Midterm

Note

This is a closed-book exam. Each problem accounts for 10 points, unless otherwise marked.

Problems

1. Consider the following theorem regarding Gray codes, for which we have sketched a proof by induction in class.

   > There exists a Gray code of length $\lceil \log_2 k \rceil$ for any number $k \geq 2$ of objects. The Gray codes for the *even* values of k are *closed*, and the Gray codes for *odd* values of k are *open*.

   Please complete the proof (giving sufficient details); in particular, try to be precise about the length of a code in the proof. You should assume and use the following facts in your proof:

   (a) Given a closed Gray code for an even number k (≥ 2) of objects, we can construct a closed Gray code with one additional bit for $2k$ objects.

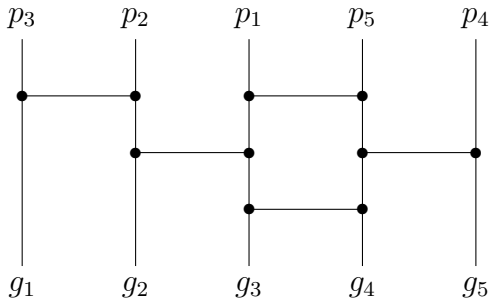
   (b) Given a closed Gray code of length i for 2^i ($i \geq 1$) objects, we can construct an open Gray code of the same length for any odd k, $2^{i-1} < k < 2^i$, of objects.

   (c) Given an open Gray code for an odd number k (≥ 2) of objects, we can construct a closed Gray code with one additional bit for $2k$ objects.

2. Consider the following two-player game: given a positive integer N, player A and player B take turns counting to N. In his turn, a player may advance the count by 1 or 2. For example, player A may start by saying "1, 2", player B follows by saying "3", player A follows by saying "4", etc. The player who eventually has to say the number N loses the game.

   A game is *determined* if one of the two players always has a way to win the game. Prove that the counting game as described is determined for any positive integer N; the winner may differ for different given integers. You must use induction in your proof. (Hint: think about the remainder of the number N divided by 3.)

3. We sometimes would use a diagram like the following to distribute n gifts (or assign n tasks) to n people. The main part of the diagram covered, each person (without seeing the horizontal line segments) is asked to choose one of the vertical lines. After everyone has made a choice, the whole diagram is revealed. Following the line chosen by p_i, go down along the line and, whenever hitting an intersection, must make a turn (to the left or right). The traced path will eventually reach a gift at the end and the gift is given to p_i.



Prove by induction that such a diagram (with arbitrary numbers of vertical and horizontal line segments) always produces a one-to-one mapping between the people and the gifts (whose number equals that of the vertical lines).

4. For each of the following pairs of functions, determine whether $f(n) = O(g(n))$ and/or $f(n) = \Omega(g(n))$. Justify your answers.

	$f(n)$	$g(n)$
(a)	$(\log n)^{\log n}$	$\frac{n}{\log n}$
(b)	$n^3 2^n$	3^n

5. Solve the following recurrence relation using *generating functions*. This is a very simple recurrence relation, but you must use generating functions in your solution.

$$\begin{cases} T(1) = 1 \\ T(2) = 2 \\ T(n) = 2T(n-1) - T(n-2), \quad n \geq 3 \end{cases}$$

6. If $f(x)$ is monotonically *decreasing*, then

$$\sum_{i=1}^{n} f(i) \leq f(1) + \int_1^n f(x)dx.$$

Show that this is indeed the case. (5 points)

7. Consider the Knapsack Problem: Given a set S of n items, where the i-th item has an integer size $S[i]$, and an integer K, find a subset of the items whose sizes sum to exactly K or determine that no such subset exists.

We have discussed in class two approaches to implementing a solution that we designed by induction: one uses dynamic programming (see the Appendix), while the other uses recursive function calls.

Suppose there are 5 items, with sizes $2, 3, 4, 5, 6$, and we are looking for a subset whose sizes sum to 13. Assuming recursive function calls are used, please give the two-dimension table P whose entries are filled with -, O, I, or left blank when the algorithm terminates. Which entries of $P[n, K]$ are visited/computed more than once?

8. Consider a variant of the Knapsack Problem where we want the subset to be as large as possible (i.e., to be with as many items as possible). How will you adapt the algorithm (see the Appendix) that we have studied in class? Your algorithm should collect at the end the items in one of the best solutions if they exist. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary (you may reuse the code for the original Knapsack Problem). Give an analysis of its time complexity. The more efficient your algorithm is, the more points you will get for this problem.

9. Let x_1, x_2, ..., x_n be a set of integers, and let $S = \sum_{i=1}^{n} x_i$. Design an algorithm to partition the set into two subsets of equal sum, or determine that it is impossible to do so. When the partitioning is possible, your algorithm should also give the two subsets of integers. The algorithm should run in time $O(nS)$.

10. In the **towers of Hanoi** puzzle, there are three pegs A, B, and C, with n (generalizing the original eight) disks of different sizes stacked in decreasing order on peg A. The objective is to transfer all the disks on peg A to peg B, moving one disk at a time (from one peg to one of the other two) and never having a larger disk stacked upon a smaller one.

    (a) Give an algorithm to solve the puzzle. Compute the total number of moves in the algorithm. (10 points)

    (b) If there is an additional fourth peg D, it is possible to reduce the number of moves. Please give a new algorithm that requires fewer moves. (5 points)

Appendix

- Below are several basic generating functions.

generating func.	power series	generated sequence
$\frac{1}{1-z}$	$1 + z + z^2 + \cdots + z^n + \cdots$	$1, 1, 1, \cdots, 1, \cdots$
$\frac{c}{1-az}$	$c + caz + ca^2z^2 + \cdots + ca^nz^n + \cdots$	$c, ca, ca^2, \cdots, ca^n, \cdots$
$\frac{1}{(1-z)^2}$	$1 + 2z + 3z^2 + \cdots + nz^{n-1} + \cdots$	$1, 2, 3, \cdots, n, \cdots$
$\frac{z}{(1-z)^2}$	$z + 2z^2 + 3z^3 + \cdots + nz^n + \cdots$	$0, 1, 2, 3, \cdots, n, \cdots$

- Below is an algorithm for determining whether a solution to the Knapsack Problem exists.

Algorithm Knapsack (S, K)**;**
begin
$P[0,0].exist := true$;
for $k := 1$ **to** K **do**
$P[0,k].exist := false$;
for $i := 1$ **to** n **do**
for $k := 0$ **to** K **do**
$P[i,k].exist := false$;
if $P[i-1,k].exist$ **then**
$P[i,k].exist := true$;
$P[i,k].belong := false$
else if $k - S[i] \geq 0$ **then**
if $P[i-1,k-S[i]].exist$ **then**
$P[i,k].exist := true$;
$P[i,k].belong := true$
end