TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

————-oOo————-

BÀI TẬP SỐ 04.e
THIẾT KẾ GIẢI THUẬT
PHƯƠNG PHÁP QUY HOẠCH ĐỘNG

Giảng viên hướng dẫn: ThS. Huỳnh Thị Thanh Thương

Nhóm sinh viên:

1. Phan Thanh Hải 18520705

TP. HỒ CHÍ MINH, 13/02/2020

Mục lục

Bài tập 3 . . . . . **2**

Bài tập 4 . . . . . **4**

Bài tập 5.1 . . . . . **6**

Bài tập 6.1 . . . . . **8**

Bài tập 6.2 . . . . . **11**

Bài tập 8 . . . . . **13**

Bài tập 9 . . . . . **15**

Bài tập 10 . . . . . **17**

Bài tập 11 . . . . . **19**

Bài tập 12 . . . . . **21**

Bài tập 13 . . . . . **23**

Bài tập 14 . . . . . **25**

Bài tập 15 . . . . . **27**

Bài tập 3

BÀI TOÁN NHÂN CHUỖI/DÃY MA TRẬN (MATRIX CHAIN MULTIPLICATION)

Consider the problem of minimizing the total number of multiplications made in computing the product of n matrices

$$A_1 \cdot A_2 \cdots A_n$$

whose dimensions are $d_0 \times d_1, d_1 \times d_2, \cdots, d_{n-1} \times d_n$, respectively. Assume that all intermediate products of two matrices are computed by the bruteforce (definition-based) algorithm.

Design a dynamic programming algorithm for finding an optimal order of multiplying n matrices.

a. Phát biểu bài toán

INPUT

Một chuỗi ma trận $A_1 \cdot A_2 \cdots A_n$, với A_i có kích thước là $d_{i-1} \times d_i$.

OUTPUT

Tổng số phép nhân thực hiện là ít nhất để nhân chuỗi ma trận trên.

b. Ý tưởng giải

Gọi $F(i, j)$ là số lượng phép nhân tối thiểu cho bài toán $A_i \cdot A_{i+1} \cdots A_j$. Để thiết lập công thức quy hoạch động cho $F(i, j)$, ta xét:

$A_i \cdot A_{i+1} \cdots A_j = (A_i \cdot A_{i+1} \cdots A_k) \cdot (A_{k+1} \cdot A_{k+2} \cdots A_j)$ $(i < j, i \leq k < j)$.

Bài toán $A_i \cdot A_{i+1} \cdots A_k$ cho kết quả là một ma trận có kích thước $d_{i-1}d_k$, tổng số lượng phép nhân tối thiểu là $F(i, k)$. Bài toán $A_{k+1} \cdot A_{k+2} \cdots A_j$ cho kết quả là một ma trận có kích thước $d_k d_j$, tổng số lượng phép nhân tối thiểu là $F(k + 1, j)$.

Hai ma trận ở trên nhân cho nhau thì tổng số lượng phép nhân là $F(i, k) + F(k + 1, j) + d_{i-1}d_k d_j$.

Để tổng số lượng phép nhân đó đạt tối thiểu thì

$F(i, j) = \min\{F(i, k) + F(k + 1, j) + d_{i-1}d_k d_j\}$ $(0 < i < j, i \leq k < j)$.

Nếu $i = j$ thì chỉ có một ma trận nên $F(i, j) = 0$.

Kết quả của bài toán gốc nằm ở $F(1, n)$.

c. Thuật giải

// Input: Kích thước của n ma trận được lưu trên mảng một chiều d.

// Output: Tổng số phép nhân thực hiện ít nhất để nhân một chuỗi các ma trận trên.

MatrixChainMultiplication$(d[1..n])$

```
1.  for i = 1 to n do
2.      F[i, i] = 0;
3.  for p = 1 to n − 1 do
4.      for i = 1 to n − p do
5.          j = i + p;
6.          F[i, j] = ∞;
7.          for k = i to j − 1 do
8.              temp = F[i, k] + F[k + 1, j] + d[i − 1] ∗ d[k] ∗ d[j];
9.              if temp < F[i, j]
10.                 F[i, j] = temp;
11. return F[1, n];
```

d. VD minh họa

INPUT	OUTPUT
2 2 3 3 4	24
6 30 35 35 15 15 5 5 10 10 20 20 25	15125
6 5 10 10 3 3 12 12 5 5 50 50 6	2010
4 14 14 14 2 2 4 4 5	572

e. Độ phức tạp của thuật toán

Độ phức tạp của hàm thời gian thực hiện chương trình là $O(n^3)$ (3 vòng lặp để tính hết giá trị của F).

Bài tập 4

CHUỖI CON CHUNG DÀI NHẤT (LONGEST COMMON SUBSEQUENCE)

Give two sequences $X = \langle x_1, x_2, \cdots, x_m \rangle$ and $Y = \langle y_1, y_2, \cdots, y_n \rangle$. Find a longest subsequence common to them both.

a. Phát biểu bài toán

INPUT

Hai chuỗi X và Y.

