

Midterm

Note

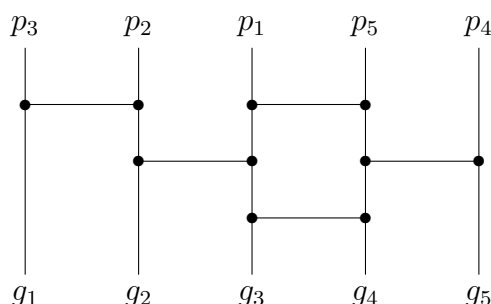
This is a closed-book exam. Each problem accounts for 10 points, unless otherwise marked.

Problems

1. Consider the following two-player *counting* game: given a positive integer N , player A and player B take turns counting to N . In her/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 *by induction* that the counting game as described is determined for any positive integer N ; the winner may differ for different given integers. (Hint: think about the remainder of the number N divided by 3.)

2. We sometimes would use a diagram like the following to distribute n gifts (or assign n tasks) to n people. With the main part of the diagram covered, each person p_i ($1 \leq i \leq n$), 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 vertical line chosen by p_i , go down along the line and, whenever hitting an intersection, make a turn and switch to a neighboring vertical line (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). Assume that the horizontal line segments do not intersect with one another.

3. In a homework problem, to determine whether $f(n) = O(g(n))$ and/or $f(n) = \Omega(g(n))$, for a given pair of monotonically growing functions f and g that map natural numbers to non-negative real numbers, you may have claimed and used the following:

If $f(n) = o(g(n))$, then $f(n) = O(g(n))$ and $f(n) \neq \Omega(g(n))$.

Prove that the claim is indeed true.

4. The Knapsack Problem that we discussed in class is defined as follows. 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 described in class an algorithm to solve the problem. Modify the algorithm to solve a variation of the Knapsack problem where the i -th item has additionally an associated value v_i . Find a way or determine it is impossible to pack the knapsack (of size K) fully, such that the items in it have the maximal total value among all possible ways to pack the knapsack. Give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem. (Hint: both the values of $P(i-1, k)$ and $P(i-1, k-S[i])$ should be considered.)

5. Show all intermediate and the final AVL trees formed by inserting the numbers 7, 6, 5, 2, 1, 3, and 4 (in this order) into an empty tree. Please use the following ordering convention: the key of an internal node is larger than that of its left child and smaller than that of its right child. If re-balancing operations are performed, please also show the tree before re-balancing and indicate what type of rotation is used in the re-balancing.
6. When analyzing the average-case time complexity of the Quicksort algorithm, we started with the following equation:

$$T(n) = n + 1 + T(i-1) + T(n-i), \text{ where } n \geq 2,$$

assuming the i -th smallest element of the input array is selected as the pivot.

What does the term $n + 1$ account for? Please explain why it is $n + 1$ (not any other value).

7. Consider rearranging the following array into a max heap using the *bottom-up* approach.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	2	8	4	1	15	7	6	3	11	10	12	13	14	9

Please show the result (i.e., the contents of the array) after a new element is added to the current collection of heaps (at the bottom) until the entire array has become a heap.

8. Draw a decision tree of the Heapsort algorithm (in increasing order) for the case of $A[1..3]$, i.e., $n = 3$. In the decision tree, you must indicate (1) which two elements of the original input array are compared in each internal node and (2) the sorting result in each leaf. Please use X_1 , X_2 , X_3 (not $A[1]$, $A[2]$, $A[3]$) to refer to the elements (in this order) of the original input array A .

9. Design an algorithm that, given a set of integers $S = \{x_1, x_2, \dots, x_n\}$, finds a nonempty subset $R \subseteq S$, such that

$$\sum_{x_i \in R} x_i \equiv 0 \pmod{n}.$$

Please present your algorithm in adequate pseudocode and make assumptions wherever necessary. Give also an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

Before presenting your algorithm, please argue why such a nonempty subset must exist. (Hint: think about the sums $x_1, x_1 + x_2, x_1 + x_2 + x_3, \dots, x_1 + x_2 + \dots + x_{n-1}$, and $x_1 + x_2 + \dots + x_{n-1} + x_n$; the difference between any two of these is also a sum.)

10. Consider the *next* table as in the KMP algorithm (the version presented in class) for string $B[1..9] = abaababaa$.

1	2	3	4	5	6	7	8	9
<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>
-1	0	0	1	1	2	3	2	3

Suppose that, during an execution of the KMP algorithm, $B[6]$ (which is an *a*) is being compared with a letter in A , say $A[i]$, which is not an *a* and so the matching fails. The algorithm will next try to compare $B[\text{next}[6] + 1]$, i.e., $B[3]$ which is also an *a*, with $A[i]$. The matching is bound to fail for the same reason. This comparison could have been avoided, as we know from B itself that $B[6]$ equals $B[3]$ and, if $B[6]$ does not match $A[i]$, then $B[3]$ certainly will not, either. $B[5]$, $B[8]$, and $B[9]$ all have the same problem, but $B[7]$ does not.

Please adapt the computation of the *next* table so that such wasted comparisons can be avoided. Also, please give, for a new string $B[1..9] = bbbabbbbaa$, the values of the original *next* table and those of the new *next* table according to the adaptation.

Appendix

- The notions of O , Ω , and o are defined as follows.
 - A function $f(n)$ is $O(g(n))$ for another function $g(n)$ if there exist constants c and N such that, for all $n \geq N$, $f(n) \leq cg(n)$.
 - A function $f(n)$ is $\Omega(g(n))$ if there exist constants c and N such that, for all $n \geq N$, $f(n) \geq cg(n)$.
 - A function $f(n)$ is $o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$.
- Below is the algorithm discussed in class for determining whether a solution to the (original) 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
 end

- Below is the Partition procedure we studied for the Quicksort algorithm:

Algorithm Partition($X, Left, Right$);
begin
 $pivot := X[Left]$;
 $L := Left + 1$; $R := Right$;
 while $L \leq R$ **do**
 while $L \leq Right$ and $X[L] \leq pivot$ **do** $L := L + 1$;
 while $R \geq Left$ and $X[R] > pivot$ **do** $R := R - 1$;
 if $L < R$ **then**
 $swap(X[L], X[R])$;
 $L := L + 1$;
 $R := R - 1$;
 $Middle := R$;
 $swap(X[Left], X[Middle])$
 end

- The algorithm for computing the *next* table in the KMP algorithm:

Algorithm Compute_Next (B, m);
begin
 $next[1] := -1$; $next[2] := 0$;
 for $i := 3$ **to** m **do**
 $j := next[i-1] + 1$;
 while $B[i-1] \neq B[j]$ and $j > 0$ **do**
 $j := next[j] + 1$;
 $next[i] := j$
 end