Theory Assignment-5: ADA Winter-2024

Rachit Arora (2022384) Mahi Mann (2022272)

1 Assumptions

- A box B_2 can only be placed inside a box B_1 if dimensions of B_2 can be rearranged to be **strictly smaller** than dimensions of B_1. If not, some trivial minor changes can be made to the implementation.
- **A box can only have at most 1 box immediately inside it.** For example, a $3\times3\times3$ box cannot contain both 2 boxes of $1 \times 1 \times 1$ immediately inside it. However, a box can contain upto 1 box which has other boxes inside it. This assumption is necessary so that constructing the auxiliary graph is polynomial-time. Otherwise, the problem becomes NP-hard.
- To see whether the dimensions of a box B_i, $d_i = (x_i, y_i, z_i)$ are smaller than those of a box B_j with dimensions $d_j = (x_j, y_j, z_j)$, it suffices to check **the sorted (increasing order) permutation** (x, y, z) **of** d_i and check that $x < x_j$, $y < y_j$ and $z < z_j$.

2 Formulation as a Network Flow problem

Let there be n boxes $B_1, B_2, \cdots, B_n$. with dimensions $d_1 = (x_1, y_1, z_1), d_2 = (x_2, y_2, z_2), \cdots, d_n$.

We want to minimise $k =$ number of visible boxes.

k can also be expressed as $n - v$ where v is the number of invisible boxes.

Thus, we have an equivalent problem of **maximising v, the number of invisible boxes.**

(Of course, every edge has weight 1).

We know the following:

1. Suppose we are putting box B_j inside B_i. Then, this operation reduces the number of visible boxes by 1 (see Theorem 1 in Proof of Correctness section) and increases the number of invisible boxes by 1.
2. So, in a sense, we need to maximise the number of "outside box"-"inside box" pairs, since every "inside box" corresponds to an invisible box.
3. Every invisible box is inside 1 visible box.
4. Every visible box has at most 1 invisible box (which may be nested) immediately inside it.

These points clearly suggest the idea of a **bipartite graph matching**, where one partition signifies visible boxes and the other partition signifies invisible boxes.

Let us construct an auxiliary **directed** graph $G = (W \cup I, E')$ where $W = \{W_1, W_2, W_3, \cdots, W_n\}$ is the set of nodes corresponding to the visible vertices, and $I = \{I_1, I_2, I_3, \cdots, I_n\}$ is the set of nodes corresponding to invisible vertices.

To construct the edge set E', we will do the following:

1. For all $i = 1, 2, 3, \cdots, n$:
   (a) For all j in $\{1, 2, 3, \cdots, n\} - \{i\}$
      i. If B_j can fit inside B_i, make an edge from W_i to I_j.

Clearly, G is bipartite with partition (W, I).

The maximum number of invisible boxes will be the size of a maximum matching in G.

The size of a maximum matching of G can be computed as the maximum (s,t)-flow value of another auxiliary graph G' which is a copy of G, including 2 new vertices s and t, containing all possible edges from s to vertices in W, and all possible edges from all vertices in I to t.

This flow can be computed using the **Ford-Fulkerson Algorithm on G'**, which eventually gives us v, and $k = n - v$**, the minimum number of visible boxes, will be our final answer.**

3 Why Max Flow corresponds to answer

The max flow produced in G' will create a maximum bipartite matching in G.

Consider every edge $(w \to i)$ in a maximum matching of G where $w \in W$and $i \in I$.

$w = W_a$ and $i = I_b$ for some $a, b \in \{1, 2, 3, \cdots, n\}$.

Then, we'll put box B_b immediately inside box B_b.

Because this is a matching, B_b will be inside exactly 1 box and B_a will have exactly one box immediately inside it.

We'll repeat this for each such edge in the matching, yielding us an arrangement of boxes with maximum invisible and thus minimum visible boxes.

4 Complexity Analysis (Using Ford-Fulkerson's Algorithm)

To solve this Max-Flow problem, we are running the Ford-Fulkerson Algorithm on our auxiliary bipartite graph construction $G' = (V, E)$ where $|V| = O(n)$ and $|E| = O(n^2)$ (see Proof of Correctness section for cardinality of E).

As will be proved shortly, the maximum possible flow in this graph is also the maximum number of invisible boxes, which is $value(flow) = n - 1 = O(n)$.

Ford-Fulkerson's Algorithm runs in $O(value(flow)(|V| + |E|))$ time, **therefore our algorithm runs in**

$$\begin{aligned} &O\Big(value(flow)\left(|V| + |E|\right)\Big) \\ &= O(value(flow))O(|V| + |E|) \\ &= O(n)O(n + n^2) \\ &= O(n)O(n^2) \\ &= O(n^3) \end{aligned}$$

$O(n^3)$ **time where n is the number of boxes.**

5 Proof of Correctness

Theorem 1: Putting a box inside another box l times (or equivalently, making l edges in G) increases number of invisible boxes by l.

