
v7
v4
v1
v2
v3
v4
v5
v6
v7


0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v5
v6
v7
v4
v1
v2
v3
v4
v5
v6
v7

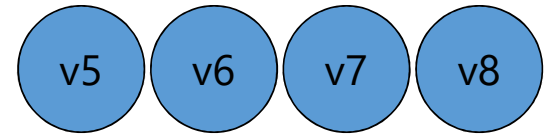

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v5
v6
v7
v4
v1
v2
v3
v4
v5
v6
v7
V4的邻接点v2已经被访问过不再访问

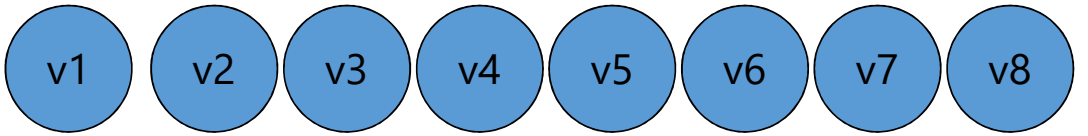

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v5
v6
v7
v4
v1
v2
v3
v4
v5
v6
v7
v8
V4的邻接点v8没有被访问过，访问之，且入队列

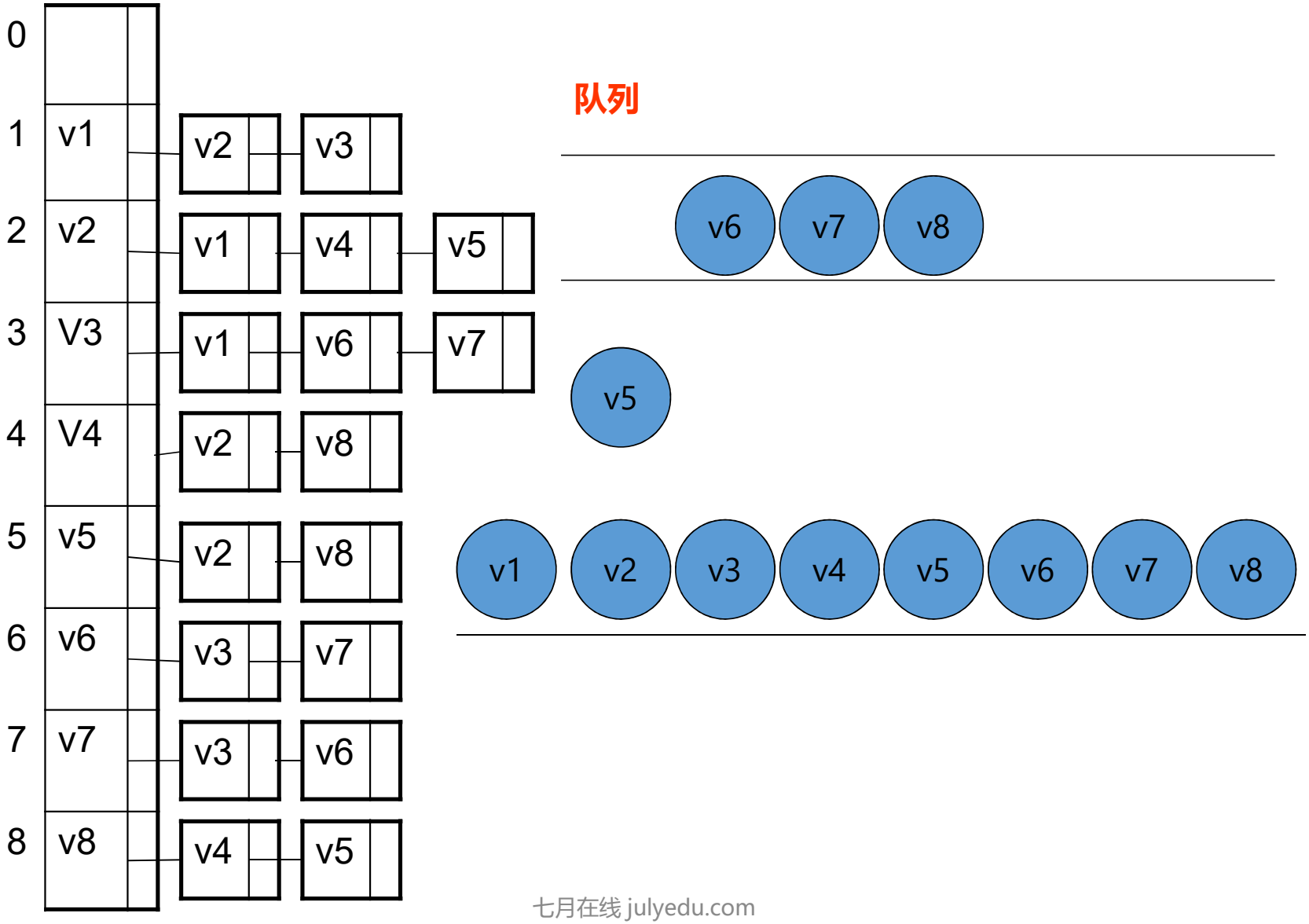

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v5
v6
v7
v8
v4
v1
v2
v3
v4
v5
v6
v7
v8

