LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

NHÓM 5

GV : Thầy Nguyễn Thanh Sơn

THÀNH VIÊN NHÓM

21522700 - Cáp Hữu Anh Trí

21521921 - Nguyễn Thành Đăng

21522153 - Nguyễn Hữu Huy

TABLE OF CONTENT

CÁC PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN ĐÃ HỌC ?

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

CÁC PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN

- Vét cạn (Brute Force)
- Quay lui (BackTracking)
- Nhánh và cận (Branch And Bound)
- Chia để trị (Divide And Conquer)
- Tham lam (Greedy Approach)
- Quy hoạch động (Dynamic Programming)

CÁC PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN

- **Vét cạn (Brute Force)** : Tạo ra hết tất cả các lời giải có thể có của một bài toán, và sau đó lựa chọn một giải pháp tốt nhất hoặc đếm số lượng lời giải phụ thuộc vào từng bài toán cụ thể.

=> Là một kỹ thuật tốt nếu dữ liệu đầu vào không quá lớn và đủ thời gian để đi qua hết tất cả các lời giải mà không tốn quá nhiều thời gian xử lý

- **Quay lui (BackTracking)** : Là một thuật toán được sử dụng trong lĩnh vực tìm kiếm và giải quyết vấn đề tổ hợp. Ý tưởng cơ bản của thuật toán quay lui là thử tất cả các khả năng có thể của một vấn đề bằng cách lần lượt đi từng bước và quay lại (backtrack) nếu không tìm được kết quả mong muốn. Thuật toán này thường được triển khai theo mô hình đệ quy.

CÁC PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN

- **Nhánh và cận (Branch And Bound) :** Là một phương pháp cải tiến từ phương pháp Quay lui, giải quyết vấn đề tối ưu bằng cách xây dựng một cây tìm kiếm và mở rộng các nhánh tiềm năng tốt hơn. Đánh giá cận được sử dụng để ước lượng giá trị tối ưu của các nhánh và cắt bỏ các nhánh không tiềm năng. Thuật toán lặp đi lặp lại cho đến khi tìm được giải pháp tối ưu hoặc không còn nhánh nào để khám phá
- **Chia để trị (Divide And ConQuer)** : Là một phương pháp giải quyết vấn đề bằng cách chia nhỏ vấn đề ban đầu thành các phần nhỏ hơn, giải quyết độc lập từng phần nhỏ đó, sau đó kết hợp các giải pháp con để đạt được giải pháp cho vấn đề ban đầu.

CÁC PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN

- **Tham lam (Greedy) :** Là một phương pháp giải quyết vấn đề bằng cách luôn lựa chọn lựa chọn tốt nhất hiện tại tại mỗi bước để hy vọng tìm được giải pháp tối ưu cho toàn bộ vấn đề. Thuật toán này không thực hiện việc đánh giá hoặc suy nghĩ về các lựa chọn tiếp theo, mà chỉ tập trung vào lựa chọn tối ưu tại từng bước.
- **Quy hoạch động (Dynamic Programming)** : Là một kỹ thuật giải quyết vấn đề bằng cách chia vấn đề thành các bài toán con nhỏ hơn và lưu trữ kết quả của các bài toán con đó để sử dụng lại trong quá trình giải quyết vấn đề lớn hơn. Kỹ thuật này thường được sử dụng khi các bài toán con có thể được giải quyết độc lập và có sự chồng chéo giữa chúng.

=> Mỗi thuật toán đều có những ưu và nhược điểm chung của chúng, vì vậy tùy thuộc vào yêu cầu của bài toán mà chọn ra phương pháp phù hợp nhất để giải quyết

BÀI TOÁN NỐI DÂY

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

BÀI TOÁN NỐI DÂY

- Cho n dây với độ dài khác nhau được lưu trữ trong mảng A[]. Nhiệm vụ của bạn là nối N sợi dây thành 1 sợi sao cho tổng chi phí nối dây là nhỏ nhất. Biết chi phí nối sợi dây thứ i và sợi dây thứ j là tổng độ dài 2 sợi dây A[i] và A[j]
- **INPUT:**
  - Dòng đầu tiên đưa vào số lượng bộ test T
  - Mỗi dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng : dòng thứ nhất đưa vào số lượng dây N, dòng tiếp theo đưa vào N số A[i] là độ dài của các sợi dây, các số được viết cách nhau một khoảng trống
  - T, N, A[i] thỏa mãn ràng buộc $1 \le T \le 100$, $1 \le N \le 10^6$, $0 \le A[i] \le 10^6$
- **OUTPUT:**
  - Đưa ra kết quả mỗi test theo từng dòng
- Ví dụ :

