Project 2: Dynamic vs. Exhaustive - Crane Unloading Problem

Authors:

- Michael Chhun
- Taylor Noh
- Quan Tru Thai
- Chenhao Wei

Exhaustive Algorithm Solution

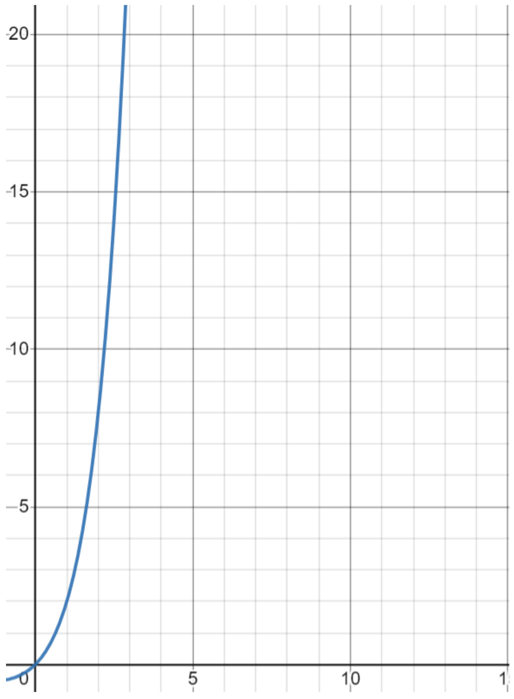
Pseudocode and Time Analysis

```
size_t max_steps = setting.rows() + setting.columns() - 2 // 2 tu
path best(setting) // 1 tu

for bits in 0 to pow(2, max_steps) - 1 do // 2^n tu
      path p(setting) // 1 tu
      vector<step_direction> candidate // 1 tu

      for k in 0 to max_steps - 1 do // n tu
            int bit = (bits >> k) & 1 // 1 tu
            if bit == 1 then
                  candidate.push_back(STEP_DIRECTION_SOUTH)
            else
                  candidate.push_back(STEP_DIRECTION_EAST)
            endif
      endfor

      for each step in candidate do // m tu
            if not p.isValidStep(step) then
                  exit inner loop
            endif
            p.addStep(step) // 1 tu
            if p.totalCranes() > best.totalCranes() then
                  best = p // 1 tu
            endif
      endfor

endfor

return best

Time Complexity:
 = 2 + 1 + 2^n * (1 + 1 + n + m)
 = 3 + 2^n * (2 + n + m)

Since ‘m’ is a constant known time complexity, which is the size of
the candidate vector.

Therefore, overall TC = 2^n * n
```

Graph for Time vs Input Size

Assuming rows = columns = n

Here’s a graph for T(n) = $2^n * n$

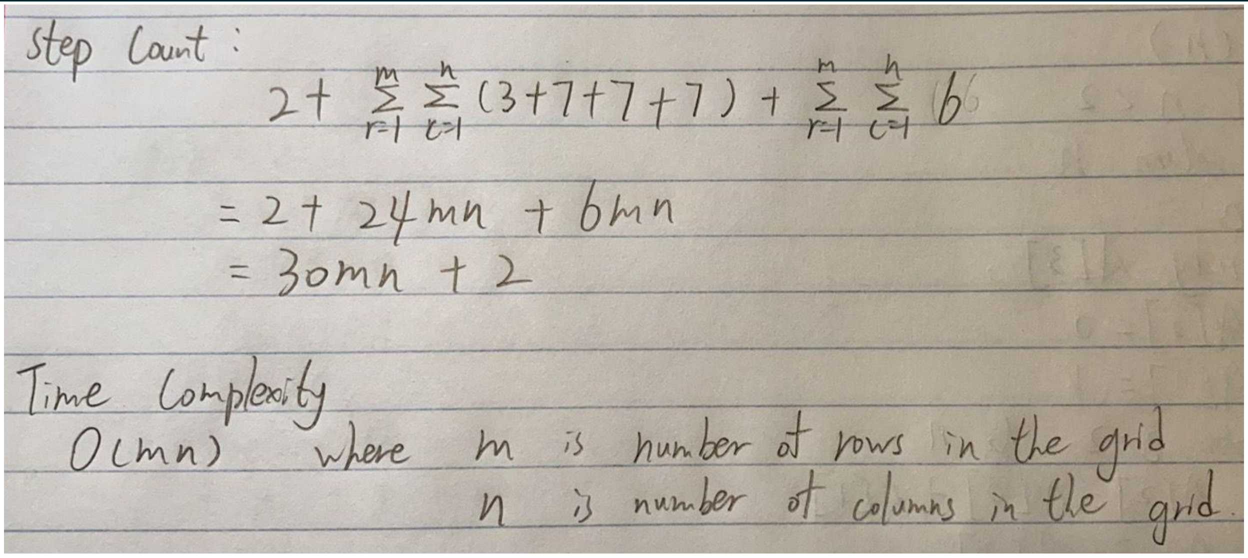


Dynamic Algorithm Solution

Pseudocode and Time Analysis

```
A = matrix of optional<path> with setting.rows() rows and
setting.columns() columns
A[0][0] = start path

for r in 1 to setting.rows():
    for c in 1 to setting.columns():
        if (Grid[r][c] == Building) {
           A[r][c].reset()
           continue;
        }

          if (r != 0 && Grid[r-1][c] != building && A[r-1][c] has
value) {
             from_above = A[r-1][c] + Step_South
          }

        if (c != 0 && Grid[r][c-1] != building && A[r][c-1] has
value) {
             from_left = A[r][c-1] + Step_East
          }

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

