Homework 7

Adam Karl

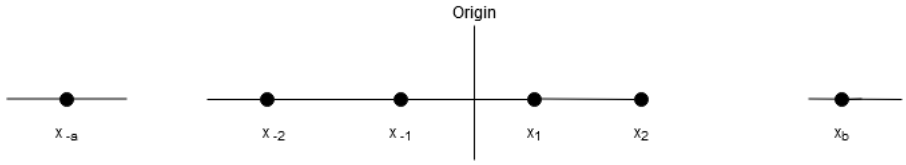
October 6, 2020

1 Problem 22 (6 points)

1.1 Setup

First, sort all points in ascending order. Then, determine the number of vertices left of the origin (let this value be a) and the number of vertices at or right of the origin (let this value be b). Note that $a + b = n$.

To make the problem easier to work with, rename the vertices left of the origin from x_{-a} to x_{-1} and the vertices right of the origin from x_1 to x_b. These are the labels we will work with for the rest of the problem.



renamed number line

Construct array A with n rows and n columns. While the rows are numbered 1 to n, the columns will be numbered from -a to -1, then 1 to b. This numbering for the columns (some negative indices, no 0 column) is unconventional, but will be useful for keeping track of where we are when filling in the array.

	-a	-a + 1	...	-2	-1	1	2	...	b-1	b
1										
2										
3										
...										
n										

A[i,j] describes a path that starts at the origin, takes i hops, and ends at vertex x_j. Note that j can be negative. Inside will be stored the minimum aggregate wait time and corresponding path length. The aggregate wait time is the "main" value used for comparisons, the path length is used in future calculations.

The core intuition of the algorithm is that at any point, the current path makes up a contiguous set of points including the origin, and when extending the path we consider the first point to the right of the current path and the first point to the left of the current path.

1.2 Algorithm Description

- fill in A[1,-1] with tuple $(|x_{-1}|, |x_{-1}|)$
  - 1 hop, ends at x_{-1}
  - aggregate wait time = path length = $|x_{-1}|$
- fill in A[1,1] with tuple (x_1, x_1)
  - 1 hop, ends at x_1

- aggregate wait time = path length = x_1

- for i from 2 to n
  - for j from MAX(-i, -a) to MIN(i, b), skipping over j=0
    - //essentially, from -i to i unless we've run out of vertices on one side of the origin
    - if j < 0:
      - consider elements A$[i-1, j+1]$ and A$[i-1, i+j+1]$. For each of these "old" endpoints (x_{j+1} and x_{i+j+1}), calculate possible aggregate wait times for A[i,j] as old aggregate wait time + old path len + distance from old endpoint to new endpoint (x_j). Choose the minimum among the two calculated aggregate wait times to store in A[i,j] along with the new path length (old path length + distance from old endpoint to current point).
    - if j > 0:
      - same as when j < 0 but consider elements A$[i-1, i-j+1]$ and A$[i-1, j-1]$ (old endpoints x_{i-j+1} and x_{j-1}.

For instance, A[5,3] (5 hops, ends at x_3) depends on A[4,-2] and A[4,2]
A[5,4] depends on A[4,-1] and A[4,3].
A[5,5] depends on only A[4,4] (there is no A[4,0])

1.3 Conclusion and Trace-back

The final path will have n hops and end at either x_{-a} or x_b. To determine which, find MIN(A[n,-a], A[n,b]) when looking at the first value in each tuple, which is the aggregate waiting time. The average waiting time is this value divided by n. This solution has a runtime of $O(n^2)$.

To trace back the path, the previous new point visited before the current point was either the next point in the direction of the origin (but on the same side of the origin as the current point), or the furthest unvisited point on the other side of the origin (if x_{-a} was the end point, then we consider x_{-a+1} and x_b). Note that these are the same points that are considered when calculating the value to store in A[i,j]. Do the same aggregate wait time calculations from each of these points in the previous row of A to see which one matches the aggregate wait time in the current cell. This was the previous point visited.

Repeat this process to trace the solution all the way back to A[1,-1] or A[1,1], the first point visited after leaving the origin.