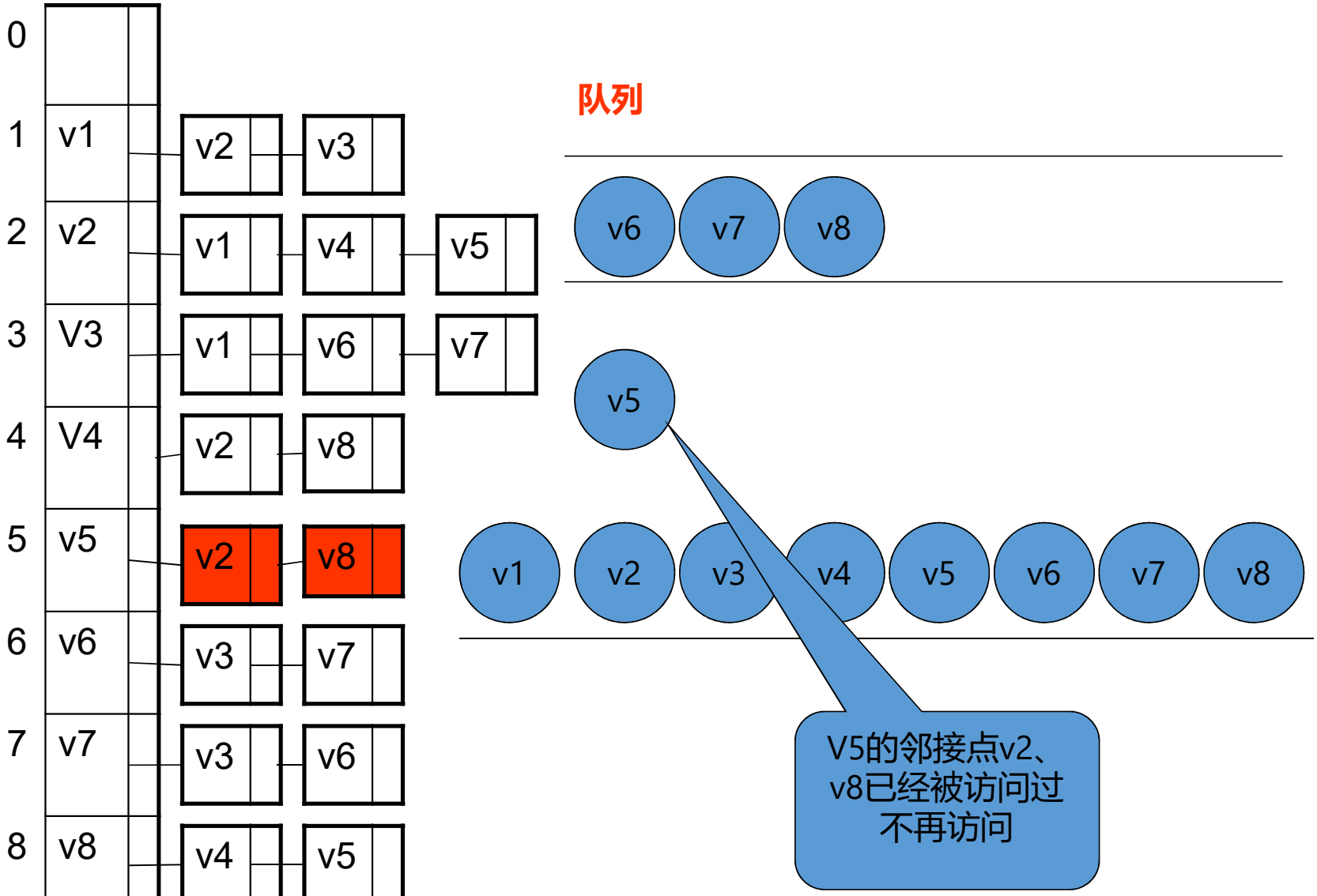


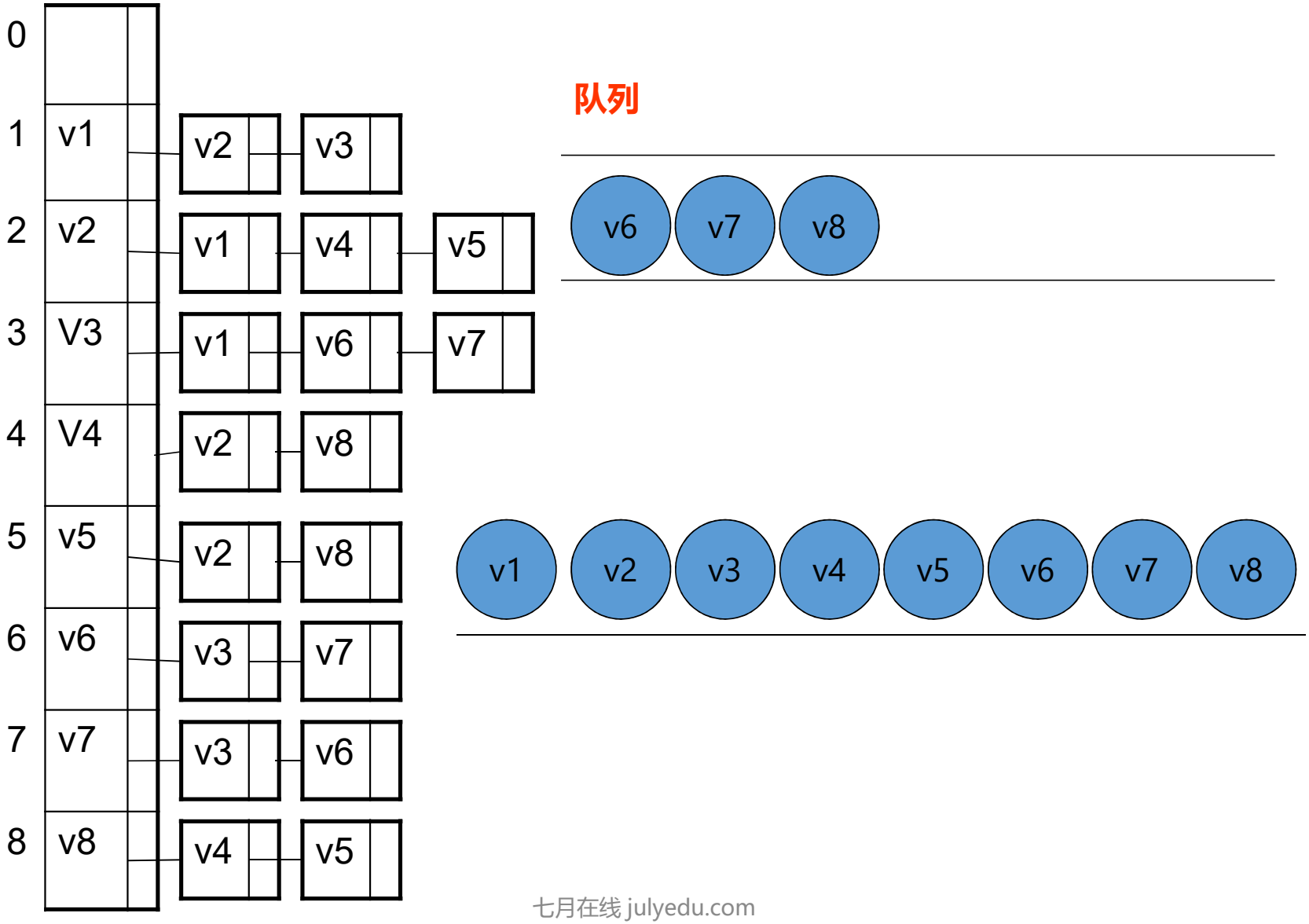
队列

