清华大学软件学院
算法分析与设计基础
2020 年春季学期

作业 **10**

吴佳龙　班级：软件 83　学号：2018013418　May 30, 2020

10.1. (CLRS Exercises 35.3-2)

证明. 集合覆盖问题的判定版本是 $L = \{(X, F, k)\}$，其中 F 是 X 的子集族，且存在一个 F 的大小为 k 的子集是 X 的覆盖。

现将顶点覆盖问题 (G, k) 规约到集合覆盖问题，从而证明集合覆盖问题是 NPC 的。

定义映射 $f : (G, k) \mapsto (X, F, k)$，其中 $X = E$，$F = \{E_v \mid v \in V, E_v$ 是v 的邻边的集合$\}$。若 G 存在一个大小为 k 的顶点覆盖 $V' \subset V$，则 $\{E_v \mid v \in V'\} \subset F$ 是 X 的一个覆盖且大小也为 k，这是因为 $\forall e \in X$，e 在 G 中被 v 覆盖，则在 X 中被 E_v 覆盖；反过来，若 $\{E_v \mid v \in V'\}$ 是 X 的覆盖，则 V' 也是 G 的一个顶点覆盖。

□

10.2. (CLRS Exercises 35.3-3)

算法伪代码如下：

```
GREEDY-SET-COVER(X,F):
    n = |X|
    m = max |S| in F
    init bucket[0..m] to empty list
    init chosen[S in F] to be false
    init super[x in X] to be empty list
    init covered[x in X] to be false
    for S in F:
        insert S into bucket[|S|]
        for x in S:
            insert S into super[x]
    i=m
    while i>0:
        if bucket[i] is empty:
            i = i-1
        pop arbitrary S from bucket[i]
        chosen[S] = true
        for x in S:
            if not covered[x]:
                covered[x] = true
                for s in super[x]:
                   if not chosen[s]:
                       move s from bucket[k] to bucket[k-1]
    return {S : chosen[S] = true}
```

算法说明：每个子集 S 被放入桶 k 中，说明该子集与当前未被覆盖的元素 U 的交集大小为 k。而每个时刻，子集中未被覆盖的元素个数的最大值是不降的，因此在算法中通过 i 来枚举。找出一个未被覆盖元素最多的子集 S 后，需要枚举他的未被覆盖的元素 x，并且更新仍在桶中包含 x 的那些子集。

时间复杂度分析：7-10 行预处理的复杂度为 $O(\sum_{S\in F}|S|)$。第 17-18 行的枚举的总复杂度也是 $O(\sum_{S\in F}|S|)$。第 19-22 行，对于每个 x 只可能执行一次，而执行一次的复杂度是 $O(\sum_{S \text{ s.t. } x\in S} 1)$，总复杂度也是

$$O(\sum_{x}\sum_{S \text{ s.t. } x\in S} 1) = O(\sum_{S}\sum_{x\in S} 1 = O(\sum_{S}|S|)$$

因此算法的总复杂度是 $O(\sum_{S\in F}|S|)$。

10.3. (CLRS Exercises 35.5-5)

算法修改：对于每一个 L_i 中的元素 s，同时维护一个集合表示一个 S 的子集，它的和恰好就是 s。在 MERGE-LISTS 的过程中，当 L_{i-1} 中的每个元素加了 x_i，就将每个元素对应的子集并上 x_i；在 TRIM 以及删除大于 t 的过程中，如果 L_i 中的某个元素被丢弃了，同样丢弃掉与之伴随的子集。最终 z^* 对应的子集就是和为 z^* 的子集。

时间复杂度：修改后的算法即使采用最朴素的实现来维护子集，也仅会增加 n 的复杂度，复杂度为 $O(\frac{n^3}{\varepsilon})$，仍为完全多项式时间。