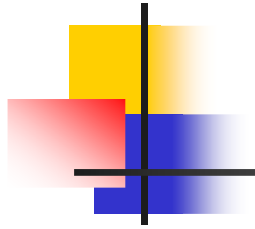




Algorithms 2005

Ramesh Hariharan



Divide and Conquer+Recursion

- Compact and Precise Algorithm Description



Min Finding

- Recursively find Minimum on the first $n-1$ items
- Compare the above min with the n th item; return the smaller of the two

$$T(n) = T(n-1) + 1$$

$$T(1) = 1$$



Sorting

- Recursively sort first $n-1$ items
- Insert the last item into the above order
 - Use sequential search with item shifting to insert
 - Use binary search?

$$T(n) = T(n-1) + O(n)$$

$$T(1) = 1$$



Sorting

- Recursively sort first $n/2$ and last $n/2$ items
- Merge the two sets.

$$T(n) = 2 * T(n/2) + O(n)$$

$$T(1) = 1$$



Merging

- Find the smaller of the $A[0]$ and $B[0]$; wlog, it is in $A[0]$.
- Return $A[0]$ followed by the recursively merge of $A[1..n-1]$ and $B[0..m]$

$$T(n,m) = \max(T(n-1,m), T(n,m-1)) + O(1)$$

$$T(n,0) = n, T[0,m] = m$$



Parallel Merging

- Recursively Merge odds in A with odds in B
- For each item $A[i]$, i even, binary search between the merged locations of $A[i-1]$ and $A[i+1]$; do likewise for $B[i]$, i even.

$$T(n,m)=T(n/2,m/2)+O(\log n)$$

$$T(n,0)=1, T[0,m]=1$$



Parallel Merging

Two Steps

- Compute final index j in the merged array for each $A[i]$ and likewise for $B[i]$
 - $j = \text{rank in } A + \text{rank in } B$
 - find rank in B using the algo on the previous slide
- Move $A[i]$ to $C[j]$; likewise for $B[i]$.



Balanced Tree Construction

N Sorted Items

- Make the middle item the root
- Recursively construct subtrees on the left and right halves; make these children of the root.

$$T(n) = 2 * T(n/2) + O(1)$$

$$T(1) = O(1)$$



Weighted Balanced Tree Construction

N Sorted Weighted Items

- Identify the “middle item” such that sum of weights either side are as balanced as possible, using a binary search.
- Make the middle item the root
- Recursively construct subtrees on the left and right halves; make these children of the root.

$$T(n) = \max_x (T(x) + T(n-x) + O(\log n))$$

$$T(1) = O(1)$$



Weighted Balanced Tree Construction

N Sorted Weighted Items

- Identify the “middle item” such that sum of weights either side are as balanced as possible, using doubling search from both ends

$$T(n) = \max_{x < n/2} (T(x) + T(n-x) + O(\log x))$$

$$T(1) = O(1)$$

How much is this?



Median Finding

- Generalize to finding a rank r item.
- Partition based on a pivot; suppose this results in a x $n-x$ split.
- If $r > x$ then recursively find the $r-x$ th item in the right half else recursively find the r th item in the left half.

$$T(n) = \max_y (T(y)) + n$$

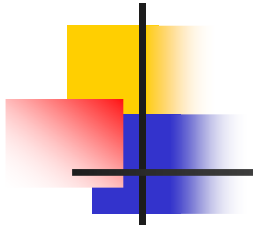
$O(n)$ if y is a constant fraction of n .



Median Finding

- Time taken for groups of size k
What should k be.

$$T(n) = T(n/k) + T(3n/4) + O(n)$$



Randomized Median Finding

Choose a random pivot

Size sequence

$n \quad X_1 \quad X_2 \quad X_3 \quad X_4 \dots$

Time = $O(n + X_1 + X_2 + X_3 + X_4 \dots)$

$E(n + X_1 + X_2 + X_3 + X_4 \dots) = E(n) + E(X_1) + E(X_2) + E(X_3) + E(X_4) \dots$



Crossing Lists and Arrays

- Arrays support binary search but insertions are expensive.
- Lists support quick insertions but searches are expensive.
- How can one cross the two? Fast search and insertion.

