DSA Project Report LUMBERJACK

Team Detail

Bhavesh Borse
200010005@iitdh.ac.in
Eshita Pagare
200010016@iitdh.ac.in
Harrithha
200010018@iitdh.ac.in
Shashank P
200010048@iitdh.ac.in

Abstract

This paper describes the ideas, heuristics, algorithms and program witten by **Quad_Bytes** for the *Lumberjack* problem present in optil.io.

1 Data Structures used

- **t, n, k** : Stores time, size of grid, number of trees
- **2D List** : Stores of details of each tree, i.e. trees[0][i] keeps track of x, y coordinates, height, thickness, unit weight and unit value of the first tree.
- **1D Lists and Array** : For storing present coordinates, direction of cut, nearest tree from present coordinates, several temporary arrays in the functions.

2 Functions used in program

- **sort(sub_li, n)** : Sorts the input 2D list (sub_li) according to the nth index of each element(list).
- **distance(x1, x2, y1, y2)** : Calculates distance between two points (x1, y1) and (x2, y2).
- **nearest(p_x, p_y, trees, t, sen)** Depending on whether sen(profit obtained on cutting a tree) is 0 or 1(profit $>$ 3904), our optimizing factor temp_div is assigned value. From present coordinates(p_x, p_y), finds the tree whose

  $$\frac{\text{distance from present position + time taken to cut tree}}{\text{temp\_div}}$$

  is minimum and returns the index of that tree.
- **value_cut(sub, trees)** : For each tree(sub) in trees 2D List, checks for domino effect in each of the 4 directions(up, right, down, left) in case we cut that particular tree. Returns an array of 2 elements: Maximum value of profit on cutting sub, the direction where this profit is obtained.
- **traversal(p_x, p_y, f_x, f_y, t)** : Moves the lumberjack from present coordinates(p_x, p_y) to new coordinates(f_x, f_y) and in the process updates the time from t to t – time taken to move from present to new coordinates.
- **cut(sub, trees, dir)** : Cuts nearest tree(sub) in a particular direction(dir) and checks for domino effect according to given conditions in the question and removes the cut trees from the trees 2D List.

3 Complexity Analysis

SORT(sub_li, n)
sub_li.sort($key = lambdax : x[n]$)
return sub_li

The Python list **sort()** uses **Timsort** algorithm. The algorithm has runtime complexity of $O(n.log(n))$.

DISTANCE(x_1, x_2, y_1, y_2)
return $abs(x_1 - x_2) + abs(y_1 - y_2)$

NEAREST(P_x, P_y, $trees$, t, sen)
$dmin, same, k = [float('inf'), [], len(trees)]$
$\boldsymbol{for} i$ in range(k) ....$O(log(n))$
$\boldsymbol{if} distance(.....)$ ......$O(1)$
continue
$\boldsymbol{if} sen == 0 :$ ......$O(1)$
$\boldsymbol{if} sen == 1 :$ ....$O(1)$
$\boldsymbol{if} (trees[i][0] == p_x andtrees[i][1] == p_y) ord > dmin :$ ....$O(1)$
continue
$\boldsymbol{if} d == dmin :$ ....$O(1)$
$\boldsymbol{if} distance(...) < distance(...) + +trees[same[0]][5]$ ....$O(1)$
$\boldsymbol{if} d < dmin :$ ....$O(1)$
$dmin = d$
$same = []$
$same.append(i)$
continue

TRAVERSAL($p\_x,\ p\_y,\ f\_x,\ f\_y,\ t$) :

while $p\_x < f\_x$: ....$O(n^2)$

  _if_ $t == 0$:

    $t -= 1$

    $print('moveright')$

    $p\_x += 1$

while $p\_y < f\_y$: ....$O(n^2)$

  { ... }

while $p\_x > f\_x$: ....$O(n^2)$

  _if_ $t == 0$:

    $t -= 1$

    $print('moveleft')$

    $p\_x += 1$

while $p\_y > f\_y$: ....$O(n^2)$

  { ... }