Lecture 8

Feb 02, 2022

- Midsem schedule is out
- Will follow the institute directives
- Syllabus: everything till last lecture before midsem

Last time:

Dynamic Programming
- → Subset sum
- → Knapsack
- → Edit distance
- → Shortest path in graphs
  - → Dijkstra's algo
  - → Bellman-Ford
  - → Floyd-Warshall

Knapsack:

Input: n items, each with a price p_i, weight w_i – positive integers

Knapsack capacity: B

Output: Subset $S \subseteq [n]$ s.t $p(S) = \sum_{i \in S} p_i$ is max

subject to $w(S) = \sum_{i \in S} w_i \leq B$

– We will see a DP based algo that outputs the max price $p(S)$ (not the set itself)

Exercise: Adapt this algorithm to find the set S of items.

As discussed – optimal substructure

S – optimal solution for $\{(p_i, w_i) : i \in [n]\}$, B

Two cases.

1) $n \in S \Rightarrow S \setminus \{n\}$ is an optimal sol. for $\{(p_i, w_i) : i \in [n-1]\}$, $B - w_n$ $(w_n \leq B)$

2) $n \notin S \implies S$ is an optimal solution for $\{(p_i, w_i) : i \in [n-1]\}$, B.

So, we can organize the subproblems by pairs (m, b), $m \in [n]$, $b \in [B]$

Knapsack $(\{(p_j, w_j) : j \le m\}, b)$

subproblems $\le n \cdot B$. → not too large?

Recipe:

1) Recursive fn:

$f(m, b) = p(S)$ for set $S \subseteq [m]$ s.t $w(S) \le b$ and $p(S)$ is max.

(Last time, we tried defining this fn to output the set S itself. Now we just want the price)

(2) Base case

$$F(0,b) = 0$$
$$F(m,0) = 0$$

(3) Recursion $\quad m \geq 1,\ b \geq 1$

$$F(m,b) = \max\left(F(m-1,b),\ F(m-1,b-w_m) + p_m\right)$$

provided $w_m \leq b$

(4) Formally proving the recurrence. ✓

$$\left.\begin{array}{l} LHS \leq RHS \\ LHS \geq RHS \end{array}\right\} \Rightarrow LHS = RHS$$

Exercise:

5) Pseudocode:

1) Base cases:

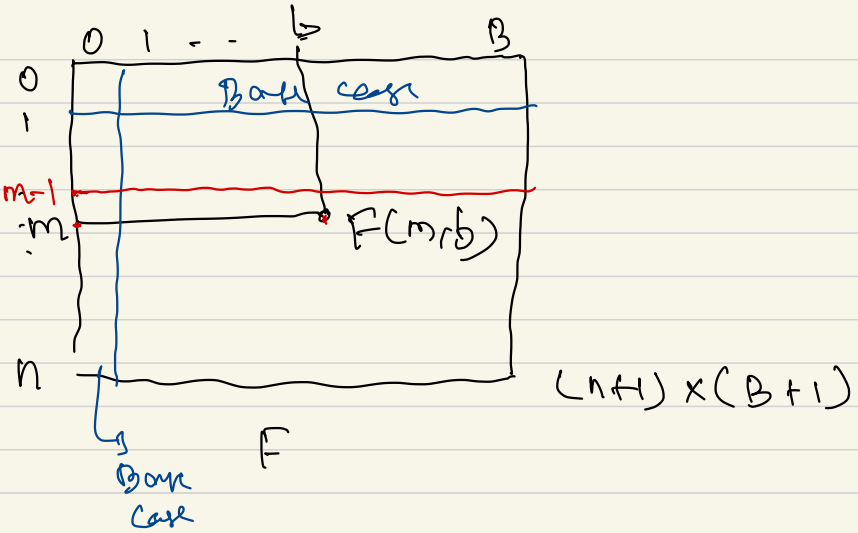
$$F[0,b] = 0 \quad \forall\, b \in [B]$$

$$F[m,0] = 0 \quad \forall\, m \in [n]$$

2) For $m \in \{1, \ldots, n\}$

For $b \in \{1, 2, \ldots, B\}$

$$\begin{cases} \text{if } b - w_m \geq 0 \\ \quad F(m,b) = \max\left(F(m-1,b),\ p_m + F(m-1, b-w_m)\right) \\ \text{else.} \quad F(m,b) = F(m-1,b) \end{cases}$$



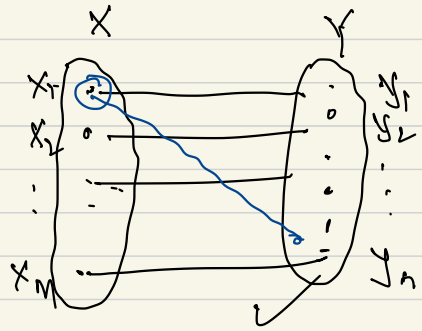
Running Time : $O(n.B)$

EDIT DISTANCE

Given two strings, $X[1:m]$, $Y[1:n]$. How 'close' or 'far' are X and Y?