OUTPUT

Độ dài chuỗi con chung dài nhất của X và Y.

b. Ý tưởng giải

Gọi $F(i, j)$ là độ dài chuỗi con chung dài nhất của $X_i = \langle x_1, x_2, \cdots, x_i \rangle$ và $Y_j = \langle y_1, y_2, \cdots, y_j \rangle$. Để thiết lập công thức quy hoạch động cho $F(i, j)$, ta xét:

Gọi $Z_k = \langle z_1, z_2, \cdots, z_k \rangle$ là chuỗi con chung dài nhất của $X_i = \langle x_1, x_2, \cdots, x_i \rangle$ và $Y_j = \langle y_1, y_2, \cdots, y_j \rangle$.

Khi đó:

Nếu $x_i = y_j$ thì $z_k = x_i = y_j$ và Z_{k-1} là chuỗi con chung dài nhất của X_{i-1} và Y_{j-1}.

Nếu $x_i \neq y_j$ thì $z_k \neq x_i$ và Z_k là chuỗi con chung dài nhất của X_{i-1} và Y_j.

Nếu $x_i \neq y_j$ thì $z_k \neq y_j$ và Z_k là chuỗi con chung dài nhất của X_i và Y_{j-1}.

Do đó, $F(i, j) = F(i-1, j-1) + 1$ nếu $i = j$.

Hoặc, $F(i, j) = \max\{F(i, j-1), F(i-1, j)\}$ nếu $i \neq j$.

Nếu $i = 0$ hoặc $j = 0$ thì không thể có chuỗi con nên $F(i, j) = 0$.

Kết quả của bài toán gốc nằm ở $F(m, n)$.

c. Thuật giải

// Input: Hai chuỗi X và Y.

// Output: Độ dài chuỗi con chung dài nhất của X và Y.

LCSLength(X, Y)

1. $m = \text{length}(X)$;
2. $n = \text{length}(Y)$;
3. **for** $i = 1$ **to** m **do**
4. $\quad F[i, 0] = 0$;
5. **for** $j = 0$ **to** n **do**
6. $\quad F[0, j] = 0$;

```
7.  for i = 1 to m do
8.      for j = 1 to n do
9.          if X[i] = Y[i]
10.             F[i, j] = F[i − 1, j − 1] + 1;
11.         else
12.             if F[i − 1, j] ≥ F[i, j − 1]
13.                 F[i, j] = F[i − 1, j];
14.             else
15.                 F[i, j] = F[i, j − 1];
16. return F[m, n];
```

d. VD minh họa

INPUT	OUTPUT
ABCBDAB BDCABA	4
ABC DABEC	3
ABDE ABDE	4
AXE CBD	0

e. Độ phức tạp của thuật toán

Quá trình tính toán mỗi giá trị $F[i, j]$ có thời gian thực hiện là O(1).

Quá trình trên thực hiện $m \cdot n$ lần nên độ phức tạp của hàm thời gian thực hiện chương trình là O(mn).

Bài tập 5.1

TÌM ĐƯỜNG ĐI NGẮN NHẤT (SHORTEST PATH PROBLEM)

Cho một đồ thị có hướng $G = (V, E)$ với các cung được gán trọng số dương. Tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát $s \in V$ đến đỉnh cuối $f \in V$.

a. Phát biểu bài toán

INPUT

Đồ thị có hướng $G = (V, E)$ với các cung được gán trọng số dương.

Đỉnh xuất phát $s \in V$ đến đỉnh cuối $f \in V$.

OUTPUT

Độ dài nhỏ nhất của đường đi từ đỉnh xuất phát $s \in V$ đến đỉnh cuối $f \in V$ (nếu có).

b. Ý tưởng giải

Gọi $adj(i, j)$ là độ dài nhỏ nhất của đường đi xuất phát từ đỉnh $i \in V$ đến đỉnh $j \in V$.

Thay vì đi trực tiếp từ i đến j, ta tìm một đường đi gián tiếp đến đỉnh k ($i \to k \to j$) và nếu đường đi gián tiếp này nhỏ hơn đường đi trực tiếp từ i đến j thì ta gán luôn cho $adj[i, j]$.

Khi đó, $adj(i, j) = \min\{adj(i, j), adj(i, k) + adj(k, j)\}$.

Kết quả của bài toán gốc nằm ở $adj(s, f)$.

c. Thuật giải

// Input:

Ma trận kề adj lưu thông tin khoảng cách giữa các đỉnh.

Vị trí bắt đầu s và vị trí kết thúc f.

// Output: Độ dài của đường đi ngắn nhất đi từ vị trí bắt đầu s đến vị trí kết thúc f.

