Theory Assignment-5: ADA Winter-2024

Devansh Grover (2022151) Garvit Kochar (2022185)

1 Pre-processing

Given a set of boxes with dimensions between 10 cm and 20 cm for each side, we aim to minimize the number of visible boxes by nesting them. A box can fit inside another if it can be rotated to have smaller dimensions in height, width, and depth. This nesting can be done recursively. A box is considered ***visible*** if it is not inside another box. Our task is to design an algorithm to achieve this **minimization of visible boxes.**

To do this, we will first initialize a structure called ***Box*** which will take the 3 dimensions as input to the constructor to return the ***Box*** object, while also having a method that takes as input 2 ***Box*** objects and checks whether the first one can fit into the second one. Given below is the pseudocode for the ***Box*** structure.

Algorithm 1 Pseudocode for the Box Structure

```
function BOX($h, w, d$)
    Constructor for Box
    return new Box with height h, width w, and depth d
end function

function CAN_FIT(Box a, Box b)
    //Checks if Box a can fit into Box b
    return $a.h < b.h \wedge a.w < b.w \wedge a.d < b.d$
end function
```

2 Our Approach

For formulating the problem into a flow network problem, we assume each box to be a node. We add an extra source node irrespective of how many boxes are present. Then, we connect that source node with all the nodes (boxes) available with us with edges directed from source to other nodes having capacity (edge weight) as 1 each. Thus, a directed graph is formed. Then, we run our algorithm which compares each box with another box and adds a directed edge from the node having smaller dimensions to the node having larger (not equal, just larger) dimensions with capacity (edge weight) of each edge as 1. We select the node which has no outgoing edges and only incoming edges (this node also happens to have maximum number of incoming edges) as our sink node from all available nodes (except the source). This way we formulate the problem into a network-flow problem. Also, we ignored direct edge from source to sink as it gives wrong answer while calculating max flow because even if there is no box that can fit into another it would have calculated max flow as 1 and given a wrong answer.

2.1 Constructing the Flow Network

We will first initalize an empty vector ***boxes*** to store the dimensions of each box. Let the total number of boxes be $\boldsymbol{n}$. For each box i, we prompt the user to input the dimensions (height, width, depth) of the box and then sort the dimensions in non-decreasing order. Add the box with sorted dimensions to the *boxes* vector. Finally, sort the *boxes* vector based on the product of dimensions (height $\times$ width $\times$ depth).
We then create a 2D vector ***graph*** representing the flow network with $n + 2$ nodes and initialize all edges to 0. Then, add a source node connected to all other nodes (boxes) with edges of capacity 1. For each pair of boxes, if

one box can fit on top of another, add a directed edge between them with capacity 1 in the graph. Identify the sink node as the box with no outgoing edges and the maximum number of incoming edges.

2.2 Ford-Fulkerson Algorithm

We will be using the Ford-Fulkerson algorithm to find the maximum flow in the constructed flow network. To do this, we first initialize the **residual graph** *rGraph* with capacities same as the original graph. While there exists an augmenting path from the source to the sink:

- Find the path flow, which is the minimum capacity of edges along the augmenting path.
- Update the residual capacities of edges in the augmenting path.
- Update the maximum flow by adding the path flow.
- Calculate the minimum number of visible boxes as n minus the maximum flow value.
- If the minimum number of visible boxes is 0, increment it by 1.

The algorithm thus outputs the minimum number of visible boxes as the final result by subtracting max flow from number of boxes.

3 Justification

Max Flow/ Min Cut corresponds to the actual answer to the problem.

We already know that $MaxFlow = MinCut$. We will henceforth be using Max Flow to refer to both. In the context of this problem, finding the maximum flow corresponds to maximizing the number of boxes that can be stacked while still being visible. This is because each unit of flow from the source to the sink represents one box that is placed on top of another. Therefore, the maximum flow value indicates the maximum number of boxes that can be stacked without any box being completely covered. By selecting the sink node as a box with no outgoing edges and the maximum number of incoming edges, we ensure that the maximum flow calculation considers the possibility of leaving as many boxes unstacked as possible. This selection strategy helps in optimizing the flow calculation and ensures that the maximum flow corresponds to the maximum number of visible boxes. Ignoring the direct edge from the source to the sink node is crucial because it could lead to incorrect results. Including this edge would falsely increase the maximum flow by one, as it would allow for an additional box to be stacked without considering the constraints of other boxes. So, we are also ignoring the direct edge from source to sink. Thus, finding the max flow will help in reaching the answer as it demonstrates the number of boxes that can be put inside one another and when we subtract max flow from total number of boxes we get visible boxes as the answer.
More formally, Let $G = (V, E)$ be the directed graph representing the problem, where V is the set of nodes and E is the set of edges.
Let s be the source node and t be the sink node.
Define $c(u, v)$ as the capacity of the edge from node u to node v.
The problem can be formulated as a max-flow (min-cut) problem:

