Lumberjack

Team Cool Cutters (Team 6)

Tarun Goyal (180010038)

Manjeet Kapil (180010021)

Shagun Bera (180010031)

IITDH

Problem Statement

You are a lumberjack in a roadside forest. Every day you would like to cut down trees of the maximum value in the limited time that you have. We represent a forest as a rectangular grid of dimensions **n** by **n** points. Each edge between the neighboring grid points has a length of 1. At the position of each point, there could be a tree. Each tree is described using the following parameters:

- height **h**,
- thickness **d**,
- unit weight **c**,
- unit value **p**.

We calculate the value of each tree according to the formula: **p** Â· **h** Â· **d**. To cut down the tree, you have to reach it first. You can move only along the edges between the neighboring grid points. To cut down the tree at the position that you occupy you have to select one of four directions (along grid lines) in which the tree will fall. At the beginning of the day, you start in the lower-left corner of the forest, which is located at the point (0,0). You can assume that there is no tree there. If the tree that you cut falls on the other tree, which is lighter than the one that is falling, then this tree will also fall without being cut directly. We say that the tree is heavier if the total weight calculated as **c** Â· **d** Â· **h** is greater than the total weight of the second tree. In this way, one can achieve a domino effect by pushing one tree onto another. We assume that a tree falls on the other when the distance between them is less the height of this tree. Weights of trees do not cumulate - we always take into account only the weight of the tree, which directly fell onto another. If for example tree A falls onto tree B and its weight is greater than weight of tree B then it can push tree B onto another tree C located in the same direction if the distance between B and C is smaller to height of B and, moreover, tree C is lighter than tree B. When comparing trees B and C we do not take into account tree A that lays on tree B. Moreover, if tree B is not lighter than tree A then it blocks tree A and it will not have a chance to fall onto tree C.

Calculate the biggest profit that you can reach during **t** time units and provide a corresponding sequence of movements. Assume that the walk between the two neighboring grid points takes 1 unit of time, and cutting the tree takes **d** units of time.

INPUT

The first line of input contains three integers: 0 < **t** <= 5,000 representing the time limit for the lumberjack, 0 < **n** <= 1,000 represending the size of the grid, and 0 < **k** <= 10,000 denoting the number of trees. In the following **k** lines there is a description of consequtive trees consisting of six integer numbers 0 <= **x**, **y** < **n**, 0 < **h**, **d** <= 20, 0 < **c**, **p** <= 500 representing the position of the trees on the grid, their height, thickness, weight and value.

OUTPUT

On the output specify a list of moves (using **move** prefix) and cuts (using **cut** prefix) with the suffix denoting the direction of the action (**up**, **right**, **down** or **left**), each on a separate line.

1. We have stored :-

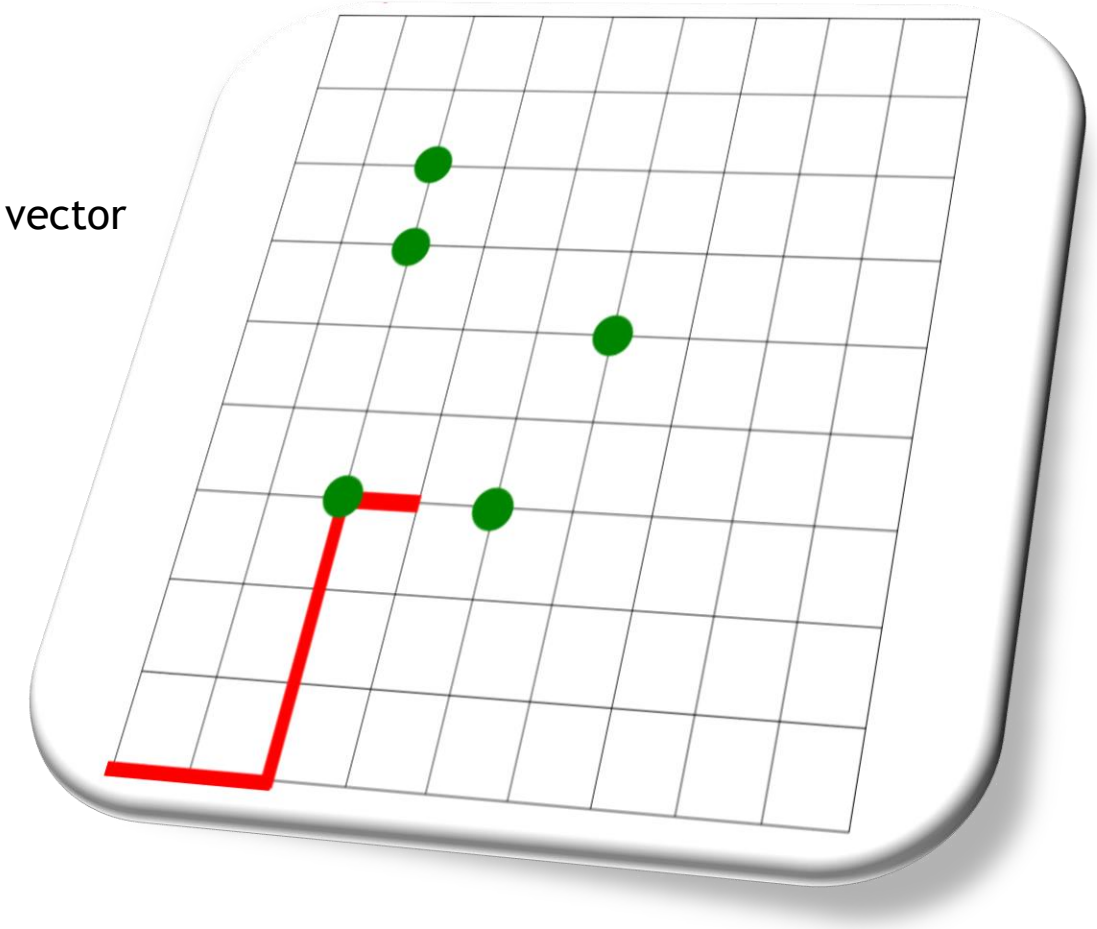
- ❖ k=number of trees
- ❖ n = size of grid n*n
- ❖ t= total time given for maximizing the profit

