CS260 Homework 3

February 19, 2020

Deadline. The homework is due **Monday Mar. 9**. You can submit your write-up to me after class or before that.

Late days. You have five late days to use throughout the quarter. This includes homework, course project proposal and project final report.

Collaboration policy. Discussion on the homework problems is allowed, after you have thought about the problems on your own. It is OK to get inspiration (but not solutions) from books or online resources, again after you have thought about the problems on your own. You must cite your collaborators and resources fully and completely (e.g., *Alice explained to me the definition of work in Problem 1* or *I used Wikipedia site* `https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)` *about Master theorem for Problem 3*). You must write up your solution independently: close the book and all of your notes, and with no one else helping you when you are writing your final answers.

Write-up. Please do not use hand-writing submissions. Using LaTeX is highly-recommended. For all problems, please explain how you get the answer instead of directly giving the final answer.

Programming Problems. For all programming assignments, you need to submit your source code through iLearn by 23:59 March 9. In the write-up, you need to provide detailed report of the experimental results and analysis. The programming part gives you up to a 5% bonus when you get very good performance or write good "report" to analyze the experiments.

1 Parallel Construction of BSTs (15pts)

In class we've seen a parallel algorithm to construct a tree from a list of elements. The first step is to sort all keys. After that, we pick up the median of the list as the root, and recursively build the left and right subtrees respectively. The cost of the algorithm is bounded by the sorting step. For the sorting algorithms we learned in class, the work is $O(n \log n)$ and the depth is $O(\log^2 n)$. There exist parallel sorting algorithms with $O(n \log n)$

work and $O(\log n)$ depth, but they are more complicated.

The implementation of a parallel sorting, however, is itself a hard problem. In this problem, you will try to design another construction algorithm without using sorting as a subroutine.

Answer the following questions:

(a) (4pts) Design a divide-and-conquer algorithm to construct a tree from n unsorted keys using the `union` algorithm (without calling sorting algorithms). Your algorithm must have polylog depth and $O(n \log n)$ work. You can assume a `union` algorithm on two trees with size n and $m \leq n$ has work $O\left(m \log\left(\frac{n}{m} + 1\right)\right)$.

(b) (2pts) Write down the recurrence of work and depth for your algorithm, respectively.

(c) (4pts) Prove the work and depth of your algorithm.

(d) (4pts) Can this algorithm be used for sorting? Describe a parallel sorting algorithm based the algorithm you designed in (a). Your input is an array of unsorted keys and your output should be a sorted array of the input keys. Show the work and depth of this sorting algorithm.

(e) (1pt) This algorithm in (d) resembles which of the following sorting algorithm the most?

A. Quicksort B. Merge sort C. Bubble sort D. Selection sort

2 Parallel Treaps and Programming (60 pts)

You can use the server provided by the department, or any other multicore machine that you have access to (better with more than 10 cores). Instructions of logging into the ti-05 server, sample code, and suggestions about write-up are available in Homework 1.

In class we've seen some parallel tree algorithms. Although the general idea works for multiple balancing schemes, it still can be painful to deal with the rotations in the `join` algorithm. In this part, we'll try to implement these algorithms simply.

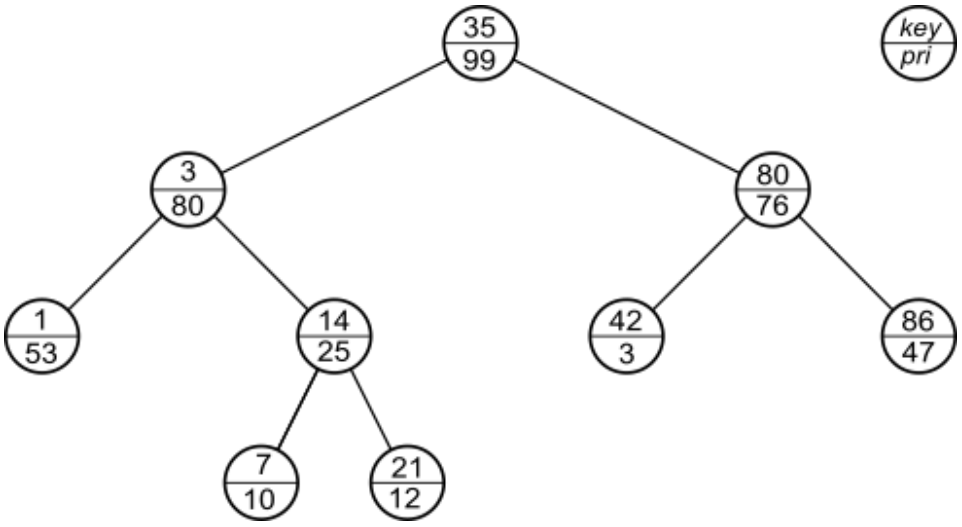
Please note that there are regular problems (not just programming) in this problem. You need to include the answer in your writeup.

