Code Jam 2014 - Round 2

Analysis: Data Packing

Given a list of file sizes, Adam wants to pack these files in the minimum number of compact discs where each disc can fit upto 2 files and a single file cannot be divided into multiple discs. All compact discs have the same capacity X MB.

There are several greedy approaches that can solve this problem. We describe one and show how it can be implemented efficiently. We first sort all file sizes. Then we linearly scan the sorted list going from smallest to biggest. For each file, we try to â€œpairâ€ it with the largest possible file that can both fit in X. If no such largest file is found, we put the file by itself. We finally report the number of discs used.

Letâ€™s analyze the running time for this solution. The sorting take O(N log N), and the linear scan combined with searching for largest possible file take $O(N^2)$. Therefore the running time for this algorithm is $O(N^2)$ which is sufficient to solve the large data set.

We can further optimize this solution by avoiding the $O(N^2)$ computation. To do so, we use a common idea used in programming competitions. First, as before, we sort the file sizes. We then keep track of two pointers (or indices) **A** and **B**: **A** initially points to the smallest index (i.e., index 0), and **B** initially points to the largest index (i.e., index N-1). The idea being that at any instance, the two pointers A and B point to two candidate files that can be paired and placed in one disc. If their combined size fits in X, then we place it in one disc and then â€œadvanceâ€ both pointers simultaneously, i.e. we increase A by 1, and decrease B by 1. If the combined file size does not fit in X, then we repeat the process of decreasing B by 1 -- which means the large disc which was previously at index B will need to be put in a disc by itself -- until the combined file size at A and B fits in X. During this iteration, we should handle the case when A and B point to the same index i.e. ensure we do not double count that file. After a matching file has been found at B, we simultaneously advance A and B as before. We should repeat this process only when B is greater than or equal to A. As for the running time, advancing the largest and smallest pointers takes O(N) time, therefore the running time is dominated by sorting which is O(N log N).

As an aside, we would like to point out that for the first solution sorting isnâ€™t actually necessary. We can instead pick any file, then find the largest file that fits with it in a disc. If no such largest file exists, we can put the file by itself. Intuitively, the reason why it works is because we greedily try to â€œpairâ€ each file with the best possible largest file. This algorithm still has a $O(N^2)$ running time.