1. DP

In this problem you will write a dynamic program to solve the following typesetting scenario. You have an input text consisting of a sequence of \(n\) words of lengths \(1, 2, \ldots, n\), where the length of a word is the number of characters it contains. Your printer can only print with its built-in Courier 10-point fixed-width font set that allows a maximum of \(M\) characters per line. (Assume that \(\ell_i \leq M\) for all \(i = 1, \ldots, n\).) When printing words \(i\) and \(i+1\) on the same line, one space character (blank) must be printed between the two words. In addition, any remaining space at the end of the line is padded with blanks. Thus, if words \(i\) through \(j\) are printed on a line, the number of extra space characters at the end of the line (after word \(j\)) is

\[
M - j + i - \sum_{k=i}^{j} \ell_k
\]

There are many ways to divide a paragraph into multiple lines. To produce nice-looking output, we want a division that fills each line as much as possible. A heuristic that has empirically shown itself to be effective is to charge a cost of the cube of the number of extra space characters at the end of each line. To avoid the unnecessary penalty for extra spaces on the last line, however, the cost of the last line is 0. In other words, the cost \(\text{linecost}(i, j)\) for printing words \(i\) through \(j\) on a line is given by

\[
\text{linecost}(i, j) = \begin{cases} 
\infty & \text{if word } i \text{ through } j \text{ do not fit on a line,} \\
0 & \text{if } j = n \text{ (i.e., last line),} \\
(M - j + i - \sum_{k=i}^{j} \ell_k)^3 & \text{otherwise.}
\end{cases}
\]

The total cost for typesetting a paragraph is the sum of the costs of all lines in the paragraph. An optimal solution is a division of the \(n\) words into lines in such a way that the total cost is minimized.

(a) Argue that this problem exhibits principal of sub-optimal substructure.

(b) Recursively define the value of an optimal solution.

(c) Give an efficient algorithm to compute the cost of an optimal solution. Analyze the running time and storage space of your algorithm.

2. You bring an \(L\)-foot log of wood to your favorite sawmill. You want the log cut at \(k\) specific places, \(L_1, L_2, \ldots, L_k\) feet from the left end, where \(L_1 < L_2 < \ldots < L_k < L\). The sawmill charges \(x\) dollars to cut an \(x\)-foot log once, in any place you want. Give an efficient algorithm to minimize the total cost.

3. In this problem, assume that graphs are given by an adjacency-list representation.

(a) An independent set of a graph \(G = (V, E)\) is a subset \(V' \subseteq V\) of vertices such that for any two vertices \(u, v \in V'\), \((u, v) \not\in E\) (i.e., there is no edge between any two vertices in an independent set). A maximal independent set is an independent set \(V'\) such that for all vertices \(v \in V - V'\), the set \(V' \cup \{v\}\) is not independent (i.e., every vertex not in \(V'\) is adjacent to some vertex in \(V'\)). Give an \(O(|V| + |E|)\) algorithm for finding a maximal independent set for any graph \(G = (V, E)\).

(b) A \(k\)-coloring of an undirected graph \(G = (V, E)\) is a function

\[
\chi : V \rightarrow \{1, 2, 3, \ldots, k\}
\]

such that \(\chi(u) \neq \chi(v)\) for every edge \((u, v) \in E\). In other words, each vertex is given a color (a number in \(\{1, 2, 3, \ldots, k\}\)), and adjacent vertices have different colors. Give an \(O(|V| + |E|)\) algorithm that finds a \((d + 1)\)-coloring of a graph \(G = (V, E)\), where \(d\) is the largest degree of the vertices of \(G\).
4. (20 Pts.) **LP relaxation of ILP for CNF-Sat**

In CNF-Sat problem, you are given a boolean formula $f$ that is a conjunction of $m$ clauses $C_1, ..., C_m$ and each clause is a disjunction of $n$ variables and their negations $x_1, ..., x_n, \bar{x}_1, ..., \bar{x}_n$. You are interested in finding a truth arrangement $x_i = 0$ or $1$ for each $i = 1, ..., n$, such that $f$ evaluates to 1. The CNF-Sat is known to be NP-complete.

(a) write down the CNF-Sat as an integer programming Problem. Interpret its LP relaxation.

(b) Suggest a simple algorithm that determines whether the LP relaxation is feasible, and if so, find a (fractional) solution.