CS575 - Design and Analysis of Algorithm

Term Project Report

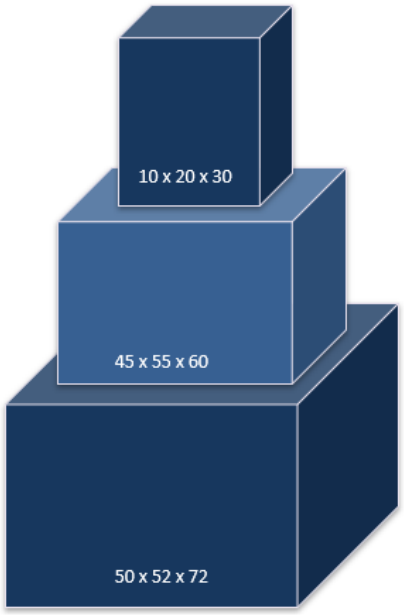
Name – Chaitanya Kulkarni **B-Number – B00184455**

Title of the Project –

THE BOX STACKING PROBLEM

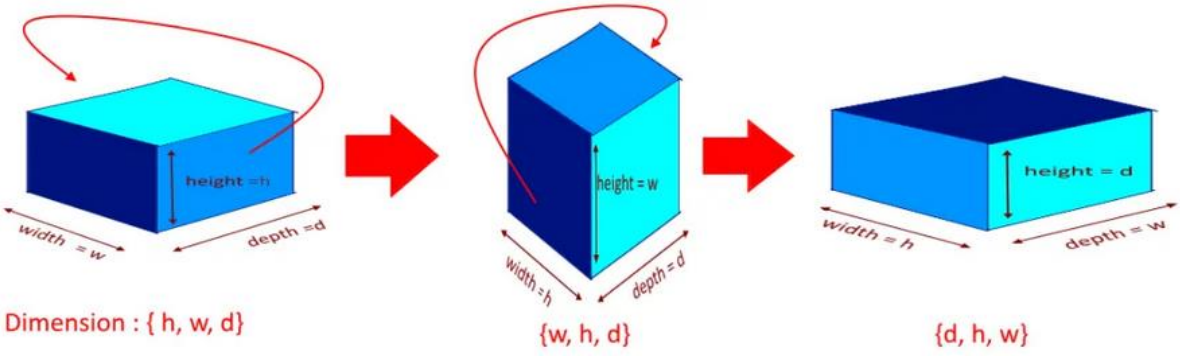
Problem Statement –

Given a set of n cuboid boxes with height(h), width(w) and depth(d). Create a stack of boxes which is as tall as possible, but a box can be placed on top of another box only if both width and depth of the upper placed box are smaller than width and depth of the lower box respectively. You can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.



Synopsis –

- Given n types of rectangular 3D boxes, with height(h), width(w) and depth(d)
- Goal is to stack them on each other such that height of stack is MAXIMUM
- Constraints – The stack should be such that any box should have its base dimensions (width & depth) strictly smaller than box below
- We can rotate boxes such that width is smaller than depth. For example, if there is a box with dimensions {1x2x3} where 1 is height, 2×3 is base, then there can be three possibilities, {1x2x3}, {2x1x3} and {3x1x2}



- We can use multiple instances of boxes. What it means is, we can have two different rotations of a box as part of our maximum height stack
- For Simplicity, we will keep `width <= depth`

Algorithm Description –

1) Generate all 3 rotations of all boxes. The size of rotation array becomes 3 times the size of original array. For simplicity, we consider depth as always smaller than or equal to width.
2) Sort the above generated 3n boxes in decreasing order of base area.
3) After sorting the boxes, the **problem is same as LIS with following optimal substructure property**. MSH(i) = Maximum possible Stack Height with box i at top of stack MSH(i) = { Max (MSH(j)) + height(i) } where j < i and width(j) > width(i) and depth(j) > depth(i). If there is no such j then MSH(i) = height(i).
4) To get overall maximum height, we return max(MSH(i)) where 0 < i < n

Since this the variation of the Longest Increasing Subsequence, We apply LIS to get the Maximum Height of the Boxes for stacking.