INPUT	OUTPUT
2 4 4 3 2 6 5 4 2 7 6 9	29 62

BÀI TOÁN NỐI DÂY

- **Đặt vấn đề :**
  - Nếu coi giá trị tổng hai dây là chi phí thì ta cần tìm tổng giá trị nhỏ nhất. Cần tìm ra 2 sợi dây nhỏ nhất trong số sợi dây để nối

BÀI TOÁN CÓ THỂ GIẢI QUYẾT BẰNG PHƯƠNG PHÁP NÀO ?

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

BÀI TOÁN NỐI DÂY

- **Ý tưởng :**
  - Thuật toán tham lam sử dụng hàng đợi ưu tiên
  - Tạo **pq** là hàng đợi ưu tiên lưu trữ độ dài n dây
  - Độ ưu tiên là độ dài của sợi dây (dây ngắn nhất sẽ được ưu tiên cao nhất)
  - Lấy 2 sợi dây ngắn nhất ra khỏi hàng đợi và tính tổng, sau khi nối 2 sợi dây rồi thì thêm lại vào hàng đợi ưu tiên
  - Lặp đến khi hết dây, trả về kết quả là chi phí nối dây nhỏ nhất

THAY VÌ DÙNG KĨ THUẬT HÀNG ĐỢI ƯU TIÊN CÓ THỂ DÙNG KĨ THUẬT NÀO KHÁC ?

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

BÀI TOÁN NỐI DÂY

- Có thể dùng phương pháp Sort thay vì dùng hàng đợi ưu tiên để có thể lấy ra 2 sợi dây ngắn nhất

=> Mỗi lần nối dây xong đều phải Sort lại => Độ phức tạp lớn

BÀI TOÁN NỐI DÂY

- **Cách hoạt động :**
  - Input: Số lượng dây 8, L = [9, 7, 12, 8, 6, 5, 14, 4]
  - Ouput: Chi phí nối dây nhỏ nhất

Bước	Giá trị First, Second	OPT	Trạng thái hàng đợi ưu tiên
		0	4, 5, 6, 7, 8, 9, 12, 14
1	First = 4; Second = 5	9	6, 7, 8, 9, **9**, 12, 14
2	First = 6; Second = 7	22	8, 9, 9, 12, **13**, 14
3	First = 8; Second = 9	39	9, 12, 13, 14, **17**
4	First = 9; Second = 12	60	13, 14, 17, **21**
5	First = 13; Second = 14	87	17, 21, **27**
6	First = 17; Second = 21	125	27, **38**
7	First = 27; Second = 38	190	**65**

=> Chi phí nối dây nhỏ nhất OPT = 190

BÀI TOÁN NỐI DÂY

- **Ta có đoạn mã giả :**

```
MinCost(L):
        pq= [] # Khởi tạo hàng đợi ưu tiên rỗng
 for i in (L):
        pq.enqueue(i) # Thêm dây vào hàng đợi ưu tiên
Total_cost = 0 # khởi tạo chi phí
While size(pq) > 1:
        r1 = dequeue(pq)
        r2 = dequeue(pq)
        Cost = r1 + r2
        Total_cost = Total_cost + cost
        enqueue(pq, cost) # Thêm dây mới nối vào hàng đợi
Return Total_cost
```

BÀI TOÁN NỐI DÂY

- **Độ phức tạp:**
  - Khởi tạo hàng đợi ưu tiên: Thêm n phần tử vào hàng đợi ưu tiên : O(n)
  - Xử lý các dây trong hàng đợi ưu tiên: Quá trình này tiếp tục cho đến khi chỉ còn một dây duy nhất trong hàng đợi, lặp lại quá trình này tối đa n-1 lần, nên độ phức tạp là O(nlog(n)), với log n là độ phức tạp của các phép toán enqueue và dequeue

