22-0: **Dynamic Programming**

- Simple, recursive solution to a problem
- Naive solution recalculates same value many times
- Leads to exponential running time

22-1: **Fibonacci Numbers**

- Calculating the nth Fibonacci number
  - Fib(0) = 1
  - Fib(1) = 1
  - Fib(n) = Fib(n-1) + Fib(n-2)

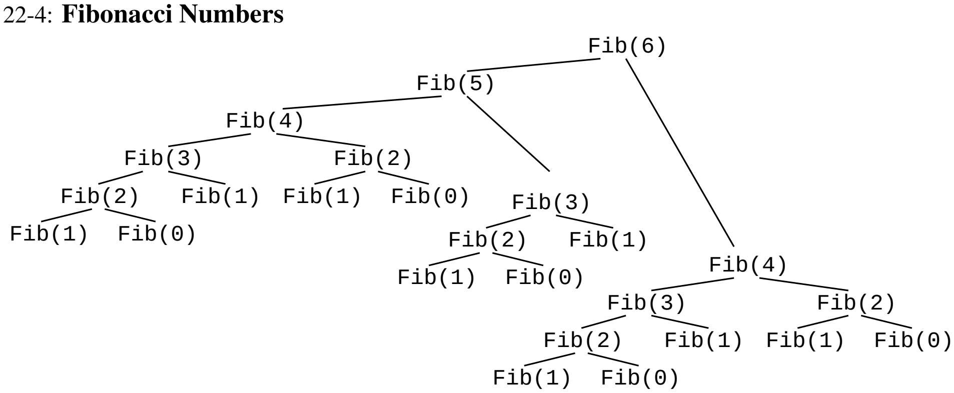
22-2: **Fibonacci Numbers**

```
int Fibonacci(int n) {

  if (n == 0)
      return 1;

  if (n == 1)
      return 1;

  return Fibonacci(n-1) + Fibonacci(n-2);
}
```

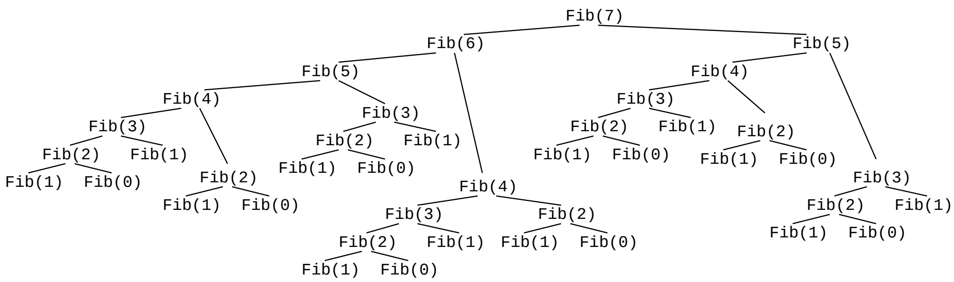
22-3: **Fibonacci Numbers**

- Why is this solution bad?
- Recalculate values many times
  - Many, many, times

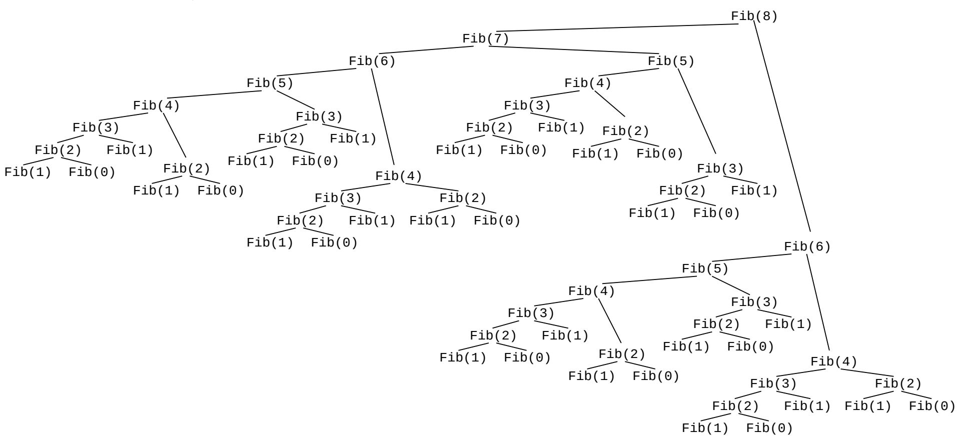
22-4: **Fibonacci Numbers**



22-5: **Fibonacci Numbers**



22-6: **Fibonacci Numbers**



22-7: **How Bad is Recalculation?**

- Assume 2 GHz machine
- Add every cycle
  - No time spent on recursive call overhead
  - Lower bound on time required
- Fibonacci(100) will take:

