Weekly Assignment 7

Munia Humaira
Student ID: 21116435
Algorithm Design and Analysis

October 31, 2024

Solution 1:

Given statement, for every connected undirected weighted graph ($G = \langle V, E, w \rangle$, there exists a vertex s such that running BFS from s on the unweighted version $G^{(u)}$ produces a shortest-paths tree for G.

- To disprove the statement, we need a graph where any BFS from any starting vertex fails to align with the shortest paths based on the weights in G.
- In an unweighted graph $G^{(u)}$, BFS finds the shortest path in terms of the number of edges. However, when weights are considered, the shortest path depends on the actual edge weights.
- Let's consider a graph with four vertices $V = a, b, c, d$ and edges as follows:
  - (a, b) with weight $w(a, b) = 1$
  - (a, c) with weight $w(a, c) = 2$
  - (b, d) with weight $w(b, d) = 1$
  - (c, d) with weight $w(c, d) = 1$
- This graph is connected, and in the unweighted version $G^{(u)}$, all edges have the same "weight" for BFS purposes.
- In the weighted graph G, the shortest path from $a \to d$ is through $b : a \to b \to d$, with total weight $1 + 1 = 2$. In the unweighted version $G^{(u)}$, BFS from a would yield $a \to d$ is through $b : a \to b \to d$ as a shortest path in terms of edge count, which is not the shortest path in the weighted graph G.
- Regardless of which vertex we start BFS from, the paths found in $G^{(u)}$ will not always correspond to the shortest paths in G, disproving the statement

Solution 2:

The Merge-Sort-Third divides the array into approximately 1/3 and 2/3 pieces, rather than the traditional half-half split.

- In original Merge Sort one uses: $q = [(p + r)/2]$. This version uses: $q = p + [(r - p + 1)/3] - 1$. There is an Insertion Sort for small arrays (< 3 elements)
- The recurrence relation: Let $n = r - p + 1$ (the size of array).
  When $n < 3 : T(n) = \Theta(n^2)$ (Insertion Sort)
  When $n \geq 3$: * First third: $T([n/3])$ * Remaining two-thirds: $T(n - [n/3])$ * Merge cost: $\theta(n)$ Therefore:

$$T(n) = \begin{cases} \theta(n^2), \text{if } n < 3 \\ T([n/3]) + T(n - [n/3]) + \theta(n), \text{if } n \geq 3 \end{cases}$$

- To solve this, we write, $T(n) = T(n/3) + T(2n/3) + cn$
  Assume $T(n) \leq kn \log n$ for some constant k. Substitute:
  $T(n) = T(n/3) + T(2n/3) + cn \leq k(n/3)\log(n/3) + k(2n/3)\log(2n/3) + cn$
  $= (kn/3)(\log n - \log 3) + (2kn/3)(\log n - \log(3/2)) + cn$
  $= kn \log n - (kn/3)\log 3 - (2kn/3)\log(3/2) + cn$
  $= kn \log n - kn(\log 3/3 + \log(3/2)/3) + cn$
  $\leq kn \log n$ (for large enough k)
  Therefore, $T(n) = \theta(n \log n)$

The intuition is even though we're splitting unevenly (1/3 and 2/3) we're still dividing the problem into smaller subproblem and combining with linear work, this leads to the same asymptotic behavior as regular Merge Sort.

Solution 3:

Stack depth calculation: In given Quicksort-New algorithm:

- In each recursive call we process the smaller partition (1 element) first.
- Then we update p or r and continue with the loop for the larger partition.
- This means we only add one frame to the call stack for each partition.
- The number of partitions needed to reduce the array to a single element is $\log_2(n)$.
- Therefore, the worst-case stack depth for Quicksort-New is $\theta(\log n)$.

Comparison with original Quicksort:
The original *Quicksort* algorithm from the textbook does not have this optimization. In the worst case, it would recurse on both partitions, potentially leading to a stack depth of $\theta(n)$ if the partitions are consistently unbalanced.

Solution 4:
The pivot is chosen uniformly at random from n distinct elements. After partitioning, the pivot will be in its final sorted position.

Since the array has distinct elements, the pivot could end up in any of the n positions with equal probability $1/n$. If the pivot ends up in position $i(1 \leq i \leq n)$, the smaller piece will have length i.

So, the expected length of the smaller piece is the average of all possible lengths, weighted by their probabilities.

Then,

$E[length] = (1 \cdot 1/n) + (2 \cdot 1/n) + ... + (n \cdot 1/n)$

This is equivalent to:

$E[length] = (1/n) \cdot (1 + 2 + ... + n)$

The sum of the first n natural numbers is given by the formula $n(n+1)/2$.

Therefore:

$E[length] = (1/n) * (n(n+1)/2) = (n+1)/2$

Since n is even, we can express this as:

$E[length] = n/2 + 1/2 = (n+1)/2$

Thus, the expected length of the smaller piece, when n is even, is $(n+1)/2$.

Solution 5:

Assume, for the sake of contradiction, that Randomized-QuickSort is correct but there exists a subset $Z_{i,j}$ for which no element is ever chosen as a pivot.

Consider, in the final sorted array for Randomized-QuickSort to be correct, all elements must be in their correct positions. For any two adjacent elements z_k and z_{k+1} in the sorted array, there must have been a comparison between them at some point during the algorithm's execution.

This comparison could only have happened in one of two ways:

a) One of z_k or z_{k+1} was chosen as a pivot.

b) Some element between z_i and z_j (inclusive) was chosen as a pivot, partitioning the array such that z_k and z_{k+1} ended up in different partitions.

However, we assumed that no element from $Z_{i,j}$ was ever chosen as a pivot. This means that for all pairs of adjacent elements z_k and z_{k+1} where $i \leq k < j$, there was never a direct comparison or a partitioning that separated them.

As a result, the algorithm has no way to determine the correct relative order of the elements in $Z_{i,j}$.

This contradicts our assumption that Randomized-QuickSort is correct, as it cannot guarantee that the elements in $Z_{i,j}$ are in their correct sorted positions.

Therefore, for Randomized-QuickSort to be correct, it is necessary that for every $i, j \in 1, ..., n$ with $i < j$, some member of $Z_{i,j}$ must be chosen as a pivot at some point during the algorithm's execution.