Homework 2*

Algorithms
Spring 2020 CS207@IITG

---

(1) Assuming the frequencies of symbols in set S are not necessarily distinct, prove the correctness of Huffman's algorithm for computing an optimal prefix-code. Prove the solution output by this algorithm is both feasible as well as optimal. In proving, you may use exchange argument at as many places as possible.

(2) Precisely identify the optimal substructure property for the following problem: given a set S of n jobs, execute all the jobs in S while using the minimum number of machines with preemption of jobs not allowed. Further, determine whether we used this property in proving the correctness (with meeting the lower bound argument) of the greedy algorithm suggested in class for this problem.

(3) Assuming the input graph $G(V, E)$ is given in adjacency list form, analyze the worst-case time complexity of local search based heuristic given for the vertex cover problem. And, prove this time complexity is indeed tight.

(4) Given a set S of n jobs, each with a start time, a finish time, and a positive integer profit, compute a set $S' \subseteq S$ such that jobs in S' are mutually compatible and $\sum_{j \in S'} profit(j)$ is the maximum possible. Prove the following properties: (i) the problem has the optimal substructure property, (ii) overlapping subproblems in a naive implementation of DP recurrence, and (iii) there exists a linear ordering of subproblems which requires solving only a polynomial number of subproblems.

Further, prove that the algorithm suggested based on the DP recurrence outputs the correct solution value; in specific, if the set $S'' = \{i'_1, i'_2, \ldots, i'_j\}$ of jobs is an optimal solution for the given problem instance, then prove that memoization essentially computes the solution value that is not inferior to the solution value of S''.

In the above problem, for any n, for every $1 \leq i \leq n$, the value of $OPT(i)$ must be known to compute $OPT(n)$. Prove or disprove.

In specific, if S'' is the only solution for the given input, then argue that both the memoization and tabulation based algorithms are guaranteed to output S''.

(5) For parenthesizing matrix-chain multiplication problem, write C code for the following:

(a) recursive implementation of DP recurrence without memoization,

(b) memoizing the DP recurrence,

(c) iterative implementation that computes a tableau with each entry corresponding to a subproblem of interest: entry (i, j) needs to save the optimum solution value $m[i, j]$ as well as the split position $k \in [i, j)$ such that
((optimal-parenthesization$(A_i, \ldots, A_k)$)(optimal-parenthesization$(A_{k+1} \ldots A_j)$))
is an optimal parenthesization of $A_i \ldots, A_j$, and

(d) given the above tableau, output an optimal parenthesization of the matrix-chain.

---

*Prepared by R. Inkulu, Department of Computer Science, IIT Guwahati, India. http://www.iitg.ac.in/rinkulu/

(6) Prove the following:

   (a) There is a correspondance between the number of parentheiszations of a matrix-chain of length n and the number of binary search trees on n keys.
   (b) The number of parenthesizations of a matrix-chain of length n is $\Omega(\frac{4^n}{n^{3/2}})$.
   (c) The number of feasible alignments of two sequences X and Y is exponential in $|X|$ and $|Y|$.

(7) Prove the following: For the fractional knapsack problem, recursively picking an item with highest value per unit weight always leads to an optimal solution.

   Give three non-trivial greedy strategies for 0-1 Knapsack problem, and prove that none of them can yield a polynomial time algorithm for this problem.

(8) Here is one more problem to depict the demarcation between the power of greedy algorithms and DP-based algorithms: [CLRS] exercise 16-1 (pages 446-447).