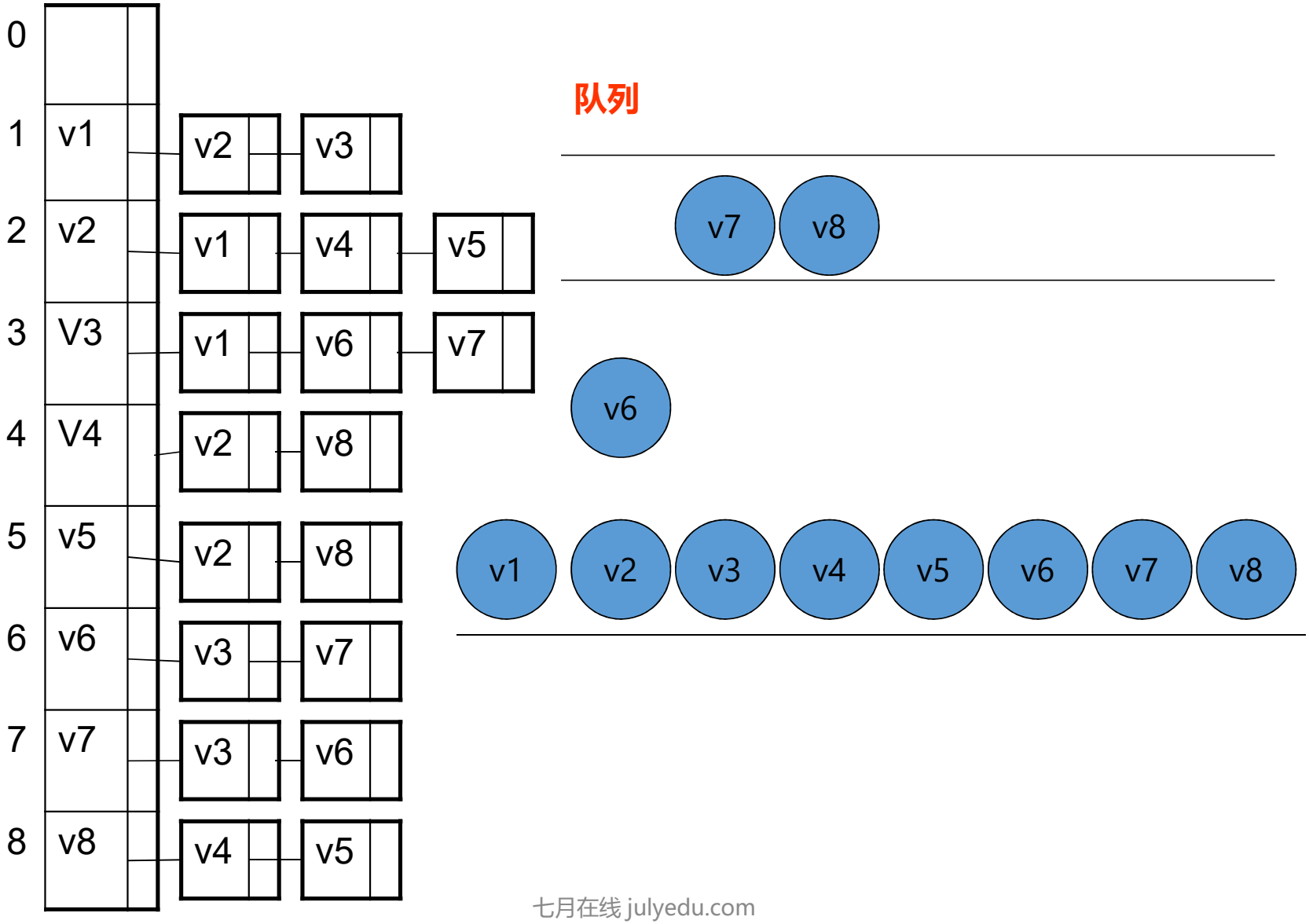


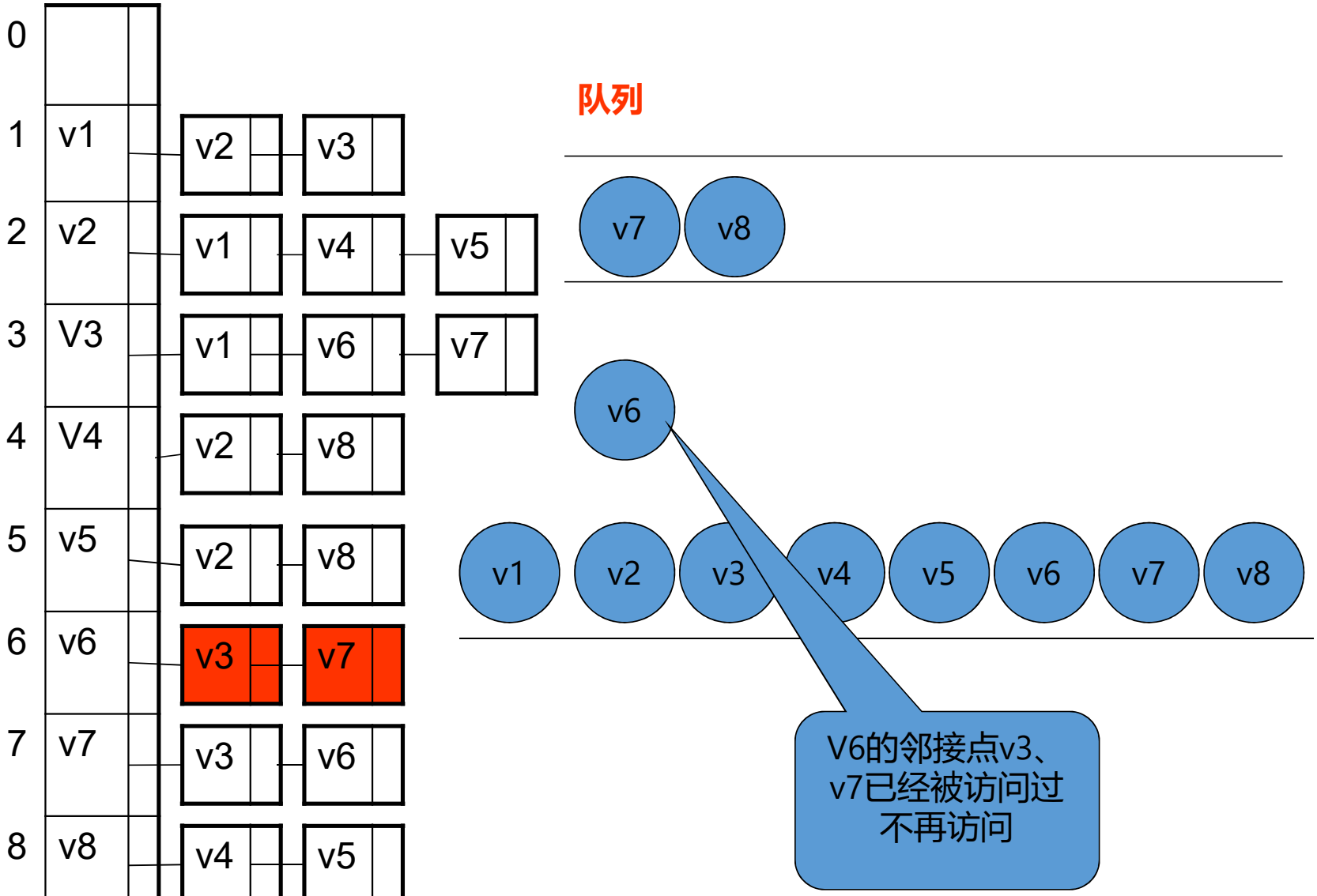


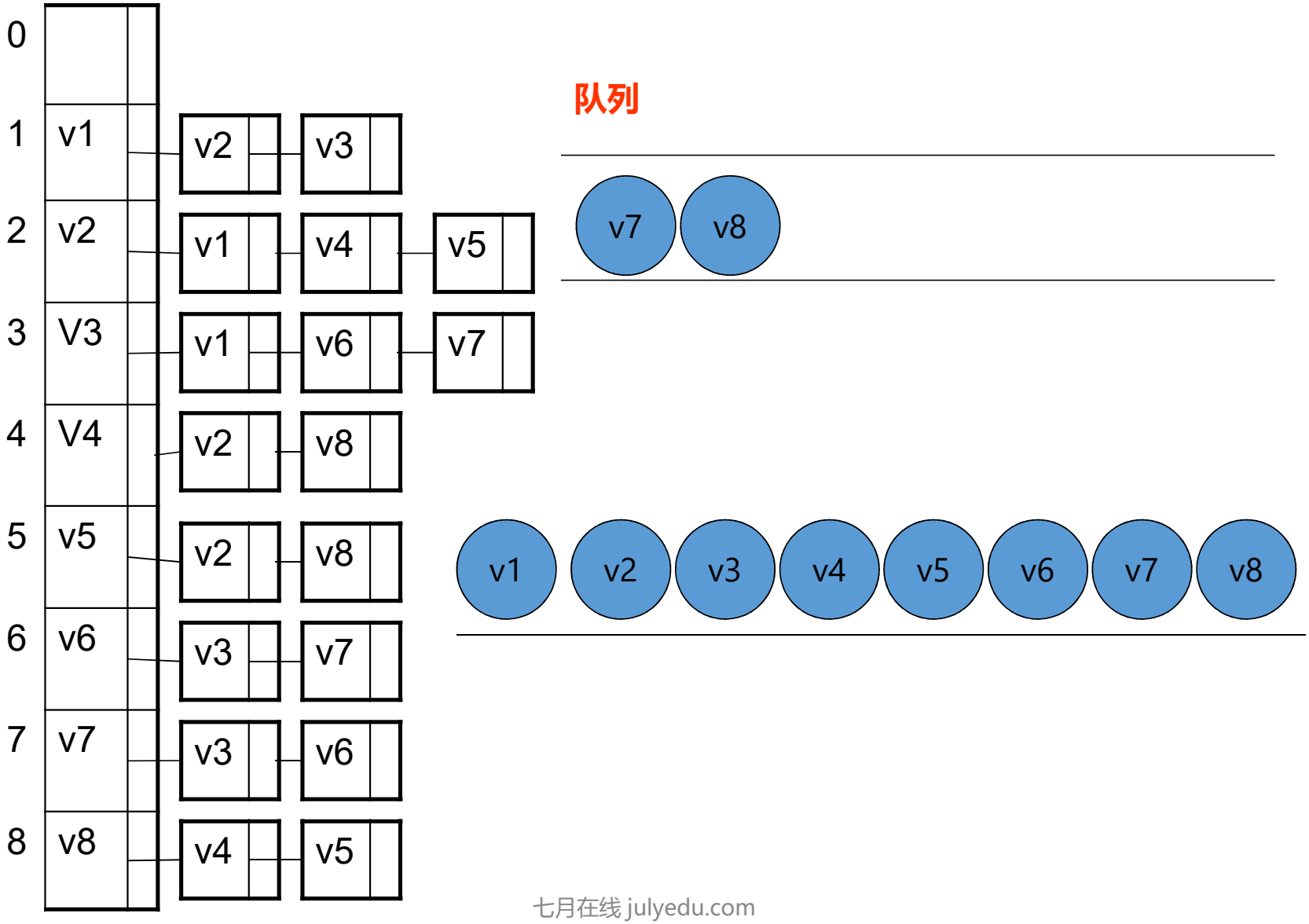