DuongDiNganNhat$(adj[1..V, 1..V], s, f)$

```
1.  for k = 1 to V do
2.      for i = 1 to V do
3.          for j = 1 to V do
4.              adj[i, j] = min{adj[i, j], adj[i, k] + adj[k, j]};
5.  return adj[s, f];
```

d. VD minh họa

Nếu không có đường đi từ điểm này đến điểm kia, ta sẽ điền vào ma trận kề giá trị 9999, giá trị này tương đương với ∞ khi tính toán.

INPUT	OUTPUT
4 0 9999 3 9999 2 0 9999 9999 9999 7 0 1 6 9999 9999 0 2 4	6
3 0 9999 3 2 9999 0 9999 9999 0 2 3	0
8 0 4 9999 9999 9999 9999 7 4 9999 0 9 9999 9999 6 8 1 9999 9999 0 9999 10 9999 9999 9999 9999 9999 9999 0 9999 9999 9999 9999 9999 9999 8 6 0 5 9999 9999 9999 9999 9999 9999 6 0 9999 9999 9999 4 9999 9999 9999 7 0 9999 9999 9999 3 9999 9999 9999 9999 0 1 4	22

e. Độ phức tạp của thuật toán

Độ phức tạp của hàm thời gian thực hiện chương trình là $O(V^3)$ (3 vòng lặp để tính hết giá trị của *adj*).

Bài tập 6.1

0/1 KNAPSACK PROBLEM

Cho n đồ vật và một cái ba lô có thể đựng trọng lượng tối đa M, mỗi đồ vật i có trọng lượng w_i và giá trị là p_i.

Chọn một cách lựa chọn các đồ vật cho vào túi sao cho trọng lượng không quá M và tổng giá trị là lớn nhất.

Mỗi đồ vật hoặc là lấy đi hoặc là bỏ lại.

a. Phát biểu bài toán

INPUT

Tập hợp n đồ vật bao gồm thông tin về giá trị và trọng lượng của chúng.

Trọng lượng tối đa của ba lô.

OUTPUT

Tổng giá trị của các đồ vật cho vào ba lô sao cho trọng lượng không quá M.

$\max \sum_{i=1}^{n} p_i x_i;$

thỏa mãn $\sum_{i=1}^{n} w_i x_i \leq M$ và $x_i \in \{0, 1\}$.

b. Ý tưởng giải

Gọi $F(i, j)$ là tổng giá trị tối đa của các đồ vật được cho vào ba lô có trọng lượng tối đa là j từ nhóm i đồ vật ban đầu. Để thiết lập công thức quy hoạch động cho $F(i, j)$, ta chia nhóm i đồ vật ban đầu ra thành 2 nhóm:

+ Nhóm 1: Nhóm không thể cho đồ vật thứ i vào trong ba lô được $(j - w_i < 0)$, khi đó $F(i, j) = F(i - 1, j)$ (trọng lượng tối đa của ba lô vẫn giữ nguyên).

+ Nhóm 2: Nhóm có thể cho đồ vật thứ i vào trong ba lô được $(j - w_i \geq 0)$, khi đó, nếu ta cho đồ vật thứ i vào trong ba lô thì trọng lượng tối đa hiện tại của ba lô là $j - w_i$ và giá trị của $F(i, j)$ là $v_i + F(i - 1, j - w_i)$.

Ta có công thức quy hoạch động của $F(i, j)$:

$$F(i,j) = \begin{cases} \max\{F(i-1,j), v_i + F(i-1, j-w_i)\} & j - w_i \geq 0 \\ F(i-1,j) & j - w_i < 0 \end{cases}$$

Hiển nhiên, ta cũng có:

$F(0, j) = 0, j \leq 0$ $\qquad\qquad$ $F(i, 0) = 0, i \leq 0.$

Kết quả của bài toán gốc nằm ở $F(n, M)$.

c. Thuật giải

// Input:

Giá trị p và trọng lượng w của n đồ vật tương ứng.

Trọng lượng tối đa của ba lô là M.

// Output: Tổng giá trị lớn nhất của các đồ vật cho vào ba lô sao cho trọng lượng không quá M.

Knapsack$(p[1..n], w[1..n], M)$

1. **for** $i = 0$ **to** n **do**
2. $\quad F[i, 0] = 0;$
3. **for** $j = 0$ **to** M **do**
4. $\quad F[0, j] = 0;$
5. **for** $i = 1$ **to** n **do**
6. $\quad$ **for** $j = 1$ **to** M **do**
7. $\quad\quad$ **if** $j - w[i] \geq 0$
8. $\quad\quad\quad F[i, j] = \max\{F[i-1, j], p[i] + F[i-1, j-w[i]]\};$
9. $\quad\quad$ **else**
10. $\quad\quad\quad F[i, j] = F[i-1, j];$
11. **return** $F[n, M];$

d. VD minh họa

INPUT	OUTPUT
3 1 4 2 5 3 1 4	3
4 10 5 40 4 30 6 50 3 10	90
5 33 15 24 20 36 17 37 8 12 31 80	130

e. Độ phức tạp của thuật toán

Độ phức tạp của hàm thời gian thực hiện chương trình là O(nM).

