Theory Assignment-5: ADA Winter-2024

Vansh Yadav (2022559) Shamik Sinha (2022468)

1 Algorithm Description

Input: Number of boxes n, dimensions of each box (height,widht,depth)
Output: Minimum number of visible boxes

Our Approach:

1. **Input Reading and Initialization**:
   - Read the number of boxes n as input.
   - Initialize a vector boxes to store the dimensions of each box, where each box is represented by a vector of three integers (height, width, depth). Sort the dimensions of each box in non-decreasing order to ensure consistent representation for comparison.
2. **Constructing the Bipartite Graph**:
   - Initialize an adjacency matrix adjMat of size $(2n+2) \times (2n+2)$. This matrix represents the bipartite graph where nodes correspond to boxes (U_i for boxes that can be nested inside others and V_i for boxes that can contain others), as well as source (s) and sink (t) nodes.
   - Populate the adjacency matrix:
     - For each pair of boxes (i, j), check if box i can fit inside box j using the `can_fit` function and ensure $i \neq j$ to avoid self-loops. If box i can fit inside box j, create an edge with a capacity of 1 from the box node U_i (indexed by $i+1$) to the container node V_j (indexed by $n+j+1$).
     - Additionally, connect each box node U_i (from 1 to n) to the source node s with an edge of capacity 1, and connect each container node V_i (from $n+1$ to $2n$) to the sink node t with an edge of capacity 1. This ensures that each box and container node can potentially be a visible box.
3. **Implementing the Ford-Fulkerson Algorithm**:
   - Define a function `bfs` for Breadth-First Search (BFS) to find an augmenting path in the residual graph from the source s to the sink t, updating a parent vector to keep track of the path.
   - Implement the Ford-Fulkerson method in the `fordFulkerson` function to compute the maximum flow in the constructed flow network (`adjMat`). This function uses the BFS to find augmenting paths repeatedly and updates the residual capacities of the edges until no more augmenting paths can be found.
4. **Computing the Minimum Number of Visible Boxes**:
   - Calculate the maximum flow in the flow network using the `fordFulkerson` function with the constructed adjacency matrix adjMat.
   - Determine the minimum number of visible boxes as $n - \text{maxFlow}$, where maxFlow is the result from the maximum flow computation.
5. **Outputting the Result**:
   - Output the minimum number of visible boxes. If the computed result is 0, increment it by 1 to ensure a minimum of one visible box is considered.

The approach leverages graph theory and network flow algorithms to solve the box nesting optimization problem efficiently, aiming to minimize the number of visible boxes by maximizing the nesting relationships between boxes.

Constructing the Flow Network

Flow Network Construction

1. **Vertices (Nodes)**:
   - **Source (s)**: Represents the source node where flow begins.
   - **Sink (t)**: Represents the sink node where flow ends.
   - **Box Nodes (Ui)**: Nodes representing boxes that can potentially be nested inside others.
   - **Container Nodes (Vi)**: Nodes representing boxes that can contain other boxes.
2. **Edges**:
   - **Source to Box Nodes** (Ui): Create an edge from the source (s) to each box node (Ui) with a capacity of 1, indicating that each box can potentially be a visible box.
   - **Box Nodes (Ui) to Container Nodes (Vi)**: For each pair of box node (Ui) and container node (Vi) where box Ui can fit inside box Vi, create a directed edge from Ui to Vi with a capacity of 1, representing the possibility of nesting Ui inside Vi.
   - **Container Nodes (Vi) to Sink (t)**: Create an edge from each container node (Vi) to the sink (t) with a capacity of 1, indicating that each container can be a visible box (if not used to contain other boxes).
3. **Source and Sink:** We add two additional nodes: s (source) and t (sink). The source s is connected to all box nodes with edges of capacity 1, indicating that each box can potentially be visible. Similarly, each visibility node is connected to the sink t with an edge of capacity 1, indicating that each box can be a visible box.
4. **Internal Connections:** For each pair of boxes i and j, if i can fit inside j (based on the constraints), we add an edge from node i to node $j + n$ with capacity 1, indicating that box i can be nested inside box j.
5. **Flow Calculation:** The goal is to find the maximum flow from the source s to the sink t, which corresponds to the minimum number of visible boxes.
6. **Implementation:** A bipartite graph represented by an adjacency matrix adjMat based on a set of n boxes with specified dimensions. First, n boxes are initialized, and their dimensions are read into a vector boxes. Each box's dimensions are then sorted to ensure consistent representation for comparison. Next, the adjacency matrix adjMat is created with a size of $(2n + 2) \times (2n + 2)$, where $2n + 2$ accounts for the source (s), sink (t), and all box and container nodes. The matrix is populated by iterating through all pairs of boxes (i, j), and for each pair where box i can fit inside box j (determined by the can_fit function and ensuring $i \neq j$), an edge with a capacity of 1 is added from the corresponding box node U_i (indexed by $i+1$) to the container node V_j (indexed by $n+j+1$). Additionally, edges with a capacity of 1 are created from the source (s) to all box nodes (U_i) and from all container nodes (V_j) to the sink (t), ensuring each box and container node can potentially be a visible box. This construction sets up the bipartite graph to solve the box nesting problem using maximum flow algorithms.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm is used to find the maximum flow in a flow network. In the context of the problem, we use the Ford-Fulkerson algorithm to determine the minimum number of visible boxes, which corresponds to the maximum flow value in the flow network.
The algorithm works by finding augmenting paths from the source node to the sink node in the flow network. An augmenting path is a path from the source to the sink where each edge has residual capacity greater than 0. The flow along this path is then increased by the minimum residual capacity of the edges in the path.
In the provided code, the Ford-Fulkerson algorithm is implemented in the function `fordFulkerson`. It takes the adjacency matrix representing the flow network, the source node, and the sink node as input, and returns the maximum flow value.

