CSCI2100 Data Structures
Dynamic Programming

Irwin King

king@cse.cuhk.edu.hk
http://www.cse.cuhk.edu.hk/~king

Department of Computer Science & Engineering
The Chinese University of Hong Kong

Outline

- Coin Changing Problem
- Longest Common Subsequence

Resources:
http://www.cs.uni.edu/~fienup/cs188s05/lectures/lec6_1-27-05.htm
http://interactivepython.org/runestone/static/pythonds/Recursion/DynamicProgramming.html
http://web.stanford.edu/class/cs161/
http://www.cs.cmu.edu/afs/cs/academic/class/15451-s15/

Coin Changing Problem

- Input:
  - k coins denominations, $1 = d_1 < d_2 < \cdots < d_k$
  - A positive integer n
- Output:
  - The minimum number of coins that changes n
- Example:
  - Making 40 cents change with coin types {1, 5, 10, 25, 50}
  - The optimal solution takes 3 coins (25+10+5)

Greedy Algorithm

- At each iteration, add coin of the largest value that does not take us pass the amount to be paid
- Example:
  - Making 40 cents change with coin types {1, 5, 10, 25, 50}
  - {25} //40-25=15 cents left
  - {25, 10} //15-10=5 cents left
  - {25, 10, 5} //5-5=0 cents left
  - Return 3
- Does it always give optimal solution?

Greedy Algorithm

- Counterexample:
  - Now we have 20-cent coins
  - Making 40 cents change with coin types {1, 5, 10, 20, 25, 50}
  - {25} //40-25=15 cents left
  - {25, 10} //15-10=5 cents left
  - {25, 10, 5} //5-5=0 cents left
  - Return 3
- But the optimal solution should be 2 coins (20+20)
- Greedy algorithm does not guarantee optimality

Coin Changing Problem

- Coin changing problem has optimal substructure
  - Optimal solutions to sub-problems are sub-solutions to the optimal solution of the original problem

$$\mathrm{MinNumCoins}(n) = \min \begin{cases} \mathrm{MinNumCoins}(n-d_1)+1 \\ \mathrm{MinNumCoins}(n-d_2)+1 \\ \dots \\ \mathrm{MinNumCoins}(n-d_k)+1 \end{cases}$$

Can we simply use a recursive function to solve it?

Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

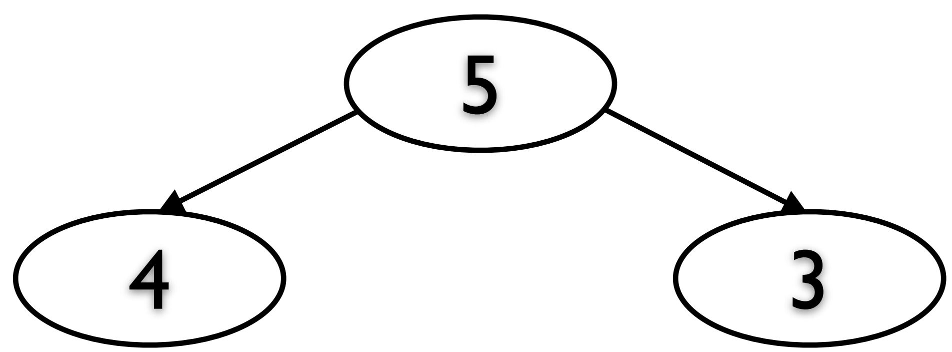
$$\text{MinNumCoins}(5) = \min \begin{cases} \text{MinNumCoins}(5-1) + 1 \\ \text{MinNumCoins}(5-2) + 1 \end{cases}$$

5

Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

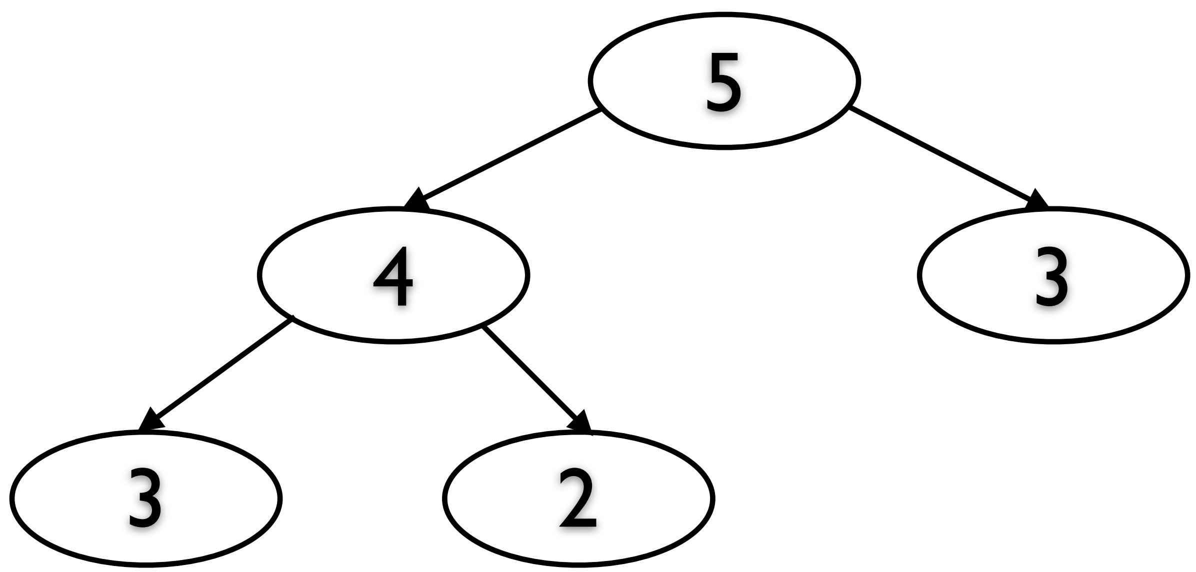
$$\mathrm{MinNumCoins}(5) = \min \begin{cases} \mathrm{MinNumCoins}(4) + 1 \\ \mathrm{MinNumCoins}(3) + 1 \end{cases}$$



Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

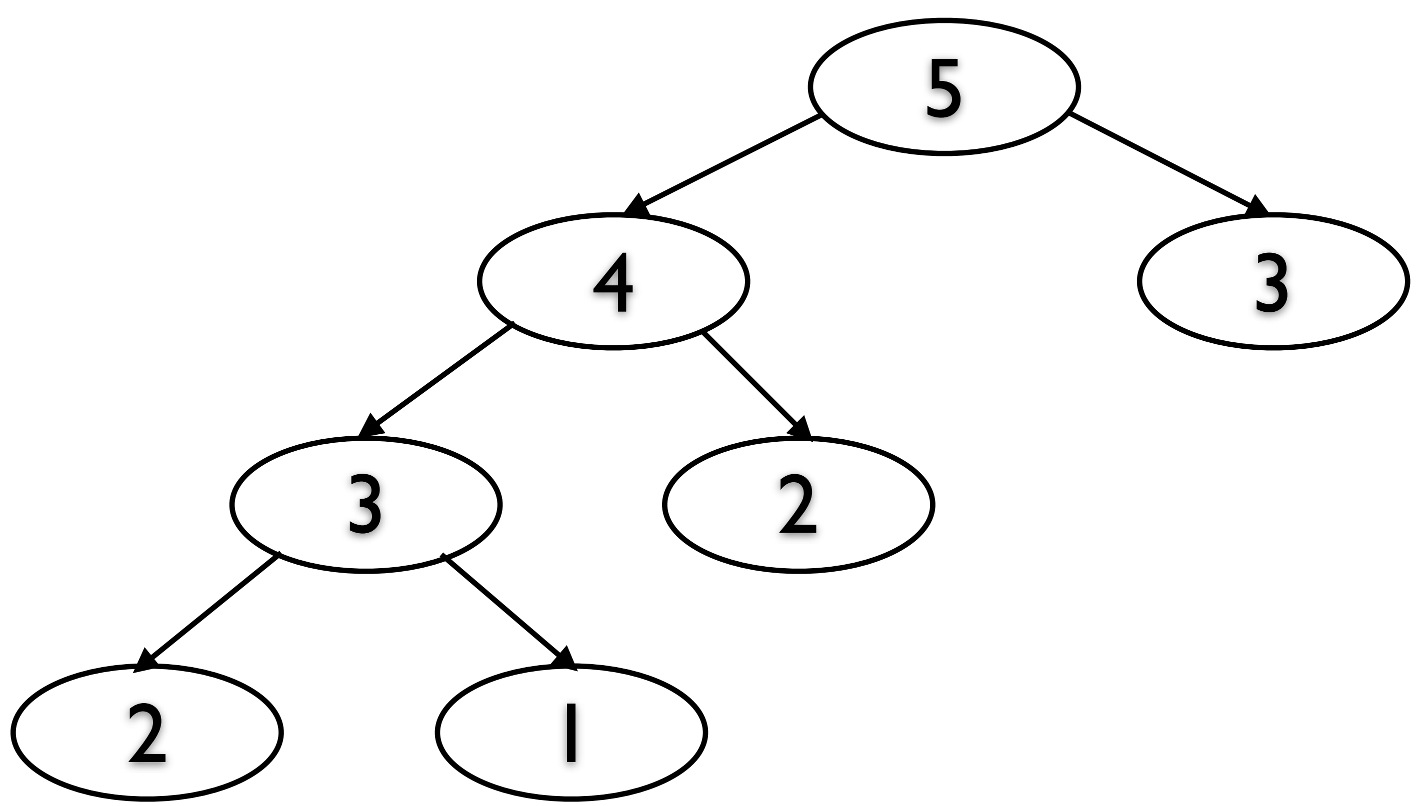
$$\text{MinNumCoins}(4) = \min \begin{cases} \text{MinNumCoins}(3) + 1 \\ \text{MinNumCoins}(2) + 1 \end{cases}$$



Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

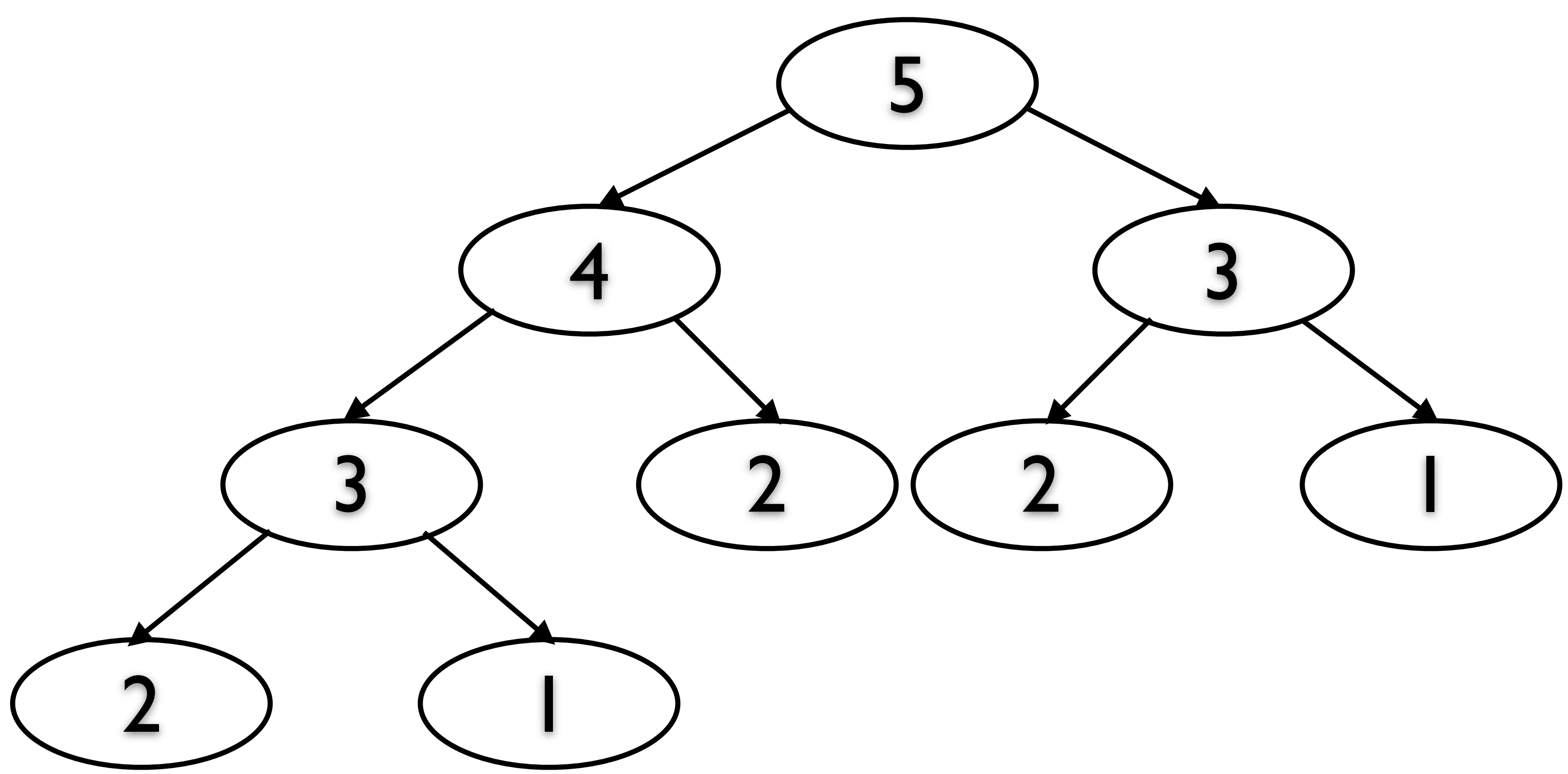
$$\text{MinNumCoins}(3) = \min \begin{cases} \text{MinNumCoins}(2) + 1 \\ \text{MinNumCoins}(1) + 1 \end{cases}$$



Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

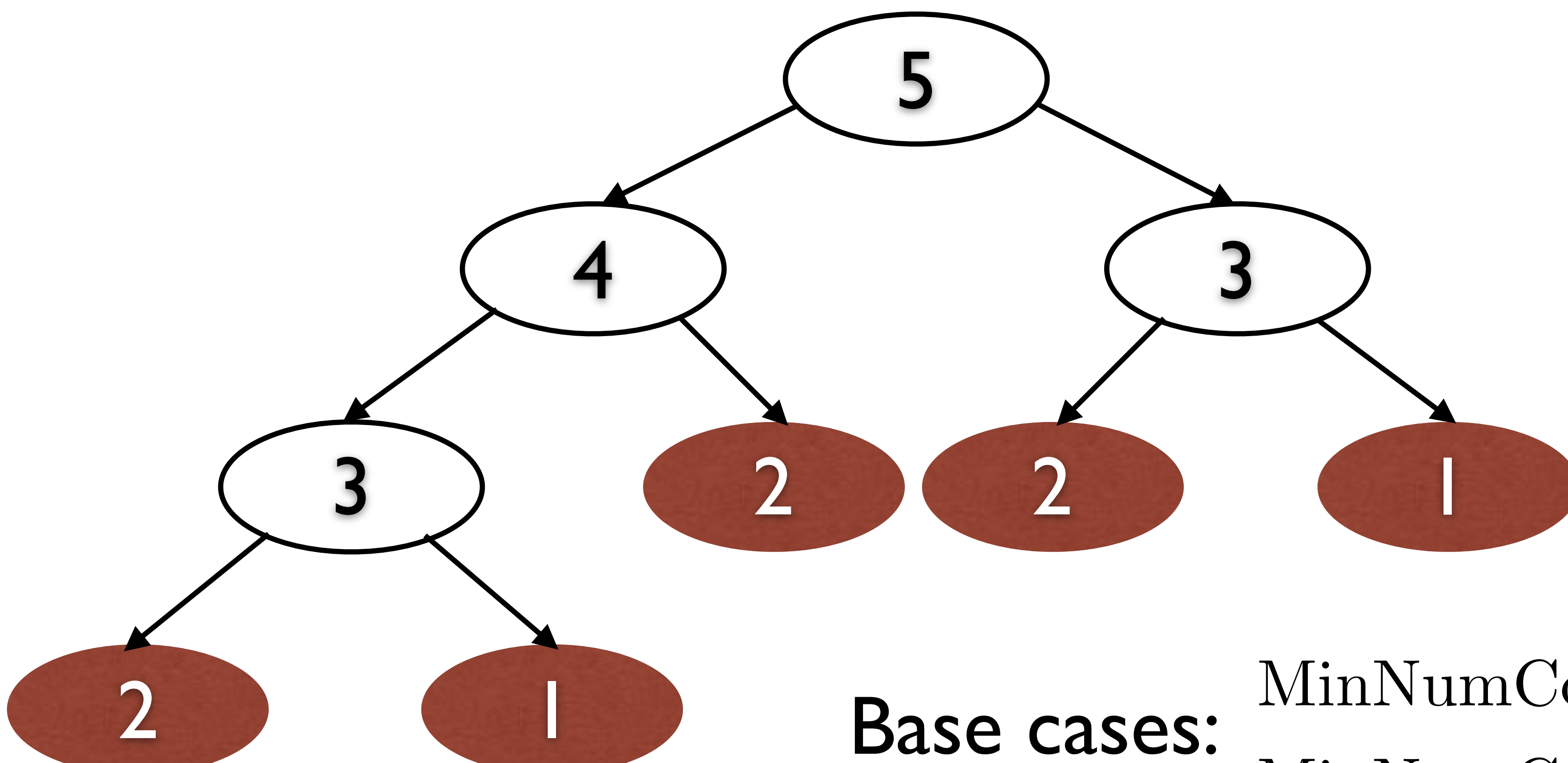
$$\text{MinNumCoins}(3) = \min \begin{cases} \text{MinNumCoins}(2) + 1 \\ \text{MinNumCoins}(1) + 1 \end{cases}$$



Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

$$\text{MinNumCoins}(3) = \min \begin{cases} \text{MinNumCoins}(2) + 1 \\ \text{MinNumCoins}(1) + 1 \end{cases}$$



