chapter 5: Greedy algorithm

You Me

date: Yesterday

目录

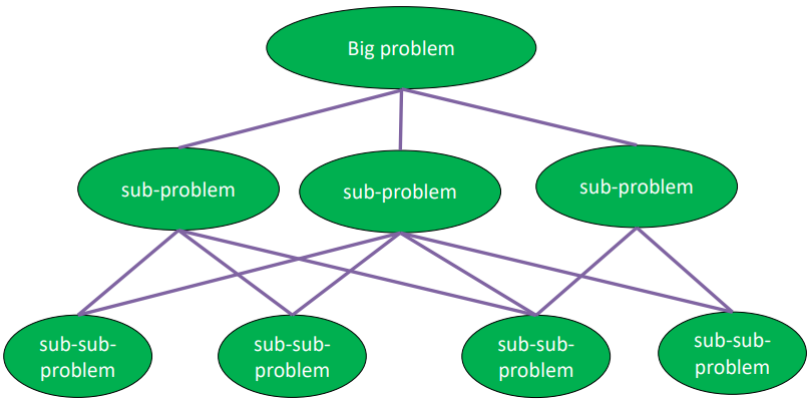

图 1: dp

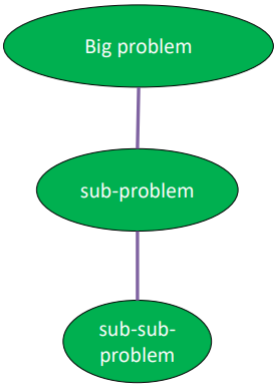

图 2: gd

1 what is greedy algorithm

We say that divide and conquer divides a problem into some small but independent sub-problem, while the dp divides the problem into some small subproblems that may overlap. figure 1 tells us about the structure of dp.

However, this greedy algorithm requires a high-demanding structure of the problem, which is called greedy 选择性, which can be described in figure 2.

The structure requires not only the optimal substrutrue but also requires the property that a big problem is based on one small problem. The algorithm is to decompose the solution to a complex problem into a sequence of operaitons of 'locally optimal choice', which extend the current local solution to a bigger one, till the solution can not be bigger.

Similar approuch can be applied to some other situation even if the resulted solution is not optimal.

2 the basic principles of greedy

The main idea of greedy algorithm can be described as follows

1. The solution is a sequence of operations, while Divide and conquer have some parrallel sequences of operations.
2. Every operaitons makes a locally optimal choice.
3. It requires proof of whether this approuch leads to optimal solution.

Definition 1. *The property that locally optimal choice leads to globally optimal choice is called* 贪心选择性.

2.0.1 the comparison to dp

dp Every choice is based on the solution to the subproblems. And it starts from button to top.

greedy Every choice is locally optimal, based on only the current situation. And it starts from top to bottom, with every choice decreasing the scale of the problems.

3 任务安排问题

3.1 问题描述

Definition 2. S 是 n 个活动的集合, 这几个活动记为 n, *i.e.* $S = \{1, 2, 3, .., n\}$. 每一个活动有开始时间和结束时间记为 $[s_i, f_i]$ [1]. 这几个活动不能有重叠, 不能同时进行活动, 我们要求出最大相容活动集合 A, 即 A 里面的活动不会发生矛盾

3.2 优化解的结构分析

Lemma 1. S 的某个优化解包括活动 1

证明. 设我们已知一个优化解 A, k 是其第一个活动的下标, j 是第二活动的下标

$k = 1$ 时候成立

$k \neq 1$ 的时候, 有 $f_1 \leq f_k \leq s_j$, 其中 $f_k \leq s_j$ 是因为相容. 令 $B = A - \{k\} \cup \{1\}$ 明显 $|A| = |B|$ □

Lemma 2 (优化子结构). 设 $S = \{1, \cdots, n\}$, 并且包括活动 1 , 那么 $A' = A - \{1\}$ 是 $S' = \{i \in S : s_i > f_1\}$ 的子问题的优化解

证明. 使用反证法.