Bài tập 6.2

UNBOUNDED KNAPSACK PROBLEM

Cho một cái ba lô có thể đựng trọng lượng M với n loại đồ vật, mỗi đồ vật loại i có trọng lượng w_i và giá trị là p_i. Chọn một cách lựa chọn các đồ vật cho vào túi sao cho trọng lượng không quá M và tổng giá trị là lớn nhất.

Có thể chọn nhiều đồ vật cùng loại. Giả sử mỗi loại đồ vật không giới hạn về số lượng.

a. Phát biểu bài toán

INPUT

Tập hợp n đồ vật bao gồm thông tin về giá trị và trọng lượng của chúng.

Trọng lượng tối đa của ba lô.

OUTPUT

Tổng giá trị của các đồ vật cho vào ba lô sao cho trọng lượng không quá M.

$\max \sum_{i=1}^{n} p_i x_i$;

thỏa mãn $\sum_{i=1}^{n} w_i x_i \leq M$ và $x_i \geq 0$, x_i là một số nguyên.

b. Ý tưởng giải

Gọi $F(i)$ là tổng giá trị tối đa của các đồ vật được cho vào ba lô có trọng lượng tối đa là i.

Lúc đầu thì $F(i) = 0$.

Nếu đồ vật thứ j có thể cho vào ba lô được ($w_i \leq i$) thì ta có công thức quy hoạch động sau: $F(i) = \max\{F(i), F(i - w_j + p_i)\}$.

Kết quả của bài toán gốc nằm ở $F(M)$.

c. Thuật giải

// Input:

Giá trị p và trọng lượng w của n đồ vật tương ứng.

Trọng lượng tối đa của ba lô là M.

// Output: Tổng giá trị của các đồ vật cho vào ba lô sao cho trọng lượng không quá M.

Knapsack$(p[1..n], w[1..n], M)$

```
1.  for i = 0 to M do
2.      F[i] = 0;
3.  for i = 1 to M do
4.      for j = 1 to n do
5.          if w[i] ≤ i
6.              F[i] = max{F[i], F[i − w[j]] + p[i]};
7.  return F[M];
```

d. VD minh họa

INPUT	OUTPUT
3 1 4 2 5 3 1 4	12
2 1 1 30 50 100	100
4 10 1 40 3 50 4 80 5 8	120

e. Độ phức tạp của thuật toán

Độ phức tạp của hàm thời gian thực hiện chương trình là O(nM).

Bài tập 8

COIN-ROW PROBLEM

There is a row of n coins whose values are some positive integers $c_1, c_2, ..., c_n$, not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

a. Phát biểu bài toán

INPUT

Giá trị của n đồng xu.

OUTPUT

Số lượng lớn nhất các đồng xu có thể lấy từ n đồng xu ban đầu với điều kiện là không thể lấy 2 đồng xu ở 2 trí liên tiếp nhau.

b. Ý tưởng giải

Gọi $F(i)$ là số lượng đồng xu lớn nhất mà ta có thể lấy từ i đồng xu ở trước. Để thiết lập công thức quy hoạch động cho $F(i)$, ta xét:

+ Trường hợp 1: Nếu ta không lấy đồng xu thứ i thì $F(i) = F(i-1)$.

+ Trường hợp 2: Nếu ta lấy đồng xu thứ i thì khi đó, theo điều kiện của bài toán do đồng xu trước đó không thể lấy được nên $F(i) = c_i + F(i-2)$.

Ta có công thức quy hoạch động của $F(i)$:

$F(0) = 0$, $F(1) = c_1$.

$F(i) = \max\{c_i + F(i-2), F(i-1)\}$ với $i > 1$.

Kết quả của bài toán gốc nằm ở $F(n)$.

c. Thuật giải

// Input: Một mảng $C[1..n]$ lưu trữ giá trị của n đồng xu.

// Output: Số lượng lớn nhất các đồng xu có thể lấy từ n đồng xu ban đầu với điều kiện là không thể lấy 2 đồng xu ở 2 trí liên tiếp nhau.

CoinRow$(C[1..n])$

1. $F[0] = 0$;
2. $F[1] = C[1]$;
3. **for** $i = 2$ **to** n **do**
4. $\quad F[i] = \max(C[i] + F[i-2], F[i-1])$;
5. **return** $F[n]$;

d. VD minh họa

INPUT	OUTPUT
6 5 1 2 10 6 2	17
7 7 9 10 9 3 5 2	23
6 7 3 9 10 8 6	24

e. Độ phức tạp của thuật toán

Quá trình tính toán mỗi giá trị $F[i]$ có thời gian thực hiện là O(1).

Quá trình trên thực hiện $n + 1$ lần nên độ phức tạp của hàm thời gian thực hiện chương trình là O(n).

Bài tập 9

COIN-COLLECTING PROBLEM

Several coins are placed in cells of an $n \times m$ board, no more than one coin per cell. A robot, located in the upper left cell of the board, needs to collect as many of the coins as possible and bring them to the bottom right cell. On each step, the robot can move either one cell to the right or one cell down from its current location. When the robot visits a cell with a coin, it always picks up that coin.

