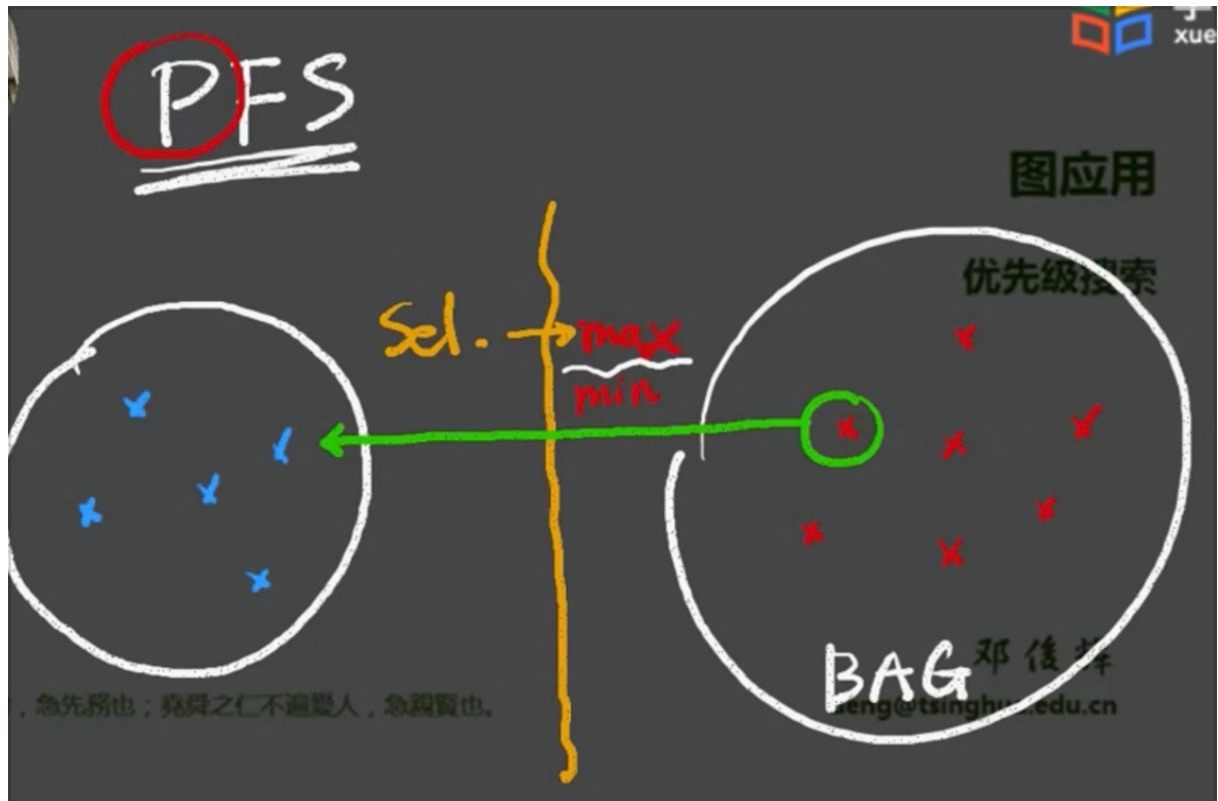


07B 优先级搜索 Priority Search

#数据结构邓神

BAG



ADT

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—— $\text{priority}(v)$
 - 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， $\text{priority}(v) == \text{INT_MAX}$ ，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( 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 (2/2)



```
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

} //while
```

```
template <typename Tv, typename Te>
template <typename PU>
void Graph<Tv, Te>::pfs(int s, PU prioUpdater){
    priority(s) = 0;
    status(s) = VISITED;
    parent(s) = -1;
    while (1){
        for (int w = firstNbr(s); -1 < w; w = nextNbr(s) {
            prioUpdater(this, s, w);
        }
    }
}
```

```

for (int shortest = INT_MAX, w = 0; w < n; w++) {
    if(UNDETERMINED == status(w)){
        if(shortest > priority(w)){
            shortest = priority(w);
            s = w;
        }
    }
}
if(VISITED == status(s)){
    break;
}
status(s) = VISITED;
type(parent(s), s) = TREE;
}
}

```

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**优先级队列**，以上两项将分别是 $O(e \cdot \log n)$ 和 $O(n \cdot \log n)$ //保持兴趣
 两项合计，为 $O((e+n) \cdot \log n)$
- ❖ 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 $O(e + n \cdot \log n)$
- ❖ 基于这个统一框架，如何解决具体的应用问题...

