Set Cover

Sometimes, greedy strategy doesn't work as the local optimum doesn't align with the global optimum. However, in some problems such as Set Cover, which is NP-hard, we still use the greedy approach as it gives a good approximation of an answer.

Using reductions to solve other questions

We can sometimes solve problems that are similar to other greedy problems by reducing them to that greedy problem. An example of that is the Maximum spanning tree, where we need to find the maximum cost of any tree that can span the graph. To reduce the maximum spanning tree to the minimum spanning tree problem, we can just multiply all the edge weights by -1, and find the MST of such a graph. The cost we get is again multiplied by -1, and we get the maximum cost.

Set Cover

Input is a set of elements B, and a family of subsets $S_1, S_2, \ldots, S_m \subseteq B$

Output; A selection of the S_i whose union is B.

Cost: Number of sets picked

Greedy Strategy

Repeat until all elements of B are covered

Pick the set S_i with the largest number of uncovered elements

Example

$S =$ {arid, dash, drain, heard, lost, nose, shun, slate, snare, thread, lid, roast}

Using each word as a set of letters, we must pick the minimum number of words to cover the set

$B = \{a, d, e, h, i, l, n, o, r, s, t, u\}$

First pick

We see that the word **length** covers the most uncovered elements

After picking the word **length,** the uncovered elements left are $\{i, l, n, o, s, u\}$

Second pick

We pick the word **lost**

We are left with uncovered elements $\{i, n, u\}$

Third pick

We pick the word **drain**

You're left with the sole element $\{u\}$

Fourth pick

You pick the word **shun** to cover the last uncovered element left

In this example, we find that greedy gives us the optimum answer which is 4, however, it's just by sheer luck, and it's not always the case we'll get the correct answer.

We can see that the **optimum substructure property** is held in this problem, However, we do not have the **greedy choice property.** If we did have a way to pick a local optimum that was aligned with the global optimum, the greedy approach would work. However, this is not the case.

Counter-example to Greedy approach

We have to cover set $B = \{1, 2, 3, 4, 5, 6\}$

we have subsets $S = \{\{1, 2, 3, 4\}, \{1, 3, 5\}, \{2, 4, 6\}\}$

If we used the greedy approach, we would pick the first set, which would force us to pick the other two sets as well.

However, it's clearly evident that the optimum solution is picking the 2nd and 3rd set. Therefore, we've proven that greedy approach is not optimum

Approximation

If we have a set B of n elements, and the optimal cover consists of k sets. Then the greedy algorithm will use at most $k \cdot \ln n$ sets.

Proof of approximation

Let n_t be the number of elements left uncovered after the t^{th} iteration.

We know that all elements are left uncovered before starting, therefore $n_0 = n$

We know we are done with iterations when $n_t = 0$, and t will be the number of sets we used in our answer

Say at the t^{th} iteration, we have n_t elements left, we know for a fact that the optimal answer k can definitely cover the elements left. Hence, there is a set that has to cover at least n_t/k elements by pigeonhole principle.

Hence, if we pick that element, the upper bound on the number of uncovered elements left in the $(t+1)^{th}$ iteration is given by

$$n_{t+1} \leq n_t - \frac{n_t}{k} = n_t \cdot \left(1 - \frac{1}{k}\right)$$

Therefore, for the general case n_t, the upper bound is given by

$$n_t \leq n_0(1 - 1/k)^t$$

We know that $1 - x \leq e^{-x}$, equality only occurs when $x = 0$

Hence, we can rewrite the upper bound for n_t as

$$n_t \leq n_0 \cdot \left(1 - \frac{1}{k}\right)^t < n_0(e^{-1/k})^t = ne^{-t/k}$$

When we plug $t = k \ln n$, then we get

$$n_t < ne^{-t/k} = ne^{-(k \ln n)/k} = ne^{-\ln n} = ne^{\log_e \frac{1}{n}} = n \cdot \frac{1}{n} = 1$$

Hence, after t iterations, we get $n_t < 1$, and we know that n_t has to be a non-negative integer, it implies that n_t is 0, which is our terminating condition

Which means that after $k \ln n$ iterations, we are guaranteed to have covered all the sets, where k is the optimal number of solutions.