Understand Dynamic Programming

Design and Analysis of Algorithms

zeshan.khan@nu.edu.pk

Dynamic Programming by Richard Bellman 1950s

- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. Develop a recursive algorithm as per recursive property
2. Develop a memoized recursive algorithm
3. Convert the memoized recursive algorithm into iterative algorithm

Dynamic Programming: Steps

- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. **Establish a recursive property that gives the solution to an instance of the problem.**
2. Develop a recursive algorithm as per recursive property
3. See if same instance of the problem is being solved again an again in recursive calls
4. Develop a memoized recursive algorithm
5. See the pattern in storing the data in the memory
6. Convert the memoized recursive algorithm into iterative algorithm
7. Optimize the iterative algorithm by using the storage as required (storage optimization)

Coin Row (CR) Problem

- *There is a row of n coins whose values are some* positive integers *c1, c2, . . . , cn, not necessarily distinct. The goal is to pick up the* maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

CR Solution

- Let *F(n) be the maximum amount that can be picked up from the row of n* coins. To derive a recurrence for *F(n), we partition all the allowed coin selections* into two groups:
- those that include the last coin and those without it.
- The largest amount we can get from the first group is equal to $cn + F(n - 2)$—*the value of the* n^{th} *coin plus the maximum amount we can pick up from the first* $n - 2$ *coins.*
- The maximum amount we can get from the second group is equal to $F(n - 1)$ *by the* definition of $F(n)$.

CR Solution

- Thus, we have the following recurrence subject to the obvious initial conditions:

$$F(n) = \max\{cn + F(n-2), F(n-1)\} \text{ for } n > 1,$$

$$F(0) = 0, F(1) = c1.$$

Coin-collecting (CC) problem

- Several coins are placed in cells of an $n \times m$ board, no more than one coin per cell. A robot, located in the upper left cell of the board, needs to collect as many of the coins as possible and bring them to the bottom right cell. On each step, the robot can move either one cell to the right or one cell down from its current location. When the robot visits a cell with a coin, it always picks up that coin. Design an algorithm to find the maximum number of coins the robot can collect

CC Solution

- $F(i,j) = \max\{F(i-1,j), F(i,j-1)\} + c_{ij} \quad for\ 1 \leq i \leq n, 1 \leq j \leq m$
- $F(0,j) = 0\ for\ 1 \leq j \leq m$
- $F(i,0) = 0\ for\ 1 \leq i \leq n$

Example

- Write the recursive relation to generate the Fibonacci series of n numbers.

Dynamic Programming: Steps

- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. Establish a recursive property that gives the solution to an instance of the problem.
2. **Develop a recursive algorithm as per recursive property**
3. See if same instance of the problem is being solved again an again in recursive calls
4. Develop a memoized recursive algorithm
5. See the pattern in storing the data in the memory
6. Convert the memoized recursive algorithm into iterative algorithm
7. Optimize the iterative algorithm by using the storage as required (storage optimization)

Recursive Definition to Algorithm

The Fibonacci numbers

- *0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . ,*
- *F(n) = F(n − 1) + F(n − 2) for n > 1*

```
Fib(N)
{       if  (N =< 1)
            return  1;
        else
            return Fib(N-1) + Fib(N-2)
}
```

Exercise

Develop a recursive algorithm for the following definition, where *P[]* is a one-dimensional array having indices from ***i-1*** to ***j,*** and ***m*** is a two-dimensional array of size ***ixj***

$$m[i,j] = \begin{cases} 0 & if \quad i == j \\ \min\limits_{i \le k < j} m[i,k] + m[k+1,j] + p[i] * p[k+1] * p[j+1] & if \quad i < j \end{cases}$$

Exercise

- Develop a recursive algorithm for the following definition
- $C(i,j) =$

$$\begin{cases} 0 & if \quad i == 0 \; or \; j == 0 \\ c(i-1, j-1) + 1 & if \quad i,j > 0 \; and \; x_i = y_i \\ \max(c(i, j-1), c(i-1, j)) & if \quad i,j > 0 \; and \; x_i \neq y_i \end{cases}$$

X = <x1, x2, x3,…..,xm>

Y = <y1, y2, y3,…..,yn>

i varies from m to 0 and j varies from n to 0

Dynamic Programming: Combinations Calculations

- The binomial coefficient is given by
- $\begin{bmatrix} n \\ k \end{bmatrix} = \frac{n!}{k!(n-k)!} \; for \; 0 \leq k \leq n$
- For values of n and k that are not small,
- We can not compute the binomial coefficient directly from this definition because n! is very large