Design an algorithm to find the maximum number of coins the robot can collect and a path it needs to follow to do this.

a. Phát biểu bài toán

INPUT

Một bảng kích thước $n \times m$ lưu trữ thông tin số lượng đồng xu tại mỗi ô trong bảng. Nếu tại ô c_{ij} có đồng xu thì $c_{ij} = 1$ và ngược lại thì $c_{ij} = 0$.

OUTPUT

Số lượng lớn nhất các đồng xu mà robot có thể lấy cho đến khi đi tới ô cuối cùng bên góc phải của bảng.

b. Ý tưởng giải

Gọi $F(i, j)$ là số lượng lớn nhất các đồng xu mà robot có thể lấy cho đến khi đi tới ô c_{ij}.

Để tới được ô c_{ij} thì ở bước trước đó, ô mà robot có thể đã đi qua là c_{i-1j} hoặc là ô c_{ij-1}.

Do đó, giá trị của $F(i, j)$ sẽ phụ thuộc vào $F(i-1, j)$ và $F(i, j-1)$ trước đó - nếu ô nào có đồng xu thì robot sẽ lấy đồng xu trong ô đó.

Ta có công thức quy hoạch động cho $F(i, j)$ là:

$F(0, j) = 0$ với $1 \leq j \leq m$ $\qquad$ $F(i, 0) = 0$ với $1 \leq i \leq n$.

$F(i, j) = \max\{F(i-1, j), F(i, j-1)\} + c_{ij}$ với $1 \leq j \leq m, 1 \leq i \leq n$.

Kết quả của bài toán gốc nằm ở $F(n, m)$.

c. Thuật giải

// Input: Một mảng $C[1..n, 1..m]$ lưu trữ thông tin số lượng đồng xu tại mỗi ô trong bảng. Nếu tại ô c_{ij} có đồng xu thì $c_{ij} = 1$ và ngược lại thì $c_{ij} = 0$.

// Output: Số lượng lớn nhất các đồng xu mà robot có thể lấy cho đến khi đi tới ô cuối cùng bên góc phải của bảng.

CoinCollecting$(C[1..n, 1..m])$

1. $F[1, 1] = C[1, 1]$;

2. **for** $j = 2$ **to** m **do**
3. $\quad F[1, j] = F[1, j-1] + C[1, j]$;
4. **for** $i = 2$ **to** n **do**
5. $\quad F[i, 1] = F[i-1, 1] + C[i, 1]$;
6. $\quad$ **for** $j = 2$ **to** m **do**
7. $\quad\quad F[i, j] = \max(F[i-1, j], F[i, j-1]) + C[i, j]$;
8. **return** $F[n, m]$;

d. VD minh họa

INPUT	OUTPUT
5 6 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 1 0	5
3 3 1 1 1 1 1 1 1 1 1	5
4 4 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0	4

e. Độ phức tạp của thuật toán

Quá trình tính toán mỗi giá trị $F[i, j]$ có thời gian thực hiện là O(1).

Quá trình trên thực hiện $n \cdot m$ lần nên độ phức tạp của hàm thời gian thực hiện chương trình là O(nm).

Bài tập 10

CHANGE-MAKING PROBLEM

Consider the general instance of the following well-known problem. Give change for amount n using the minimum number of coins of denominations $d_1 < d_2 < \cdots < d_m$.

Design a dynamic programming algorithm for the general case, assuming availability of unlimited quantities of coins for each of the m denominations $d_1 < d_2 < \cdots < d_m$ where $d_1 = 1$.

a. Phát biểu bài toán

INPUT

Các đồng xu được sắp xếp tăng dần theo giá trị tương ứng và đồng xu n cần đổi.

OUTPUT

Số lượng đồng xu cần đổi là nhỏ nhất.

b. Ý tưởng giải

Gọi $F(i)$ là số lượng đồng xu cần đổi là nhỏ nhất để đổi đồng xu i.

Gọi x là đồng xu đầu tiên trong đáp án tối ưu của bài toán. Khi đó, giá trị của đồng xu cần đổi còn lại là $i - x$.

Như vậy, ta có $F(i) = F(i - x) + 1$.

Vấn đề được đặt ra là ta không biết x là cái nào nên ta sẽ thử mọi khả năng có thể có của x để tính $F(i)$ và tìm giá trị của x tương ứng làm cho $F(i)$ nhỏ nhất.

Ta có công thức quy hoạch động cho $F(i)$ là:

$F(0) = 0$.

$F(i) = \min_{j:d_j \leq i}\{F(i - d_j)\} + 1$ với $i > 0$.

Kết quả của bài toán gốc nằm ở $F(n)$.

c. Thuật giải

// Input: Một mảng d lưu giá trị của các đồng xu sắp xếp theo thứ tự tăng dần và đồng xu n cần đổi.

