算法分析与设计基础 第十五周作业

徐浩博 软件02 2020010108

Problem 1

算法如下：

辅助数/数组/链表：max_num：当前剩余节点数最多的集合的剩余节点数；node_list[v]：包含点v的所有set的链表；left_node_num[S_i]：S_i的剩余节点数；left_num_list[i]：剩余节点数为i的所有set的链表.

```
for each S_i in F:
    for each v in S_i:
        node_list[v].insert(S_i)
    left_node_num[S_i] = |S_i|
    left_num_list[S_i].insert(S_i)
    max_num = max(max_num, |S_i|)

while max_num > 0:
    while true:
        if left_num_list[max_num] is empty:
            break
        head = left_num_list[max_num].head
        if left_node_num[head] != max_num:
            left_num_list[head].insert(head)
        else:
            for each v in head:
                for s in node_list[v]:
                    left_node_num[s] -= 1
                clear node_list[v]
        delete head from left_num_list[max_num]
    max_num -= 1
```

大概思路是：创建一个剩余节点数分别为1-max_num这max_num个链表，每个链表i内放置剩余节点数i的set；每次从max_num中拿出一个头结点set，如果该set内剩余节点数已经变化，则将其插入对应的剩余节点数链表里，否则就要将该set内的点全部删去，同时更新各个其他set剩余节点数（注意，不是在剩余节点数链表中直接更新，而是用一个数组储存当前set剩余节点数）.

该算法是$O(\sum_{S\in F}|S|)$的，原因是我们剩余节点数链表，每次对头结点进行操作：如果set内剩余节点数已经变化，那么操作复杂度是O(1)的，我们将这个复杂度归结到使set内剩余节点数变化的原因上（节点被删去）；如果set内剩余节点仍是max_num，那么就要删去该set内全部剩余节点，删除每个节点时需要更新所有包含该节点的set，每次更新O(1)，这里可以将使set剩余节点数变化，头结点到达该set时需要O(1)更新的代价也加进来，一共也是O(1)的. 因此，总代价可以看成每个点在所有包含该节点的set内进行的O(1)操作，是$O(1)\times\sum_{S\in F}|S| = O(\sum_{S\in F}|S|)$的.

Problem 2

```
1  APPROX_SUBSET_SUM(S, t, ε):
2      n = |S|
3      L_0 = <0>
4      for i = 1 to n:
5          for each (e, set) in L_{i-1}:
6              new_e = e + x_i
7              pre[new_e] = x_i
8              L_i.insert(new_e)
9          L_i = TRIM(L_i, ε/2n)
10         remove from L_i every element that is greater than t
11     let z* be the largest value in L_n
12
13     S = ∅
14     while z* > 0:
15         S.insert(pre[z*])
16         z* = z* - pre[z*]
17     return S
```

即我们记录L_i内每个新产生元素的前驱x_i，最后的集合如13-17行，通过前驱获知每一步加入的数. 显然，并没有改变原有程序的复杂度.