MA/CSSE 473 – Design and Analysis of Algorithms

Homework 10 (70 points total) Updated for Summer, 2014

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

AS usual, how many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

6.1.1 [6.1.2]	(closest numbers in an array with pre-sorting)
6.1.2 [6.1.3]	(intersection with pre-sorting)
6.1.8 [6.1.10]	(open intervals common point)
6.1.11	(anagram detection)
6.2.8ab	(Gauss-Jordan elimination)
6.3.9	(Range of numbers in a 2-3 tree)
6.5.3	(efficiency of Horner's rule)
6.5.4	(example of Horner's rule and synthetic division)
7.1.7	(virtual initialization)

Problems to write up and turn in:

1. (10) 6.1.5 [6.1.7] (to sort or not to sort)
2. (10) 6.2.8c (compare Gaussian Elimination to Gauss-Jordan) **You should compute and compare actual number of multiplications, not just say that both are Θ(n^3). Use division when you compare.**
3. (6) 6.3.7 (2-3 tree construction and efficiency)
4. (20) Not in book (sum of heights of nodes in a full tree) In this problem, we consider completely full binary trees with N nodes and height H (so that $N = 2^{H+1} - 1$)

   (a) (5 points) Show that the sum of the heights of all of the nodes of such a tree can be expressed as $\sum_{k=0}^{H} k\,2^{H-k}$.

   (b) (10 points) Prove by induction on H that the above sum of the heights of the nodes is N - H - 1. You may base your proof on the summation from part (a) (so you don't need to refer to trees at all), **or** you may do a "standard" binary tree induction based on the heights of the trees, using the definition that a non-empty binary tree has a root plus left and right subtrees. I find the tree approach more straightforward, but you may use the summation if you prefer.

   (c) (3 points) What is the big Θ estimate for the sum of the *depths* of all of the nodes in such a tree?

   (d) (2 points) How does the result of parts (b) and (c) apply to Heapsort analysis?

   Example of height and depth sums: Consider a full tree with height 2 (7 nodes).
   Heights: root:2, leaves: 0. Sum of all heights: 1*2 + 2*1 + 4*0 = 3.
   Depths: root: 0, leaves: 2. Sum of all depths: 1*0 + 2*1 + 4*2 = 10.
5. (10) 6.4.12 [6.4.11] (spaghetti sort)
6. (4) 6.5.10 [6.5.9] (Use Horner's rule for this particular case?)
7. (10) 7.1.6 (ancestry problem). You may **NOT** assume any of the following:

   - The tree is binary
   - The tree is a search tree (i.e. that the elements are in some particular order)
   - The tree is balanced in any way.

The tree for this problem is simply a connected directed graph with no cycles and a single source node (the root).

MA/CSSE 473 HW 10 textbook problems and hints

1. Problem 6.1.5 [6.1.7] (10)

7. To sort or not to sort? Design a reasonably efficient algorithm for solving each of the following problems and determine its efficiency class.

a. You are given n telephone bills and m checks sent to pay the bills ($n \geq m$). Assuming that telephone numbers are written on the checks, find out who failed to pay. (For simplicity, you may also assume that only one check is written for a particular bill and that it covers the bill in full.)

b. You have a file of n student records indicating each student's number, name, home address, and date of birth. Find out the number of students from each of the 50 U.S. states.

Author's hint:

a. The problem is similar to one of the preceding problems in these exercises.

b. How would you solve this problem if the student information were written on index cards? Better yet, think how somebody else, who has never taken a course on algorithms but possesses a good dose of common sense, would solve this problem.

Problem 2: 6.28c (10)

The ***Gauss-Jordan elimination*** method differs from Gaussian elimination in that the elements above the main diagonal of the coefficient matrix are made zero at the same time and by the same use of a pivot row as the elements below the main diagonal.

a. Apply the Gauss-Jordan method to the system of Problem 1 of these exercises.

b. What general design technique is this algorithm based on?

c.▷ In general, how many multiplications are made by this method while solving a system of n equations in n unknowns? How does this compare with the number of multiplications made by the Gaussian elimination method in both its elimination and its back-substitution stages?

Students are only required to do part c. I put the rest here for context. You should compute and compare actual number of multiplications, not just say that both are Θ(n^3). Use division when you compare.

Author's hint:

8. a. Manipulate the matrix rows above a pivot row the same way the rows below the pivot row are changed.

   b. Are the Gauss-Jordan method and Gaussian elimination based on the same algorithm design technique or on different ones?

   c. Derive the formula for the number of multiplications in the Gauss-Jordan method the same way it was done for Gaussian elimination in Section 6.2.

Problem 3 6.3.7 (6)

7. a. Construct a 2-3 tree for the list C, O, M, P, U, T, I, N, G. (Use the alphabetical order of the letters and insert them successively starting with the empty tree.)

   b. Assuming that the probabilities of searching for each of the keys (i.e., the letters) are the same, find the largest number and the average number of key comparisons for successful searches in this tree.

Author's hint:

7. a. Trace the algorithm for the input given (see Figure 6.8) for an example.

   b. Keep in mind that the number of key comparisons made in searching for a key in a 2-3 tree depends not only on its node's depth but also whether the key is the first or second one in the node.

Problem 4 (20) not from textbook

(sum of heights of nodes in a full tree) In this problem, we consider completely full binary trees with N nodes and height H (so that $N = 2^{H+1} - 1$)

(a) (5 points) Show that the sum of the heights of all of the nodes of such a tree can be expressed as $\sum_{k=0}^{H} k 2^{H-k}$.

(b) (10 points) Prove by induction on H that the above sum of the heights of the nodes is N - H - 1. You may base your proof on the summation from part (a) (so you don't need to refer to trees at all), **or** you may do a "standard" binary tree induction based on the heights of the trees, using the definition that a non-empty binary tree has a root plus left and right subtrees. I find the tree approach more straightforward, but you may use the summation if you prefer.

(c) (3 points) What is the big Θ time for the sum of the *depths* of all of the nodes in such a tree?

(d) (2 points) How does the result of parts (b) and (c) apply to Heapsort analysis?

Example of height and depth sums: Consider a full tree with height 2 (7 nodes).
Heights: root:2, leaves: 0. Sum of all heights: 1*2 + 2*1 + 4*0 = 3.
Depths: root: 0, leaves: 2. Sum of all depths: 1*0 + 2*1 + 4*2 = 10.

Problem 5 6.4.12 [6.4.11] (10)

11. *Spaghetti sort* Imagine a handful of uncooked spaghetti, individual rods whose lengths represent numbers that need to be sorted.

    a. Outline a "spaghetti sort"—a sorting algorithm that takes advantage of this unorthodox representation.

    b. What does this example of computer science folklore (see [Dew93]) have to do with the topic of this chapter in general and heapsort in particular?

Author's hint:

11. Pick the spaghetti rods up in a bundle and place them end-down (i.e., vertically) onto a tabletop.

Problem 6 6.5.10 [6.5.9] (4)

9. Is it a good idea to use a general-purpose polynomial evaluation algorithm such as Horner's rule to evaluate the polynomial $p(x) = x^n + x^{n-1} + ... + x + 1$?

Author's hint:

9. Use a formula for the sum of the terms of this special kind of a polynomial.

Problem 7 7.1.6 (10)

6. $\triangleright$ The ***ancestry problem*** asks to determine whether a vertex u is an ancestor of vertex v in a given binary (or, more generally, rooted ordered) tree of n vertices. Design a $O(n)$ input enhancement algorithm that provides sufficient information to solve this problem for any pair of the tree's vertices in constant time.

Author's hint:

Take advantage of the standard traversals of such trees.

Instructor notes

7. (10) 7.1.6 (ancestry problem). You may **NOT** assume any of the following:

- The tree is binary
- The tree is a search tree (i.e. that the elements are in some particular order)
- The tree is balanced in any way.

