The background is a dark navy blue. In the top-left corner, there are two overlapping triangles: a blue one on the left and a light green one on the right. In the top-right corner, there is a grey, 3D-rendered circuit board pattern. In the bottom-left corner, there is a circular, grayscale image of a printed circuit board (PCB) with various electronic components. The title 'Dynamic Programming (DP)' is written in a large, white, sans-serif font in the center-right area.

Dynamic Programming (DP)

Wally



Dynamic Programming (From Wiki)

Dynamic programming is both a mathematical optimization method and a computer programming method. In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively. Likewise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have optimal substructure.



A Practical Example: Currency

Typical currency: { \$1, \$2, \$5, \$10, \$20, \$50, \$100}:

$$\$16 = \$10 + \$5 + \$1$$

$$\$174 = \$100 + \$50 + \$20 + \$2 + \$2$$



A Practical Example: Currency

A different currency: { \$1, \$3, \$4, \$10, \$30, \$40, \$100 }



Issue? How to get \$6, \$60?

$$\$6 = \$4 + \$1 + \$1?$$

$$\text{Or } \$6 = \$3 + \$3?$$

$$\$60 = \$40 + \$10 + \$10?$$

$$\text{Or } \$60 = \$30 + \$30?$$




Dynamic Programming

- Optimize Global Solution

$$P(0) = 0$$

$$P(1) = 1 (\$1)$$

$$P(2) = 2 (\$1 + \$1)$$

$$P(3) = 1 (\min(\$3, 1 + P(2)))$$

-- Derive a solution from sub problems



Pseudocode for Currency Problem

```
array<int> bills = { 1, 3, 4 };
```

```
int minBills(int dollars) {
```

```
    if (dollars == 0) return 0;
```

```
    else if (dollars == 1) return 1;
```

```
    else {
```

```
        return min(1 + minBills(dollars - bills[1]),
```

```
                    1 + minBills(dollars - bills[2]), ...
```

```
                    1 + minBills(dollars - bills[j]));
```


```
    } j -> largest where bills[j] <= dollars;
```



Dynamic Programming

- Memoization

- Create an array of $P[0 \dots n-1]$ represent the problem and subproblems
- Create an array of bool $hasValue[0 \dots n-1]$ to represent whether $P[i]$ is solved
- If $hasValue[i]$ is true, use $P[i]$, else set $P[i]$



Pseudocode for Currency Problem V2.0

```
array<int> bills = { 1, 3, 4 };
```

```
array<int> nBills = { 0, 1, ... };
```

```
array<bool> solved = { true, true, false, ... false };
```

```
int minBills(int dollars) {
```

```
    for i = 2 ... dollars:
```

```
        nBills[i] = min(1 + nBills[0],
```

```
                        1 + nBills[1],
```

```
                        ..., 1 + nBills[j]));
```

```
    return nBills[dollars];
```

```
} j -> largest where bills[j] <= dollars;
```



<https://open.kattis.com/problems/canonical>