Proof: Induction on l.

For $l = 0$, number of invisible boxes created is 0.

For $l = k + 1$ for some k, Let the l^{th} edge be putting box B_j inside box B_i. By inductive hypothesis, the current number of invisible boxes is k. Then, adding this edge makes only box B_j invisible, thus making the count of invisible boxes $k + 1$.

Theorem 2: Number of edges in G' is $O(n^2)$

Proof: Let $G' = (V, E)$.

$V = \{s, t\} \cup W \cup I$. Thus, $|V| = 2n + 2 = O(n)$.

$|E| \leq \frac{|V|(|V|-1)}{2} = O(n^2)$

Addendum 1: C++ implementation

```
#include <bits/stdc++.h>

using namespace std;

// checking whether dimensions fit inside
bool ismore(vector<int>& mr, vector<int>& ls){
    for(int i = 0; i < 3; ++i){
        if(mr[i] <= ls[i])
            return false;
    }
    return true;
}

// Ford-Fulkerson algorithm implementation from cp-algorithms.com

int bfs(int source, int target, vector<int>& parent, vector<vector<int>>& edgeWeight,
↪   vector<vector<int>>& adjacencyList) {
    fill(parent.begin(), parent.end(), -1);
    parent[source] = -2;
    queue<pair<int, int>> q;
    q.push({source, INT_MAX});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adjacencyList[cur]) {
            if (parent[next] == -1 && edgeWeight[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, edgeWeight[cur][next]);
                if (next == target)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int fordFulkerson(int source, int target, int n, vector<vector<int>>& edgeWeight,
↪   vector<vector<int>>& adjacencyList) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(source, target, parent, edgeWeight, adjacencyList)) {
        flow += new_flow;
        int cur = target;
        while (cur != source) {
            int prev = parent[cur];
            edgeWeight[prev][cur] -= new_flow;
            edgeWeight[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}

int main(){
    int n;
    cin >> n;

    vector<vector<int>> d(n+1);

    for(int i = 0; i < n; ++i){
        d[i+1] = vector<int>(3,0);
        // inputting every dimension
        cin >> d[i+1][0] >> d[i+1][1] >> d[i+1][2];
        // sorting them because order doesn't matter
        sort(d[i+1].begin(), d[i+1].end());
    }

    // 0 is s
    // 1, 2, .... n are W
    // n+1, n+2, ... 2n are I
    // 2n + 1 is t


    vector<vector<int>> adjacencyList(2*n + 2);
    vector<vector<int>> capacity(2*n + 2, vector<int>(2*n + 2, 0));

    // adding all source edges and edges to target

    for(int i = 1; i <= n; ++i){
        adjacencyList[0].push_back(i);
        capacity[0][i] = 1;
        adjacencyList[n + i].push_back(2*n + 1);
        capacity[n+i][2*n + 1] = 1;
    }

    // adding all dimension edges

    for(int i = 1; i <= n; ++i){
        for(int j = 1; j <= n; ++j){
            if(ismore(d[i], d[j])){
                adjacencyList[i].push_back(n + j);
                capacity[i][n+j] = 1;
            }
        }
    }

    cout << n - fordFulkerson(0, 2*n + 1, 2*n + 2, capacity, adjacencyList) << '\n';

}
```

Addendum 2: Sample Tests

5.1 Input Format

First line contains n, the number of boxes.

Next n lines contain x, y and z, dimensions for each box.

5.2 Test 1

Input

```
4
1 1 1
1 1 1
1 1 1
1 1 1
```

Output

```
4
```

Explanation

All boxes of equal dimension, thus none can be contained in each any other, and all are visible.

5.3 Test 2

Input

```
6
6 6 6
5 5 5
4 4 4
3 3 3
2 2 2
1 1 1
```

Output

```
1
```

Explanation

Box 1 contains box 2
Box 2 contains box 3
and so on.
Only box 1 is visible.

5.4 Test 3

Input

```
8
10 11 8
7 9 10
9 6 8
7 5 8
12 13 1
13 14 2
14 15 3
15 16 4
```

Output

```
2
```

Explanation

Box 1 has box 2 inside it, which has box 3 inside it, which has box 4 inside it.
Box 8 has box 7 inside it, which has box 6 inside it, which has box 5 inside it.
So, boxes 2,3,4,5,6 and 7 are invisible.
Boxes 1 and 8 are visible.
Thus answer is 2

Addendum 3: People discussed with

Swapnil Panigrahi, Sahil Gupta and Yash Bhardwaj.

The Ford-Fulkerson Algorithm implementation was taken from the https://cp-algorithms.com website.

□□□□□
Thank you.