Lecture 25: Sequence Alignment

COSC 311 *Algorithms*, Fall 2022

Announcement

Homework 5 Posted

- 3 Questions
- Third question is challenge question

"Longest Inc. Subseq."

Overview

1. Finishing Knapsack Problem
2. Sequence Alignment

Knapsack Problem

Input:

1. A set R of n requests, each having
   - duration (weight) $\boxed{b_r}$ how long to service req. r
   - value v_r
2. Total time (weight) budget B

Output: A set S of requests to service with

1. sum of durations of requests in S is at most B
2. sum of values of requests is maximized

A Recurrence Relation

index of request

Idea. keep track of *remaining budget*

- if r_n is *not* serviced, remaining budget is B
- if r_n is serviced, remaining budget is $B - b_n$

Definition. For $j = 0, 1, \ldots, n$, $\boxed{\mathrm{opt}(j, C)}$ is optimal value of set of requests from $1, 2, \ldots, j$ with budget C.

Recurrence relation:

$$\mathrm{opt}(n, B) = \max(\boxed{\mathrm{opt}(n-1, B)}, \boxed{v_n + \mathrm{opt}(n-1, B - b_n)})$$

opt. if r_n not serviced

opt. if r_n is serviced

Computing Optimal Values

Assume. All durations b_i are integers at most B.

Compute. To compute opt(n, B):

- Generate a *two dimensional array* `max` where $\boxed{\texttt{max[j, C]}}$ stores the value opt(j, C)

Initialization. max[~~j~~ 0, ~~0~~ C] <- 0 for all ~~j~~ C

Apply Recursion Relation.

- `max[j, C] <- Max(max[j-1, C], v[j] + max[j-1, (C - b[j])])`

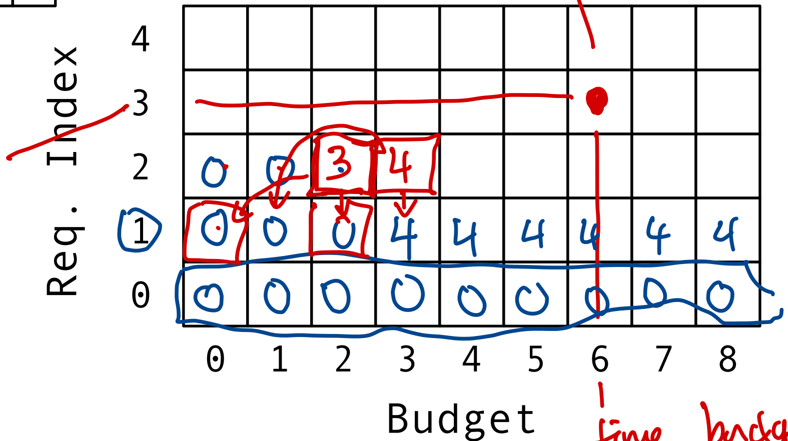
Picture

Requests: B=8

i	1	2	3	4
b[i]	3	2	5	4
v[i]	4	3	5	6

contains max profit (opt) for budget 6, servicing subset of first 3 requests

max index



fine budget

Requests: B=8

i	1	2	3	4
b[i]	3	2	5	4
v[i]	4	3	5	6

+5 = 8

Req. Index									
4									
3	0	0						8	
2	0	0	3	4	4	7	7	7	7
1	0	0	0	4	4	4	4	4	4
0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8

Budget

Requests: B=8

OPT(n,B)

i	1	2	3	4
b[i]	3	2	5	4
v[i]	4	3	5	6

Req. Index \ Budget	0	1	2	3	4	5	6	7	8
4 (n)	0	0	3	4	6	7	9	10	10
3	0	0	3	4	4	7	7	8	8
2	0	0	3	4	4	7	7	7	7
1	0	0	0	4	4	4	4	4	4
0	0	0	0	0	0	0	0	0	0

Budget (8 = B)

not same, so 4 is included

same so 3 not included

neither is 2

not same so 2 is included.

1, 4 is opt set.