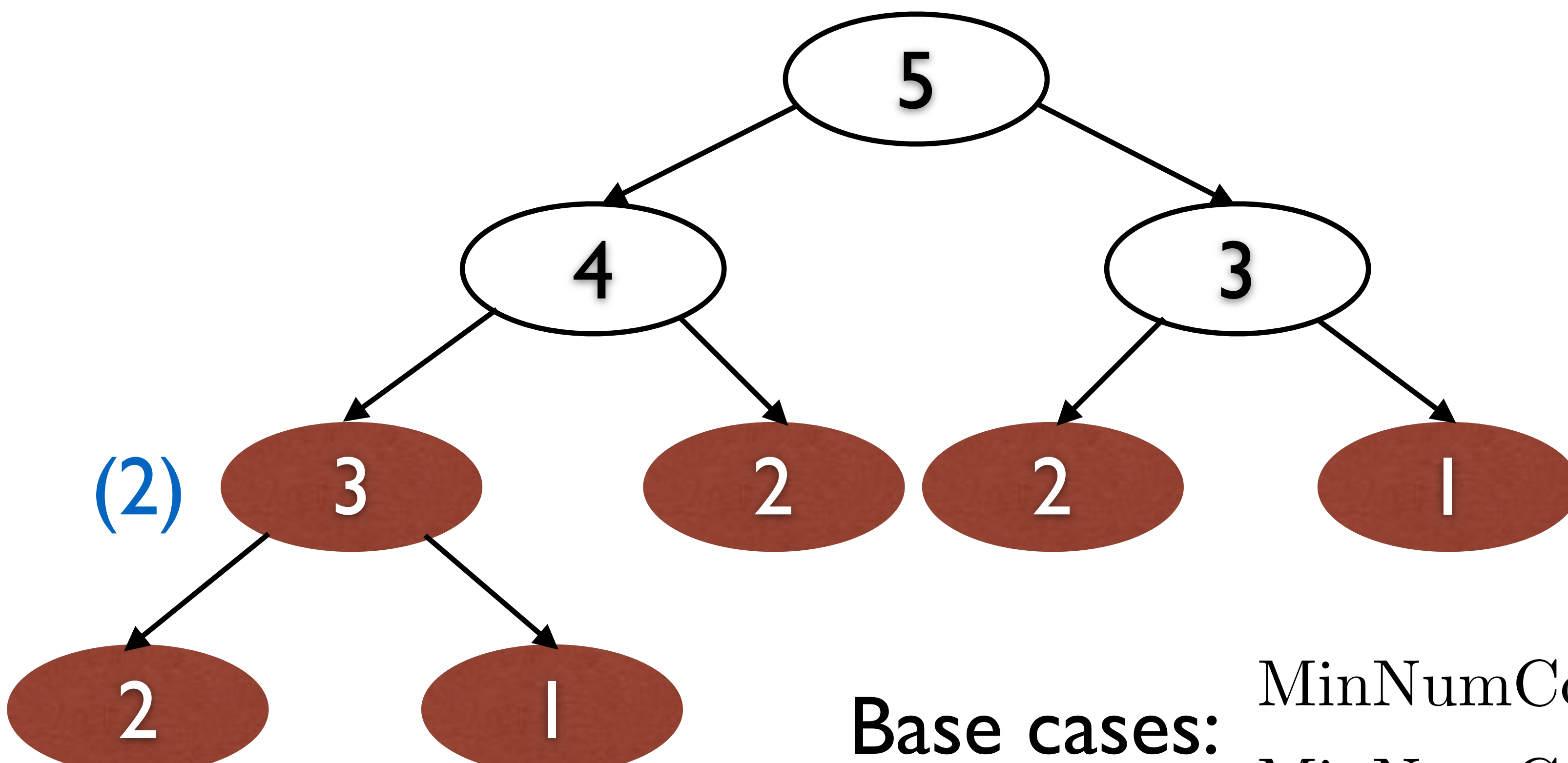
Base cases: $\text{MinNumCoins}(2) = 1$

$\text{MinNumCoins}(1) = 1$

Recursive Approach

- Example:
    - Making 5 cents change with coin types {1, 2}

$$\text{MinNumCoins}(3) = \min \begin{cases} 1+1 \\ 1+1 \end{cases} = 2$$

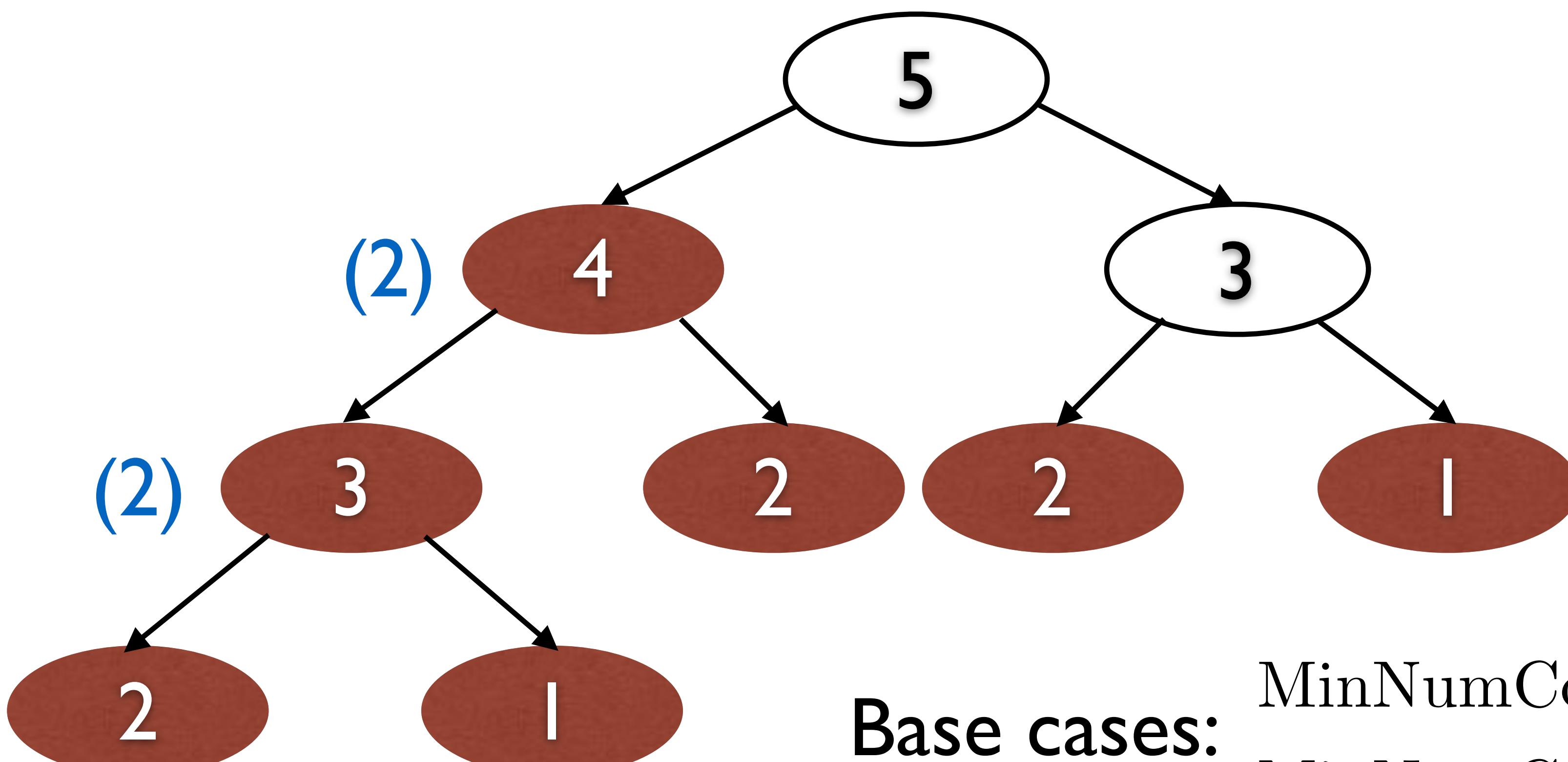


Base cases:
$$\text{MinNumCoins}(2) = 1$$
$$\text{MinNumCoins}(1) = 1$$

Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