2. We store properties of each trees in there respective vector

- ❖ x → vector x
- ❖ y → vector y
- ❖ height → vector height
- ❖ thick → vector thick
- ❖ profit → vector profit
- ❖ weight →vector weight

3. We also have some temporarily variables

- Xinitial=0
- Yinitial=0
- Flag=0



- We execute findyprofit() which tells us the trees which fall due to the domino effect.
- We store this information in 4 vectors proLeft , proRight ,proUp and proDown.
- These Pro Vectors are 2D vectors , the first element stores net profit (considering domino effect) and then the index of the trees that fall due to the domino effect .

```
void findyProfit(int k,int n)
{
        proLeft.resize(k);
        proRight.resize(k);
        proUp.resize(k);
        proDown.resize(k);
    for(int i=0;i<k;i++)
    {
        /*horz(i,i,k,n);
        vert(i,i,k,n);*/
        proUp[i].push_back(profit[i]);  //first element of each vector is their net profit in that direction
        proDown[i].push_back(profit[i]);
        proLeft[i].push_back(profit[i]);
        proRight[i].push_back(profit[i]);
        left(i,i,k);    //finding index in left
        right(i,i,k,n);
        up(i,i,k,n);
        down(i,i,k);
    }

}
```

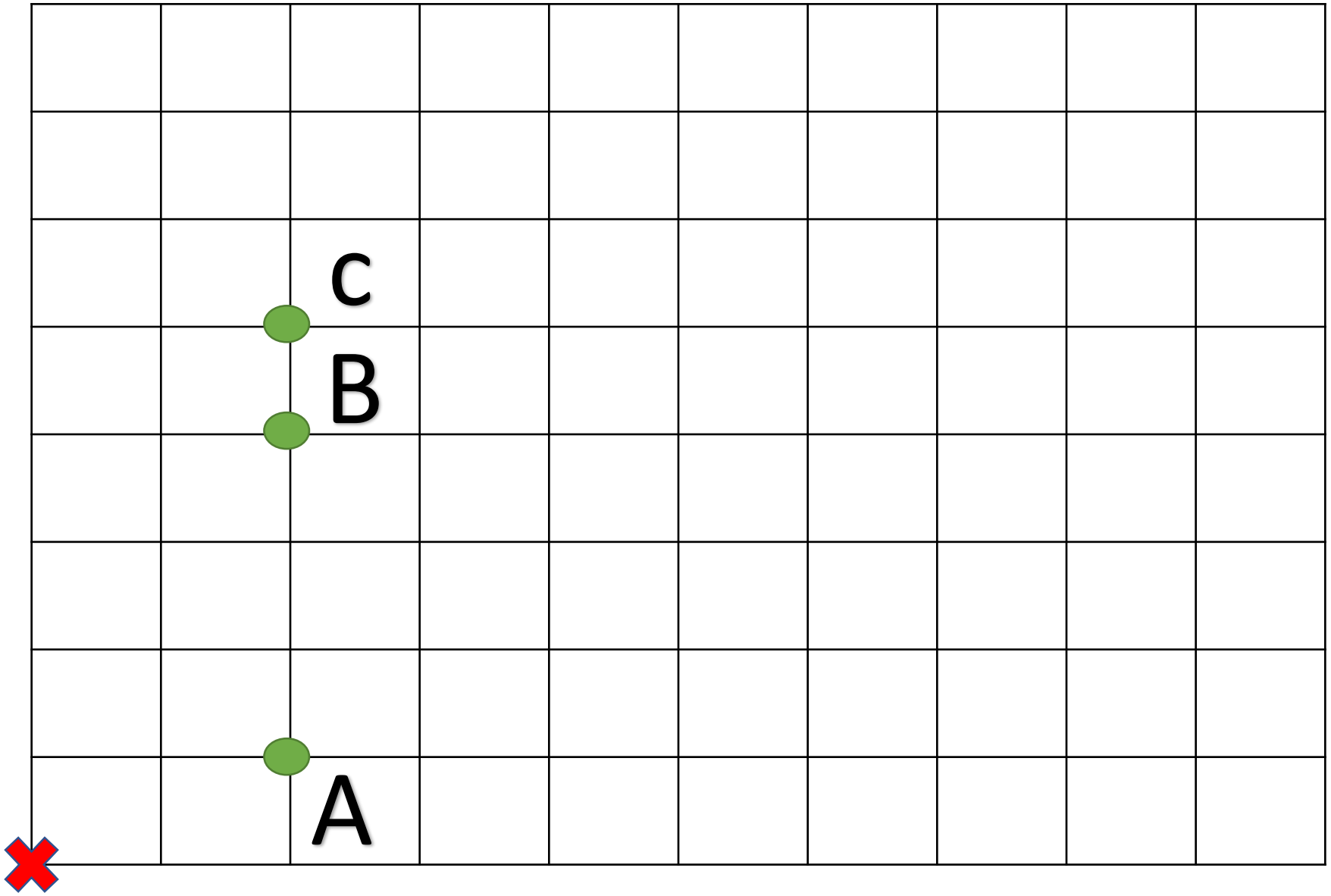
- We have 4 functions left() , right() , up() , down() which identify the trees which fall due to the domino effect using recursion.It also checks all the relevant conditions like the distance between the trees and their weight .
- This information is stored in the relevant vectors proUp , proDown .
- We have 3 more vectors diru , uttuprofit and flag which store the direction in which that tree is to be cut i.e. the direction which gives maximum profit , the trees that fall due to the domino effect and if the tree is already cut or not respectively. These vectors also use the same index.

```
void left(int a,int b,int k)  //find profits when tree fall in left direction
{
    for(int i=0;i<k;i++)
    {
        if(y[i]==y[a])
            {
                if(x[i]<x[a])    //for left //check condition
                {
                    if(weight[i]<weight[a] && height[a]>(x[a]-x[i]))
                    {
                        if(!(count(proLeft[b].begin()+1,proLeft[b].end(),i)))
                        {