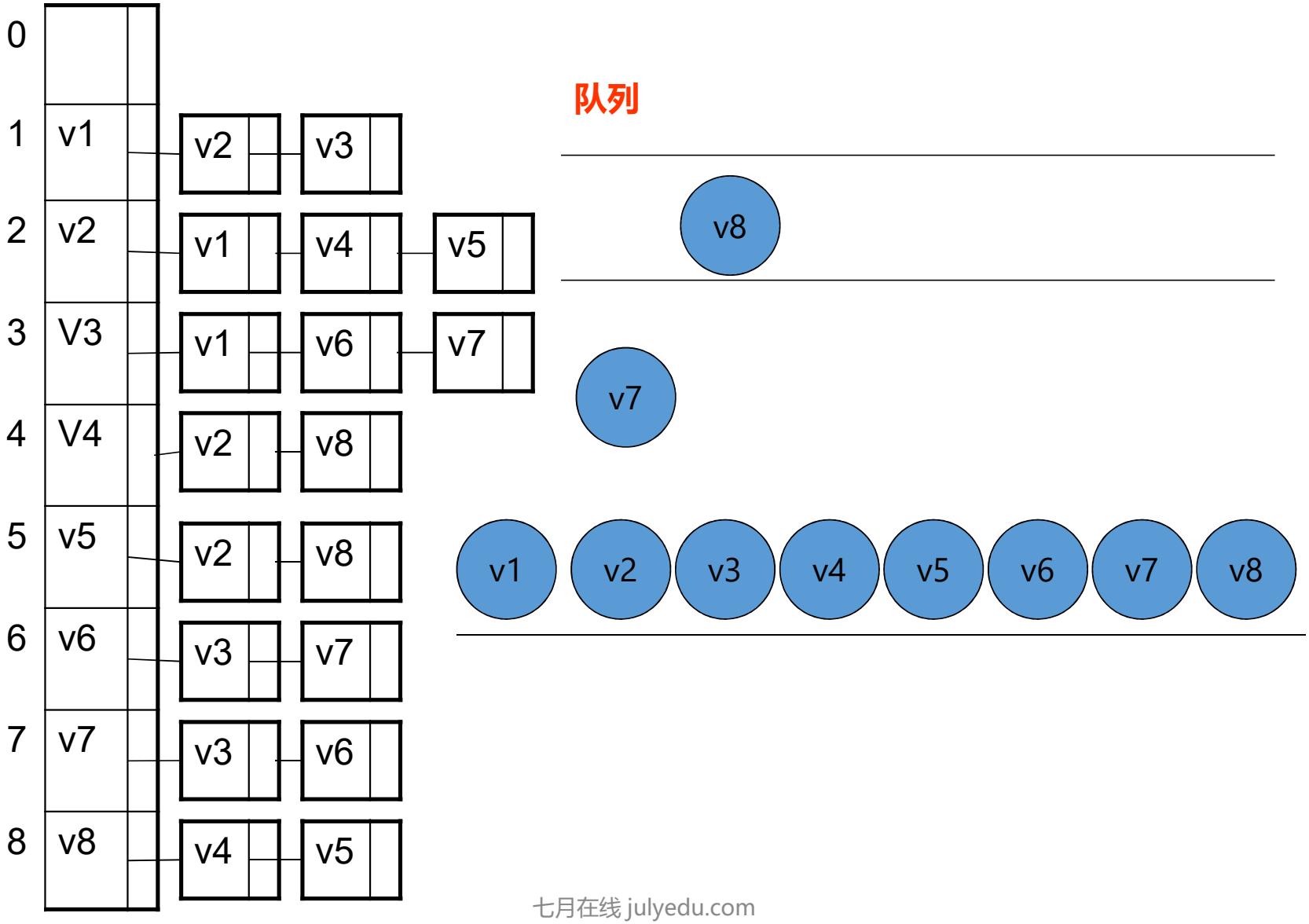


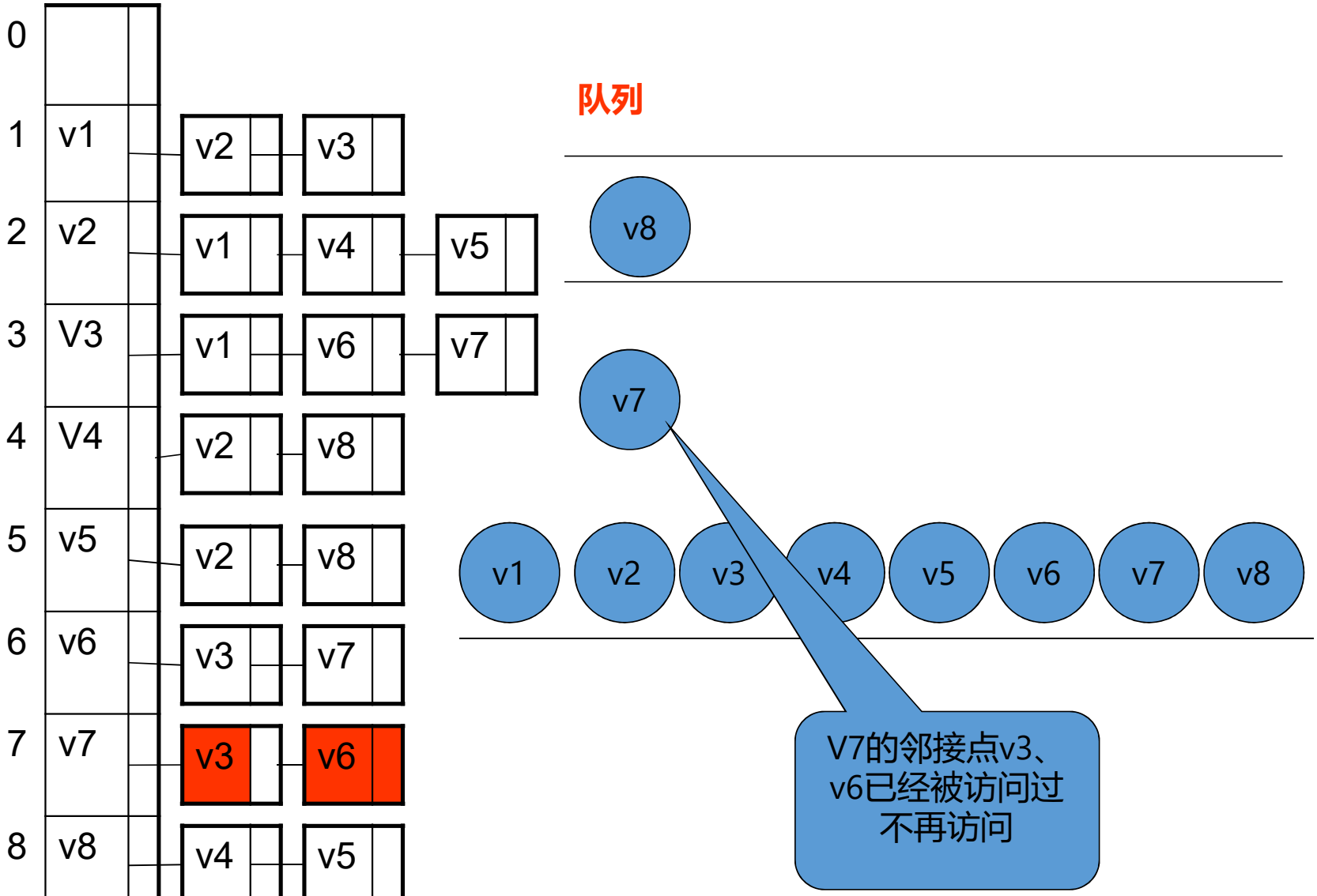

