Dynamic Programming

Gilles Englebert

April 3rd 2025

Overview

What is Dynamic Programming?

Dynamic Programming or **memoization** is the a way to speed up recursive computations (i.e. solving a problem in terms of one or more smaller/simpler instances of the same problem) by saving intermediate results to memory instead of doing the same computation multiple times. It is a **tradeoff** between memory and computation, and is most useful when sub-problems **overlap**.

Statement

The **Fibonacci sequence** $F_n(n \geq 0)$ is defined as $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. Or in C++:

```
int fib (int n) {
        if (n <= 1) return n;
        return fib(n-1)+fib(n-2);
}
```

What is the runtime of this recursive approach? How many computations do we really perform to determine F_n?
The runtime is $O(n!)$ (where ! is the factorial $n! = n(n-1)(n-2)\ldots 2 \cdot 1$).
We only compute $n+1$ different values (F_0 up to F_n)

Solution

We can save to memory the intermediate computations. This is called the **top-down** approach.

```
int f[1000] = {0};
int fib(int n) {
        if (n <= 1) return n;
    if (f[n] != 0) return f[n];

        f[n] = fib(n-1)+fib(n-2);
        return f[n];
}
```

The new runtime is $O(n)$, but we also have to allocate $O(n)$ memory, which we did not have to before (kind of).

Better Solution

We can do the computation using only $O(1)$ additional memory, using a **bottom-up** approach.

```
int f[2];
f[0] = 0;
f[1] = 1;
int fib(int n) {
for (int i = 2; i <= n; i++)
      f[i%2] += f[(i+1)%2];

  return f[n%2];
}
```

This has the downside of being considerably harder to implement, but it can be worth it sometimes (e.g. when the DP array is only sparsely populated).

Even better Solution

There is a very clever way do compute F_n in $O(\log n)$, using fast matrix exponentiation. I encourage you to look it up yourselves!

Statement

You are given n objects that have weight w_i units and value p_i respectively. You also have a bag (= knapsack) which can hold objects with a total weight of at most C units. What is the most expensive set of objects (i.e. highest total value) you can put in your bag?

Example

$n = 3$ and $C = 5$ with $w = \{3, 3, 5\}$, $p = \{5, 5, 6\}$.

Solution

We iterate through the objects one after another and go through any possible combinations of objects using recursive brute force.

```
int w[n]; //Weights
int p[n]; //Value/Price
int knap(int c, int i) {
        if (c < 0) return NEGATIVE_INFINITY;
        if (i > n) return 0;

        return max(knap(c, i+1),
            knap(c-w[i], i+1)+p[i]);
}
```

Runtime $O(2^n)$, memory $O(n)$.

Better solution

We realise that once we get to the $i-th$ object, all we need is $knap(\tilde{c}, it)$ for every $0 \leq \tilde{c} \leq C$.

```
int w[n]; //Weights
int p[n]; //Value/Price
int dp[C][n] = {-1};
int knap(int c, int i) {
        if (c < 0) return NEGATIVE_INFINITY;
        if (i > n) return 0;

        if (dp[c][i] == -1)
          dp[c][i] = max(knap(c, i+1),
            knap(c-w[i], i+1) +p[i]);
        return dp[c][i]; }
```

Runtime $O(nC)$, memory $O(nC)$. We traded more memory for a faster execution.

Statement

Problem

Given an array $a[1 \dots n]$ of n integers, compute the length of the longest strictly decreasing subsequence $\mathrm{LDS}(a)$ *of a.*

Example

Let $a[1 \dots 6] = \{5, 3, 4, 3, 3, 1\}$. Then $\mathrm{LDS}(a) = 4$.

Solution in $O(n^2)$

Solution

Let $dp[i]$ denote the maximal length of a decreasing subsequence ending at index i. Then

$$dp[i+1] = \max\{1\} \cup \{dp[j] + 1 : a[j] > a[i], 1 \leq j \leq i \leq n\}$$

Then $LDS(a) = \max_{1 \leq i \leq n} dp[i]$.

Editing Distance

Problem

Let s, t be two strings of length m and n respectively. The **editing distance** $\text{Edit}(s, t)$ *between s and t is the minimum number of replacements, deletions and additions of single characters to be performed on s to obtain t. Compute* $\text{Edit}(s, t)$ *efficiently.*

Example

Let $s =$ "*kitten*" and $t =$ "*sitting*", then $\text{Edit}(s, t) = 3$.

The editing distance is important in genetics, where it is used as a measure of similarity between two segments of DNA.

Traveling Salesman

Problem

Given n cities with pairwise distances $d[1 \ldots n][1 \ldots n]$, what is the minimal cost of completing a a loop which visits every city exactly once?

Problems

Here are some further problems (clickable links!):
LIO 2023 - Traffic Control
LIO 2020 - Packaging
LIO 2019 - Peche
LIO 2016 - Champignons

References

And some references (also clickable!)
Competitive Programming book
USACO
Also, if you have any questions, don't hesitate to write me at gilles@englebert.lu.