ADA Theory Homework-5

Aditya Kumar (2022033)   Nishant Kumar (2022326)

April 18, 2024

1 Algorithm Description

1.1 Formulating the Problem into a Flow Network:

We can build the flow network with the vertices representing individual boxes and edges indicating the potential nesting relationship between them. Introducing a source vertex s to denote the external space and t (sink vertex) to signify the final arrangement where all visible boxes must culminate. We can construct the network flow by the following manner:

- We start by adding vertices for each box, as well as vertices for the source (denoted as s) and sink (denoted as t).
- From the source vertex s, we add edges to each box vertex with a capacity of 1. This represents the option for each box to either remain visible or not.
- For every pair of boxes i and j, where box i can potentially be nested inside box j, we establish an edge from vertex i to vertex j with a capacity of 1.
- Finally, we attach edges from each box vertex to the sink vertex t with a capacity of 1, indicating the possibility of the box being visible.
- In summary, this flow network effectively captures the nesting relationships between boxes and enables the determination of the maximum number of nested boxes i.e. minimize the number of visible boxes as small as possible. while ensuring that each box is either visible or nested within another box.

1.2 Justification for Maximum Flow/Minimum Cut Correspondence:

The maximum flow value obtained from the source vertex s to the sink vertex t signifies the maximum potential nesting of boxes within one another. This value delineates the upper limit of how many boxes can be nested, with each box having the option of either being directly visible or nested inside another box. The flow paths depict these two possibilities: a direct path from s to the box vertex and then to t signifies visibility, while a path starting from s, traversing through various boxes, and eventually reaching t indicates nesting within other boxes. The capacity constraints within the flow network ensure that each box is either visibly connected to the sink or nested within another box, but not both simultaneously. Consequently, the minimum cut size, representing the minimum number of boxes directly linked to t and hence visible, aligns with the minimum number of visible boxes required. In essence, the maximum flow value and minimum cut size encapsulate the trade-off between visibility and nesting, providing insights into the optimal arrangement of boxes to maximize space utilization while maintaining visibility.

1.3 Time Complexity Analysis:

- Constructing Vertices: The process of creating vertices for each box takes $O(n)$ time, where n represents the number of boxes.
- Adding Edges from s to Box Vertices: $O(n)$.

- Adding Edges between Box Vertices: This step involves checking every pair of boxes for nesting possibility, resulting in a time complexity of $O(n^2)$.
- Adding Edges from Box Vertices to t: $O(n)$.
- Overall Time Complexity for Constructing the Flow Network: $O(n^2)$.

Ford-Fulkerson Algorithm:

- The Ford-Fulkerson algorithm's time complexity is $O(\text{value(flow)} \times (|V| + |E|))$.
- In this scenario, the maximum flow value is at most n (since each box can either be visible or nested inside another box).
- With $O(n)$ vertices and $O(n^2)$ edges, the Ford-Fulkerson algorithm's time complexity becomes $O(n^3)$.

Therefore, the dominant time complexity is $O(n^3)$, resulting in an overall time complexity of $O(n^3)$ for the algorithm.

1.4 Space Complexity Analysis:

Input Reading:

- Space complexity: $O(n)$
  Requires storage for n, the number of boxes.

Edge Structure:

- Space complexity: $O(1)$ per edge
  The Edge struct stores information for each edge, including the source vertex, destination vertex, capacity, and current flow. Since the number of edges is determined by the number of boxes and their relationships, the space complexity for storing edges can be considered constant relative to n.

Adjacency List:

- Space complexity: $O(n^2)$
  The vector `adj` is used to store the adjacency list representation of the flow network. In the worst case, each vertex can have edges to every other vertex, leading to $O(n^2)$ space complexity.

Queue in BFS:

- Space complexity: $O(n)$
  The queue in the BFS function stores at most n elements, where n is the number of vertices in the flow network.

Parent Vector in BFS:

- Space complexity: $O(n)$
  The parent vector in the BFS function stores the parent edges for each vertex in the augmenting path. Requires storage for n integers.

Edges Vector:

- Space complexity: $O(k)$
  The edges vector stores information for each edge, where k is the total number of edges in the flow network. The number of edges depends on the number of boxes and their relationships.

Flow Network Construction:

- Space complexity: $O(k)$
  The space required for constructing the flow network includes the space for the edges vector and the adjacency list.

Total Space Complexity: The overall space complexity is the sum of space requirements for all components:

$$O(n) + O(1) + O(n^2) + O(n) + O(n) + O(k) + O(k)$$

Simplifying, the dominant term is $O(n^2)$ due to the adjacency list representation.

In summary, the space complexity of the provided code is $O(n^2)$ due to the adjacency list representation of the flow network.

2 Pseudo Code

```
Declare INF (very large integer constant)

Define Edge : structure
    u, v, cap, flow

Declare int: n
Declare a 2D matrix: adj
Declare a matrix of Edge objects: edges


function bfs(s, t, parent):
    Initialize parent array with -1
    Set parent[s] to -2
    Create a queue q
    Push (s, INF) into q

    while q is not empty:
        Dequeue a pair (cur, flow) from q
        For each index idx in adj[cur]:
            Access the corresponding Edge object e from edges[idx]
            If parent[e.v] is -1 and e.cap > e.flow:
                Set parent[e.v] to idx
                Calculate new_flow as min(flow, e.cap - e.flow)
                If e.v is equal to t:
                    Return new_flow
                Push (e.v, new_flow) into q

    Return 0

function max_flow(s, t):
    Initialize flow to 0
    Create a vector parent of size n and initialize it with zeros

    while True:
        Call bfs(s, t, parent) and assign the result to new_flow
        If new_flow is 0
            break
        Increment flow by new_flow
        Set cur to t
        While cur is not equal to s:
            Access the index of the parent of cur and assign it to prev
            Increment the flow of edges[prev] by new_flow
            Decrement the flow of edges[prev XOR 1] by new_flow
            Set cur to the u attribute of edges[prev]

    Return flow

function main():
    Read input n
    Create a 2D matrix of size n: boxes

    For i from 0 to n-1:
        Read dimensions of box i and store them in boxes[i]
```

```
Resize adj to have n+2 rows

Initialize s to n and t to n+1

For i from 0 to n-1:
    Append a new Edge object (s, i, 1) to edges
    Append the index of the last element of edges to adj[s]
    Append the index of the last element of edges to adj[i]
    Append a new Edge object (i, t, 1) to edges
    Append the index of the last element of edges to adj[i]
    Append the index of the last element of edges to adj[t]

For i from 0 to n-1:
    For j from 0 to n-1:
        If dimensions of box i are strictly less than dimensions of box j:
            Append a new Edge object (i, j, 1) to edges
            Append the index of the last element of edges to adj[i]
            Append the index of the last element of edges to adj[j]

Compute max_flow_val by calling max_flow(s, t)

Print "Minimum number of visible boxes:", n - max_flow_val
```

Credit

This assignment was completed by Nishant Kumaar and Aditya Kumar, who contributed to the development and analysis of the algorithm. Their collaborative efforts have resulted in a comprehensive solution to the problem at hand. We utilized few online resources for research and guidance, including:

- **GeeksforGeeks - Box Stacking Problem** https://www.geeksforgeeks.org/box-stacking-problem-dp-22/
- **stackexchange - Max - Flow and Min - Cut** https://math.stackexchange.com/questions/361685
- **Chegg -** https://www.chegg.com/homework-help/questions-and-answers/problem-q171114589

These resources provided valuable insights and knowledge that aided in the successful completion of the assignment.