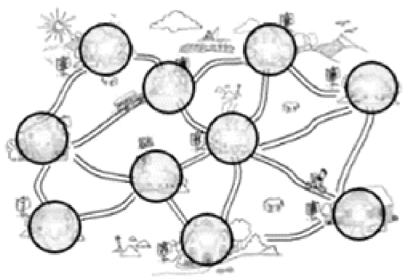
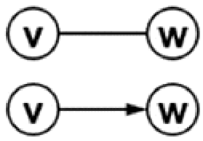
什么是图

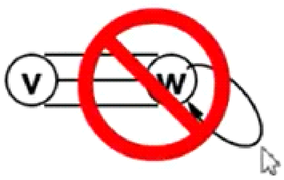
2023年7月30日　9:54

什么是“图”（Graph）



- 表示“多对多”的关系
- 包含
  - 一组顶点：通常用 **V (Vertex)** 表示顶点集合
  - 一组边：通常用 **E (Edge)** 表示边的集合
    - 边是顶点对：**(v, w)** ∈ **E**，其中 **v, w** ∈ **V**
    - 有向边 **<v, w>** 表示从v指向w的边（单行线）
    - 不考虑重边和自回路





屏幕剪辑的捕获时间: 2023/7/30 10:00

图的抽象数据类型

抽象数据类型定义

- 类型名称：图（Graph）
- 数据对象集：G(V,E)由一个非空的有限顶点集合V和一个有限边集合E组成。
- 操作集：对于任意图 G ∈ Graph，以及 v ∈ V，e ∈ E
  - `Graph Create()`：建立并返回空图；
  - `Graph InsertVertex(Graph G, Vertex v)`：将v插入G；
  - `Graph InsertEdge(Graph G, Edge e)`：将e插入G；
  - `void DFS(Graph G, Vertex v)`：从顶点v出发深度优先遍历图G；
  - `void BFS(Graph G, Vertex v)`：从顶点v出发宽度优先遍历图G；
  - `void ShortestPath(Graph G, Vertex v, int Dist[])`：计

- void BFS(Graph G, Vertex v)：从顶点v出发宽度优先遍历图G；
- void ShortestPath(Graph G, Vertex v, int Dist[])：计算图G中顶点v到任意其他顶点的最短距离；
- void MST(Graph G)：计算图G的最小生成树；
- ......

屏幕剪辑的捕获时间: 2023/7/30 10:01

无向图：图中所有边都是没有方向的

有向图：图中的边有可能双向，有可能单向

网络：每一条边上可以给出权重，形成的图叫做网络

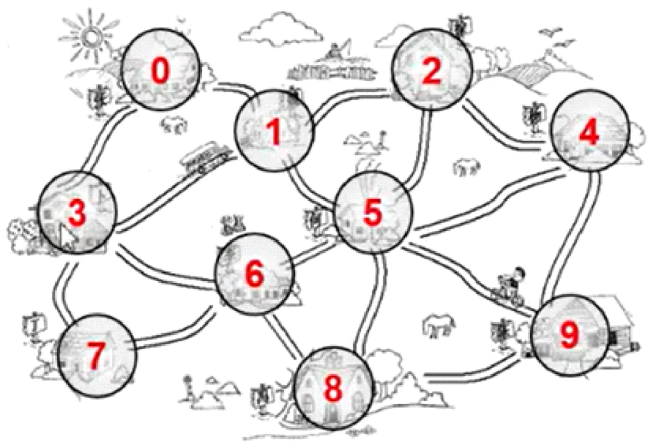
图的邻接矩阵表示法

2023年7月30日　　10:00

怎么在程序中表示一个图

- 邻接矩阵G[N][N]——N个顶点从0到N-1编号

$$G[i][j] = \begin{cases} 1 & \text{若} \langle v_i, v_j \rangle \text{是G中的边} \\ 0 & \text{否则} \end{cases}$$



	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
v_0	0	1	0	1	0	0	0	0	0	0
v_1	1	0	1	1	0	1	0	0	0	0
v_2	0	1	0	0	1	1	0	0	0	0
v_3	1	1	0	0	0	0	1	1	0	0
v_4	0	0	1	0	0	1	0	0	0	1
v_5	0	1	1	0	1	0	1	0	1	1
v_6	0	0	0	1	0	1	0	1	1	0
v_7	0	0	0	1	0	0	1	0	0	0
v_8	0	0	0	0	0	1	1	0	0	1
v_9	0	0	0	0	1	1	0	0	1	0

屏幕剪辑的捕获时间: 2023/7/30 10:12

- 邻接矩阵
  - 问题：对于无向图的存储，怎样可以省一半空间？

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
v_0	0	1	0	1	0	0	0	0	0	0
v_1	1	0	1	1	0	1	0	0	0	0
v_2	0	1	0	0	1	1	0	0	0	0
v_3	1	1	0	0	0	0	1	1	0	0
v_4	0	0	1	0	0	1	0	0	0	1
v_5	0	1	1	0	1	0	1	0	1	1
v_6	0	0	0	●	0	1	0	1	1	0
v_7	0	0	0	1	0	0	1	0	0	0
v_8	0	0	0	0	0	1	1	0	0	1
v_9	0	0	0	0	1	1	0	0	1	0

