ADA-2024: Homework-5

Siddhant Bali (2022496) Rijusmit Biswas (2022400)

April 21, 2024

1 Problem Statement

Suppose you are given a set of boxes, each specified by their height, width, and depth in centimeters. All three side lengths of every box are strictly between 10 cm and 20 cm. One box can be placed in another if the first box can be rotated so that its height, width, and depth are respectively smaller than the height, width, and depth of the second box. Boxes can be nested recursively. Design an algorithm to nest the boxes so that the number of visible boxes is as small as possible.

2 Algorithm Description

To solve this problem, we'll construct a flow network where vertices represent boxes and edges represent nesting relationships. We'll then apply the Ford-Fulkerson algorithm using Breadth First Search at Queue implementation with relevance to find the maximum flow, which corresponds to the minimum number of visible boxes.

3 Recurrence Relation

Not applicable for this problem.

4 Complexity Analysis

- Time Complexity :

  Building the flow network takes $\mathcal{O}(n^2)$ time, where n is the number of boxes. Using the Ford-Fulkerson algorithm to find the maximum flow adds $\mathcal{O}(f \cdot (|V| + |E|))$ complexity, where f is the maximum flow, and $|V|$ and $|E|$ are the number of vertices and edges, respectively. So, overall, it's $\mathcal{O}(n^2 + f \cdot (|V| + |E|))$.

5 Pseudocode

6 Proof of Correctness

We construct a flow network where each vertex represents a box, and there is an edge from vertex i to vertex j if box i can be nested inside box j. We also add a source vertex s with edges to all box vertices, and a sink vertex t with edges from all box vertices.

The maximum flow in this network represents the maximum number of boxes that can be nested. This is because each unit of flow corresponds to a box being nested inside another box.

Therefore, the minimum number of visible boxes is equal to the total number of boxes minus the maximum flow value. This is because the boxes that are nested inside other boxes are not visible.

By using the Ford-Fulkerson algorithm to find the maximum flow in the constructed network, we can correctly compute the minimum number of visible boxes for the given set of boxes.

Algorithm 1 NestBoxes

function NestBoxes($boxes$)
    $n \leftarrow$ length of $boxes$
    Initialize $capacity$ matrix with all zeros
    $source \leftarrow 0$
    $sink \leftarrow 2 \times n + 1$
    for $i \leftarrow 1$ **to** n **do**
        $capacity[source][i] \leftarrow 1$
        $capacity[i][i + n] \leftarrow 1$
        for $j \leftarrow 1$ **to** n **do**
            if $i \neq j$ **and** canBeNested($boxes[i - 1]$, $boxes[j - 1]$) **then**
                $capacity[i + n][j] \leftarrow 1$
            end if
        end for
    end for
    $maxFlow \leftarrow$ FordFulkerson($capacity$, $source$, $sink$, $2 \times n + 2$)
    return $n -$ maxFlow
end function

Algorithm 2 canBeNested

function canBeNested(A, B)
    return ($A.height < B.height$ **and** $A.width < B.width$ **and** $A.depth < B.depth$)
end function

Algorithm 3 FordFulkerson

function FordFulkerson($capacity$, $source$, $sink$, n)
    $maxFlow \leftarrow 0$
    Initialize $parent$ array
    while bfs($capacity$, $source$, $sink$, n, $parent$) **do**
        $pathFlow \leftarrow \infty$
        for $v \leftarrow sink$ **to** $source$ **do**
            $u \leftarrow parent[v]$
            $pathFlow \leftarrow \min(pathFlow, capacity[u][v])$
        end for
        for $v \leftarrow sink$ **to** $source$ **do**
            $u \leftarrow parent[v]$
            $capacity[u][v] \leftarrow capacity[u][v] - pathFlow$
            $capacity[v][u] \leftarrow capacity[v][u] + pathFlow$
        end for
        $maxFlow \leftarrow maxFlow + pathFlow$
    end while
    return $maxFlow$
end function

Algorithm 4 bfs

```
function BFS(capacity, source, sink, n, parent)
    Initialize visited array with false values
    Create an empty queue
    Enqueue source into the queue
    visited[source] ← true
    parent[source] ← −1
    while queue is not empty do
        u ← Dequeue from the queue
        for v ← 0 to n − 1 do
            if not visited[v] and capacity[u][v] > 0 then
                Enqueue v into the queue
                parent[v] ← u
                visited[v] ← true
            end if
        end for
    end while
    return visited[sink]
end function
```