Programming Assignment Problems

The project has two parts: Part I and Part II.
For each problem, you are required to submit (by email) the algorithm, the code, sample data, results, a one page report, and instructions for executing the code, to the GTA.
Presentation of Results: Measure CPU times, compare results and use tables or plots.

PLEASE NOTE THAT your PROGRAMS WILL BE TESTED WITH DIFFERENT DATA SETS by the GTA.

The submissions should be made on or before 9 AM December 15, 2010
(You can finish the project earlier if you wish).

Assignment Problems

3-Person Team
Part I: Any Four Problems
PLUS
Part II: All Problems

2-Person Team
Part I: Any three Problems
PLUS
Part II: Problem 1 AND (Problem 2 or Problem 3.)

1-Person Team
Part I: Any two Problems
PLUS
Part II: Any one Problem

PART I
(for Problems 1 through 5 Please refer Text book: Algorithm Design by Kleinberg and Tardos)

2. Chapter 6, Pg 319, Problem 8.
5. Chapter 7, Pg 439, Problem 38.
6. See next page
7. See next page
Part I, Problem 6:

The Quicksort algorithm is an efficient and popular sorting technique that sorts a list of keys \(S[1], S[2], \ldots, S[n]\), recursively by choosing a pivot key. The best-case running time of Quicksort is \(O(n \log_2 n)\) and its worst-case running time is \(O(n^2)\). Several improvements and modifications have been proposed to improve Quicksort’s worst-case behavior. For example, the paper by Wainwright [1] presents Bsort, a variation of Quicksort that combines Bubble-sorting techniques with the Quicksort algorithm. Other methods include, Quickersort[2], qsort[3], CKsort[4]. You can choose ONE improvisation of Quicksort (of your choice) – let’s call it MY_CHOICE_QSORT. Alternatively, you can choose a method not listed above, but available in the literature, please include the reference in your report.

Write programs to implement sorting algorithms that employ MY_CHOICE_QSORT, Quicksort, Mergesort and Heapsort for sorting keys. Execute your sorting programs for the following sets of data:

a. Set_1: Ordered List
b. Set_2: Reverse order List
c. Set_3: A list containing the same value throughout
d. Set_4: Random List
e. Set_5: 25% of the List sorted

Presentation of Results: Measure CPU time, number of partitions (only for Quicksort and, MY_CHOICE_QSORT) and number of comparisons for data sizes 1000, 10K, and 1M. Present your results using tables or graphs and write a 1-page report. The report should have a pseudocode for MY_CHOICE_QSORT and summarize the behavior of all Sorting algorithms tested and their suitability.

References

Part I, Problem 7

Write a program to extend the Rabin-Karp method to handle the problem of looking for a given \(m \times m\) pattern in an \(n \times n\) array of characters. (The pattern may be shifted vertically or horizontally, but it may not be rotated). Use another method, for example, a variation of the KMP algorithm, a different method from a research paper, or your own method and compare the results with those of the Rabin-Karp extension.

a. Set_1: \(n=64, m=4, 8, \text{ and } 16\)
b. Set_2: \(n=256, m=4, 16, \text{ and } 64\)
c. Set_3: \(n=1024, m=4, 16, \text{ and } 64\)

Presentation of Results: Measure CPU time, compare results and present using tables or plots.
Part II, Problem 1:

The list in [http://www.fec.gov/pages/elevote.htm](http://www.fec.gov/pages/elevote.htm) gives the number of electoral votes for each state for the Presidential election (the candidate receiving the majority of the votes in a state collects all the electoral votes for that state.) There are altogether 538 electoral votes. Write a heuristic algorithm (e.g., Branch and Bound, Backtracking, or other) to determine if it is possible for the election to end up in a tie. If so what are the possible configurations. Implement your algorithm.

Part II, Problem 2:

Consider a communication network that can be modeled as a weighted undirected connected graph \( G = (V, E) \). Each site in the network is represented as a vertex and each line of communication is bidirectional and has a cost associated with it. The cost may correspond to the expected delay on the line, or to the tariff for using each line. Each site has online local information; i.e., it knows online the edges (and vertices) adjacent to it. An MCST of the network can be used to broadcast messages to all sites. If we broadcast the messages by using only the edges of the MCST, then the total cost is minimized. Assume that such an MCST is computed by some method and that each site knows which of the edges adjacent to it belongs to the MCST. Assume now that the sites in a certain subset \( U \) subset of \( V \) share between them the information that is known to all of them. In other words, every site in \( U \) knows not only about the edges and vertices adjacent to itself, but it also knows all the edges and vertices adjacent to all vertices of \( U \). Furthermore, assure that the partial MCST restricted to \( U \) is connected (i.e. it is a tree). Consider an edge \( e \in U \), which belongs to the MCST, whose cost has just changed.

a. Find the conditions under the change in \( e \)’s cost is guaranteed not to affect the MCST. Consider only conditions that can be checked with the information known to the sires in \( U \). In other words, how can sites in \( U \) determine that no action needs to be taken top modify the MCST after the change?

b. Find the conditions under which the modified MCST is different from the original one only in edges that belong to \( U \) (hence, the change can be handled locally). Consider only conditions that can be checked with the information known to the sites in \( U \).

c. Design and implement an algorithm to check for the conditions in parts a and b (again only within \( U \), and to modify the MCST accordingly. The algorithm does not need to handle the case where the change to the MCST may be outside \( U \). Generate a random graph with uniform distribution of edges to be the underlying structure.

Part II, Problem 3:

Write a program that constructs Huffman code for a given English text and encode it. Write a program for decoding an English text that has been encoded with Huffman code. Experiment with your encoding and decoding programs for at least 5, 10, 25, 50 and 100 different texts, each text having a minimum of 50 words and at least one page in length. What range of compression ratios do you get? Can you improve the compression ratios by using some estimates of frequencies instead of actual frequencies? You are encouraged to look into published literature for ideas to improve the compression ratio.