11-C

图应用

优先级搜索

邓俊辉
deng@tsinghua.edu.cn

堯舜之知而不遍物，急先務也；堯舜之仁不遍愛人，急親賢也

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 (1/2)

```
template <typename Tv, typename Te>
template <typename PU> //优先级更新器（函数对象）
void Graph<Tv, Te>::PFS( Rank v, PU prioUpdater ) { //PU的策略，因算法而异
   priority(v) = 0; status(v) = VISITED; //起点v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 (2/2)

```
for ( Rank k = 1; k < n; k++ ) { //逐步将n-1顶点和n-1条边加至PFS树中

   for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //对v的每一个邻居u
      prioUpdater( this, v, u ); //更新其优先级及其父亲

   int shortest = INT_MAX;
   for ( Rank u = 0; u < n; u++ ) //从尚未加入遍历树的顶点中，选出下一个优先级
      if ( (UNDISCOVERED == status(u)) && (shortest > priority(u)) ) //最高的
         { shortest = priority(u); v = u; } //顶点v

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

} //for
```

复杂度

- 执行时间主要消耗于内、外两重循环；其中内循环有两个（类），前、后并列
- 前一类循环（更新）：若采用**邻接矩阵**，累计耗时 $\mathcal{O}(n^2)$；若采用**邻接表**，耗时 $\mathcal{O}(n+e)$

  后一类循环（择优）：每次耗时 $\mathcal{O}(n)$，累计 $\mathcal{O}(n^2)$

  两类合计，为 $\mathcal{O}(n^2)$
- 后面将会看到：若采用**优先级队列**，以上两项将分别是 $\mathcal{O}(e \cdot \log n)$ 和 $\mathcal{O}(n \cdot \log n)$ //保持兴趣

  合计为 $\mathcal{O}((n+e) \cdot \log n)$
- 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$\mathcal{O}$(e + n*logn)的算法
- 基于这个统一**框架**，如何解决具体的应用问题...