22-8: **How Bad is Recalculation?**

- Assume 2 GHz machine
- Add every cycle
  - No time spent on recursive call overhead
  - Lower bound on time required
- Fibonacci(100) will take:
  - 9087 Years

22-9: **How Bad is Recalculation?**

- Assume 2 GHz machine
- Add every cycle
  - No time spent on recursive call overhead

  - Lower bound on time required
- Fibonacci(100) will take:
  - 9087 Years
- Fibonacci(200) will take:

22-10: **How Bad is Recalculation?**

- Assume 2 GHz machine
- Add every cycle
  - No time spent on recursive call overhead
  - Lower bound on time required
- Fibonacci(100) will take:
  - 9087 Years
- Fibonacci(200) will take:
  - 719770570404153908544 millennia
  - Well after heat death of the universe (100 trillion years)

22-11: **Dynamic Programming**

- Recalculating values can lead to unacceptable run times
  - Even if the total number of values that needs to be calculated is small
- Solution: Don't recalculate values
  - Calculate each value once
  - Store results in a table
  - Use the table to calculate larger results

22-12: **Dynamic Programming**

- To calculate Fibonacci(100), only need to calculate 101 values
- Fibonacci(n) can be calculated in time $O(1)$
  - Assuming we have values for Fibonacci(n-1) and Fibonacci(n-2)

22-13: **Dynamic Programming**

- Create a table: FIB[]
  - FIB[n] = nth Fibonacci number
- Fill the table from left to right
- Use old values in table to calculate new values

22-14: **Faster Fibonacci**

```
int Fibonacci(int n) {

 int[] FIB = new int[n+1];

  FIB[0] = 1;
  FIB[1] = 1;

  for (i=2; i<=n; i++)
    FIB[i] = FIB[i-1] + FIB[i-2];

  return FIB[n];
}
```

22-15: **Dynamic Programming**

- To create a dynamic programming solution to a problem:
    - Create a simple recursive solution (that may require a large number of repeat calculations
    - Design a table to hold partial results
    - Fill the table such that whenever a partial result is needed, it is already in the table

22-16: **World Series**

- Two teams T_1 and T_2
- T_1 will win any game with probability p
    - T_2 will win any game with probability $1 - p$
- What is the probability that T_1 will win a best-of-seven series?
    - Answer is *not* p : why not?

22-17: **World Series**

- Calculate the probability that T_1 will win the series, given T_1 needs to win x more games, and T_2 needs to win y more games
    - PT1win(x,y)
- The probability that P_1 will win a best-of-seven series is then PT1win(4,4)
- The probability that P_1 will win a best-of-seven series, if P_1 has already one 2 games, and P_2 has won 1 game is then PT1win(2,3)

22-18: **World Series**

- Base cases:
    - What is PT1win(0,x)?

22-19: **World Series**

- Base cases:
  - What is PT1win(0,x)?
    - 1! T_1 has already won!
  - What is PT1win(x,0)?

22-20: **World Series**

- Base cases:
  - What is PT1win(0,x)?
    - 1! T_1 has already won!
  - What is PT1win(x,0)?
    - 0! T_1 has already lost!

