Project 2:
Crane Problem

Group Members
Chence Shi: chenceshi@csu.fullerton.edu
Vivian Tran: vtran2535@csu.fullerton.edu

Cal State Fullerton
CPSC 335 - Algorithm Engineering
May 15, 2023

Exhaustive Search

Pseudocode

```
For steps = 0 to max_steps do:
        pow_set_size = pow(2,steps)
        For counter = 0 to pow_set_size do:
                current_path = new path
                is_valid _path = true

                For k = 0 to steps - 1 do:
                        direction = undefined
                        bit = (counter >> k) AND 1

                        If bit == STEP_DIRECTION_EAST
                                direction = STEP_DIRECTION_EAST
                        Else
                                direction = STEP_DIRECTION_SOUTH
                        ENDIF

                        If current.path is step valid(direction)
                                current_path add_step(direction)
                        Else
                                is_valid_path = false
                                Break
                        ENDIF
                ENDFOR

        If is_valid_path

                If current_path.total_cranes()>best_path.total_cranes()
                        best_path = current_path

                ENDIF
        ENDIF
   ENDFOR
ENDFOR

return best_path
```

Time Analysis

The time of exhaustive search algorithm affected by the nested loops:

- The outer loop iterates over all possible step lengths from 0 to max_steps, where max_steps = setting.rows() + setting.columns() - 2. The maximum value of max_steps is (rows - 1) + (columns - 1), or rows + columns - 2.

- The inner loop iterates over all possible combinations of steps using powerset. There are 2^steps possible combinations, as each step can be either east or south.
    - The inner loop's body mainly involves checking if the current path is valid and adding a step if it is. These operations take exponential time, O(2^n), since they involve powerset bitwise operations.

To find the overall time complexity, we need to consider the number of iterations in both loops and the time complexity of the inner operations.

- For the outer loop, we have max_steps iterations.
- For the inner loop, we have 2^steps iterations. In the worst case, steps = max_steps, so we have 2^(rows + columns - 2) iterations for the inner loop. Since the outer operations take n time, the overall time complexity is O(n * 2^(rows + columns - 2)).

- This algorithm has an exponential time complexity, which means its runtime will grow exponentially with the size of the grid. As the grid size increases, the time it takes to find the best path will increase rapidly, making it impractical for large grids.

Dynamic Search

Pseudocode

```
If (r > 0 AND A[r-1][c] has value):
        From_above = A[r-1][c]
        if(from_above -> is_step_valid(STEP_DIRECTION_SOUTH)):
                From_above -> add_step(STEP_DIRECTION_SOUTH)
        ENDIF
ENDIF

if(c > 0 AND A[r][c-1] has value):
        From_left = A[r][c-1]
        if(from_left->is_step_valid(STEP_DIRECTION_EAST)):
                from_left->add_step(STEP_DIRECTION_EAST)
        ENDIF
ENDIF

if(from_above has value AND from_left has value)
        if(from_above->total_cranes() > from_left->total_cranes())
                A[r][c] = from_above
        else
                A[r][c] = from_left
        ENDIF
else if (from _left has value)
        A[r][c] = from_left
else if(from_above has value)
        A[r][c] = from_above
ENDIF

Cell_type *best = &(A[0][0])

assert(best -> has_value())
```

```
For each r in A do:
        For each c in A do:
                If (A[r][c] has value AND A[r][c] ->total_cranes() > (*best) ->
total_cranes())
                        best = &(A[r][c])
                ENDIF
        ENDFOR
ENDFOR

assert(best -> has_value())

return *best
```

Time Analysis

The dynamic programming algorithm affected by the nested loops:

- The nested loops:
    - The outer loop iterates over all rows in the grid, which is 'setting.rows()' times.
    - The inner loop iterates over all columns in the grid, which is 'setting.columns()' times.

- The inner operations:
    - The inner loop's body mainly involves checking if the current cell is a building, finding paths from above and from the left, and updating the current cell with the best path found so far. These operations take constant time, O(n) since they involve copying from the previous position and adding extra path assignments.

To find the overall time complexity, we need to consider the number of iterations in both loops and the time complexity of the inner operations.