如果说 A' 不是 S' 的优化解 (优化解在我这里 aka 最优解) , 设存在另一个 A'' 比 A' 更优, 那么 $A'' + \{1\}$ 比 A 更优, 则矛盾.

一般优化子结构都是使用反证法. □

[1] s for start, and f for fin

Lemma 3 (符合贪心选择性). *设 S 是 n 个活动, l_i 是 $S_i = \{j \in S : s_j \geq f_{i-1}\}$ (特别的, $f_0 = 0$) 中具有最小结束时间的活动, 设 A 是包含了活动 1 的优化解, 那么 $A = \bigcup_{i=1}^{k} \{l_i\}$*

证明. 使用归纳法

$|A| = 1$ 时候, 结论成立

$|A| < k$ 的时候, 假设成立; 那么 $|A| = k$ 的时候, 考虑 $A' = A - 1$, 由于引理 2 和假设: $A' = \bigcup_{i=2}^{k} \{l_i\}$

就有 $A = A' \cup \{1\} = \bigcup_{i=1}^{k} \{l_i\}$ □

算法正确性证明:

- 活动选择问题具有优化子结构
- 活动选择问题具有贪心选择性
- 算法按照贪心选择性计算优化解

3.3 伪代码

算法

Greedy-Activity-Selector

Input: 活动的开始，结束时间数组S，F，设$f_1 \leq f_2 \leq \cdots \leq f_n$（已排序）

Output: 选择的活动集合

执行: Greedy-Activity-Selector(S, F)

过程: Greedy-Activity-Selector(S, F)

```
1. n ← length(S)
2. A ← {1}
3. j ← 1
4. For  i←2  To  n  Do
5.      IF   s_i ≥ f_j(当前第i个活动开始的时间大于第j个活动结束的时间)
6.      Then  A←A∪{i}; j←i;
7. Return  A
```

图 3: 活动优化伪代码

Theorem 1. *我们上面给出的算法能够产生最优解*

证明. 使用上面给出的三个引理就足以证明.

由于引理 1, 我们选取活动 1, 这是合法的. 然后找出 $S' = j : s_j \geq f_1$ 的优化解. 是递归地来求解. □

3.4 复杂度分析

不用怎么分析.

$$T(n) = \Theta(n)$$

上面只有一个循环.

4 哈夫曼编码问题

4.1 编码问题

我们有很多种编码方法, 比如说什么 ascii 码什么的. 我们可以进行一个分类.

1. 二进制编码: 每一个编码使用一个二进制 sequence 来表示
2. 固定长度编码: 每一个字符都是使用固定长度的 01 sequence 来表示.
3. 可变长度编码: 经常出现的字符使用比较短的编码, 不经常出现的字符则相反. 这样传输效率会高一点.
4. 前缀编码: 没有任何字符的编码是另一个字符编码的前缀. 不产生歧义, 理论上都得这样搞.

在这里, 前缀编码是非常重要的, 哈夫曼编码就是其中一种. 我们说, 前缀编码可以表达为一棵树, 其中一个 leaf 就是一个字符, 然后, 这个字符对应的一个 path 就是这个字符对应的编码. 能够看出, 这个情况下, 没有任何字符的编码是另一个字符编码的前缀.

我们可以简单证明一下, 如果说存在一个编码是另一个编码的前缀, 那么这个编码对应的 path 是另一个 path 的子集, 这说明那个节点并不是 leaf. 毕竟是子集嘛. 那么产生矛盾.

4.2 哈夫曼编码

反正我们使用的是前缀编码, 每一个字母的编码不允许是另一个字母编码的前缀, 如果说有前缀, 不等长编码就没什么意义了. 哈夫曼编码就是这样的一个编码.

另一方面, 前缀编码实际上可以表示为二叉树, 每一个叶子节点就是一个编码, 从根节点到叶子的路径对应着相应的编码.