22-21: **World Series**

- Recursive Case: PT1win(x,y)
  - If T_1 wins the next, game, then the probability that T_1 will win the rest of the series is

22-22: **World Series**

- Recursive Case: PT1win(x,y)
  - If T_1 wins the next, game, then the probability that T_1 will win the rest of the series is
    - PT1win(x-1,y)
  - If T_1 loses the next game, then the probability that T_1 will win the rest of the series is:

22-23: **World Series**

- Recursive Case: PT1win(x,y)
  - If T_1 wins the next, game, then the probability that T_1 will win the rest of the series is
    - PT1win(x-1,y)
  - If T_1 loses the next game, then the probability that T_1 will win the rest of the series is
    - PT1win(x,y-1)

22-24: **World Series**

- Recursive Case: PT1win(x,y)
  - If T_1 wins the next, game, then the probability that T_1 will win the rest of the series is
    - PT1win(x-1,y)
  - If T_1 loses the next game, then the probability that T_1 will win the rest of the series is
    - PT1win(x,y-1)
  - Probability that T_1 will win is p
  - Probability that T_1 will lose is $1 - p$
  - What then is PT1win(x,y)?

22-25: **World Series**

- Recursive Case: PT1win(x,y)
  - If T_1 wins the next, game, then the probability that T_1 will win the rest of the series is
    - PT1win(x-1,y)
  - If T_1 loses the next game, then the probability that T_1 will win the rest of the series is
    - PT1win(x,y-1)
  - Probability that T_1 will win is p
  - Probability that T_1 will lose is $1 - p$
  - PT1win(x,y) = p * PT1win(x-1,y) +
    $(1 - p)$ * PTwin(x,y-1)

22-26: **World Series**

```
float PT1win(int x, int y, int p) {
   if (x == 0) return 1;
   if (y == 0) return 0;

   return    p    * PT1win(x-1,y  ,p) +
          (1 - p) * PT1win(x,  y-1,p);
}
```

22-27: **World Series**

```
float PT1win(int x, int y, int p) {
   if (x == 0) return 1;
   if (y == 0) return 0;

   return    p    * PT1win(x-1,y  ,p) +
          (1 - p) * PT1win(x,  y-1,p);
}
```

- Just like Fibonacci, recalculating values – exponential time
- How many total values do we need to calculate for PT1win(n,n)?

22-28: **World Series**

```
float PT1win(int x, int y, int p) {
   if (x == 0) return 1;
   if (y == 0) return 0;

   return    p    * PT1win(x-1,y  ,p) +
          (1 - p) * PT1win(x,  y-1,p);
}
```

- Just like Fibonacci, recalculating values – exponential time
- How many total values do we need to calculate for PT1win(n,n)?
  - n^2

22-29: **World Series**

- $P[x, y]$ = # of games required for T_1 to win, if T_1 needs to win x more games, and T_2 needs to win y more games.
  - $P[0, x] = 1$ for all $x > 0$
  - $P[x, 0] = 0$ for all $x > 0$
  - $P[x, y] = p * P[x - 1, y] + (1 - p) * P[x, y - 1]$
- Need to fill out the table such that when we need a partial value, it has already been computed

22-30: **World Series**

y →, x ↓

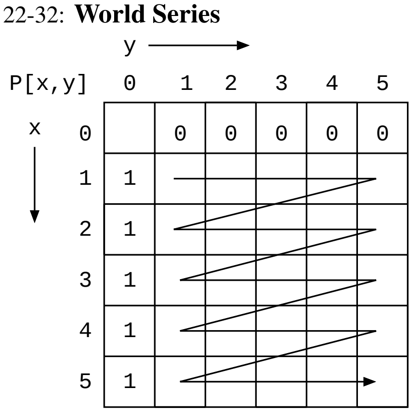
P[x,y]	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

22-31: **World Series**

y →, x ↓

P[x,y]	0	1	2	3	4	5
0		0	0	0	0	0
1	1					
2	1					
3	1					
4	1					
5	1					

22-32: **World Series**



22-33: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0					
1					
2					
3					
4					

22-34: **World Series**

- P(x,y)

$p = 0.9$

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

22-35: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9			
2	0				
3	0				
4	0				

22-36: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99		
2	0				
3	0				
4	0				

22-37: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	
2	0				
3	0				
4	0				

22-38: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0				
3	0				
4	0				

22-39: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81			
3	0				
4	0				

22-40: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972		
3	0				
4	0				

22-41: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	
3	0				
4	0				

22-42: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0				
4	0				

22-43: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729			
4	0				

22-44: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729	.9477		
4	0				

22-45: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729	.9477	.9914	
4	0				

22-46: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729	.9477	.9914	.9987
4	0				

22-47: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729	.9477	.9914	.9987
4	0	.6561			

