CPSC 335 Project 4: Iceberg Avoidance

Group Members:

Timothy Bui | timothybui98@csu.fullerton.edu

Jimmy Phong | jimmyphong16@csu.fullerton.edu

Project Description

In this project we will implement two algorithms that both solve the *iceberg avoidance problem*. The first algorithm uses exhaustive search, is similar to the exhaustive search algorithm from project 2, and takes exponential time. The second algorithm uses dynamic programming, and takes quadratic time.

The Iceberg Avoidance Problem

The "iceberg avoidance problem" is a puzzle that comes from real life, where ships have to travel through iceberg ridden water in order to find safe passage to reach their destination. For example, we could consider supply ships trying to reach habited places in Antarctica. But due to global warming or changes in the ocean salinity, there are icebergs (that we consider stationary) of various sizes that are impassable. The ship starts in Chile which is represented by one corner of a matrix (location [0][0]) and needs to reach safely Antarctica, represented by the corner across the ocean (location [r-1][c-1]). The icebergs are impenetrable and crossing the ocean can be done only moving right, from location [i][j] to location [i][j+1], or down, from location[i][j] to [i+1][j]. There is no path that can go through an iceberg so the ship can move into a new location if there is no iceberg at that location.

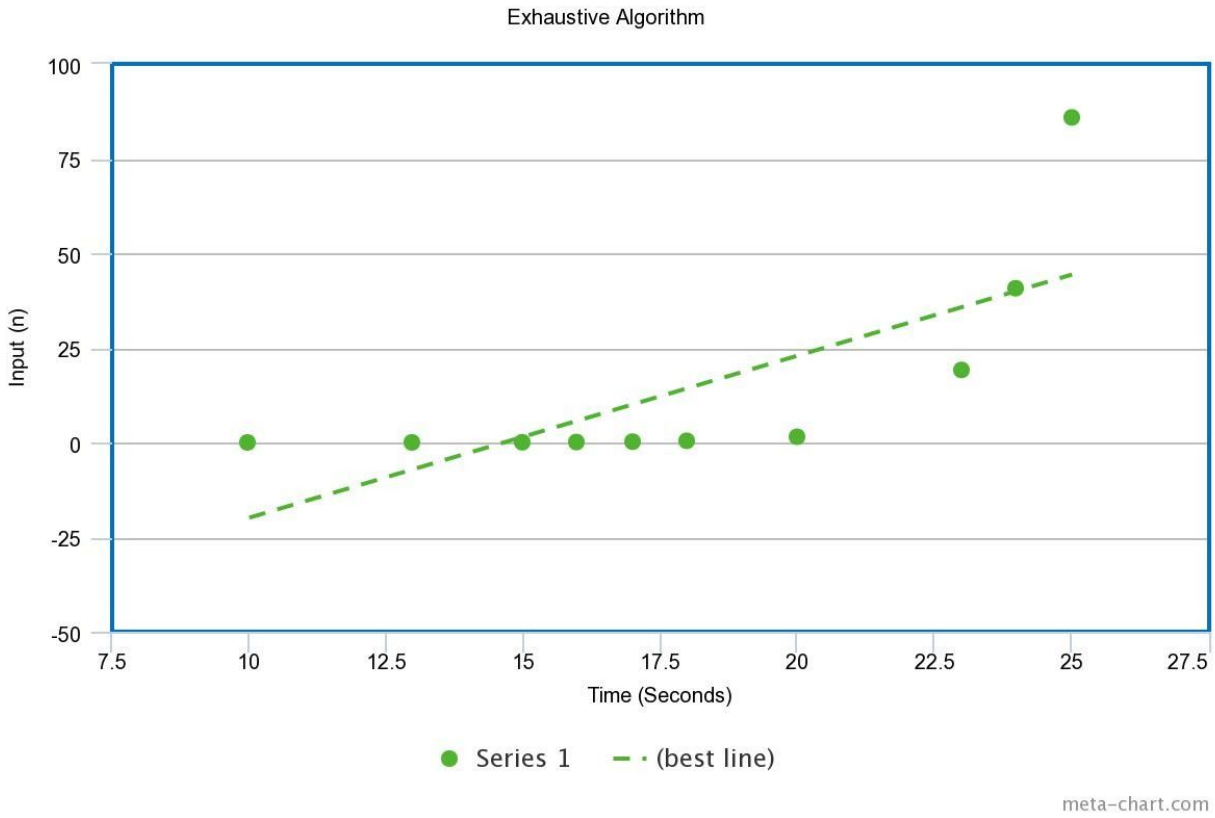
Iceberg avoidance problem
input: a r×c matrix G where each cell is one of . (passable) or X (impassable); and G[0][0]= . **output:** the number of different paths starting at (0, 0) **and end at location [r-1][c-1]**; where each step is either a start, right move, or down move; that does not visit any X cell

The ship starts at row 0 and column 0, i.e. coordinate (0, 0), at the top-left corner. Each `.` cell represents clear water (i.e. a passable spot) and each `X` represents an iceberg (i.e. an impenetrable spot). The ship's goal is to plan a passable route to cross the ocean while avoiding the icebergs. The problem objective is to compute the number of different paths to cross the ocean. Two paths are different if they differ by at least one spot.