The tree is simply a connected directed graph with no cycles and a single source node (the root)

MA/CSSE 473 – Design and Analysis of Algorithms

Homework 11 (72 points total) Updated for Summer, 2014

This is probably the longest assignment of the course, and it has some difficult problems. Start early!

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

7.2.2 [7.2.2]	(Horspool for patterns in DNA)
7.2.5 [7.2.5]	(is there a case where Horspool does more comparisons than brute force?)
7.2.9 [7.2.9]	(left-to-right checking OK after a single character match in Horspool, Boyer-Moore?)
7.3.1 [7.3.1]	(insert specific keys into hash table with specific hash function and separate chaining)
8.1.1 [8.1.1]	(Compare and contrast dynamic programming with divide-and-conquer)
8.1.4 [8.1.9]	(Space efficiency of dynamic programming for Binomial coefficients)

Problems to write up and turn in:

1. (6) 7.2.3 [7.2.3] (Horspool for binary strings)

2. (9) 7.2.7 [7.2.7] (Boyer-Moore for binary strings)

3. (4) 7.2.8 [7.2.8] (does Boyer-Moore still work with just one table?)

4. (8) 7.2.11 [not in 2nd ed] (right cyclic shift) 3 points for part a, 5 for part b.

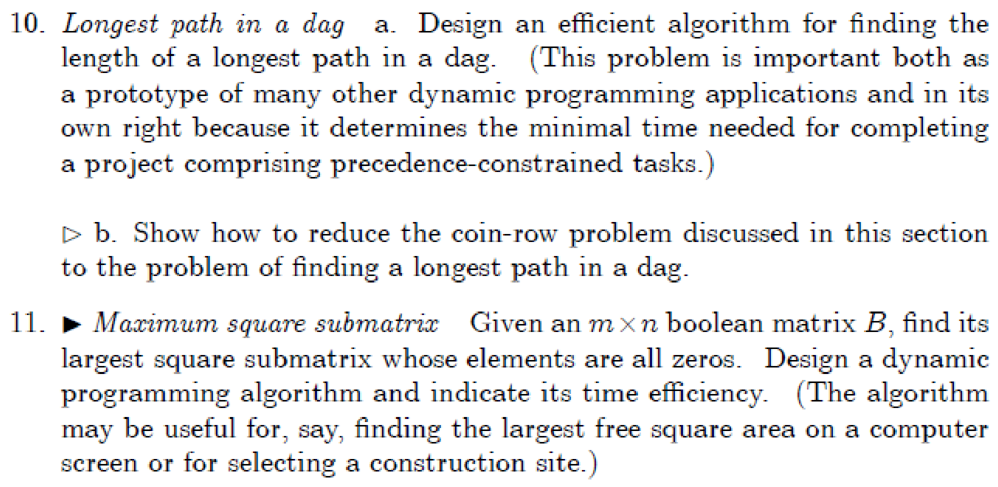
   You are given two strings S and T, each n characters long. You have to establish whether one of them is a right cyclic shift of the other. For example, PLEA is a right cyclic shift of LEAP, and vice versa. (Formally, T is a right cyclic shift of S if T can be obtained by concatenating the (n - i)-character suffix of S and the i-character prefix of s for some $1 \leq i \leq n$).
   a. Design a space-efficient algorithm for the task. Indicate the space and time efficiencies of your algorithm.
   b. Design a time-efficient algorithm for the task. Indicate the time and space efficiencies of your algorithm.

5. (5) 7.3.4 [7.3.4] (probability that n keys all hash to the same table location)

6. (6) 7.4.3[7.4.3] (minimum order of a B tree with no more than 3 disk accesses in a tree with 10^8 elements)

The list of problems continues on next page

7. (12) 8.1.10 [not in 2nd ed] longest path in a DAG. Note that the material from Section 8.1 of the third edition of Levitin is not in the second edition. I have posted that section on ANGEL in the Reading Materials folder.



10. *Longest path in a dag* a. Design an efficient algorithm for finding the length of a longest path in a dag. (This problem is important both as a prototype of many other dynamic programming applications and in its own right because it determines the minimal time needed for completing a project comprising precedence-constrained tasks.)

    ▷ b. Show how to reduce the coin-row problem discussed in this section to the problem of finding a longest path in a dag.

11. ▶ *Maximum square submatrix* Given an $m \times n$ boolean matrix B, find its largest square submatrix whose elements are all zeros. Design a dynamic programming algorithm and indicate its time efficiency. (The algorithm may be useful for, say, finding the largest free square area on a computer screen or for selecting a construction site.)

8. (12) 8.1.11 [not in 2nd ed] Maximum square submatrix. See description above

9. (10) 8.1.12 [8.1.10] (World Series odds) Note: In a 7-game series (such as the real American baseball World Series), the first team to win 4 games wins the series. 7 is the maximum number of games that can be played before one of the teams must win four games. But if one team wins 4 games sooner, the series ends immediately.

HW 11 textbook problems and hints

Problem 1 7.2.3 (6) (Horspool for binary strings)

3. How many character comparisons will be made by Horspool's algorithm in searching for each of the following patterns in the binary text of 1000 zeros?

   a. 00001

   b. 10000

   c. 01010

Author's hint:

3. For each pattern, fill in its shift table and then determine the number of character comparisons (both successful and unsuccessful) on each trial and the total number of trials.

Problem 2 7.2.7 (9) (Boyer-Moore for binary strings)

7. How many character comparisons will the Boyer-Moore algorithm make in searching for each of the following patterns in the binary text of 1000 zeros?

   a. 00001

   b. 10000

   c. 01010

Author's hint:

7. For each pattern, fill in the two shift tables and then determine the number of character comparisons (both successful and unsuccessful) on each trial and the total number of trials.

Problem 3: 7.2.8 (4) (does Boyer-Moore still work with just one table?)

8. a. Would the Boyer-Moore algorithm work correctly with just the bad-symbol table to guide pattern shifts?

   b. Would the Boyer-Moore algorithm work correctly with just the good-suffix table to guide pattern shifts?

Author's hint:

8. Check the description of the Boyer-Moore algorithm.

Problem 4: 7.2.11 (3, 5) (right cyclic shift)

11. You are given two strings S and T, each n characters long. You have to establish whether one of them is a right cyclic shift of the other. For example, PLEA is a right cyclic rotation of LEAP, and vice versa. (Formally, T is a right cyclic shift of S if T can be obtained by concatenating the $(n-i)$-character suffix of S and the i-character prefix of S for some $1 \le i \le n$.)

    a. Design a space-efficient algorithm for the task. Indicate the space and time efficiencies of your algorithm.

    b. Design a time-efficient algorithm for the task. Indicate the time and space efficiencies of your algorithm.

Author's hint:

11. a. A brute-force algorithm fits the bill here.

    b. Enhance the input before a search.

Problem 5: 7.3.4 (5) (probability that n keys all hash to the same table location)

4. Find the probability of all n keys being hashed to the same cell of a hash table of size m if the hash function distributes keys evenly among all the cells of the table.

Author's hint:

4. The question is quite similar to computing the probability of having the same result in n throws of a fair die.

Problem 6: 7.4.3 (6)

(minimum order of a B tree that guarantees no more than 3 disk accesses in a tree with 10^8 elements)

3. Find the minimum order of the B-tree that guarantees that the number of disk accesses in searching in a file of 100 million records does not exceed 3. Assume that the root's page is stored in main memory.

Author's hint:

3. Find this value from the inequality in the text that provides the upper-bound of the B-tree's height.

Problems 7-8 : 8.1.10, 8.1.11 (12 each) (Longest path in a dag, Maximum square submatrix)

Note that the material from Section 8.1 of the 3rd edition of Levitin is not in the 2nd edition. I have posted that section on ANGEL in the Reading Materials folder.