22-48: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729	.9477	.9914	.9987
4	0	.6561	.9185		

22-49: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729	.9477	.9914	.9987
4	0	.6561	.9185	.9841	

22-50: **World Series**

- P(x,y)

$p = 0.9$

	0	1	2	3	4
0		1	1	1	1
1	0	.9	.99	.999	.9999
2	0	.81	.972	.9963	.9995
3	0	.729	.9477	.9914	.9987
4	0	.6561	.9185	.9841	.9972

22-51: **World Series**

- P(x,y)

$p = 0.6$

	0	1	2	3	4	5
0		1	1	1	1	1
1	0	.6	.84	.936	.9744	.98976
2	0	.36	.648	.820	.91296	.95904
3	0	.216	.4752	.68208	.82061	.90367
4	0	.1296	.33696	.54403	.70998	.82619
5	0	.07776	.23328	.41973	.59388	.73327

22-52: **Sequences & Subsequences**

- A *sequence* is an ordered list of elements
  - <A, B, C, B, D, A, B>
- A *subsequence* is a sequence with some elements left out;
- Subsequences of <A, B, C, B, D, A, B>
  - <B, B, A>
  - <A, B, C>
  - <B, D, A, B>
  - <C>

22-53: **Sequences & Subsequences**

- A *sequence* is an ordered list of elements
  - <A, B, C, B, D, A, B>
- A *subsequence* is a sequence with some elements left out
- *NON*-Subsequences of <A, B, C, B, D, A, B>
  - <D, A, C>
  - <A, B, B, C>
  - <C, A, D>
  - <B, D, B, A>

22-54: **Common Subsequences**

- Given two sequences S_1 and S_2, a *common subsequence* is s subsequence of both sequences
- <A, B, C, B, D, A, B>, <B, D, C, A, B, A>
- Common Subsequences:
  - <B, C, A>
  - <B, D>
  - <B, A, B>
  - <B, C, B, A>

22-55: **LCS**

- Longest Common Subsequence
- Need not be unique
- <A, B, C, B, D, A, B>, <B, D, C, A, B, A>
  - <B, C, B, A>
  - <B, D, A, B>

22-56: **LCS**

- Given the sequences:
  <A, B, A, B, B> <B, C, A, B>
- LCS must end in B.
  - Why?

22-57: **LCS**

- Given the sequences:
  <A, B, A, B, B> <B, C, A, B>
- LCS must end in B.
- Length of LCS:
  - 1 + lengthLCS (<A, B, A, B> , <B, C, A>)

22-58: **LCS**

- Given the sequences:
  <A, B, A, B> <B, C, A>
- The last element in the LCS must be:
  - *not* B
  - *not* A

22-59: **LCS**

- Given the sequences:
  <A, B, A, B> <B, C, A>
- The last element in the LCS must be:
  - *not* B
  - *not* A
- Length of LCS: Maximum of:
  - lengthLCS (<A, B, A> , <B, C, A>)
  - lengthLCS (<A, B, A, B> , <B, C>)

22-60: **LCS Pseudo-Code**

```
LCS(Seq1, Seq2) {
   if (Seq1 is empty) || (Seq2 is empty)
      return 0;
   if (last elem in Seq1 = last elem in Seq2)
      return 1 + LCS(Seq1 - last element,
                     Seq2 - last element)
   return MAX(LCS(Seq1 - last element, Seq2),
              LCS(Seq1, Seq2 - last element))
}
```

22-61: **LCS Pseudo-Code**

```
LCS(int x, int y, String S1, String S2) {
  if ((x == 0) || (y == 0))
    return 0;

  if (S1.charAt(x-1) == S2.charAt(y-1))
    return 1 + LCS(x-1, y-1, S1, S2);
  else
    return MAX(LCS(x-1, y, S1, S2),
               LCS(x, y-1, S1, S2));
}
```

22-62: **LCS Pseudo-Code**

```
LCS(int x, int y, String S1, String S2) {
  if ((x == 0) || (y == 0))
    return 0;

  if (S1.charAt(x-1) == S2.charAt(y-1))
    return 1 + LCS(x-1, y-1, S1, S2);
  else
    return MAX(LCS(x-1, y, S1, S2),
               LCS(x, y-1, S1, S2));
}
```