1. **Max-Flow Formulation**:
   - **Objective**: Maximize the flow from the source node s to the sink node t.
   - **Decision Variables**: Let $f(u, v)$ denote the flow through the edge from node u to node v.
   - **Constraints**:
     (a) **Capacity constraint**: $0 \leq f(u, v) \leq c(u, v)$ for all $(u, v) \in E$.
     (b) **Flow conservation**: For each node u except the source and sink, the sum of incoming flow equals the sum of outgoing flow: $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$.
   - **Maximize**: $\sum_{v \in V} f(s, v)$

2. **Conversion to the Problem Statement**:

   - The maximum flow represents the maximum number of boxes that can be stacked while still being visible.
   - Each unit of flow from the source to the sink represents one box that is placed on top of another.
   - To ensure maximum visibility, we select the sink node t as a box with no outgoing edges and the maximum number of incoming edges.
   - Ignoring the direct edge from the source to the sink node prevents falsely increasing the maximum flow by one, as it would allow for an additional box to be stacked without considering the constraints of other boxes.
   - Therefore, the number of visible boxes is given by Total number of boxes − Maximum flow.

   Mathematically, this can be represented as:

$$\text{Number of visible boxes} = \text{Total number of boxes} - \text{Maximum flow}$$

So, the problem is effectively solved by finding the maximum flow in the graph G, where s is the source node, t is the sink node, and the edges represent the constraints and capacities of the boxes.

4 Algorithm Description

1. Box Structure: As mentioned in the pre-processing part, the Box struct is defined with height, width, and depth as its properties. A method can_fit checks if one box can fit into another based on their dimensions.

2. BFS: This is a standard Breadth-First Search function which is used to find if there is a path from source to sink in the residual graph. It also fills parent[] to store the path.

3. Ford-Fulkerson Algorithm: This function is the implementation of the Ford-Fulkerson algorithm which returns the maximum flow in a graph from a source to a sink node. It uses the BFS function to check if more flow is possible and to find the path.

To be done in main: The main function is the driver function. It does the following:
1. Reads the number of boxes and their dimensions. Sorts the boxes based on their volumes.
2. Constructs a graph where each box is a node and there is an edge from box i to j if box i can fit into box j.
3. Finds a sink node which has the maximum incoming edges and no outgoing edges.
4. Applies the Ford-Fulkerson algorithm to find the maximum flow from the source to the sink node in the graph.
5. The minimum number of visible boxes is then calculated as the total number of boxes minus the maximum flow.

This algorithm assumes that one box can be put into another if its dimensions are strictly smaller. The boxes are sorted by volume to ensure that a box only needs to check the boxes that come after it when constructing the graph. The Ford-Fulkerson algorithm is then used to find the maximum number of boxes that can be put into each other, and this number is subtracted from the total number of boxes to find the minimum number of visible boxes. If no valid sink node is found, the program outputs a message and terminates. If there is a direct path from source to sink, it is ignored. If the answer is zero after applying the Ford-Fulkerson algorithm, it is incremented by one to ensure that at least one box is visible. The result is then printed out.

Note: We are using 0-based indexing here.

Algorithm 2 Pseudocode for BFS

```
function BFS(rGraph, s, t, parent)
    $V \leftarrow$ size($rGraph$)
    $visited \leftarrow$ vector($V$, false)
    $q \leftarrow$ queue()
    $q$.push($s$)
    $visited[s] \leftarrow$ true
    $parent[s] \leftarrow -1$
    while $\neg q$.empty() do
        $u \leftarrow q$.front()
        $q$.pop()
        for $v \leftarrow 0$ to $V - 1$ do
            if $\neg visited[v] \wedge rGraph[u][v] > 0$ then
                $q$.push($v$)
                $parent[v] \leftarrow u$
                $visited[v] \leftarrow$ true
            end if
        end for
    end while
    return $visited[t]$
end function
```

