COMP 355
Advanced Algorithms

Dynamic Programming:
Knapsack & LCS
Section 6.4 (KT)

Rhodes College
—1848—

Knapsack Problem

There are two versions of the problem:
(1) “0-1 knapsack problem” and
(2) “Fractional knapsack problem”

(1) Items are indivisible; you either take an item or not. Solved with *dynamic programming*
(2) Items are divisible: you can take any fraction of an item. Solved with a *greedy algorithm*.

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$.
- Knapsack has capacity of W kilograms.
- Goal: fill knapsack so as to maximize total value.

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Ex: { 3, 4 } has value 40.

Greedy: repeatedly add item with maximum ratio v_i / w_i.

Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Knapsack Problem: Bottom-Up

Knapsack. Fill up an n-by-W array.

```
Input: n, w_1,…,w_N, v_1,…,v_N

for w = 0 to W
   M[0, w] = 0

for i = 1 to n
   for w = 1 to W
      if (w_i > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], v_i + M[i-1, w-w_i ]}

return M[n, W]
```

Knapsack Algorithm

W + 1

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	34	40

OPT: { 4, 3 }
value = 22 + 18 = 40

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack Problem: Running Time

Running time. $\Theta(n\ W)$.

- Not polynomial in input size!
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete.

Knapsack approximation algorithm. There exists a polynomial algorithm that produces a feasible solution that has value within 0.01% of optimum.

Using DP to compare strings

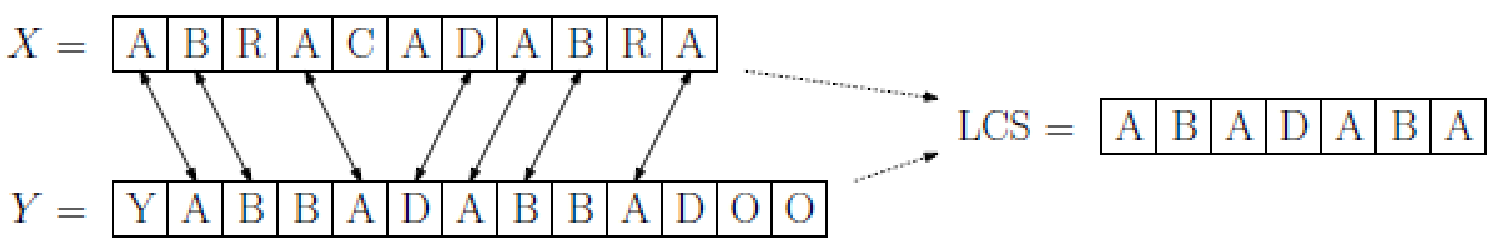
- Determining the degree of similarity between two strings
  - Applications in computational biology (sequence alignment)
  - Applications in document processing and retrieval

- One common measure of similarity between two strings is the lengths of their longest common subsequence.

Longest Common Subsequence (LCS)

Given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Z = \langle z_1, z_2, \ldots, z_k \rangle$, we say that Z is a subsequence of X if there is a strictly increasing sequence of k indices $\langle i_1, i_2, \ldots, i_k \rangle$ $(1 \leq i_1 < i_2 < \ldots < i_k \leq n)$ such that $Z = \langle x_{i1}, x_{i2}, \ldots, x_{ik} \rangle$.

For example, let X = <ABRACADABRA> and let Z = <AADAA>, then Z is a subsequence of X.



LCS Problem: Given two sequences $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$ determine the length of their longest common subsequence, and more generally the sequence itself.