2.1 Treaps (15pts)

The basic idea is to implement a **treap** data structure. Treaps are randomized balanced binary trees. To maintain a set of keys using treaps, each key will be assigned a random **priority**, e.g., a random value in a some range (e.g, a real number in [0,1]), or based on a random permutation of $1..n$. For simplicity, we assume all priorities are **unique** (otherwise you can break ties arbitrarily). When constructing the tree, we will guarantee that the priority of a node is always higher than its both children. In other words, all the keys on

the tree forms a binary search tree, not necessary balanced, and all the priorities on the tree form a heap.

An example of a treap is shown in the following figure.



Source: http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_8_D

Answer the following questions:

(a) (2pts) Given the input keys and their priorities, what is the root of the tree?

(b) (3pts) For AVL or red-black trees, on a list of input keys, the shape of the tree is not fixed, and there are multiple valid state of the tree. However, this is no the case for treaps. Given the input keys and their priorities, show that the final state of a treap built upon these keys is unique.

(c) (5pts) The height of a treap depends on the random priority. However, the probability distribution of the height of a treap is exactly the same as the probability distribution of the number of rounds in a **quicksort** algorithm, as long as we choose pivots in the quicksort uniformly at random. Please prove this.

(d) (5pts) Based on the answers for the previous problems, propose an algorithm to construct a treap in $O(n \log n)$ work and $O(\log^2 n)$ depth with high probability, and implement it.

Because of the conclusion in (c), the height of a treap is $O(\log n)$ with high probability. As a result, treap is also a "balanced" tree.

2.2 Joining and Spliting Treaps (25pts)

Joining two treaps `join`(T_L, k, T_R) can be very easy if k has a higher priority than all keys in T_1 and T_2. Then this `join` only need to connect T_L and T_R using k, in a constant time. We call such a `join` invocation a **direct** `join`.

Answer the following questions:

(a) (2pt) Show how to determine if the priority of k is higher than *all* keys in T_L and T_R in a constant time.

(b) (5pts) In the `split` algorithm, show that when a `join` algorithm is called, it is always a direct `join`.

(c) (5pts) Show that the cost of `split` on a treap is $O(\log n)$ with high probability.

(d) (13pts) Implement such a direct `join` and a `split` algorithm (8pts). Test the performance of your code(5pts). You can use your algorithm in 2.1(d) to construct the input trees.

(e) (2pts bonus) Describe a **general** `join` algorithm on two treaps and a key k with priority p_k. Now you **cannot** assume that p_k is higher than the priorities of all keys in the input trees.

2.3 Union/intersection/difference on two treaps (20pts)

Since we can implement a `split` and a direct `join` on treaps, we can also implement the `union` algorithm. Recall the algorithm:

```
union(T1,T2) {
  if T1 is empty return T2;
  if T2 is empty return T1;
  k = key at the root of T2;
  L2 = the left subtree of T2;
  R2 = the right subtree of T2;
  (L1, b, R1) = split(T1, k);
  in parallel:
    TL = union(L1, L2);
    TR = union(R1, R2);
  return join(TL, k, TR);
}
```

Answer the following questions:

(a) (3pts) Now that we know the `split` operation only calls direct `join`s, we still need to guarantee the last join at the last line is a direct join. Please describe how you can guarantee the `join` at the last line is always a direct join by briefly modifying the algorithm.

(b) (2pts) Does the same modification in (a) work also for `intersection` and `difference` algorithms? If so, please describe the algorithms. If not, explain the reason.

(c) (15pts) Implement the `union` algorithm based on your proposed idea in (b). Test the performance of your algorithm (5pts). You can consider comparing your algorithms

with direct implementations based on arrays or trees. For example, for union, you can compare with the merging algorithm as in the merge sort, and with inserting all keys in one tree directly into the other tree (5pts). Please also consider different (relative) input sizes (e.g., two sets of equal sizes, or very different sizes) and different key overlaps. (5pts)

(d) (5pt bonus) Describe and implement the `intersection` and `difference` algorithms on treaps.