IITM-CS2800-2025 : Design and Analysis of Algorithms

Practice Problem Sheet #9 Released On : Apr 9, 9 AM

Part A - to be done in class

In this tutorial, we will practice the basic dynamic programming techniques through three simple examples. Recall from the preliminary form of this technique that we did in the lecture (we revise this form to be more precise in the second part of this tutorial) that a dynamic programming solution for an algorithmic optimization problem involves the following.

1. By a divide and conquer type argument or analysing the structure of the optimum solution, derive a recursive formulation of the optimum value in terms of optimum value for the subproblems. The subproblems may be "overlapping".

2. The subproblems must have an ordering among them. Usually this may be based on the size of the problem, but it may vary.

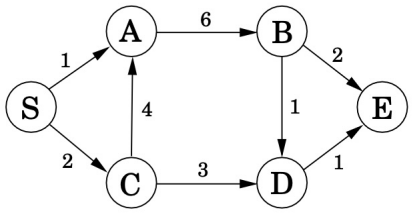
3. Derive a DP DAG which has vertices as the subproblems. Let A and B be subproblems, we add an edges (A, B) if solving subproblem B requires solving subproblem A as per our recursive formulation in the step 1.

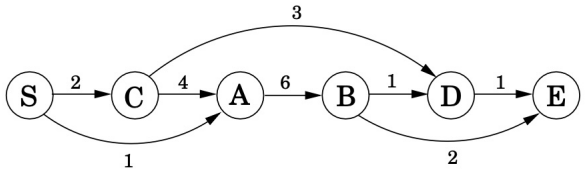
4. In the topological ordering of the DP DAG, incrementally compute the optimal solution at the first node in the DP DAG in the topological order, and then the second etc. This way, whenever the computation uses the above recursive formulation to compute the optimal value at a subproblem B, the required optimal solution for the subproblems (like A in the above example) have already been computed (since it would be a node that is before in the topological order).

5. We actually want the adjacency list of the reverse of the DP graph. In some DP problems, this is going to be straightforward from the recursive formulation itself, and for some other problems it may take computational steps to find it.

Using the above framework, we will attempt three simple algorithmic optimization problems which can be solved by using DP method.

1. Given a DAG $G(V, E)$, with lengths on edges $\ell : E \to \mathbb{R}$. Let s be a source vertex (and the first vertex) in the graph G, let the length of a path be the sum of the lengths of its edges. This problem asks you to compute the length of the shortest path from s to every vertex. For example, compute this for the following graph (RHS is a topological sort of the LHS graph):





(a) For a vertex v, define $dist(v)$ to be the length of the shortest path from s to v. Prove the following recursive formulation formally:

$$dist(v) = \min_{(u,v)\in E}\{dist(u) + \ell(u,v)\}$$

(b) $D(v)$ be the subproblem of computing $dist(v)$ be a subproblem. Describe the DP DAG obtained by the above formulation.

(c) Write down the DP algorithm for solving the problem.

(d) Analyse the running time. How do we find the adjacency list of the reverse of the DP DAG?

(e) Can you modify the above formulation to compute the length of the longest paths to each vertex from s in the DAG?

2. In the longest increasing subsequence problem, the input is a sequence of numbers $a_1, \ldots, a_n$. A subsequence is any subset of these numbers taken in order, of the form $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ where $1 < i_1 < i_2 < \ldots < i_k \leq n$, and an increasing subsequence is one in which the numbers in the sequence are strictly increasing. The task is to find the increasing subsequence of greatest length. For instance, the longest increasing subsequence of $5, 2, 8, 6, 3, 6, 9, 7$ is $2, 3, 6, 9$.

(a) We analyse the optimal solution structure. Consider the following DAG that is constructed out of the sequence. Which all indices can take part in an increasing subsequence that ends at the index 5 in the above sequence. Indeed, it can only be index 2. So the longest increasing subsequence ending at index 5 is at least one more than the longest increasing subsequence ending at index 2.

To capture this, let us define the following DAG. The vertices are the indices $\{1, 2, \ldots, n\}$. add directed edges (i, j) if a_i and a_j be consecutive elements in some increasing subsequence. That is, $E = \{(i, j) \mid i < j, a_i < a_j\}$

Let $\ell(j)$ denote the length of the longest subsequence ending at the index j (equivalently, length of the longest path in the above graph G that ends at vertex j) Prove that:

$$\ell(j) = 1 + \max_{(i,j)\in E} \ell(i)$$

(b) Thus, there is an ordering on the subproblems, and a relation that shows how to solve a subproblem given the answers to "smaller" subproblems, that is, subproblems that appear earlier in the ordering. Let $L(j)$ be the problem of computing $\ell(j)$ for $j \in [n]$. Explicitly define the DP DAG. What are the vertices and what are the edges?

(c) Write down the Dynamic programming solution for the problem and analyse the running time.

(d) How do you produce the actual increasing subsequence and not just the length of the longest increasing subsequence.

3. The maximum sum subarray problem that we had seen in the *divide-and-conquer* theme of this course is the following problem : Given an array A containing n integers, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum. We had solved this using a divide and conquer strategy giving $O(n \log n)$ time algorithm for it. We now give a linear time algorithm using the dynamic programming strategy.

   (a) We define the subproblem first. Let $MS[i]$ be the maximum sum contiguous subarray that ends at index i. By analyzing the structure of an optimal solution, prove that:

   $$MS[i] = \max\{MS[i-1] + A[i], MS[i]\}$$

   (b) Define the DP DAG from the above. Aruge that the reverse of the DAG can be efficiently obtained from the above formulation.

