

Assignment 3 - Hady Ibrahim

SFWRENG 2CO3: Data Structures and Algorithms–Winter 2023

Deadline: March 5, 2023

Department of Computing and Software
McMaster University

Please read the *Course Outline* for the general policies related to assignments.

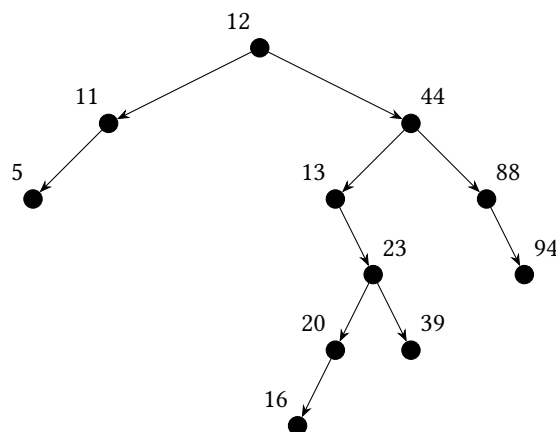
Plagiarism is a serious academic offense and will be handled accordingly.
All suspicions will be reported to the Office of Academic Integrity
(in accordance with the Academic Integrity Policy).

This assignment is an *individual* assignment: do not submit work of others. All parts of your submission *must* be your own work and be based on your own ideas and conclusions. Only *discuss or share* any parts of your submissions with your TA or instructor. You are *responsible for protecting* your work: you are strongly advised to password-protect and lock your electronic devices (e.g., laptop) and to not share your logins with partners or friends! If you *submit* work, then you are certifying that you have completed the work for this assignment by yourself. By submitting work, you agree to automated and manual plagiarism checking of all submitted work.

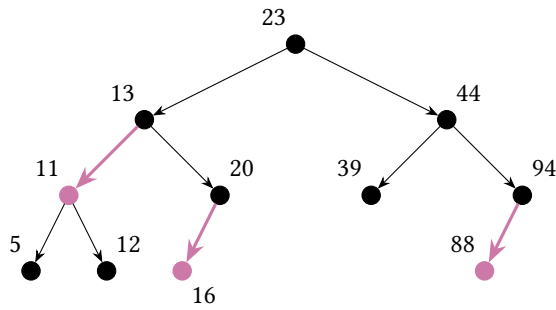
Late submission policy. Late submissions will receive a late penalty of 20% on the score per day late (with a five hour grace period on the first day, e.g., to deal with technical issues) and submissions five days (or more) past the due date are not accepted. In case of technical issues while submitting, contact the instructor *before* the deadline.

Problem 1. Consider the sequence of values $S = [12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5]$.

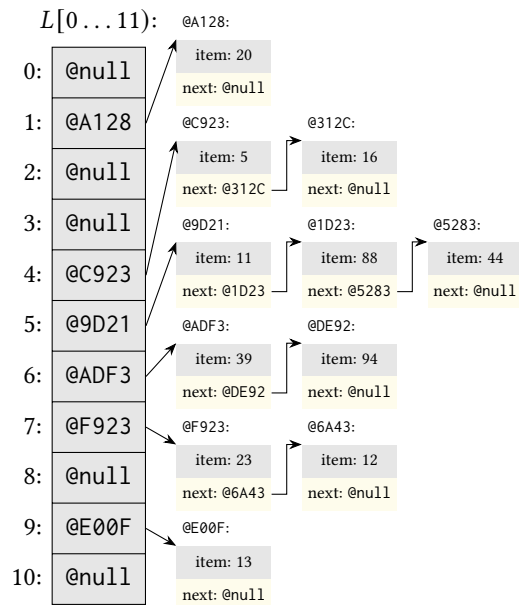
P1.1. Draw the binary search tree obtained by adding the values in S in sequence.



P1.2. Draw the red-black tree obtained by adding the values in S in sequence.



P1.3. Consider the hash-function $h(k) = (2k + 5) \bmod 11$ and a hash-table of 11 table entries that uses hashing with separate chaining. Draw the hash-table obtained by adding the values in S in sequence.



