A appraisal paper on Breadth-first search, Depth-first search and Red black tree

Ms. Avinash Kaur[*], Ms. Purva Sharma[**], Ms.Apurva Verma[**]

[*] Faculty of GDRCET
[**] Student of GDRCET

***Abstract*-** Breadth-first and depth-first search are indispensable adventure strategies leading which many other rummage around algorithms are built. In this dissertation, we projected an loomed of integrating these two strategies in a solitary algorithm that combines the corresponding strengths of mutual. We account preliminary computational outcome using the tree width predicament as an example. This dissertation explores the configuration of red-black trees by solving an apparently straightforward crisis: grant an uphill progression of rudiments, erect, in linear time, a red-black tree that contains the rudiments in symmetric sort. Numerous tremendous red-black ranking shapes are characterized: trees of least quantity and maximum loftiness, trees with a nominal and with a maximal fraction of red nodes.

***Index Terms*-** BFS (Breadth-First Search), DFS (Depth-First Search), Red Black Tree, Algorithm

I. INTRODUCTION

Breadth-first and Depth-first search are indispensable investigated strategies for incisive. specified the very diverse way in which they categorize node expansions, it is not apparent that they can be pooled in the same search algorithm. To demonstrate the reimbursement of this loom, we exploit the tree thickness predicament as an example. The tree breadth of a grid (also known as the induced tree width) is a appraise of how analogous the grid is to a hierarchy, which has a tree width of 1. A entirely attached graph is slightest akin to a hierarchy, in addition has a tree breadth of (n–1), where n is the number of vertices in the grid. Nearly every graphs enclose a tree width that is everywhere in sandwiched between 1 and the numeral of vertices minus 1. Red-black trees are well-designed search-tree proposal, with the intention of guarantees O (logn) worst-case administration era of vital dynamic-set operations. Recently,[1,2] vacant a stunning serviceable accomplishment of red-black trees[5]. In this dissertation we stab deeper into the formation of red-black trees by solving an according to the grapevine effortless quandary: prearranged a mounting progression of rudiments, assemble a red-black tree[6] that contains the ground rules in symmetric array. A red-black hierarchy is a binary search tree (BST) with five linked red-black properties[7]:

1. All node is either red or black.
2. The derivation nodule is black.
3. peripheral nodes are black.
4. A red node's children are mutually black.
5. All paths on or after a knob to its sheet descendants restrain the equivalent amount of black nodes. The erection ought to merely take linear instance. This algorithm has a uncomplicated recursive portrayal ***[3]:***

1. **procedure** DFS(v)
2. **begin**
3. smudge v as visited;
4. **while** there is an intact vertex w contiguous to v **do**
5. add (v, w) to T
6. DFS(w)
7. **end** { while)
8. **end** { DFS)

Algorithm of BFS[3]:

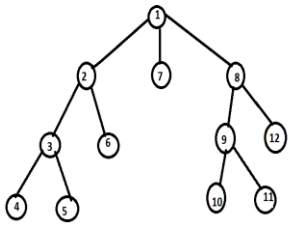

Fig:-Depth First Searc

1. procedure BFS(Graph,source):
2. generate a queue Q
3. enqueue foundation onto Q
4. blotch source
5. while Q is not vacant:
6. dequeue an item from Q into v
7. for each margin e incident on v in Graph:
8. let w be the other conclusion of e
9. if w is not marked:
10. mark w
11. enqueue w onto Q

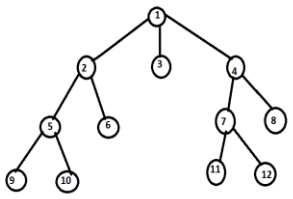

Fig:-Breadth First Search

Algorithm of RED BLACK TREE[5]:

```
1.  void insert_case3(struct node *n)
2.  {
3.  struct node *u = uncle(n), *g;

4.  if ((u != NULL) && (u->color == RED)) {
5.  n->parent->color = BLACK;
6.  u->color = BLACK;
7.  g = grandparent(n);
8.  g->color = RED;
9.  insert_case1(g);
10. } else {
11. insert_case4(n);
12. }
13. }
```

