CS 124 Programming Assignment 3: Spring 2021

Your name(s) (up to two): Yumi Yi

Collaborators:

No. of late days used on previous psets: 7
No. of late days used after including this pset: 7

I. Dynamic Programming

We first define a recursive description of the problem. If $A = (a_1, a_2, ..., a_n)$ is our sequence of integers, then we can let

$$x = \sum_{k=1}^{i} s_k a_k$$
$$y = \sum_{k=i+1}^{n} s_k a_k$$

(i.e. x is the signed sum of all the integers from a_1 to a_i, and y is the signed sum of all the integers from a_{i+1} to a_n). Then we can define a recursion:

$$f(i, x) = \min(f(i+1, x + a_{i+1}), f(i+1, x - a_{i+1}))$$

where $f(i, x)$ is the minimum difference between x and y. For each a_i, we want to consider both a_i and $-a_i$ to determine the appropriate sign to assign to it, hence why we consider both $f(i+1, x + a_{i+1})$ and $f(i+1, x - a_{i+1})$ and choose the assignment that minimizes the difference. As dictated by the recursion, we do this for each a_i until we reach the base case $f(n, x) = x$, which is the minimum residue of the entire sequence A.

We can derive the appropriate signs s_i by simply storing the sign we choose for a_i at each step of the recurrence in a lookup table. The running time of this algorithm is $O(nb)$ — there are n integers, and at any given point, x is at most b (and at least $-b$).

II. O(nlogn)

The Karmarkar-Karp algorithm can be implemented in $O(n \log n)$ time by using a priority queue, in the form of a binary max heap. Building a binary max heap from an unsorted array takes $O(n)$ time, removing the maximum element takes $O(\log n)$ time, and inserting an element takes $O(\log n)$ time. The KK algorithm involves building a heap, removing $O(n)$ elements, and inserting $O(n)$ elements, for a total runtime of $O(n + n \log n + n \log n) = O(n \log n)$.

III. Data and Discussion

	KK	repeated random	hill climbing	simulated annealing
standard	average: 604,424.45 min: 4,248 max: 2,749,078 avg. time: 22.89	average: 237,544,148.71 min: 4,870,713 max: 1,305,842,511 avg. time: 46,777.74	average: 370,623,294.20 min: 726,458 max: 1,129,609,151 avg. time: 16,666.51	average: 312,097,885.75 min: 35,956 max: 1,726,595,382 avg. time: 31,654.36
prepartition		average: 229.88 min: 0 max: 790 avg. time: 731,500.32	average: 442.61 min: 2 max: 4,817 avg. time: 677,299.86	average: 214.34 min: 0 max: 883 avg. time: 1,831,636.04

The table above shows the average residue, minimum and maximum residue, and average time taken by each of the algorithms over the 100 instances. Overall, using the prepartitioning representation yielded smaller residues on average than the standard representation, though they also took much longer to run (especially the simulated annealing algorithm). Hill climbing using the standard representation yielded the largest residues on average, but it was also the fastest in terms of runtime. The residues yielded by the hill climbing and simulated annealing algorithms also had more variance, possibly due to the fact that they are highly dependent on the random solution you begin with.

The algorithms could potentially be improved by first using the Karmarkar-Karp algorithm to find a solution, then using the solution as a starting point for the algorithms rather than a random one. From the data, we can see that the KK algorithm yields a much lower residue on average than the other algorithms (using the standard representation). Therefore, using the KK algorithm as a starting point should lead to lower residues on average.