DYNAMIC PROGRAMMING

Nhóm 9

Mai Trung Kiên - 20520066

Trần Hữu Khoa - 20520222

Nội dung trình bày

01. Bài toán mở đầu

Các cách giải quyết bài toán đã học và hạn chế của chúng

02. Giới thiệu

Giới thiệu quy hoạch động và các khái niệm liên quan

03. Nhận dạng bài toán

Đặc điểm của bài toán có thể áp dụng quy hoạch động

04. So sánh

So sánh quy hoạch động với chia để trị và tham lam

05. Ưu/Nhược điểm

Ưu và Nhược điểm của thuật toán quy hoạch động

06. Ứng dụng

Ứng dụng của thuật toán quy hoạch động

BÀI TOÁN MỞ ĐẦU

Bài toán mở đầu

- Cho ma trận vuông m * n, với các phần tử trong các ô nhận giá trị nguyên không quá 100000 giá trị tuyệt đối. Tìm đường đi từ ô (1,1) đến ô (m, n) sao cho tổng nhận được trên đường đi là lớn nhất.
- Quy tắc đi : ô (i, j) chỉ được đi đến ô (i+1, j) hoặc ô (i, j+1)

- Input : m, n, ma trận m*n.
- Output : tổng lớn nhất có thể.

Bài toán mở đầu

9	0	8	1	7
-2	-1	-2	-1	2
6	7	8	7	6
2	7	2	2	2
1	1	5	1	1
3	3	3	6	3
4	3	2	1	7

Các hướng giải quyết đã học

- Duyệt trâu ?
- Quay lui, nhánh cận ?

=>*Nhận xét : Luôn cho kết quả đúng, nhưng chậm.*

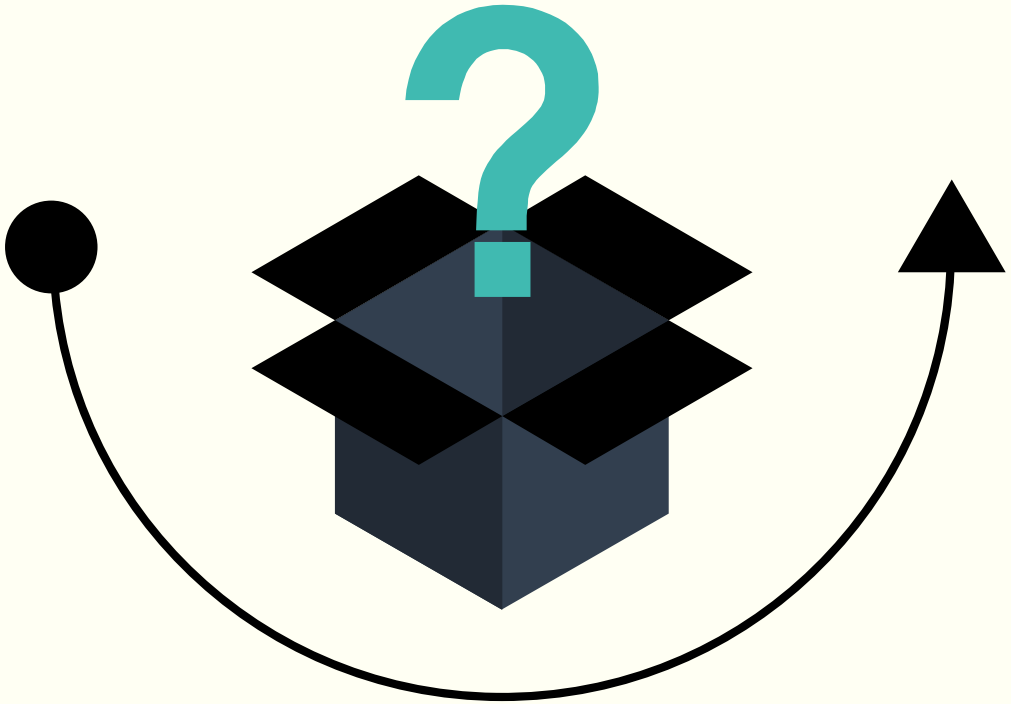
- Tham lam ?
- Thậm chí là random ?