10. *Longest path in a dag* a. Design an efficient algorithm for finding the length of a longest path in a dag. (This problem is important both as a prototype of many other dynamic programming applications and in its own right because it determines the minimal time needed for completing a project comprising precedence-constrained tasks.)

    ▷ b. Show how to reduce the coin-row problem discussed in this section to the problem of finding a longest path in a dag.

11. ▶ *Maximum square submatrix* Given an $m \times n$ boolean matrix B, find its largest square submatrix whose elements are all zeros. Design a dynamic programming algorithm and indicate its time efficiency. (The algorithm may be useful for, say, finding the largest free square area on a computer screen or for selecting a construction site.)

Author's hint:

10. a. Topologically sort the dag's vertices first.

    b. Create a dag with $n + 1$ vertices: one vertex to start and the others to represent the coins given.

11. Let $F(i, j)$ be the order of the largest all-zero submatrix of a given matrix with its low right corner at (i, j) Set up a recurrence relating $F(i, j)$ to $F(i-1, j)$, $F(i, j-1)$, and $F(i-1, j-1)$.

Problem 9: 8.1.12 (10) (World Series odds) **Note from Claude:** In a 7-game series (such as the real American baseball World Series), the first team to win 4 games wins the series. 7 is the maximum number of games that can be played before one of the teams must win four games. But if one team wins 4 games sooner, the series ends immediately.

10. ▷ *World Series odds* Consider two teams, A and B, playing a series of games until one of the teams wins n games. Assume that the probability of A winning a game is the same for each game and equal to p and the probability of A losing a game is $q = 1 - p$. (Hence, there are no ties.) Let $P(i, j)$ be the probability of A winning the series if A needs i more games to win the series and B needs j more games to win the series.

    a. Set up a recurrence relation for $P(i, j)$ that can be used by a dynamic programming algorithm.

    b. Find the probability of team A winning a seven-game series if the probability of it winning a game is 0.4.

    c. Write a pseudocode of the dynamic programming algorithm for solving this problem and determine its time and space efficiencies.

Author's hint:

10. a. In the situation where teams A and B need i and j games, respectively, to win the series, consider the result of team A winning the game and the result of team A losing the game.

    b. Set up a table with five rows $(0 \le i \le 4)$ and five columns $(0 \le j \le 4)$ and fill it by using the recurrence derived in part (a).

    c. A pseudocode should be guided by the recurrence set up in part (a). The efficiency answers follow immediately from the table's size and the time spent on computing each of its entries.

Instructor note:

In a 7-game series (such as the real American baseball World Series), the first team to win 4 games wins the series. 7 is the maximum number of games that can be played before one of the teams must win four games. But if one team wins 4 games sooner, the series ends immediately.

MA/CSSE 473 – Design and Analysis of Algorithms

Homework 12 (88 points total) Updated for Summer, 2014

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

8.4.2 [8.2.2]	(Time efficiency of Warshall's Algorithm)
8.4.6 [8.2.6]	(Use Warshall to determine whether a digraph is a dag)
8.3.1	(Practice optimal BST calculation)
8.3.2	(Time and space efficiency of optimal BST calculation)
8.3.5	(Root of Optimal tree)
8.3.9	(Include unsuccessful searches in optimal BST calculation)
8.3.8	(n^2 algorithm for optimalBST. Not for the faint of heart!)
------	For the frequencies of the Day 33 class example (AEIOU), find the optimal tree if we consider only successful searches (set all qi to 0)
------	For the frequencies of the Day 33 class example (AEIOU), find the optimal tree if we consider only unsuccessful searches (set all pi to 0)

Problems to write up and turn in: (I have posted Levitin 3rd edition section 8.4 on Moodle)

1. (5) 8.4.3 [8.2.3] (Warshall with no extra memory use)
2. (10) 8.4.4 [8.2.4] (More efficient Warshall inner loop)
3. (25) Optimal BST problem: described below.
   Part (d) is extra credit. Not many people have gotten it in past terms.
   In the past, a number of students have said that this problem is long and difficult, especially part a. I have placed on Moodle an excerpt form the original source from which I got this example. Also note that there is some relevant Python code linked from the schedule page (days 31 and 32 in Summer 2014).
4. (10) 8.3.3 (Optimal BST from root table)
5. (5) 8.3.4 (Sum for optimalBST in constant time).
6. (10) 8.3.6 (optimalBST--successful search only--if all probabilities equal)
7. (5) 8.3.11a [8.3.10a] (Matrix chain multiplication) Also think about parts (b) and (c), which may appear on a later assignment or exam.
8. (10) 8.4.7 [8.2.7] (Floyd example)
9. (8) 8.4.8 [8.2.8] (Floyd: Do we need to save previous matrix?)

Optimal BST Dynamic Programming Problem (problem #3)

In a binary search tree, the key of each node in the left subtree is smaller than the key of the root, and the key of each node in the right subtree is larger than the root. The same property holds for each subtree. Section 8.3 discusses a dynamic programming algorithm to find an optimal static tree if only successful searches are taken into account. In class (Days 31-32 in Fall, 2012) we discussed a modified algorithm that also takes unsuccessful searches into account.

Suppose that we have a static set of N keys $K_1, K_2, \ldots, K_n$ (in increasing order), and that we have collected statistics about the frequency of searches for each key and for items in each "gap" between keys (i.e. each place that an unsuccessful search can end up).

For i = 1 ... n, let a_i be the frequency of searches that end successfully at K_i.

For i = 1 ... n-1, let b_i be the frequency of unsuccessful searches for all "missing keys" that are between K_i and K_{i+1} (also, b_0 is the frequency of searches for keys smaller than K_1, and b_n is the frequency for keys that are larger than K_n).

We build an extended BST T (see Figure 4.5 for a diagram of an extended tree) whose internal nodes contain the N keys, $K_1, \ldots, K_n$. Let x_i be the depth of the node containing K_i, and let y_i be the depth of the "external node" that represents the gap between K_i and K_{i+1} (where y_0 and y_n are the depths of the leftmost and rightmost external nodes of T). Recall that the depth of a tree's root is 0. The optimal tree for the given keys and search frequencies is one that minimizes the *weighted path length* C(T), where C is defined by

$$C = \sum_{i=1}^{N} a_i[1 + x_i] + \sum_{i=0}^{N} b_i y_i$$

For example, in class we considered the following data:

i	K_i	a_i	b_i
0			0
1	A	32	34
2	E	42	38
3	I	26	58
4	O	32	95
5	U	12	21

If we choose to build the BST with I as the root, E and O as I's children, A as E's child, and U as O's child, then C = 948, and the average search length is 948/390 = 2.43 (you should verify this, to check your understanding of the formula for C). It turns out that this tree is not optimal.

In class we discussed a dynamic programming algorithm that finds a tree that minimizes C for any set of n keys and 2n+1 associated frequencies. It uses an alternate, recursive, formulation of C:

(a) (10) **The recursive formulation:** Let T be a BST. If T is empty, then C(T) = 0. Otherwise T has a root and two subtrees T_L and T_R. Then $C(T) = C(T_L) + C(T_R) + \text{sum}(a_i, i=1..n) + \text{sum}(b_i, i=0..n)$. Show by induction that this recursive definition of C(T) is equivalent to the summation definition given above. [The recursive definition is used in the code provided online (linked from the schedule page].

(b) (5) The algorithm and a Python implementation are provided, along with a table of final values for the above inputs. Use the information from that table to draw the optimal tree.

(c) (10) What is the big-theta running time for the optimal-tree-generating algorithm? Show your computations that lead you to this conclusion

10. (d) (10) **(Part (d) is extra credit. Not many people have gotten it in past terms)** (This is a challenging problem) Find a way to improve the given algorithm so that it has a smaller big-theta running time, and show that it really is smaller.