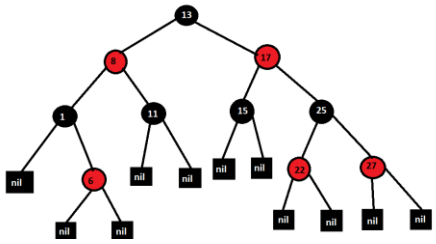

Fig:-Red Black Tree

II. LITERATURE REVIEW

Finding the accurate tree width of a general table is an NP-complete trouble. One come near to result the exact tree width is depth first [1,2]branch-and-bound rummage around in the space of vertex elimination orders. However, recent studies showed that best-first search can dramatically outperform depth-first branch-and-bound search by avoiding repeated generation of spare investigate nodes. In the search space of the tree width problem, each node corresponds to an in-between graph that results from eliminating a set of vertices from the original graph. Each elliptical represents a search node that is identified by the set of vertices eliminated so distant from the original graph. A lane from the start node which has an drain set of eliminated vertices to the ambition join which has all vertices eliminated to an exclusion order, and there is a one-to-one mapping from the set of exclusion orders to the set of paths from the start to the goal node. Even though there are n! different elimination orders for a graph of n vertices, there are only 2n diverse search nodes. This is since dissimilar ways of eliminating the same set of vertices always reach your destination at the same in-between graph, and there is only one distinct in-between graph for each grouping (as opposed to permutation) of the vertices. Depth-first[1,2] branch-and-bound search treats the search space as a tree with n! separate states instead of a graph with only 2n states. The faster presentation of best-first tree width search reflects the difference in size connecting a search tree and a search graph. Unfortunately, the scalability of best-first search is limited by its memory necessities, which tend to grow exponentially with the search depth. To improve scalability, use a memory-efficient version of best-first search called breadth-first heuristic search. Red-black trees[10] are a form of balanced search trees which can be without problems implemented [8,9]. They can also be seen as a variant of (2;4)-trees. Definition 1 (Red-black tree). A red-black tree is a binary tree whose inner nodes are associated with keys. Keys are elements of a entirely ordered set. A node can either be red or black.

III. PROPOSED METHODOLOGY

To reduce the time overhead of intermediate graph age band, we portray a search algorithm that does not generate the in-between graph from the original graph at the root node of the search space. Instead, it generates it from the intermediate graph of a close neighbor of the node that is being expanded. A red-black tree[2] is a binary tree whose inner nodes are coupled with keys. Keys are elements of a totally ordered set. A node can either be red or black.The advantage is that the in-between graph of a close fellow citizen is already very similar, and so there is much less overhead in transforming it into a new intermediate graph. The simplest way to find the node's closest neighbour is by computing shortest paths from a node to all of its neighbors and alternative the contiguous one. But at first, this does not seem to work for the tree width problem, since its state space is a partially ordered graph in which the distance between any pair of nodes at the same depth is infinite. Our idea is to measure the distance between a pair of nodes in a meta search space, in its place of the original search space. A meta search space has accurately the same set of states as the original search space, but is increased with a set of meta proceedings that can transform one node into another in ways not allowed in the original search space. For example, a meta action for the tree width problem can be an action that uneliminates a vertex by reversing the changes made to a graph when the vertex was eliminated. For the tree width problem augmented with the uneliminate meta action, its search graph is an undirected adaptation of the graph. In this new graph, called a meta search graph, actions i.e., edges can go back and forth between a pair of flanking nodes, and this allows us to cause the intermediate graph of a node from another knob at the same depth. This is very of use for breadth-first heuristic search, which expands nodes in order of their depth in the search freedom. Since a node is uniquely known by the set of vertices eliminated, we use the same lower-case letter (e.g., n, u, and v) to denote both a knob and a set of eliminated vertices in the rest of this paper. To implement the uneliminate meta action, each edge of the meta search graph is labeled by a tuple u, v, ΔE+, ΔE−, where u & v is the set of vertices eliminated so far at the source (destination) node of the edge, and ΔE+ (ΔE−) is the set of edges added to (deleted from) the graph when the vertex in the singleton set v \ u is eliminated.

Let, $G_n = V_n$, E_n be the intermediate graph associated with node n. The task of adding a previously-eliminated vertex backside to the graph can be uttered formally as: given $G_v = V_v, E_v$_ and e = _u, v,ΔE+,ΔE−_, how to compute $G_u = _V_u, E_u$ Since all the changes are recorded with theedge e, one can reconstruct $G_u = _V_u, E_u$_ as follows, $V_u = V_v \cup {}_v \setminus u$ (1)
$E_u = E_v \cup \Delta E- \setminus \Delta E+$ (2)