=>*Nhận xét : Cho thời gian chạy tốt hơn. Kết quả có thể không là tốt nhất.*

Bài toán mở đầu

Vấn đề hiện tại

Các cách giải quyết đã học không đảm bảo về mặt thời gian hoạc độ chính xác



Phương án giải quyết?

Phương pháp đáp ứng các yêu cầu trên

=> *Quy hoạch động*

DYNAMIC PROGRAMMING

Giới thiệu

Quy hoạch động (Dynamic programming)

là một kỹ thuật nhằm đơn giản hóa việc tính toán các công thức truy hồi bằng cách lưu trữ toàn bộ hay một phần kết quả tính toán tại mỗi bước với mục đích sử dụng lại

_Giải thuật và lập trình - thầy Lê Minh Hoàng

Thường có 2 cách tiếp cận là **Top-Down** và **Bottom-Up**

Được phát minh vào 1950 bởi nhà Toán học người Mỹ Richard Ernest Bellman khi ông tìm kiếm phương pháp mới cho các bài toán tối ưu hoá

Một số khái niệm

- Bài toán giải theo phương pháp quy hoạch động gọi là **bài toán quy hoạch động**
- Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là **công thức truy hồi** của quy hoạch động
- Tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là **cơ sở quy hoạch động**
- Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là **bảng phương án của quy hoạch động**

Các bước thực hiện

- Giải tất cả các bài toán cơ sở (thông thường rất dễ), lưu các lời giải vào bảng phương án.
- Dùng công thức truy hồi phối hợp những lời giải của những bài toán nhỏ đã lưu trong bảng phương án để tìm lời giải của những bài toán lớn hơn và lưu chúng vào bảng phương án.
  Cho tới khi bài toán ban đầu tìm được lời giải.
- Dựa vào bảng phương án, truy vết tìm ra nghiệm tối ưu.

Sử dụng cho bài toán mở dầu

- Gọi f(i, j) là hàm trả về giá trị tối đa có thể nhận khi đi từ ô (0, 0) đến ô (i,j).

  -> kết quả cần tìm là f(m-1, n-1)

- Nhận xét : xét tổng quát, để đến (i, j) chỉ có thể trực tiếp đi qua (i-1,j) hoặc (i,j-1)

=> có thể tính f(i, j) từ việc tính f(i-1,j) và f(i,j-1)