HW 12 textbook problems and hints

Problem 1. 8.4.3 [8.2.3] (5) (Warshall with no extra memory use)

Explain how to implement Warshall's algorithm without using extra memory
for storing elements of the algorithm's intermediate matrices.

Author's hints:

Show that we can simply overwrite elements of $R^{(k-1)}$ with elements of $R^{(k)}$ without any other changes in the algorithm.

Problem 2. 8.4.4[8.2.4] (10) (More efficient Warshall inner loop)

3. Explain how to implement Warshall's algorithm without using extra memory for storing elements of the algorithm's intermediate matrices.

4. Explain how to restructure the innermost loop of the algorithm *Warshall* to make it run faster at least on some inputs.

Author's hints:

3. Show that we can simply overwrite elements of $R^{(k-1)}$ with elements of $R^{(k)}$ without any other changes in the algorithm.

4. What happens if $R^{(k-1)}[i, k] = 0$?

Problem 3. OptimalBST problem (25 points plus extra credit)

Not from the textbook. Description is in the assignment document

Problem 4. 8.3.3 (10) (Optimal BST from root table)

3. Write a pseudocode for a linear-time algorithm that generates the optimal binary search tree from the root table.

Author's hint:

3. $k = R[1, n]$ indicates that the root of an optimal tree is the kth key in the list of ordered keys $a_1, ..., a_n$. The roots of its left and right subtrees are specified by $R[1, k-1]$ and $R[k+1, n]$, respectively.

Problem 5. 8.3.4 (5) (Sum for optimalBST in constant time)

4. Devise a way to compute the sums $\sum_{s=i}^{j} p_s$, which are used in the dynamic programming algorithm for constructing an optimal binary search tree, in constant time (per sum).

Author's hint

4. Use a space-for-time tradeoff.

:

Problem 6. 8.3.6 (10) (optimalBST--successful search only--if all probabilities equal)

6. How would you construct an optimal binary search tree for a set of n keys if all the keys are equally likely to be searched for? What will be the average number of comparisons in a successful search in such a tree if $n = 2^k$?

Author's hint:

6. The structure of the tree should simply minimize the average depth of its nodes. Do not forget to indicate a way to distribute the keys among the nodes of the tree.

Problem 7. 8.3.10a (5) (Matrix chain multiplication)

10. *Matrix chain multiplication* Consider the problem of minimizing the total number of multiplications made in computing the product of n matrices

$$A_1 \cdot A_2 \cdot \ldots \cdot A_n$$

whose dimensions are d_0 by d_1, d_1 by $d_2, \ldots, d_{n-1}$ by d_n, respectively. (Assume that all intermediate products of two matrices are computed by the

brute-force (definition-based) algorithm.

a. Give an example of three matrices for which the number of multiplications in $(A_1 \cdot A_2) \cdot A_3$ and $A_1 \cdot (A_2 \cdot A_3)$ differ at least by a factor 1000.

Author's hint:

10. a. It is easier to find a general formula for the number of multiplications needed for computing $(A_1 \cdot A_2) \cdot A_3$ and $A_1 \cdot (A_2 \cdot A_3)$ for matrices A_1 with dimensions d_0-by-d_1, A_2 with dimensions d_1-by-d_2, and A_3 with dimensions d_2-by-d_3 and then choose some specific values for the dimensions to get a required example.

Problem 8. 8. 4.7 [8.2.7] (10) (Floyd algorithm example)

7. Solve the all-pairs shortest path problem for the digraph with the following weight matrix:

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Author's hint:

See an example of applying the algorithm to another instance of the problem in the section.

Problem 9. 8. 4.8 [8.2.8] (8) (Floyd algorithm matrix overwrite)

Prove that the next matrix in sequence (8.14) of Floyd's algorithm can be written over its predecessor.

Instructor's notes:

Here is the sequence that the problem refers to (I think it is 8.8 in the second edition):

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \quad \text{for } k \geq 1, \quad d_{ij}^{(0)} = w_{ij}. \qquad \textbf{(8.14)}$$

Author's hint:

What elements of matrix $D^{(k-1)}$ does $d_{ij}^{(k)}$, the element in the ith row and the jth column of matrix $D^{(k)}$, depend on? Can these values be changed by the overwriting?

MA/CSSE 473 – Design and Analysis of Algorithms

Homework 13 (70 points total) Updated for Summer, 2014

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

Not in 3rd ed [9.1.1]	(Greedy change-making not optimal) Give an instance of the change-making problem for which the greedy algorithm does not yield an optimal solution
9.1.5	(greedy bridge crossing)

Problems to write up and turn in:

1. (10) 9.1.3 (Greedy job scheduling)
2. (6) 9.1.9b [9.1.7b] (Prim example) Start with node a. Whenever you have a choice because edge weights are equal, choose the vertex that is closest to the beginning of the alphabet. Then everyone should get the same answer, making it easier for us to check your work.
   |
3. (5) 9.1.10 [9.1.8] (Prim prior connectivity check?)
4. (10) 9.1.15 [9.1.11] (change value of an item in a min-heap)
5. (6) 9.2.1b (Kruskal example) Whenever you have a choice because edge weights are equal, choose the edge whose vertices are closest to the beginning of the alphabet. Then everyone should get the same answer, making it easier for us to check your work.
6. (8) 9.2.2 (Kruskal TF questions) Briefly explain your answers.
7. (5) 9.2.8 (efficiency of *find* in union-by-size)
8. (8) 9.4.1 (Huffman codes) (a) 4 points. When there is a choice due to a tie, place the one that appears first in the problem statement's character list "on the left" in the tree. (b) 2 points. (c) 2 points.
9. (12) 9.4.3 (Huffman TF) (a) 5 points (b) 7 points Explain your answers.

MA/CSSE 473 HW 13 textbook problems and hints

Problem #1 (10) 9.1.3 (Greedy job scheduling)

3. Consider the problem of scheduling n jobs of known durations $t_1, ..., t_n$ for execution by a single processor. The jobs can be executed in any order, one job at a time. You want to find a schedule that minimizes the total time spent by all the jobs in the system. (The time spent by one job in the system is the sum of the time spent by this job in waiting plus the time spent on its execution.)

   Design a greedy algorithm for this problem. ▷ Does the greedy algorithm always yield an optimal solution?

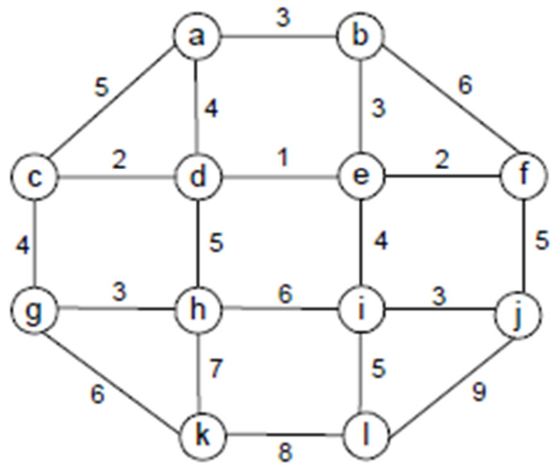
Author's hint:

3. Considering the case of two jobs might help. Of course, after forming a hypothesis, you will have to either prove the algorithm's optimality for an arbitrary input or find a specific counterexample showing that it is not the case.

Problem #2 (6) 9.1.9b [9.1.7]b (Prim example)

(Prim example) Start with node a. Whenever you have a choice because edge weights are equal, choose the vertex that is closest to the beginning of the alphabet. Then everyone should get the same answer, making it easier for us to check your work.

b. Apply Prim's algorithm to the following graph. Include in the priority queue only the fringe vertices (the vertices not in the current tree which are adjacent to at least one tree vertex).