Pseudocode

```
FindMax(R, n, B):
  max <- new 2d array of dimensions n+1, B+1
  set max[0, C] <- 0 for C = 0 to B
  for j from 1 to n
    (b, v) <- i-th request in R
    for C from 0 to B
      if b <= C then
        max[j, C] <- Max(v + max[j-1, C-b], max[j-1,C])
      else
        max[j, C] <- max[j-1, C]
  return max[n, B]
```

$O(n \cdot B)$

$O(B)$

B+1 iterations

n iter

Running time?

$O(Bn)$

Correctness

Claim. For all j and C, $\max[j, C] = \mathrm{opt}(j, C)$

Proof. Induction on j. ← max index considered

Base case $j = 0$. Optimal subset of size 0 has value 0.

Correctness

Claim. For all j and C, $\max[j, C] = \text{opt}(j, C)$

Proof. Induction on j.

Base case $j = 0$. Optimal subset of size 0 has value 0.

Inductive step $j \implies j + 1$.

ind. hyp is these are =

- suppose claim true for all $i \leq j$
- consider two possibilities:

1. request $j + 1$ is in optimal subset S

$$\text{opt}(j+1, C) = \boxed{v_{j+1}} + \boxed{\text{opt}(j, C - \boxed{b_{j+1}})} = v_{j+1} + \boxed{\max[j, C - b_{j+1}]}$$

2. request $j + 1$ is not in optimal subset S

$$\text{opt}(j+1, C) = \underbrace{\text{opt}(j, C) = \max[j, C]}_{= \text{ by ind. hyp.}}$$

Running Time?

```
FindMax(R, n, B):
  max <- new 2d array of dimensions n+1, B+1
  set max[0, C] <- 0 for C = 0 to B
  for j from 1 to n
    (b, v) <- i-th request in R
    for C from 0 to B
      if b <= C then
        max[j, C] <- Max(v + max[j-1, C-b], max[j-1,C])
      else
        max[j, C] <- max[j-1, C]
  return max[n, B]
```

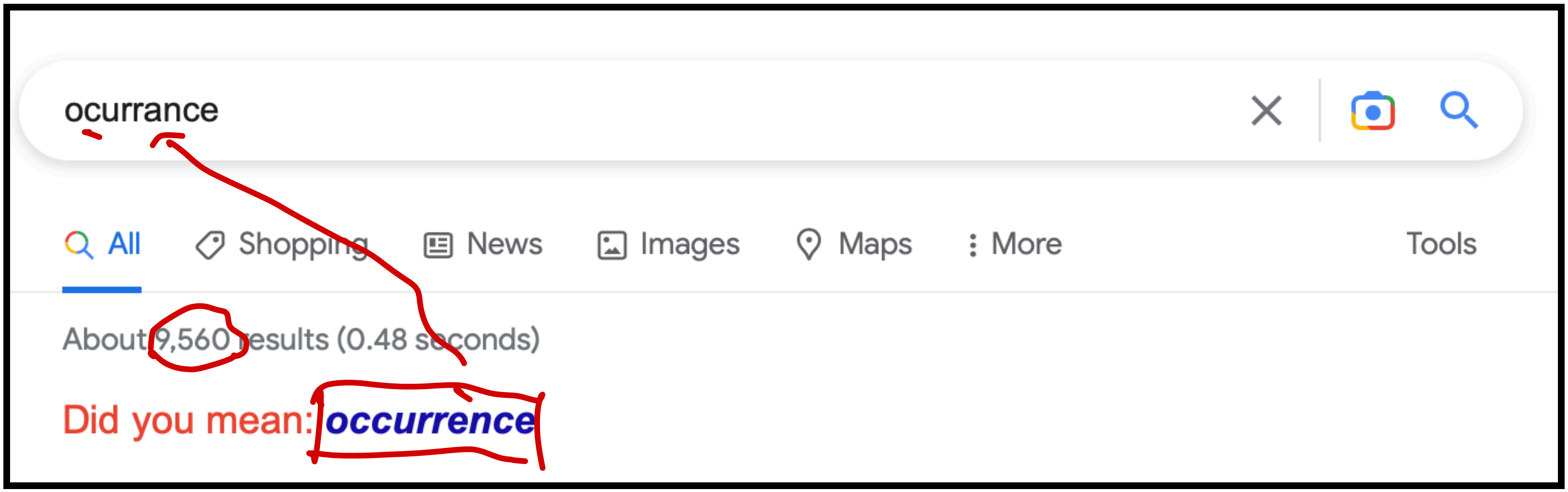
$O(Bn)$

Conclusion

For the knapsack problem with n requests and budget B, we can find compute $\text{opt}(n, B)$ in $O(Bn)$ time.