Alignments of X, Y.

Matching



$$\begin{cases} X = x_1 x_2 \cdots x_m \\ Y = y_1 y_2 \cdots y_n \end{cases}$$

Alphabet $\Sigma = \{0,1\}$

$= \{A, \cdots, Z\}$

$= \{0, \cdots 9\}$

$x_i, y_i \in \Sigma$

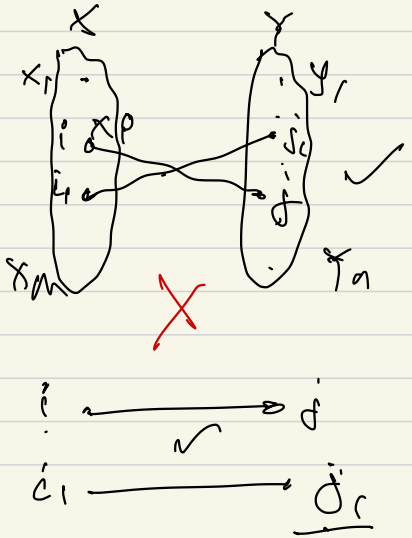
Matching: A matching M on a set V of vertices is a set of edges such that the degree of every vertex is at most 1.

Alignment: A matching M between X, Y is an alignment if the following situation does not happen.

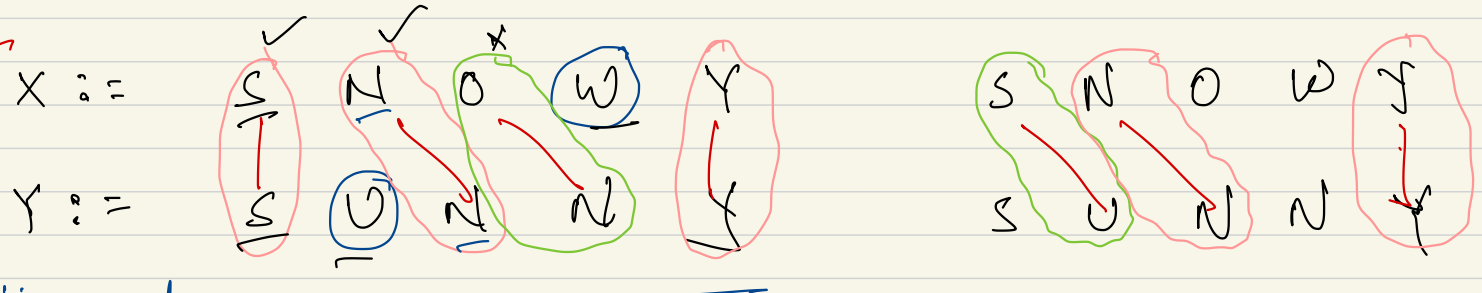
$(i,j), (i_1,j_1) \in M$

s.t

$i < i_1$ and $j > j_1$



$\Sigma = \{A, B, \ldots, Z\}$



Alignment:

1) Unmatched symbol/Letter: a position/letter that is not matched.

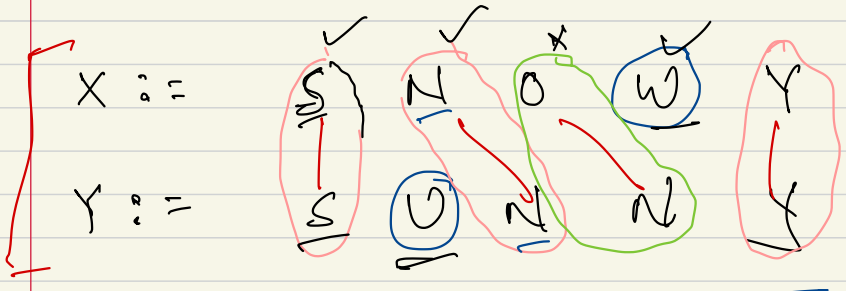
2) Good edge: Edge in the matching is good, if the alphabet symbol on the end points is the same

Else a Bad edge

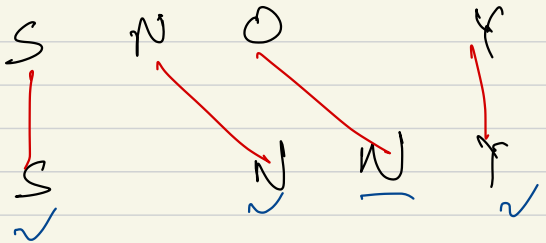
From alignment to editing

✓ ① Delete all unmatched symbols

✓ ② For every bad edge edit the label of one of the endpoints so that the labels are equal.



