CHECKPOINT

Coding Club, IIT Guwahati

< WEEK 1 >

INTRO TO DP

We are introduced to dynamic programming from the time we first implement prefix sums of an array. However, the concept of storing previous results and using them can enable us to solve many problems with exponential complexity in polynomial time.
Techniques to solve Dynamic Programming Problems:
1. Top-Down(Memoization):
Break down the given problem in order to begin solving it. If you see that the problem has already been solved, return the saved answer. If it hasn't been solved, solve it and save it. This is usually easy to think of and very intuitive, This is referred to as Memoization.

2. Bottom-Up(Dynamic Programming):
Analyse the problem and see in what order the subproblems are solved, and work your way up from the trivial subproblem to the given problem. This process ensures that the subproblems are solved before the main problem. This is referred to as Dynamic Programming.

No doubt this is the widest topic in Competitive Programming, but it's not something which cannot be taught.
Few resources:-

- CPH (pg 65 to 69) (If you don't understand in one read, read it again)
- Errichto (**Lecture 1** and **Lecture 2**)
- Luv Competitve Programing **Dp playlist**
- Suggested Read: ***Geeks For Geeks***

If you didn't understand even after referring to the resources, don't worry it's normal, just jump to solving problems.

DP IN PYTHON

Dp in python is the same as that in cpp. Here you use lists to memorize instead of arrays in cpp. You can refer **here** to learn how to use python for dp.

There is one problem with python when it comes to dp though. Python functions take up more time and space when compared to C++ counterparts .There is a high chances of getting **runtime error**(RE) or **Time limit exceeded error**. (TLE) in recursive code due to the following reason:

Maximum Recursion Depth exceeded (MRD):

Problem: In the backend, all recursion calls are stored in a stack. When the number of recursion calls exceeds the stack size, you get this error.

Solution: Import the sys module and use the **set recursion limit** attribute to increase the recursion limit to a value that doesn't give RE (a matter of trial and error). A general value that works in many problems is 3* 10^5.

Memory Limit Exceeded (MLE):

Problem : Every problem has a particular memory limit that you shouldn't exceed. But when your stack size increases, it uses more memory giving out MLE.

Solution : Like the soln above, you could reduce the stack size using the **set recursion limit** attribute but then again it would give you a MRD. So the best thing would be to avoid recursive calls at all. This can be done by writing the iterative version of the recursive logic you used.

Things to keep in mind:

So always remember that, if you are using python to solve a dp problem, it is highly advised to submit an iterative logic rather than a recursive logic. If the iterative logic gives TLE even if your complexity is well within the Intended complexity of the problem, switch to cpp for solving problems of dynamic programming.

Standard Problems

- **Frog 1**
- **Frog 2**
- **Vacation**
- **Knapsack 1 (Knapsack Problem,** Very Important, try it first if you have no clue how to solve it then refer this or this)
- **Buy and Sell Stocks(1):** Think of a simple greedy approach. Does it work??
- **Buy and Sell Stocks(2):** Try to apply DP. A greedy solution exists too. Can you prove it?
- **Buy and Sell Stocks(3):** You can see how why DP is important as greedy solution fails.
- **Buy and Sell Stocks(4):** Now try to generalize for k steps
- **Longest Increasing Subsequence (A):** Here approach of O(n^2) will work. Refer to Additional problems for solving this in O(nlogn).
- **Longest Common Subsequence**

Some Additional Problems

- **Coin Combinations I**
- **Coin Combinations II**
- **Longest Increasing Subsequence (B):** There exist a solution in O(nlogn), which is bit tricky to figure out on it's own. But you should try to understand how it is done.

  Solution
- **Array Description**
- **New Year and the Permutation Concatenation**
- **Super Egg Drop**

ROLLING OPTIMISATION

Reduces the space complexity form O(n) to O(1)

Based on the fact that you do not need to store data once it has be used to calculate all the states it can. Recall the iterative version of Fibonacci number problem. We only need the information about two previous states for the calculation of our current state (F[n] = F[n-1] + F[n-2]). So, the idea is to start calculating F[3] with the help of F[2] and F[1] and then forget about F[1] as F[4] doesn't depends on F[1] and so on.

If your logic seems correct, but still Memory Limit Exceeded in showing, try this approach

PROBABILITIES AND DP STATES

Probabilistic Dynamic Programming (DP) constitutes a complete chapter on its own. However, its concept of finding expectations using DP states proves highly valuable in certain competitive programming problems.

The fundamental idea is to decompose the expectation value function into two or more terms involving previously calculated values multiplied by their respective probabilities. This concept may seem logical, but it can also be a bit vague at first – particularly in understanding how probabilities are multiplied with expectation values. This understanding, in fact, arises from the Law of Total Expectations.

- **Ilyn and Escalators** | **Solution**
- **Sushi** | **Solution**

BITMASK DP(optional)

What is Bit masking?

Suppose we have a collection of elements which are numbered from 1 to N. If we want to represent a subset of this set then it can be encoded by a sequence of N bits (we usually call this sequence a “mask”). In our chosen subset the i-th element belongs to it if and only if the i-th bit of the mask is set i.e., it equals to 1. For example, the mask 10000101 means that the subset of the set [1... 8] consists of elements 1, 3 and 8. We know that for a set of N elements there are total 2N subsets thus 2N masks are possible, one representing each subset. Each mask is, in fact, an integer number written in binary notation.

Bitmasking DP usually have exponential time complexity, so value of n is very small (<25)

Few resources:-

- CPH (102-106)
- Nwatx

PROBLEMS:

- **O - Matching**
- **Team Building**
- **Elevator Rides**
- **Carrying Conundrum**
- **Hammiltonian Flights**

*In future if you want to study more things in DP then you can refer **Atcoder DP Contest(It covers almost all DP techniques)** and for reading more **USACO Guide**

Recommended practice problems for mastering DP:

- **CSES DP Section:** Visit the DP section on the CSES problem set and try to solve problems from there.
- **AtCoder DP Contest:** Explore the DP section in the AtCoder Beginner Contest series, known for its comprehensive DP problems.

Additional Intermediate Problems:

- Tenzing and Balls
- Jellyfish and MEX
- Sending a Sequence Over the Network
- Zero Path
- Road Optimization
- Korney Korneevich and XOR
- Block Sequence
- Selection(kriti-junior)
- Journey
- Coloring Trees
- Mahmoud and a Message

Additional Hard Problems:

- Blocking Elements
- Magic Will Save the World
- The Values You Can Make
- Counting Rhyme