                            proLeft[b].push_back(i);   //pushing index of fallen tree
                            proLeft[b][0]+=profit[i];  //increasing profit
                            left(i,b,k);  //reciursion
                        }


                    }
                }

            }
    }
}
```

Our Algorithm

- We now have a new vector timy which stores the time required to reach and cut that tree (if a tree is cut already we store infinity as the time required to reach it).
- We find the ratio of profit obtained and time for reaching and cutting tree , and proceed to cut the tree which has that maximum ratio .
- After cutting the tree the domino effect is considered using information from uttuprofit .The corresponding steps “move up” and “move down” etc are also printed here.The direction is also printed using information from the diru vector. The flags are also updated and run() is called again recursively till the maximum time limit is reached or we cant find any feasible tree .

```
    for(int i=0;i<k;i++)
    {

        if(timy[i]<=t)     //find overall max profit in all tree which is in given time limit
        {
            if(float(profit[i]/timy[i])>ratio)
            {
                ratio=float(profit[i]/timy[i]);
                timu=timy[i];
                index=i;   //setting index
                indexback=index;
                flag=1;   //flag is the sign that we find a tree to cut
            }

            if(float(profit[i]/timy[i])==ratio)
            {
                timu=min(timu,timy[i]);
                if(timu==timy[i])       //this thing i add new
                {
                    index=i;
                }
                else
                {
                    index=indexback;
                }
                flag=1;
            }
        }

    }

        if(flag==1)   //we find a tree to cut at index
        {
            go_to(xInitial,yInitial,index,diru[index]);
            for(int l=0;l<uttuprofit[index].size();l++)
            {
                cuttedflag[uttuprofit[index][l]]=1;
            }
            int xlater=x[index];
            int ylater=y[index];
            timy.clear();
            run(n,k,t-timu,xlater,ylater);

        }
}
```



Profit of cutting

A : 10

B : 25

C : 15

Thickness

A : 2

B: 3

C: 4

Thank You