=> **Độ phức tạp : O(nlog(n))**

BÀI TOÁN HOÁN VỊ

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

BÀI TOÁN HOÁN VỊ

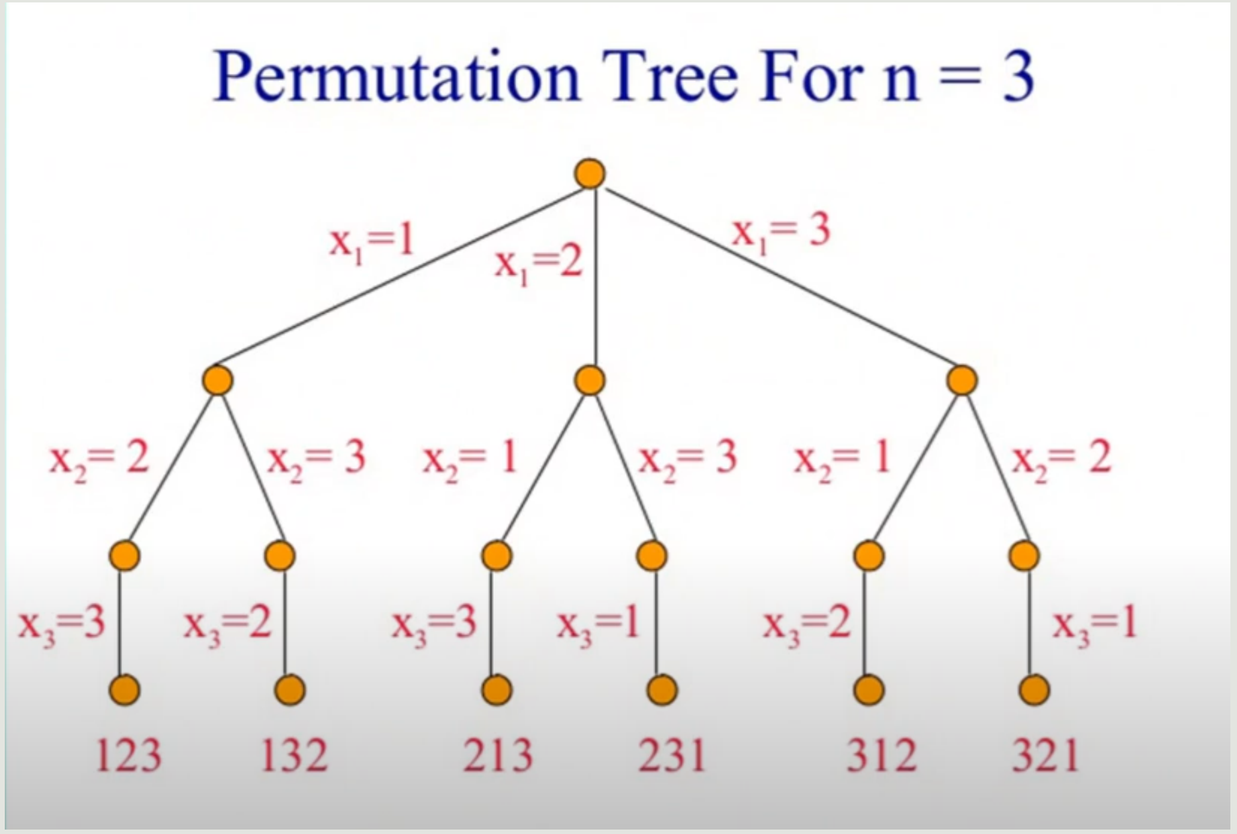
- Cho số nguyên dương N. Nhiệm vụ của bạn là hãy liệt kê tất cả các hoán vị của 1, 2, …. , N. Ví dụ với N = 3, ta có kết quả : 123, 132, 213, 231, 312, 321
- **INPUT:**
  - Dòng đầu tiên đưa vào số lượng bộ test T
  - Mỗi dòng tiếp theo đưa vào các bộ test là số nguyên dương N
  - T, N, thỏa mãn ràng buộc $1 \le T < 10$, $1 \le N < 10$
- **OUTPUT:**
  - Đưa ra kết quả mỗi test theo từng dòng
- Ví dụ :

