CSE 21 Fall 2022
Homework 5 , Solution guide

INSTRUCTIONS
These are problems you should only discuss with your homework group and instructional staff. Do exactly one of the review problems, and all the remaining problems.
KEY CONCEPTS Inclusion/Exclusion, Ranking/Unranking, Huffman Encoding, Graph representations.

Review question for homework 1 -20 points A group of n people are playing frisbee, throwing the disk from one person to another. Amazingly, no one ever drops the frisbee or misses the throw. Let $Catch_i$ be the number of times person i has caught the frisbee, $Throw_i$ the number of times person i has thrown the frisbee. Prove that at the end of the game, $\sum_i Catch_i = \sum_i Throw_i$.

We can prove this by induction on the number of throws. If there have been 0 throws, all $Catch_i = 0$ and all $Throw_i = 0$, so $0 = \sum_i Catch_i = \sum_i Throw_i$.

Assume after t throws, $\sum_i Catch_i = \sum_i Throw_i$. Then the next throw is between from some person i_1 to some person i_2. If we let $Catch'_i$ be the number of catches after this throw, $Catch'_i = Catch_i$ for $i \neq i_2$, and $Catch'_{i_2} = Catch_{i_2}+1$. Similarly, if we let $Throw'_i$ be the number of throws after this throw, $Throw'_i = Throw_i$ for $i \neq i_1$, and $Throw'_{i_1} = Throw_{i_1}+1$. So $\sum_i Catch'_i = Catch'_{i_2}+\sum_{i\neq i_2} Catch'_i == Catch_{i_2}+1+\sum_{i\neq i_2} Catch_i = 1+\sum_i Catch_i = 1+\sum_i Throw_i = 1+Throw_{i_1}+\sum_{i\neq i_1} Throw_i == Throw'_{i_1}+\sum_{i\neq i_1} Throw'_i == \sum_i Throw'_i$. So the claim is still true after $t+1$ throws.

So by induction on the number of throws, the claim is true after any number of throws.

Review question for homework 2- 20 points Say we are given two sorted lists $A[1..k]$ and $B[1..n]$ where $k < n/2$. Give an algorithm that decides whether the lists intersect, i.e., is some $A[i] = B[j]$? Your algorithm should work in time $O(k \log n/k)$. Hint: use a method similar to window encoding, breaking B up into blocks of size n/k and using binary search for each block.

Our algorithm is below. It uses a Binary search procedure that given a sorted list $B[1..m]$ and a value V returns True if V is in the list and False otherwise. : IntersectSorted(A[1..k], B[1..n]): sorted lists of integers.

1. If k ==1 return $BinarySearch(B[1..n], A[1])$.
2. Else let $w = \lceil n/k \rceil$

3. $J = 1$.
4. FOR $I = 1$ to k do:
5. While $A[I] > B[Jw]$ and $Jw \leq n$ do: $J++$.
6. IF $BinarySearch(B[(J-1)w+1, ..min(n, Jw)], A[I])$ then return True.
7. Return False.

First, for the running time, we do at most one binary search in a range of at most w for each I. In addition, since we only increment J and if it exceeds k, $Jw > n$ and the while loop condition is false, we increment J at most k times, and check the condition for the while loop at most once per I when it is false. Thus, the total time is $O(k+k+k\log w) = O(k \log n/k)$, since the condition $n > k/2$ implies the last term is the largest.

The algorithm maintains the invariant that if $J > 1$, then $B[(J-1)w] < A[I]$. We only increment J if $A[I] > B[Jw]$, so after we increment J it is true. The other place where we change I and J is when we increment I in the FOR loop, but since A is non-decreasing, if we increment I, $A[I]$ only increases.

For each I then, after the $While$ loop terminates, we either have $Jw > n$ and $B[(J-1)w] < A[I]$ or $B[(J-1)w] < A[I] < B[Jw]$. In either case, $A[I]$ cannot match any value outside the sub-array $B[(J-1)w+1...min(n, Jw)]$. So by the correctness of binary search, we return True at this point if and only if $A[I]$ is in the array $B[1...n]$. Since we do this for all I, overall we return $True$ if and only if the arrays intersect.