// Output: Số lượng đồng xu cần đổi là nhỏ nhất.

ChangeMaking$(D[1..m], n)$

```
1.  F[0] = 0;
2.  for i = 1 to n do
3.      temp = ∞; j = 1;
4.      while j ≤ m and i ≥ D[j] do
5.          temp = min(F[i − D[j]], temp);
6.          j = j + 1;
```

7. $F[i] = temp + 1$;
8. **return** $F[n]$;

d. VD minh họa

INPUT	OUTPUT
3 1 3 4 6	2
3 1 3 5 9	3
3 1 5 10 12	3

e. Độ phức tạp của thuật toán

Quá trình tính toán điền giá trị của mỗi $F[i]$ (trừ $F[0]$) có thời gian thực hiện là $O(m)$.

Quá trình trên thực hiện n lần nên độ phức tạp của hàm thời gian thực hiện chương trình là $O(nm)$.

Bài tập 11

TÍNH

$$p[i][j] = \begin{cases} 1 & i = 0; j > 0 \\ 0 & i > 0; j = 0 \\ \frac{1}{2} \cdot (p[i-1][j] + p[i][j-1]) & i > 0; j > 0 \end{cases}$$

a. Phát biểu bài toán

INPUT

Hai số nguyên không âm m và n.

OUTPUT

Giá trị của $p(m, n)$ được tính theo công thức ở trên.

b. Ý tưởng giải

Ta có công thức quy hoạch động của $p(m, n)$ là:

$$p(m,n) = \begin{cases} 1 & m = 0; n > 0 \\ 0 & m > 0; n = 0 \\ 0.5 * [p(m-1,n) + p(m,n-1)] & m > 0; n > 0 \end{cases}$$

Kết quả của bài toán gốc nằm ở $p(m, n)$.

c. Thuật giải

// Input: Hai số nguyên không âm m và n.

// Output: Giá trị của $p(m, n)$ được tính theo công thức quy hoạch động ở trên.

Tinh(m, n)

```
1.  for i = 0 to m do
2.      for j = 0 to n do
3.          if i = 0
4.              p[i, j] = 1;
5.          else
6.              if j = 0
7.                  p[i, j] = 0;
8.              else
9.                  p[i, j] = 0.5 * (p[i − 1, j] + p[i, j − 1]);
10. return p[m, n];
```

d. VD minh họa

INPUT	OUTPUT
0 8	1
10 0	0
2 2	0.5
10 16	0.885239
25 24	0.442717

e. Độ phức tạp của thuật toán

Quá trình tính toán giá trị của mỗi $p[i, j]$ có thời gian thực hiện là O(1).

Quá trình trên thực hiện $(m + 1) \cdot (n + 1)$ lần nên độ phức tạp của hàm thời gian thực hiện chương trình là O(mn).

Bài tập 12

SỐ CATALAN

Tính số catalan thứ n theo công thức: $\sum_{i=1}^{n-1} T(i)T(n-i-1)$; $n \in \mathbb{N}^*$.

a. Phát biểu bài toán

Trong toán tổ hợp, số Catalan là dãy các số tự nhiên xuất hiện nhiều trong các bài toán đếm, thường bao gồm những đối tượng đệ quy. Được đặt tên theo nhà toán học Đức Eugène Charles Catalan (1814 - 1894).

Những số catalan đầu dãy $n = 0, 1, 2, 3, ...$ là: $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, ...$

INPUT

Số nguyên dương n.

OUTPUT

Giá trị của số catalan thứ n.

b. Ý tưởng giải

Gọi $F(i)$ là giá trị của số catalan thứ i. Ta có công thức quy hoạch động của $F(i)$ là:

$$F(i) = \begin{cases} 1 & i \leq 1 \\ \sum_{j=0}^{i-1} F(j) * F(i-j-1) & i > 1, 0 < j < i \end{cases}$$

Kết quả của bài toán gốc nằm ở $F(n)$.

c. Thuật giải

// Input: Số nguyên dương n.

// Output: Giá trị của số catalan thứ n.

TinhCatalan($a[1..n]$)

1. $F[0] = F[1] = 1$;
2. **for** $i = 2$ **to** n **do**
3. $\quad F[i] = 0$;
4. $\quad$ **for** $j = 0$ **to** $i - 1$ **do**
5. $\quad\quad F[i] = F[i] + (F[j] * F[i - j - 1])$;
6. **return** $F[n]$;

d. VD minh họa

INPUT	OUTPUT
2	2
5	42
10	16796

e. Độ phức tạp của thuật toán

Độ phức tạp của hàm thời gian thực hiện chương trình là O(n^2).

Bài tập 13

DÃY CON GIẢM DẦN DÀI NHẤT (LONGEST DECREASING SUBSEQUENCE)

Cho dãy số nguyên dương $a = (a_1, a_2, ..., a_n)$. Tìm trong a một dãy con giảm dần dài nhất.

a. Phát biểu bài toán

INPUT

Dãy số nguyên dương $a = (a_1, a_2, ..., a_n)$.

