1 Method for Dynamic Programming

1. UNDERSTAND THE PROBLEM
2. Write a BT algorithm to solve the problem
3. Identify Repeated Subproblems
4. Name these subproblems and create an array to hold them
5. Find Top-Down order
6. Convert to Bottom-Up

Then to write the dynamic programming algorithm:

1. Init arrays
2. fill in base cases in array
3. solve subproblems by building upon itself
4. return solution to desired subproblem (typically just the element sitting in some array position)

2 Zoo Benches Problem

2.1 Problem Description

2.1.1 High-Level

- We are brought in to help the Zoo
- Have a typical path people follow to go through the zoo
- Lots of old people visit the zoo and can’t walk far without resting
- Thus we need to put benches every so often on the path
• Not all spots along the path are created equal. Due to topography and access difficulties, it costs more money to put benches in certain places.

• Thus we need the benches so they are never too far apart, but we want to minimize the cost of putting them in.

2.1.2 Low-Level

• Input: Given array of costs $C[1..n]$. Imagine the path through the zoo is $n$ exhibits long, and $C[i]$ is the cost of putting a bench at exhibit $i$. Also given maximum separation $k$.

• Output and constraint: A set of indices $i_1 < ... < i_m$ such that $i_j - i_{j-1} \leq k$.

• Objective: Minimize the cost $\sum_j C[i_j]$.

2.2 Solving the problem

2.2.1 Understand problem

Important. Do not move on until you do.

2.2.2 Develop A Backtracking Algorithm

Algorithm BTBC($C[1..n], k$)

if $n < k$ then
    return 0
else
    $C_{\text{min}} \leftarrow \infty$
    for $i = 1$ to $k$:
        $C_{\text{min}} \leftarrow \min \left\{ C_{\text{min}}, C[i] + \text{BTBC}(C[i+1..n], k) \right\}$
    return $C_{\text{min}}$

The idea is we can put a bench in front of exhibits 1 to $k$, and then when we place a bench at exhibit $i$, we need to find the most cost effective way to put benches on the rest of the path (exhibits $i+1$ to $n$).

This algorithm is very slow

$T(n) = T(n-1) + T(n-2) + ... + T(n-k) \leq kT(n-1) \leq k^n$

2.2.3 Identify Repeated Subproblems

Notice that we only call BTBC for different first indices of the array $C$. Thus there are really only subproblems for each possible first index, i.e. $C[j, n]$. 
2.2.4 Name Subproblems
Since only first index changes, we only need 1-D array. Let’s call it $A$ and let
$$A[j] = C[j..n]$$

2.2.5 Find ordering
- We see that our backtracking algorithms makes recursive calls with larger $j$ values. This means that $C[1..n]$ depends on $C[2..n]$ and maybe $C[3..n]$, etc.
- We also want to reinterpret our base cases with this new form.
- Why did we quit in our base case? Think of it as when we call the algorithm on $C[1..n]$ there is a bench at 0, and since the next index cannot be more than $k$ greater than 0, this is the same as saying $n < 0 + k$ or $n < 1 - 1 + k$.
- Now if we replace 1 with $j$, this becomes $n < j - 1 + k$, so this is our new base case.

We can then rewrite the dynamic programming algorithm in these terms...

Algorithm BTBC($C[j..n], k$)
if $n < j - 1 + k$ then  // Note can be rewritten $n + 2 - k \leq j$. This will be useful later.
    return 0
else
    $C_{min} \leftarrow \infty$
    for $i = j$ to $j + k - 1$:
        $C_{min} \leftarrow min\left\{ C_{min}, C[i] + BTBC(C[i+1..n], k) \right\}$
    return $C_{min}$

This for loop represents putting a bench at position $i$ (between $j$ and $j + k - 1$), and then recursively calling from the next available position ($i + 1$).

2.2.6 Converting to DP by inverting
Now that we’ve got our top-down backtracking algorithm, we convert it to DP, following the usual four steps (they are numbered inside the algorithm below).

Algorithm DPBC($C[1..n], k$)
// 1. Initialize Array
    $A[1..n+1]$
// 2. Fill in Base Case/s
    for $j = n + 2 - k$ to $n + 1$
        $A[j] \leftarrow 0$
// 3. In Bottom-up order solve subproblems
for \( j = n + 1 - k \) down to 1
    \( A[j] \leftarrow \infty \)
for \( i = j \) to \( j + k - 1 \)
    \( A[j] \leftarrow \min \left\{ A[j], C[i] + A[i + 1] \right\} \)

// 4. Return solution to final subproblem
return \( A[1] \)

2.3 Time Analysis?

Base cases takes about time \( O(k) \). Main for loop at most about \( O(n) \), and inside loops \( O(k) \) times. Thus total time is \( O(nk) \).

How much space? Single 1-D array of size \( n + 1 \) so \( O(n) \) space.

2.4 Finding the positions

The Dynamic programming algorithm above is great, but it only finds the minimum cost, not the actual positions where the zoo should put the benches to achieve the minimum cost. Luckily we just need to make a small modification:

Algorithm DPBC2(\( C[1..n], k \))
    \( A[1..n + 1] \) for \( j = n + 2 - k \) to \( n + 1 \)
    \( A[j] \leftarrow 0 \)
for \( j = n + 1 - k \) down to 1
    \( A[j] \leftarrow \infty \)
    \( Next[j] \leftarrow X \) // \( X \) represents a null value
for \( i = j \) to \( j + k - 1 \)
        \( Next[j] \leftarrow i \) // The best position thus far is at \( i \)
    \( A[j] \leftarrow C[i] + A[i + 1] \)
return \( A[1] \)

2.5 An Example

To help illustrate the above algorithm, here is an example.

Let \( k = 3 \),
<table>
<thead>
<tr>
<th>Exhibit</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Next</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>10</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

From this example, we look at $A[1]$ and see that the minimum cost is 9. The Next array tells us where to put the next bench. One important point is that we look at $Next[1]$ and see the first bench should be put in front of exhibit 3, but then to find the position for the next bench, we need to look at $Next[4]$ since after putting a bench at 3, the next subproblems is starting from exhibit 4.

So above, we see we should put benches in front of exhibits 3, 6, 7, and 10. ($Next[1]=3$, $Next[3+1]=6$, $Next[6+1]=7$, $Next[7+1]=10$, and $Next[10+1]=X$ so we’re done)