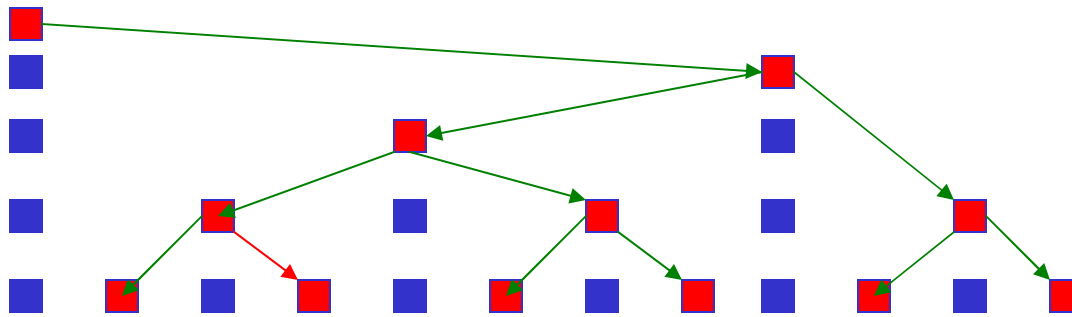
AVL trees, Red-Black Trees:
elaborate balancing conditions and cases

The Hybrid

Sampling and Fractional Cascading

Assign levels to items

- Give all items level 0
- Increment level of odd items; then recurse on these.
- Each item $A[i]$ at level j has pointers to the highest below-level- j items to the left and right.
- Height $O(\log n)$

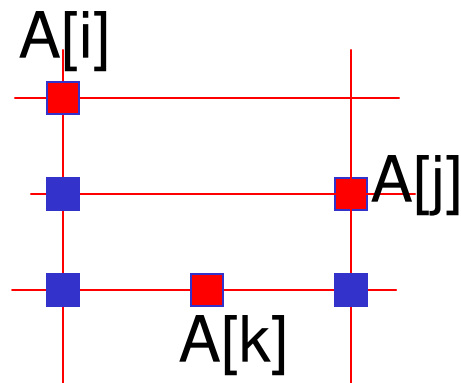




Searching the Hybrid

Start at the top and search downwards.

- Suppose the search has been narrowed to between $A[i]$ (level x) and $A[j]$ (level $y < x$)
- There is exactly one item $A[k]$ between $A[i]$ and $A[j]$ at level $y-1$.
- Given $A[j]$, $A[k]$ can be identified in constant time
- One comparison can narrow the search down to the range $A[i]..A[k]$ or $A[j]..A[k]$
- Level reduces by 1, Recurse; Timing $O(\log n)$



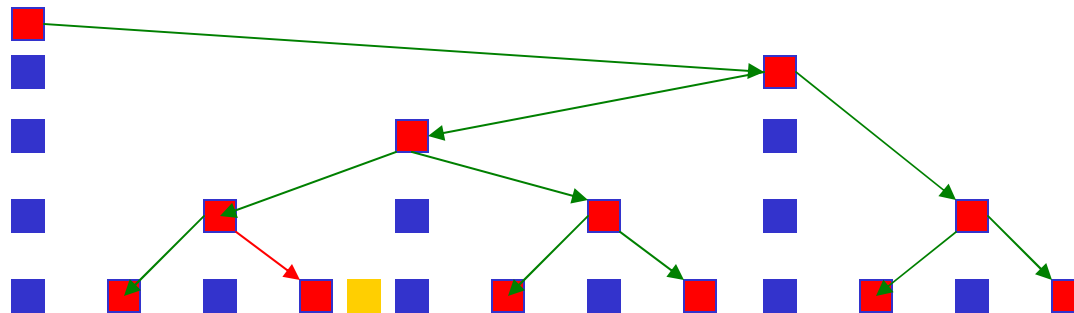


Inserting into the Hybrid

Search and insert at the appropriate place at level 0.