$$\text{MinNumCoins}(4) = \min \begin{cases} 1+1 \\ 2+1 \end{cases} = 2$$

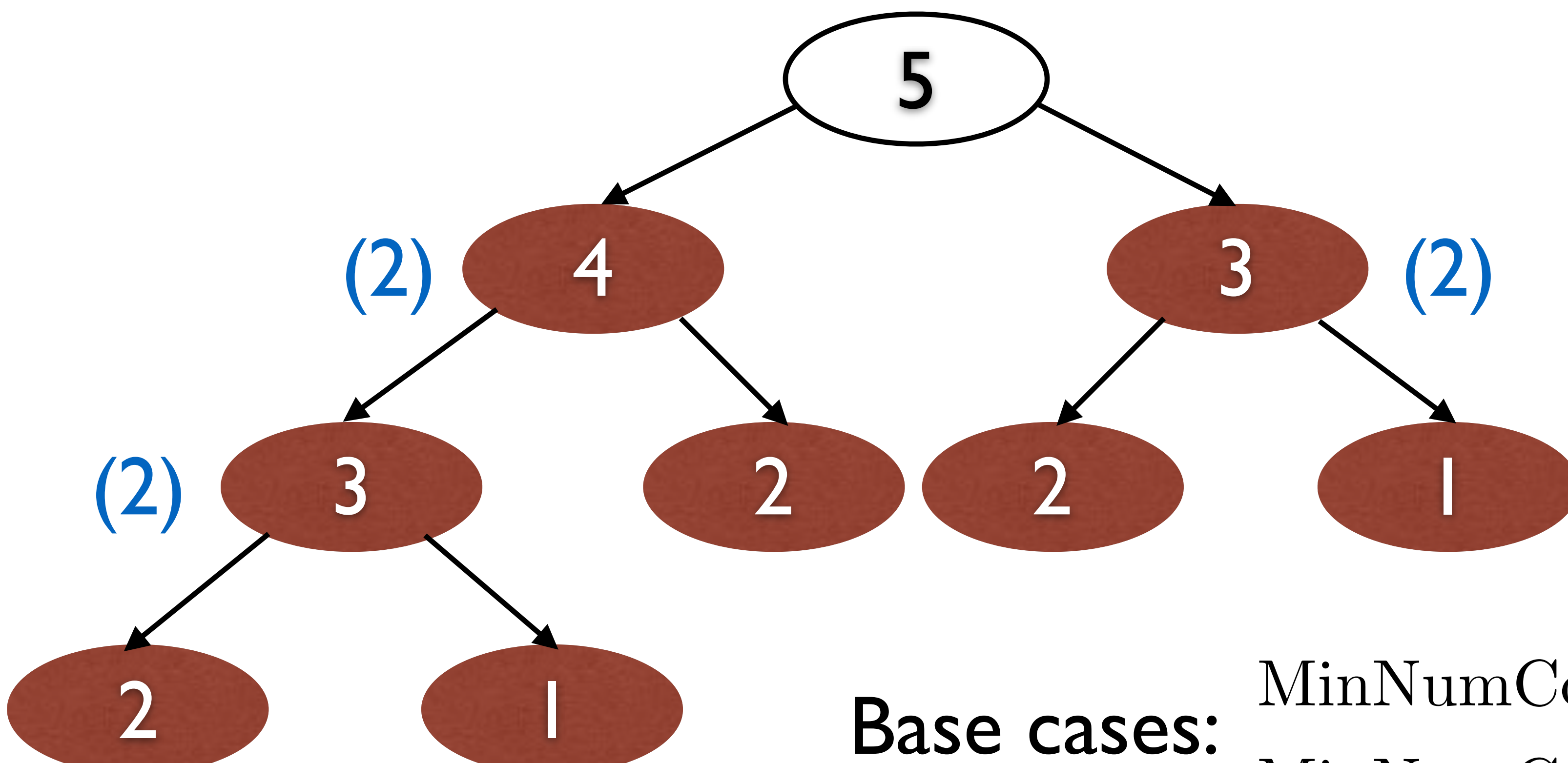


Base cases: $\text{MinNumCoins}(2) = 1$
$\text{MinNumCoins}(1) = 1$

Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

$$\text{MinNumCoins}(3) = \min \begin{cases} 1+1 \\ 1+1 \end{cases} = 2$$

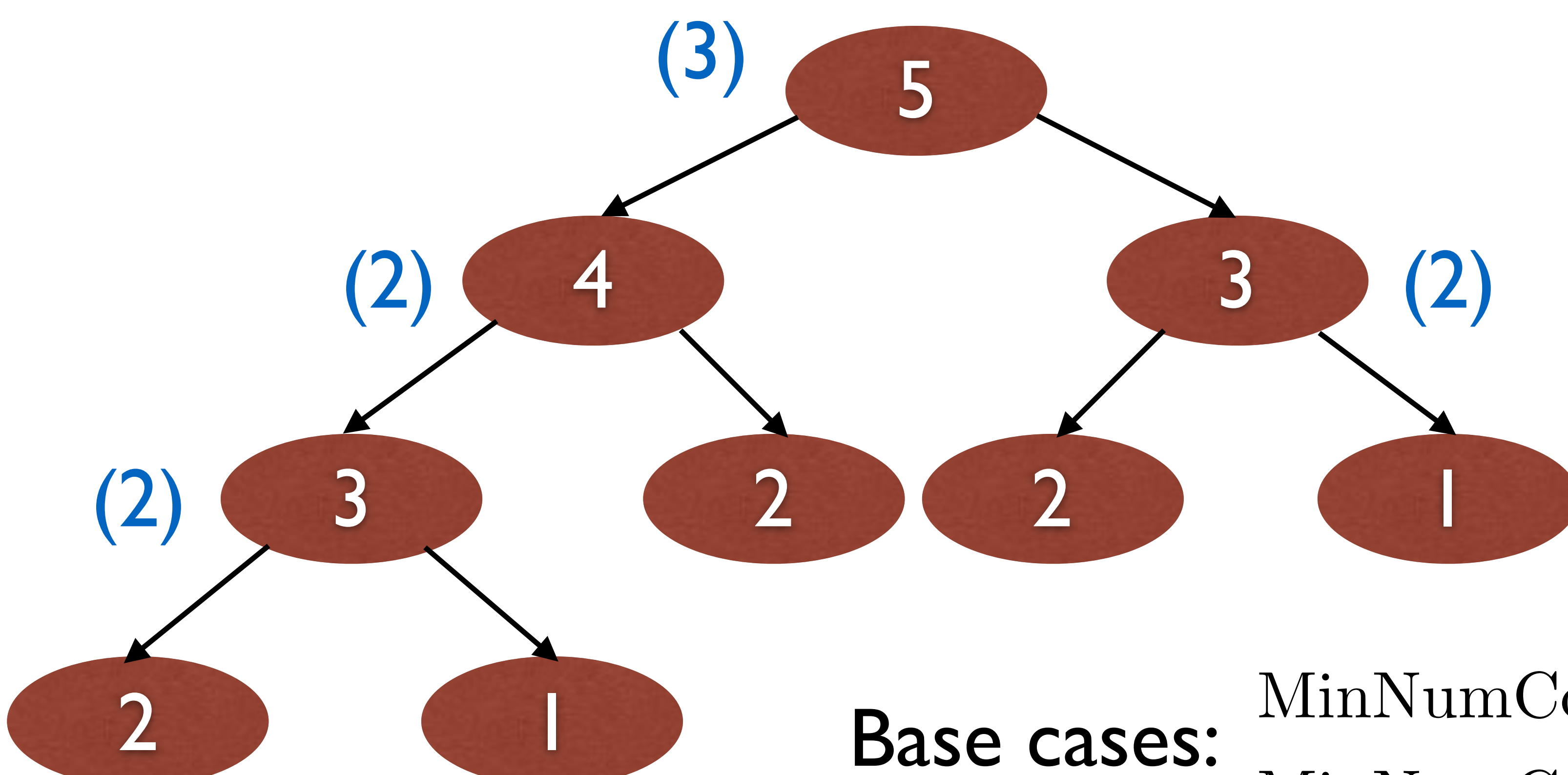


Base cases: $\text{MinNumCoins}(2) = 1$
$\text{MinNumCoins}(1) = 1$

Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}

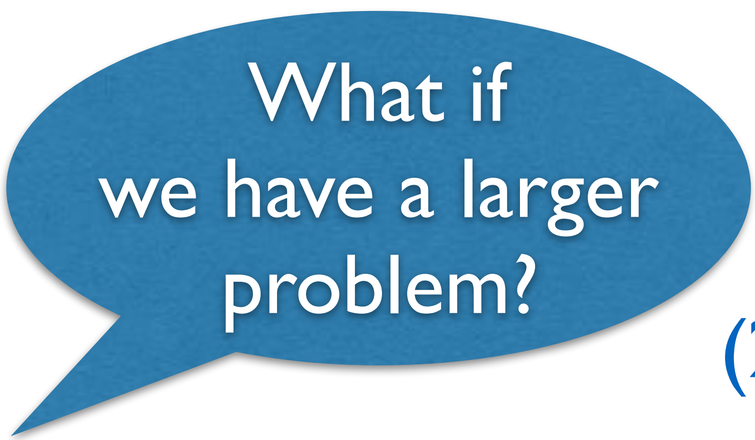
$$\text{MinNumCoins}(5) = \min \begin{cases} 2+1 \\ 2+1 \end{cases} = 3$$

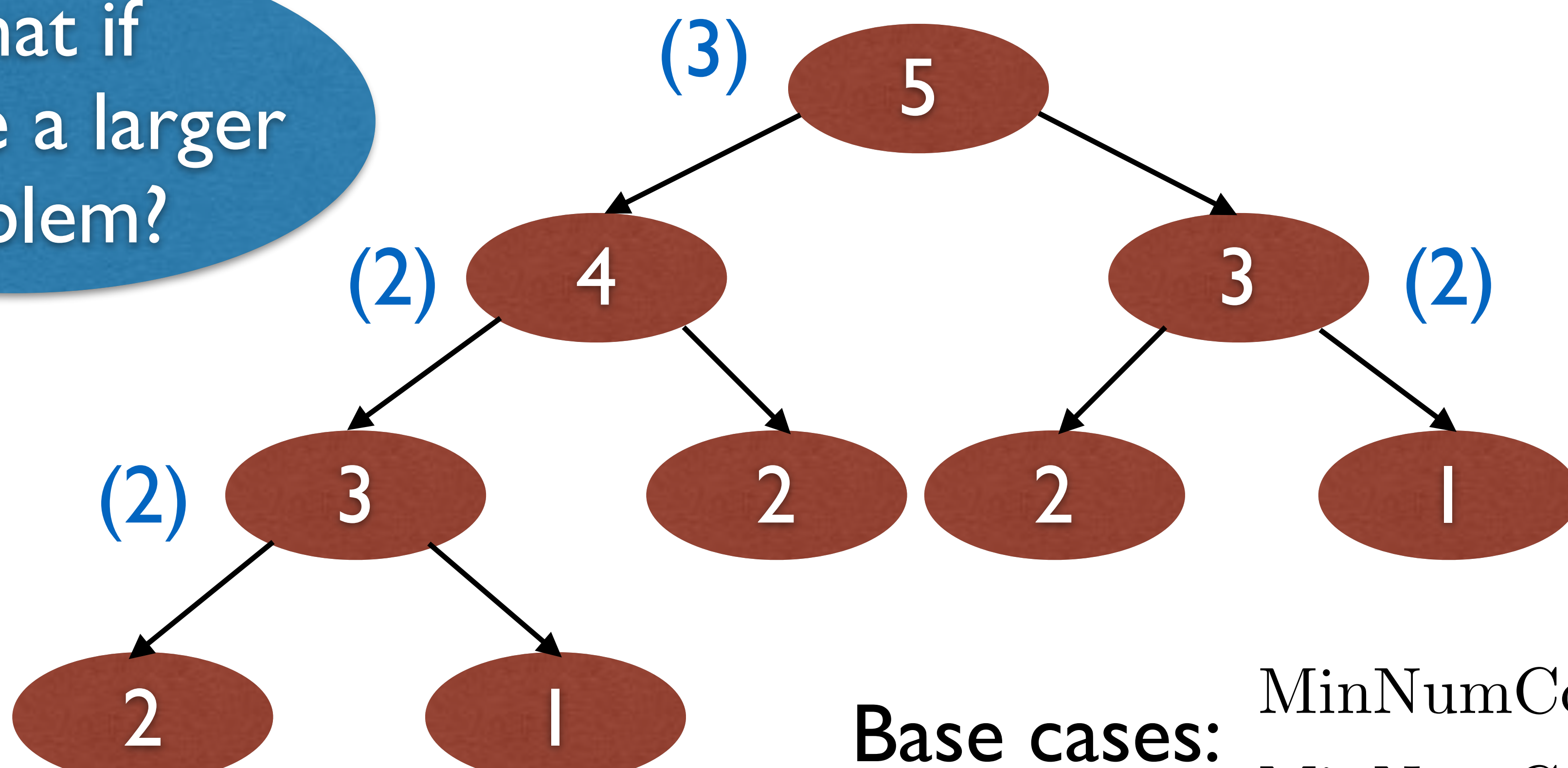


Base cases:
$$\text{MinNumCoins}(2) = 1$$
$$\text{MinNumCoins}(1) = 1$$

Recursive Approach

- Example:
  - Making 5 cents change with coin types {1, 2}
  - Output 3, which is the optimal solution



