CPSC 335 Project 2

Alex Ly - aly26@csu.fullerton.edu
Saad Ansari - Saadansari@csu.fullerton.edu

Exhaustive Algorithm Pseudocode and Time Analysis

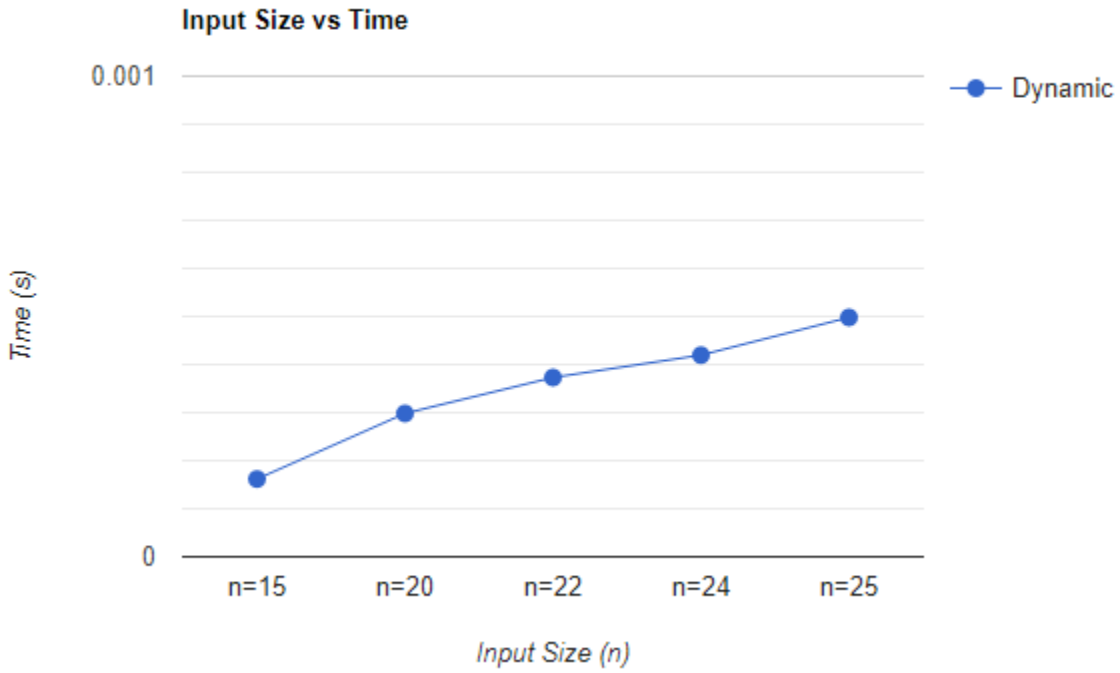
```
  const size_t max_steps = setting.rows() + setting.columns() - 2;

  path best(setting);
  for (size_t steps = 0; steps <= pow(2, max_steps); steps++) {
    path current_path(setting);
    std::vector<step_direction> count_steps;
    for(size_t i = 0; i < max_steps; i++) {
      int iter = (steps >> i) & 1;
      count_steps.push_back((iter == 1) ? STEP_DIRECTION_SOUTH : STEP_DIRECTION_EAST);
    }
    for(step_direction step: count_steps) {
      if(!current_path.is_step_valid(step)) {
        break;
      }
      current_path.add_step(step);
    }
    if(current_path.total_cranes() > best.total_cranes()) {
      best = current_path;
    }
  }
return best;
```

We will assert n = rows, and m = column
OUTER LOOP
`for (size_t steps = 0; steps <= pow(2, max_steps); steps++)`
Loop will iterate 2^(max_steps) times.
Max_steps = `setting.rows() + setting.columns() - 2`
n+m-2
INNER LOOP
Will take max_steps as well

Loop2
Check for collisions in max_steps iterations

Time Complexity O(n^(rows + columns))

Graph for Time vs Input Size



Dynamic Algorithm Pseudocode and Time Analysis

```
A[0][0] = path(setting);
assert(A[0][0].has_value());

for (coordinate r = 0; r < setting.rows(); ++r) {
  for (coordinate c = 0; c < setting.columns(); ++c) {

    if (setting.get(r, c) == CELL_BUILDING){
      A[r][c].reset();
      continue;
    }

    if (r > 0 && setting.get(r - 1, c) != CELL_BUILDING && A[r - 1][c].has_value()) {
      from_above = A[r - 1][c].value();
      from_above->add_step(STEP_DIRECTION_SOUTH);
    }

    if (c > 0 && setting.get(r, c - 1) != CELL_BUILDING && A[r][c - 1].has_value()) {
      from_left = A[r][c - 1].value();
      from_left->add_step(STEP_DIRECTION_EAST);
    }

    if (from_above.has_value() && from_left.has_value()) {
      A[r][c] = from_above->total_cranes() >= from_left->total_cranes() ? from_above : from_left;
    } else if (from_above.has_value() && !from_left.has_value()) {
      A[r][c] = from_above;
    } else if (!from_above.has_value() && from_left.has_value()) {
      A[r][c] = from_left;
    }
  }
}

for (coordinate row = 0; row < setting.rows(); ++row) {
  for (coordinate col = 0; col < setting.columns(); ++col) {
    if (A[row][col].has_value() && A[row][col]->total_cranes() > max_cranes) {
      row_max = row;
      col_max = col;
      max_cranes = A[row][col]->total_cranes();
```

