图

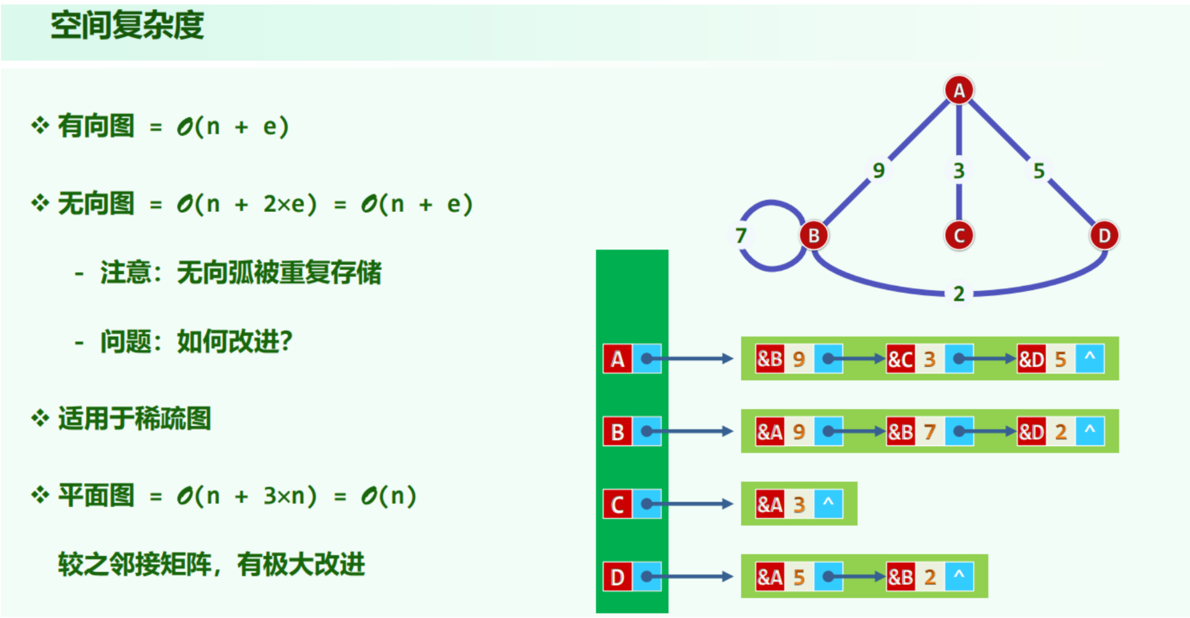
图的表示

邻接矩阵

优点：直观，判断两点是否存在边，判断顶点出度入度均为 $\mathrm{O}(1)$

缺点：空间复杂度 $\mathrm{O}(n^2)$

邻接表



时间复杂度（1/2）

❖ 建立邻接表（递增式构造）：$\mathcal{O}$(n + e) //如何实现

❖ 枚举所有以顶点v为尾的弧：$\mathcal{O}$(1 + deg(v)) //遍历v的邻接表

❖ 枚举（无向图中）顶点v的邻居：$\mathcal{O}$(1 + deg(v)) //遍历v的邻接表

❖ 枚举所有以顶点v为头的弧：$\mathcal{O}$(n + e) //遍历所有邻接表

可改进至$\mathcal{O}$(1 + deg(v)) //建立逆邻接表——为此，空间需增加多少？

❖ 计算顶点v的出度/入度：
- 增加度数记录域：$\mathcal{O}$(n)附加空间
- 增加/删除弧时更新度数：$\mathcal{O}$(1)时间 //总体$\mathcal{O}$(e)时间
- 每次查询：$\mathcal{O}$(1)时间！

时间复杂度（2/2）

❖ 给定顶点u和v，判断是否<u, v> ∈ E
- 有向图：搜索u的邻接表，$\mathcal{O}$(deg(u)) = $\mathcal{O}$(e)
- 无向图：搜索u或v的邻接表，$\mathcal{O}$(max(deg(u), deg(v))) = $\mathcal{O}$(e)
- “并行”搜索：$\mathcal{O}$(2 × min(deg(u), deg(v))) = $\mathcal{O}$(e)

能够达到邻接矩阵的$\mathcal{O}$(1)吗？

❖ 散列！如果装填因子选取得当 //保持兴趣
- 弧的判定：expected-$\mathcal{O}$(1)，与邻接矩阵“相同”
- 空间：$\mathcal{O}$(n + e)，与邻接表相同