Scatter plots:

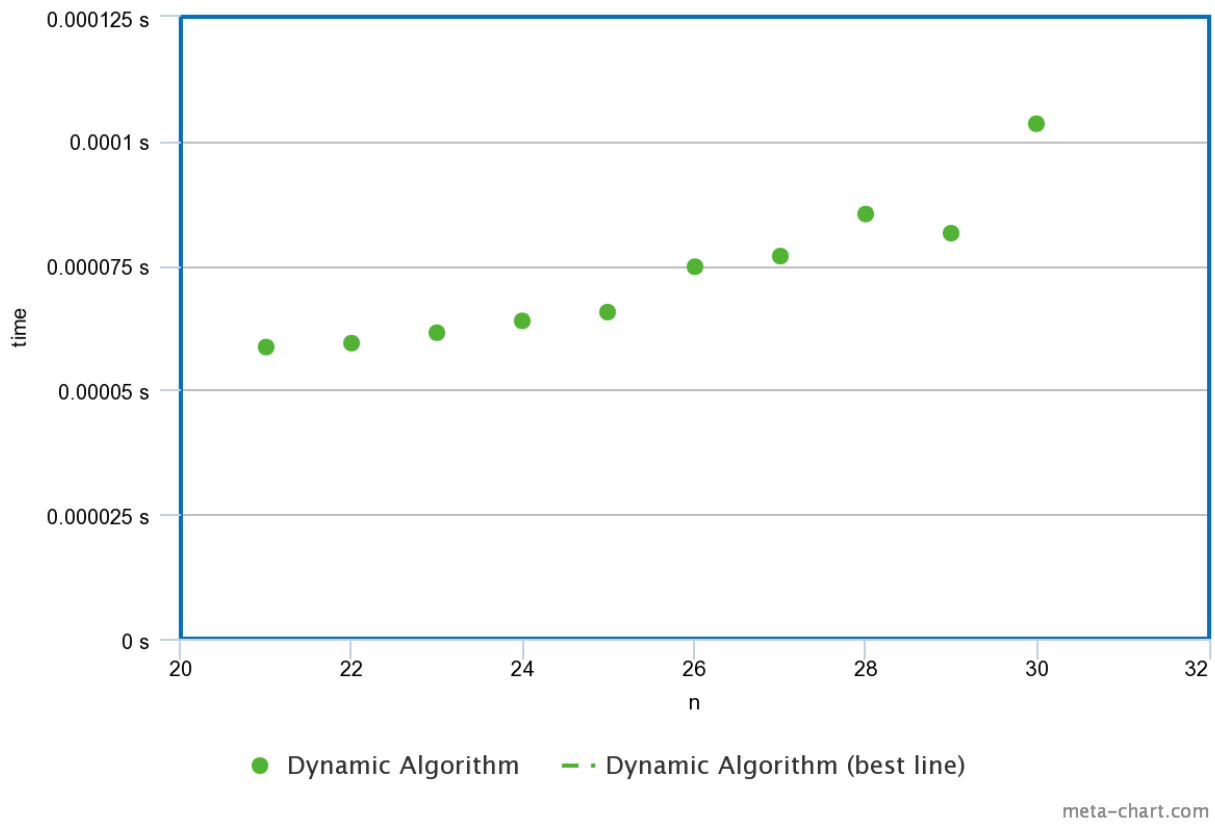
Exhaustive Algorithm

Exhaustive Algorithm Data Plot					
n	# of paths	time (s)	n	# of paths	time (s)
10	38	0.000984854	18	3504	0.435615
13	7	0.0100229	20	2025	1.49233
15	305	0.0513965	23	27153	19.1568
16	614	0.102072	24	83826	40.7378
17	849	0.205936	25	126132	85.8584