INPUT	OUTPUT
2 2 3	12 21 123 132 213 231 312 321

BÀI TOÁN HOÁN VỊ

- **Ý tưởng :**
  - Sử dụng phương pháp **quay lui** để tìm các hoán vị.
  - **Xác định điều kiện dừng**: Khi đã có một hoán vị hoàn chỉnh, tức là đã chọn đủ các phần tử => in ra
  - **Chọn phần tử tiếp theo**: Bắt đầu từ vị trí đầu tiên, ta thử chọn từng phần tử chưa được chọn trước đó để điền vào vị trí hiện tại.
  - **Kiểm tra tính hợp lệ**: Trước khi chọn một phần tử, ta cần kiểm tra xem nó có hợp lệ hay không. Nếu phần tử đã được chọn trước đó, ta bỏ qua nó và thử các phần tử khác. Điều này đảm bảo rằng các phần tử trong hoán vị là duy nhất và không trùng lặp.
  - **Đánh dấu và đệ quy:** Khi đã chọn một phần tử hợp lệ, ta đánh dấu nó là đã được chọn và điền vào vị trí hiện tại trong hoán vị. Sau đó, ta đệ quy gọi thuật toán để tiếp tục điền các phần tử tiếp theo.
  - **Quay lui**: Sau khi hoàn thành quá trình đệ quy, ta quay lui bằng cách bỏ đánh dấu phần tử hiện tại và điều chỉnh vị trí để thử các phần tử khác.

BÀI TOÁN HOÁN VỊ



BÀI TOÁN HOÁN VỊ

- **Ta có đoạn mã giả :**

```
Try(int i):
    for j←1 to n #Duyệt phần tử
      if Check(j) ==0 #Kiểm tra phần tử đã được chọn chưa
            Check(j) = 1; # Chọn phần tử
            a[i] = j; # Lưu phần tử
            if(i == n) : print(a) # Điều kiện dừng : in ra hoán vị
            else Try(i+1) #Đệ quy
            Check(j) = 0 # Bỏ chọn, quay lui
```

BÀI TOÁN HOÁN VỊ

- **Độ phức tạp:**
  - Trong thuật toán quay lui, ta thử từng phần tử chưa được chọn trước đó để tạo hoán vị. Với mỗi vị trí trong hoán vị, ta có n phần tử có thể chọn. Do đó, số lần gọi đệ quy trong thuật toán là n, n-1, n-2,..., 2, 1, tức là n! lần.

=> **Độ phức tạp : O(n!)**

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Trong các thuật toán của bộ môn **Khoa học máy tính**, khái niệm **Khoảng cách Levenshtein** thể hiện khoảng cách khác biệt giữa 2 chuỗi ký tự. **Khoảng cách Levenshtein** giữa chuỗi S và chuỗi T là số bước ít nhất biến chuỗi S thành chuỗi T
- Khoảng cách này được đặt theo tên **Vladimir Levenshtein**, người đã đề ra khái niệm này vào năm 1965. Nó được sử dụng trong việc **tính toán sự giống và khác nhau** giữa 2 chuỗi

ĐỂ THỰC HIỆN BIẾN CHUỖI S THÀNH CHUỖI T THÌ TA CÓ CÁC PHÉP BIẾN ĐỔI CƠ BẢN NÀO ?

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Các phép biển đổi cơ bản :
  - Xóa một ký tự (**DELETE**)
  - Thêm một ký tự (**INSERT**)
  - Thay thế một ký tự này thành một ký tự khác (**REPLACE**)

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Cho hai xâu ký tự A, B bao gồm các ký tự in thường và các thao tác dưới đây :
  - **INSERT** : Chèn một ký tự bất kì vào A
  - **DELETE** : Loại bỏ một ký tự bất kì trong A
  - **REPLACE** : Thay thế một ký tự bất kì trong A
- Nhiệm vụ của bạn là đếm số các phép **INSERT, DELETE, REPLACE** ít nhất thực hiện trên A để trở thành B
- **INPUT:**
  - Dòng đầu tiên đưa vào số lượng bộ test T
  - Mỗi dòng tiếp theo đưa vào các bộ test. Mỗi bộ test là bộ đôi 2 xâu A, B
  - T, A, B thỏa mãn ràng buộc $1 \le T \le 100$, $1 \le length(A), length(B) \le 100$
- **OUTPUT:**
  - Đưa ra kết quả mỗi test theo từng dòng
- Ví dụ :

INPUT	OUTPUT
1 geek gesek	1

BÀI TOÁN CÓ THỂ GIẢI QUYẾT BẰNG PHƯƠNG PHÁP NÀO ?