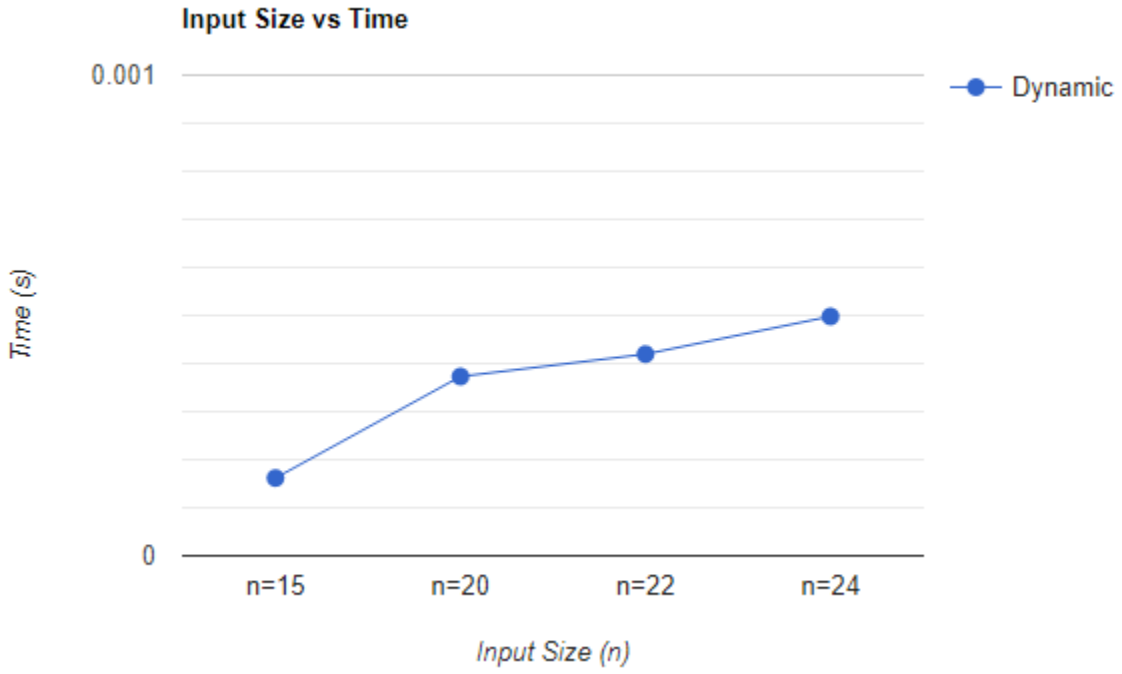
```
    }
  }
}

cell_type *best = &A[row_max][col_max];

assert(best->has_value());

return **best;
```

First Nested Loop
Iterates through any cell in the grid, each cell receives a constant number of operations.
O(nm)

Second Nested Loop
Iterates every cell in grid, and performs constant operations on each cell.
O(nm)

O(nm)

Graph for Time vs 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. Yes, there is a noticeable difference in the performance of the two algorithms. The dynamic programming algorithm is significantly faster than the exhaustive optimization algorithm. The time complexity of the exhaustive optimization algorithm is exponential since it needs to explore all possible paths through the grid. On the other hand, the dynamic programming algorithm is quadratic since it only needs to loop over all cells in the grid once and compute the best path for each cell based on the best paths for its neighboring cells. It's not surprising to me that the dynamic programming algorithm is more efficient than the exhaustive search algorithm.
2. **Are your empirical analyses consistent with your mathematical analyses? Justify your answer.**
    a. Yes, by testing with various input sizes we could find a consistency between what is expected from our mathematical analysis and our empirical. Upon observation of the graphs, it is visually clear that the dynamic programming algorithm is able to return solutions in much faster time. In analyzing the time complexity, we found the first algorithm to have a worst-case time complexity of O(2^(n+m)) which is exponential. The dynamic programming implementation was found to have a worse-case time complexity of O(nm) . An example of our empirical evidence being consistent with this mathematical analysis is the observation in difference between input size and time returns. In the case of an input size = 30, the exhaustive optimization algorithm returned a solution in 1251.47 seconds. The Dynamic programming solution returned the same result in 0.000442797 seconds. This would be expected when considering the efficiency of the two worst-case time complexities of the respective algorithms.
3. **Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.**
    a. There seems to only be one hypothesis: "Polynomial-time dynamic programming algorithms are more efficient than exponential-time exhaustive search algorithms that solve the same problem." In this case we would have to conclude that this hypothesis is correct. The time analysis and empirical analysis of the efficiency support this conclusion as the time to complete the same task had drastic time outcomes. Overall, the evidence suggests that when solving optimization problems like the crane problem, dynamic programming algorithms are more efficient and a more practical approach. This is because dynamic programming avoids redundant computations by storing solutions in a table and reusing them when required, whereas exhaustive optimization algorithms will need to

generate and evaluate all possible solutions, and upon observation of our results, it becomes apparent that this will become infeasible for larger input sizes.

4. **Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.**
    a. Exhaustive optimization algorithms are only feasible for problems with small input sizes, in this case it would be untrue to say that the dynamic programming solutions are less efficient than the exponential-time exhaustive search algorithms. This is evident in both our time analysis and empirical evidence of the efficiencies of each respective algorithm.

Hypothesis: Polynomial-time dynamic programming algorithms are more efficient than exponential-time exhaustive search algorithms that solve the same problem.