用一个长度为N(N+1)/2的1维数组A存储 $\{G_{00}, G_{10}, G_{11}, \ldots\ldots, G_{n-1\ 0}, \ldots, G_{n-1\ n-1}\}$，则 G_{ij} 在A中对应的下标是：

$$(i*(i+1)/2 + j)$$

对于网络，只要把G[i][j]的值定义为边 $\langle v_i, v_j \rangle$ 的权重即可。

屏幕剪辑的捕获时间: 2023/7/30 10:12

- 邻接矩阵 —— 有什么好处？
  - ☑ 直观、简单、好理解
  - ☑ 方便检查任意一对顶点间是否存在边
  - ☑ 方便找任一顶点的所有“邻接点”（有边直接相连的顶点）
  - ☑ 方便计算任一顶点的“度”（从该点发出的边数为“出度”，指向该点的边数为“入度”）
    - 无向图：对应行（或列）非0元素的个数
    - 有向图：对应行非0元素的个数是“出度”；对应列非0元素的个数是“入度”

屏幕剪辑的捕获时间: 2023/7/30 10:15

- 邻接矩阵 —— 有什么不好？
  - ☑ 浪费空间 —— 存稀疏图（点很多而边很少）有大量无效元素
    - 对稠密图（特别是完全图）还是很合算的
  - ☑ 浪费时间 —— 统计稀疏图中一共有多少条边

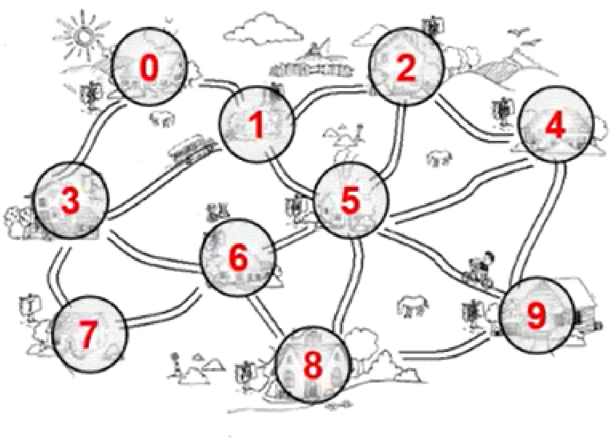
屏幕剪辑的捕获时间: 2023/7/30 10:16

图的邻接表表示法

2023年7月30日 10:18

- 邻接表：G[N]为指针数组，对应矩阵每行一个链表，只存非0元素

对于网络，结构中要增加权重的域。



G[0] → 1 → 3 → ●
G[1] → 5 → 3 → 0 → 2 → ●
G[2] → 1 → 5 → 4 → ●
G[3] → 7 → 1 → 0 → 6 → ●
G[4] → 2 → 5 → 9 → ●
G[5] → 2 → 1 → 4 → 6 → 8 → 9 → ●
G[6] → 5 → 8 → 7 → 3 → ●
G[7] → 6 → 3 → ●
G[8] → 9 → 5 → 6 → ●
G[9] → 4 → 5 → 8 → ●