That is, by adding (deleting) the edges that have been beforehand deleted (added) to the graph, the "uneliminate" metan deed can undo the things of an elimination action in the original search space. In general, adding up meta actions can turn bound for search graphs into undirected graphs. This guarantees that any changes made to the current state (e.g., the intermediate graph) is reversible, creating a graph with the subsequent charming assets: for any two states accessible from the start state, there is always a lane that maps one into the other. This material goods allows a search algorithm to produce the state illustration of a node from any stored node, as if all actions are deterministic, then a state s' is uniquely identified by another state s plus a path from s to s'. If it takes less space to correspond to a path involving s and s', then this approach to state training can save memory, even if at the cost of some computational overhead. For the tree width problem, this means the intermediate graph of a node can be generated from any node instead of only from the node's unswerving relatives, such as the start node. Thus, one only needs to preserve a single transitional graph, which can be personalized to develop into the transitional graph for any node in the search space. An motivating question is how to diminish the overhead of generating the transitional graph from one node to another. The response depends on the search approach, because finally our goal is to minimize the overhead of getting higher not just a single node save for a set of nodes.

Red condition: Each red node has a black parent.

Black condition: Each path from the root to an empty node contains exactly the same number of black nodes (this number is called the tree's black height).

Note that the red circumstance implies that the root of a red-black tree is black. The algorithm for inserting an element into a red-black tree is nearly indistinguishable to the standard algorithm for unbalanced binary trees. The main difference is that the constructor for building nodes, *N*, is replaced by a smart constructor [1] that maintains the invariants.

Algorithm for Red Black Tree

```
insert : (Ord a))a !RBTree a !RBTree a
insert a t =blacken (ins t)
where ins E N R E a E
ins (jN c l b r) a <b =bal c (ins l)b r ja ==b N c l a r a >b =bal c l
b (ins r)blacken (N l a r) N B l a r
```

IV. CONCLUSION

We had offered a novel arrangement of breadth-first and depth-first search that allows a single search algorithm to acquire the matching strengths of both. While our paper focuses on the tree width problem, many of the ideas have the prospective to be applied to other search troubles, especially graph-search harms with large encoding sizes, for which memory-reference neighborhood is the key to achieving good piece. Possibilities include model checking, where a large data structure that represents the current state is typically stored with each search node, and constraint-based forecast and scheduling, where a simple secular group is stored with each search node. As long as the similarities among unusual search nodes can be captured in a form that allows depth-first search to waste the state-representation locality in node expansions, the approach we have described could be ineffectual. Red-Black trees let us apply all dictionary operations in O(log n). Further, in no case are more than 3 rotations done to rebalance. Certain very complex data structures have data stored at nodes which requires a lot of work to adjust after a revolving red-black trees ensure it won't go off habitually.

REFERENCES

[1] Chris Okasaki. Purely Functional Data Structures. Cambridge University Press, 1998.

[2] Chris Okasaki. Functional Pearl: Red-Black trees in a functional setting. Journal OF FunctionalProgramming,9(4), July 1999.

[3] Richard S. Bird. Functional Pearl: On building trees with minimum height. Journal of Functional Programming, 7(4):441–445, July 1997.

[4] Ralf Hinze.Institut f¨ur Informatik III, Universit¨at Bonn Romerstraße 164.

[5] Rong Zhou Palo Alto Research Center 3333 Coyote Hill Road Palo Alto, CA 94304.Eric A. Hansen Dept. of Computer Science and Engineering Mississippi State University Mississippi State, MS39762.

[6] Paolo Baldan, Andrea Corradini, Javier Esparza, Tobias Heindel, Barbara K□onig, and Vitali Kozioura,"Verifying Red Black Trees", RTN 2-2001-00346.

[7] Jong Ho Kim, Helen Cameron, Peter Graham "Lock-Free Red-Black Trees Using CAS" October 20, 2011.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cli_ord Stein.Introduction to Algorithms. MIT Press, 2001. Second Edition.

[9] Kurt Mehlhorn. Data Structures and Algorithms 1: Sorting and Searching. EATCS,Monographs on Theoretical Computer Science. Springer, 1984.

[10] Paolo Baldan1, Andrea Corradini2, Javier Esparza3, Tobias Heindel3,Barbara K□onig3, and Vitali Kozioura3" Verifying Red-Black Trees" Research partially supported by DFGproject SANDS and EC RTN 2-2001-00346.

AUTHORS

First Author – Ms. Avinash Kaur, Faculty of GDRCET, Email: Sanjotre.avinash@gmail.com
Second Author – Ms. Purva Sharma, Student of GDRCET, Email: Purva.sharma06@gmail.com
Third Author – Ms.Apurva Verma, Student of GDRCET, Email: apurva.verma0011@gmail.com