- Requires exponential time in (x+y)

22-63: **LCS**

- For x,y:
  - Total number of subproblems

22-64: **LCS**

- For x,y:
  - Total number of subproblems
    - (x + 1) * (y + 1) ($O(x * y)$)

22-65: **LCS**

- Create a table T

- T[i,j] = LCS(i, j, S1, S2)
- T[x,0] = 0
- T[0,x] = 0
- T[x,y] =
  - 1 + T[x-1,y-1] if S1[x] = S2[y]
  - MAX(T[x-1,y], T[x,y-1]) otherwise

22-66: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0							
A	1							
B	2							
C	3							
B	4							
D	5							
A	6							
B	7							

22-67: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0						
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-68: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0					
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-69: **LCS**

22-70: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0				
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-71: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0			
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-72: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1		
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-73: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-74: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0						
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-75: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1					
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-76: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1				
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-77: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1			
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-78: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1		
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-79: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-80: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0						
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-81: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1					
B	4	0						
D	5	0						
A	6	0						
B	7	0						

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1				
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-82: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2			
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-83: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2		
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-84: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-85: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0						
D	5	0						
A	6	0						
B	7	0						

22-86: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1					
D	5	0						
A	6	0						
B	7	0						

22-87: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1				
D	5	0						
A	6	0						
B	7	0						

22-88: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2			
D	5	0						
A	6	0						
B	7	0						

22-89: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2		
D	5	0						
A	6	0						
B	7	0						

22-90: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	
D	5	0						
A	6	0						
B	7	0						

22-91: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0						
A	6	0						
B	7	0						

22-92: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1					
A	6	0						
B	7	0						

22-93: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2				
A	6	0						
B	7	0						

22-94: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2			
A	6	0						
B	7	0						

22-95: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2		
A	6	0						
B	7	0						

22-96: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	
A	6	0						
B	7	0						

22-97: **LCS**

22-98: **LCS**

			B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0						
B	7	0						

22-99: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1					
B	7	0						

22-100: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2				
B	7	0						

22-101: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2			
B	7	0						

22-102: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3		
B	7	0						

22-103: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	
B	7	0						

22-104: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0						

22-105: **LCS**

		0	B 1	D 2	C 3	A 4	B 5	A 6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1					

22-106: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2				

22-107: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2			

22-108: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3		

22-109: **LCS**

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	4

22-110: **Memoization**

- Can be difficult to determine order to fill the table
- We can use a table together with recursive solution
  - Initialize table with sentinel value
  - In recursive function:
    - Check table – if entry is there, use it
    - Otherwise, call function recursively
      Set appropriate table value
      return table value

22-111: **LCS Memoized**

```
LCS(int x, int y, String S1, String S2) {
  if ((x == 0) || (y == 0))
    T[x,y] = 0;
    return 0;
  if (T[x,y] != -1)
    return T[x,y];
  if (S1.charAt(x) == S2.charAt(y))
    T[x,y] =  1 + LCS(x-1, y-1, S1, S2);
  else
    T[x,y] =  MAX(LCS(x-1, y, S1, S2),
                  LCS(x, y-1, S1, S2));
  return T[x,y];
}
```

22-112: **Fibonacci Memoized**

```
int Fibonacci(int n) {

  if (n == 0)
      return 1;

  if (n == 1)
      return 1;

  if (T[n] == -1)
    T[n] = Fibonacci(n-1) + Fibonacci(n-2);
```

```
  return T[n];
}
```

22-113: **Making Change**

- Problem:
  - Coins: 1, 5, 10, 25, 50
  - Smallest number of coins that sum to an amount X?
- How can we solve it?

22-114: **Making Change**

- Problem:
  - Coins: 1, 4, 6
  - Smallest number of coins that sum to an amount X?
- Does the same solution still work? Why not?

22-115: **Making Change**

- Problem:
  - Coins: d_1, d_2, d_3, . . ., d_k
    - Can assume $d_1 = 1$
  - Value X
  - Find smallest number of coins that sum to X
- Solution:

22-116: **Making Change**

- Problem:
  - Coins: d_1, d_2, d_3, . . ., d_k
    - Can assume $d_1 = 1$
  - Value X
  - Find smallest number of coins that sum to X
- Solution:
  - We can use any of the coins d_i whose value is less than or equal to X
  - We then have a smaller subproblem: Finding change for value up to $X - d_i$.
  - How do we know which one to chose? Try them all!