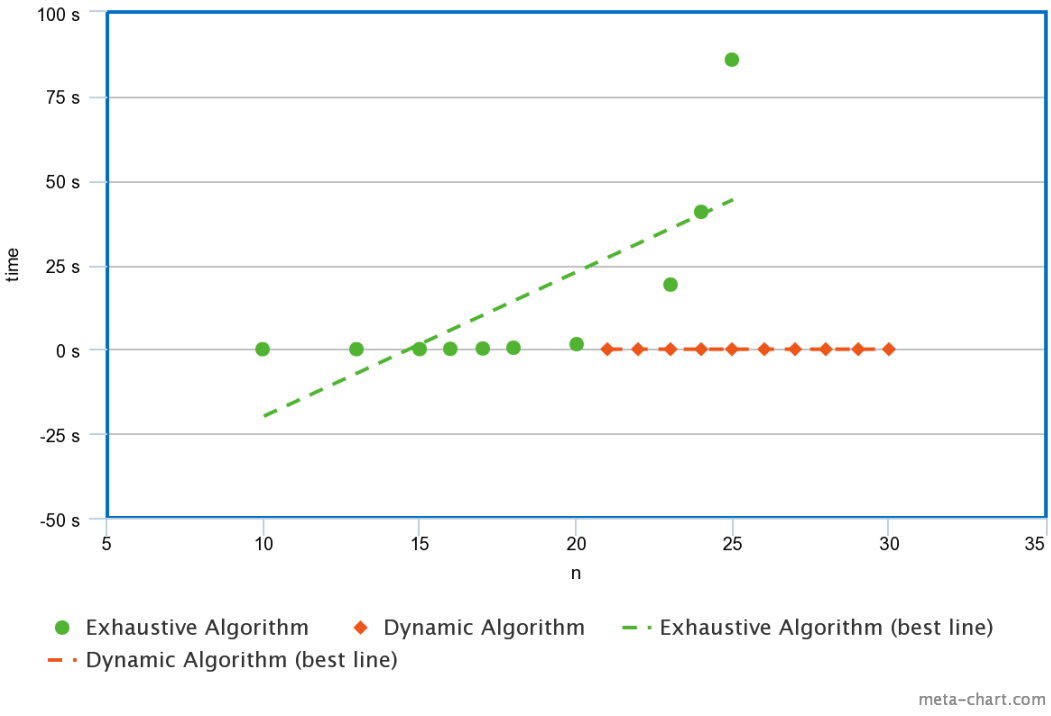


Dynamic Algorithm

Dynamic Algorithm Data Plot					
n	# of paths	time (s)	n	# of paths	time (s)
3	1	0.000048763	17	849	0.000086445
4	2	0.000051134	18	3504	0.000058980
5	3	0.000051123	19	3671	0.000054786
6	6	0.000051445	20	2025	0.000057172
7	7	0.000052034	21	8940	0.000058621
8	16	0.000052435	22	49808	0.000059413
9	13	0.000057250	23	27153	0.000061502
10	38	0.000215087	24	83826	0.000063882
11	51	0.000058890	25	126132	0.000065644
12	61	0.000055857	26	154014	0.000074772
13	7	0.000056260	27	212425	0.000076905
14	272	0.000042570	28	79495	0.000085359
15	305	0.000043727	29	0	0.000081508
16	614	0.000109810	30	1428396	0.000103504


0.000125 s
0.0001 s
0.000075 s
0.00005 s
0.000025 s
0 s
time
20
22
24
26
28
30
32
n
Dynamic Algorithm
Dynamic Algorithm (best line)
meta-chart.com

Both


100 s
75 s
50 s
25 s
0 s
-25 s
-50 s
time
5
10
15
20
25
30
35
n
Exhaustive Algorithm
Dynamic Algorithm
Exhaustive Algorithm (best line)
Dynamic Algorithm (best line)
meta-chart.com

Project Questions

1. Are the fit lines on your scatter plots consistent with these efficiency classes? Justify your answer.
The fit line for the exhaustive algorithm is accurate to the expected time complexity of O(n2^2). The parabolic incline ascends between 20 - 25 seconds. The fit line for the dynamic programming algorithm is accurate to the expected time complexity of O(n^2). The graph shows a flat parabolic shape as time increases.

2. Is the evidence consistent or inconsistent with the hypothesis stated on the first page? Justify your answer.
Yes, the evidence indicates that the dynamic algorithm is exponentially faster as the size of the grid increases. The exhaustive algorithm was slower at every instance of n, and after n=20, the exhaustive algorithm's runtime increased exponentially. The dynamic algorithm was fairly consistent in runtime regardless of how many paths it calculated.

3. Compare and contrast the difficulty you found in implementing the two algorithms. What was the most challenging part of implementing each algorithm. Overall, which implementation did you find harder, and why? Which algorithm implementation do you prefer?
Implementing the two algorithms was proven difficult in our experience. The most challenging part of this project was understanding the given classes and their relationship with each other. For the exhaustive algorithm, the classes took a while to understand. The dynamic programming algorithm was a little easier because we finished the exhaustive program first. Between these two algorithms, it would be harder to implement the dynamic programming algorithm because the syntactic complexity of the algorithm was messy and hard to read. We preferred to implement the exhaustive algorithm because the classes relations and functions helped simplify the code structure.