LUMBERJACK

Team Blaze

Abhinav Pratap Singh (180010001)

Rishit Saiya (180010027)

Utkarsh Prakash (180030042)

Environment Variables

- For storage of different parameters of trees, a structure has been used.
- Profit = Price+ Profit earned by Domino Effect i.e. Price of other tree that fall due to domino effect.

- “direc” variable stores the **direction** in which the tree should be cut so as to get the maximum profit.

- Vector “v” stores the information of all the trees.
- “c_x” and “c_y” store the information about the current x,y coordinates i.e. the coordinates of the most recently cut tree.

- “n_x” and “n_y” store the coordinates of next tree to be cut.
- “t” is the time that has been given to us cut the tree(not the execution time of the program).

- Color tells whether the tree has been cut or not.

NOTE : The other variables used are as per the Optil problem statement only.

Main Algorithm Idea

The main idea remains to maximize the profit to maximum extent.

The main factors affecting the profit are :

- The extraction of maximum profit from the Domino Effect.
- Taking the path involving the trees in shorter range to previous range.

Hence, we decided the tree which needs to cut on basis of the quantity

profit/time

Extraction of maximum profit from Domino Effect

The functions used for the above mentioned factors are calculate_profit(), cutup_profit(), cutdown_profit(), cutright_profit(), cutdown_profit().

We will explain the functions explicitly in the coming slides.

cutup_profit() function

```
int cutup_profit(int a, vector<int> &temp){
        if(v.neighbour.up){
                int n = v.neighbour;
        if(bool domino(n) = TRUE)
                if(bool domino(n.neighbour) = TRUE)
                        Recursive call;
                profit = profit.tree + profit.domino;
    }
return profit;
}
```

NOTE: Similar function is performed by cutright_profit(), cutdown_profit() and cutleft_profit().

cutup_profit() function

This function first finds the **nearest** neighbour of the given tree (in the “**up**” direction) and then checks whether that tree can have a domino effect or not.

Then it makes a recursive call to find that whether the next nearest neighbour of the given tree (in the up direction) can participate in the domino effect.

The function returns the **extra-profit** that can be made when the tree is cut in the up-direction due to the domino effect.

calculate_profit() function

```
void calculate_profit() {
        for (i=0 to v.size()) {
                int cost= abs(v[i].x-c_x) + abs(v[i].y-c_y) + v[i].d;
                if (cost <= t) {
                        vector<int> temp;vector<int> track;
                        upProfit= cutup_profit(i, temp);
                        track = temp;
                        dprofit= upProfit;
                        temp.clear();
                        rightProfit= cutright_profit(i, temp);
//Doing it for all directions
                        v[i].profit += dprofit;
                        v[i].track= track;

}
```

calculate_profit() function

This function calculates the direction in which the profit is maximum and updates the profit of each tree in that direction.

Choosing the tree which needs to be cut next

The function used for the above mentioned factor is path().

The explicit explanation of this function is in the coming slide.

path()

```
int path() {
        float max_para=0;
        int n_i=0;
        for (int i=0; i<v.size(); i++) {
                int cost= abs(v[i].x-c_x) + abs(v[i].y-c_y) + v[i].d;
                if (cost <= t) {
                        float para= (float)v[i].profit/cost;
                        if (max_para <= para) {
                                max_para= para;
                                n_i= i;
                        }
                }
        }
        return n_i;
}
```

path()

This function first computes the cost of cutting a tree i.e. the cost of reaching to that tree plus the diameter of the tree.

If this cost is less than the time left with us, then it calculates the profit/time for that tree. Then it finds the maximum of this quantity among all the trees and the next tree to be cut will be the tree which has the maximum of this quantity.

Now that we have achieved the maximum profit by our algorithm and heuristics, we proceed to completing the question by printing the path.

printPath()

This function simply prints the path that is to be followed to reach a tree.

printPath()

```
void printPath() {
        if (n_x > c_x)
                for (int i=0; i<(n_x-c_x) && t>0; i++)
                        cout << "move right" << endl;
                        t--;
        else
                for (int i=0; i< (c_x - n_x) && t>0; i++)
                        cout << "move left" << endl;
                        t--;
        if (n_y > c_y)
                for (int i=0; i< (n_y - c_y) && t>0; i++)
                        cout << "move up" << endl;
                        t--;
        else
                for (int i=0; i< (c_y-n_y) && t>0; i++)
                        cout << "move down" << endl;
                        t--;
}
```

Updating our database

When we want cut a tree we color it black. We update the c_x and c_y as n_x and n_y respectively.

```
{
v[i].color = "black";
c_x = n_x;
c_y = n_y;
for (int j=0; j<v[i].track.size(); j++)
                for (int k=0; k<v.size(); k++)
                                for (int l=0; l< v[k].track.size(); l++)
                                                if (v[k].track[l] == v[i].track[j])
                                                                v[k].profit=- v[v[i].track[j]].price;
}
```

This nearly sums up all the algorithms and heuristics which we used for this project. Using these, we were able to achieve a profit of **14,44,21,499 units** according to Optil.io standards.

The programming language used was C++.

THANK YOU