Author's hint:

b. After the next fringe vertex is added to the tree, add all the unseen vertices adjacent to it to the priority queue of fringe vertices.

Problem #3 (5) 9.1.10 [9.1.8] (Prim prior connectivity check?)

8. The notion of a minimum spanning tree is applicable to a connected weighted graph. Do we have to check a graph's connectivity before applying Prim's algorithm or can the algorithm do it by itself?

Author's hint:

8. Applying Prim's algorithm to a weighted graph that is not connected should help in answering this question.

Problem #4 (10)) 9.1.15 [9.1.11] (change value of an item in a min-heap)

11. Outline an efficient algorithm for changing an element's value in a min-heap. What is the time efficiency of your algorithm?

Author's hint

11. Consider two cases: the key's value was decreased (this is the case needed for Prim's algorithm) and the key's value was increased.

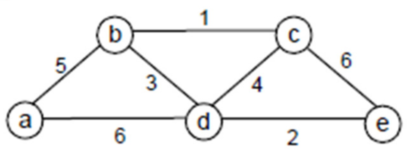
:

Problem #5 (6) 9.2.1b (Krushkal example)

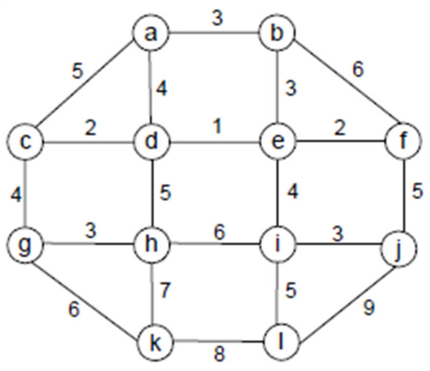
(Krushkal example) Whenever you have a choice because edge weights are equal, choose the edge whose vertices are closest to the beginning of the alphabet. Then everyone should get the same answer, making it easier for us to check your work.

1. Apply Kruskal's algorithm to find a minimum spanning tree of the following graphs.

a.



b.



Author's hint:

1. Trace the algorithm for the given graphs the same way it is done for another input in the section.

Problem #6 (8) 9.2.2 (Kruskal TF questions) Briefly explain your answers.

2. Indicate whether the following statements are true or false:

   a. If e is a minimum-weight edge in a connected weighted graph, it must be among edges of at least one minimum spanning tree of the graph.

   b. If e is a minimum-weight edge in a connected weighted graph, it must be among edges of each minimum spanning tree of the graph.

   c. If edge weights of a connected weighted graph are all distinct, the graph must have exactly one minimum spanning tree.

   d. If edge weights of a connected weighted graph are not all distinct, the graph must have more than one minimum spanning tree.

Author's hint:

2. Two of the four assertions are true, the other two are false.

Problem #7 (5) 9.2.8 (efficiency of find in union-by-size)

8. Prove that the time efficiency of $find(x)$ is in $O(\log n)$ for the union-by-size version of quick union.

Author's hint:

8. The argument is very similar to the one made in the section for the union-by-size version of quick find.

Problem #8 (8) 9.4.1 Huffman Code construction

(a) 4 points. When there is a choice due to a tie, place the one that appears first in the problem statement's character list "on the left" in the tree. (b) 2 points. (c) 2 points.

1. a. Construct a Huffman code for the following data:

character	A	B	C	D	_
probability	0.4	0.1	0.2	0.15	0.15

   b. Encode the text ABACABAD using the code of question a.

   c. Decode the text whose encoding is 100010111001010 in the code of question a.

Author's hint:

1. See the example given in the section.

Problem #9 (12) 9.4.3 Huffman TF **Explain your answers**

3. Indicate whether each of the following properties are true for every Huffman code.

   a. The codewords of the two least frequent characters have the same length.

   b. The codeword's length of a more frequent character is always smaller than or equal to the codeword's length of a less frequent one.

Author's hint:

3. You may base your answers on the way Huffman's algorithm works or on the fact that Huffman codes are known to be optimal prefix codes.

MA/CSSE 473 – Design and Analysis of Algorithms

Homework 14 (79 points total) Updated for Summer, 2014

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do. **All problem numbers in this assignment (except #10) are the same in editions 3 and 4.**

9.4.6	(linear time Huffman code algorithm)
10.2.2a	(maximum flow example)
10.2.5	(maximum flow algorithm for tree)
10.2.6a	(prove equation 10.9)
10.4.3	(stable marriage example)
10.4.6	(unique stable marriage solution)
10.4.10	(roommate problem)

Problems to write up and turn in:

1. (5) 9.4.4 (maximal Huffman codeword length) Once you have figured out the answer, describe a set of probabilities (or frequencies) that make that maximum happen.

2. (10) 9.4.10 (card guessing)

3. (10) Answer the questions from class(below) about the induction proof of the correctness of Kruskal's algorithm. See details below.

4. (6) 10.2.1 (sources and sinks with negative weights)

5. (5) 10.2.3a (Maximum flow solution unique?) Explain your answers.

6. (5) 10.2.4a (Maximum flow single source and sink)

7. (8) 10.4.1 (stable marriage example)

8. (4) 10.4.2 (stable marriage check algorithm)

9. (6) 10.4.5 (time efficiency of stable marriage algorithm)

10. (20) 8.3.11bc [8.3.10bc] (Matrix Chain Multiplication)

Details of problem #3 Kruskal's algorithm

The questions (based on the information below)

(a) How do we know that v was already part of some connected component of G'?

Does the addition of e to C satisfy the hypothesis of the lemma? For each statement below, explain why it is true.

(b) G’ is a subgraph of some MST for G:

(c) C is a connected component of G':

(d) e connects a vertex in C to an vertex in G – C:

(e) e satisfies the minimum-weight condition of the lemma:

The algorithm:

- To find a MST for a connected undirectedG:
  - Start with a graph G' containing all of the n vertices of G and no edges.
  - for i = 1 to n – 1:
    - Among all of G’s edges that can be added without creating a cycle, add one (call it e) that has minimal weight.

The property we are trying to prove: Before every loop execution, G' is a subgraph of some MST of G.

Proof is (of course) by induction on i.
BASE CASE: When i = 1, G' consists of only vertices of G. Since all vertices must be part of every MST for G, G is a subgraph of every MST.

INDUCTION STEP. Assume that G' is a subgraph of an MST for G. Choose e according to the above algorithm. Show that G' $\cup$ {e} is a subgraph of an MST of G.

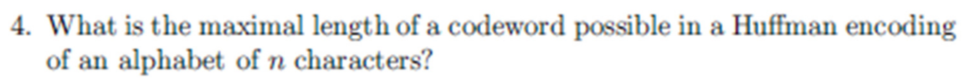
The Lemma we want to use: Let G be a weighted connected graph with a MST T; let G′ be any subgraph of T, and let C be any connected component of G′. If we add to C an edge *e=(v,w)* that has minimum-weight among all of the edges that have one vertex in C and the other vertex not in C, then G has an MST that contains the union of G′ and *e*.

In order to be able to use the lemma, we have to pick a connected component of C, and show that it satisfies the conditions of the lemma. We let v be one of the vertices of e, and let C be the connected component of G' that contains v.

Within this context, answer the 5 questions above.