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v6
v7
v8
v1
v2
v3
v4
v5
v6
v7
v8

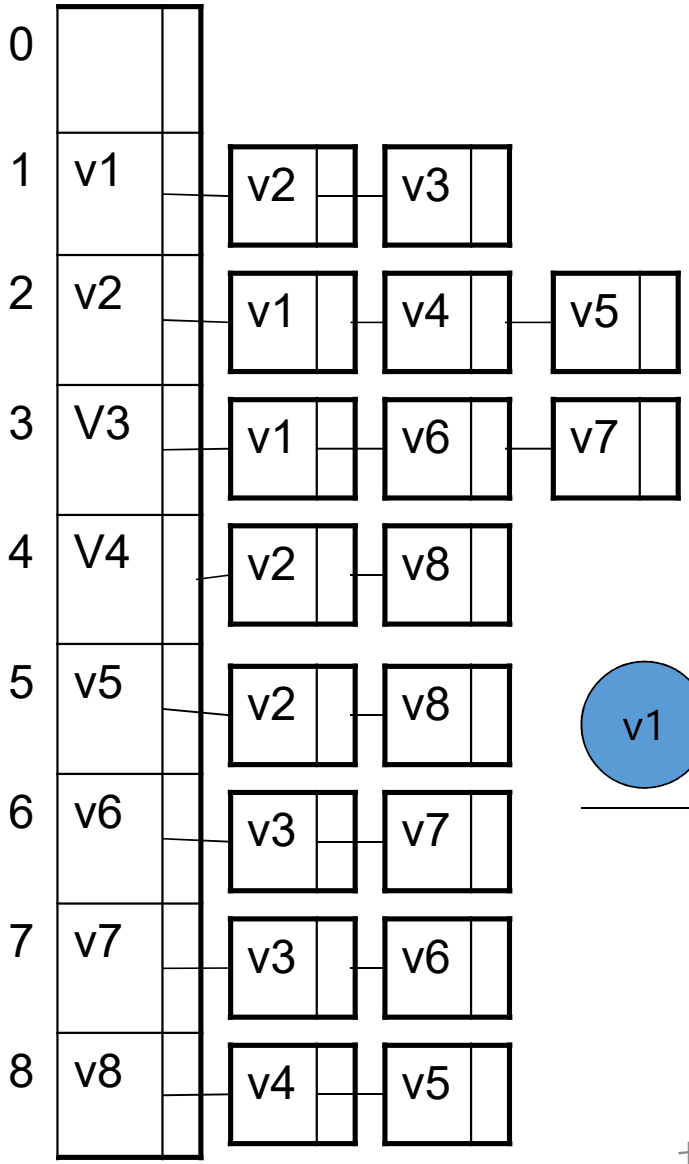

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v7
v8
v6
v1
v2
v3
v4
v5
v6
v7
v8



