CS5010 Algorithm Assignment 3

Q1 Red Black Tree

Given an empty tree to begin with, draw the red black tree formed in each stage by inserting following keys in order:

39, 36, 29, 10, 17, 6

Answer

I'll simply use R and B to represent Red and Black color.

First insert 39

```
        39 B
```

Insert 36

```
          39 B
         /
      36 R
```

Insert 29

```
          36 B
         /    \
     29 R     39 R
```

Insert 10

```
          36 B
         /    \
     29 B     39 B
     /
  10 R
```

Insert 17

```
          36 B
         /    \
     17 B     39 B
```

```
    /    \
  10 R  29 R
```

Insert 6

```
          36 B
         /    \
      17 R    39 B
     /    \
   10 B  29 B
   /
  6 R
```

Q2 Red Black Tree Proof

Prove that in any red black tree, at least half of the nodes on any path from root to a given leaf must be BLACK nodes.

Answer

The Red Black tree has several limitation, but the main limitation is every red node's children must be black. Nil node is default black.

1. Now let's see if a red black tree only contains one red node(Invalid red black tree). Since its children must be black, we can form the tree to something like

   ```
       1 R
      /   \
   Nil B Nil B
   ```

   There's two paths from root to leaf, both contains 1 red node and 1 black node, which red_cnt = black_cnt.

2. Now let's see the black node, black node's children has no limitation, so a black node's child can either be black node or red node. If its child is black node, great, repeat this procedure; If its child is red, go back to situation 1, red node must have black child, so now we will have at least 2 black node and 1 red node.

3. Now we see the complicated red black tree. We have already seen that in every subtree with a root of red node, the number of black nodes is at least equals to the number of red nodes in a path from root to leaf node, once there's a node is black, the number of black node will be bigger than the number of red node in a path.

So in any red black tree, at least half of the nodes on any path from root to a given leaf must be BLACK nodes. In invalid tree which only contains one red node as root, the number of red node will equals to the

number of black node in a path from root to leaf, but it is invalid. So at least half of the nodes on any path must be BLACK nodes.

Q3 Skie Fun

There are N skiers each with a height H_i. There are also N skies each at a height of S_j. Both i and j are 1,2,...,N. Your task is to find an assignment of skier to skies such that the sum total of skie disparity is minimum.

Skie disparity for a given assigned skier is measured by the absolute difference between the height of the skier and height of the skie.

Also prove the optimality of your algorithm.

Answer

Algorithm:

1. I think this is not about sorting problemts, so I'm going to sort the two lists H_i and S_j. Apply the quick sort, which has a time complexity of O(nlogn).
2. Then we apply the greedy choice, since we sorted the two lists, for each pair of skie and skier should be arranged linearly, which means that for H_i[i], the skie should be S_j[i].
3. Then we simply iterate the lists from head to tail, compute the absolute value of H_i[i] - S_j[i].

   Code in Python:

```
def minimumDisparity(H_i,S_j):
    disparity = 0
    H_i.sort()
    S_j.sort()

    for i in xrange(len(H_i)):
        disparity += abs(H_i[i]-S_j[i])

    return disparity
```

Proof:

Now we use cut-and-paste to prove the greedy solution can be applied to this scenario.

Assume we now have a pair solution that does not follow the greedy choice. Still make the S_i list to be sorted in order to simplify the question.

We make H_i to be in ascending order, and S_j to be the pairing order, which means under non-greedy optimal approach H_i[i] pairs with S_j[i] and S_j does not need to be sorted.

Now we apply the cut and paste, we take a subarray with length of m. Rename them to be H_m and S_m.

In order to simplify the question again, we make the length m to be 2, since the size can be spanned as big as possible.

So assume S0 pair with H0, S1 pair with H1, H0 < H1 and S0 > S1 since we decide not to sort the list. We now have several situations:

- **H0 < H1 < S1 < S0**

  parity0 = S0 - H0, parity1 = S1 - H1

  parity_sum = S0 + S1 - H0 - H1

  Now we apply the greedy choice, then

  parity0 = S0 - H1, parity1 = S1 - H0

  parity_sum = S0 + S1 - H1 - H0

  parity_sum is the same.

- **S1 < S0 < H0 < H1:**

  we can similaly process this situation like previous one.

- **H0 < S1 < H1 < S0 and S1 < H0 < S0 < H1:**

  parity0 = S1 - H0, parity1 = S0 - H1

  parity_sum = S0 + S1 - H0 - H1

  Now we apply the greedy choice:

  parity0 = S0 - H0, parity1 = H1 - S1

  parity_sum = S0 - S1 + H1 - H0

  The first parity_sum is bigger than the second by 2S1 - 2H0.

  So the greedy is even better than the imagined optimized solutions under such circumstances.

- **H0 < S1 < S0 < H1:**

  parity0 = S1 - H0, parity1 = H1 - S0

  parity_sum = S1 + H1 - H0 - S0

  Now we apply the greedy choice to such situation, so we have

  parity0 = H1 - S0, parity1 = S1 - H0

  parity_sum = S1 + H1 - H0 - S0

  parity_sum is the same.

- **S1 < H0 < H1 < S0**

we can similaly process this situation like previous one.

As all the situations has been proved that the parity sum is the same, we can say that cut-and-paste can be successfuly used to prove right under size 2 situation. We can then expand the size to as big as we want.

So in conclusion, the greedy choice we make here is optimized solutions.

Q4 Swanton Berry Picking

You are at the world famous Swanton Berry Farm in Davenport. It is the peak of strawberry season and you want to pick maximum number of strawberries as possible. There are N bushes and each bush i has S_i number of strawberries.

However, for some reason if you pick strawberries from bush k, you cannot pick the berries from bush k+1.

Given this constraint, your task is figure out the maximum number of strawberries that can be picked given N and values of S_i for each bush i.

Answer

As there's no constraints other than we can not pick consecutive bush, we can assume that we can choose either bush in two bushes, and leave both of them not to choose would be a waste of strawberries, so in every two bushes we have to choose one bush to pick.

We use dynamic programming to solve this problem.

At index i, where i is from 0 to N-1, whether to pick this bush is determined by the max value of :1) Pick the current bush, and add the strawberries this bush has and the maximum value you can get from 0 to i-2 bush; 2) Do not pick the current bush, then current max strawberry number can be reached at index i is max strawberry number reached at index i - 1.

Now let's implement this algorithm in Pyhton.

```
def pickStrawberryFromBush(S_i, N):
    if not S_i:
        return 0
    if N < 3:
        return max(S_i)

    res = [0] * N
    res[0] = S_i[0]
    res[1] = max(S_i[0],max(S_i[1]))

    for i in xrange(2,N):
        res[i] = max(res[i-1],arr[i-2] + S_i[i])

    return res[-1]
```

Now in this function, res[i] represents the max number of strawberry you can get from 0 to i bush. res[-1] means the whole line of bushes are evaluate and chose.