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Trong bài này chúng ta sẽ thiết kế thuật toán dựa trên **giải thuật quy hoạch động**

- **Đặt vấn đề :** Liệu ta có thể tính khoảng cách ED giữa hai chuỗi A[1,2,..., n] và B[1,2,..., m] nếu ta biết khoảng cách giữa các chuỗi con A[1,2,..., n'] và B[1,2,..., m'] với m' ≤ m và n '≤ n, trong đó ít nhất một trong hai m' ≠ m hoặc n' ≠ n
- **Ý tưởng :** tính khoảng cách ED giữa A[1,2,..., n] và B[1,2,..., m] dựa vào các bài toán con nhỏ hơn.
- Gọi ED[m, n] là khoảng cách ED giữa hai dãy con A[1,2,..., n] và B[1,2,..., m].
- Nếu ta có hai chuỗi con A[1,2,..., n] và B[1,2,..., m], ta có ba lựa chọn để áp dụng ba thao tác với A[n]:
  - **Xóa A[n]**. Như vậy ED[m, n] = ED[m, n-1] + 1
  - **Chèn kí tự B[m] vào sau A[n]**. Như vậy ED[m, n]=ED[m-1, n] + 1
  - **Thay thế A[n] bằng B[m]**. Ta có hai trường hợp con ở đây: nếu A[n] ≠ B[m], khi đó ED[m, n] = ED[m−1,n−1] + 1. Ngược lại A[n]=B[m], ED[m, n] = ED[m−1,n−1]

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Ta có phương trình quy hoạch động :

$$S(A\text{->}B) = \min \begin{cases} S(A \text{ -> } B[0 \dots m\text{-}1]) + 1 : \text{tương ứng với thao tác INSERT} \\ S(A[0 \dots n\text{-}1] \text{ -> } B) + 1 : \text{ tương ứng với thao tác DELETE} \\ S(A[0 \dots n\text{-}1] \text{ -> } B[0 \dots m\text{-}1]) + COST : \text{ tương ứng với thao tác REPLACE} \end{cases}$$

Trong đó :

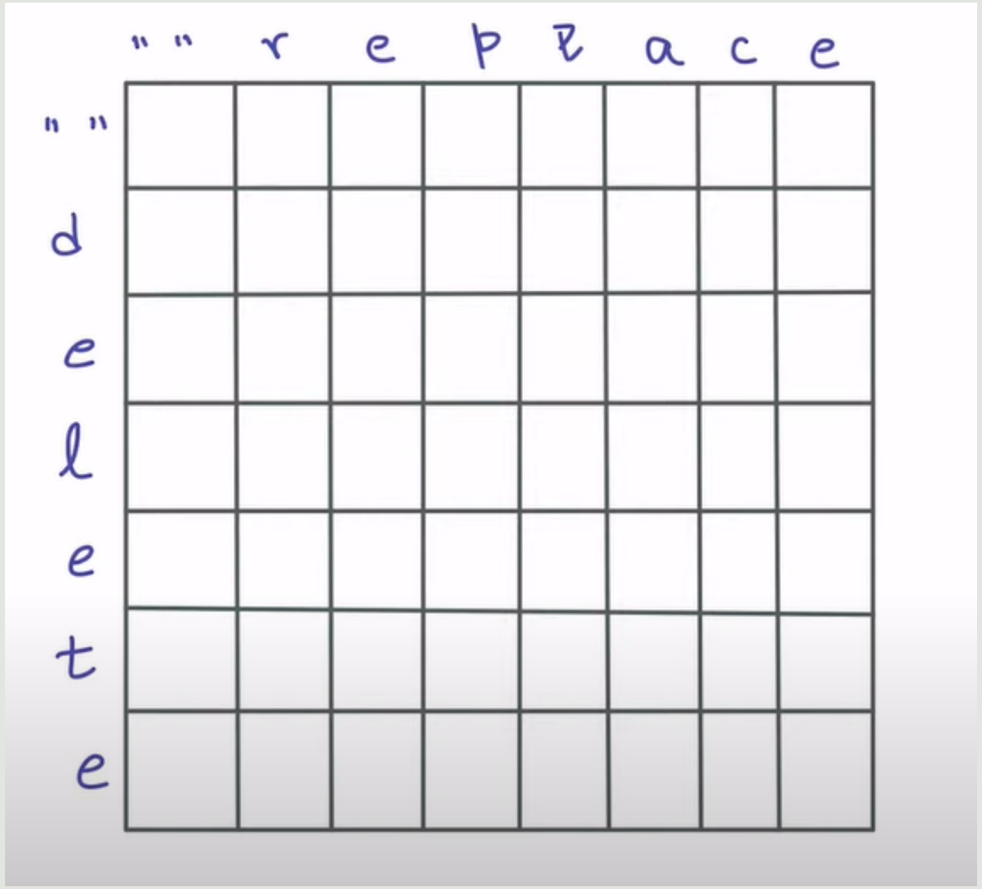
- Nếu A[n] = B[m] : COST = 0
- Ngược lại : COST = 1

