Theory Assignment-5: ADA Winter-2024

Aarzoo (2022008) Anushka Srivastava (2022086)

1 Input

The input is taken as an array of dimensions of the boxes in the form of (l, b, h), where the length, breadth, and height of the boxes are given in centimeters, and each of them is strictly between $10cm$ and $20cm$.

2 Objective

An algorithm to nest the boxes so that the number of visible boxes is as small as possible.

3 Construction of Flow Network

We create two vertices for each box and keep them in two different sets, A and B. To create a network flow for this graph, we create a directed edge uv between two boxes u and v, only if each dimension of v is less than each dimension of u, that is the box v can be nested inside box u, making box v invisible. Now we add a source vertex s and sink vertex t to the graph. We connect all the vertices in set A to s, and we connect all vertices in set B to t. We assign a capacity 1 to every directed edge in the graph.
By constructing the flow network using the steps given above, we ultimately reduce the question to a problem similar to that of *Maximum Bipartite Matching*.

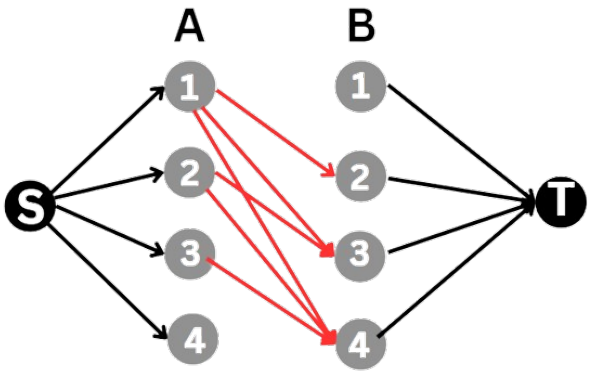

Figure 1: Example

In this figure, there are four boxes labeled (1-4). The red directed edges between any two vertices u and v indicate that v box can be placed inside u box.

4 Algorithm Description

After constructing the graph, we run a max flow algorithm to find out the max flow through the network. For this, we choose *Ford-Fulkerson Algorithm*.

1. The capacity of each edge is 1. We set the initial flow of each edge as 0.

2. We add edges of the residual graph of this graph.
3. We find an augmenting path between the *source* and the *sink* using the BFS algorithm and set the flow of every edge in the path equivalent to the minimum residual capacity along the edges of the path.
4. We subtract path flow from all edges along the path, and we add path flow along the reverse edges.
5. The above steps are repeated until an augmenting path does not exist between the *source* and the *sink*.
6. We simultaneously calculate the flow leaving the *source* vertex and the *target* vertex.

After *Ford-Fulkerson* has stopped running, the resulting network flow gives us the *Maximum Bipartite Matching* of the graph, where no two edges share the same endpoint, and the flow of each edge is either 0 or 1. This matching represents that if there is a matching between the box u and box v, box v can be nested in box u.

We get the max flow of the graph from the above algorithm, which is equivalent to the number of times we can put a box into a larger box. Following the path of the graph gives us the actual nested configuration. Every time a box is nested inside another box, the number of visible boxes reduces by 1. Hence, our final answer is represented by:

$$\text{minimum number of visible boxes} = \text{total number of boxes} - \text{max flow of the graph}$$

5 Proof of Correctness

(Explaining why Max-flow value/Min-Cut size would correspond to the actual value of the problem)

Claim: The maximum number of nested boxes is equal to $|E|$ where $|E|$ is the value of the number of maximum matching in the graph

According to our assumption, each box can contain exactly one box. Hence, this is a valid matching.
Now, we prove why the difference of total boxes and the max flow of the network is a valid solution to the problem.
Let $|E|$ be the final number of edges selected after running the *Ford-Fulkerson algorithm* on the graph. It is maximized as per the problem of *Maximum Bipartite Matching*. If there is a better way to arrange the boxes, then according to our algorithm, there will be a matching of a larger size, which contradicts our initial assumption. Hence, by contradiction, $|E|$ is a maximum matching, which is ultimately equal to the total number of nested boxes. Subtracting this number from the total number of boxes gives us the number of minimum visible boxes possible.

6 Complexity Analysis

6.1 Construction of Flow network

To construct an edge between any two boxes, we have to compare each box with every other box and create an edge if the second box is dimensionally smaller than the first box. This step takes $O(n^2)$ time.

6.2 Ford-Fulkerson Algorithm

The time complexity of the Ford-Fulkerson algorithm is $O(value\_flow * (|V| + |E|))$. In the worst case scenario, the max flow possible is $O(n)$ *i.e.* the number of boxes and the number of edges in the worst case scenario is $O(n^2)$, which evaluates to $O(n * n^2)$.

Thus, the overall time complexity is $O(n^3)$.

7 Pseudocode

Algorithm 1 Ford-Fulkerson Algorithm

```
1: function FordFulkerson(graph, source, sink)
2:     n ← size of graph
3:     edges ← empty list of Edge
4:     for u ← 0 to n − 1 do
5:         for each v in graph[u] do
6:             append Edge(u, v, 1, 0) to edges
7:         end for
8:     end for
9:     residual ← a n × n matrix filled with 0
10:    for each e in edges do
11:        residual[e.from][e.to] ← e.capacity
12:    end for
13:    parent ← a list of size n filled with −1
14:    maxFlow ← 0
15:    while true do
16:        fill parent with −1
17:        parent[source] ← −2
18:        q ← an empty queue of pairs of integers
19:        push (source, ∞) to q
20:        while q is not empty do
21:            (u, flow) ← front of q
22:            pop q
23:            for each v in graph[u] do
24:                if parent[v] = −1 and residual[u][v] > 0 then
25:                    parent[v] ← u
26:                    new_flow ← min(flow, residual[u][v])
27:                    if v = sink then
28:                        maxFlow += new_flow
29:                        while v ≠ source
30:                        u ← parent[v]
31:                        residual[u][v] −= new_flow
32:                        residual[v][u] += new_flow
33:                        v ← u
34:                        end while
35:                        goto found_path
36:                    end if
37:                    push (v, new_flow) to q
38:                end if
39:            end for
40:        end while
41:        break
42:        found_path:
43:    end while
44:    return maxFlow
45: end function
```

Algorithm 2 Build Flow Network

```
function BuildFlowNetwork(boxes)
    n ← size of boxes
    source ← 0
    sink ← n × 2 + 1
    graph ← a list of n × 2 + 2 empty lists
    for i ← 1 to n do
        append i to graph[source]
    end for
    for i ← n + 1 to n × 2 do
        append sink to graph[i]
    end for
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            can_nest ← true
            for k ← 0 to 2 do
                if boxes[j][k] ≥ boxes[i][k] then
                    can_nest ← false
                    break
                end if
            end for
            if can_nest then
                append j + n + 1 to graph[i + 1]
            end if
        end for
    end for
    return graph
end function
```

Algorithm 3 Main Function

```
function Main
    n ← input from user
    boxes ← a list of n empty lists
    for i ← 0 to n − 1 do
        l, b, h ← input from user
        append l, b, h to boxes[i]
    end for
    graph ← BuildFlowNetwork(boxes)
    source ← 0
    sink ← n × 2 + 1
    maxFlow ← FordFulkerson(graph, source, sink)
    min_visible_boxes ← n − maxFlow
    output "Minimum number of visible boxes: " concatenated with min_visible_boxes
end function
```

8 References

- **Math Stack Exchange** - Minimize the number of visible boxes
- **Github** - iveney
- **GeeksForGeeks** - Ford-Fulkerson Algorithm