12-E

优先级队列

多叉堆

他说到一件事，时常萦绕我的心头，就是每个人若能窥清其他人的心意，那么愿意下来的人会多于愿意高升的人

邓俊辉
deng@tsinghua.edu.cn

优先级搜索

❖ 回顾图的PFS以及统一框架：g->pfs()...

❖ 无论何种算法，差异仅在于所采用的优先级更新器prioUpdater()

- Prim算法： g->pfs(0, PrimPU());
- Dijkstra算法： g->pfs(0, DijkstraPU());

❖ 每一节点引入遍历树后，都需要

- 更新树外顶点的优先级（数），并
- 选出新的优先级最高者

❖ 若采用邻接表，两类操作的累计时间，分别为$\mathcal{O}(n+e)$和$\mathcal{O}(n^2)$

❖ 能否更快呢？

优先级队列

❖ 自然地，PFS中的各顶点可组织为**优先级队列**

❖ 为此需要使用PQ接口

heapify():	由n个顶点创建初始PQ	总计$\mathcal{O}(n)$
delMax():	取优先级最高（极短）跨边(u, w)	总计$\mathcal{O}(n * \log n)$
increase():	更新所有关联顶点到U的距离，提高优先级	总计$\mathcal{O}(e * \log n)$

❖ 总体运行时间 = $\mathcal{O}((n+e)*\log n)$

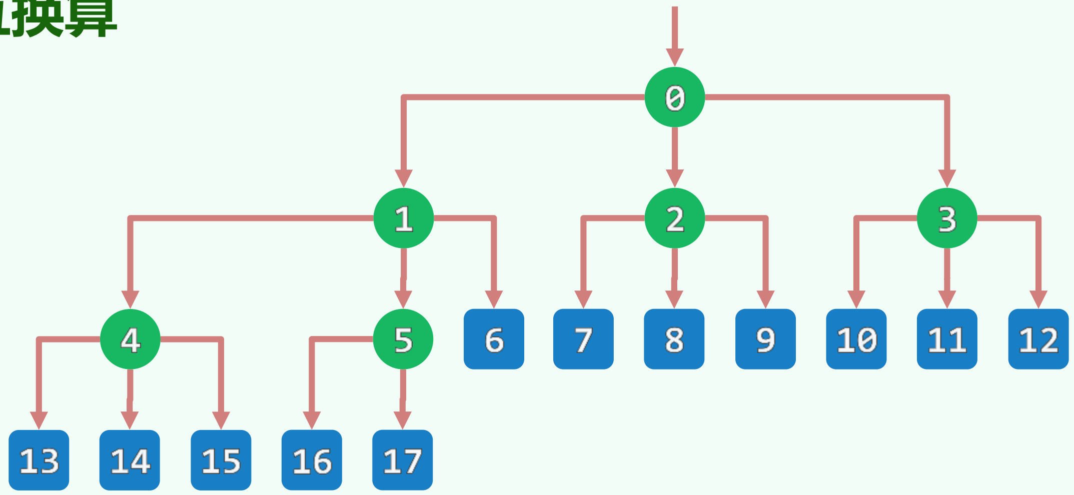
- 对于**稀疏图**，处理效率很高
- 对于**稠密图**，反而不如常规实现的版本

❖ 有无更好的办法？

多叉堆

❖ 仍可基于向量实现，且父、子节点的**秩**可简明地相互换算

- $parent(k) = \lfloor (k-1)/d \rfloor$
- $child(k,i) = k \cdot d + i,\ 0 < i \le d$



❖ 当然，d不再是2的幂时

将不再能够借助**移位**加速秩的换算

❖ heapify()：$\mathcal{O}(n)$

不可能再快了（直接写入，亦不过如此）

❖ delMax()：$\mathcal{O}(\log n)$

实质就是percolateDown() —— 已是极限了（为什么？）

❖ increase()：$\mathcal{O}(\log n)$ —— 实质就是percolateUp() —— 似乎仍有改进空间...