Algorithm 3 Pseudocode for Ford Fulkerson Algorithm

```
function FORDFULKERSON(graph, s, t)
    $V \leftarrow$ size($graph$)
    $rGraph \leftarrow$ vector($V$, vector($V$))
    for $u \leftarrow 0$ to $V - 1$ do
        for $v \leftarrow 0$ to $V - 1$ do
            $rGraph[u][v] \leftarrow graph[u][v]$
        end for
    end for
    $parent \leftarrow$ vector($V$)
    $max\_flow \leftarrow 0$
    while BFS(rGraph, s, t, parent) do
        $path\_flow \leftarrow$ INT_MAX
        for $v \leftarrow t$; $v \neq s$; $v \leftarrow parent[v]$ do
            $u \leftarrow parent[v]$
            $path\_flow \leftarrow \min(path\_flow, rGraph[u][v])$
        end for
        for $v \leftarrow t$; $v \neq s$; $v \leftarrow parent[v]$ do
            $u \leftarrow parent[v]$
            $rGraph[u][v] -= path\_flow$
            $rGraph[v][u] += path\_flow$
        end for
        $max\_flow += path\_flow$
    end while
    return $max\_flow$
end function
```

Algorithm 4 Pseudocode for Main a.k.a MinVisibleBoxes

```
function MinVisibleBoxes(n, boxes)
    graph ← vector($n + 2$, vector($n + 2$, 0))
    source ← $n + 1$
    for $i ← 0$ to $n - 1$ do
        graph[source][$i$] ← 1
        for $j ← i + 1$ to $n - 1$ do
            if can_fit(boxes[$i$], boxes[$j$]) then
                graph[$i$][$j$] ← 1
            else if can_fit(boxes[$j$], boxes[$i$]) then
                graph[$j$][$i$] ← 1
            end if
        end for
    end for
    sink ← $-1$
    max_incoming ← $-1$
    for $i ← 0$ to $n - 1$ do
        outgoing ← false
        incoming ← 0
        for $j ← 0$ to $n - 1$ do
            if graph[$i$][$j$] then
                outgoing ← true
            end if
            if graph[$j$][$i$] then
                incoming += 1
            end if
        end for
        if ¬outgoing ∧ incoming > max_incoming then
            sink ← $i$
            max_incoming ← incoming
        end if
    end for
    if sink = $-1$ then
        return "No valid sink node found."
    end if
    if graph[source][sink] then
        graph[source][sink] ← 0
    end if
    ans ← $n$ − fordFulkerson(graph, source, sink)
    if ans = 0 then
        ans += 1
    end if
    return "The minimum number of visible boxes is " + ans
end function
```

5 Correctness of the Algorithm

Flow Network Construction

The correctness of the algorithm begins with the proper construction of the flow network. Each box is represented as a node in the network, and directed edges between nodes represent the possibility of stacking one box on top of another. By ensuring that the source node is connected to all other nodes with edges of capacity 1, the algorithm ensures that each box can be considered as a potential candidate for stacking.
Furthermore, the algorithm accurately models the stacking possibilities between boxes by adding directed edges with capacity 1 between nodes where one box can fit on top of another. This step ensures that the flow network correctly captures the constraints of the problem, where boxes can only be stacked if their dimensions allow it.
The algorithm also correctly identifies the sink node as the box with no outgoing edges and the maximum number of incoming edges. This ensures that the sink node represents the optimal arrangement of boxes, where the minimum number of boxes are visible.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm is employed to find the maximum flow in the constructed flow network. This algorithm ensures that the flow is optimally routed through the network, maximizing the number of boxes that can be stacked while minimizing the number of visible boxes.
By iteratively finding augmenting paths from the source to the sink in the residual graph and updating the flow along these paths until no more augmenting paths exist, the algorithm guarantees that it calculates the maximum flow. This maximum flow value corresponds to the minimum number of visible boxes in the arrangement.

6 Running Time of the Algorithm

The running time of our algorithm is $O(n^3)$. It is determined by the following parts:

Construction of Flow Network

1. The construction of the flow network involves iterating over each box and comparing it with every other box to determine stacking possibilities.
2. The outer loop runs for n iterations, where n is the number of boxes. Within this loop, the inner loop also runs for n iterations in the worst case.
3. Therefore, the time complexity of constructing the flow network is $O(n^2)$.

Ford-Fulkerson Algorithm

1. The Ford-Fulkerson algorithm iteratively finds augmenting paths in the residual graph until no more paths exist.
2. Each iteration of the algorithm involves running a breadth-first search (BFS) to find an augmenting path, which has a time complexity of $O(|V| + |E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph.
3. Additionally, the maximum number of iterations the algorithm can perform is limited by the maximum flow value, denoted as f. 4. Therefore, the overall time complexity of the Ford-Fulkerson algorithm is $O(f(|V|+|E|))$.

Summary

1. The construction of the flow network takes $O(n^2)$ time.
2. The Ford-Fulkerson algorithm takes $O(f(|V| + |E|))$ time.
3. So, the overall time complexity can be considered to be $O(n^3)$ considering $O(f(|V| + |E|))$ as $O(n)$.