❖ 为何有时仍使用邻接矩阵？仅仅因为实现简单？不，有更多用处！比如，可处理 Euclidean graph和intersection graph之类的隐式图（implicitly-represented graphs）

图的遍历

广度优先搜索

用队列实现

遍历过程中 BFS 会把所有点分成三类：

- undiscovered，还没有被找到
- discovered，刚进入函数，正在进行处理
- visited，也就是遍历到并处理完了这个节点。

遍历完成后，BFS 会把所有边分成两类：

TREE 边，从 v 到 u 的时候， u 还处于 undiscovered

CROSS 边，从 v 到 u 的时候， u 是 discovered or visited

BFS性质：树边构成为最短路

应用：

- Bipartite Graph（不存在一条跨边，两端点到根距离相等）
- Eccentricity/Radius/Diameter/Center：定义以该点为根BFS最远距离为该点的“偏心距”

深度优先搜索

在遍历过程中，DFS 会把所有点分成三类，与 BFS 是相同的。

在遍历完成后，DFS 会给所有点一个 dtime 和一个 ftime ，分别代表着 discovered time 和 finished time。所以也就有了活跃期： `active[u] = ( dTime[u], fTime[u])` ，活跃期是一种“划分的关系”，要么交为空，要么包含。

DFS 也会把所有边分成三类：

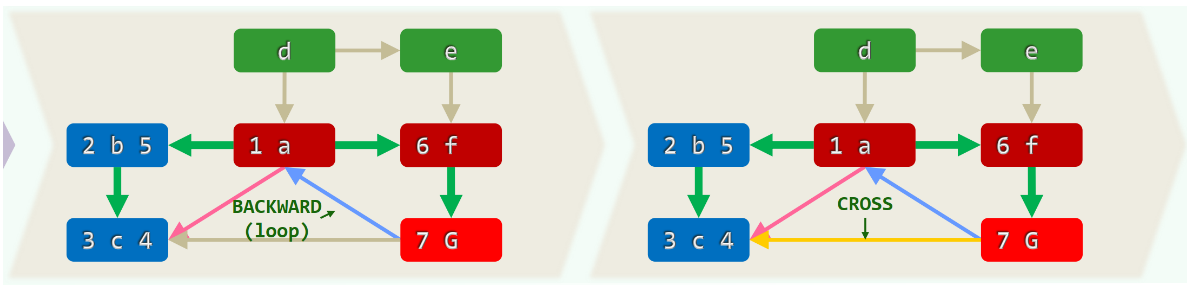
TREE，从 v 到 u 的时候， u 还是 undiscovered

BACKWARD，从 v 到 u 的时候，u 正在被处理 discovered ，这说明 u 在 v 到根节点的链上。

FORWARD / CROSS，从 v 到 u 的时候，u 已经 visited(只有有向图)，根据时间戳来给forward 或者 cross。

```
template<typename Tv, typename Te> void Graph<Tv, Te>::DFS( Rank v, Rank& clock )
   dTime(v) = ++clock; status(v) = DISCOVERED; //发现当前顶点v
v  for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //考察v的每一邻居u
      switch ( status(u) ) { //并视其状态分别处理
      u  case UNDISCOVERED: //u尚未发现，意味着支撑树可在此拓展
            type(v, u) = TREE; parent(u) = v; DFS( u, clock ); break; //递归
      u  case DISCOVERED: //u已被发现但尚未访问完毕，应属被后代指向的祖先
            type(v, u) = BACKWARD; break;
      u  default: //u已访问完毕（VISITED，有向图），则视承袭关系分为前向边或跨边
            type(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS; break;
      } //switch
v  status(v) = VISITED; fTime(v) = ++clock; //至此，当前顶点v方告访问完毕
}
```

CROSS连接的是不在同一分支（相对于根来说）的结点，FOWARD和BACKWORD连接同一分支的结点，如下图中表示的就是以1a为根DFS的结果



DFS性质：A的活跃周期（Discovered到Visited）完全包含于B，当且仅当A是B的后代，B是A的祖先。

时间复杂度分析

广度优先搜索：$O(n+e)$

深度优先搜索：$O(n+e)$

一定要区分 n 和 e...

拓扑排序

DAG 的零入度、出度算法。

“Begin with the end in mind.”