Knapsack Problem

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.

Ex: { 3, 4 } has value 40.

W = 11

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

Greedy: repeatedly add item with maximum ratio v_i / w_i.
Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Dynamic Programming: False Start

Def. OPT(i) = max profit subset of items 1, …, i.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 }

- Case 2: OPT selects item i.
  - accepting item i does not immediately imply that we will have to reject other items
  - without knowing what other items were selected before i, we don't even know if we have enough room for i

Conclusion. Need more sub-problems!

Dynamic Programming: Adding a New Variable

Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

- Case 1: OPT does not select item i.
    - OPT selects best of { 1, 2, …, i-1 } using weight limit w

- Case 2: OPT selects item i.
    - new weight limit = w – w_i
    - OPT selects best of { 1, 2, …, i–1 } using this new weight limit

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w), \; v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

Knapsack Problem: Bottom-Up

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

```
Input: n, w₁,…,w_N, v₁,…,v_N

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

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

return M[n, W]
```

Knapsack Algorithm

W + 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

n + 1

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.

Sequence Alignment

String Similarity

How similar are two strings?

- **ocurrance**
- **occurrence**

| o | c | u | r | r | a | n | c | e | - |

| o | c | c | u | r | r | e | n | c | e |

5 mismatches, 1 gap

| o | c | - | u | r | r | a | n | c | e |

| o | c | c | u | r | r | e | n | c | e |

1 mismatch, 1 gap

| o | c | - | u | r | r | - | a | n | c | e |

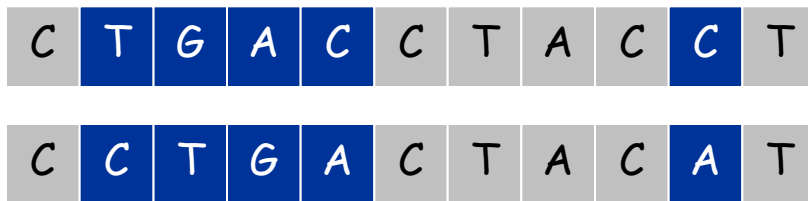| o | c | c | u | r | r | e | - | n | c | e |

0 mismatches, 3 gaps
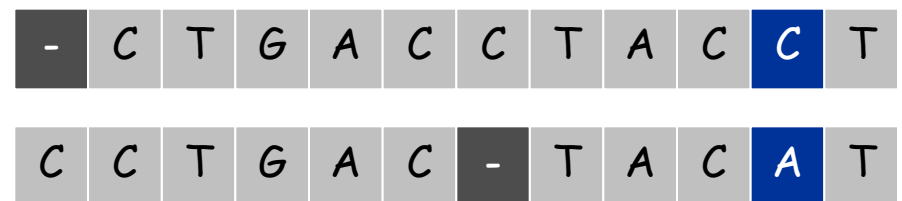
Edit Distance

Applications.

- Basis for Unix diff.
- Speech recognition.
- Computational biology.

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty δ; mismatch penalty α_{pq}.
- Cost = sum of gap and mismatch penalties.

C	T	G	A	C	C	T	A	C	C	T
C	C	T	G	A	C	T	A	C	A	T

$$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$$

-	C	T	G	A	C	C	T	A	C	C	T
C	C	T	G	A	C	-	T	A	C	A	T

$$2\delta + \alpha_{CA}$$

Sequence Alignment

Goal: Given two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$ find alignment of minimum cost.

Def. An alignment M is a set of ordered pairs x_i-y_j such that each item occurs in at most one pair and no crossings.

Def. The pair x_i-y_j and $x_{i'}$-$y_{j'}$ cross if i < i', but j > j'.

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Ex: `CTACCG` vs. `TACATG`.

Sol: $M = x_2\text{-}y_1, x_3\text{-}y_2, x_4\text{-}y_3, x_5\text{-}y_4, x_6\text{-}y_6.$

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G

	x_2	x_3	x_4	x_5	x_6	
-	T	A	C	A	T	G

| | y_1 | y_2 | y_3 | y_4 | y_5 | y_6 |

Sequence Alignment: Problem Structure

Def. OPT(i, j) = min cost of aligning strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.
- Case 1: OPT matches x_i-y_j.
  - pay mismatch for x_i-y_j + min cost of aligning two strings
    $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$
- Case 2a: OPT leaves x_i unmatched.
  - pay gap for x_i and min cost of aligning $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_j$
- Case 2b: OPT leaves y_j unmatched.
  - pay gap for y_j and min cost of aligning $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_{j-1}$

$$OPT(i,\ j) = \begin{cases} j\delta & \text{if } i=0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1,\ j-1) \\ \delta + OPT(i-1,\ j) \\ \delta + OPT(i,\ j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j=0 \end{cases}$$

Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x₁x₂...xₘ, y₁y₂...yₙ, δ, α) {
    for i = 0 to m
        M[0, i] = iδ
    for j = 0 to n
        M[j, 0] = jδ

    for i = 1 to m
        for j = 1 to n
            M[i, j] = min(α[xᵢ, yⱼ] + M[i-1, j-1],
                            δ + M[i-1, j],
                            δ + M[i, j-1])
    return M[m, n]
}
```

Analysis. $\Theta(mn)$ time and space.

English words or sentences: $m, n \leq 10$.

Computational biology: $m = n = 100,000$. 10 billions ops OK, but 10GB array?