Algorithms Homework Problems: Lab 8 — The Skis Problem

Henry Daniels-Koch & Bo Bleckel

November 11, 2016

1. If the number of skiers and skis are the same, the solution is as follows: sort both arrays (skis and skiers) and then assign the shortest skis to the shortest skier, the second shortest skis to the second shortest skier, and so on.

   To prove the correctness, we must show $|P1 - S1| + |P2 - S2| \leq |P1 - S2| + |P2 - S1|$ holds for all of the following (note that we can eliminate the absolute values in some cases because we are assuming strict inequality):

   (a) $P1 < S1 < P2 < S2$:

   The difference between $P1$ and $S2$ encompasses the difference between $P1$ and $S1$, and $P2$ and $S2$. Therefore the inequality holds.

   (b) $P1 < S1 < S2 < P2$:

   The difference between $P2$ and $S1$ encompasses the difference between $P2$ and $S2$, and $P1$ and $S2$ encompasses the difference between $P1$ and $S1$. Therefore the inequality holds.

   (c) $P1 < P2 < S1 < S2$:

   Here, both sides of the inequality are equal. This is because $S1 - P1 + S2 - P2 = S2 - P1 + S1 - P2$. That implies $|P1 - S1| + |P2 - S2| = |P1 - S2| + |P2 - S1|$.

   (d) $S1 < P1 < P2 < S2$:

   $P1 - S1 + S2 - P2 \leq S2 - P1 + P2 - S1 \implies P1 - P2 \leq P2 - P1$ Because $P2$ is greater than $P1$, the inequality holds.

   (e) $S1 < P1 < S2 < P2$:

   $P1 - S1 + P2 - S2 \leq S2 - P1 + P2 - S1 \implies P1 - S2 \leq S2 - P1$. Because $P1$ is less than $S2$, this holds.

   (f) $S1 < S2 < P1 < P2$:

   Here, both sides are equal. $P1 - S1 + P2 - S2 \leq P1 - S2 + P2 - S1 \implies 0 \leq 0$. This is true because zero equals zero.

2. Hard copy of code attached in back.

3. Solution to part (4): the optimal paring of skis for skis $= \{1, 2, 5, 7, 13, 21\}$ and skiers $= \{3, 4, 7, 11, 18\}$ yields an sum of the disparities equal to $\boxed{7}$. The table looks like:

2	1	0	0	0	0
0	4	2	0	0	0
0	0	6	2	0	0
0	0	0	10	4	0
0	0	0	0	0	7

The reason that our table doesn't fill in everything is for two reasons: (1) We will never fill in below the diagonal because it is not an option for a skier to get a ski if there are fewer skis than skiers. and (2) We will not fill in more than one diagonal besides the two that make it down the diagonal because we have a heuristic base case that doesn't let a skier get a pair of skis that are shorter than the skier. This works because at any point we are looking at two indices: skis and skiers. If those indices are such that the skier is taller, it is optimal for us to put the skier with those skis.

4. We tested our algorithm on an iMac in the lab. This machine 4GHz Intel i7 with 16 GB of memory. We compiled it in c++ using the g++ compiler with a -o flag. We tested the algorithm first on sets that we knew the answer to (i.e. sets with the same number of skiers and skis, and smaller sets). Once we were convinced that the algorithm was working properly, we began testing it on larger sets to make sure that the recursive and dynamic programming versions were giving the same results.

5. We were able to find significant differences in speed between the two algorithms. Below are our results:

Num Skiers: 40	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	**Avg Run time**
Num Skies: 40	6.0E-3	3.9E-6	1.9E-6	5.0E-6	5.0E-6	1.2032E-3
Num Skies 45	0.43	1.53	5.0E-6	0.23	7.3E-3	0.4
Num Skis: 47	2.5E-2	8.4	4.7E-2	0.98	0.28	1.9576
Num Skis 48	6.0E-3	1.51	19.0	8.5E-2	3.0E-6	4.1

The average run time for the dynamic programming algorithm was consistently between 1.0×10^{-6} and 4.0×10^{-6}.

We noticed a very large difference in results from run to run. We reconciled this with our heuristic base case, noting that whenever the skier that we're looking at is taller than the skis we're looking at, we return. So if the random array gets generated so that the skiers are all taller than the skis (or most of the skiers are taller than the skis) the algorithm runs in linear time. This results in a very very fast algorithm. However, this doesn't always happen, ultimately giving us a very wide variety of run times for the recursive algorithm.

6. We conclude that the dynamic programming algorithm is far superior to the non-dynamic programming one. In addition, it takes little to almost no work to implement the dynamic programming version once the recursive algorithm has been designed. The test results for the dynamic algorithm showed run times on the order of magnitude of 10^{-6} seconds.