Review question for homework 3- 20 points Remember that the Fibonacci sequence is given by $F(0) = F(1) = 1, F(n) = F(n-1) + F(n-2)$ for $n \geq 2$. On the review question 1 in the previous assignment, the following identity was proved, $\forall 0 < k < n$, $Fib(n) = Fib(k)Fib(n-k) + Fib(k-1)Fib(n-k-1)$. Use this identity to give a divide-and-conquer recursive algorithm that, given n, returns $Fib(n)$ and $Fib(n+1)$. Prove correctness using the identity, and give a time analysis assuming arithmetic operations are constant time.

If $n \leq 3$ we can just use a table look-up.

Assume $n > 3$ is even and we use the identity with $k = n/2$ for both n and $n+1$. Then $Fib(n) = Fib(n/2)^2 + Fib(n/2-1)^2$ and $Fib(n+1) = Fib(n/2)Fib(n/2+1) + Fib(n/2-1)Fib(n/2)$. Calling our algorithm recursively on $n/2-1$ gives us $Fib(n/2-1)$ and $Fib(n/2)$. We can then add the two together to get $Fib(n/2+1)$ and use these in the above formulas to compute $Fib(n)$ and $Fib(n+1)$ with a constant number of additional arithmetic operations. If $n > 3$ is odd, we can apply the above to $n-1$, and then add $Fib(n-1)$ and $Fib(n)$ to get $Fib(n+1)$.

In all cases, we compute one sub-problem at most $n/2$ and then do a constant number of arithmetic operations. So assuming arithmetic is constant time, the recurrence is $T(n) = T(n/2) + O(1)$. Applying the Master Theorem with $a = 1, b = 2$ and $d = 0$, since $a = b^d$, we are in the steady-state case, and $T(n) \in O(\log n)$.

Review question for homework 4- 20 points Remember that the Fibonacci sequence is given by $F(0) = F(1) = 1, F(n) = F(n-1) + F(n-2)$ for $n \geq 2$. Consider the set of sequences of 0's and 1's of length n that do not have two consecutive 1's. Prove that the number of such sequences is $Fib(n+1)$.

When $n = 0$ there is one sequence (the empty sequence) and it contains no two consecutive 1's, so the number is $1 = Fib(0+1)$. When $n = 1$ there are two sequences $(0, 1\ )$ and neither contain two consecutive 1s, so the number is $2 = Fib(1+1)$.

Assume for $N \geq 1$ and all $0 \leq n \leq N$ that the number of such strings of length n is $Fib(n+1)$. Then we can break the strings of length $N+1$ with no consecutive 1's into two groups: those with first bit 0 and those with first bit 1. The remaining N bits in the first group can be any string of length N with no consecutive 1's, so by the induction hypothesis, there are $Fib(N+1)$ of them. The second group must all have second bit 0, and then be any string of length $N-1$ with no two consecutive 1's, so again by hte induction hypothesis, there are $Fib(N)$ of these. So the total number is $Fib(N+1) + Fib(N) = Fib(N+2)$, as desired. Thus, by strong induction on n, there are always $Fib(n+1)$ such strings of length n.

Inclusion-exclusion (Short answer, with brief explanation, 10 points) How many alphanumeric passwords of length 9 have at least one upper case letter, at least one lower case letter, and at least one numeral?

Let A be the set of strings with no upper case letters, B be the set of strings with no lower case letters, and C the set of strings with no numerals. The strings we are counting are all those that are not in $A \cup B \cup C$. Using inclusion-exclusion, $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$. There are $26 + 10$ possible characters for each position of a string in A or B, and 52 for a string in C, so $|A| = (36)^9 = |B|$ and $|C| = (52)^9$. Strings in $A \cap B$ have no lower case or upper case letters, so only numerals, so there are 10^9 of these. Similarly, $A \cap C$ strings have only lower case letters, and $B \cap C$ only upper case, so both sets have size $(26)^9$. There are no strings with no upper case letters, lower case letters, or numerals, so the intersection of all three sets is empty. Thus, $|A \cup B \cup C| = 2 * (36)^9 + (52)^9 - 2(26)^9 - (10)^9$. Finally, these are the ones we aren't counting out of $(62)^9$ total, so the number we are counting is Thus, $(62)^9 - 2 * (36)^9 - (52)^9 + 2(26)^9 + (10)^9$.