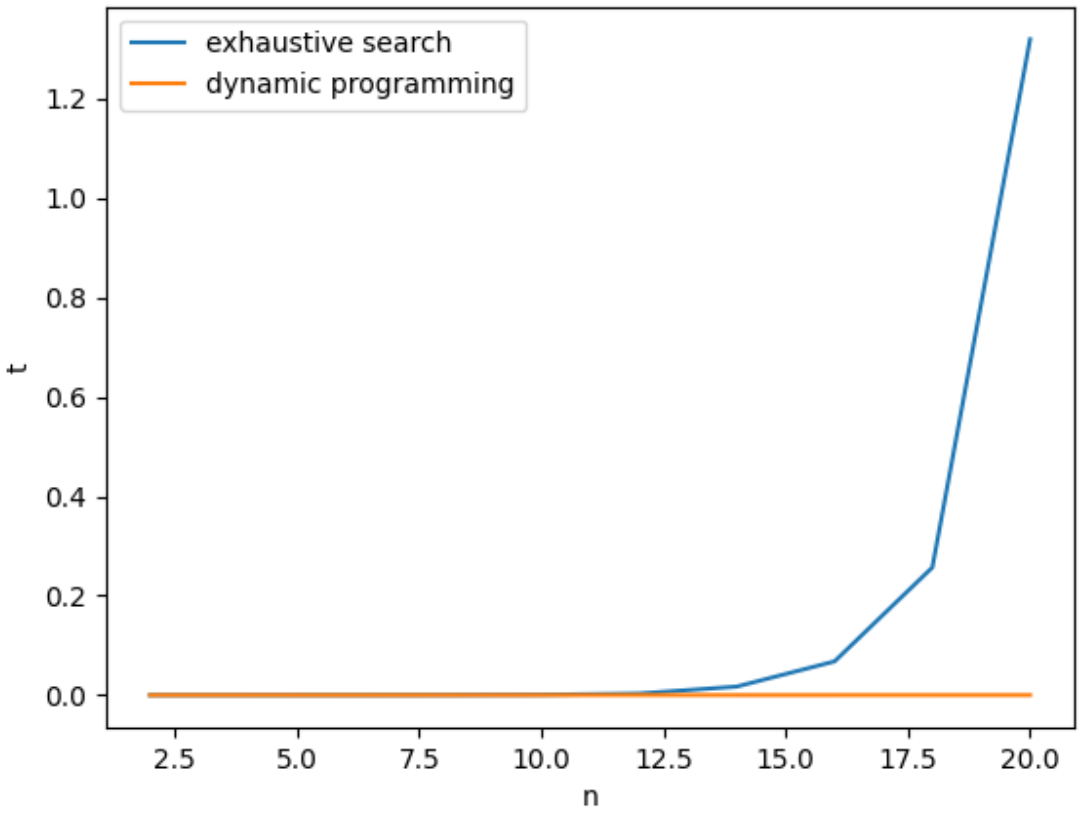
- For the outer loop, we have 'setting.rows()' iterations, and for the inner loop, we have 'setting.columns()' iterations. The nested loops have a total of 'setting.rows()' * 'setting.columns()' iterations. Since the inner operations take n time, the overall time complexity for these loops is O(rows * columns * n) = O(n^3).

- Additionally, there is another set of nested loops at the end of the function that iterates over all rows and columns to find the best path. This also has a time complexity of O(rows * columns).

- Therefore, the total time complexity of the dynamic programming algorithm is O(n^3), which is significantly better than the exponential time complexity of the exhaustive search algorithm. This algorithm is more practical for larger grids as its runtime grows linearly with the size of the grid.