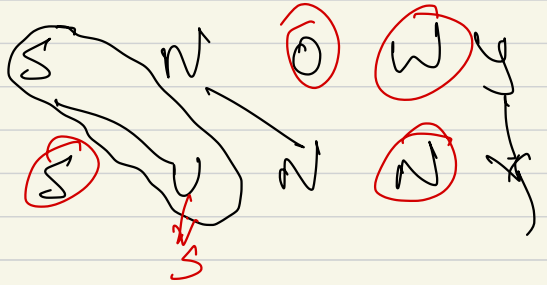
① Deleting unmatched symbols

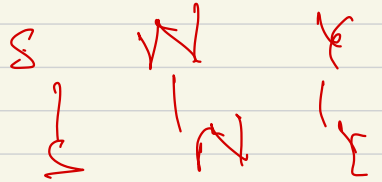


② Bad edges → good edges by editing the symbol at one of the end points.

Operations = 2 Deletions + 1 Relabeling = 3 ✓

Alignment 2





operations = 4 Deletions + 1 Relabeling = 5 operations

Alignment ⟺ A sequence of edit operations that makes the strings identical.

cost of an alignment = # unmatched symbols
+ # bad edges

Edit dist(X, Y) := minimum cost of an alignment between x and y.

EDIT DISTANCE PROBLEM

Input: strings X[1:m], Y[1:n] over some alphabet Σ.

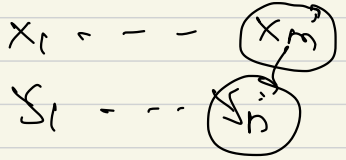
Output: Edit Distance of X, Y.

$= \min_{\text{alignment } A} \{ \text{cost}(A) \}$

$= \min_{\text{alignment } A} \{ \#\text{unmatched symbols} + \#\text{bad edges} \}$

Optimal substructure

A : optimal alignment between x, y (min cost)



What does A look like on the last symbols x_m, y_n ?

(1) $x_m - y_n$ } They are matched to each other.

(2) $x_m \longrightarrow y_j$. $j < n$
y_n is unmatched.

(3) x_m — unmatched
$x_i - y_n$ $i < m$

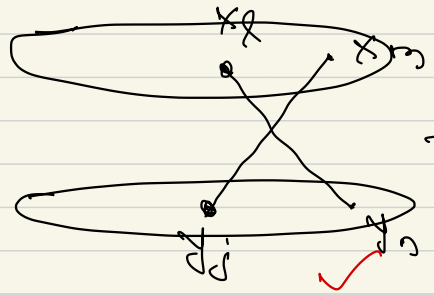
(3.5) both unmatched

} At least one of x_m, y_n is matched

(4) x_m — y_j $\quad j < n$

x_q — y_n $\quad q < m$

} Cannot happen.



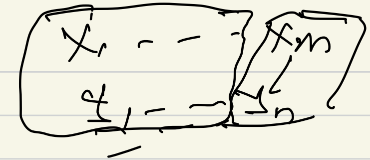
— contradicts that A is an alignment.

Lemma:
If A is an alignment, then one of the following two things happens:

1) x_m — y_n matched

2) At least one of x_m, y_n is unmatched.

case 1 $x_m - y_n$

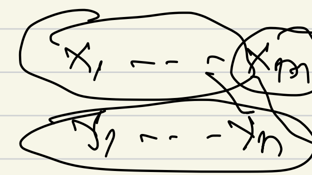


A is optimal for $x[1:m], y[1:n]$

then $A \setminus \{(x_m, y_n)\}$ is an optimal alignment for

$x[1:m-1], y[1:n-1]$.

Case 2.1 : x_m is unmatched



A is optimal for $x[1:m], y[1:n]$

then, A is optimal $x[1:m-1], y[1:n]$.

2.2 y_n is unmatched

A ~~~~~~ ~~~~~~

~~~~~~~~~~ $x[1:m], y[1:n-1]$.
~~~~~~~~~~

Subproblems:

Rec. fn $E(i,j)$ = Min cost of an alignment between $x[1:i]$, $y[1:j]$.

$i \in [m]$, $j \in [n]$

$m \times n$ subproblems

Base calls:

Exercise

Recurrence:

$$E(i,j) = \min \begin{cases} 1 + E[i-1, j] \\ 1 + E[i, j-1] \\ 1_{x_i \neq y_j} + E[i-1, j-1] \end{cases}$$

- $\rightarrow x_i$ is unmatched
- $\rightarrow y_j$ is unmatched
- $\rightarrow x_i - y_j$

$$\mathbb{1}_{x_i \neq y_j} = \begin{cases} 1 & \text{if } x_i \neq y_j \\ 0 & \text{o/w.} \end{cases}$$

Exercise: Write the pseudocode + prove correctness.