OUTPUT

Độ dài của một dãy con giảm dần dài nhất có trong a.

b. Ý tưởng giải

Gọi $F(i)$ là độ dài của dãy con giảm dần dài nhất của $a(i) = (a_1, a_2, ..., a_i)$. Để thiết lập công thức quy hoạch động cho $F(i)$, ta xét:

$F(j)$ là độ dài của dãy con giảm dần dài nhất của $a(j) = (a_1, a_2, ..., a_j)$ trong $a(i) = (a_1, a_2, ..., a_j, ..., a_i)$.

Nếu $a_i < a_j$ thì dãy con giảm dần của a_i có thể bao gồm dãy con giảm dần dài nhất của $a(j)$ và phần tử a_i.

Để dãy con đó giảm dần dài nhất thì ta sẽ tìm giá trị lớn nhất của độ dài dãy con đó ứng với mỗi giá trị j từ 1 đến $i - 1$.

Do đó, $F(i) = \max_{0<j<i}\{F(j)\} + 1$ nếu $a_i < a_j$.

Ngược lại thì $F(i) = 1$.

Kết quả của bài toán gốc nằm ở $\max_{0<i\leq n} F(i)$.

c. Thuật giải

// Input: Dãy số nguyên dương $a = (a_1, a_2, ..., a_n)$.

// Output: Độ dài của một dãy con giảm dần dài nhất có trong a.

LDSLength($a[1..n]$)

```
1.  max = 0;
2.  for i = 1 to n do
3.      F[i] = 1;
4.  for i = 1 to n do
5.      for j = 1 to i − 1 do
6.          if a_i < a_j and F[i] < F[j] + 1
7.              F[i] = F[j] + 1;
```

8. **for** $i = 1$ **to** n **do**
9. **if** $max < F[i]$
10. $max = F[i]$;
11. **return** max;

d. VD minh họa

INPUT	OUTPUT
2 10 16	1
6 98 67 54 33 23 1	6
7 5 0 2 4 3 8 1	4
9 15 26 13 36 62 55 45 64 80	3

e. Độ phức tạp của thuật toán

Quá trình tính toán giá trị của mỗi $F[i]$ có thời gian thực hiện là $O(n)$.

Quá trình trên thực hiện n lần, cộng thêm quá trình duyệt mảng F để tìm giá trị lớn nhất max thì độ phức tạp của hàm thời gian thực hiện chương trình là $O(n^2)$.

Cách khác: Có thể sử dụng lại ý tưởng của bài **CHUỖI CON CHUNG DÀI NHẤT (LONGEST COMMON SUBSEQUENCE)**, trong đó chuỗi đầu tiên là mảng ban đầu của bài toán, chuỗi thứ hai là mảng ban đầu sau khi sắp xếp các giá trị trong mảng theo thứ tự giảm dần.

Bài tập 14

DÃY CON KHÔNG GIẢM DÀI NHẤT (LONGEST NON-DECREASING SUBSEQUENCE)

Cho dãy số nguyên $a = (a_1, a_2, ..., a_n)$. Hãy xóa 1 số lượng ít nhất các số trong dãy sao cho dãy con còn lại (giữ nguyên thứ tự) là dãy không giảm.

a. Phát biểu bài toán

INPUT

Dãy số nguyên $a = (a_1, a_2, ..., a_n)$.

OUTPUT

Số lượng ít nhất các số cần xóa trong dãy sao cho dãy con còn lại là dãy không giảm.

b. Ý tưởng giải

Để tìm số lượng ít nhất các số cần xóa trong dãy sao cho dãy con còn lại là dãy không giảm, ta đi tìm độ dài dãy con không giảm dài nhất của dãy ban đầu.

Sau đó lấy kích thước của dãy ban đầu trừ đi độ dài của dãy con không giảm dài nhất đó.

Gọi $F(i)$ là độ dài của dãy con không giảm dài nhất của $a(i) = (a_1, a_2, ..., a_i)$. Để thiết lập công thức quy hoạch động cho $F(i)$, ta xét:

$F(j)$ là độ dài của dãy con không giảm dài nhất của $a(j) = (a_1, a_2, ..., a_j)$ trong $a(i) = (a_1, a_2, ..., a_j, ..., a_i)$.

Nếu $a_i \geq a_j$ thì dãy con không giảm của a_i có thể bao gồm dãy con không giảm dài nhất của $a(j)$ và phần tử a_i.

Để dãy con đó không giảm dài nhất thì ta sẽ tìm giá trị lớn nhất của độ dài dãy con đó ứng với mỗi giá trị j từ 1 đến $i - 1$.

Do đó, $F(i) = \max\limits_{0<j<i}\{F(j)\} + 1$ nếu $a_i \geq a_j$.

Ngược lại thì $F(i) = 1$.