我们计算代价: C 是字母, $\forall c \in C$, $p(c)$ 是 c 出现的频率, $d(c)$ 是其深度, 对应的是其编码的长度, 代价的计算公式是

$$B(T) = \sum_{c \in C} p(c) d(c)$$

4.3 问题描述

输入: 字母表 C, 以及对应的概率的表
输出: 具有最小 $B(T)$ 的前缀编码树, 即 C 的哈夫曼编码树

4.4 算法

我们之前已经学过了, 但是还没有验证其正确性, 下面是一个使用了 heap 的算法, 非常简洁 (伪代码写得很难看, 所以还是用了点 c, 不影响观看)

```
1  int main (){
2      int n = |C|;
3      init (Q, C); // initialize Q with C , Q is a heap
4      for (int i = 1 ; i <= n-1 ; i++){
5          z = AllocateNode (); // z 是一个节点.
```

```
        left(z) = Extract-Min(Q); // 取出 Q 中最小的
        right(z)= Extract-Min(Q); // 同上
        x = left(z); // 将 z 的左边赋给 x
        y = right(z);
        p(z) = p(x) + p(y); // 给出 z 的概率.
        insert (Q, z); // 将 z 插进 Q
    }
    return Q
}
```

来看一下这个代码干了啥, 首先将字母表 C 塞进了 heap Q 中, 关键字是其概率. 然后我们自底向上的求解, 拿出两个最小的, 合并一下, 塞回去. 循环 $n-1$ 次即可[2]. 实际上有点小问题, 因为到最后 Q 就只剩下一个点, return 的只是最终的代价. [3] 所以我们要将每次合并的操作进行一个记录.

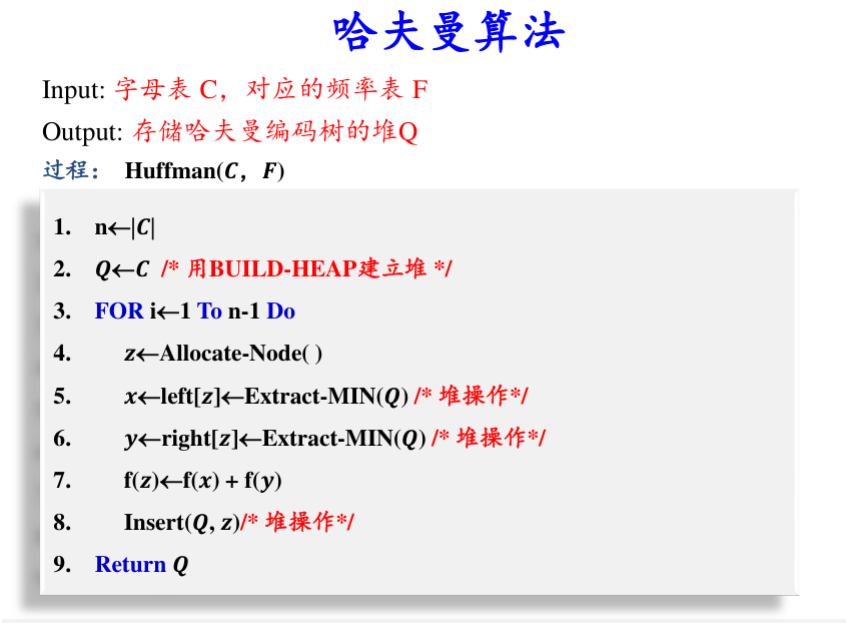

图 4: 哈夫曼编码之伪代码

喂, 怎么记录啊, 你咋没说.

[2]这是因为每次合并是 $2 \to 1$, 所以说执行了 $n-1$ 次之后就只剩下一个节点了

[3]我这里其实想错了, return 的确实是完整的哈夫曼编码树.