22-117: **Making Change**

- Problem:
  - Coins: d_1, d_2, d_3, . . ., d_k
    - Can assume $d_1 = 1$

  - Value X
  - Find smallest number of coins that sum to X

- Solution:
  - $C[X]$ = smallest number of coins required for amount X
  - What is the base case?
  - What is the recursive case?

22-118: **Making Change**

- $C[X]$ = smallest number of coins required for amount X, using coins $d_1, d_2, d_3 \ldots d_k$
  - Base Case:
$$C[0] = 0$$
  - Recursive Case:
$$C[X] = \min_{1 \leq i \leq n} 1 + C[X - d_i]$$

  (where d_n is the largest coin $\leq X$)

22-119: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-120: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-121: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-122: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-123: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-124: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-125: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-126: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-127: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-128: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-129: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

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

22-130: **Making Change**

- Given the table, can we determine the optimal way to make change for a given value X? How?

22-131: **Making Change**

- Given the table, can we determine the optimal way to make change for a given value X? How?
  - Look back through table, determine which coin was used to get the smallest number of coins
    - (examples)
- We could also store which coin we used to get the smallest number of coins

22-132: **Making Change**

$d_1 = 1, d_2 = 4, d_3 = 6$

0	0	0	0
1	1	0	d_1 (1)
2	2	0	d_1 (1)
3	3	0	d_1 (1)
4	1	0	d_2 (4)
5	2	0	d_2 (4)
6	1	0	d_3 (6)
7	2	0	d_3 (6)
8	2	0	d_2 (4)
9	3	0	d_2 (4)
10	2	0	d_3 (6)

22-133: **Matrix Multiplication**

- Quick review (on board)
  - Matrix A is $i \times j$
  - Matrix B is $j \times k$
  - # of scalar multiplications in $A * B$?

22-134: **Matrix Multiplication**

- Quick review (on board)
  - Matrix A is $i \times j$

- Matrix B is $j \times k$
- # of scalar multiplications in $A * B$?
  - $i * j * k$

22-135: **Matrix Chain Multiplication**

- Multiply a chain of matrices together
  - $A * B * C * D * E * F$
- Matrix Multiplication is associative
  - $(A * B) * C = A * (B * C)$
  - $(A * B) * (C * D) = A * (B * (C * D)) = ((A * B) * C) * D = A * ((B * C) * D) = (A * (B * C)) * D$

22-136: **Matrix Chain Multiplication**

- Order Matters!
- $A : (100 \times 100), B : (100 \times 100), C : (100 \times 100), D : (100 \times 1)$
  - $((A * B) * C) * D$ Scalar multiplications:
  - $A * (B * (C * D))$ Scalar multiplications:

22-137: **Matrix Chain Multiplication**

- Order Matters!
- $A : (100 \times 100), B : (100 \times 100), C : (100 \times 100), D : (100 \times 1)$
  - $((A * B) * C) * D$ Scalar multiplications: 2,010,000
  - $A * (B * (C * D))$ Scalar multiplications: 30,000

22-138: **Matrix Chain Multiplication**

- Matrices $A_1, A_2, A_3 \ldots A_n$
- Matrix A_i has dimensions $p_{i-1} \times p_i$
- Example:
  - $A_1 : 5 \times 7$, $A_2 : 7 \times 9$, $A_3 : 9 \times 2$, $A_4 : 2 \times 2$
  - $p_0 = 5, p_1 = 7, p_2 = 9, p_3 = 2, p_4 = 2$
  - How can we break $A_1 * A_2 * A_3 * \ldots * A_n$ into smaller subproblems?
  - Hint: Consider the last multiplication

22-139: **Matrix Chain Multiplication**

- $M[i, j]$ = smallest # of scalar multiplications required to multiply $A_i * \ldots * A_j$
- Breaking $M[1, n]$ into subproblems:
  - Consider last multiplication
  - (use whiteboard)

22-140: **Matrix Chain Multiplication**

- $M[i, j]$ = smallest # of scalar multiplications required to multiply $A_i * \ldots * A_j$
- Breaking $M[1, n]$ into subproblems:
  - Consider last multiplication:
  - $(A_1 * A_2 * \ldots * A_k) * (A_{k+1} * \ldots * A_n)$
  - $M[1, n] = M[1, k] + M[k + 1, n] + p_0 p_k p_n$
  - In general, $M[i, j] = M[i, k] + M[k + 1, j] + p_{i-1} p_k p_j$
    - What should we choose for k? which value between i and $j - 1$ should we pick?

