Dynamic Programming

Aditya Mishra - **IICPC**

June 9, 2024

1 Introduction

Dynamic programming is a powerful tool you can use to solve a problem in a way that can take its time complexity from exponential to pseudo polynomial-time!
Some things you must keep in mind while using this handout:

- The solutions of all these problems can be found easily on the internet.
- You can use IICPC community on discord or other social media to discuss the problems in case you are stuck.
- You must code all the problems yourself if you want to learn.
- If you get stuck on a problem, read the first few lines of its solution, take a hint, and try again.
- The author strongly emphasizes that if you just read the solution and think to yourself "Oh I got this" and move on to the next problem without properly coding the problem, you will not learn.

2 Fibonacci!

The DP journey of any programmer starts whith understanding a linear time algorithm for calculating n^{th} fibonacci number!

$$F_n = F_{n-1} + F_{n-2}$$

For finding F_n, all information you need is the values of $F_n - 1$ and $F_n - 2$. Given $F_0 = 1$ and $F_1 = 1$, we can calculate $F_n + 1$ in n operations! Comparing with the naive recursive implementation:

```
def fib(n):
    if n == 1 or n == 0:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

This implementation makes exponential number of function calls, with each step making 2 calls until you reach the base-case, so basically there an exponential number of operations involved and this implementation is too slow. As an exercise, try to find number of function calls for fib(n) for the exponential implementation.
This leads us to an important class of dynamic programming problems, where we need to find the value of a function (call it dp[.]) over an array of length n (dp[n]), and we find this value dp[n] using previously calculated values:
dp[n-1], dp[n-2], dp[n-3]... Some array DP problems around these concepts are:

- **Climbing Stairs**
- **Jump Game I**
- **Counting Towers-CSES**
- **Jump Game II**
- **Jump Game III**

3 Knapsack!

The most classic use of dynamic programming!
Given a list of n chocolates, with each chocolate having a joy value of a[i] and a diabetes value of b[i], maximise the total joy value you can achieve while keeping the total diabetes value¡=W. This problem is called the 0/1 knapsack problem and is NP-Complete, which means there is no known algorithm that is polynomial-time in N. Dynamic programming allows us to have an algorithm that is pseudo-polynomial time!
The algorithm goes as follows: Make a 2-D array DP with the integers 1,2,3,4....W on 1 axis and the items on the other axis. For each item, traverse over the diabetes axis and calculate the maximum joy you can achieve for each max diabetic value that is less than or equal to W. You need to calculate this by updating the array such that

DP[i,j]=max(1+DP[i-1,j-b[j]])

where i is the i-th item and j is your position on the weight-axis. We arrive at this relation by considering the fact that for each element, we either include it for the given j, or we dont! If we include the element for the given j, the problem reduces to finding the max joy value for the items 1 through i-1 with max permissible diabetic value j-b[i]. As we are traversing a 2-D DP array of size WxN, the time complexity of the algorithm is O(W*N). A nice explanation of the above algorithm can be found here: YouTube
Solve these problems properly and you will understand Knapsack problems very well:

- **Knapsack 1**
- **Knapsack 2**
- **0/1**
- **Book Shop**
- **+ or -**

4 Coin Problems

Such combination-based problems have the soul of 0/1 Knapsack problem. They essentially ask what values we can conjure up by including or excluding different numbers of elements from a given set.

- **CSES**
- **CSES**
- **CSES**
- **CSES**
- **CSES**
- **Subset sum**

5 Meet in the Middle: (Not DP but useful)

Try this problem: Count subsets with a given sum
With dynamic programming this is a cakewalk.
Now try this problem: ***Subset Sum***
At first glance it may seem like the same problem, but submitting the same solution will lead you to a TLE!
When you look at the constraints, you would see that the target sum is of the order of $1e9$, which makes the same solution too slow despite n being $leq40$!
This is because the time complexity of the solution was $O(n * Target)$.
This basically tells us we need to think of this question from a new perspective.
We can use the fact that n is really small!

- Divide the array into 2 equal parts, first half and second half. Call them A and B.
- Find the set of all possible sums of each of the parts, call the sets X and Y.
- Find the number of subsets of A(B) with sum equal to each element of X(Y). Do this using bruteforce.
- The reason we can do this is because the time complexity for this is $O(2^n/2)$
- 2^{20} is a six digit number, so we don't have any problem.
- we couldn't brute-force our way through the entire array because 2^{20} is of order of $1e12$.
- Sort X
- For each element of Y, do binary search over X to find the suitable element such that the sum of the elements is = target.
- This algorithm is called Meet in the middle and has a time complexity of $O\left(n \cdot 2^{n/2}\right)$.

Do these problems to practice meet in the middle:

- **Last stone standing**
- **Subset Sum**
- **Largest possible subset sum *leq* Target**
- **4-Some**
- **Maximum Subsequence**

6 More 2-D DP array Problems:

- **LCS**
- **LIS**
- **Number of islands**

7 2-D Grid Problems:

Problems of this breed involve traversing over a grid of integers and either finding the value of a certain function, or optimizing a certain function.

- **Grid Paths-CSES**
- **Max Sum Along Grid Path**
- **1 Rectangle**
- **Max Rectangle Sum**
- **Swap**

8 Bitmask + DP

Not easy to understand and implement in problems for a new learner, but some really great problems are crafted using this concept. Bitmasking is basically when you have a set of objects and you have to work on its subsets, and each subset is denoted by a number in binary, with a '0' corresponding to exclusion of an object, and a '1' corresponding to the exclusion of an object. A mask is a binary number representing a particular subset, and a submask is a subset of that subset. A lot of problems involving bitmasking + DP are usually done by bottom-up approach:
Solving the problem for submasks and then building up 1 bit at a time. Here are some examples that illustrate this:

- **Elevator Rides**
- **O Matching**
- **Team Building**
- **Cat and Mice**
- **Make it 1**

9 Digit DP

This breed of problems usually revolves around doing a certain number of operations on a number or a set of digits, and finding the value of a function at the end of those operations.

- **Perfect Number - Codeforces**
- **Removing Digits-CSES**
- **IICPC POTD-5**
- **Counting Numbers-CSES**

10 IICPC DP-Week

The first week of IICPC Summer Camp 2024 was centered around Dynamic Programming. We compiled many study materials including live lectures from top competitive programmers, Problem of the Day, notes, and in the end, this handout.

- **DP Week - POTD**
- **Intro to DP - Vivek Gupta, ICPC World Finalist**
- **Intermediate DP concepts - Gaurish Baliga, Master, Codeforces**
- **Advanced DP Concepts - Gaurish Baliga**