Graph

Large Input



Small Input

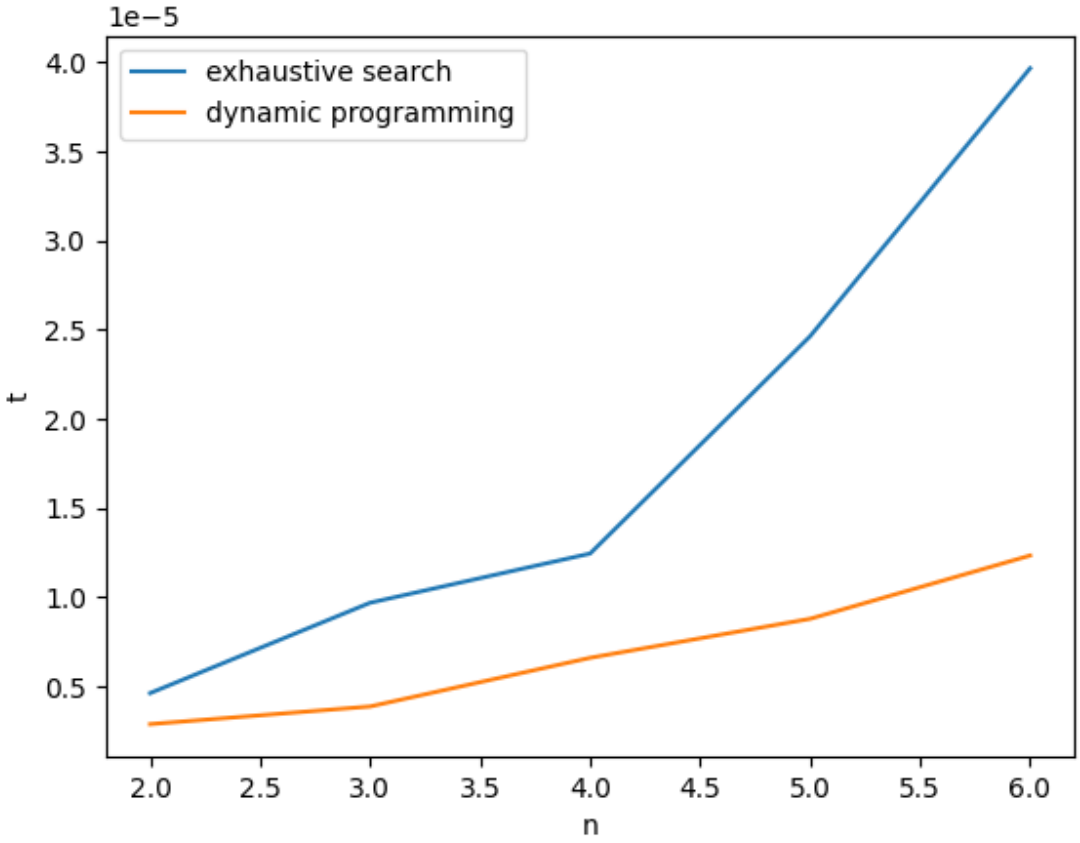


Figure 1: Graph that shows exhaustive and dynamic relative to time and input size

Questions

1. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?
    a. There is a noticeable difference between the performance of dynamic and exhaustive. To the size of 'n', the run times of the dynamic move significantly faster than exhaustive. When comparing various sizes of 'n', the significant difference was shown in n = 6. This illustrated the dramatic difference between dynamic and exhaustive. Dynamic became significantly faster than exhaustive.

2. Are your empirical analysis consistent with your mathematical analyses? Justify your answer.
    a. When looking at the graph, it seems that exhaustive has a steeper slope in contrast to dynamic. The slope shows how much time exhaustion is required, especially when dealing with input sizes larger than 6. Therefore, empirical analysis supports and is consistent with the mathematical analysis conducted.

3. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.
    a. With evidence provided, polynomial time dynamic programming shows a higher efficiency than exponential time exhaustive search. As mentioned in the time analysis, the dynamic is able to take larger inputs and continue to run in less amount of time. Unlike exhaustive search, it needs to take more time in order to fulfill larger inputs. As a result, it shows that the evidence is consistent with the hypothesis.
4. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.
    a. As mentioned previously, polynomial time dynamic programming demonstrates a higher efficiency than exponential time exhaustive. Therefore, the evidence shown is inconsistent with hypothesis 2. As seen through Figure 1, exponential time exhaustive needs more time when it receives an input greater than n = 6. This is in contrast to dynamic programming which remains low time as n grows. This further proves that the evidence is inconsistent with hypothesis 2.