Coding bounded sequences A *bounded sequence* of length n is a sequence of integers so that the next element is either one greater than or less than the previous element.

Here are a few examples of *bounded sequences* of length 8, starting with1:

$(1,2,3,4,3,2,1,0)$, $(1,2,1,2,1,2,1,2)$, $(1,0,-1,-2,-3,-4,-3,-2)$

1. For $m, n \geq 1$, how many *bounded sequences* of length n are there that start with an integer between 1 and m? (give a brief justification.) (5 points)

   We have m possible values for the first element, and once we fix the previous one, two possible values for each consecutive element. So the total number is $m * 2^{n-1}$.

2. How many bits would the most efficient encoding of such sequences use? (in terms of n and m. Justify your work) (5 points)

   We thus need $\lceil log_2(m2^{n-1}) \rceil = \lceil \log m \rceil + n - 1$ bits to code such sequences.

3. Develop your own encoding/decoding algorithm where the code uses this number of bits. Describe your algorithms, and show the decoding recovers the encoded message. (10 points)

   Say the sequence is $a_1, a_2, ...a_n$. We use a fixed length code for the first element in the sequence, such as the binary representation of $a_1 - 1$ with leading 0's to make it fixed length. Let this code be $C(a_1)$. Then for each $1 < i \leq n$, we let b_i be 0 if $a_i = a_{i-1} - 1$ and 1 if $a_i = a_{i-1} + 1$. So our code for the sequence is : $C(a_1)b_2...b_n$. and has length exactly $\lceil \log_2 m \rceil + n - 1$.

   To decode, we let A_1 be the first $log(m)$ bits as a binary number, +1. Then for $i = 2$ to n, if $b_i = 1$, we let $A_i = A_{i-1} + 1$ and otherwise we let $A_i = A_{i-1} - 1$. Since $C(a_1)$ is the binary representation of $a_1 - 1$, $A_1 = a_1$. Assume $A_{i-1} = a_{i-1}$. If $a_i = a_{i-1} - 1$, $b_i = 0$, and $A_i = A_{i-1} - 1 = a_{i-1} - 1 = a_i$. If $a_i = a_{i-1} + 1$, $b_i = 1$, and $A_i = A_{i-1} + 1 = a_{i-1} + 1 = a_i$. So by induction on i, the decoded sequence A_i is equal to the original sequence a_i.

4. How many bounded sequences of length n have first and last element 0?

   Consider the part $b_2..b_n$ of the encoding above. To begin and end at 0, we must have an equal number of steps that increase by 1, and decrease by 1. So there must be $(n-1)/2$ 0's and $(n-1)/2$ 1's in the sequence. In particular, there are no such sequences when n is even. When n is odd, any suc bit string is the code of some sequence begining and ending at 0, so the number of such sequences is the number of strings of length $n-1$ with exactly $(n-1)/2$ 1's, $\binom{(n-1)}{(n-1)/2}$. (10 points)

Coding using ranking In review problem 4 above, it is shown that the number of binary sequences of length n with no consecutive pair of 1's is $F(n+1)$, the $n+1$'st fibonacci number. How many bits do we need to encode such a string ? (5 points).

$\lceil log_2 Fib(n+1) \rceil = \lceil \log_2(\Phi^n) \rceil = \lceil n \log_2 \Phi \rceil$, where Φ is the golden ratio.

Give a ranking based encoding algorithm to encode such strings with this number of bits (5 points).

We know that there are $Fib(n)$ strings in our set that start with 0 and $Fib(n-1)$ that start with 10. We can give each string its lexicographical ranking within this set as follows: $Rank(b_1..b_n)$

1. If $n == 0$ return 0.
2. IF $n == 1$ return b_1.
3. If $b_1 == 0$ THEN return $Rank(b_2..b_n)$.
4. ELSE return $Fib(n) + Rank(b_3..b_n)$.