P1.4. Consider the hash-function $h(k) = (3k + 2) \bmod 11$ and a hash-table of 11 table entries that uses hashing with linear probing. Draw the hash-table obtained by adding the values in S in sequence.

$L[0 \dots 11)$:

0:	16
1:	5
2:	44
3:	88
4:	11
5:	12
6:	23
7:	20
8:	13
9:	94
10:	39

Problem 2. We say that a hash function $h : \mathcal{U} \rightarrow \mathbb{N}$ that maps values from a set \mathcal{U} to integers in the range $[0 \dots M)$ is *n-perfect* if there exists at most n distinct values $u_1, \dots, u_j \in \mathcal{U}$ such that $h(u_1) = \dots = h(u_j)$.

P2.1. Consider the hash function $h(k) = (2k + 5) \bmod 11$. Is this hash function 2-perfect for the inputs $0, \dots, 21$? Explain why or why not.

Answer: This function is 2-perfect for the inputs $0, \dots, 21$ since after computation, there is only 2 values mapped to one spot in the hash table. In summary, since the function has $2k$, as you loop through the values, the new value will most often be 2 more than the previous. This is until you hit 11 as it is mod 11, then it will start back at either 0 or 1 depending on if the previous value was 9 or 10. The following is the computation.

v	$h(v)=(2v+5)\bmod 11$	v	$h(v)$	v	$h(v)$	v	$h(v)$
0	5	6	6	12	7	18	8
1	7	7	8	13	9	19	10
2	9	8	10	14	0	20	1
3	0	9	1	15	2	21	3
4	2	10	3	16	4		
5	4	11	5	17	6		

Rearrange to see that there are only two values per mapping. We can see that if there was 1 more value, it would no longer be 2-perfect:

$h(v)$	v	$h(v)$	v	$h(v)$	v	$h(v)$	v
0	3, 14	3	10, 21	6	6, 17	9	13, 2
1	9, 20	4	5, 16	7	12, 1	10	8, 19
2	4, 15	5	11, 0	8	7, 18		

P2.2. Prove that a hash function $h : \mathcal{U} \rightarrow \mathbb{N}$ can only be *n-perfect* if $|\mathcal{U}| \leq n \cdot M$.

is max h height and the right subtree is h since we just added T_2 which is a tree of height h . We also know that all values in T_2 are greater than the values in the current T_1 , including the root. This allows us to put the root of T_2 as the right child of m , ensuring we keep BST properties.

4) return T_1

To conclude, the resulting tree will be of height $h + 1$ since we took 2 tree of height h and added 1 level to it (the root). We also know it keeps BST properties as we know all values in T_2 are larger than the maximum value in T_1 , which allows us to put T_2 in the right subtree of the root of new T_1

P3.2. Assume that T_1 and T_2 are red-black trees with the same black height h . Show how to implement a SETUNION operation that returns a red-black tree in $\sim h$.

- Remove min value in T_2 and store it in variable m
Removing the min value is an h complexity function. This can make the tree T_2 a height of either h or $h - 1$ If left child is black and his parent is black then you can
- Calculate black height of T_2 and store it as h_{new}
- Make the min value of T_2 (m) the root of our new tree T_3

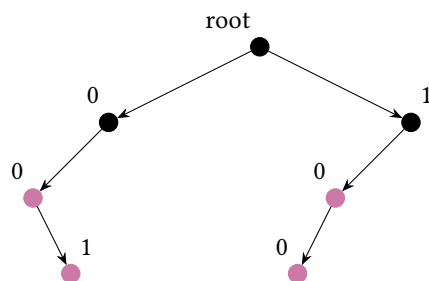
P3.3. Assume that T_1 and T_2 are red-black trees with black heights $h_1 > h_2$. Show how to implement a SETUNION operation that returns a red-black tree in $\sim h_1$.

Problem 4. Consider *binary strings* (sequences of zeros and ones).