一定要够稀疏才合算啊~~~~~

屏幕剪辑的捕获时间: 2023/7/30 10:21

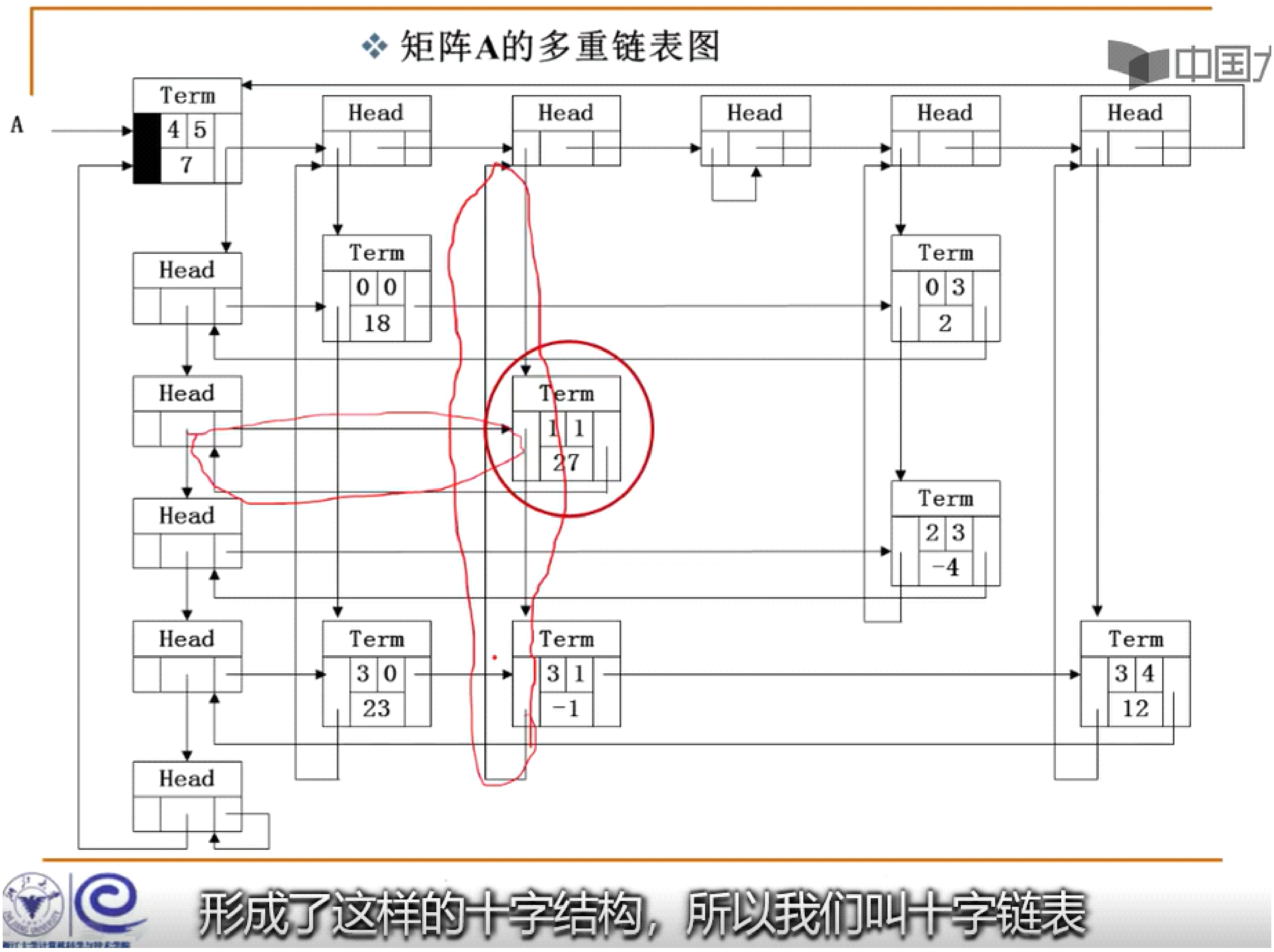
- 邻接表
  - ☑ 方便找任一顶点的所有“邻接点”
  - ☑ 节约稀疏图的空间
    - 需要N个头指针 + 2E个结点（每个结点至少2个域）
  - ☑ 方便计算任一顶点的“度”？
    - 对无向图：是的
    - 对有向图：只能计算“出度”；需要构造“逆邻接表”（存指向自己的边）来方便计算“入度”
  - ☑ 方便检查任意一对顶点间是否存在边？

    ☹ No

屏幕剪辑的捕获时间: 2023/7/30 10:23

十字链表

2023年7月30日 10:27



屏幕剪辑的捕获时间: 2023/7/30 10:27

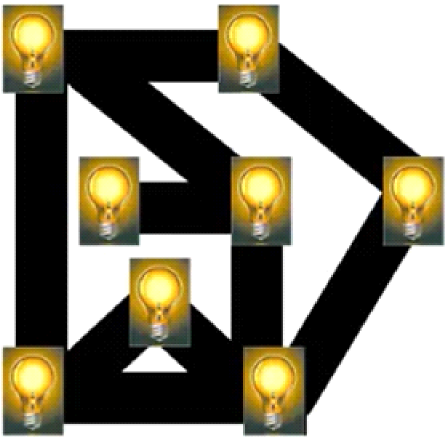
图的遍历

2023年7月30日　　17:49

深度优先搜索（DFS）

深度优先搜索(Depth First Search, DFS)

类似于树的先序遍历



```
void DFS ( Vertex V )
{ visited[ V ] = true;
  for ( V 的每个邻接点 W )
     if ( !visited[ W ] )
        DFS( W );
}
```

