CS575: Final Project Report

Project Title: Cat and Mouse, Pattern searching, Red Black Tree and Page Ranking

Team Member(s): Karthick Gunasekar, Harshitha Guru Raj, Yashaswi Dattatreya Hasarali

I. Problem

1. MiniMax: Used for decision making and obtaining the optimal solution for the player, assuming the opponent also plays optimally. And widely used in two player games
2. Pattern Searching: To improve efficiency in pattern searching and take lesser time when the length of the pattern increases. To minimize the number of comparisons being mad.
3. Red Black Tree Implementation: Balancing the tree is needed to guarantee good performance, as other algorithms like AVL, rebalancing takes more effort. And deletion of nodes takes more rotation.
4. Page Ranking: HITS algorithm is same as PageRank. They both make use of the link structure of the Web graph in order to decide the relevance of the pages. The difference is that unlike the PageRank algorithm, HITS only operates on a small subgraph (the seed SQ) from the web graph. This subgraph is query dependent; whenever we search with a different query phrase, the seed changes as well. HITS ranks the seed nodes according to their authority and hub weights. The highest ranking pages are displayed to the user by the query engine.

II. Algorithms

A. MiniMax Algorithm

Step 1: A directed graph with possible directions for both players, in this scenario a cat and a mouse, to make their moves
Step 2: The Mouse will start its move from the given starting position and try to maximize its outcome.
Step 3: In this problem, Cat is the opponent and it tries to minimize the opponent's outcome.
Step 4: Recursively, both the players try to maximize and minimize their stats.
Step 4: Result will be the optimal solution of the Mouse for the given directed graph.
Time Complexity : O(b*m)
where, b = the number of legal moves at each point and m = maximum depth of the tree

B. Boyer Moore

Step 1: Accept the String and the pattern to be searched.
Step 2: Searching can be done in two ways: bad character heuristic and good suffix heuristics.
Step 3: Based on the user selection, call the search function and pass the string and pattern as parameters.
Step 4: Algorithm takes a backward approach, the pattern is aligned with the string, and the last character of the pattern is compared with the string character.
Step 5a: If there is a match then the characters are compared one by one in backward.
Step 5b: If there is a mismatch, then the algorithm computes the new alignment for the pattern and aligns it accordingly.
Time Complexity of precomputation : O(m+size of alphabet)
Time Complexity of searching : O(n)

C. Red Black Tree Implementation

A type of self-balancing binary search tree known as a "red-black tree" has one extra bit at each node, which is frequently read as the color (red or black). In order to maintain the balance of the tree during insertions and deletions, these colors are used. The tree's balance, though not perfect, is sufficient to drastically reduce on searching time and keep it at or below O(log n), where n is the total number of elements in the tree.
(i) Insertion of the Node
Step 1: Check if the tree is empty. If the tree is empty, make it as the root node. Root should always be black as per the property of the Red Black tree. Then Exit.
Step 2: If not empty, check the value of the node and decide as per the binary search tree property to attach to the left or right. Make the node as a leaf with red color.
Step 3: If the parent of the new node is black, then exit from the operation.
Step 4: If the new node's parent is red, change the color of the parent node's sibling to the new node.
Step 5: If it is colored black or null, then make a suitable rotation (left or right) and recolor it.
Step 6: If it is colored red, then perform a recolor. Repeat the same until the tree becomes a red-black tree.

Time complexity of Insertion - O(logn)

(ii) Deletion of the Node
Step 1: The deletion process in BST must be carried out first. Like other BSTs, this phase is identical. This sort of deletion always results in the deletion of a node that either has one child or is a leaf. As a result, we only need to deal

with nodes that are leaf nodes or have a single child. Let u be the child node that replaces node_to_be_deleted and node_to_be_deleted be the node that will be eliminated.
Step 2: If either node_to_be_deleted or child is red, we mark the replaced child as black. Note that as node_to_be_deleted is the parent of child, and two consecutive reds are not allowed in the red-black tree, both child and node_to_be_deleted cannot be red.
Step 3: If Both child and node_to_be_deleted are Black.

Step 3.1: Child is represented as double black. Making this double black into a single black is now the only thing left to do. Recall that if node_to_be_deleted is a leaf, the child is a null node, and the null node's color is regarded as black. A double black is therefore created when a black leaf is removed.

Step 3.2: For a double-black child sibling node of the current node that is not the root.

Case 1: Rotate to balance the red-black tree if sibling_node is black and at least one of its children is red.

Case 2: if both of the sibling node's children and the sibling_node are black. recolor nodes, and if the parent is Black, repeat for the parent.

Case 3: If sibling node is red, rotate the old sibling node to the top and change the color of the parent and old sibling. New siblings are almost always black. As a result, the tree is transformed into the black sibling case and connects to earlier cases. This case is divided into two subcases; left and right cases. In the right case, rotate the parent to move the node up. In the left case, perform the right rotation.
Step 4: If child is the root, make it single black

Time complexity of Insertion - O(logn)

D. *HITS Algorithm*

Step 1: Assign Hub and Authority score to 1
Step 2: Iterate for k(no of iterations given) times
Step 3: Hub Score Updation : Σ (Authority score of each node it points to)
Step 4 : Authority Score Updation : Σ (Hub score of each node it points to)
Step 5 : Normalize the both Hub and Authority Score by dividing each hub and authority scores by the square root of the sum of squares of all authority and hub scores respectively.
Step 6: Check for convergence, i.e., over some iterations if the current authority and hub scores calculate the same values as previous ones, then terminate.

III. Software Design and Implementation

A. *Software Design*

Three of the algorithms are implemented using Java and one is written in Python.

B. *Implementation and Tools Used*

(1) Eclipse/STS for Java
(2) Visual Studio for Python

C. *Performance Evaluation (Optional)*

IV. Project outcomes

(1) Report
(2) Source Code - Github
(3) Presentation Slides

References

[1] A Fast String Searching Algorithm https://www.cs.utexas.edu/users/moore/publications/fstrpos.pdf
[2] SriHarsha Oddiraju, Boyer-Moore, http://cs.indstate.edu/~soddiraju/abstract.pdf
[3] http://www.rnlkwc.ac.in/pdf/study-material/comsc/Design.pdf
[4] https://www.codingninjas.com/codestudio/library/deletion-in-red-black-trees-586
[5] https://www.cs.auckland.ac.nz/software/AlgAnim/red_black.html
[6] https://medium.com/analytics-vidhya/deletion-in-red-black-rb-tree-92301e1474ea
[7] https://www.cs.usfca.edu/~galles/visualization/RedBlack.html
[8] https://www.cs.purdue.edu/homes/ayg/CS251/slides/chap13c.pdf
[9] https://medium.com/analytics-vidhya/deletion-in-red-black-rb-tree-92301e1474ea
[10] https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html
[11] http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture4/lecture4.html