Theory Assignment-5: ADA Winter-2024

Kshitij (2022256) Krishna Shukla (2022254)

1 Preprocessing

Before applying the algorithm, we need to preprocess the input data. The preprocessing steps include:

1. Read input data: Read the dimensions of each box from the input. We assume that the input is in the format of block ID, length, breadth, and height.
2. Remove duplicates: We are Removing duplicate boxes if present in the input.
3. we are assuming strictly greater dimensions for nesting

These preprocessing steps help ensure that the input data is in a suitable format and that unnecessary computations are avoided during the main algorithm execution.

2 Algorithm Description

The algorithm is designed to find the maximum number of nested boxes and the number of visible boxes based on the dimensions of the boxes provided. The process involves the following steps:

1. Create Nodes: For each box, create two nodes: u_i and v_i, where i represents the box index.
2. Construct Graph:
    - Create a directed graph with a super source and a super sink.
    - Connect the super source to each u_i node with an edge of capacity 1.
    - Connect each v_i node to the super sink with an edge of capacity 1.
    - For each pair of nodes (u_i, v_j) where $i \neq j$, if box v_j can be nested inside box u_i, add a directed edge from u_i to v_j with capacity 1.
3. Find Maximum Flow: Apply any maximum flow algorithm (e.g., Ford-Fulkerson) to find the maximum flow from the super source to the super sink in the constructed graph.
4. Interpret Maximum Flow:
    - The maximum flow represents the maximum number of times a box can be nested inside another box.
    - Calculate the minimum number of visible boxes as: number of boxes - max flow.
5. Determine Nesting Configuration: Retrieve the matching edges from the maximum flow, where each matched edge (u_i, v_j) indicates that box v_j is nested inside box u_i.
6. Output: Return the number of visible boxes and the nesting configuration.

3 Data Structure

The data structure used in this algorithm is the 'Box' class, which represents each box. It has the following attributes:

- **ID**: An identifier for the box.
- **Width**: The width of the box.
- **Height**: The height of the box.
- **Depth**: The depth of the box.

4 Complexity Analysis

The time and space complexity of the algorithm can be analyzed as follows:

- **Time Complexity**:
  - **Bipartite Graph Construction**: Constructing the bipartite graph involves comparing each box with every other box to determine if one can be nested inside the other. This process requires $O(n^2)$ comparisons, where n is the number of boxes. Thus, the time complexity for constructing the graph is $O(n^2)$.
  - **Ford-Fulkerson Algorithm**: In the worst case, the Ford-Fulkerson algorithm has a time complexity of $O(m \cdot f)$, where m is the number of edges in the graph and f is the maximum flow value. In our context, m is bounded by $O(n^2)$ and f is at most $O(n)$. Therefore, the time complexity of the Ford-Fulkerson algorithm is $O(n^3)$.
  - **Counting Nested Boxes**: Counting the number of nested boxes involves iterating over the matching obtained from the Ford-Fulkerson algorithm. This operation has a time complexity of $O(n)$.
- **Space Complexity**:
  - **Bipartite Graph**: The space complexity for storing the bipartite graph is $O(n^2)$, as there can be at most $O(n^2)$ edges in the graph.
  - **Matching Dictionary**: The matching obtained from the Ford-Fulkerson algorithm is stored as a dictionary, which requires $O(n)$ space.

Overall, the algorithm has a time complexity of $O(n^3)$ and a space complexity of $O(n^2)$.

5 Pseudocode

Box Class

```
class Box:
    attributes: id, width, height, depth

    method can_nest_inside(other_box):
        if (self.width < other_box.width and
            self.height < other_box.height and
            self.depth < other_box.depth):
                return True
            return False
```

Construct Bipartite Graph

```
function CONSTRUCT_BIPARTITE_GRAPH(boxes)
    $U \leftarrow$ boxes
    $W \leftarrow$ boxes
    $graph \leftarrow \{\}$
    for $u$ in $U$ do
        $graph[u] \leftarrow []$
        for $w$ in $W$ do
            if $u$.can_nest_inside($w$) then
                $graph[u]$.append($w$)
            end if
        end for
    end for
    return graph
end function
```

Augment Path

```
function AUGMENT_PATH(graph, u, visited, matching)
    for $v$ in $graph[u]$ do
        if not $visited[v]$ then
            $visited[v] \leftarrow$ True
            if $matching[v]$ is None or AUGMENT_PATH($graph, matching[v], visited, matching$) then
                $matching[v] \leftarrow u$
                return True
            end if
        end if
    end for
    return False
end function
```

Find Maximum Matching

```
function FIND_MAXIMUM_MATCHING(graph)
    $matching \leftarrow \{v :$ None for $v$ in $graph\}$
    for $u$ in $graph$ do
        $visited \leftarrow \{v :$ False for $v$ in $graph\}$
        AUGMENT_PATH($graph, u, visited, matching$)
    end for
    return matching
end function
```

Maximum Nested Boxes

```
function MAXIMUM_NESTED_BOXES(boxes)
    $graph \leftarrow$ construct_bipartite_graph($boxes$)
    $matching \leftarrow$ find_maximum_matching($graph$)
    $nested\_boxes \leftarrow 0$
    for $v$ in $matching.values()$ do
        if $v$ is not None then
            $nested\_boxes \leftarrow nested\_boxes + 1$
        end if
    end for
    return $nested\_boxes$
end function
```

6 Proof of Correctness

Theorem :the maximum number of nested boxes equals the size of the maximum matching in the bipartite graph G. We provide a proof for this theorem as follows:

Proof: We begin by assuming we have knowledge of the maximum number of nested boxes and their containment relationships. This information can be translated into a maximum matching in the bipartite graph G by selecting an edge $e = (u, v)$ in G for each box u containing box v. This correspondence yields a valid matching, as each box can accommodate exactly one other box, ensuring consistency with the nesting relationships.

Conversely, given a maximum matching M, we can deduce the arrangement of the boxes by placing each box u into its matched box v for every edge $e = (u, v)$ in M. Since each box u is matched exactly once, each box can be nested inside another box only once, resulting in a valid solution. Furthermore, this matching configuration must represent the maximum number of nested boxes, as any alternative arrangement would result in a larger matching, contradicting the maximality of M. Therefore, the size of the maximum matching $|M|$ is equivalent to the number of nested boxes, as each edge in M corresponds to a nested box. This concludes the proof.