   (c) Use the above and describe an iterative DP solution for the problem which works in $O(n)$ time.

   (d) How will you modify the above algorithm to output the actualy subarray indices which achieves the maximum sum.

Part B - to be worked out offline

We refine our terminology a bit more now:

- **State:** A subproblem that we want to solve. The subproblem may be complex or easy to solve but the final aim is to solve the final problem which may be defined by a relation between the smaller subproblems. Represented with some parameters.
- **Transition:** Calculating the answer for a state (subproblem) by using the answers of other smaller states (subproblems). Represented as a relation b/w states.

General Technique to solve any DP problem

1. **State:** Clearly define the subproblem. Clearly understand when you are saying dp[i][j][k], what does it represent exactly.
2. **Transition:** Define a relation b/w states. Assume that states on the right side of the equation have been calculated. Don't worry about them.
3. **Base Case:** When does your transition fail? Call them base cases answer before hand. Basically handle them separately.
4. **Final Answer:** What is the problem demanding you to find?

Time and Space Complexity in DP

- **Time Complexity:**
  Estimate (Upper bound): Number of States * Transition time for each state
  Exact: Total transition time for all states that are used.
- **Space Complexity:** Number of States * Space required for each state

---

For all the problems that follow, write down the DP state (what it represents), the base case, the final answer and the transition.

1. Given a 2D grid (NxM) with numbers written in each cell, find the path from top left $(0, 0)$ to bottom right $(N-1, M-1)$ with minimum sum of values on the path.
2. Given an array of integers (both positive and negative). Pick a subsequence of elements from it such that no 2 adjacent elements are picked and the sum of picked elements is maximized.
3. We have N stones, numbered $1, 2, \ldots, N$. For each i $(1 \leq i \leq N)$, the height of Stone i is h_i.

   A frog starts on Stone 1 and can jump either to Stone $i+1$ or Stone $i+2$. The cost incurred for a jump from Stone i to Stone j is $|h_i - h_j|$.

   Find the minimum possible total cost incurred before the frog reaches Stone N.

4. The frog can now jump from Stone i to any of the Stones $i+1, i+2, \ldots, i+K$. The cost is still calculated as $|h_i - h_j|$.

   Find the minimum possible total cost incurred before the frog reaches Stone N.

5. Taro's summer vacation consists of N days. On each day i ($1 \leq i \leq N$), he can choose one of the following activities:

   - A: Swim in the sea and gain a_i points of happiness.
   - B: Catch bugs in the mountains and gain b_i points of happiness.
   - C: Do homework at home and gain c_i points of happiness.

   Taro cannot do the same activity on consecutive days. Find the maximum possible total happiness he can obtain.

6. You are given a grid with H rows and W columns. The cell (i, j) is described by a character $a_{i,j}$:

   - If $a_{i,j} =$ ".", it is an empty square.
   - If $a_{i,j} =$ "#", it is a wall square.

   The starting position $(1, 1)$ and the ending position (H, W) are guaranteed to be empty squares. Taro can only move right or down to an adjacent empty square.

   Find the number of ways Taro can reach (H, W) from $(1, 1)$.

7. Your task is to count the number of ways to construct sum n by throwing a dice one or more times. Each throw produces an outcome between 1 and 6.

8. Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to produce a sum of money x using the available coins in such a way that the number of coins is minimal and output the number of coins.
   For example, if the coins are $\{1, 5, 7\}$ and the desired sum is 11, an optimal solution is $5+5+1$ which requires 3 coins.

9. Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to calculate the number of distinct ways you can produce a money sum x using the available coins.
   For example, if the coins are $\{2, 3, 5\}$ and the desired sum is 9, there are 8 ways: $2+2+5$, $2+5+2$, $5+2+2$, $3+3+3$, $2+2+2+3$, $2+2+3+2$, $2+3+2+2$, $3+2+2+2$.

10. Consider an $n \times n$ grid whose squares may have traps. It is not allowed to move to a square with a trap.
    Your task is to calculate the number of paths from the upper-left square to the lower-right square. You can only move right or down.

11. Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to calculate the number of distinct ordered ways you can produce a money sum x using the available coins. For example, if the coins are $\{2, 3, 5\}$ and the desired sum is 9, there are 3 ways: $2+2+5$, $3+3+3$, $2+2+2+3$.

12. There is a street with $n \times 2$ plots, where there are n plots on each side of the street. The plots on each side are numbered from 1 to n. On each plot, a house can be placed.

    Return the number of ways houses can be placed such that no two houses are adjacent to each other on the same side of the street.

    Note: If a house is placed on the i-th plot on one side of the street, a house can also be placed on the i-th plot on the other side of the street.

13. You are given a 0-indexed binary string s which represents the types of buildings along a street where:

    - $s[i] = 0$ denotes that the i-th building is an office.
    - $s[i] = 1$ denotes that the i-th building is a restaurant.

    As a city official, you would like to select 3 buildings for random inspection. However, to ensure variety, no two consecutive buildings out of the selected buildings can be of the same type.

    For example, given s = "001101", we cannot select the 1st, 3rd, and 5th buildings as that would form "011", which is not allowed due to having two consecutive buildings of the same type.

    Return the number of valid ways to select 3 buildings.

14. The Tribonacci sequence T_n is defined as follows:

$$T_0 = 0, \quad T_1 = 1, \quad T_2 = 1$$

$$T_{n+3} = T_n + T_{n+1} + T_{n+2} \quad \text{for } n \geq 0.$$

    Given n, return the value of T_n.

15. You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

    Return `true` if you can reach the last index, or `false` otherwise.