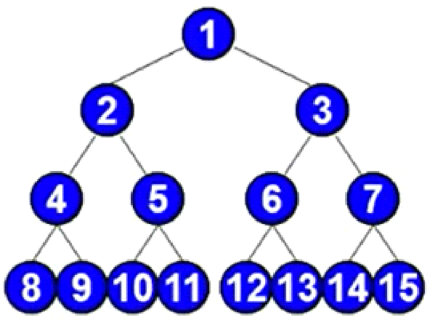
若有N个顶点、E条边，时间复杂度是

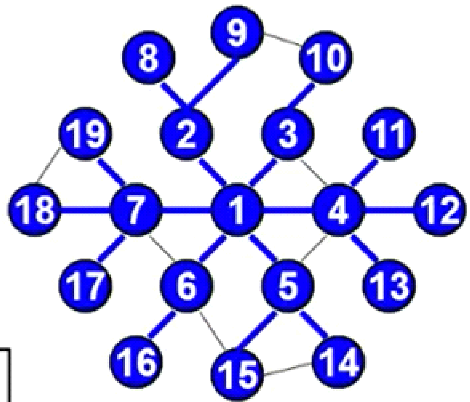
- 用邻接表存储图，有O(N+E)
- 用邻接矩阵存储图，有$O(N^2)$

屏幕剪辑的捕获时间: 2023/7/30 20:59

广度优先搜索（BFS）

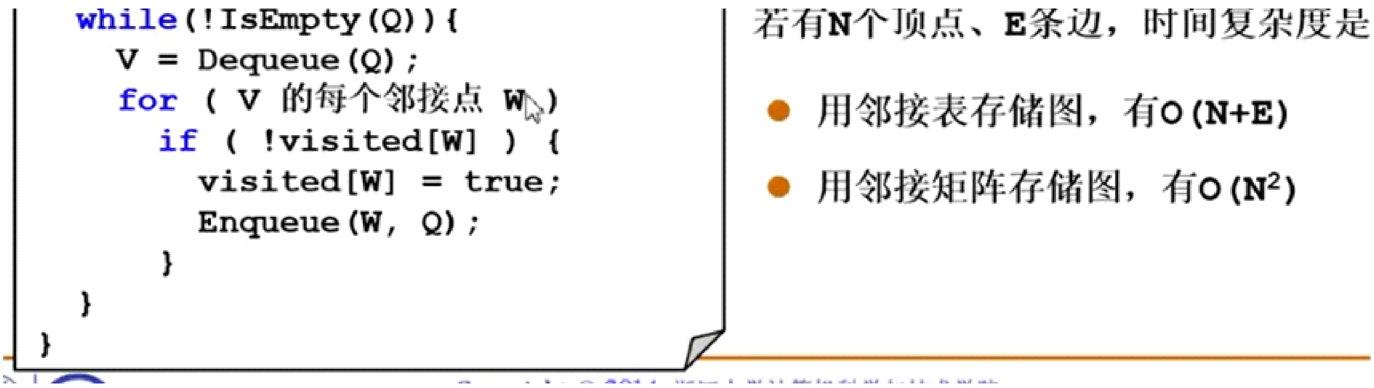
广度优先搜索(Breadth First Search, BFS)






```
void BFS ( Vertex V )
{ visited[V] = true;
  Enqueue(V, Q);
  while(!IsEmpty(Q)){
    V = Dequeue(Q);
    for ( V 的每个邻接点 W )
```