Sử dụng cho bài toán mở dầu

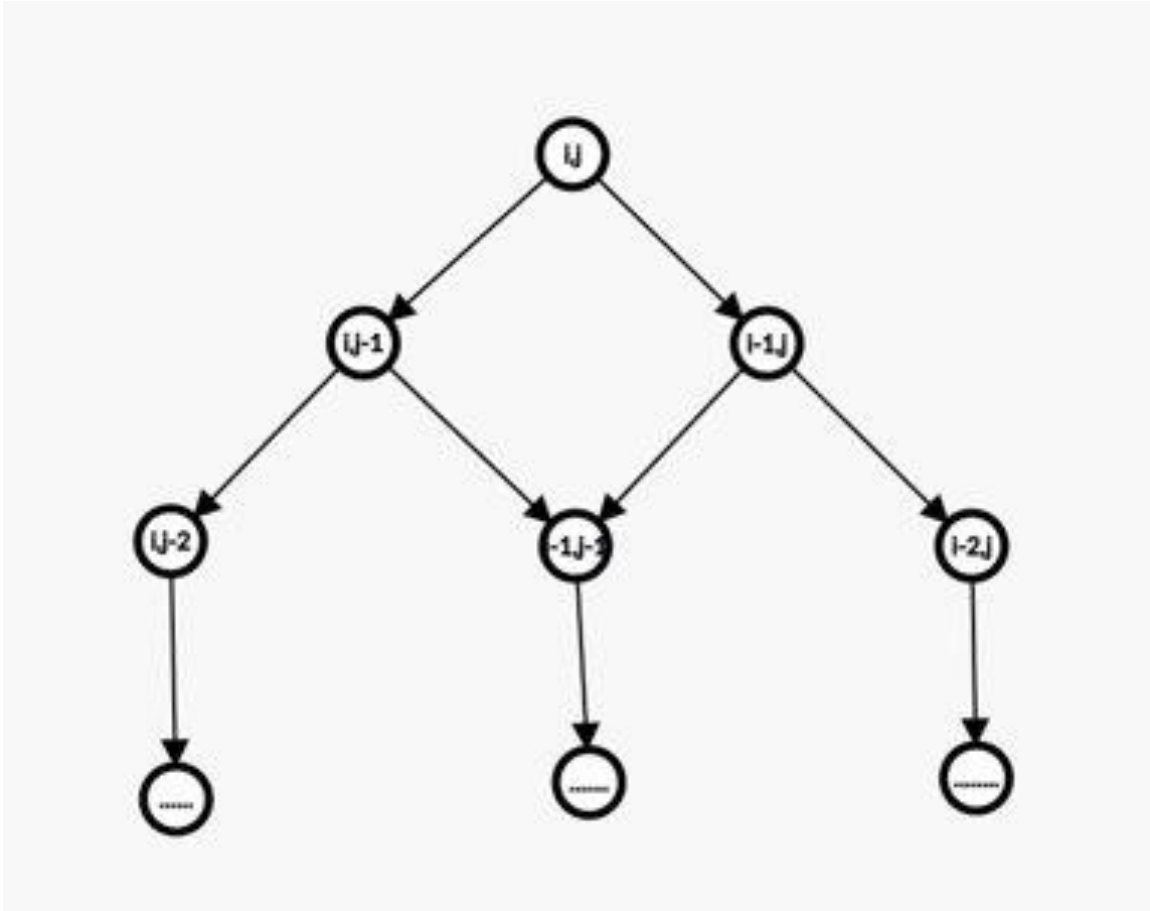
- Cơ sở quy hoạch động

  $f(i,j) = -\infty \ \forall\ i, j \in Z$

  $f(0,0) = matrix[0,0]$

- Công thức truy hồi :

  $f(i,j) = max(f(i-1,j), f(i,j-1)) + matrix[i, j]$



Code

Độ phức tạp ***O(m * n)***

```python
m = len(matrix)
n = len(matrix[0])

INF = 999999989999
dp = [[-INF]*n for i in range(m)]

def f(i, j):
  global dp
  global matrix
  global m, n

  # -f function define
  if (i<0 or i>=m or j<0 or j>=n):
    return -INF

  # -if dp[i][j] has been caculated, return it.
  if (dp[i][j] != -INF):
    return dp[i][j]

  # -else calculate f(i,j)
  dp[i][j] = matrix[i][j] + max(f(i-1, j), f(i, j-1))

  return dp[i][j]

dp[0][0] = matrix[0][0]

result = f(m-1, n-1)
```

BÀI TẬP

PALINDROME

PALINDROME

ĐỀ BÀI

Cho một chuỗi kí tự độ dài không quá 1000, chỉ chứa các kí tự latin thường {a,b,c,..,z} hãy **đếm xem có bao nhiêu xâu con là palindrome.**

Xâu gọi là palindrome khi và chỉ khi đảo ngược của nó bằng chính nó.

(vd : "aba" là palindrome, "abc" không là palindrome)

Xâu con ở đây gồm các kí tự không cần liên tiếp được tạo thành bằng cách xoá một hoặc nhiều kí tự từ xâu gốc.

(vd : "abc" là xâu con của "acbedc")

INPUT

Một dòng duy nhất chứa chuỗi kí tự gốc.

OUTPUT

Số lượng xâu con là palindrome.

