Set Cover Problem | Set 1 (Greedy Approximate Algorithm)

Given a universe U of n elements, a collection of subsets of U say S = {S_1, S_2…,S_m} where every subset S_i has an associated cost. Find a minimum cost subcollection of S that covers all elements of U.

Example:

```
   U = {1,2,3,4,5}
   S = {S1,S2,S3}

   S1 = {4,1,3},    Cost(S1) = 5
   S2 = {2,5},      Cost(S2) = 10
   S3 = {1,4,3,2},  Cost(S3) = 3

Output: Minimum cost of set cover is 13 and
        set cover is {S2, S3}

There are two possible set covers {S1, S2} with cost 15
and {S2, S3} with cost 13.
```

Why is it useful?
It was one of Karp's NP-complete problems, shown to be so in 1972. Other applications: edge covering, vertex cover
Interesting example: IBM finds computer viruses (wikipedia)
Elements- 5000 known viruses
Sets- 9000 substrings of 20 or more consecutive bytes from viruses, not found in 'good' code.
A set cover of 180 was found. It suffices to search for these 180 substrings to verify the existence of known computer viruses.

Another example: Consider General Motors needs to buy a certain amount of varied supplies and there are suppliers that offer various deals for different combinations of materials (Supplier A: 2 tons of steel + 500 tiles for $x; Supplier B: 1 ton of steel + 2000 tiles for $y; etc.). You could use set covering to find the best way to get all the materials while minimizing cost
Source: http://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf

Set Cover is NP-Hard:
There is no polynomial time solution available for this problem as the problem is a known NP-Hard problem. There is a polynomial time Greedy approximate algorithm, the greedy algorithm provides a Logn approximate algorithm.

2-Approximate Greedy Algorithm:

Let U be the universe of elements, {S_1, S_2, … S_m} be collection of subsets of U and Cost(S_1), C(S_2), … Cost(S_m) be costs of subsets.

```
1) Let I represents set of elements included so far.  Initialize I = {}

2) Do following while I is not same as U.
    a) Find the set Si in {S1, S2, ... Sm} whose cost effectiveness is
       smallest, i.e., the ratio of cost C(Si) and number of newly added
       elements is minimum.
       Basically we pick the set for which following value is minimum.
           Cost(Si) / |Si - I|
    b) Add elements of above picked Si to I, i.e.,  I = I U Si
```

Example:

Let us consider the above example to understand Greedy Algorithm.

First Iteration:

I = {}

The per new element cost for S_1 = Cost(S_1)/|S_1 – I| = 5/3

The per new element cost for S_2 = Cost(S_2)/|S_2 – I| = 10/2

The per new element cost for S_3 = Cost(S_3)/|S_3 – I| = 3/4

Since S_3 has minimum value S_3 is added, I becomes {1,4,3,2}.

Second Iteration:

I = {1,4,3,2}

The per new element cost for S_1 = Cost(S_1)/|S_1 – I| = 5/0
Note that S_1 doesn't add any new element to I.

The per new element cost for S_2 = Cost(S_2)/|S_2 – I| = 10/1
Note that S_2 adds only 5 to I.

The greedy algorithm provides the optimal solution for above example, but it may not provide optimal solution all the time. Consider the following example.

```
S1 = {1, 2}
S2 = {2, 3, 4, 5}
S3 = {6, 7, 8, 9, 10, 11, 12, 13}
S4 = {1, 3, 5, 7, 9, 11, 13}
S5 = {2, 4, 6, 8, 10, 12, 13}

Let the cost of every set be same.

The greedy algorithm produces result as {S3, S2, S1}

The optimal solution is {S4, S5}
```

Proof that the above greedy algorithm is Logn approximate.

Let OPT be the cost of optimal solution. Say (k-1) elements are covered before an iteration of above greedy algorithm. The cost of the k'th element <= OPT / (n-k+1) (Note that cost of an element is evaluated by cost of its set divided by number of elements added by its set). How did we get this result? Since k'th element is not covered yet, there is a S_i that has not been covered before the current step of greedy algorithm and it is there in OPT. Since greedy algorithm picks the most cost effective S_i, per-element-cost in the picked set must be smaller than OPT divided by remaining elements. Therefore cost of k'th element <= OPT/|U-I| (Note that U-I is set of not yet covered elements in Greedy Algorithm). The value of |U-I| is n - (k-1) which is n-k+1.

```
Cost of Greedy Algorithm = Sum of costs of n elements
                         [putting k = 1, 2..n in above formula]
                          <= (OPT/n + OPT(n-1) + ... + OPT/n)
                          <= OPT(1 + 1/2 + ..... 1/n)
                         [Since 1 + 1/2 + .. 1/n ≈ Log n]
                          <= OPT * Logn
```

Source:
http://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf

This article is contributed by **Harshit**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

4 Comments Category: Greedy Tags: Greedy Algorithm , NPHard

Related Posts:

- Job Sequencing Problem | Set 2 (Using Disjoint Set)
- Find smallest number with given number of digits and sum of digits
- Minimize the maximum difference between the heights
- Dial's Algorithm (Optimized Dijkstra for small range weights)
- Fractional Knapsack Problem
- Bin Packing Problem (Minimize number of used Bins)
- Find minimum time to finish all jobs with given constraints
- Shortest Superstring Problem