The time complexity of the Ford-Fulkerson algorithm depends on the implementation of the augmenting path search. In the provided code, a breadth-first search (BFS) is used to find augmenting paths, resulting in a time complexity of $O(value(flow)(|V| + |E|))$, where $value(flow)$ is the maximum flow value, $|V|$ is the number of vertices, and $|E|$ is the number of edges in the flow network.

Justification

The concept of Max Flow and Min Cut corresponds directly to solving the box nesting problem using flow network theory. Max Flow in this context represents the maximum number of boxes that can be effectively nested inside others. Each unit of flow from the source to the sink in the flow network signifies one box being nested inside another, reflecting the capacity of the nesting relationships between boxes. The computation of maximum flow optimizes the nesting configuration to allow as many boxes as possible to be nested within larger containers, thereby maximizing space utilization. The value of max flow is constrained by the capacities of the edges, which dictate the feasible nesting relationships between boxes and containers. Therefore, finding the max flow demonstrates the maximum achievable nesting arrangement, ensuring the most efficient use of space and minimizing the number of visible boxes required in the optimal configuration. The Min Cut in the flow network, which separates the source from the sink after computing max flow, directly translates to the minimum number of visible boxes needed. This approach leverages graph theory principles to provide a systematic solution for optimizing box nesting and minimizing visibility constraints.
Formally, the problem can be represented as a max-flow (min-cut) problem in a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. The source node s and the sink node t are defined, and $c(u, v)$ represents the capacity of the edge from node u to node v. The objective is to maximize the flow from the source node s to the sink node t, subject to the capacity constraints and flow conservation constraints. The maximum flow value corresponds to the maximum number of visible boxes, and the number of visible boxes is calculated as Total number of boxes - Maximum flow.

Time Complexity Analysis

Construction of Flow Network

- The construction of the flow network involves iterating over each box and comparing it with every other box to determine stacking possibilities.
- 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.
- Therefore, the time complexity of constructing the flow network is $O(n^2)$.

Ford-Fulkerson Algorithm

- The Ford-Fulkerson algorithm iteratively finds augmenting paths in the residual graph until no more paths exist.
- 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.
- Additionally, the maximum number of iterations the algorithm can perform is limited by the maximum flow value, denoted as f.
- Therefore, the overall time complexity of the Ford-Fulkerson algorithm is $O(f(|V| + |E|))$.

In summary:

We used the Ford-Fulkerson algorithm without modification to find the maximum flow. The time complexity of the Ford-Fulkerson algorithm is $O(\text{value(flow)} \cdot (|V| + |E|))$. In the constructed graph, $|V| = 2n + 2$. In

the worst case, $|E| = n^2$ (as each box can potentially fit inside every other box). Therefore, the total time complexity is:

$$O(n \cdot (n + n^2)) = O(n^3)$$

2 PseudoCode

Algorithm 1 BFS Function

```
function BFS(residualAdj, s, t, parent)
    V ← size(residualAdj)
    visited ← vector(V, false)
    q ← queue()
    q.push(s)
    visited[s] ← true
    parent[s] ← −1
    while ¬q.empty() do
        u ← q.front()
        q.pop()
        for v ← 0 to V − 1 do
            if ¬visited[v] ∧ residualAdj[u][v] > 0 then
                q.push(v)
                parent[v] ← u
                visited[v] ← true
            end if
        end for
    end while
    return visited[t]
end function
```