若有N个顶点、E条边，时间复杂度是



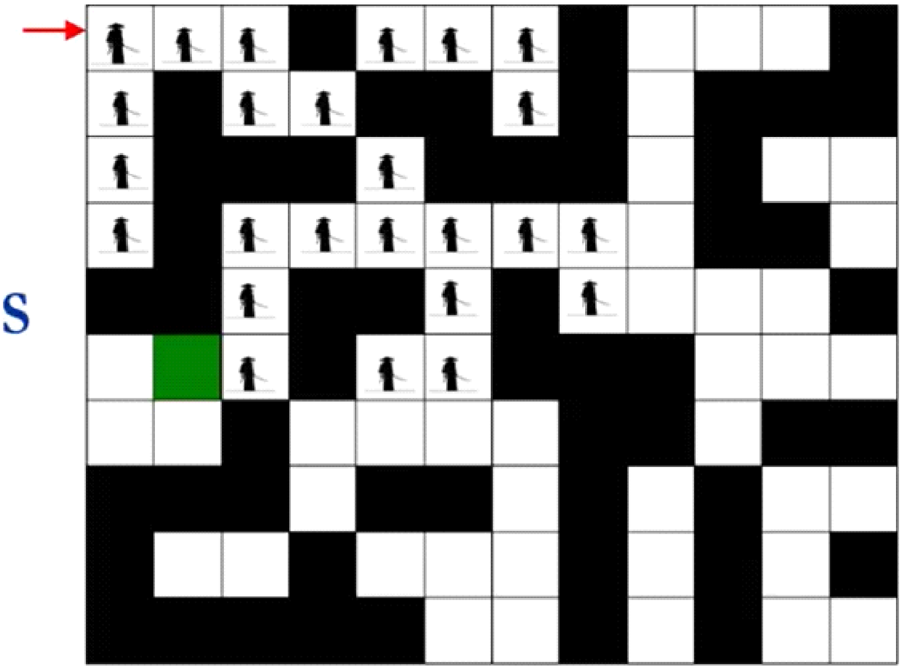
屏幕剪辑的捕获时间: 2023/7/30 21:05

两种算法功能的比较（迷宫为例）：

屏幕剪辑的捕获时间: 2023/7/30 21:18

为什么需要两种遍历？

BFS



屏幕剪辑的捕获时间: 2023/7/30 21:18

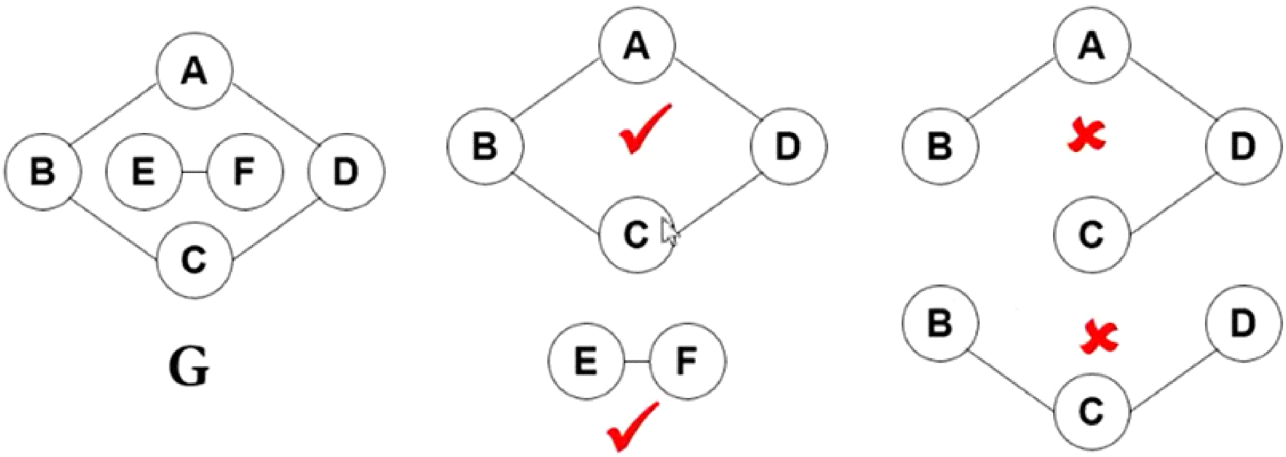
图的连通性

2023年7月30日 21:18

- 连通：如果从V到W存在一条（无向）路径，则称V和W是连通的
- 路径：V到W的路径是一系列顶点{V，v_1，v_2，…，v_n，W}的集合，其中任一对相邻的顶点间都有图中的边。路径的长度是路径中的边数（如果带权，则是所有边的权重和）。如果V到W之间的所有顶点都不同，则称简单路径
- 回路：起点等于终点的路径
- 连通图：图中任意两顶点均连通

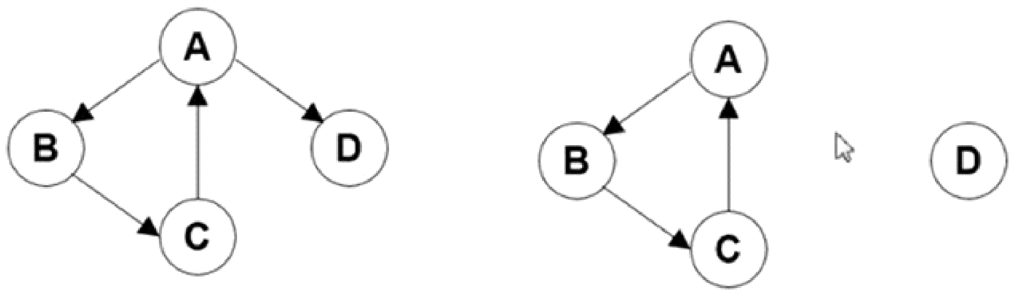
屏幕剪辑的捕获时间: 2023/7/30 21:21

- 连通分量：无向图的极大连通子图
  - 极大顶点数：再加1个顶点就不连通了
  - 极大边数：包含子图中所有顶点相连的所有边



屏幕剪辑的捕获时间: 2023/7/30 21:24

- **强连通**：有向图中顶点V和W之间存在双向路径，则称V和W是强连通的
- **强连通图**：有向图中任意两顶点均强连通
- **强连通分量**：有向图的极大强连通子图



屏幕剪辑的捕获时间: 2023/7/31 17:40

图不连通怎么办?

```
void DFS ( Vertex V )
{ visited[ V ] = true;
  for ( V 的每个邻接点 W )
     if ( !visited[ W ] )
        DFS( W );
}
```

每调用一次DFS(V)，就把V所在的连通分量遍历了一遍。BFS也是一样。

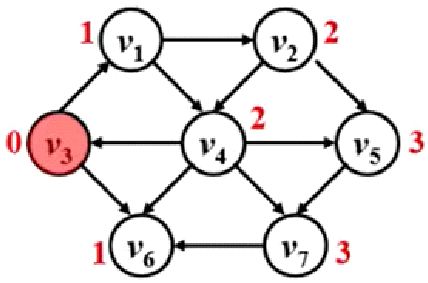
```
void ListComponents ( Graph G )
{ for ( each V in G )
    if ( !visited[V] ) {
      DFS( V ); /*or BFS( V )*/
    }
}
```