PALINDROME

Ví dụ:

INPUT : IOICAMP

OUTPUT : 9

Xâu "IOICAMP" có 9 xâu con là palidrome : "I", "O", "I", "C","A","M","P","II", "IOI"

Nhận xét :

+ Xét(s) xâu con trong khoảng [i, j]. Nếu s[i] == s[j] thì tạo được thêm xâu palindrome từ các xâu con trong khoảng [i+1, j-1]

+ Nếu s[i] != s[j], không có xâu palindrome nào được tạo thêm

PALINDROME

Gọi **dp[i][j]** là số palindrome là xâu con của đoạn *[i, j].*

=> Kết quả bài toán là dp[0][len(s)-1]

Cơ sở quy hoạch động : dp[i][i] = 1

Công thức truy hồi:

$$dp[i][\text{j}]=\begin{cases} dp[i+1][j]+dp[i][j-1]+1 & \forall s[i]==s[j] \\ dp[i+1][j]+dp[i][j-1]-dp[i+1][\text{j-1}] & \forall s[\text{i}]\ !=\ \text{s}[j] \end{cases}$$

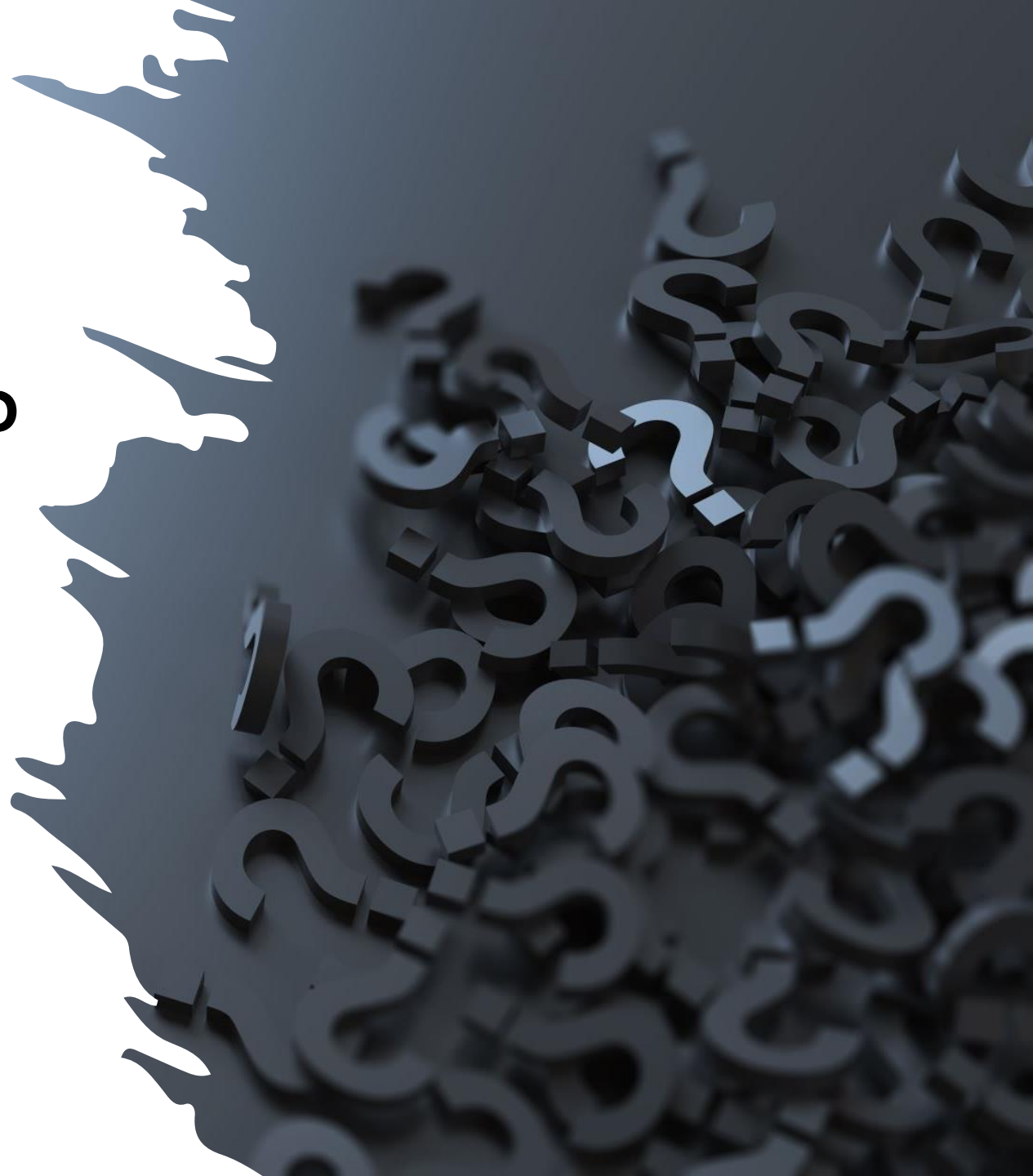
Code

Độ phức tạp $O(n^2)$

```python
s = input().strip()
n = len(s)
dp = [[0 for i in range(n+1)] for j in range(n+1)]

for i in range(n):
    dp[i][i] = 1

for len in range(2, n+1):
    for i in range(n):
        j = i + len - 1
        if (j < n):
            dp[i][j] = dp[i][j-1] + dp[i+1][j] - dp[i+1][j-1];
            if s[i] == s[j] :
                dp[i][j] = dp[i][j] + dp[i+1][j-1] + 1;

print(dp[0][n-1])
```

So, what 's điều kiện ?

Mọi problems đều có thể sử dụng ?



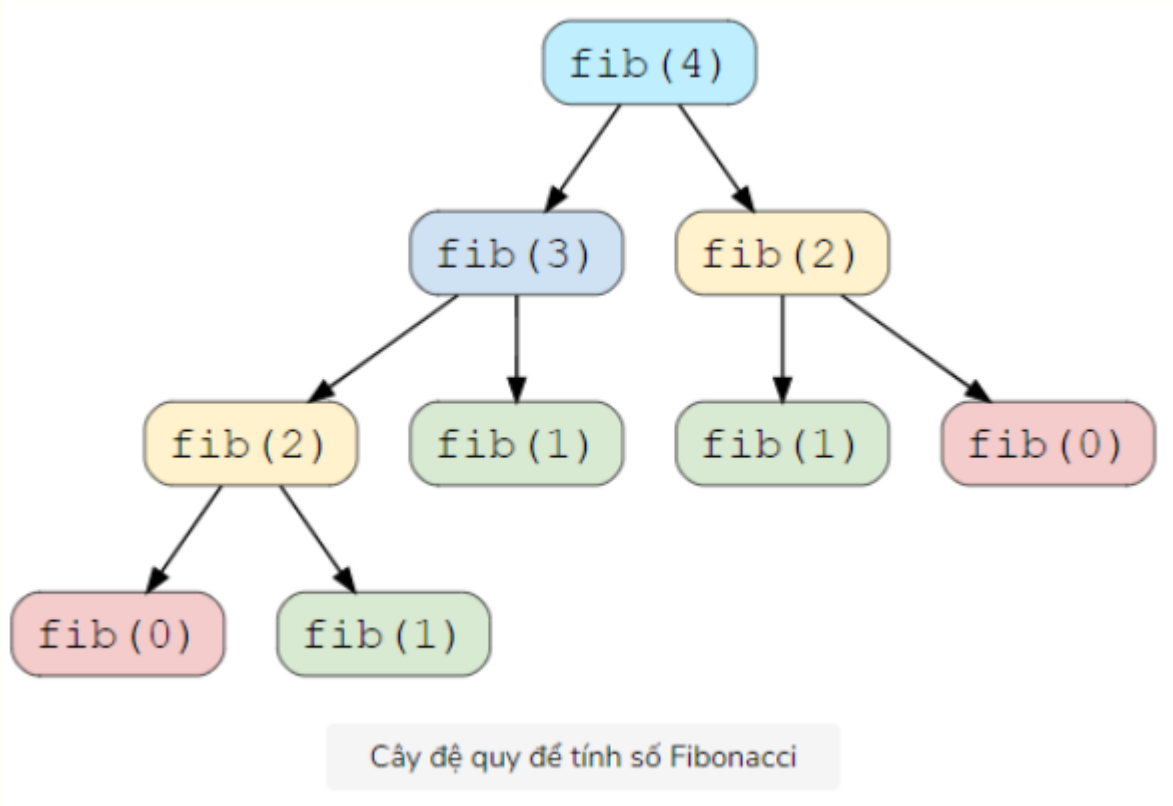
NHẬN DẠNG BÀI TOÁN

Các bài toán con gối nhau (overlapping subproblem)