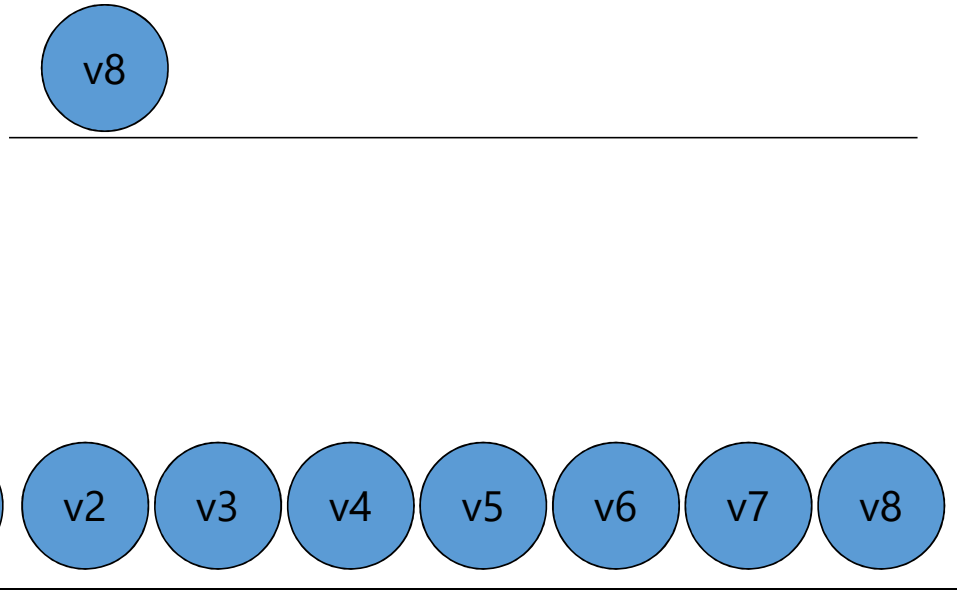

0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5
队列
v7
v8
v1
v2
v3
v4
v5
v6
v7
v8