屏幕剪辑的捕获时间: 2023/7/31 17:41

无权图单源最短路径问题

2023年8月2日　　10:57

- 按照递增（非递减）的顺序找出到各个顶点的最短路



0: ☞ v_3

1: ☞ v_1 and v_6

2: ☞ v_2 and v_4

3: ☞ v_5 and v_7

BFS！

James Bond 从孤岛跳上岸，最少需要跳多少步？

屏幕剪辑的捕获时间: 2023/8/2 11:01

无权图的单源最短路算法

```
void BFS ( Vertex S )
{ visited[S] = true;
  Enqueue(S, Q);
  while(!IsEmpty(Q)){
    V = Dequeue(Q);
    for ( V 的每个邻接点 W )
      if ( !visited[W] ) {
        visited[W] = true;
        Enqueue(W, Q);
      }
  }
}
```

```
void Unweighted ( Vertex S )
{ Enqueue(S, Q);
  while(!IsEmpty(Q)){
    V = Dequeue(Q);
    for ( V 的每个邻接点 W )
      if ( dist[W]==-1  ) {
        dist[W] = dist[V]+1;
        path[W] = V;
        Enqueue(W, Q);
      }
  }
}
```

$T = O(|V| + |E|)$

dist[W] = S到W的最短距离

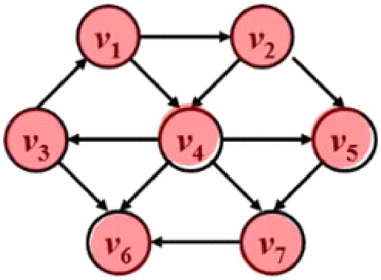
dist[S] = 0

path[W] = S到W的路上经过的某顶点

屏幕剪辑的捕获时间: 2023/8/3 10:00

无权图的单源最短路算法

```
void Unweighted ( Vertex S )
{ Enqueue(S, Q);
  while(!IsEmpty(Q)){
    V = Dequeue(Q);
    for ( V 的每个邻接点 W )
      if ( dist[W]==-1  ) {
        dist[W] = dist[V]+1;
        path[W] = V;
        Enqueue(W, Q);
      }
  }
}
```



下标	1	2	3	4	5	6	7
dist	1	2	0	2	3	1	3
path	3	1	-1	1	2	3	4

屏幕剪辑的捕获时间: 2023/8/3 10:21

有权图单源最短路径

2023年8月3日 10:21

- **Dijkstra 算法**
  - 令S={源点s + 已经确定了最短路径的顶点v_i}
  - 对任一未收录的顶点v，定义dist[v]为s到v的最短路径长度，但该路径仅经过S中的顶点。即路径$\{s\rightarrow(v_i\in S)\rightarrow v\}$的最小长度
  - 若路径是按照递增（非递减）的顺序生成的，则
    - 真正的最短路必须只经过S中的顶点（为什么？）
    - 每次从未收录的顶点中选一个dist最小的收录（贪心）
    - 增加一个v进入S，可能影响另外一个w的dist值！

屏幕剪辑的捕获时间: 2023/8/3 10:39

Dijkstra算法成立的最重要的前提是**路径按照非递减的顺序生成**

每把一个v收进集合S，如果使得s（源点）到w（未被收进S）的最短距离变小，则v一定在s->w的路径上，而且一定有一条v指向w的边

- 增加一个v进入S，可能影响另外一个w的dist值！
  - dist[w] = min{dist[w], dist[v] + <v,w>的权重}

屏幕剪辑的捕获时间: 2023/8/3 10:43

因此Dijkstra算法解决不了有负边的情况，满足不了路径按非递减顺序生成的条件

- 方法1：直接扫描所有未收录顶点 – $O(|V|)$
  - $T = O(|V|^2 + |E|)$ —— 对于稠密图效果好

- 方法2：将dist存在最小堆中 – $O(\log|V|)$
  - 更新dist[w]的值 – $O(\log|V|)$
  - $T = O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$

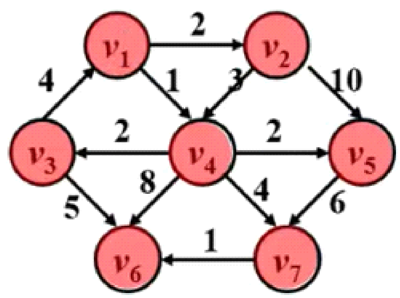
- $T = O(\,|V|\log|V| + |E|\log|V|\,) = O(\,|E|\log|V|\,)$

对于稀疏图效果好

屏幕剪辑的捕获时间: 2023/8/3 10:51

有权图的单源最短路算法

```
void Dijkstra( Vertex s )
{ while (1) {
    V = 未收录顶点中dist最小者;
    if ( 这样的V不存在 )
      break;
    collected[V] = true;
    for ( V 的每个邻接点 W )
      if ( collected[W] == false )
        if ( dist[V]+E<V,W> < dist[W] ) {
          dist[W] = dist[V] + E<V,W> ;
          path[W] = V;
        }
  }
}
```



