Week 14

软73 沈冠霖 2017013569

May 27, 2019

1 T1

先把集合覆盖问题化简为如下形式：给定集合S和若干个子集$S_1, S_2....S_m$和常数k，是否存在k个子集，使其并为S。
1.如果给定了这k个子集，则只需要把它们并起来就可以验证了，时间复杂度O(n)。因此集合覆盖是NP问题。
2.给定一个顶点覆盖问题的图G和值k，可以找到一个函数$f(G,k) = (S, S_1, S_2....S_m, k')$（以下把集合覆盖问题简写为（S,k）），使得k'=k,S为所有边构成的集合，子集$S_1, S_2...S_m$分别对应图G的顶点1到m关联的边集。这个函数能把任意的顶点覆盖问题映射为集合覆盖问题。
3.如果顶点覆盖问题(G,k)成立，那么可以找到k个顶点，使得每条边的起点或终点都在这k个顶点中找到。那么对应的集合覆盖问题，集合中每个元素都可以在这k个顶点对应的集合中找到，对应的集合覆盖问题f(G,k)=(S,k)也成立。反之，如果集合覆盖问题(S,k)=f(G,k)成立，那么必定存在k个集合，使得全集S中任何一个元素都能在这k个子集中被找到，那么对于顶点覆盖问题(G,k)，任何一个边都能在这k个子集对应的顶点的边集中找到，也就是这k个顶点覆盖了整个图，顶点覆盖问题(G,k)也成立。
4.算法只需要复制k，同时统计每条边，把每条边加入全集和对应的点子集就可以了，时间复杂度$\theta(E)$，是多项式算法。

2 T2

算法 1: 遍历每个子集，来初始化如下数据结构：R[size]:每个元素R[i]是一个链表，存储着剩余元素（未被覆盖元素）为i个的集合，size为子集中元素最多数目。R[i]的元素是集合S，S.key是这个集合的剩余元素个数。A[n]:每个元素是一个链表，存储着每个元素位于哪些集合。visit[n]:存储每个元素是否被覆盖。ans：存储被覆盖的元素个数。max:存储当前子集剩余元素最多是几。
2：之后进行k次循环，步骤如下：
2.1：提取并删除R[max]的表尾集合S_{max}（一会要扩增这个集合），将ans加上max，如果ans=n了就结束，如果R[max]空了就更新max。
2.2：遍历集合S_{max}的每个元素a_i，如果$visit[a_i] = 0$，就更新visit，同时遍历$A[a_i]$，把得到的每个集合，也就是用到这个元素的每个集合的key都-1，从R表的对应位置R[key]删除，并且加入R[key-1]的尾。

证明 首先，每次提取的都一定是剩余元素最多（可以扩增元素最多）的集合，算法正确。
其次，分析其复杂度。初始化需要遍历每个子集的所有元素，复杂度是$O(\Sigma|S|)$。考虑所有循环结束后的所有操作数。首先，提取最大的集合提取了k次，每次都是O(1)，因此总共是O(k)。其次，max从size减小到了0，更新max的复杂度是O(size)。最后，每次遍历需要遍历每个扩增元素的所有对应子集，整套遍历之后相当于遍历了每个子集的每个元素各一次，遍历了$O(\Sigma|S|)$次。而每遍历每个元素对应的子集，操作链表的复杂度是O(1)。因此遍历和更新复杂度是$O(\Sigma|S|)$。总复杂度是$O(\Sigma|S| + k + size) = O(\Sigma|S|)$。