MA/CSSE 473 HW 14 textbook problems and hints

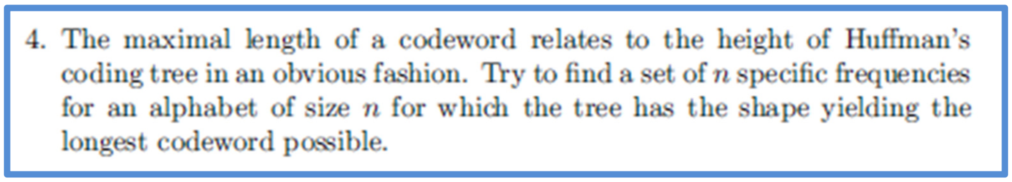
Problem #1 (5) 9.4.4 (maximal Huffman codeword length)



4. What is the maximal length of a codeword possible in a Huffman encoding of an alphabet of n characters?

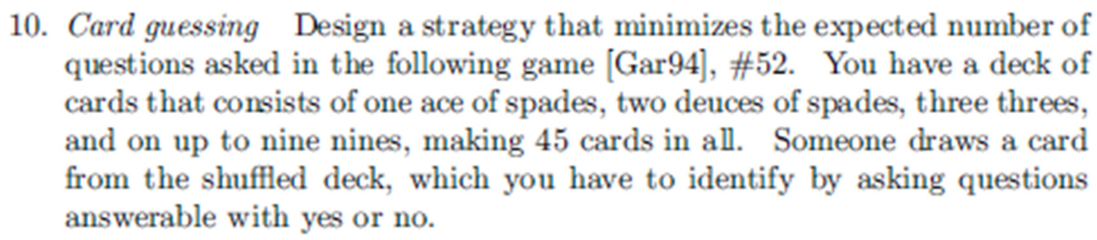
Once you have figured out the answer, describe a set of probabilities (or frequencies) that make that maximum happen.

Author's hint:



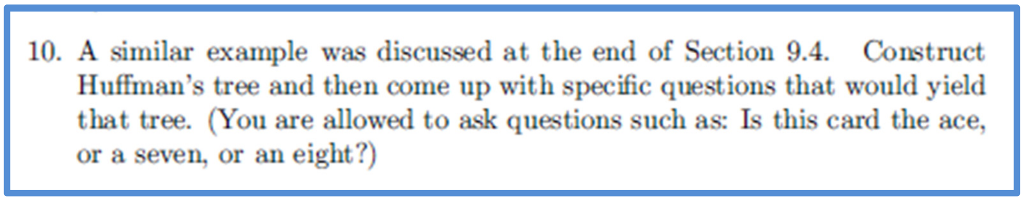
4. The maximal length of a codeword relates to the height of Huffman's coding tree in an obvious fashion. Try to find a set of n specific frequencies for an alphabet of size n for which the tree has the shape yielding the longest codeword possible.

Problem #2 (10) 9.4.10 (card guessing)



10. *Card guessing* Design a strategy that minimizes the expected number of questions asked in the following game [Gar94], #52. You have a deck of cards that consists of one ace of spades, two deuces of spades, three threes, and on up to nine nines, making 45 cards in all. Someone draws a card from the shuffled deck, which you have to identify by asking questions answerable with yes or no.

Author's hint:



10. A similar example was discussed at the end of Section 9.4. Construct Huffman's tree and then come up with specific questions that would yield that tree. (You are allowed to ask questions such as: Is this card the ace, or a seven, or an eight?)

Problem #3 (10) Kruskal proof

Problem on Kruskal's algorithm:

The questions

(a) How do we know that v was already part of some connected component of G'?

Does the addition of e to C satisfy the hypothesis of the lemma? For each statement below, explain why it is true.

(b) G' is a subgraph of some MST for G:

(c) C is a connected component of G':

(d) e connects a vertex in C to an vertex in G – C:

(e) e satisfies the minimum-weight condition of the lemma:

The algorithm:

- To find a MST for a connected undirectedG:
  - Start with a graph G' containing all of the n vertices of G and no edges.
  - for i = 1 to n – 1:
    - Among all of G's edges that can be added without creating a cycle, add one (call it e) that has minimal weight.

The property we are trying to prove: Before every loop execution, G' is a subgraph of some MST of G.

Proof is (of course) by induction on i.
BASE CASE: When I = 1, G' consists of only vertices of G. Since all vertices must be part of any MST for G, G is a subgraph of every MST.

INDUCTION STEP. Assume that G' is a subgraph of an MST for G. Choose e according to the above algorithm. Show that G' $\cup$\{e\} is a sungraph of an MST of G.

The Lemma we want to use: Let G be a weighted connected graph with a MST T; let G′ be any subgraph of T, and let C be any connected component of G′. If we add to C an edge $e=(v,w)$ that has minimum-weight among all of the edges that have one vertex in C and the other vertex not in C, then G has an MST that contains the union of G′ and e.

In order to be able to use the lemma, we have to pick a connected component of C, and show that it satisfies the conditions of the lemma. We let v be one of the vertices of e, and let C be the connected component of G' that contains v.

Within this context, answer the 5 questions above.

Problem #4 (6) 10.2.1 (sources and sinks with negative weights)

1. Since maximum-flow algorithms require processing edges in both directions, it is convenient to modify the adjacency matrix representation of a network as follows: If there is a directed edge from vertex i to vertex j of capacity u_{ij}, then the element in the ith row and the jth column is set to u_{ij}, while the element in the jth row and the ith column is set to $-u_{ij}$; if there is no edge between vertices i and j, both these elements are set to zero. Outline a simple algorithm for identifying a source and a sink in a network presented by such a matrix and indicate its time efficiency.

Author's hint

:

1. What properties of the adjacency matrix elements stem from the source and sink definitions, respectively?

Problem #5 (6) 10.2.3a (Maximum flow solution unique?) Explain your answers.

3. a. Does the maximum-flow problem always have a unique solution? Would your answer be different for networks with different capacities on all their edges?

.

Author's hint:

3. Of course, the value (capacity) of an optimal flow (cut) is the same for any optimal solution. The question is whether distinct flows (cuts) can yield the same optimal value.

Problem #6 (5) 10.2.4a (Maximum flow single source and sink)

4. a. Explain how the maximum-flow problem for a network with several sources and sinks can be transformed to the same problem for a network with a single source and a single sink.

Author's hint:

4. a. Add extra vertices and edges to the network given.

Problem #7 (8) 10.4.1 (stable marriage example)

1. Consider an instance of the stable marriage problem given by the ranking matrix

$$\begin{array}{cccc} & A & B & C \\ \alpha & 1,3 & 2,2 & 3,1 \\ \beta & 3,1 & 1,3 & 2,2 \\ \gamma & 2,2 & 3,1 & 1,3 \end{array}$$

For each of its marriage matchings, indicate whether it is stable or not. For the unstable matchings, specify a blocking pair. For the stable matchings, indicate whether they are man-optimal, woman-optimal, or neither. (Assume that the Greek and Roman letters denote the men and women, respectively.)

Author's hint:

1. A marriage matching is obtained by selecting three matrix cells, one cell from each row and column. To determine the stability of a given marriage matching, check each of the remaining matrix cells for containing a blocking pair.

Problem #8 (4) 10.4.2 (stable marriage check algorithm)

2. Design a simple algorithm for checking whether a given marriage matching is stable and determine its time efficiency class.