- Ta thấy : Phần tử mảng ED(m,n) chỉ phụ thuộc vào 3 phần tử liền kề trong bảng. Như vậy để xác định ED(m,n), ta cần biết phần tử liền kề trái, liền kề phải và liền kề đường chéo

	n-1	n
m-1	REPLACE	INSERT
m	DELETE	ED(m,n)

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Ví dụ:
  - Chuỗi A : replace
  - Chuỗi B : delete
- Dùng quy hoạch động xây dựng mảng 2 chiều ED[][] với kích thước mxn

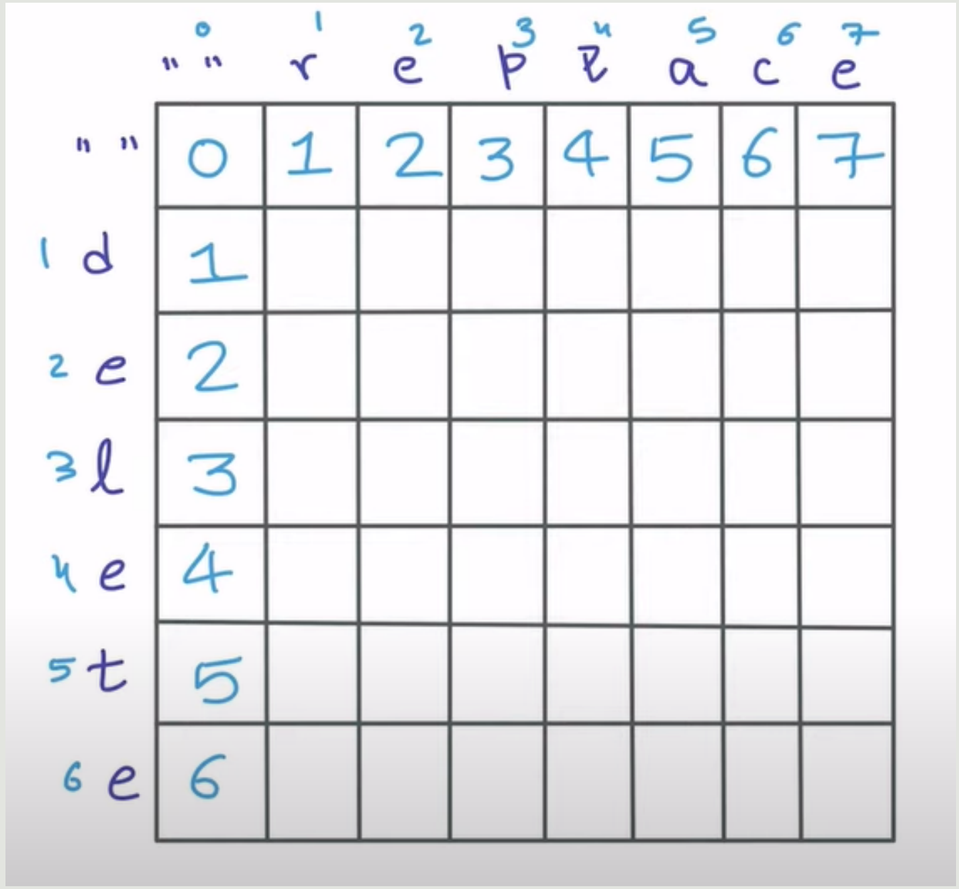


	n-1	n
m-1	REPLACE	INSERT
m	DELETE	ED(m,n)