Cấu trúc con tối ưu (optimal substructure)

Các bài toán con gối nhau (overlapping subproblem)

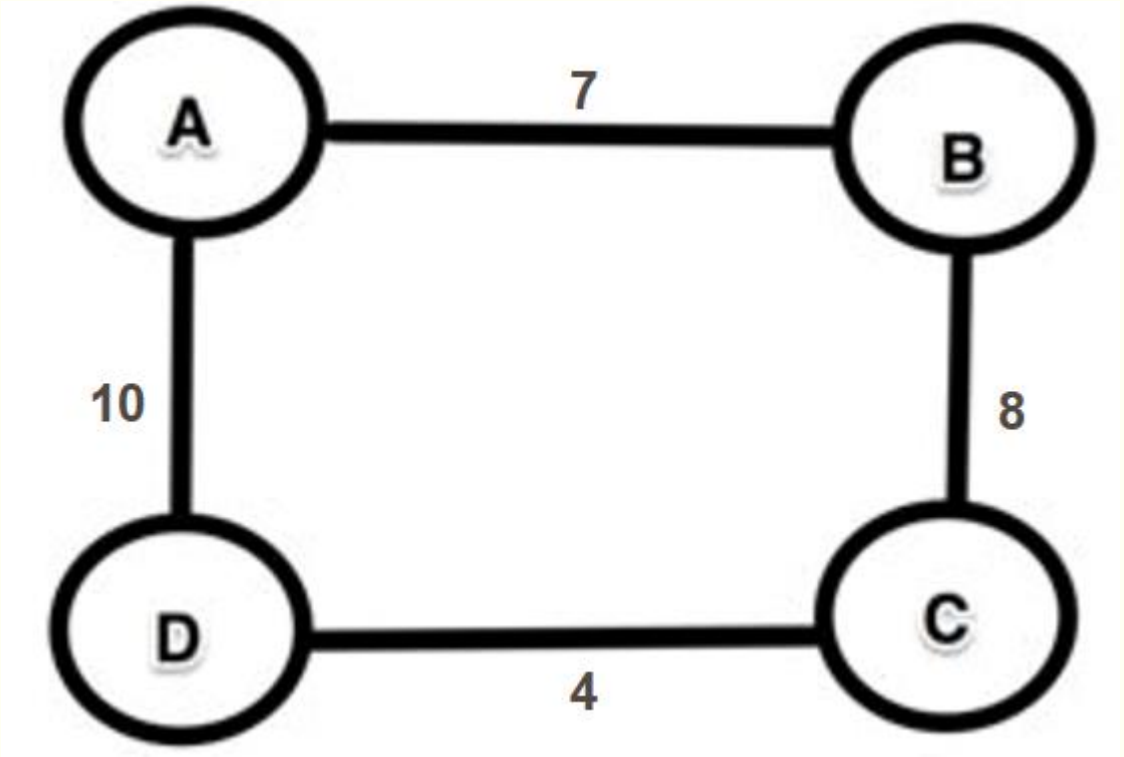
- Bài toán được chia thành các bài toán con
- Các bài toán con được tính đi tính lại nhiều lần



Cây đệ quy để tính số Fibonacci

Cấu trúc con tối ưu (optimal substructure)

- Nghiệm tối ưu của bài toán lớn là sự phối hợp các nghiệm tối ưu của bài toán con
- Không phải bài toán nào cũng có tính chất này



Shortest Path ?

Longest Path ?

BÀI TẬP

EXPR

EXPR

ĐỀ BÀI

Cho biểu thức $A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$
trong đó A_i là các số thực không âm và "·" là một phép toán "+" hoặc "*" cho trước. Hãy đặt các dấu ngoặc để biểu thức thu được có kết quả lớn nhất.

INPUT

Xâu kí tự chỉ chứa các số [0, 9], "+", "-"

OUTPUT

Kết quả lớn nhất thu được theo yêu cầu đề bài.

