

6. 图

(f) 优先级搜索

邓俊辉

deng@tsinghua.edu.cn

通用算法

❖ 各种遍历算法的区别，仅在于选取顶点进行访问的次序

广度 / 深度：优先访问与更早 / 更晚 被发现的顶点相邻接者

...

❖ 不同的遍历算法，取决于顶点的选取策略

❖ 不同的顶点选取策略，取决于存放和提供顶点的 数据结构——Bag

❖ 此类结构，为每个顶点 v 维护一个 优先级数 $priority(v)$

每个顶点都有 初始 优先级数；并可能随算法的推进而 调整

❖ 通常的习惯是，优先级数越 大/小，优先级越 低/高

特别地， $priority(v) == INT_MAX$ ，意味着 v 的优先级最低

统一框架

```
❖ template <typename Tv, typename Te> //顶点类型、边类型  
    template <typename PU> //优先级更新器 (函数对象)  
    void Graph<Tv, Te>::pfs( int s, PU prioUpdater ) { //PU的策略, 因算法而异  
        priority(s) = 0; status(s) = VISITED; parent(s) = -1; //起点s加至PFS树中  
        while (1) { //将下一顶点和边加至PFS树中  
            /* ... 依次引入n - 1个顶点 (和n - 1条边) ... */  
        } //while  
    } //如何推广至非连通图?
```

统一框架

```
❖ while (1) { //依次引入n - 1个顶点 (和n - 1条边)

    for ( int w = firstNbr(s); -1 < w; w = nextNbr(s, w) ) //对s各邻居w

        prioUpdater( this, s, w ); //更新顶点w的优先级及其父顶点

    for ( int shortest = INT_MAX, w = 0; w < n; w++ )

        if ( UNDISCOVERED == status(w) ) //从尚未加入遍历树的顶点中

            if ( shortest > priority(w) ) //选出下一个

                { shortest = priority(w); s = w; } //优先级最高的顶点s

    if ( VISITED == status(s) ) break; //直至所有顶点均已加入

    status(s) = VISITED; type( parent(s), s ) = TREE; //将s加入遍历树

} //while
```

复杂度

- ❖ 执行时间主要消耗于内、外两重循环；其中两个内循环前、后并列
- ❖ 前一内循环的累计执行时间： 若采用邻接矩阵，为 $O(n^2)$ ；若采用邻接表，为 $O(n + e)$
后一循环中，优先级更新的次数呈算术级数变化 $\{ n, n - 1, \dots, 2, 1 \}$ ，累计为 $O(n^2)$
两项合计，为 $O(n^2)$
- ❖ 后面将会看到：若采用优先队列，以上两项将分别是 $O(e \log n)$ 和 $O(n \log n)$ //保持兴趣
两项合计，为 $O((e + n) * \log n)$
- ❖ 这是很大的改进——尽管对于稠密图而言，反而是倒退 //已有接近于 $O(e + n \log n)$ 的算法
- ❖ 基于这个统一框架，如何解决具体的应用问题...