Dynamic Programming

a. Fibonacci Series
b. Weighted Interval Scheduling
c. Knapsack
d. Longest Common Subsequence

Common Subsequences

- Given a string $x \in \Sigma^n$ a **subsequence** is any string obtained by deleting a subset of the symbols

r	e	c	u	r	a	n	c	e

- Given two strings $x \in \Sigma^n, y \in \Sigma^m$, a **common subsequence** is a **subsequence** of both x and y

r	e	c	u	r	a	n	c	e

r	e	c	u	r	r	e	n	c	e

Longest Common Subsequence (LCS)

- **Input:** Two strings $x \in \Sigma^n, y \in \Sigma^m$
- **Output:** The longest common subsequence of x and y

Writing the Recurrence

Recurrence:

$$\mathrm{LCS}(i,j) = \begin{cases} 1 + \mathrm{LCS}(i-1, j-1) & \text{if } x_i = y_j \\ \max\{\mathrm{LCS}(i-1, j), \mathrm{LCS}(i, j-1)\} & \text{if } x_i \neq y_j \end{cases}$$

Base Cases:

$\mathrm{LCS}(i, 0) = 0, \mathrm{LCS}(0, j) = 0$

Solving the Recurrence: Bottom-Up

```
// All inputs are global vars
FindOPT(n,m):
  M[i,0] ← 0,    M[0,j] ← 0

  for (i= 1,…,n):
    for (j = 1,…,m):
      if (x_i = y_j):
        M[i,j] ← 1 + M[i-1,j-1]
      else:
        M[i,j] ← max{M[i-1,j],M[i,j-1]}

  return M[n,m]
```

Ask the Audience

x = peat

y = leapt

Compute LCS(i,j) for each subproblem

		j = 0	1	2	2	4	5
		-	l	e	a	p	t
i = 0	-	**0**	**0**	**0**	**0**	**0**	**0**
1	p	**0**					
2	e	**0**					
3	a	**0**					
4	t	**0**					

Ask the Audience

x = peat

y = leapt

Compute LCS(i,j) for each subproblem

	j = 0	1	2	2	4	5
		-	**l**	**e**	**a**	**p**	**t**
i = 0	**-**	**0**	**0**	**0**	**0**	**0**	**0**
1	**p**	**0**	**0**	**0**	**0**	**1**	**1**
2	**e**	**0**	**0**	**1**	**1**	**1**	**1**
3	**a**	**0**	**0**	**1**	**2**	**2**	**2**
4	**t**	**0**	**0**	**1**	**2**	**2**	**3**

Finding the Solution

```
// All inputs are global vars
FindLCS(i,j):
  if (i = 0 or j = 0)
    return ""
  if (x_i = y_j):
    return FindLCS(i-1,j-1)+ x_i
  else:
    if (M[i-1,j] > M[i,j-1])
      return FindLCS(i-1,j)
    else:
      return FindLCS(i,j-1)
  return M[n,m]
```

Dynamic Programming

a. Fibonacci Series
b. Weighted Interval Scheduling
c. Knapsack
d. Longest Common Subsequence
e. Longest Increasing Subsequence

Longest Increasing Subsequence (LIS)

- **Input:** a sequence of numbers $x_1, \dots, x_n$

sequence

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

Longest Increasing Subsequence (LIS)

- **Input:** a sequence of numbers $x_1, \dots, x_n$

sequence

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

increasing subsequence

- Increasing Subsequence:
  indices $1 \le i_1 \le i_2 \le \cdots \le i_k \le n$
  such that $x_{i_1} < x_{i_2} < \cdots < x_{i_k}$

Longest Increasing Subsequence (LIS)

- **Input:** a sequence of numbers $x_1, \ldots, x_n$

sequence

4	**0**	8	**2**	9	3	1	2	**3**	7	**4**	**6**	3

increasing subsequence

- **Output:** a longest increasing subsequence

sequence

4	**0**	8	2	9	3	**1**	**2**	**3**	7	**4**	**6**	3

longest increasing subsequence

Ask the Audience

- Find a longest increasing subsequence of

14	**7**	**5**	**6**	**2**	**12**

Identifying the Subproblems

- Start by finding the value of the optimal solution:
  - In this problem: length of the LIS

Identifying the Subproblems

- Start by finding the value of the optimal solution:
  - In this problem: length of the LIS
- What about defining $\mathrm{LIS}(j)$ to be the length of the longest increasing subsequence between the first j elements?

8	9	12	3	6	10

Writing the Recurrence

- Let $\mathrm{LIS}(j)$ be the length of the longest increasing subsequence **that ends with** x_j

8	9	12	3	6	10

Writing the Recurrence

- Let $\mathrm{LIS}(j)$ be the length of the longest increasing subsequence **that ends with** x_j

- **Case i:** the previous element is x_i

6	7	14	5	12	8

Writing the Recurrence

- Let $\text{LIS}(j)$ be the length of the longest increasing subsequence **that ends with** x_j

- **Case i:** the previous element is x_i
  - Some cases are invalid

6	7	14	5	12	8

Writing the Recurrence

- Let $\text{LIS}(j)$ be the length of the longest increasing subsequence **that ends with** x_j
- **Case i:** the last two numbers are x_i and x_j

Recurrence:

$$\text{LIS}(j) = 1 + \max_{1 \le i < j \text{ and } x_i < x_j} \text{LIS}(i)$$

Base Case:

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

Ask the Audience

- Fill out the values $\mathrm{LIS}(j)$ for $j = 1, \ldots, 6$

6	10	5	14	8	7

j	1	2	3	4	5	6
LIS(j)	1					

Ask the Audience

Is $\mathrm{LIS}(n)$ the length of the optimal solution?

Solving the Recurrence: Bottom-Up

```
// All inputs are global vars
FindOPT(n):
  M[1] ← 1

  for (j = 2,…,n):
```

$$M[j] = 1 + \max_{1 \le i < j \text{ and } x_i < x_j} M[i]$$

```
  return
```

$$\max_{1 \le j \le n} M[j]$$

Solving the Recurrence: Bottom-Up

```
FindOPT(n):
  M[1] ← 1

  for (j = 2,…,n):
```
$$M[j] = 1 + \max_{1 \le i < j \text{ and } x_i < x_j} M[i]$$

```
  return
```
$$\max_{1 \le j \le n} M[j]$$

Running time:

Recovering the LIS

- Fill out the values $\mathrm{LIS}(j)$ for $j = 1, \ldots, 6$

6	10	5	14	8	7

j	1	2	3	4	5	6
LIS(j)	1	2	1	3	2	2

Recovering the LIS

```
FindLIS(n):
  if (n=1)
    return x_1
  j= argmax_{1≤i<n and x_i<x_n} M[i]
  return FindLIS(j) + {x_n}
```

Summary

- Can compute a LIS in time $O(n^2)$
  - Same algorithm works for longest non-decreasing, longest decreasing, longest non-increasing, and more
- Dynamic Programming:
  - Question: What is the final symbol in the LIS?
  - Subproblems represent LIS **with a specific final symbol**
  - The actual optimal value is not always in LIS(n)