MAT-72006 Advanced Algorithms and Data Structures
October 6, 2016
HW 5: 11 Hash Tables, 14 Augmenting Data Structures

1.

Consider inserting the keys $10, 22, 31, 4, 15, 28, 17, 88, 59$ into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing and using quadratic probing with $c_1 = 1$ and $c_2 = 3$.

2.

We want to process elements from the universe $U = \{1, \ldots, n\}$. In particular, for our application, we need a data structure that supports

- MINIMUM, INSERT, and SEARCH in $O(1)$ worst-case time, and
- PRINT in time $\Theta(n)$, where PRINT outputs the contents of the data structure in the order they were inserted in.

Describe a data structure D fulfilling the above requirements. We can assume that D will not have to hold multiples of the same value, and that initially D will be empty.

3.

An $n \times n$ square matrix is implemented as a two-dimensional array, i.e., as an array of size n where each cell contains an array of length n. Given an initial such matrix A, we perform a sequence of operations where each operation is either

- reading an entry, i.e, $A[i, j]$,
- writing value k to an entry, i.e., $A[i, j] := k$, or
- a transposition $A := A^T$, i.e., A is transformed into A^T.

All entry accesses are valid, i.e., $1 \leq i \leq j \leq n$. How should the two-dimensional array be augmented such that each operation in the sequence takes $O(1)$ time?

4.

A retailer sells boxes of candy to customers (e.g., stores and supermarkets). Through a careful data analysis, the retailer has observed that customers typically have a hard price limit. For instance, a customer will not buy candy for more than say 1000 €, 2000 €, or in general, not for more than t €. To maximize revenue, the retailer believes it is a good idea to sell ready-made bundles that come as close to t € as possible.

More precisely, the retailer has a collection $\mathcal{C}$ of n boxes. An element of $\mathcal{C}$ is a price p_i, where $1 \leq i \leq n$. Given a target price t, the retailer wants to know whether there is a subcollection $\mathcal{C}'$ of $\mathcal{C}$ of at most n boxes such that their total price matches t precisely, i.e., such that $\sum_{p \in \mathcal{C}'} p = t$.

(a) How many possible subcollections of at most n boxes are there?
(b) Give an algorithm for generating all possible subcollections of at most n boxes. (Hint: binary numbers).

5.

The boxes of candy the retailer of Problem 4 sells very seldom change. That is, the retailer gets resupplied, but the boxes (or more precisely, the prices p_i and n) stay the same. Thus, the retailer wants to quickly answer queries for different values of t (e.g., "is there a subcollection of size at most n that sum to $t = 50$? What about $t = 55$, or $t = 65$?"). In other words, the retailer can afford to spend even a large amount of preprocessing time if queries can then be answered quickly.

Describe an algorithm that uses $2^{n/2}\mathsf{poly}(n)$ time to preprocess $\mathcal{C}$ after which queries of the form "$\exists \mathcal{C}' \subseteq \mathcal{C} : \sum_{p \in \mathcal{C}'} p = t$?" can be answered in $\Theta(\log n)$ time. Here, $\mathsf{poly}(n)$ denotes any polynomial in n.

(Hint: carefully consider the bounds $2^{n/2}$ and $\Theta(\log n)$. What might they hint at? One solution uses a general well-known trick which might be challenging to discover, but quite easy once you see it).

Update and reflection on Problem 5 (6.10.2016)

In the exercises, roughly the following algorithm was suggested for the problem:

1. If $\mathcal{C}$ is of odd size, add a dummy zero element to $\mathcal{C}$. Then, split the input into two parts S and Q of size $n/2$ each.

2. Enumerate all possible $2^{n/2}$ sums of elements arising from S and Q, and denote them by S' and Q', respectively. Using a $\Theta(n \log n)$-time sorting algorithm, sort S' in time $2^{n/2}(n/2)$.

3. For each $q \in Q'$, perform a binary search in S' for the value $(t - q)$ (but if $t - q = 0$, we are done). Since S' has $2^{n/2}$ elements, the binary search takes $\Theta(\log(2^{n/2})) = \Theta(n)$ time.

This approach is viable for the case of a *single* query when done as a part of the preprocessing step. However, it doesn't still quite reach logarithmic time. Moreover, observe that for each of possibly many queries, we perform a scan over Q', which is of size $2^{n/2}$ as well. In other words, we spend time exponential in the input size per query (this is far from logarithmic or linear time per query!). Thus, the problem statement is highly optimistic in asking for $\Theta(\log n)$-time

queries. Many smart people in the exercise session correctly made this observation. To summarize, it is unclear whether there is an algorithm solving the problem within the required time bounds for multiple queries.

Nevertheless, the general design technique this answer showcases is known as *meet-in-the-middle* or *split and list*. More precisely, the idea is to first split the task into two (or more parts), typically of equal size. Then, enumerate all solutions for the small parts. Finally, combine the solutions from the different parts using some efficient algorithm. Typically, the approach is applied to computationally hard problems.