上山容易下山难

❖ 若将二叉堆改成多叉堆（d-heap），则堆高降至

$$\mathcal{O}(\log_d n)$$

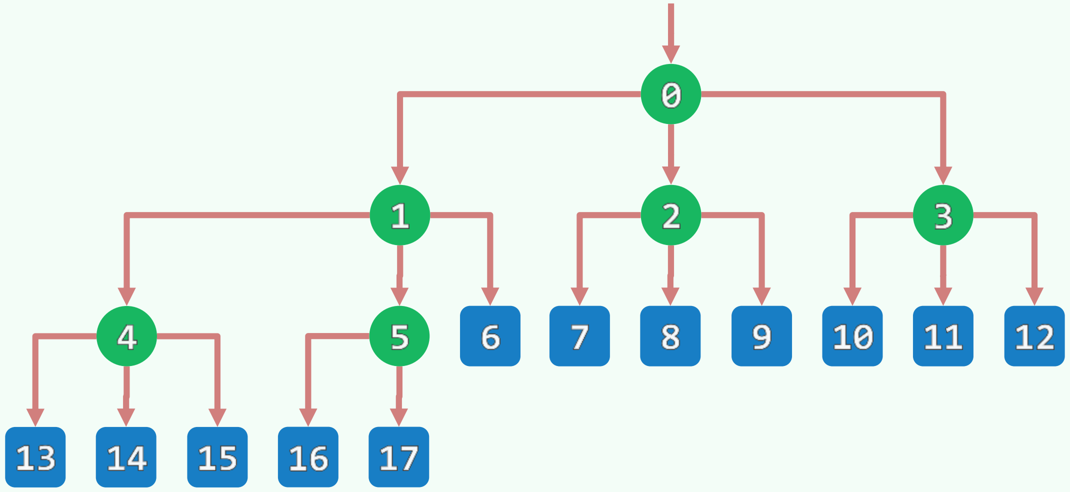
❖ 相应地，上滤成本降至

$$\log_d n$$

但下滤成本却增至

$$d \cdot \log_d n \;=\; \frac{d \cdot \ln 2}{\ln d} \cdot \log_2 n$$

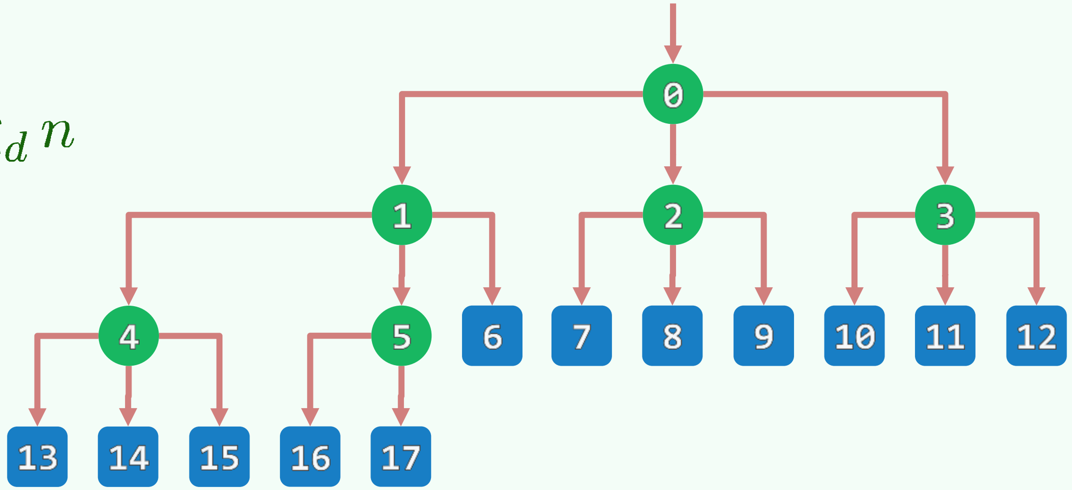
❖ 综合考量，利大于弊...



PFS

❖ 如此，PFS的运行时间将是：

$$n \cdot d \cdot \log_d n \ + \ e \cdot \log_d n \ = \ (n \cdot d \ + \ e) \cdot \log_d n$$

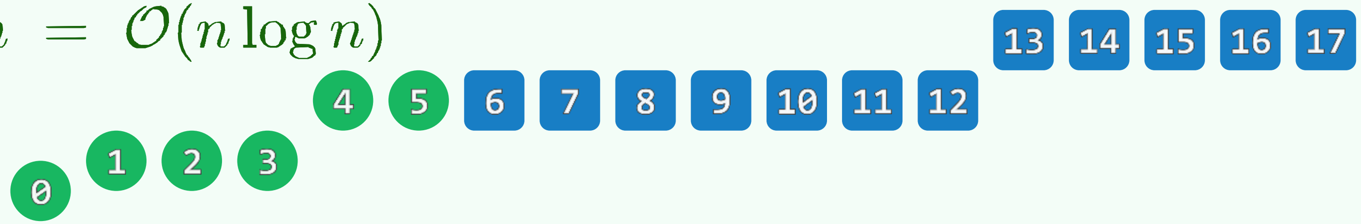


❖ 取 $d \ \approx \ e/n + 2$ 时

总体性能达到最优：$\mathcal{O}(e \cdot \log_{(e/n+2)} n)$

❖ 对于稀疏图保持高效：

$$e \cdot \log_{(e/n+2)} n \ \approx \ n \cdot \log_{(n/n+2)} n \ = \ \mathcal{O}(n \log n)$$



对于稠密图改进极大：

$$e \cdot \log_{(e/n+2)} n \ \approx \ n^2 \cdot \log_{(n^2/n+2)} n \ \approx \ n^2 \ = \ \mathcal{O}(e)$$

对于一般的图，会自适应地实现最优

Fibonacci堆

- 左式堆 x (新的上滤算法 + 懒惰合并)
- 各接口的**分摊**复杂度

delMax()	$\mathcal{O}(\log n)$
insert()	$\mathcal{O}(1)$
merge()	$\mathcal{O}(1)$
increase()	$\mathcal{O}(1)$

- 于是，基于**PFS框架**的算法采用Fibonacci堆后，运行时间自然就是

$$n \cdot \mathcal{O}(\log n) \;+\; e \cdot \mathcal{O}(1) \quad = \quad \mathcal{O}(e + n\log n)$$