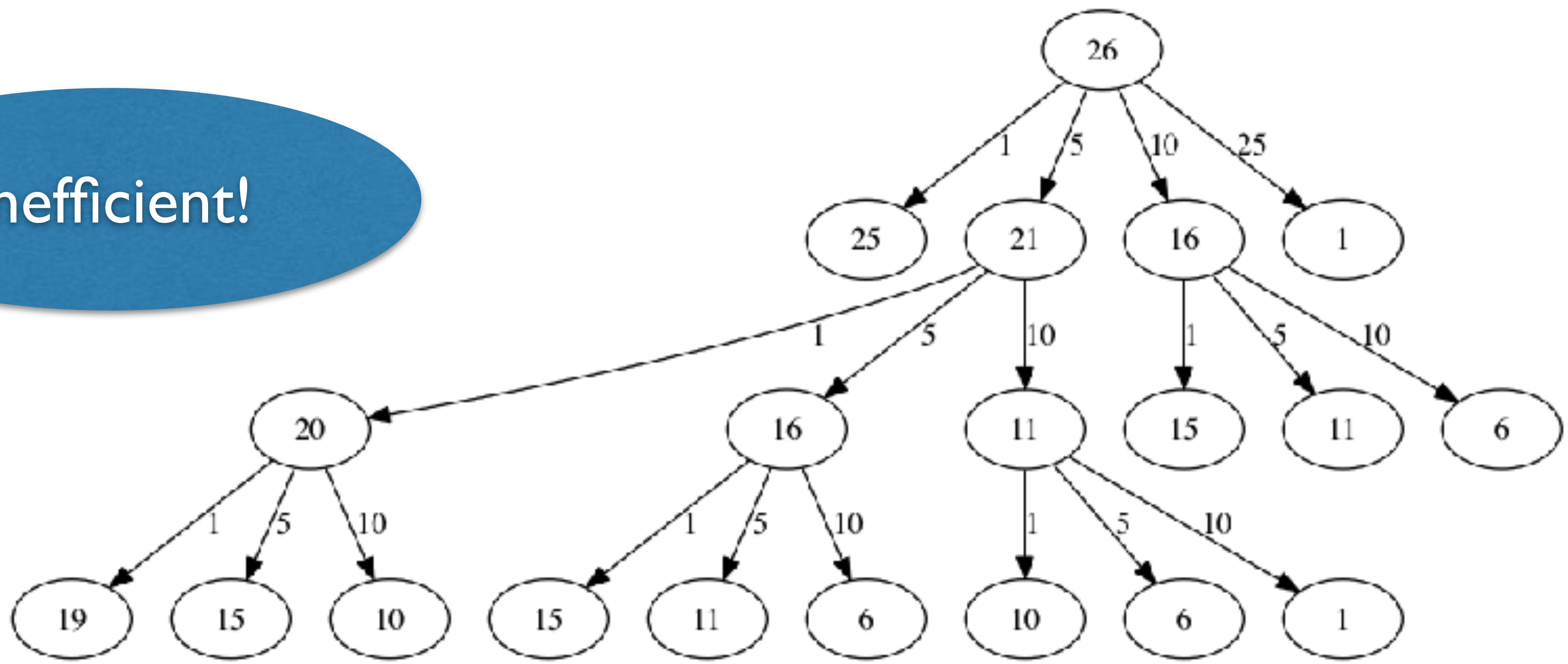


Base cases:
$$\text{MinNumCoins}(2) = 1$$
$$\text{MinNumCoins}(1) = 1$$

Recursive Approach

- Another Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

Inefficient!



- Sub-problems overlap a lot!
  - MinNumCoin(15) is computed at least 3 times!
  - Computing MinNumCoin(15) takes 52 function calls

Coin Changing Problem

- Two properties of coin changing problem
  - Optimal substructure
    - $$\text{MinNumCoins}(n) = \min \begin{cases} \text{MinNumCoins}(n - d_1) + 1 \\ \dots \\ \text{MinNumCoins}(n - d_k) + 1 \end{cases}$$
  - Overlapping sub-problems
    - Lots of different MinNumCoins(i) will use MinNumCoins(j)
- We should apply dynamic programming (DP) to reuse answers to sub-problems

Recipe of Applying DP

- Step 1: identify optimal substructure
  - We already have

$$\text{MinNumCoins}(n) = \min \begin{cases} \text{MinNumCoins}(n - d_1) + 1 \\ \dots \\ \text{MinNumCoins}(n - d_k) + 1 \end{cases}$$

- Step 2: devise a table lookup strategy
  - Solve smaller problems before larger ones
  - Look-up answers to smaller problems when solving larger subproblems

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0										

$\ldots$

$$\text{MinNumCoin}(0) = 0$$

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0	1									

$\ldots$

$$\begin{aligned}\text{MinNumCoins}(1) &= \text{MinNumCoins}(1-1)+1\\ &= \text{MinNumCoins}(0)+1\\ &= 1\end{aligned}$$

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0	1	2								

$\ldots$

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

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0	1	2	3							

$\ldots$

$$\begin{aligned}\text{MinNumCoins}(3) &= \text{MinNumCoins}(3-1)+1\\ &= \text{MinNumCoins}(2)+1\\ &= 3\end{aligned}$$

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
$\mathrm{MinNumCoin}$	0	1	2	3	4						

$\ldots$

$$\begin{aligned}\mathrm{MinNumCoins}(4) &= \mathrm{MinNumCoins}(4-1)+1\\ &= \mathrm{MinNumCoins}(3)+1\\ &= 4\end{aligned}$$

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0	1	2	3	4	1					

$\dots$

$$\text{MinNumCoins}(5) = \min \begin{cases} \text{MinNumCoins}(5-5)+1 \\ \text{MinNumCoins}(5-1)+1 \end{cases} = 1$$

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0	1	2	3	4	1	2				

$\ldots$

$$\text{MinNumCoins}(6) = \min \begin{cases} \text{MinNumCoins}(6-5)+1 \\ \text{MinNumCoins}(6-1)+1 \end{cases} = 2$$

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0	1	2	3	4	1	2	3	4	5	1

$\dots$

$$\mathrm{MinNumCoins}(10) = \min \begin{cases} \mathrm{MinNumCoins}(10-10)+1 \\ \mathrm{MinNumCoins}(10-5)+1 \\ \mathrm{MinNumCoins}(10-1)+1 \end{cases} = 1$$

Table Lookup Strategy

- Example:
  - Making 26 cents change with coin types {1, 5, 10, 25}

n	0	1	2	3	4	5	6	7	8	9	10
MinNumCoin	0	1	2	3	4	1	2	3	4	5	1

n	11	12	13	14	15	16	17	18	19	20	21
MinNumCoin	2	3	4	5	2	3	4	5	6	2	3

n	22	23	24	25	26						
MinNumCoin	4	5	6	1	2						

DP Algorithm

```
int MinNumCoin(int coins[], int num_coin_types, int n){
    int i, j;
    int table[n+1];
    table[0] = 0;
    // In the worst case, we need n coins to change n cents
    for (i=1; i<=n; i++)
        table[i] = i;
    // Compute minimum coins required for all values from 1 to n
    for (i=1; i<=n; i++){
        for (j=0; j<num_coin_types; j++)
            if (coins[j] <= i){
                int sub_res = table[i-coins[j]];
                if (sub_res + 1 < table[i])
                    table[i] = sub_res + 1;
            }
    }
    return table[n];
}
```

Complexity: *O(nk)*

Resources: https://www.geeksforgeeks.org/find-minimum-number-of-coins-that-make-a-change/

Longest Common Subsequence

- Input:
  - Two strings: X and Y
- Output:
  - The longest sequence of characters that appear left-to-right (but not necessarily in a contiguous block) in both X and Y
- Example:
  - X=ABCDEFGH and Y=ABDFGHI
  - Their longest common subsequence (LCS) is ABDFGH

Recipe of Applying DP

- Step 1: identify optimal substructure
- Step 2: devise a table lookup strategy

Recipe of Applying DP

- Step 1: identify optimal substructure
- Step 2: devise a table lookup strategy

Optimal Substructure

- Define prefix X_i as the first i consecutive characters in X
- Example: X=ACGGT, X_4=ACGG
- Subproblem:
  - Finding LCS of a prefix of X and a prefix of Y
  - e.g., LCS of X_i and Y_j

Optimal Substructure

- For simplicity, let's worry first about finding the length of the LCS
- Then modify the algorithm to produce the LCS
- Problem:
  - Find the length of LCS of *X* and *Y*
- Subproblem:
  - Find the length of LCS of prefixes to *X* and *Y*
  - Let C[i,j]=length_of_LCS(X_i,Y_j)

