4. Dynamic Programming

Angel Napa

September 16, 2022

Dynamic programming

When Greedy Fails

- Given sufficiently 1,6,7 S/. coins
- Device a method to pay the amount M using the least number of coins

When Greedy Fails

- Greedy?
- No
- Counterexample: $n = 12$
- Greedy tells us to choose 6 coins $7 + 1 + 1 + 1 + 1 + 1$
- But optimal choice is 2 coins $6 + 6$
- Brute Force? Yes but inneficient.
- Dynamic Programming is efficient brute force.

What is dynamic programming?

- Divide and conquer recap:
  - Divide the problem into *independent* subproblems and solve it recursively.
  - Merge the subproblems into the original problem
- Dynamic programming:
  - Split the problem into *overlapping* subproblems
  - Combine the solutions to subproblems into a solution for the given problem
  - *Don't compute the answer to the same subproblem more than once*

Dynamic programming Recursive pseudocode

1. Formulate the original problem in terms of it smaller versions
2. Solve this using recursion
3. Memoize the function (save previous results to avoid unnecessary work)

The Fibonacci sequence

Given that $F_1 = 1, F_2 = 1$ and $F_{n+1} = F_n + F_{n-1}\ \forall n \geq 2$,
Compute F_n

$$\begin{aligned}
\text{fibonacci}(1) &= 1\\
\text{fibonacci}(2) &= 1\\
\text{fibonacci}(n) &= \text{fibonacci}(n-2) + \text{fibonacci}(n-1)
\end{aligned}$$

The Fibonacci Sequence

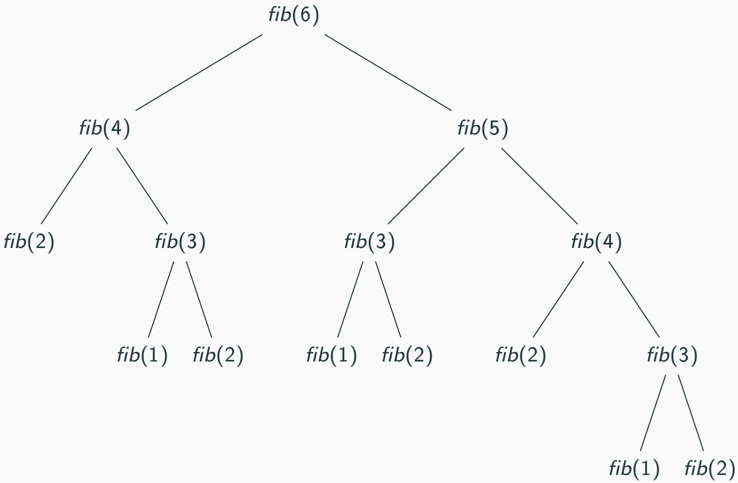
```
//int canti=0;
ll fibonacci(int n) {
    //canti++;cout<<canti<<"\n";
    if (n <= 2) {
        return 1;
    }
    ll res = fibonacci(n-2) + fibonacci(n-1);
    return res;
}

int main() {
    cout << fibonacci(50) << endl;

    return 0;
}
```

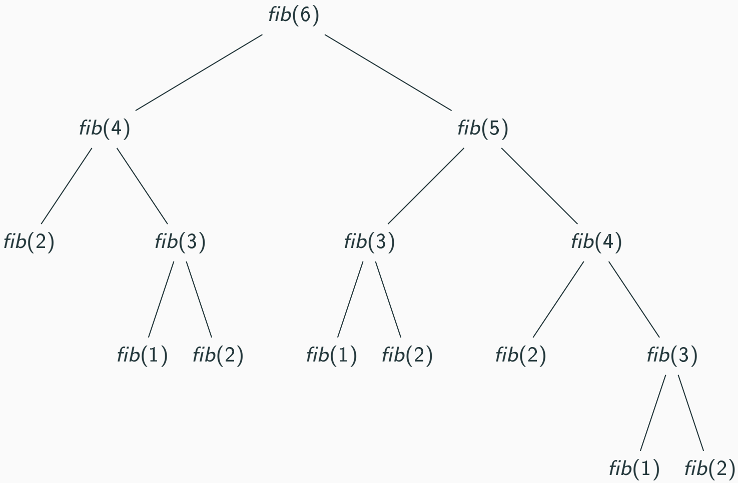
The Fibonacci sequence

- What is the time complexity of this?