- Invariant Fails:

There are more than one $A[k]$ s between $A[i]$ and $A[j]$ at **level $y-1$** .

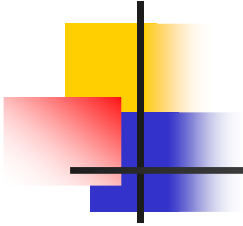




Inserting into the Hybrid

- Relax the invariant: Allow 1 or 2 such $A[k]$ s
- Search and insert at the appropriate place at level 0.
- If this creates more than 2 (i.e., 3) consecutive red nodes at a particular level, increase the level number of the middle node, and recurse upwards.
- Timing $O(\log n)$

Data structurally, what links are maintained??
Deletion??



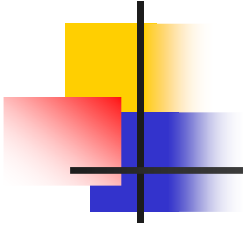
DFS in Graphs

Aim: To visit each node and walk on each edge at least once, starting at node s .

- Visit s
- Visit a neighbour t of s and recurse from t : **Incomplete Coverage**
- For each neighbour t of s , visit t and then recurse from t . **Infinite Loop**

Mark a vertex as visited when it is visited for the first time;
recurse only at unmarked (i.e., newly visited) vertices.

```
Mark s  
For each neighbour t of s  
  {If t is visited for the first time  
    { Mark t  
      Recurse from t  
    }  
  }
```



BFS in Graphs

Aim: To visit each node and walk on each edge at least once, starting at node s .
Nodes must be visited in increasing order of distance from s .

- Inductive Invariant: Nodes with distances at most i from s have been visited.
 - X_i : nodes at distance i from s .
 - $\text{Neighbourhood}(X_i)$: neighbours of nodes in X_i
 - X_{i+1} : $\text{Neighbourhood}(X_i)$ – already visited nodes
 - Given $X_1..X_i$, Compute X_{i+1} . and recurse on $X_1..X_{i+1}$.



BFS Implementation

- Use a queue.

Initialize queue to {s}

Mark s as visited

while queue not empty {

 x=dequeue

 Enqueue and mark each unmarked neighbour of x

}



Closest Pair

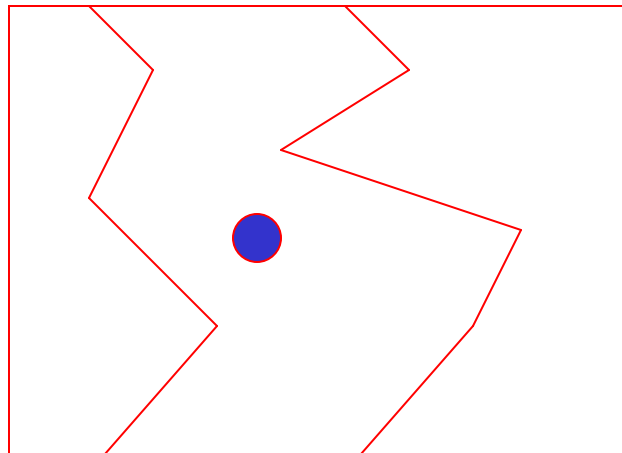
Given n points on a plane, find the closest pair

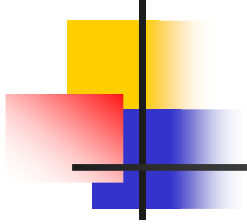
- Choose an arbitrary direction.
- Sort points along that direction.
- Split into halves and find closest pair in each half.
- Find closest pair straddling halves. How is this done in linear time?



Fractional Cascading Question

Given downward monotonic chains comprising n points in all, show how to set up a data structure which will perform point location (which two chains does it lie between?) fast.





Thank You
