**İ.T.Ü.**

**Bilgisayar ve Bilişim Fakültesi**

**Bilgisayar Mühendisliği Bölümü**



**ANALYSIS OF ALGORITHMS**

**HW3**

**Aykut Akın**

**040080177**

**09.05.2011**

**Description of how my program should be compiled and run:**

I use Microsoft Visual C++ 2010 Express to develop the program. Program can be compiled and run on Linux/Unix using g++.

You should write a command like that for compile the Tower of Babylon solution Linux terminal.

g++ main.cpp tower.cpp –o ToB

You should give a command like that for run the Tower of Babylon solution after compilation.

./ToB

**Description of homework**

Apart from the Hanging Gardens the Babylonians (around 3000-539 b.c.) built the Tower of Babylon as well. The tower was meant to reach the sky and God, but the project failed because of a confusion of language imposed from much higher above.

For the 2645th anniversary a model of the tower will be rebuilt. *n* different types of blocks are available. Each type has a width *x,* a depth *y* and a height *z*. The blocks are to be stacked one upon each other so that the resulting tower is **as high as** possible. Each one of them may be duplicated as many times as you like, of course after rotating it. Make sure that for the reasons of stability a block can only be stacked upon another if **both** dimensions of the 2-D base of the lower block are each **strictly** larger than those of the 2-D base of the upper block.

**Input**

The number of types of blocks *n* is located in the first line of each test case. On the subsequent *n* lines the width *xi,* the depth *yi* and the height *zi* of each type of blocks are given. There are never more than 30 different types available.

There are many test cases, which come one by one. Input terminates with n = 0.

The name of the input file is **babylon.txt** Read it from your code.

**Sample input:** (in file *babylon.txt*)

3

1 1 1

1 2 3

4 5 6

1

1 1 1

3

1 1 4

4 5 6

1 2 3

0

**Output**

For each test case your program should output one line with the height of the highest possible tower first. Then the dimensions of the blocks forming the tower should be given properly considering the placement of the block (x, y, z). **Print** the output on screen (terminal).

**Sample output:** (on screen)

Max height: 14

Blocks respectively:

1 2 3

2 3 1

4 5 6

5 6 4

//////////////////////////////////////////////

Max height: 1

Blocks respectively:

1 1 1

//////////////////////////////////////////////

Max height: 15

Blocks respectively:

1 1 4

2 3 1

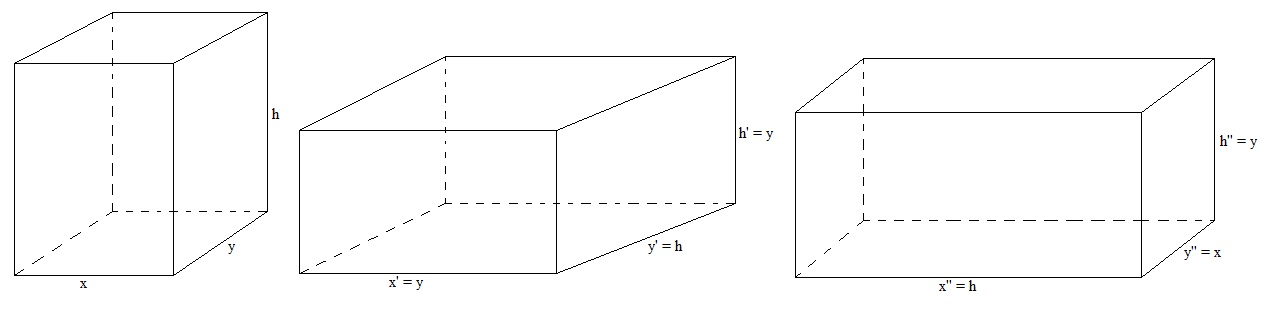
4 5 6

5 6 4

//////////////////////////////////////////////

**Solution of the problem**

First of all I rotate the blocks with three different dimensions. We don’t need every different sight because it will be the same with each others. For example;



I choose the rotate each block with the same directions so I eliminate the other same results.

Then I made a graph which is topological order. Priority of node is decided according to its x and y values. If there is an edge between two nodes then one of them is **strictly larger** than another. For example;

After that I put in order nodes according to their adjacencies. How much times a node is seemed in as another node adjacent, and then it comes much more after. If we consider same graph above, then the result is 111 – 123 – 456. Because, 111 and 123 are not an adjacent of another node. 456 is an adjacent of two nodes. So it comes after than 111 and 123. Actually this method gives compatibilities of each node. Again, if we consider same graph; 456 has more compatibility than others. Because, it is on the last in result vector. I mean, we can use 456 with much more blocks. For example; 111-456 and 123-456 can be used but 111-123 can not be used. 456 exists two times, 111 and 123 exist one time.

After that I started to calculate maximum highness for each block. I store them inside an array. When the algorithm is finished, we can see maximum highness for each block. I mean, if we look at the m[0] then we see a maximum highness which is occurred by using 0th block and its proper adjacent. If we look at the m[1] then we see a maximum highness which is occurred by using 1st block and its proper adjacent. Maximum element of this array is an answer that we were looking for.

**Memoization** is occurred while we are using **m** array. For example, we checked and we decided to add 2nd block onto 4th block. If there was a block or blocks onto 2nd block then we add them onto 4th block too.

**m[4] = 4th block height + 2nd block height(m[2]).**

**m[2]= 2nd block height + other blocks which are add onto 2nd block(m[..]).**

**My header files & explanations**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_block.h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Block{

public:

int x; //x axes of block

int y; //y axes of block

int h; //z axes of block

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_blocks.h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Blocks{

public:

Block blockList[90];//includes 90 block because there can’t be more than 30\*3 block

int numofblocks; //number of blocks inside the blockList array

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_node.h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Node{

public:

int count; //number of adjacency of a node

vector<int> adj; //adjacent of a node

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_node.h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Graph{

public:

Node nodeList[90]; //includes 90 nodes because there can’t be more than 30\*3 node

int numofnodes; //number of blocks inside the nodeList array

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_tower.h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Towers{

vector<Blocks\*> towerList; //each different data

vector<vector<int> > TopOrderedList;//result from toporder(int) function

vector<Graph\*> graphList; //graph from nodes

vector<int> toporder(int); //Order nodes according to their adjacencies

void readData(string); //read data from file

public:

Towers(string); //Constructor

void setGraphWithTopOrder();//fill the TopOrderedList and graphList vectors

void getAnswers();//calculate the answer

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_