Then give the corresponding deconding algorithm (5 points).

$UnRank(V, n) : 0 \leq V < Fib(n+1)$

1. If $n == 0$ return the empty string.
2. IF $n == 1$ return V.
3. If $V < Fib(n)$ THEN return $0 \circ UnRank(V, n-1)$.
4. ELSE return $10 \circ UnRank(V - Fib(n), n-2)$.

Show that the decoding algorithm applied to the coding algorithm is the original string (5 points).

We prove this by strong induction on n, i.e., prove that $Unrank(Rank(b_1..b_n), n) = b_1..b_n$ for any string with no two consecutive 1's.

If $n = 0$, $Rank$ of the empty string returns 0, and $UnRank$ with $n = 0$ returns the empty string, which is our input.

If $n = 1$, $Rank(b_1) = b_1$, and $Unrank(b_1, 1) = b_1$.

Assume the equation is true for some $N \geq 1$ and all $0 \leq n \leq N$ and let $b_1...b_{N+1}$ be an input.

If $b_1 = 0$, let $V = Rank(b_1..b_{N+1}) = Rank(b_2..b_{N+1}) < Fib(N+1)$. Then $Unrank(V, N+1) = 0 \circ Unrank(V, N) = 0 \circ Unrank(Rank(b_2..b_{N+1}), N) = 0b_2..b_{N+1} = b_1..b_{N+1}$ where the second to last step is the induction hypothesis.

If $b_1 = 1$, let $V = Rank(b_1..b_{N+1}) = Fib(N+1) + Rank(b_3..b_{N+1}) \geq Fib(N+1)$. Then $Unrank(V, N+1) = 10 \circ Unrank(V - Fib(N+1), N-1) = 10 \circ Unrank(Rank(b_3..b_{N+1}), N-1) = 10b_3..b_{N+1} = b_1b_2..b_{N+1}$ since we must have $b_2 = 0$ in this case.

So by induction on N , this is true for all lengths of input.

Modelling frisbee problem as graph For review problem 1 above, explain how we could model the situation with a dircted graph (possibly with parallel edges) and interpret the formula in the problem in terms of degrees of vertices in this graph. (10 points)

Consider a graph whose vertices are the players, and where each throw from i to j is a possibly parallel edge from i to j. Then in this graph $Throw_i$ is the out-degree of vertex i and $Catch_i$ is the in-degree of vertex i, and the previous equation is that the sum of the in-degrees equals the sum of the out-degrees. We know both are equal to the number of edges, so this is always true.

Modeling word puzzle as graph In a word morphing puzzle , you are given a list of words and want to find a way to move from the first word to the second word using only words that appear in the list and only changing one letter each step. Consider the following example:

$$WORD, LIST, CORD, CORE, WOOD, FOOD, WORE, FOOL, WOOL, FOOT, LOOT, LOST$$

. Describe or draw a graph that would help you solve this puzzle for this example, and say what a solution would mean in terms of the graph. (10 points)

We put a vertex for each word, and connect vertices if the words differ in exactly one letter. Any solution would represent a path from WORD to LIST in this graph, and vice versa. This would be an undirected graph, because words differing in one letter is a symmetric relation.

For this example, the adjacency list for the graph looks like:

- $WORD : CORD, WOOD, WORE$
- $LIST : LOST$
- $CORD : WORD, CORE$
- $CORE : CORD, WORE$
- $WOOD : WORD, FOOD, WOOL$
- $FOOD : WO0D, FOOL, FOOT$
- $WORE : CORE, WORD$
- $FOOL : WO0L, FOOT, FOOD$
- $WOOL : FO0L, WOOD$
- $FOOT : FOOL, FOOD, LOOT$
- $LOOT : FOOT$
- $LOST : LIST, LOOT$

One path is : WORD, WOOD, WOOL, FOOL, FOOT, LOOT, LOST, LIST