Optimal Substructure

- Case 1: If X[i]=Y[j], where X[i] is the i-th character in X

X	A	C	G	G	A
					X[i]

Y	A	C	G	C	T	T	A
							Y[j]

- Then C[i,j]=1+C[i-1,j-1]
- For example:
  - LCS(X_{i-1},Y_{j-1})=ACG
  - LCS(X_i,Y_j)=ACGA = LCS(X_{i-1},Y_{j-1}) followed by A

Optimal Substructure

- Case 2: If X[i]!=Y[j]

X	A	C	G	G	T
					X[i]

Y	A	C	G	C	T	T	A
							Y[j]

- Then C[i,j] = max{ C[i-1,j], C[i,j-1] }.
- For example:
  - Either LCS(X_i,Y_j) = LCS(X_{i-1},Y_j) and T is not involved,
  - Or LCS(X_i,Y_j) = LCS(X_i,Y_{j-1}) and A is not involved

Optimal Substructure

Base case:

X	A	C	G	G	T
Y					

- Recursive Formulation

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Case 1:

X	A	C	G	G	A		
Y	A	C	G	C	T	T	A

Case 2:

X	A	C	G	G	T		
Y	A	C	G	C	T	T	A

Recipe of Applying DP

- Step 1: identify optimal substructure

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

- Step 2: devise a table lookup strategy
  - Solve smaller problems before larger ones
  - Look-up answers to smaller problems when solving larger subproblems

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0					
1					
2					
3					
4					
5					

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0				
2	0				
3	0				
4	0				
5	0				

Base case!

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1			
2	0				
3	0				
4	0				
5	0				

$$C[1,1] = C[0,0] + 1 = 1$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1		
2	0				
3	0				
4	0				
5	0				

$$C[1,2] = \max\{C[1,1], C[0,2]\} = 1$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1,j-1]+1 & \text{if } X[i] = Y[j] \text{ and } i,j > 0 \\ \max\{C[i,j-1], C[i-1,j]\} & \text{if } X[i] \neq Y[j] \text{ and } i,j > 0 \end{cases}$$

Table Lookup Strategy

X: | A | C | G | G | A |

Y: | A | C | T | G |

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	
2	0				
3	0				
4	0				
5	0				

$$C[1,3] = \max\{C[1,2], C[0,3]\} = 1$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1,j-1]+1 & \text{if } X[i] = Y[j] \text{ and } i,j > 0 \\ \max\{C[i,j-1], C[i-1,j]\} & \text{if } X[i] \neq Y[j] \text{ and } i,j > 0 \end{cases}$$

Table Lookup Strategy

X					
	A	C	G	G	A

Y				
	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0				
3	0				
4	0				
5	0				

$$C[1,3] = \max\{C[1,3], C[0,4]\} = 1$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i,j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i,j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1			
3	0				
4	0				
5	0				

$$C[2,1] = \max\{C[2,0], C[1,1]\} = 1$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i,j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i,j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2		
3	0				
4	0				
5	0				

$$C[2,2] = C[1,1] + 1 = 1$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	
3	0				
4	0				
5	0				

$$C[2,3] = \max\{C[2,2], C[1,3]\} = 2$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1,j-1]+1 & \text{if } X[i] = Y[j] \text{ and } i,j > 0 \\ \max\{C[i,j-1], C[i-1,j]\} & \text{if } X[i] \neq Y[j] \text{ and } i,j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A
Y	A	C	T	G	

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0				
4	0				
5	0				

$$C[2,4] = \max\{C[2,3], C[1,4]\} = 2$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Table Lookup Strategy

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

Length of LCS of *X* and *Y* is 3

$$C[5,4] = \max\{C[5,3], C[4,4]\} = 3$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1,j-1]+1 & \text{if } X[i] = Y[j] \text{ and } i,j > 0 \\ \max\{C[i,j-1], C[i-1,j]\} & \text{if } X[i] \neq Y[j] \text{ and } i,j > 0 \end{cases}$$

DP algorithm

```
int lcs(char* X,char* Y){
    int i, j, score[m][n];
    int m = strlen(X), n = strlen(Y);
    for(i=0;i<=m;i++) {
        for(j=0;j<=n;j++){
            if(i==0 || j==0)
                score[i][j]=0;
            else if(X[i] == Y[j] )
                score[i][j] = score[i-1][j-1] + 1;
            else{
                if(score[i][j-1]>score[i-1][j])
                    score[i][j] = score[i][j-1];
                else
                    score[i][j] = score[i-1][j];
            }
        }
    }
    return score[m][n];
}
```

Complexity: *O(mn)*
m is the length of *X*
n is the length of *Y*

Longest Common Subsequence

- We have found the length of LCS
- Next step: print out the LCS
- Traverse the table starting from L[m][n]
- For every cell C[i][j]
    - If characters (in X and Y) corresponding to C[i][j] are same
        - Include this character as part of LCS
    - Else
        - Compare values of C[i-1][j] and C[i][j-1] and go in direction of greater value.

Recovering LCS

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

- X[5]!=Y[4]
- So compare C[5,3] and C[4,4]
- C[4,4] is greater
- Go to C[4,4]

LCS=""

Recovering LCS

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

- X[4]=Y[4]
- Append G to LCS
- Go to X[3,3]

LCS="G"

Recovering LCS

X					
X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

- X[3]!=Y[3]
- So compare C[2,3] and C[3,2]
- They are equal, choose either one
- Go to C[2,3]

LCS="G"

Recovering LCS

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

- X[2]!=Y[3]
- So compare C[1,3] and C[2,2]
- C[2,2] is greater
- Go to C[2,2]

LCS="G"

Recovering LCS

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

- X[2]=Y[2]
- Append C to LCS
- Go to X[1,1]

LCS="CG"

Recovering LCS

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

- X[1]=Y[1]
- Append A to LCS
- Go to X[0,0]

LCS="ACG"

Recovering LCS

X	A	C	G	G	A

Y	A	C	T	G

C[i,j]

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	2	2
3	0	1	2	2	3
4	0	1	2	2	3
5	0	1	2	2	3

- Reach C[0,0]
- Return LCS

LCS="ACG"

DP Algorithm

```
// Write this code segment after the table is constructed
int index = score[m][n];
char LCS[index+1];
LCS[index] = '\0';
i = m;
j = n;
while (i > 0 && j > 0)
{
   if (X[i-1] == Y[j-1])
   {
       LCS[index-1] = X[i-1];
       i--; j--; index--;
   }
   else if (score[i-1][j] > score[i][j-1])
      i--;
   else
      j--;
}
printf("LCS: %s\n", LCS );
```

Longest Common Subsequence

- Building the table: $O(nm)$
- Recovering the LCS from the table: $O(n + m)$
  - We walk up and left in an n-by-m array
  - We can only do that for $n+m$ steps.
- Overall complexity of DP algorithm: $O(nm)$

Thank You!