P4.1. Design a data structure BSET that can be used to represent *sets of binary strings* such that any binary string W of length $|W| = N$ can be added or removed in $\sim N$ and such that one can check whether W is in the data structure in $\sim N$. Sketch why your data structure BSET supports the stated operations in $\sim N$.

The data structure would be a binary tree where any left child is a 0 and any right child is a 1. Each bitstring is stored as a path down the tree. Each node has a flag (represented by red), which represents if the node is the terminal bit of a string.

In the following example tree, this would represent the data structure holding the following bitstrings: "00", "001", "10", "100"



Adding:

To add a bitstring, you must traverse the path in relation to the new bitstring, and for any bit that isn't already in the data structure, add it. Once reaching the terminal bit of the bitstring, make the last node a red node. This traverses the tree only for the amount of bits in the bitstring, making it $\sim N$ complexity.

ex. Adding bitstring "1100", traverse tree 1->1->0->0, and for any non-existence node, create it. Make the last 0 a red node.


```

15:   end for
16:   if currNode.color == "red" then
17:     print string "//this is to handle printing the prefix if it is in the tree."
18:   end if
19:   PRINTPREFIX(currNode.left, currNode.data + "") "//print all bitstrings in left subtree"
20:   PRINTPREFIX(currNode.right, currNode.data + "") "//print all bitstrings in right subtree"
21: else
22:   string.append(node.data) "//Add current node data to string"
23:   if node.color == "red" then
24:     "//if current node is a terminal node, print its string (the path traversed to get here)"
25:     print string
26:   end if
27:   PRINTPREFIX(currNode.left, currNode.data + "") "//print all bitstrings in left subtree"
28:   PRINTPREFIX(currNode.right, currNode.data + "") "//print all bitstrings in right subtree"
29: end if

```

You can call the method via `PRINTPREFIX(root, prefix)`. The complexity is $\sim N + k$ since the algorithm traverses the path of the given prefix which is $\sim N$. Then it traverses every path below it recursively. Anytime it gets to a red node, it prints out the path it took to get there, including the current node (note: it does not need to retrace the graph backwards as it holds the path in a string). This second part is k as it checks only all characters after the prefix. As long as the data structure makes sure to only have paths that have a red node at the end (explained in q4.1 where deleting a string will remove all terminal black nodes), it will always be $\sim k$ since the longest path to traverse is as long as the largest binary string to print.

P4.3. Professor X claims to have developed a data structure `BSETX` to which any binary string W of length $|W| = N$ can be added in $\sim N$. Furthermore, Professor X claims that `BSETX` provides *ordered iteration*: one can iterate over all $M = |S|$ strings in a set S , implemented via `BSETX`, in a lexicographical order in $\sim M + T$ in which T is the combined length of the M strings. Professor X claims that this method of sorting binary strings proves that the worst-case lower bound for sorting M binary strings is *not* $\sim M \log_2(M)$ comparisons. Argue why Professor X is wrong.

We note that strings S_1 and S_2 are *lexicographical ordered*, denoted by $S_1 < S_2$, if S_1 comes before S_2 in an alphabetical sort (e.g., as used in a dictionary). Next, we formalize $S_1 < S_2$ for binary strings: we have $S_1 < S_2$ if S_1 and S_2 are equivalent up to the $0 \leq i \leq \min(|S_1|, |S_2|)$ -th character ($S_1[0] = S_2[0]$, ..., $S_1[i-1] = S_2[i-1]$) and either $|S_1| = i < |S_2|$ or the $(i+1)$ -th character of S_1 comes before the $(i+1)$ -th character of S_2 (in which case $S_1[i] = 0$ and $S_2[i] = 1$). For example, $0 < 00$ and $00 < 01$, but not $100 < 10$.

Assignment Details

Write a report in which you solve each of the above problems. Your submission:

1. must be a PDF file;
2. must have clearly labeled solutions to each of the stated problems;
3. must be clearly presented;
4. must *not* be hand-written: prepare your report in \LaTeX or in a word processor such as Microsoft Word (that can print or exported to PDF).

Submissions that do not follow the above requirements will get a grade of zero.

Grading

Each problem counts equally toward the final grade of this assignment.