下标	1	2	3	4	5	6	7
dist	0	2	3	1	3	6	5
path	-1	1	4	1	4	7	4

屏幕剪辑的捕获时间: 2023/8/3 11:03

多源最短路算法

2023年8月3日　　10:42

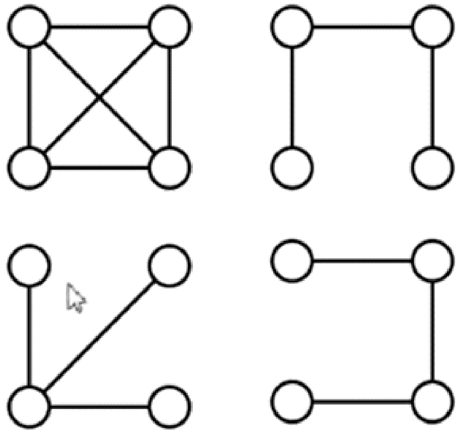
Floyd算法

最小生成树问题

2023年8月5日 9:22

什么是最小生成树(Minimum Spanning Tree)

- 是一棵树
  - 无回路
  - |V|个顶点一定有|V|-1条边
- 是生成树
  - 包含全部顶点
  - |V|-1条边都在图里



屏幕剪辑的捕获时间: 2023/8/5 9:24

贪心算法

- 什么是“贪”：每一步都要最好的
- 什么是“好”：权重最小的边
- 需要约束：
  - 只能用图里有的边
  - 只能正好用掉|V|-1条边
  - 不能有回路

屏幕剪辑的捕获时间: 2023/8/5 9:31

Prim算法 — 让一棵小树长大

```
void Dijkstra( Vertex s )
{ while (1) {
    V = 未收录顶点中dist最小者;
    if ( 这样的V不存在 )
      break;
    collected[V] = true;
    for ( V 的每个邻接点 W )
      if ( collected[W] == false )
        if ( dist[V]+E<V,W> < dist[W] ){
          dist[W] = dist[V] + E<V,W> ;
          path[W] = V;
        }
  }
}
```

dist[V] = $E_{(s,V)}$或 正无穷

```
void Prim()
{ MST = {s};
  while (1) {
    V = 未收录顶点中dist最小者;
    if ( 这样的V不存在 )
      break;
    将V收录进MST:
    for ( V 的每个邻接点 W )
      if ( W未被收录 )
        if ( E(V,W) < dist[W] ){
          dist[W] = E(V,W) ;
          parent[W] = V;
        }
  }
}
```

屏幕剪辑的捕获时间: 2023/8/5 9:38

Kruskal算法 — 将森林合并成树

```
void Kruskal ( Graph G )
{   MST = { } ;
    while ( MST 中不到 |V| -1 条边 && E 中还有边 ) {
        从 E 中取一条权重最小的边 E(v,w) ;   /* 最小堆 */
        将 E(v,w) 从 E 中删除;
        if ( E(V,W)不在 MST 中构成回路)   /* 并查集 */
            将 E(V,W) 加入 MST;
        else
            彻底无视 E(V,W);
    }
}
```

屏幕剪辑的捕获时间: 2023/8/5 9:48

难点在于如何判断是否构成回路，其实利用并查集就很好解决：

注意到Kruskal算法在开始时把每个结点都视为一棵树，各个结点是不同的集合，所以之后在

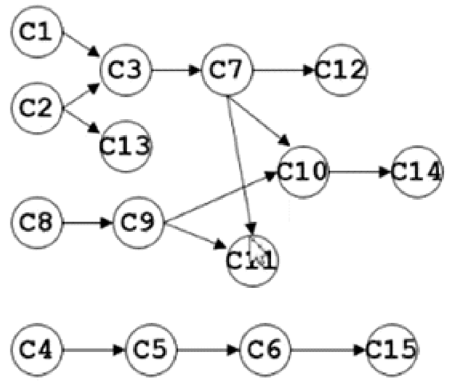
把一条边<V,W>加入最小生成树中时，判断V与W是否属于同一个集合即可，如果是同一集合，那加上这条边后一定会构成回路。

拓扑排序

2023年8月5日 9:57

例：计算机专业排课

课程号	课程名称	预修课程
C1	程序设计基础	无
C2	离散数学	无
C3	数据结构	C1, C2
C4	微积分（一）	无
C5	微积分（二）	C4
C6	线性代数	C5
C7	算法分析与设计	C3
C8	逻辑与计算机设计基础	无
C9	计算机组成	C8
C10	操作系统	C7, C9
C11	编译原理	C7, C9
C12	数据库	C7
C13	计算理论	C2
C14	计算机网络	C10
C15	数值分析	C6