4.5 算法复杂度分析

复杂性分析

- 设Q由一个堆实现
- 第2步用堆排序的BUILD-HEAP实现: $O(n)$
- 每个堆操作要求$O(\lg n)$，循环$n-1$次: $O(n\lg n)$
- $T(n) = O(n) + O(n\lg n) = O(n\lg n)$

关于堆的相关知识在《算法导论》第6章

图 5: 哈夫曼编码复杂性分析

4.6 正确性证明

Lemma 4 (子结构). C 的一个优化前缀树 T 的子树 T' 也是一个优化前缀树

证明. 反证法, 这和前面的类似, 如果说 T' 不是最优的, 就说明存在另一个 T'' 代价更低, 于是就可以构造一个比 T 更优的树 □

Lemma 5 (不知道叫什么). 设 T 是 C 的一个优化前缀树, x, y 是两个相邻的叶子, z 记作其父节点, 将 x, y 视为一个点 z , 并且 $p(z) = p(x) + p(y)$

则 $T' = T - \{x, y\}$ 是 C' 的优化前缀树. 其中 $C' = C - \{x, y\} + \{z\}$

证明. 明显有: $B(T) = B(T') + p(x) + p(y)$

考虑 $B(T) - B(T')$ 来证明.

而后使用反证法, 如果说存在 T'' 是 C' 的优化前缀树, 比 T' 更优, 那么 $T'' + \{x, y\}$ 就..., 矛盾. □

Lemma 6 (贪心选择性). 设 x, y 是 C 中具有最小频率的两个字母, 则存在编码树使得 x, y 编码具有相同长度, 并且仅最后一位不同, *i.e.* 是相邻的两个叶子

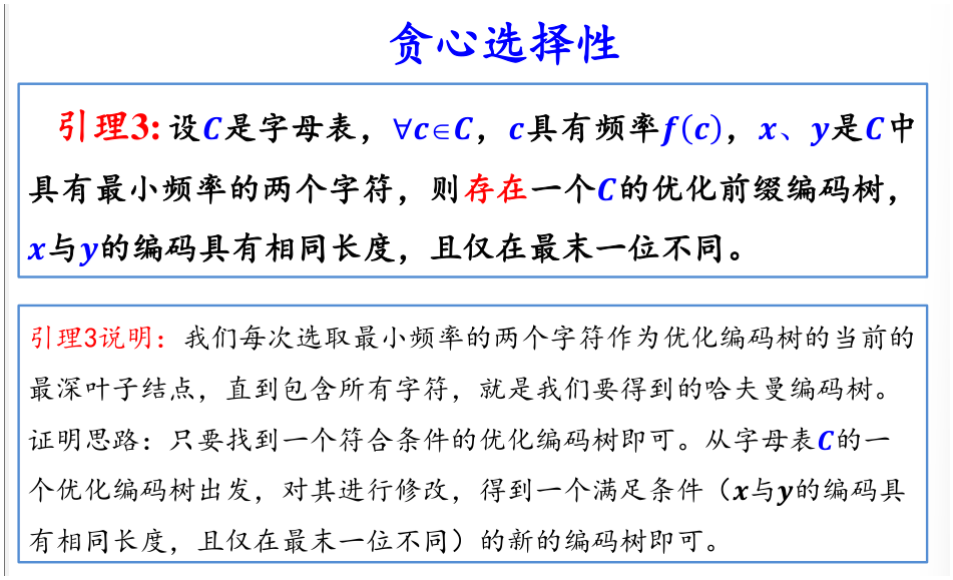

图 6: 贪心选择性

证明. 对于一个编码树, 设其不满足 x, y 相邻, 我们交换一下叶子节点, 构造出一个新的编码树, 并且是最优的即可.

