Federal University of Ouro Preto
PCC104 - Project and Analysis of Algorithms
Brute Force and Exhaustive Search

Prof. Rodrigo Silva

October 17, 2024

Instructions

- Implementing the practical activities in C++ is highly recommended.
- Make the most use of algorithms and data structures from the STL library. `https://www.geeksforgeeks.org/the-c-standard-template-library-stl/`.
- Avoid using pointers as much as possible, but if necessary, use smart pointers `https://alandefreitas.github.io/moderncpp/basic-syntax/pointers/smart-pointers/`.
- When you need a linear data structure, always first consider using the `vector` class (`https://en.cppreference.com/w/cpp/container/vector`).

1 Recommended Reading

- Chapter 3 - *Introduction to the Design and Analysis of Algorithms (3rd Edition)* - Anany Levitin
- Book - *Problem Solving with Algorithms and Data Structures using C++* (available at: `https://runestone.academy/runestone/books/published/cppds/index.html#`)
- Arrays `https://www.interviewcake.com/concept/python/array?`
- Stacks `https://www.interviewcake.com/concept/python/stack?`
- Queues `https://www.interviewcake.com/concept/python/queue?`
- Graphs `https://www.interviewcake.com/concept/python3/graph`
- Book - *Introduction to Programming* - Alan de Freitas (available at `http://www.decom.ufop.br/alan/bcc702/livrocpp.pdf`)

2 Practical Activities

1. Implement the *Selection Sort* algorithm.
2. Implement the *SequentialSearch2* algorithm (see Section 3.2 of *Introduction to the Design and Analysis of Algorithms (3rd Edition)* - Anany Levitin).
3. Implement the breadth-first search algorithm for graphs.
4. Implement the depth-first search algorithm for graphs.
5. Implement an exhaustive search solution for the Traveling Salesman Problem.
6. Implement an exhaustive search solution for the Binary Knapsack Problem.

7. Given a grid of size $n \times n$ filled with 0, 1, 2, 3, check if there is a possible path from the origin to the destination. You can move up, down, right, and left.

   Cell Descriptions:

   - A cell value of 1 means Origin.
   - A cell value of 2 means Destination.
   - A cell value of 3 means an empty cell.
   - A cell value of 0 means a Wall (a blocked cell that cannot be traversed).

   Note: There is only one origin and one destination.

   Examples:

   - **Input:** `grid` $= \begin{bmatrix} 3 & 0 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 0 & 2 & 3 & 0 & 0 \\ 3 & 0 & 0 & 1 & 3 \end{bmatrix}$
     Output: 0
     Explanation: The grid looks like this:

     $$\begin{bmatrix} 3 & 0 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 0 & 2 & 3 & 0 & 0 \\ 3 & 0 & 0 & 1 & 3 \end{bmatrix}$$

     There is no path to reach the destination (3,1) from the origin (4,3).

   - **Input:** `grid` $= \begin{bmatrix} 1 & 3 \\ 3 & 2 \end{bmatrix}$
     Output: 1
     Explanation: The grid looks like this:

     $$\begin{bmatrix} 1 & 3 \\ 3 & 2 \end{bmatrix}$$

     There is a path from the origin (0,0) to the destination (1,1).

   Expected Time Complexity: $O(n^2)$
   Expected Auxiliary Space Complexity: $O(n^2)$

For each implementation, present an analysis of the worst-case and best-case (if applicable) time complexity of the algorithm. This analysis should include:

- A mathematical expression defining the number of operations (recurrence relation for recursive algorithms or summations for iterative algorithms).
- The final expression of the cost function.
- Indication of the efficiency class (O or Θ). The indication of the class must be justified. You can prove it by definition, by bounds, or use results demonstrated in the first exercise list (related to Chapter 2 of the book).