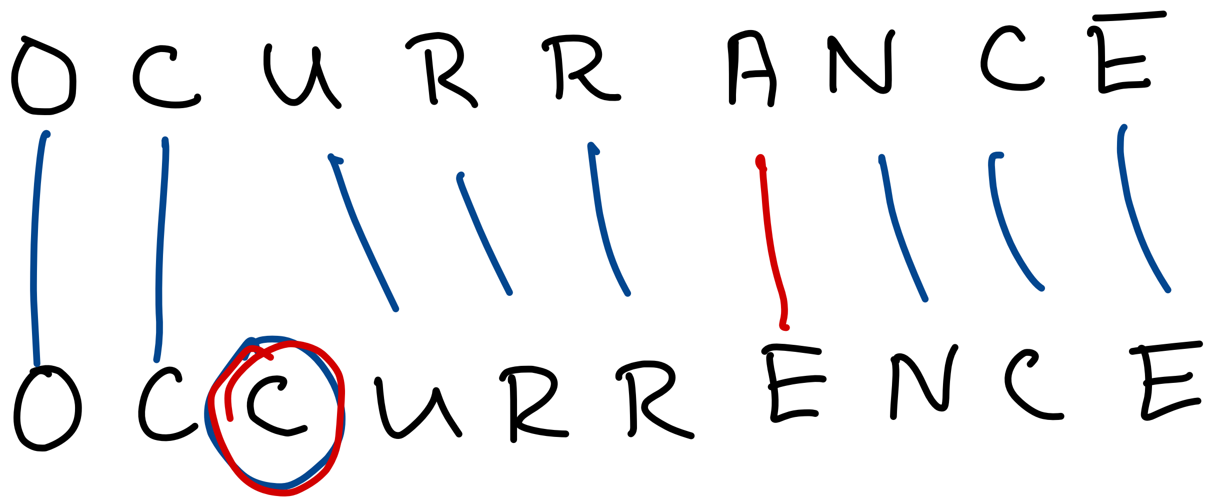
- *assuming* the duration of each request is an integer

Sequence Alignment



Question

How similar are the following strings?