AOV (Activity On Vertex) 网络

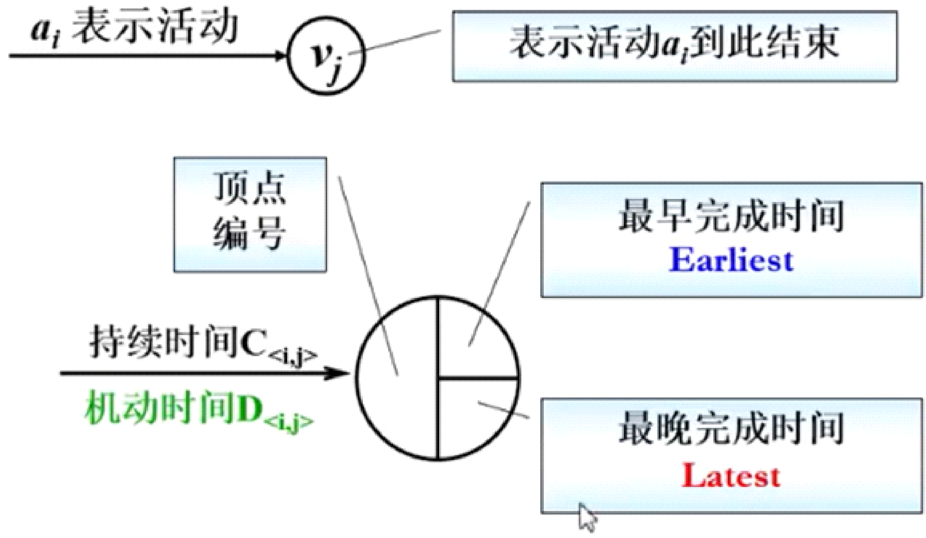
屏幕剪辑的捕获时间: 2023/8/5 10:01

- 拓扑序：如果图中从V到W有一条有向路径，则V一定排在W之前。满足此条件的顶点序列称为一个拓扑序
- 获得一个拓扑序的过程就是拓扑排序
- AOV如果有合理的拓扑序，则必定是有向无环图（Directed Acyclic Graph, DAG）

屏幕剪辑的捕获时间: 2023/8/5 10:02

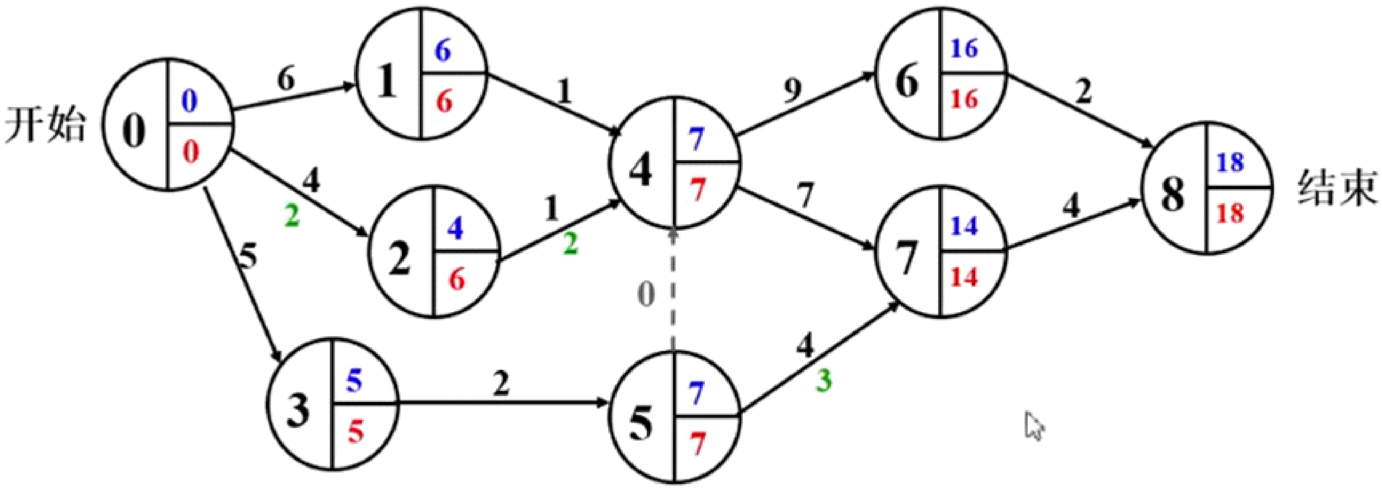
关键路径问题

- **AOE (Activity On Edge)** 网络
  - 一般用于安排项目的工序



屏幕剪辑的捕获时间: 2023/8/5 10:18

关键路径问题 由绝对不允许延误的活动组成的路径



问题1：整个工期有多长？ Earliest[8] = 18

Earliest[0] = 0;

$\text{Earliest}[j] = \max\limits_{<i,j>\in E} \{ \text{Earliest}[i]+C_{<i,j>} \};$

问题2：哪几个组有机动时间？ $D_{<i,j>} = \text{Latest}[j]-\text{Earliest}[i]-C_{<i,j>}$

Latest[8] = 18;

$\text{Latest}[i] = \min\limits_{<i,j>\in E} \{ \text{Latest}[j]-C_{<i,j>} \};$

屏幕剪辑的捕获时间: 2023/8/5 10:35