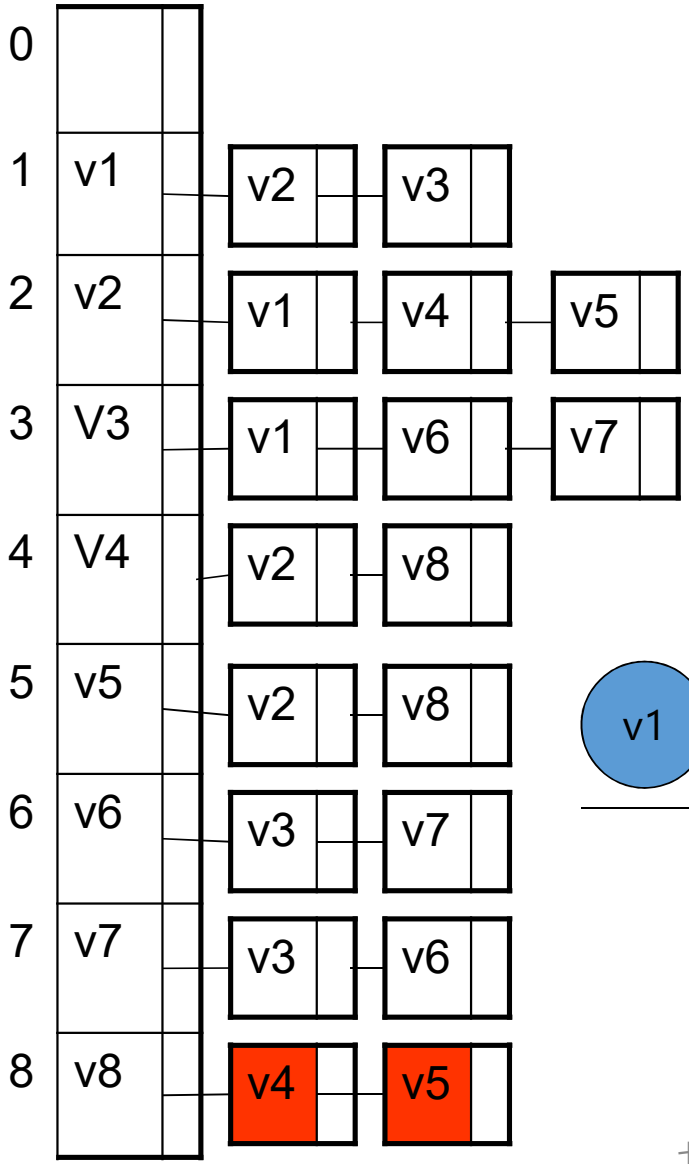




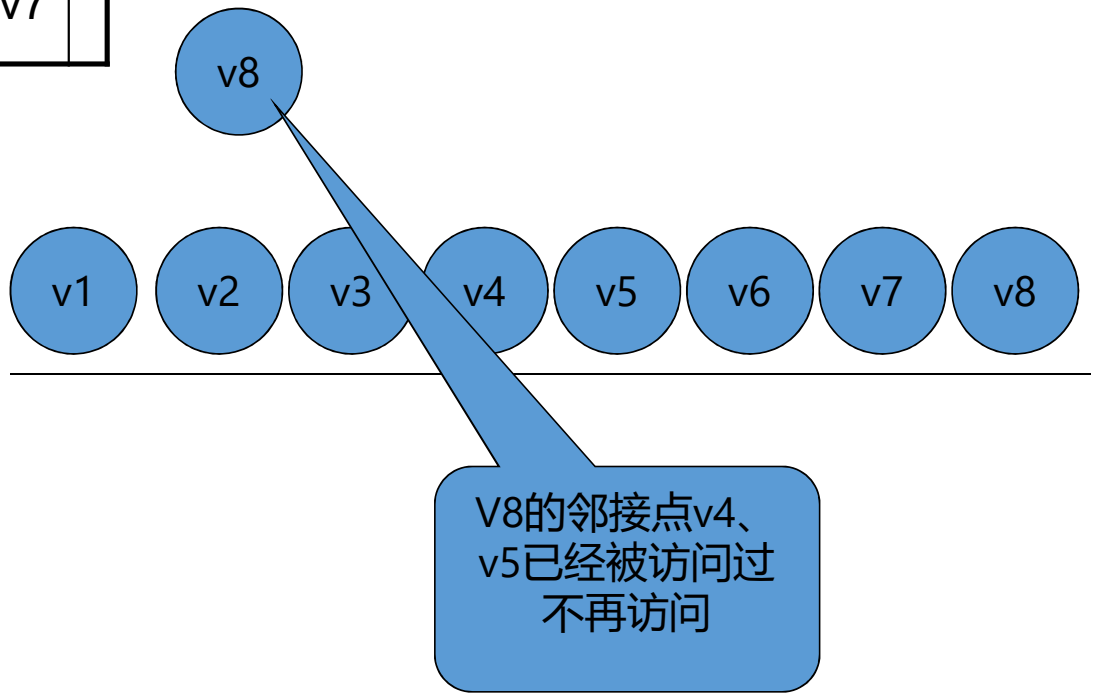


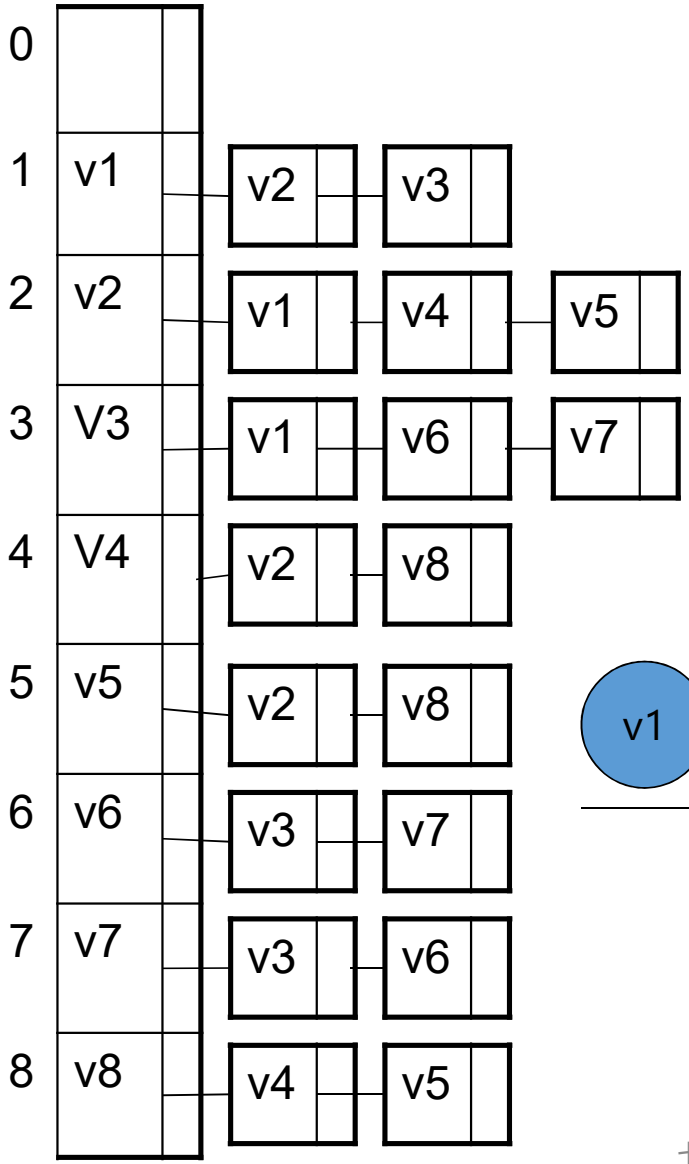
队列





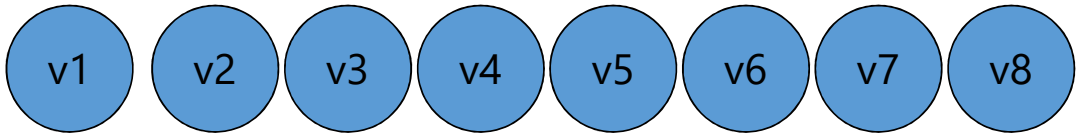
队列




0
1
v1
v2
v3
2
v2
v1
v4
v5
3
V3
v1
v6
v7
4
V4
v2
v8
5
v5
v2
v8
6
v6
v3
v7
7
v7
v3
v6
8
v8
v4
v5

队列

队列为空，算法结束


v1
v2
v3
v4
v5
v6
v7
v8

深度优先：伪代码

```
void DFS(int v)
        visited[v] = true
        for (v的每一个邻接点w)
                if (!visited[w])          //如果没有被访问过
                        DFS(w)
```

N皇后问题

- 在N*N的棋盘上摆放N个皇后，使得任意两个皇后都不能处于同一行、同一列或同一斜线上
- 回溯法（暴力搜索万金油）
- 本质：深度优先（隐式图搜索）

N皇后问题

- 如何定义状态，关系?
- 时间复杂度 $O(N^N)$ （状态空间）
- 剪枝（确定性）
- 作业题：你写的代码能跑多少个皇后?
- Leetcode 51

骑士游历问题

- 国际象棋棋盘上，有一个骑士（马）从左下角出发，是否能不重复的遍历每一个格子
- 没啥好的办法，考虑暴力
- 如何定义状态，关系？
- 除了剪枝，还有什么办法？（求任意解和所有解）
- 启发式(A*)
  - 改变搜索顺序
  - 不确定性

广度优先：伪代码

```
void BFS(int x)
        visited[x] = true
        Q.push(x)
        While (!Q.empty())
                v = Q.pop()
                for (v的每个邻接点w)
                        if (!visited[w])
                                visited[w] = true
                                Q.push(w)
```

种子填充法

- Flood Fill 洪水填充法
- 目标：标记某块封闭的区域，并找出其边界
- 如何定义状态，关系？
- BFS犹如墨汁滴入清水

种子填充法

- Leetcode 200. Number of Islands
- Leetcode 130. Surrounded Regions

八数码

- 3*3的方格内有编号1-8的方块，求最少的步数，恢复这些方块的顺序
- 深度优先 or 广度优先?
- 判重（Hash）

八数码

- 双向搜索
  - 起始点和目标点，轮流扩展
  - Hash表判断相遇
  - 复杂度
- 启发式
  - 价值函数（启发函数）
  - 优先队列（堆）

迭代加深

- 求最优 -> 求判定
- 不需要判重
- 搜到即是最优
- 前一个阶段相对下一个阶段仅仅是常数
- 迭代加深 + 启发式（IDA*）

总结

- DFS vs BFS
  - 都为暴力搜索，但搜索顺序不同
  - 栈 vs 队列
  - 可行解 vs 最优解
  - 递归 vs 非递归
  - 空间占用，BFS需要存储状态，DFS无需
- 事无绝对，仅供参考