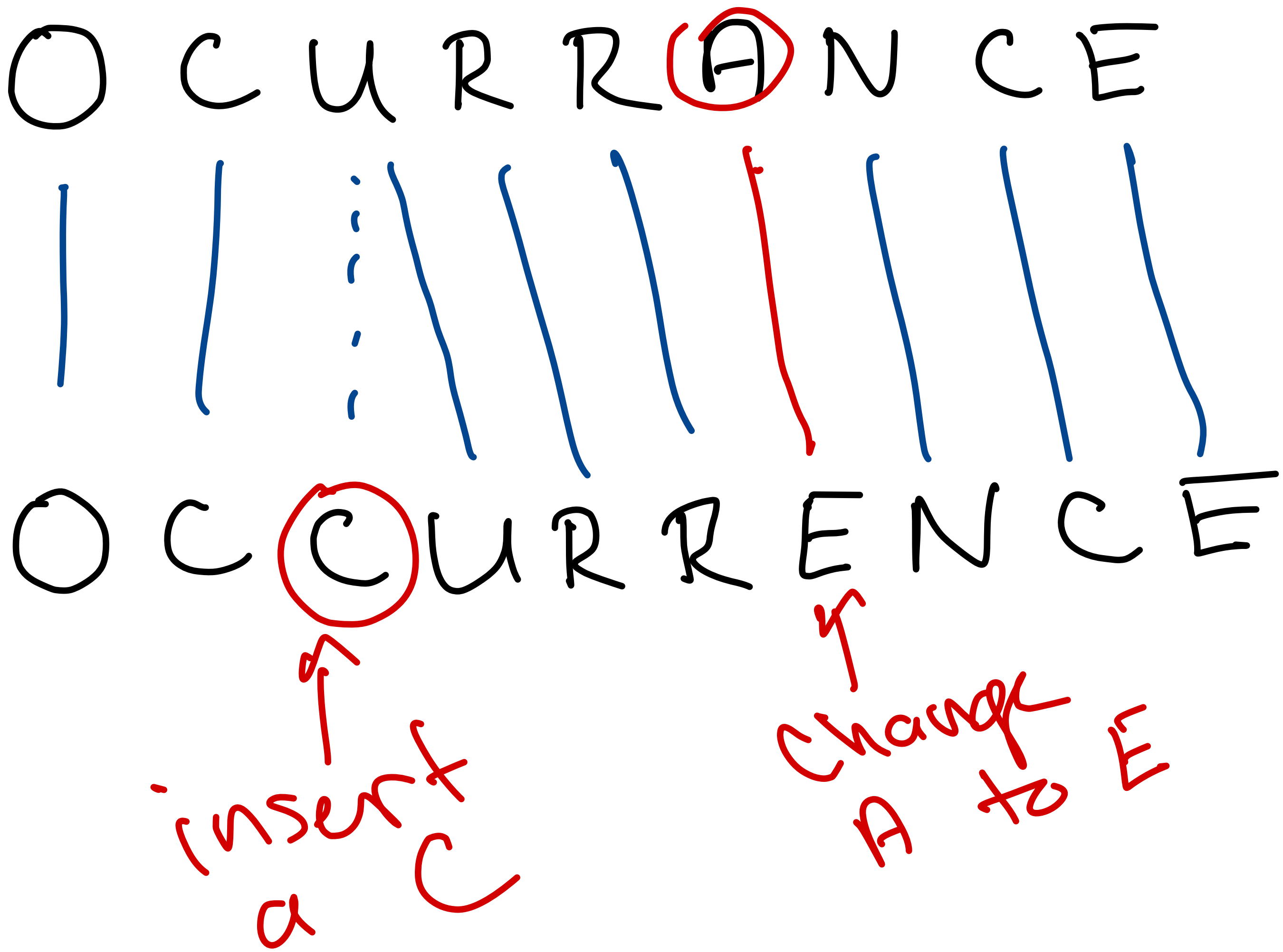
Hamming Distance

For how many indices do the strings *disagree*?