Algorithm for LIS using LCS –

1) Consider the array to be used and sort the array using any of the sorting algorithm
2) Now place the sorted against the initial unsorted array by applying logic of Longest Common Subsequence on it
3) Get the Longest Increasing Subsequent set of numbers

Recurrence Equation –

optHeight[i]	$=Max\{_{\text{for x = i-1 to 1}}(optHeight[x] + height(i))\}$	IF width(j) > width(i) and depth(j) > depth(i).
	= height(i)	otherwise

Time and Space Complexity –

Time Complexity T(n) = $O(n^2)$

Space Complexity S(n) = O(n)

Implementation –

```
#include <bits/stdc++.h>
using namespace std;

//Declare a structure for the Dimensions of the boxes
struct Box {
	int height;
	int width;
	int depth;
};

void title(){
	printf("----- WELCOME TO THE BOX STACKING PROBLEM -----\n");
	printf("Given a set of n cuboid boxes with height(h), width(w) and depth(d). \n"
		"Create a stack of boxes which is as tall as possible, but a box can be
placed on top of another box \n"
		"only if both width and depth of the upper placed box are smaller than
width and depth of the lower box respectively. \n"
		"You can rotate a box so that any side functions as its base. \n"
		"It is also allowable to use multiple instances of the same type of box
\n");
	printf("-----------------------------------------------\n");
}

void displayInput(Box boxDimensions[], int n){
	printf("Given number of Boxes are %d and their dimensions are as follows - \n", n);
	for (int i = 0; i < n; i++ ) {
		printf("{%d x %d x %d}\n", boxDimensions[i].height, boxDimensions[i].width,
boxDimensions[i].depth);
	}
	printf("---------------------------------\n");
}

// C library compare function for Quick Sort
int compare (const void *a, const void * b) {
	return ( (*(Box *)b).depth * (*(Box *)b).width ) -
		( (*(Box *)a).depth * (*(Box *)a).width );
}

int boxStackingHeight( Box boxDimensions[], int n ) {
	//Generate all 3 rotations of all boxes.
	//The size of rotation array becomes 3 times the size of original array

	Box rotation[3*n];
	int index = 0;
	for (int i = 0; i < n; i++) {
		// Original dimensions of box - First Rotation
		rotation[index].height = boxDimensions[i].height;
		rotation[index].depth = max(boxDimensions[i].depth, boxDimensions[i].width);
		rotation[index].width = min(boxDimensions[i].depth, boxDimensions[i].width);
		index++;

		// Second rotation of box
		rotation[index].height = boxDimensions[i].width;
		rotation[index].depth = max(boxDimensions[i].height, boxDimensions[i].depth);
		rotation[index].width = min(boxDimensions[i].height, boxDimensions[i].depth);
		index++;

		// Third rotation of box
		rotation[index].height = boxDimensions[i].depth;
		rotation[index].depth = max(boxDimensions[i].height, boxDimensions[i].width);
		rotation[index].width = min(boxDimensions[i].height, boxDimensions[i].width);
		index++;
	}
```

```
	n = 3*n;

	// Sort the array 'rotation[]' in decreasing order of base area using Quick Sort
	qsort (rotation, n, sizeof(rotation[0]), compare);

	// Printing all rotations of the Box
	printf("All Rotations of the Boxes \n");
	for (int i = 0; i < n; i++ ) {
		printf("{%d x %d x %d}\n", rotation[i].height, rotation[i].width,
rotation[i].depth);
	}
	printf("--------------------------------\n");

	// Initialize maxHeightOfStack values for all indexes
	// maxHeightOfStack[i] --> Maximum possible Stack Height with box i on top
	int maxHeightOfStack[n];
	for (int i = 0; i < n; i++ ) {
		maxHeightOfStack[i] = rotation[i].height;
	}

	// Compute optimized maxHeightOfStack values in bottom up manner
	for (int i = 1; i < n; i++ ){
		for (int j = 0; j < i; j++ ){
			if ( rotation[i].width < rotation[j].width &&
					rotation[i].depth < rotation[j].depth &&
					maxHeightOfStack[i] < maxHeightOfStack[j] +
rotation[i].height) {
				maxHeightOfStack[i] = maxHeightOfStack[j] + rotation[i].height;
			}
		}
	}

	// Pick up the maximum height from the stack
	int max = -1;
	for ( int i = 0; i < n; i++ ) {
		if ( max < maxHeightOfStack[i] ) {
			max = maxHeightOfStack[i];
		}
	}
	return max;
}

// Main Function
int main() {
	title();
	// Given input with 3 boxes of the height h, depth d and width w
	Box boxDimensions[] = { {10, 20, 30},
			{45, 55, 60},
			{50, 52, 72} };

	// Dynamic array size declaration
	int n = sizeof(boxDimensions)/sizeof(boxDimensions[0]);

	//Display input
	displayInput(boxDimensions, n);

	int maxHeight = boxStackingHeight(boxDimensions, n);
	printf("The MAXIMUM Possible Height of the Stack = %d\n", maxHeight);

	return 0;
}
```

How to run the code –

Refer **boxStacking.cpp** for the code in the same folder.