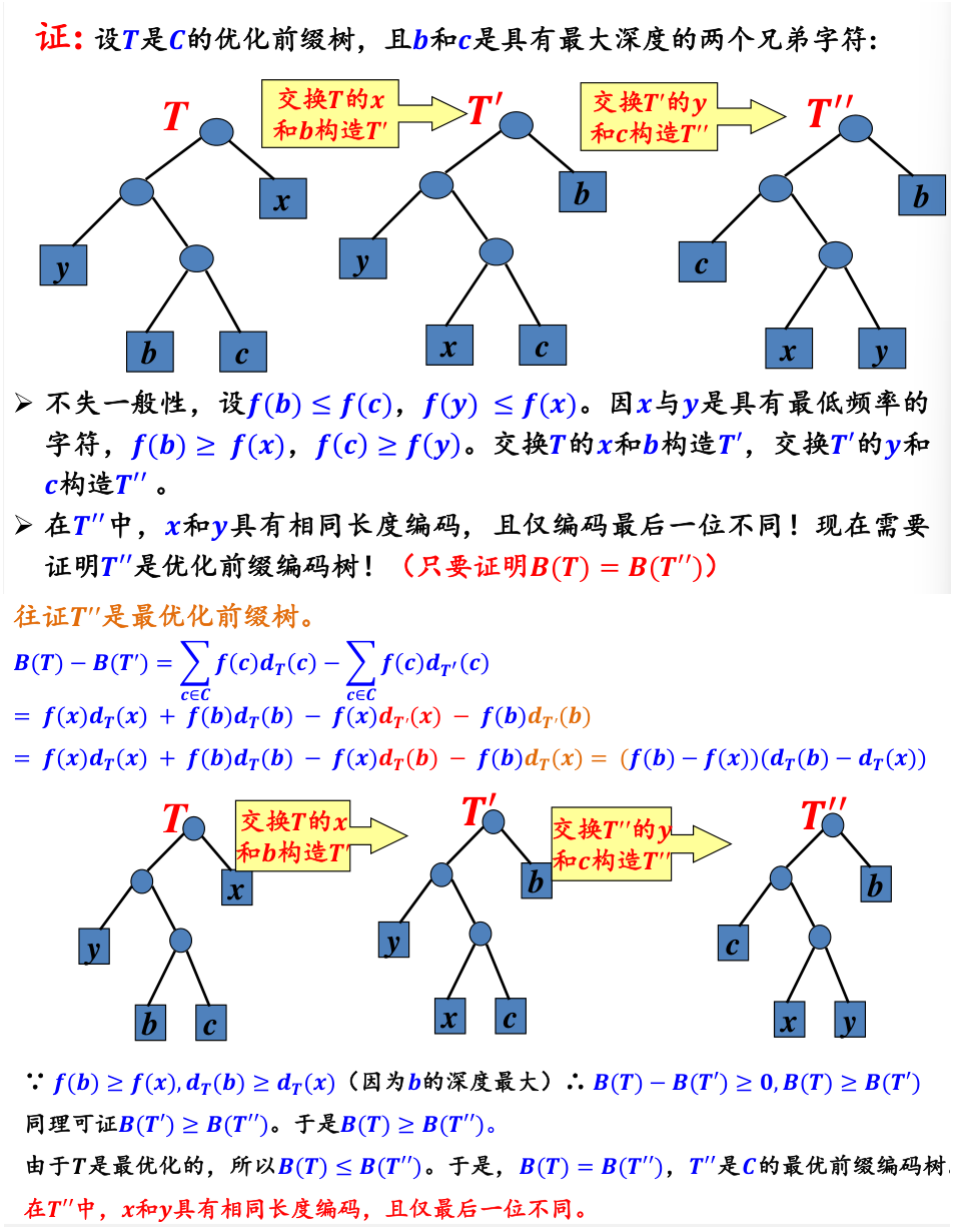

图 7: 解

说实话, 这个非常明显, 不需要使用这么复杂的证明.