Kết quả của bài toán gốc được tính toán thông qua $n - \max\limits_{0<i\leq n} F(i)$.

c. Thuật giải

// Input: Dãy số nguyên dương $a = (a_1, a_2, ..., a_n)$.

// Output: Số lượng ít nhất các số cần xóa trong dãy sao cho dãy con còn lại là dãy không giảm.

LNDSLength$(a[1..n])$

1. $max = 0$;

```
2.  for i = 1 to n do
3.      F[i] = 1;
4.  for i = 1 to n do
5.      for j = 1 to i do
6.          if a_i ≥ a_j and F[i] < F[j] + 1
7.              F[i] = F[j] + 1;
8.  for i = 1 to n do
9.      if max < F[i]
10.         max = F[i];
11. return n − max;
```

d. VD minh họa

INPUT	OUTPUT
2 10 16	0
4 2 5 5 9	0
5 5 6 1 7 4	2
9 30 40 2 5 1 7 45 50 8	4

e. Độ phức tạp của thuật toán

Quá trình tính toán giá trị của mỗi $F[i]$ có thời gian thực hiện là $O(n)$.

Quá trình trên thực hiện n lần, cộng thêm quá trình duyệt mảng F để tìm giá trị lớn nhất max thì độ phức tạp của hàm thời gian thực hiện chương trình là $O(n^2)$.

Cách khác: Có thể sử dụng lại ý tưởng của bài **CHUỖI CON CHUNG DÀI NHẤT (LONGEST COMMON SUBSEQUENCE)**, trong đó chuỗi đầu tiên là mảng ban đầu của bài toán, chuỗi thứ hai là mảng ban đầu sau khi sắp xếp các giá trị trong mảng theo thứ tự không giảm.

Bài tập 15

XẾP HÀNG MUA VÉ

Có n người xếp hàng mua vé. Thời gian bán vé cho người thứ i là t_i. Mỗi người mua tối thiểu 1 vé và được tối đa 2 vé, có thể mua vé cho người đứng sau mình. Người thứ i mua vé hộ cho người thứ $i+1$ thì thời gian mua vé cho 2 người là p_i. Xác định phương án sao cho n người đều có vé với thời gian ít nhất.

a. Phát biểu bài toán

INPUT

Một số nguyên dương n thể hiện số lượng người xếp hàng mua vé.

Thời gian bán vé riêng cho n người.

Thời gian bán vé cho người ở trước để mua vé hộ cho người tiếp theo.

OUTPUT

Tổng thời gian là ít nhất để n người đều có vé.

b. Ý tưởng giải

Gọi $F(i)$ là tổng thời gian ít nhất để hết i người đều có vé. Để thiết lập công thức quy hoạch động cho $F(i)$, ta xét:

+ Trường hợp 1: Giả sử người thứ $i-1$ mua vé hộ cho người thứ i thì thời gian mua vé cho người thứ i = thời gian bán vé cho người thứ $i-1$ để mua vé hộ cho người thứ i là $p(i-1)$ + thời gian mua vé của những người trước người thứ $i-1$ là $F(i-2)$.

+ Trường hợp 2: Giả sử người thứ $i-1$ không mua vé hộ cho người thứ i, khi đó người thứ i tự mua vé nên thời gian mua vé cho người thứ i = thời gian bán vé cho người thứ i là $t(i)$ + thời gian mua vé của những người trước người thứ i là $F(i-1)$.

Ta lấy kết quả nhỏ nhất trong 2 trường hợp trên để được tổng thời gian ít nhất $F(i)$. Do đó, $F(i) = \min\{t(i) + F(i-1), p(i-1) + F(i-2)\}$.

Hiển nhiên, ta cũng có: $F(1) = t_1$.

Kết quả của bài toán gốc nằm ở $F(n)$.

c. Thuật giải

// Input:

Một số nguyên dương n thể hiện số lượng người xếp hàng mua vé.

Một mảng t lưu giá trị của thời gian bán vé riêng cho người thứ i ($1 \le i \le n$).

Một mảng p lưu giá trị của thời gian bán vé cho người j để mua vé hộ cho người tiếp theo $j+1$ ($1 \le j \le n-1$).

// Output: Tổng thời gian là ít nhất để n người đều có vé.

MuaVe$(t[1..n], p[1..n-1])$

1. $F[1] = t[1]$;
2. $F[2] = \min(t[1] + t[2], p[1])$;
3. **for** $i = 3$ **to** n **do**
4. $\quad F[i] = \min(t[i] + F[i-1], p[i-1] + F[i-2])$;
5. **return** $F[n]$;

d. VD minh họa

INPUT	OUTPUT
4 3 2 4 3 4 3 7	9
5 2 5 7 8 4 4 9 10 10	18
4 5 7 8 4 25 26 27	24

e. Độ phức tạp của thuật toán

Quá trình tính toán điền giá trị của mỗi $F[i]$ có thời gian thực hiện là O(1).

Quá trình trên thực hiện n lần nên độ phức tạp của hàm thời gian thực hiện chương trình là O(n).