CMPSC 465 Data Structures & Algorithms
Fall 2024 Mingfu Shao and Debarati Das

WS 12

November 11

1. Recall the set cover problem:

   Input: A set of elements B and sets $S_1, \dots, S_m \subseteq B$.

   Output: A selection of the S_i whose union is B (i.e. that contain every element of B).

   Cost: Number of sets picked.

   The natural strategy to solve this problem is a greedy approach: At every step, pick the set that covers the most uncovered elements of B. In the book, we proved that this greedy strategy over-estimates the optimal number of sets by a factor of at most $O(\log n)$, where $n = |B|$. In this problem we will prove that this bound is tight.

   Show that for any integer n that is a power of 2, there is an instance of the set cover problem (i.e. a collection of sets $S_1, \dots, S_m$) with the following properties:

   (a) There are n elements in the base set B.

   (b) The optimal cover uses just two sets.

   (c) The greedy algorithm picks at least $O(\log n)$ sets.

   Solution:

   Consider the base set $U = \{1, 2, 3, \dots, 2^k\}$ for some $k \geq 2$. Let $T_1 = \{1, 3, 5, \dots, 2^k - 1\}$ and $T_2 = \{2, 4, 6, \dots, 2^k\}$. These two sets comprise an optimal cover. We add sets $S_1, \dots, S_{k-1}$ by defining $l_i = 2 + \sum_{j=1}^{i} 2^{k-j}$ (take $l_0 = 0$) and letting $S_i = \{\ell_{i-1} + 1, \dots, \ell_i\}$. We think of S_i as covering a tiny bit more than a $1/2^i$-fraction of the universe, and in particular S_1 is larger than both T_1 and T_2.

   Since S_1 contains $2^{k-1} + 2$ elements, it will be picked first. After the algorithm has picked $\{S_1, S_2, \dots, S_i\}$, each of T_1 and T_2 covers $2^{k-i-1} - 1$ new elements while S_{i+1} covers 2^{k-i-1} new elements. Hence, the algorithm picks the cover $S_1, \dots, S_{k-1}$ containing $k - 1 = \log n - 1$ sets.

   An example of the above algorithm is with, say, $n = 64$. Let $T_1 = \{1, 3, \dots, 63\}$, and $T_2 = \{2, 4, \dots, 64\}$. Let picking T_1 and T_2 be the optimal answer. Now let $S_1 = \{1, 2, \dots, 32, 33, 34\}$, or 2 elements more than $64/2$. Let S_2 contains the next $64/4 = 16$ elements, let S_3 contain the next $64/8 = 8$ elements, S_4 contain the next $64/16 = 4$ elements, and S_5 contain the remaining 2 elements. This works because $64 = 32 + 16 + 8 + 4 + 2 + 2$. We are just regrouping the terms as follows $64 = (32 + 2) + 16 + 8 + 4 + 2$ and creating sets out of them.

   Our greedy algorithm will pick S_1, S_2, S_3, S_4, S_5 instead of T_1, T_2. Our greedy algorithm ended up picking $\log(n) - 1 = k - 1$ sets, since $k = 6$ for $n = 64$. The first two paragraphs of the solutions explain how this is generalized for all k.

2. Consider the following inventory problem. Here a company sells cars, and the predictions tell the quantity of sales to expect over the next n months. Let d_i denote the number of expected sales in month i. We assume that all sales happen at the beginning of the month and cars that are not sold are stored until the beginning of the next month. You can store at most S cars and the cost of storing a

single car for a month is C. The company receives shipments of cars by placing order for them and there is a fixed ordering fee K to place each order (regardless of the number of cars ordered). The company starts with no cars. The problem is to design an algorithm that decides how to place orders so that all demands $\{d_i\}$s are satisfied while minimizing the total cost. In summary:

(a) There are two parts of the cost
   - Storage. It costs C for every car every month.
   - Ordering fee. It costs K for every order placed.

(b) In each month the company needs enough cars to satisfy the demand d_i but the number leftover after satisfying the demand for the month should not exceed the inventory limit S.

Give an algorithm that solves this problem in time that is polynomial in n and S.

Solution:

The subproblems will represent the optimum way to satisfy orders $1, \ldots, i$ with an inventory of s cars left over after the month i. Let $OPT(i,s)$ denote the value of the optimal solution for this subproblem. There are $n \cdot (S+1)$ subproblems to solve as there could be $0, \ldots, S$ cars left over after any month. The problem we eventually wish to solve is $OPT(n,0)$.

Now we solve the subproblem $OPT(i,s)$ assuming the values of previous subproblems (that is, $OPT(i-1,z)$ for every z) are accessible. Note that

- If in the previous month z cars were left over, then we have to pay zC storage fee.
- In order to satisfy the demand of d_i and have s cars left over, we need to order $s+d_i-z$ cars if $s+d_i > z$, which costs K ordering fee.

Putting them together,

$$OPT(i,s) = \min\left(OPT(i-1, s+d_i) + C(s+d_i), \min_{z: z < \min(s+d_i, S)} OPT((i-1), z) + zC + K \right).$$

Hence, we solve this problem by computing all $OPT(i,s)$ in the increasing order of i. There are $O(n \cdot S)$ entries to compute, and provided the values for all the previous entries are already known, each update takes $O(S)$ time, so this is a polynomial-time algorithm.