因为在底层的计数次数就越多, 既然 $p(x)$ 是最小的, 那么调换顺序之后, 权重变化值就是

$$\text{层数之差} \times (p(x) - p(a)) \leq 0$$

就是

$$B(T'') \leq B(T)$$

然后既然 $B(T)$ 是最优的, 那么 $B(T'') = B(T)$, T'' 也是最优的. □

那么引理 6 为什么是证明了贪心选择性呢? 这是因为它说明

$$B(T) = B(T') + p(x) + p(y)$$

并且这两个 x, y 是能够知道的.

5 最小生成树

问题引入, 比如说我们要修公路, 要使用在最短的公路将所有的站点链接起来. 在互联网上也有应用.

5.1 问题描述

Definition 3 (图的定义)**.** G 是一个图, 其中 V 是顶点集, E 是边集, 一个边写为 (u,v), $u,v \in V$, 边可以有序, 也可以无序.

我们这里无序, 因为我们只涉及无向图 (目前).

Definition 4 (生成树)**.** G 的生成树记为 T, 为了某种意义上的严谨, 我们将 T 写为 (V_t, E_t), 符号用得并不是很熟练. 其中 $V_t = V$. 并且 T 确实是一棵树.

正[4]权重的无向简单图: 我们可以为边赋上权值. 总之我们这里只涉及正权值的图.

简单图: 如果说两个顶点之间的边小于等于 1 条, 则称这个图是简单图 (应该把, 定义有点忘了)

我们现在问题就是:

Input：给定一个图 $G = (V, E)$

Output：给出这个图的最小生成树, i.e. 权重和最小的生成树.

5.2 graph cut

Definition 5 (graph cut)**.** $S \subset V$, 那么 $\{S, V - S\}$ 是 V 的一个划分. 这就是一个图割

Definition 6 (三个概念)**.** (u,v) 是一个横跨边, 如果 $u \in S, v \in V - S$

A 是边集的子集, $A \subset E$, S 尊重 A, 如果 $\forall e \in A \implies e$ 不是横跨边.

e 是轻量型边, 如果 e 是横跨边中权值最小的那个.

Lemma 7. $E' \subset E$, 如果存在一个 cut 尊重 E', 如果

1. 存在包含了 E' 的最小生成树.
2. (u,v) 是轻量型边.

则有: 存在包含了 $E' \cup \{(u,v)\}$ 的最小生成树.

证明. 设已知有一个树 T' 包含了这个 E'. 我们断言, T' 中仅包含了一个横跨边. 这是因为, 如果说不包含横跨边, 那么边的数量为 $|V| - 2$ 不可能是树. 又如果说, 有两个横跨边, 则边的数量为 $|V|$ 一定存在圈, 则不是树.

我们能够构造一个最小生成树, 只需要, 将这个横跨边删去, 再加上轻量型边.

明显有: $B(T'') \leq B(T')$ 因为是轻量型边嘛, 但同时 T' 是最小生成树. 所以说 T'' 也是最小生成树. □

5.3 Prim 算法

5.3.1 图解过程

鸽了

[4]这为什么要求是正的?

5.3.2 伪代码

初始化相当于加入了初始节点 r, 然后继续加边.

这边第 1-3 都是一个初始化. key 向量储存的是某节点 v 的轻量型边的权重. (如果 v 已经在树里面了该怎么办?)

π 储存 v 的父节点, 我们就是根据这个构造解.

第 5 行没看懂.

6-7 行也没看懂, 这是在干什么? 为什么这个循环能够找出轻量型边的端点?

我这个 key 是用来干什么的? 是用在 `Extract-Min` 这个函数上吗?