EXPR

Ví dụ:

INPUT : 1+3*2+1+2*5

OUTPUT : 100

Giải thích : (1+3)*(2+1+2)*5

Nhận xét:

+ Có thể chia nhóm $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$ thành 2 nhóm

$(A_i \cdot A_{i+1} \cdot \ldots \cdot A_k) \cdot (A_{k+1} \cdot A_{k+2} \cdot \ldots \cdot A_j)$

+ Vì "+" và "*" đồng biến, A_i không âm => $(A_i \cdot A_{i+1} \cdot \ldots \cdot A_j)$ đạt max khi $(A_i \cdot A_{i+1} \cdot \ldots \cdot A_k)$

và $(A_{k+1} \cdot A_{k+2} \cdot \ldots \cdot A_j)$ đạt max

EXPR

- Gọi dp[i][j] là giá trị lớn nhất có thể của biểu thức $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$

=> Kết quả bài toán là dp[1][n]

- Cơ sở quy hoach động : dp[i][i] = A[i]

- Công thức truy hồi : dp[i][j] = max(dp[i][k] · dp[k+1][j]) với k = i, i+1, …, j-1

Code

Độ phức tạp $O(n^3)$

```python
n = len(numList)

dp = [[0 for i in range(n + 1)] for j in range(n+1)]

for i in range(n):
    dp[i][i] = int(numList[i])

for len in range(2, n+1):
    for i in range(n):
        j = i + len - 1
        if j < n :
            for k in range(i,j):
                if operatorList[k] == '+':
                    dp[i][j] = max(dp[i][j], dp[i][k] + dp[k+1][j])
                if operatorList[k] == '*':
                    dp[i][j] = max(dp[i][j], dp[i][k] * dp[k+1][j])

print(dp[0][n-1])
```

SO SÁNH

Quy hoạch động và chia để trị

Chia để trị	Quy hoạch động
▪ Chia bài toán thành các bài toán con **độc lập** ▪ Không lưu lại kết quả của các bài toán con(Dẫn đến việc một vài bài toán con có thể được tính toán nhiều lần)	▪ Chia bài toán thành các bài toán con **gối nhau** ▪ Lưu lại kết quả của bài toán con(Nhờ đó mỗi bài toán con chỉ phải tính một lần)

Quy hoạch động và tham lam

Tham lam	Quy hoạch động
▪ Chỉ chọn ra giải pháp tốt nhất ở mỗi thời điểm nên không đảm bảo tìm ra giải pháp tối ưu	▪ Đảm bảo tìm ra giải pháp tối ưu dựa vào tính chất **cấu trúc con tối ưu**
▪ Thường tốn ít thời gian để thực hiện	▪ Thường mất nhiều thời gian hơn

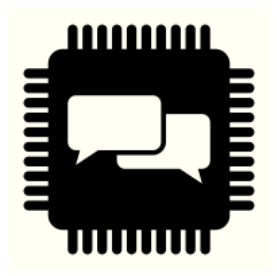
ƯU/NHƯỢC ĐIỂM

ƯU/NHƯỢC ĐIỂM

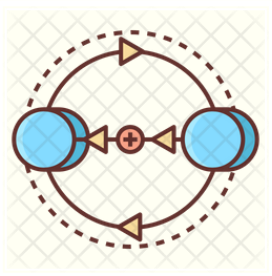
ƯU ĐIỂM	NHƯỢC ĐIỂM
▪ Tiết kiệm được thời gian do có lưu lại kết quả của các bài toán con ▪ Tìm ra được giải pháp tối ưu	▪ Tốn bộ nhớ để lưu lại kết quả của tất cả bài toán con ▪ Mỗi bài toán lại có một cách quy hoạch động khác nhau

ỨNG DỤNG

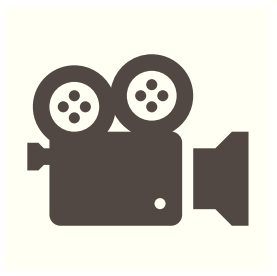
Ứng dụng



Minimum Edit Distance trong NLP



Bellman Equation trong RL



Video Summarization trong CV


Thank you!