Dynamic Programming: Combinations Calculations

The binomial coefficient is given by

$$T\begin{bmatrix} n \\ k \end{bmatrix} = \begin{cases} \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + \begin{bmatrix} n-1 \\ k \end{bmatrix} & if \quad 0 < k < n \\ 1 & if \quad k == 0 \; or \; k == n \end{cases}$$

```
Bin(n,k){
    if(k==0||k==n){return 1}
    else return Bin(n-1,k-1)+ Bin(n-1,k)
}
```

Dynamic Programming: Combinations Calculations

```
Int bin(int n, int k)
{
   if (k = 0 or n = k )
     return 1;
   else
     return(bin(n-1, k-1) + bin(n-1, k))
}
```

Dynamic Programming: Steps

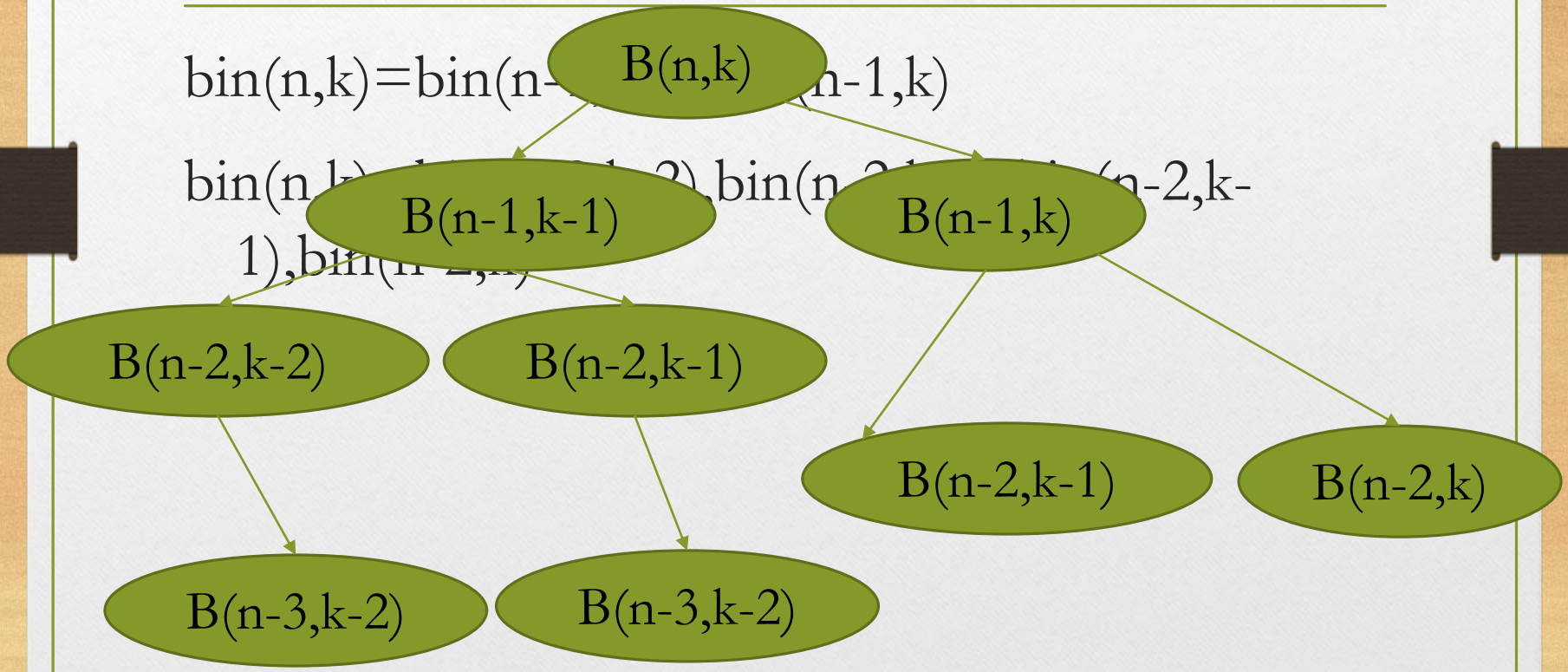
- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. Establish a recursive property that gives the solution to an instance of the problem.
2. Develop a recursive algorithm as per recursive property
3. **See if same instance of the problem is being solved again an again in recursive calls**
4. Develop a memoized recursive algorithm
5. See the pattern in storing the data in the memory
6. Convert the memoized recursive algorithm into iterative algorithm
7. Optimize the iterative algorithm by using the storage as required (storage optimization)

Dynamic Programming: Combinations Calculations

bin(n,k)=bin(n-1,k)

bin(n,k)=bin(n-1,k-1),bin(n-2,k-1),bin(n-2,k)



Dynamic Programming: Steps

- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. Establish a recursive property that gives the solution to an instance of the problem.
2. Develop a recursive algorithm as per recursive property
3. See if same instance of the problem is being solved again an again in recursive calls
4. **Develop a memoized recursive algorithm**
5. See the pattern in storing the data in the memory
6. Convert the memoized recursive algorithm into iterative algorithm
7. Optimize the iterative algorithm by using the storage as required (storage optimization)

Develop a memoized recursive algorithm

```
Int bin(int n, int k)
{
    if (k = 0 or n = k )
      return 1;
    else
      return(bin(n-1, k-1) +
    bin(n-1, k))
}
```

Take an array of size?