Prim算法

```
MST-Prim(G, W, r)
Input: 无向连通图G = (V,E), 权值函数W, 初始结点r
Output: G的最小生成树
1.  FOR ∀v ∈ V DO
2.      key[v] ← ∞; /*保存结点v和树中结点的所有边中最小边的权重, 即
    轻量级边的权重*/
3.      π(v) ← null; /*结点v在树中的父结点*/
4.   key[r] ← 0;      /*保证结点r第一个被处理*/
5.   Q ← V[G];       /*最小优先队列*/   《算法导论》90页有关
6.  WHILE Q ≠ ∅ DO                      于优先队列的解释
7.      u ← Extract-Min(Q);/*找到Q中横跨割 (V-Q, Q) 的轻量级边的端点*/
8.  FOR ∀v ∈ Adj[u] DO  /*更新与u邻接的, 但不在树中结点v的key及
                                        其父结点*/
9.          IF v ∈ Q and w(u,v) < key[v] Then
10.             π(v) ← u;
11.             key[v]← w(u,v); /*更新信息*/
12. Return A={(v,π(v))| v ∈ V - r}.
```

图 8: Prim 算法

这算法是真几把牛逼. 看不懂

5.3.3 正确性证明

使用归纳法.

1. 基本情况: 增加 0 条边的时候, 边集为空, 确实是最小生成树的一部分.
2. 归纳步骤: 对于 S 已经是最小生成树的一部分, 那么我们加一条边 (轻量型边) 也是最小生成树的一部分.
3. 结论: 每次得到的边集都是最小生成树的一部分, 当所有的点都加进去了, 得到的是最小生成树.

5.4 Kruskal 算法

加入轻量型边. 也称为 “加边法” (佛了, 难道 prim 就不加边了吗). 我们先是将单独一个顶点作为一棵树, 形成一个森林.

划分是森林中的每棵树, 依然是找出轻量型边, 然后合并两颗树. 然后继续加边.

但在实现过程中, 会有更新某某数组的情况出现.

5.4.1 伪代码

废话不多说, 直接来看伪代码

注: $S \subset E$, `union` 是将 u, v 所在的树连在一起, 合并为一棵树.

Kruskal算法

```
MST-Kruskal(G, W)
Input: 无向连通图G = (V,E), 权值函数W
Output: G的最小生成树
1. S = ∅; /*初始化为空树*/
2. FOR ∀v ∈ V DO
3.     Make-Set(V);      /*创建|V|棵树*/
4. 按照W权值的非递减顺序排序E;
5. FOR ∀(u,v) ∈ E (按照W权值的非递减顺序) DO
6.     IF Find-Set(u) ≠ Find-Set(v)
              /*检查节点u,v是否属于同一棵树 */
7.         S = SU{(u,v)}; /*边(u,v)加入到集合S*/
8.         Union(u,v);/*对节点u,v所属的子树进行合并*/
9.  Return S.
```

图 9: Kruskal 算法之伪代码

这里我们能够看到为什么说 `Kruskal` 是加边法了. 输出都是边, 不需要特殊的存储信息的过程.

5.4.2 复杂度分析

- 令 $|V| = n, |E| = m$
- 4 需要时间 $O(m \log m)$
- 2-3 需要执行 $O(n)$ 个 `Make-set` 操作.
- 5-8 需要 $O(m)$ 个 `Find-Set` 和 `Union` 操作. 需要时间 $O((n+m)\alpha(n))$ [5]
- $m \geq n-1$, $\alpha(n) = \log n = \log m$ [6]
- 总的时间复杂度就是 $O(m \log m)$

[5] 为什么??

[6] 这又是什么?

Kruskal算法

```
MST-Kruskal(G, W)
Input: 无向连通图G = (V,E), 权值函数W
Output: G的最小生成树
1. S = ∅; /*初始化为空树*/
2. FOR ∀v ∈ V DO
3.     Make-Set(V);      /*创建|V|棵树*/
4. 按照W权值的非递减顺序排序E;
5. FOR ∀(u,v) ∈ E (按照W权值的非递减顺序) DO
6.     IF Find-Set(u) ≠ Find-Set(v)
              /*检查节点u,v是否属于同一棵树 */
7.         S = S∪{(u,v)}; /*边(u,v)加入到集合S*/
8.         Union(u,v);/*对节点u,v所属的子树进行合并*/
9.  Return S.
```

图 10: Kruskal 算法之伪代码