Algorithm 2 Ford-Fulkerson Algorithm for Maximum Flow

```
1:  function FordFulkerson(adjMat, s, t)
2:      V ← size of adjMat
3:      residualAdj ← initialize a new matrix of size V × V with zeros        ▷ Residual graph
4:      residualAdj ← adjMat                          ▷ Initialize residual graph with capacities
5:      parent ← array of size V initialized to −1          ▷ Parent array for path reconstruction
6:      maxFlow ← 0
7:      while BFS(residualAdj, s, t, parent) do
8:          pathFlow ← ∞                                   ▷ Initialize path flow to infinity
                                    ▷ Find the maximum flow in the augmenting path
9:          for v ← t to s by parent[v] do
10:             u ← parent[v]
11:             pathFlow ← min(pathFlow, residualAdj[u][v])      ▷ Find minimum capacity in the path
12:         end for
                                    ▷ Update residual capacities along the augmenting path
13:         for v ← t to s by parent[v] do
14:             u ← parent[v]
15:             residualAdj[u][v] ← residualAdj[u][v] − pathFlow          ▷ Update forward edge
16:             residualAdj[v][u] ← residualAdj[v][u] + pathFlow          ▷ Update backward edge
17:         end for
18:         maxFlow ← maxFlow + pathFlow                ▷ Update max flow with path flow
19:     end while
20:     return maxFlow
21: end function
```

Algorithm 3 Main Function

```
1:
2: function canFit(a, b)
3:     return a[0] < b[0] and a[1] < b[1] and a[2] < b[2]
4: end function
5:
6: function main
7:     n ← input from user
8:     boxes ← vector of size n containing vectors of size 3
9:     for i from 0 to n − 1 do
10:        input boxes[i][0], boxes[i][1], boxes[i][2]
11:        sort boxes[i]
12:    end for
13:    adjMat ← vector of size 2n + 2 containing vectors of size 2n + 2
14:    for i from 0 to n − 1 do
15:        for j from 0 to n − 1 do
16:            if canFit(boxes[i], boxes[j]) and i ≠ j then
17:                adjMat[i + 1][n + j + 1] ← 1
18:            end if
19:        end for
20:    end for
21:    for i from 1 to n do
22:        adjMat[0][i] ← 1
23:        adjMat[i + n][2n + 1] ← 1
24:    end for
25:    ans ← n− fordFulkerson(adjMat, 0, 2n + 1)
26:    if ans == 0 then
27:        ans ← ans + 1
28:    end if
29:    output "Minimum number of visible boxes is ans"
30: end function
```

3 Correctness of the Algorithm

To demonstrate the correctness of the algorithm for minimizing the number of visible boxes through a flow network approach, we need to establish two key aspects: the construction of the flow network and the computation of the maximum flow.

Flow Network Construction

The algorithm constructs a flow network to model the box nesting problem. Each box is represented as a node in the graph, and directed edges with capacities represent potential nesting relationships between boxes. Specifically:

- For each pair of boxes i and j, the algorithm checks if box i can fit inside box j using the `can_fit` function. If this condition holds true, an edge with capacity 1 is added from the corresponding box node U_i to container node V_j.
- Additionally, edges are added from the source node s to each box node U_i and from each container node V_i to the sink node t with capacities 1, ensuring that each box and potential container can be considered in the optimal nesting configuration.

Computing Maximum Flow

The algorithm then applies the Ford-Fulkerson method to compute the maximum flow in the constructed flow network. This involves:

- Implementing the BFS (Breadth-First Search) algorithm to find augmenting paths from the source s to the sink t, updating a parent vector to track the path.
- Using the residual capacity graph to iteratively update flow values along the augmenting path until no more augmenting paths can be found.
- The value of the maximum flow obtained from the flow network computation corresponds to the maximum number of boxes that can be nested inside others, minimizing the number of visible boxes in the optimal configuration.

Correctness Assertion

The correctness of the algorithm hinges on the following assertions:

- The flow network accurately represents the constraints and relationships between boxes, ensuring that valid nesting configurations are modeled.
- The computation of maximum flow using the Ford-Fulkerson method correctly identifies the optimal nesting configuration that minimizes visibility while maximizing space utilization.
- Subtracting the maximum flow value from the total number of boxes provides the minimum number of visible boxes required in the optimal nesting arrangement, thus validating the effectiveness of the flow network approach for solving the box nesting problem.

Therefore, the algorithm's correctness is established through the construction of a valid flow network and the computation of maximum flow, leading to the determination of an optimal solution that minimizes the number of visible boxes in the nested configuration.