```
            }
     endfor
endfor

unsigned max
for r in 1 to setting.rows():
    for c in 1 to setting.columns():
        if (A[r][c] has_value && A[r][c].total_cranes() > max) {
          best = A[r][c]
          max = A[r][c].total_cranes()
        }
     endfor
endfor
return best
```



Step Count :

$$2 + \sum_{r=1}^{m}\sum_{c=1}^{n}(3+7+7+7) + \sum_{r=1}^{m}\sum_{c=1}^{n} 6$$

$$= 2 + 24mn + 6mn$$

$$= 30mn + 2$$

Time Complexity

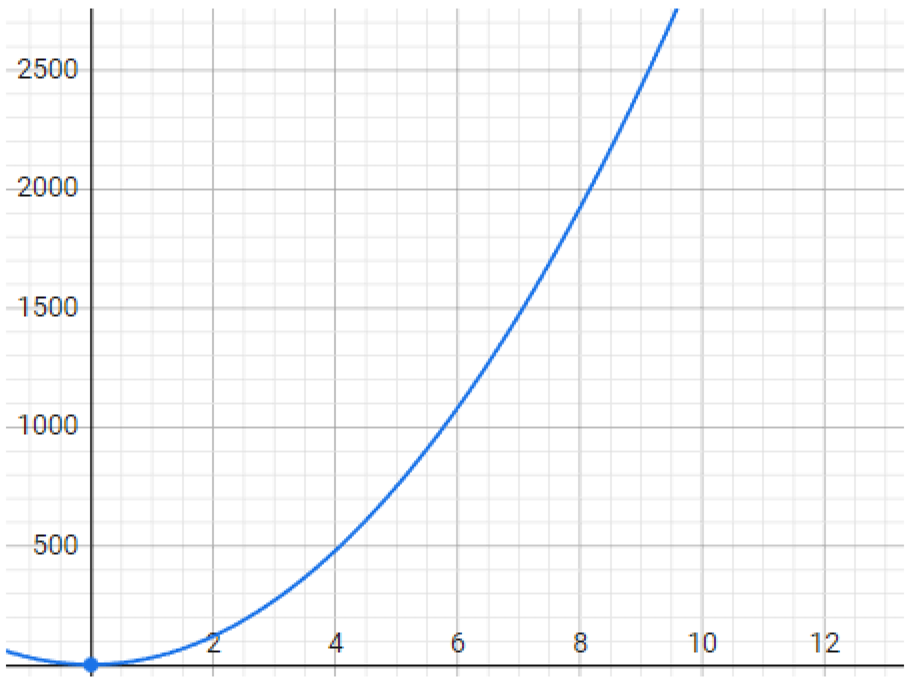
$O(mn)$ where m is number of rows in the grid

n is number of columns in the grid.

Graph for Time vs Input Size

Assuming rows = columns = n

Here's a graph for y = 30n^2 + 2



Algorithm Run Time Observations

1. **Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?**

   The dynamic programming algorithm (crane_unloading_dyn_prog) is significantly faster than the exhaustive optimization algorithm (crane_unloading_exhaustive).

   The time complexity of the exhaustive optimization algorithm is exponential, as it explores all possible combinations of directions for the truck. This means that as the grid size increases, the runtime of the algorithm grows exponentially. In contrast, the dynamic programming algorithm has a quadratic time complexity of O((rows * columns)^2), which is much more efficient.

   It is not surprising that the dynamic programming algorithm outperforms the exhaustive optimization algorithm in this case. Dynamic programming is specifically designed to efficiently solve problems with overlapping subproblems, allowing for reusing intermediate results. On the other hand, exhaustive optimization explores the entire search space, which becomes impractical for larger problem instances due to the exponential growth of possibilities.

2. **Are your empirical analyses consistent with your mathematical analyses? Justify your answer.**

   Our empirical analyses are consistent with our mathematical analyses. The exhaustive optimization algorithm has a time complexity of $O(2^n * n)$, indicating that its execution time grows exponentially as the number of steps increases. In practice, running the algorithm with larger grid sizes or more steps will result in a significant increase in execution time due to the exponential growth of the search space. Therefore, the empirical analysis is expected to demonstrate a clear trend of increasing execution time as the input size (number of steps or grid size) increases.

   On the other hand, the dynamic programming algorithm has a time complexity of $O(m * n)$, where m and n are the dimensions of the grid. This indicates that its execution time grows quadratically with the dimensions of the grid. The algorithm efficiently computes the optimal path by considering previously calculated subproblems. Consequently, the empirical analysis for the dynamic programming algorithm is expected to show a more moderate increase in execution time as the grid size increases. The execution time is likely to remain relatively stable, with a moderate increase proportional to the dimensions of the grid.

   Considering these characteristics of the time complexities and the nature of empirical analyses, it can be concluded that our empirical analyses would be consistent with our mathematical analyses for the exhaustive optimization and dynamic programming algorithms.

3. **Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.**

   The evidence is consistent with Hypothesis 1: Quadratic-time dynamic programming algorithms are more efficient than exponential-time exhaustive search algorithms that solve the same problem.

The dynamic programming algorithm (crane_unloading_dyn_prog) in the provided code has a time complexity of O((rows * columns)^2), which is quadratic. It processes each cell once, efficiently calculating the optimal path based on previously computed subproblems. This approach is known to be more efficient than exponential-time algorithms that explore all possible combinations.

4. **Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.**

   The evidence is **not** consistent with Hypothesis 2: Exponential-time exhaustive search algorithms are more efficient than Quadratic-time dynamic programming algorithms that solve the same problem.

   The exhaustive optimization algorithm (crane_unloading_exhaustive) explores all possible combinations of directions for the truck, resulting in an exponential time complexity. As the grid size increases, the runtime of the algorithm grows exponentially, making it less efficient compared to the quadratic-time dynamic programming approach.