# 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.

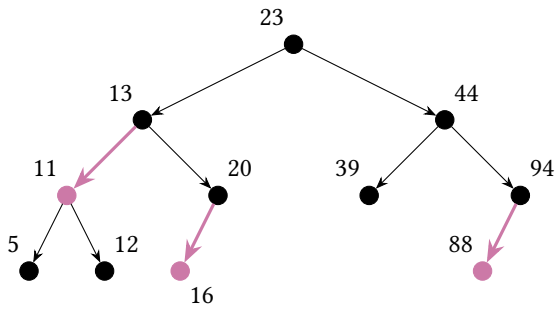
**Plagiarism is a *serious academic offense* and will be handled accordingly.**
**All suspicions will be reported to the *Office of Academic Integrity***
**(in accordance with the Academic Integrity Policy).**

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.

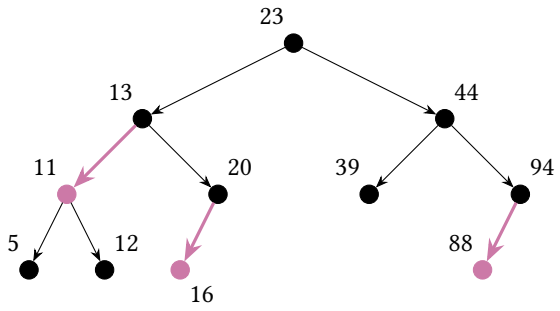
*Late submission policy.* Late submissions will receive a late penalty of 20% on the score per day late (with a five hour grace period on the first day, e.g., to deal with technical issues) and submissions five days (or more) past the due date are not accepted. In case of technical issues while submitting,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

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

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

**Answer:** This function is 2-perfect for the inputs 0,...,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)mod 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 $h : \mathcal{U} \to \mathbb{N}$ can only be $n$-perfect if $|\mathcal{U}| \leq n \cdot M$.

**Answer:**

Assume $h : \mathcal{U} \to \mathbb{N}$ is n-perfect and let $u_1, \ldots, u_j$ be j distinct values in $\mathcal{U}$ such that $h(u_1) = \cdots = h(u_j)$. Then we will have that $j \leq n$.

$\mathcal{S}_i = \{x \in \mathcal{U} \mid h(x) = i\}$ -> In other words, $\mathcal{S}_i$ has elements in $\mathcal{U}$ that map to $i$

This means, $\mathcal{U} = \sum_{i=0}^{M-1} \mathcal{S}_i$ where $|\mathcal{U}| = M$

since $h$ is n-perfect, we know $|\mathcal{S}_i| \leq n$

$\mathcal{U} = \sum_{i=0}^{M-1} \mathcal{S}_i$
$|\mathcal{U}| = |\sum_{i=0}^{M-1} \mathcal{S}_i|$
$= \sum_{i=0}^{M-1} |\mathcal{S}_i|$
$\leq \sum_{i=0}^{M-1} n$
$= nM$

In conclusion, we have proven that assuming $h : \mathcal{U} \to \mathbb{N}$ is a n-perfect hash function, then $|\mathcal{U}| \leq nM$ holds.

As we provided an upper bound on the number of elements in $\mathcal{U}$ that can be hashed without collisions, we know that adding any more elements will result in more than n values being mapped to a single hash value, which violates n-perfect rules.

P2.3. Can a general purpose hash function be $n$-perfect for any $M$? Argue why or why not.

**Answer:**

No a general-purpose hash function cannot be n-perfect for any M because a general-purpose hash function must be able to take in any set of input and ensure each hash value contains $\leq n$ values from the input. This means it needs to handle inputs such as lists that have near infinity values (huge M), which is impossible to be n-perfect as you'd have to have an unrealistic number of spots in the hash-table to put the values in.

Mathematically, the Probability a value $V_r$ in the input set collides with a previously hashed value is:

Probability($V_r$ collides with $V_1$) $\cdot$ Probability($V_r$ collides with $V_2$) $\cdot \ldots$ Probability($V_r$ collides with $V_{r-1}$)

1/M + 2/M + 3/M... n/M, where n is the $n_{th}$ element in the input list.

FIX THISSSSSSSSSSSSSSSSSSSSS

the probability to have a collision is $\sim 1 - \frac{n}{M}$

**Problem 3.** Consider non-empty binary search trees $T_1$ and $T_2$ such that all values in $T_1$ are smaller than the values in $T_2$. The SetUnion operation takes binary search trees $T_1$ and $T_2$ and returns a binary search trees holding all values originally in $T_1$ and $T_2$ (destroying $T_1$ and $T_2$ in the process).

P3.1. Assume the binary search trees storing $T_1$ and $T_2$ have the same height $h$. Show how to implement the SetUnion operation in $\sim h$ such that the resulting tree has a height of at-most $h + 1$.

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 $\sim h$.

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 BSSet 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 BSSet supports the stated operations in $\sim N$.

P4.2. Let $W$ of length $|W| = N$ be a binary string and let $S$ be a BSSet set. Provide an algorithm that prints all strings $V \in S$ that start with the prefix $W$ (the first $|W|$ characters of $S$ are equivalent to $W$). Your algorithm should have a worst-case complexity of $\sim N + k$ in which $k$ is the number of characters printed to the output.

P4.3. Professor X claims to have developed a data structure BSSetX to which any binary string $W$ of length $|W| = N$ can be added in $\sim N$. Furthermore, Professor X claims that BSSetX provides *ordered iteration*: one can iterate over all $M = |S|$ strings in a set $S$, implemented via BSSetX, 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 $S_1 \prec S_2$, if $S_1$ comes before $S_2$ in an alphabetical sort (e.g., as used in a dictionary). Next, we formalize $S_1 \prec S_2$ for binary strings: we

have $S_1 \prec S_2$ 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, $0 \prec 00$ and $00 \prec 01$, but not $100 \prec 10$.

## 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.