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Ví dụ:
  - Chuỗi A : replace
  - Chuỗi B : delete

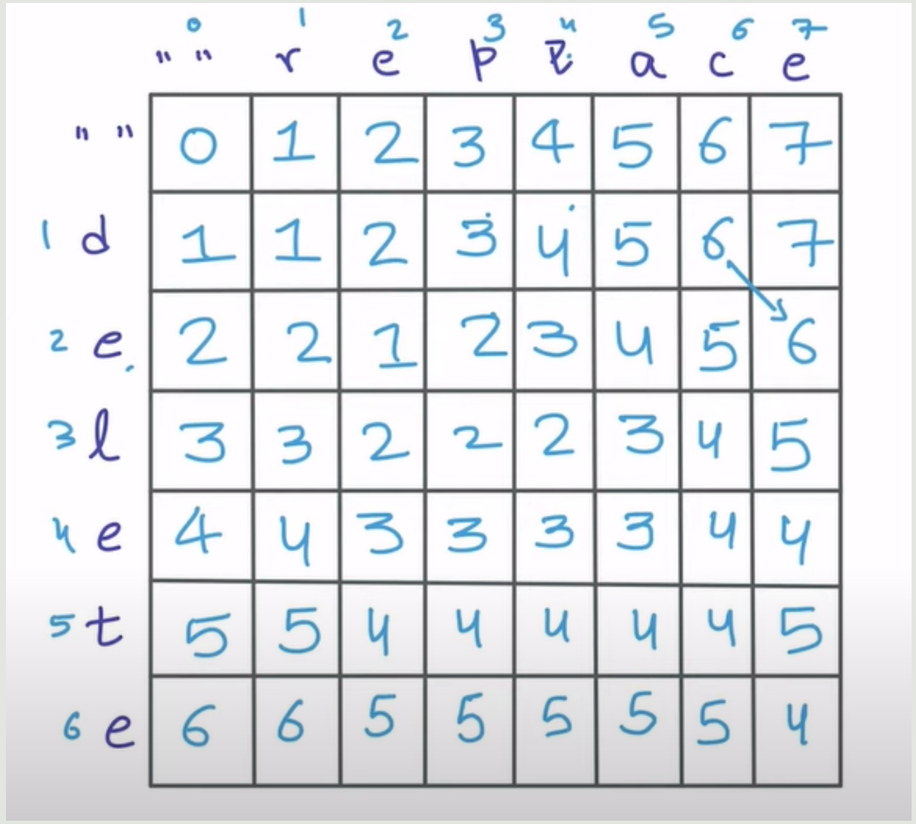
		n-1	n
m-1	REPLACE	INSERT
m	DELETE	ED(m,n)



BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- Ví dụ:
  - Chuỗi A : replace
  - Chuỗi B : delete

	n-1	n
m-1	REPLACE	INSERT
m	DELETE	ED(m,n)



=> Kết quả : ED(m,n) = 4

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- **Ta có đoạn mã giả :**

```
EditDistance(A[1,2,...,n],B[1,2,...,m]):
    for i←1 to n
      ED[0, i]←i
    for j←1 to m
      ED[j, 0]←j
    for j←1 to m
      for i←1 to n
          if A[i] = B[j] : cost ← 0
          else : cost ← 1
          ED[j, i]←min{ ED[j−1, i]+1, ED[j, i−1]+1, ED[j−1, i−1]+cost }
  return ED(m,n)
```

BÀI TOÁN KHOẢNG CÁCH LEVENSHTEIN

- **Độ phức tạp:**
  - Vòng lặp chạy theo length(A) : O(n)
  - Vòng lặp chạy theo length(B) : O(m)
  - Vòng lặp xét từng phần tử mảng ED (mxn) : O(m*n)

=> **Độ phức tạp : O(m*n)**

QUIZZ

LUYỆN TẬP THIẾT KẾ THUẬT TOÁN