1	2	3	4	5	6	7	8	9	10
O	C	U	R	R	A	N	C	E	
O	C	C	U	R	R	E	N	C	E

(Dis)similarity and Alignment

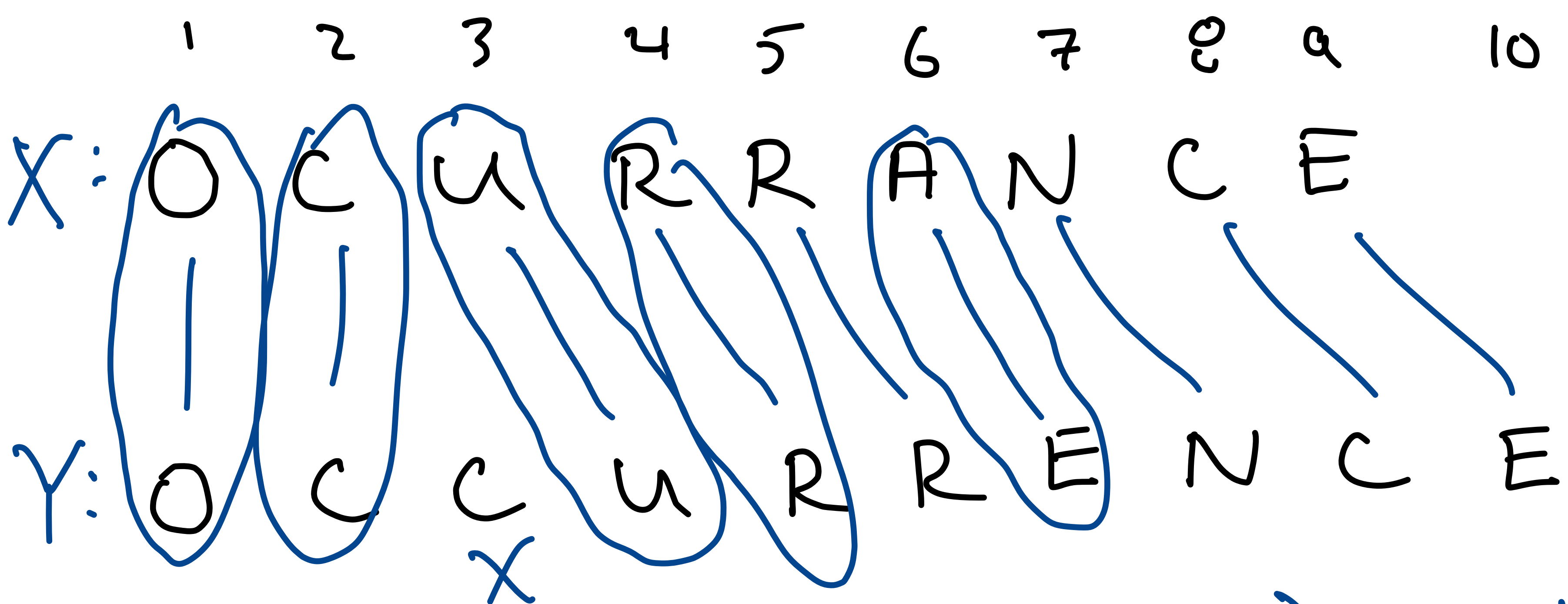
How could we *transform* one string into the other?



Optimal Alignment

Given two strings/arrays X and Y form a *matching* between characters

- matching M is a set of pairs of matched indices



$(1,1), (2,2), (3,4), \ldots, (9,10) = M$

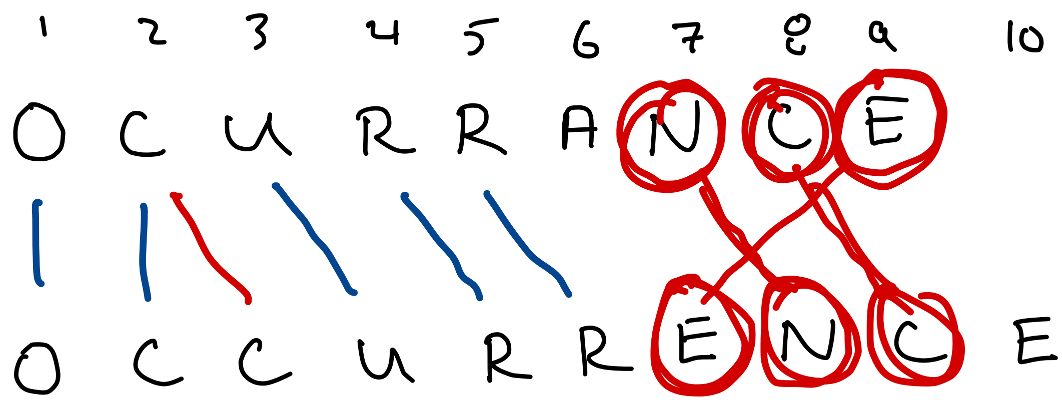
Optimal Alignment

Given two strings/arrays X and Y form a *matching* between characters

- matching M is a set of pairs of matched indices

Rules for matching:

- each character is matched with at most one other character
  - some characters may be unmatched
- matched characters cannot “cross”
  - if (i, j), (i', j') are matched with $i < i'$, then $j < j'$