22-141: **Matrix Chain Multiplication**

- Recursive case:

$$M[i, j] = \min_{i \le k < j} (M[i, k] + M[k + 1, j] + p_{i-1} * p_k * p_j$$

- What is the base case?

22-142: **Matrix Chain Multiplication**

- Recursive case:

$$M[i, j] = \min_{i \le k < j} (M[i, k] + M[k + 1, j] + p_{i-1} * p_k * p_j$$

- What is the base case?

$$M[i, i] = 0$$

for all i

22-143: **Matrix Chain Multiplication**

$$M[i, j] = \min_{i \le k < j} (M[i, k] + M[k + 1, j] + p_{i-1} * p_k * p_j$$

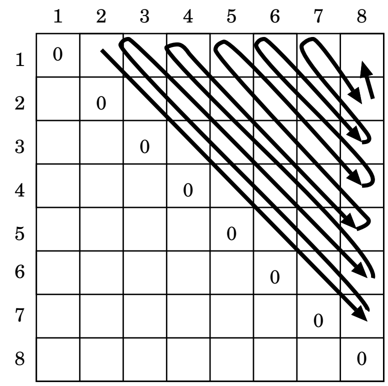
- In what order should we fill in the table? What to we need to compute $M[i, j]$?

	1	2	3	4	5	6	7	8
1	0							
2		0						
3			0					
4				0				
5					0			
6						0		
7							0	
8								0

22-144: **Matrix Chain Multiplication**

$$M[i, j] = \min_{i \le k < j} (M[i, k] + M[k + 1, j] + p_{i-1} * p_k * p_j$$

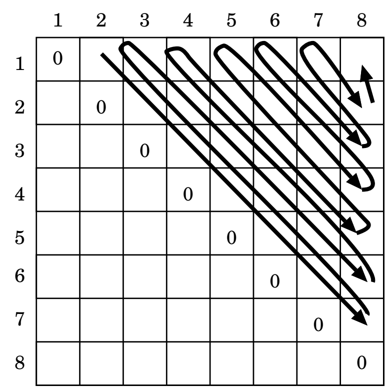
- In what order should we fill in the table? What to we need to compute $M[i, j]$?



22-145: **Matrix Chain Multiplication**

$$M[i, j] = \min_{i \le k < j} (M[i, k] + M[k + 1, j] + p_{i-1} * p_k * p_j$$

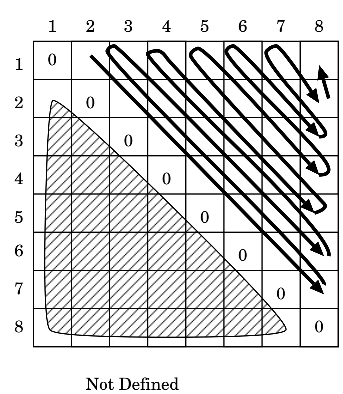
- What about the lower-left quadrant of the table?



22-146: **Matrix Chain Multiplication**

$$M[i, j] = \min_{i \le k < j} (M[i, k] + M[k + 1, j] + p_{i-1} * p_k * p_j$$

- What about the lower-left quadrant of the table?



22-147: **Matrix Chain Multiplication**

```
Matrix-Chain-Order(p)
    n ← # of matrices
    for i ← 1 to n do
        M[i,i] ← 0
    for l ← 2 to n do
        for i ←1 to n − l + 1
            j ← i + l − 1
            M[i,j] ← ∞
            for k ← i to j − 1 do
                q ← M[i,k] + M[k+1,j] + p_{i−1} * p_k * p_j
                if q < M[i,j] then
                    M[i,j] = q
                    S[i,j] = k
```