Author's hint:

2. It suffices to consider each member of one sex (say, the men) as a potential member of a blocking pair.

Problem #9 (6) 10.4.5 (Huffman TF)

5. Determine the time-efficiency class of the stable-marriage algorithm

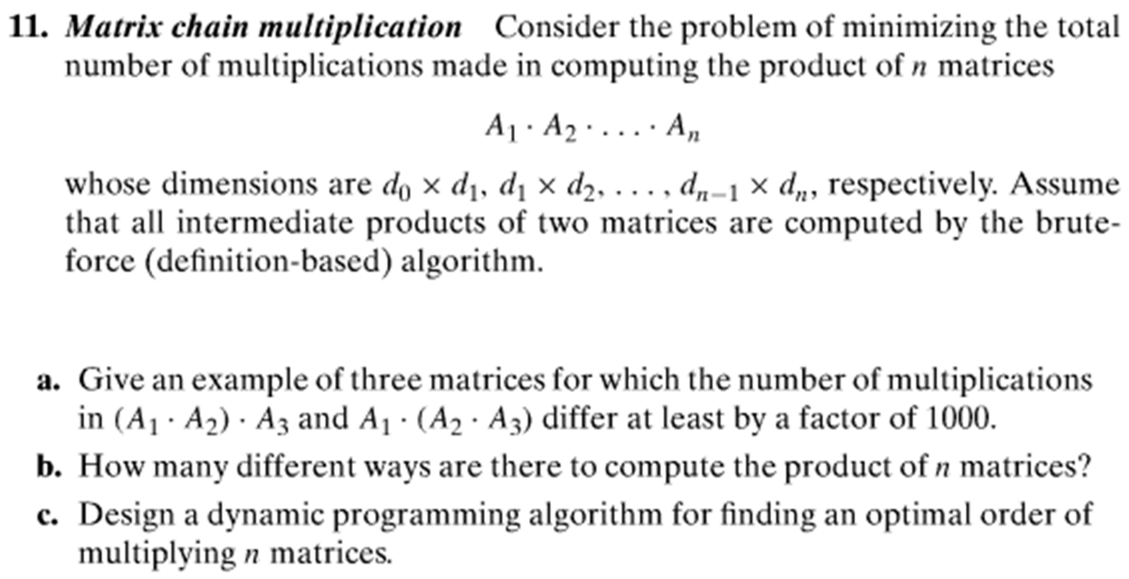
   (a) in the worst case.

   (b) in the best case.

Author's hint:

5. The time efficiency is clearly defined by the number of proposals made. You may but are not required to provide the exact number of proposals in the worst and best cases, respectively; an appropriate Θ class will suffice.

Problem #10 (20) 8.3.11bc [8.3.10bc] matrix chain mltiplication



11. ***Matrix chain multiplication*** Consider the problem of minimizing the total number of multiplications made in computing the product of n matrices

$$A_1 \cdot A_2 \cdot \ldots \cdot A_n$$

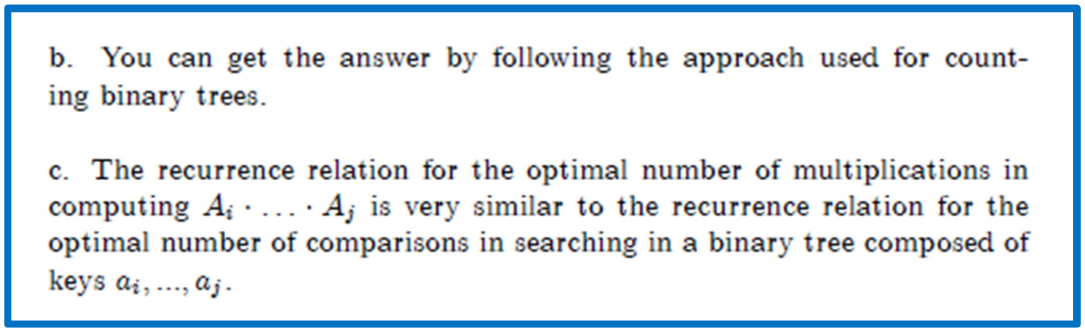
whose dimensions are $d_0 \times d_1$, $d_1 \times d_2$, ..., $d_{n-1} \times d_n$, respectively. Assume that all intermediate products of two matrices are computed by the brute-force (definition-based) algorithm.

a. Give an example of three matrices for which the number of multiplications in $(A_1 \cdot A_2) \cdot A_3$ and $A_1 \cdot (A_2 \cdot A_3)$ differ at least by a factor of 1000.

b. How many different ways are there to compute the product of n matrices?

c. Design a dynamic programming algorithm for finding an optimal order of multiplying n matrices.

Author's hint:



b. You can get the answer by following the approach used for counting binary trees.

c. The recurrence relation for the optimal number of multiplications in computing $A_i \cdot \ldots \cdot A_j$ is very similar to the recurrence relation for the optimal number of comparisons in searching in a binary tree composed of keys $a_i, \ldots, a_j$.

MA/CSSE 473 – Design and Analysis of Algorithms

Homework 15 (59 points total, plus 10 points optional extra-credit)

updated for summer 2014

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

11.1.2	(lower bound towers of Hanoi)
11.1.3	(trivial lower bounds)
11.1.6	(lower bound on sorting y exchanging adjacent elements)
11.1.11	(tight lower bound for closest numbers problem)
11.2.2	(median of 3 lower bound)
11.2.4	(best comparison-based sort for 4 elements)
11.2.9	(tournament tree)
11.2.11 [11.2.10]	(jigsaw puzzle)
11.3.5	(polynomial-time 2-coloring algorithm)

Problems to write up and turn in:

1. (5) 11.1.1 (lower bound for alternating disk algorithm)
2. (5) 11.1.4 (fake coin minimum number of guesses)
3. (12) 11.1.10 (matrix multiplication and squaring) (6, 6)
4. (9) 11.2.10ab [11.2.8ab] (advanced fake-coin problem) (4, 5)
5. (5) 11.3.1 (Chess decidable?) Explain your answer.
6. (8) 11.3.2 (tractable?) Explain your answer.
7. (5) 11.3.6 (brute force composite number)
8. (5) 11.3.7a (polynomial –time check of knapsack solution)
9. (5) 11.3.11 [11.3.10] (Venn diagrams)
10. (10) 11.3.12 [11.3.11] (King Arthur problem) Optional, extra-credit problem

MA/CSSE 473 HW 15 textbook problems and hints

Problem #1 (5) 11.1.1 (lower bound for alternating disk algorithm)

1. Prove that any algorithm solving the alternating-disk puzzle (Problem 11 in Exercises 3.1) must make at least $n(n+1)/2$ moves to solve it. Is this lower bound tight?

Author's hint:

1. Is it possible to solve the puzzle by making fewer moves than the brute-force algorithm? Why?

Problem #2 (5) 11.1.4 (fake coin minimum number of guesses)

4. Consider the problem of identifying a lighter fake coin among n identical-looking coins with the help of a balance scale. Can we use the same information-theoretic argument as the one in the text for the number of questions in the guessing game to conclude that any algorithm for identifying the fake will need at least $\lceil \log_2 n \rceil$ weighings in the worst case?

Author's hint:

4. Reviewing Section 5.5, where the fake-coin problem was introduced, should help in answering the question.

Problem #3 (12) 11.1.10 (matrix multiplication and squaring) (6, 6)

10. a. Can we use this section's formulas that indicate the complexity equivalence of multiplication and squaring of integers to show the complexity equivalence of multiplication and squaring of square matrices?

    b. Show that multiplication of two matrices of order n can be reduced to squaring a matrix of order $2n$.

Author's hint:

10. a. Check whether the formulas hold for two arbitrary square matrices.

    b. Use a formula similar to the one showing that multiplication of arbitrary square matrices can be reduced to multiplication of symmetric matrices.

Problem #4 (9) 11.2.10ab [11.2.8ab] (advanced fake-coin problem) (4, 5)

8. *Advanced fake-coin problem* There are $n \geq 3$ coins identical in appearance; either all are genuine or exactly one of them is fake. It is unknown whether the fake coin is lighter or heavier than the genuine one. You have

a balance scale with which you can compare any two sets of coins. That is, by tipping to the left, to the right, or staying even, the balance scale will tell whether the sets weigh the same or which of the sets is heavier than the other, but not by how much. The problem is to find whether all the coins are genuine and, if not, to find the fake coin and establish whether it is lighter or heavier then the genuine ones.

a. Prove that any algorithm for this problem must make at least $\lceil \log_3(2n+1) \rceil$ weighings in the worst case.

b. Draw a decision tree for an algorithm that solves the problem for $n = 3$ coins in two weighings.