Then store results in the array, no need to compute it again and Again

Then see the pattern for data storage in the array

Dynamic Programming: Combinations Calculations

- An efficient algorithm will be developed using dynamic programming approach.
- We will use the recursive definition to construct our solution in an array B.
- Where B[i, j] will contain

Dynamic Programming: Combinations Calculations

- The steps for constructing a dynamic programming algorithm for this problem are
- Establish a recursive property and write it in terms of B i.e.
- $$B[i,j] = \begin{cases} B[i-1,j-1] + B[i-1,j] & if \quad 0 < i < j \\ 1 & if \quad i == j \; or \; j == 0 \end{cases}$$
- Solve an instance of the problem in a bottom-up fashion by computing the rows in B in sequence starting with the first row

Dynamic Programming: Steps

- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. Establish a recursive property that gives the solution to an instance of the problem.
2. Develop a recursive algorithm as per recursive property
3. See if same instance of the problem is being solved again an again in recursive calls
4. Develop a memoized recursive algorithm
5. **See the pattern in storing the data in the memory**
6. Convert the memoized recursive algorithm into iterative algorithm
7. Optimize the iterative algorithm by using the storage as required (storage optimization)

Dynamic Programming: Combinations Calculations

	0	1	2	3	4	j	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		

Each successive row is computed from the row preceding it using the recursive property.

$$B[i,j] = B[i-1,j-1]\, B[i-1,j]$$

Dynamic Programming Combinations Calculations Example

$$\begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

- Compute B[4, 2] =
- Compute row 0: B[0, 0] = 1
- Compute row 1: B[1, 0] = 1
  B[1, 1] = 1
- Compute row 2:
  B[2, 0] = 1
  B[2, 1] = B[1, 0] + B[1, 1] = 1 + 1 = 2
  B[2, 2] = 1
- Compute row 3: B[3, 0] = 1
  B[3, 1] = B[2, 0] + B[2, 1] = 1 + 2 = 3
  B[3, 2] = B[2, 1] + B[2, 2] = 2 + 1 = 3
- Compute row 4: B[4, 0] = 1
  B[4, 1] = B[3, 0] + B[3, 1] = 1 + 3 = 4
  B[4, 2] = B[3, 1] + B[3, 2] = 3 + 3 = 6

Dynamic Programming: Steps

- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. Establish a recursive property that gives the solution to an instance of the problem.
2. Develop a recursive algorithm as per recursive property
3. See if same instance of the problem is being solved again an again in recursive calls
4. Develop a memoized recursive algorithm
5. See the pattern in storing the data in the memory
6. **Convert the memoized recursive algorithm into iterative algorithm**
7. Optimize the iterative algorithm by using the storage as required (storage optimization)

Dynamic Programming Combinations Calculations Algorithm

```
Int bin(int n, int k)
{
    int i, j;
    int B[0..n, 0..k];
    for i = 0 to n
        for j = 0 to minimum(i, k)
            if( j = 0 or j = i)
                B[i, j] = 1;
            else
                B[i, j] = B[i-1, j-1] + B[i-1, j];
    return B[n, k]
}
```

Dynamic Programming: Steps

- **The seven steps in the development of a dynamic programming algorithm are as follows*:***

1. Establish a recursive property that gives the solution to an instance of the problem.
2. Develop a recursive algorithm as per recursive property
3. See if same instance of the problem is being solved again an again in recursive calls
4. Develop a memoized recursive algorithm
5. See the pattern in storing the data in the memory
6. Convert the memoized recursive algorithm into iterative algorithm
7. **Optimize the iterative algorithm by using the storage as required (storage optimization)**

Exercise

- Initialise a two dimensional array A[1..n, 1..n] using following definition with an efficient algorithm.
- $A[i, j] = i + j$ if $i = j$
- $A[i, j] = \text{Min}(A[i, j-1], A[i+1, j]) + (i+j)$ if $i < j$
- $A[i, j] = \text{Max}(A[i-1, j], A[i, j+1]) + (i * j)$ if $i > j$
- Where i is for row index and j is for column index.