The Fibonacci sequence

- What is the time complexity of this? Exponential, almost $O(2^n)$



The Fibonacci Sequence

```
ll memory[1000];
ll fibonacci(int n) {
    if (n <= 2) {
        return 1;
    }
    if (memory[n] != -1) {
        return memory[n];
    }
    ll res = fibonacci(n-2) + fibonacci(n-1);
    memory[n] = res;
    return res;
}

int main() {
    memset(memory, -1, sizeof(memory));
    cout << fibonacci(500) << endl;

    return 0;
```

The Fibonacci sequence

- What is the time complexity now?
- For each of the n inputs the result wil either:
  - be returned from memory
  - be computed, and the result saved
- Each saved input was computed exactly once (memoization)
- Total time complexity is $O(n)$

Coin change

- Given an array of coin c_0, c_1, $\ldots$, c_{n-1}, and some amount M: Compute minimum number of coins to represent the value of M

Coin change

- Let $\mathrm{opt}(i, M)$ denote the minimum number of coins needed to reach the value M using only sufficient c_0, ..., c_i coins

- Base case: $\mathrm{opt}(i, x) = \infty$ if $x < 0$
- Base case: $\mathrm{opt}(i, 0) = 0$
- Base case: $\mathrm{opt}(-1, x) = \infty$

- $\mathrm{opt}(i, x) = \min \begin{cases} 1 + \mathrm{opt}(i, x - d_i) \\ \mathrm{opt}(i - 1, x) \end{cases}$

Coin Change

```
int INF = 100000;
int d[10];

int opt(int i, int x) {
    if (x < 0) return INF;
    if (x == 0) return 0;
    if (i == -1) return INF;

    int res = INF;
    res = min(res, 1 + opt(i, x - d[i]));
    res = min(res, opt(i - 1, x));

    return res;
}
```

Coin Change

```
int INF = 100000;
int d[10];
int mem[10][10000];
memset(mem, -1, sizeof(mem));

int opt(int i, int x) {
    if (x < 0) return INF;
    if (x == 0) return 0;
    if (i == -1) return INF;

    if (mem[i][x] != -1) return mem[i][x];

    int res = INF;
    res = min(res, 1 + opt(i, x - d[i]));
    res = min(res, opt(i - 1, x));

    mem[i][x] = res;
    return res;
```

Coin Change

$$
\begin{aligned}
\texttt{solve}(0) &= 0\\
\texttt{solve}(1) &= 1\\
\texttt{solve}(2) &= 2\\
\texttt{solve}(3) &= 1\\
\texttt{solve}(4) &= 1\\
\texttt{solve}(5) &= 2\\
\texttt{solve}(6) &= 2\\
\texttt{solve}(7) &= 2\\
\texttt{solve}(8) &= 2\\
\texttt{solve}(9) &= 3\\
\texttt{solve}(10) &= 3
\end{aligned}
$$

Coin Change

$$\begin{aligned}\texttt{solve}(x) = \min(&\texttt{solve}(x-1)+1,\\ &\texttt{solve}(x-3)+1,\\ &\texttt{solve}(x-4)+1).\end{aligned}$$

$$\texttt{solve}(10) = \texttt{solve}(7)+1 = \texttt{solve}(4)+2 = \texttt{solve}(0)+3 = 3.$$

Coin Change

$$\texttt{solve}(x) = \begin{cases} \infty & x < 0 \\ 0 & x = 0 \\ \min_{c \in \texttt{coins}} \texttt{solve}(x - c) + 1 & x > 0 \end{cases}$$

Coin change

- Time complexity?
- Each input of the $n \times M$ matrix is calculated exactly once.
- Total time complexity is $O(n \times M)$

Coin change

- Recursive version: Top-Bottom DP
- Iterative version: Botton-Top DP

Coin change

```
value[0] = 0;
for (int x = 1; x <= n; x++) {
    value[x] = INF;
    for (auto c : coins) {
        if (x-c >= 0) {
            value[x] = min(value[x], value[x-c]+1);
        }
    }
}
```

Basic DP problems

- Coin distribution Problem
- Longest Common Subsequence (LCS)
- Longest Increasing Subsequence (LIS)
- Knapsack problem
- Edit Distance

Subsequence

- Given an array $a[n]$
- An array $b[n]$ is a subsequence of a if you can obtain it by removing elements from a.

Longest Common Subsecuence

- Given 2 arrays of integers $a[n]$ and $b[m]$
- Find the length of the largest common subsequence

Longest Increasing Subsecuence

- Given an array $a[0], a[1], \ldots a[n-1]$ integers
- Find the length of the largest increasing subsequence

Knapsack Problem

- Given a knapsack(a bag) that support a weight W
- Given n objects o_i that have weight w_i and value v_i (positive values and weights)
- Find the maximun value that you can put in the knapsack without break it?

Some Problems

- Kattis: knapsack, solitaire
- Spoj: knapsack, scubadiv
- Codeforces: 180C, 456C, 166E, 698A, 1196D1, 1196D2, 607A, 106C
- Atcoder: https://atcoder.jp/contests/dp

Some Problems

Muchas Gracias!