Open Command Prompt in Windows/ Terminal in Linux

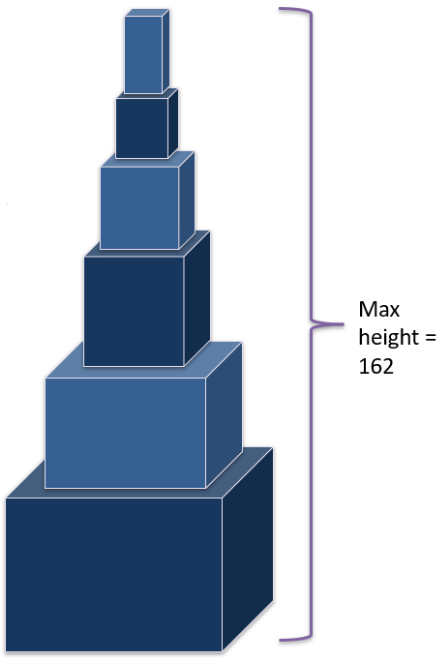
Run ‘make’ command

Get the result on the screen

Output –

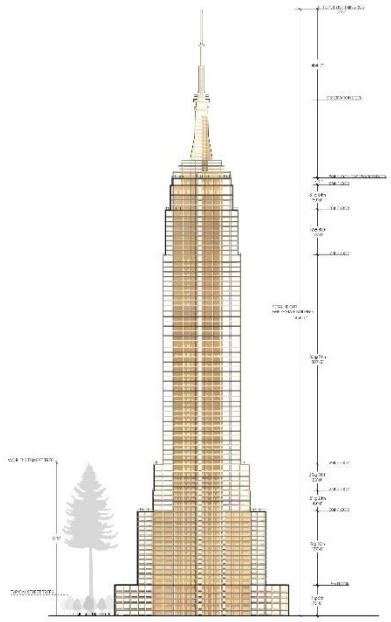
```
C:\Windows\System32\cmd.exe

C:\Users\chait\OneDrive\Documents\Semester - 1\Design and Ananlysis of Computer Algorithms\Term Project>make
g++ boxStacking.cpp -o submission -lm
./submission
----- WELCOME TO THE BOX STACKING PROBLEM -----
Given a set of n cuboid boxes with height(h), width(w) and depth(d).
Create a stack of boxes which is as tall as possible, but a box can be placed on top of another box
only if both width and depth of the upper placed box are smaller than width and depth of the lower box respectively.
You can rotate a box so that any side functions as its base.
It is also allowable to use multiple instances of the same type of box
-----------------------------------------------
Given number of Boxes are 3 and their dimensions are as follows -
{10 x 20 x 30}
{45 x 55 x 60}
{50 x 52 x 72}
--------------------------------
All Rotations of the Boxes
{50 x 52 x 72}
{52 x 50 x 72}
{45 x 55 x 60}
{55 x 45 x 60}
{72 x 50 x 52}
{60 x 45 x 55}
{10 x 20 x 30}
{20 x 10 x 30}
{30 x 10 x 20}
--------------------------------
The MAXIMUM Possible Height of the Stack = 162

C:\Users\chait\OneDrive\Documents\Semester - 1\Design and Ananlysis of Computer Algorithms\Term Project>
```

Result –



Applications of Box Stacking –

1. **Construct Classic Architectural Buildings**



2. **Stacking the boxes in the Amazon/ Flipkart warehouses**

References –

https://stackoverflow.com/questions/2329492/box-stacking-problem

https://algorithmsandme.com/box-stacking-problem-dynamic-programming/

https://algorithms.tutorialhorizon.com/dynamic-programming-box-stacking-problem/

https://www.youtube.com/watch?v=KgWy0fY0dRM

https://www.youtube.com/watch?v=9mod_xRB-O0