Author's hint

8. a. How many outcomes does this problem have?

   b. Draw a ternary decision tree that solves the problem.

:

Problem #5 (5) 11.3.1 (Chess decidable?) Explain your answer.

1. A game of chess can be posed as the following decision problem: given a legal positioning of chess pieces and information about which side is to move, determine whether that side can win. Is this decision problem decidable?

.

Author's hint:

1. Check the definition of a decidable decision problem.

Problem #6 (8) 11.3.2 (tractable?) Explain your answer.

2. A certain problem can be solved by an algorithm whose running time is in $O(n^{\log_2 n})$. Which of the following assertions is true?

   a. The problem is tractable.

   b. The problem is intractable.

   c. None of the above.

Author's hint:

2. First, determine whether $n^{\log_2 n}$ is a polynomial function. Then read carefully the definitions of tractable and intractable problems.

Problem #7 (5) 11.3.6 (brute force composite number)

6. Consider the following brute-force algorithm for solving the composite number problem: Check successive integers from 2 to $\lfloor n/2 \rfloor$ as possible

divisors of n. If one of them divides n evenly, return yes (i.e., the number is composite), if none of them does, return no. Why does this algorithm not put the problem in class P?

Author's hint:

6. What is a proper measure of an input's size for this problem?

Problem #8 (5) 11.3.7a (polynomial –time check of knapsack solution)

7. State the decision version for each of the following problems and outline a polynomial-time algorithm that verifies whether or not a proposed solution solves the problem. (You may assume that a proposed solution represents a legitimate input to your verification algorithm.)

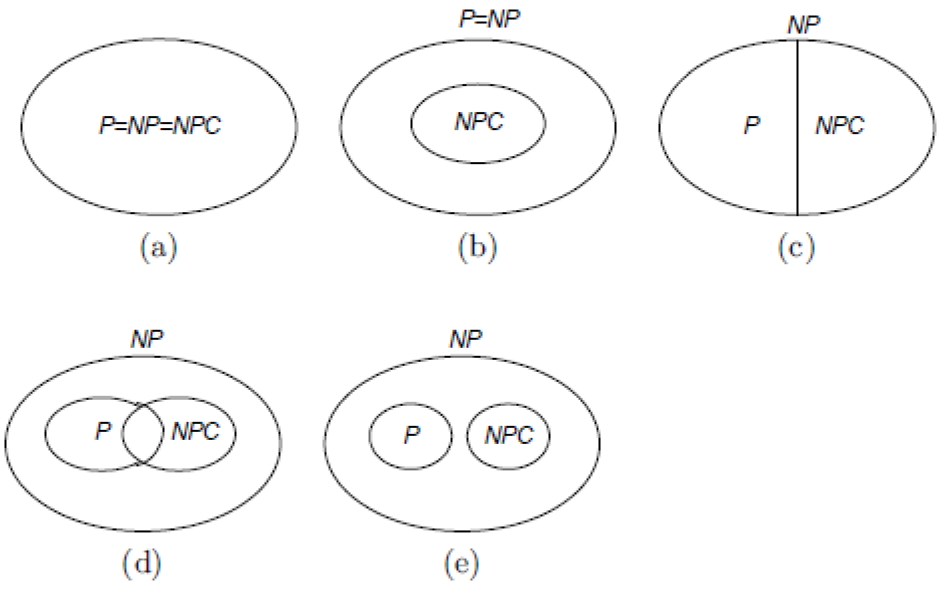
   a. knapsack problem

Author's hint:

7. See the formulation of the decision version of graph coloring and the verification algorithm for the Hamiltonian circuit problem given in Section 11.3.

Problem #9 (5) 11.3.11 [11.3.10] (Venn diagrams)

10. ▷ Which of the following diagrams do not contradict the current state of our knowledge about the complexity classes P, NP, and NPC (NP-complete problems)?



Author's hint:

10. Just two of them do not contradict the current state of our knowledge about the complexity classes.

Problem #10 (10) 11.3.12 [11.3.11] (King Arthur problem) Optional, extra-credit problem

11. King Arthur expects 150 knights for an annual dinner at Camelot. Unfortunately, some of the knights quarrel with each other, and Arthur knows who quarrels with whom. Arthur wants to seat his guests around a table so that no two quarreling knights sit next to each other.

    a. Which standard problem can be used to model King Arthur's task?

    b. As a research project, find a proof that Arthur's problem has a solution if each knight does not quarrel with at least 75 other knights.

Author's hint:

11. The problem you need is mentioned explicitly in Section 11.3.

MA/CSSE 473 – Design and Analysis of Algorithms

Homework 16 (25 points total)

updated for summer 2014

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems for enlightenment/practice/review (not to turn in, but you should think about them): How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

12.1.3	(n-queens implementation)
12.1.11 [12.1.10]	(puzzle pegs)

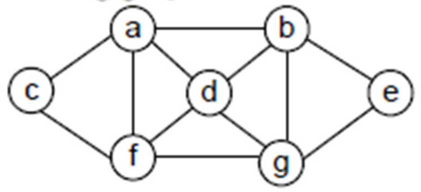
Problems to write up and turn in:

1. (5) 12.1.5 [12.1.4] (Hamiltonian Circuit) Show the state space.
2. (5) 12.2.1 (data structure for best-first branch-and-bound)
3. (5) 12.2.5 (use branch-and-bound to solve instance of knapsack problem)
4. (10) 12.3.1 (nearest-neighbor algorithm example) (4, 6)

MA/CSSE 473 HW 16 textbook problems and hints

Problem #1 (5) 12.1.5 [12.1.4] (Hamiltonian circuit backtracking) Show the state space.

4. Apply backtracking to the problem of finding a Hamiltonian circuit in the following graph.



Author's hint:

4. Another instance of this problem is solved in the section.

Problem #2 (5) 12.2.1 (data structure for best-first branch-and-bound)

1. What data structure would you use to keep track of live nodes in a best-first branch-and-bound algorithm?

Author's hint:

1. What operations does a best-first branch-and-bound algorithm perform on the live nodes of its state-space tree?

Problem #3 (5) 12.2.5 (use branch-and-bound to solve instance of knapsack problem)

5. Solve the following instance of the knapsack problem by the branch-and-bound algorithm

item	weight	value
1	10	\$100
2	7	\$63
3	8	\$56
4	4	\$12

$W = 16$

Author's hint:

5. A similar problem is solved in the section.

Problem #4 (10) 12.3.1 (nearest-neighbor algorithm example) (4, 6))

1. a. Apply the nearest-neighbor algorithm to the instance defined by the distance matrix below. Start the algorithm at the first city, assuming that the cities are numbered from 1 to 5.

$$\begin{bmatrix} 0 & 14 & 4 & 10 & \infty \\ 14 & 0 & 5 & 8 & 7 \\ 4 & 5 & 0 & 9 & 16 \\ 10 & 8 & 9 & 0 & 32 \\ \infty & 7 & 16 & 32 & 0 \end{bmatrix}$$

b. Compute the accuracy ratio of this approximate solution.

Author's hint

1. a. Start by marking the first column of the matrix and finding the smallest element in the first row and an unmarked column.

   b. You will have to find an optimal solution by exhaustive search as explained in Section 3.4.

:

MA/CSSE 473 – Design and Analysis of Algorithms

Virtual Homework 17

This is not an actual assignment to be turned in. I present this list of problems to give an idea of the level of understanding of Chapter 11 material that you should have for the Final Exam.

1. 11.1.2
2. 11.1.3
3. 11.1.4
4. 11.1.7
5. 11.1.10
6. 11.1.11
7. 11.2.2
8. 11.2.4
9. 11.3.1
10. 11.3.2
11. 11.3.5
12. 11.3.9