Matching Penalties

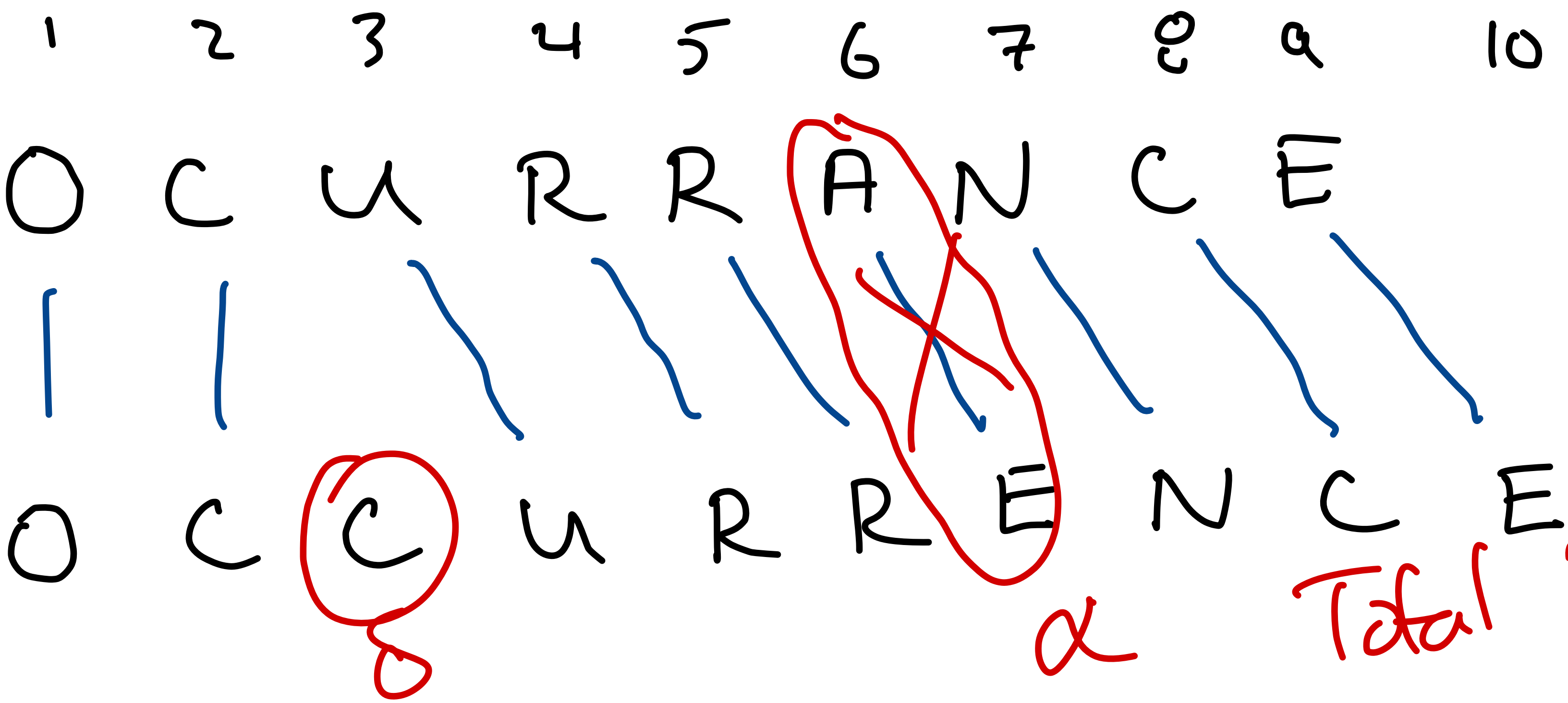
Given a matching M between strings X and Y

- incur penalty $\boxed{\delta}$ if an index i in X or Y is unmatched
- incur penality α if (i, j) matched, $X[i] \neq X[j]$

$\delta = \alpha = 1$

Total penalty is sum of individual penalties

Example.



Total penalty
$\delta + \alpha = 2$

Sequence Alignment Problem

Input:

- Sequences X and Y of characters of length n and m, respectively
- Penalties δ, α for omission/mismatch

Output:

- A matching M between indices of X and Y
- M minimizes total penalty of matching