BỘ GIÁO DỤC VÀ ĐÀO TẠO

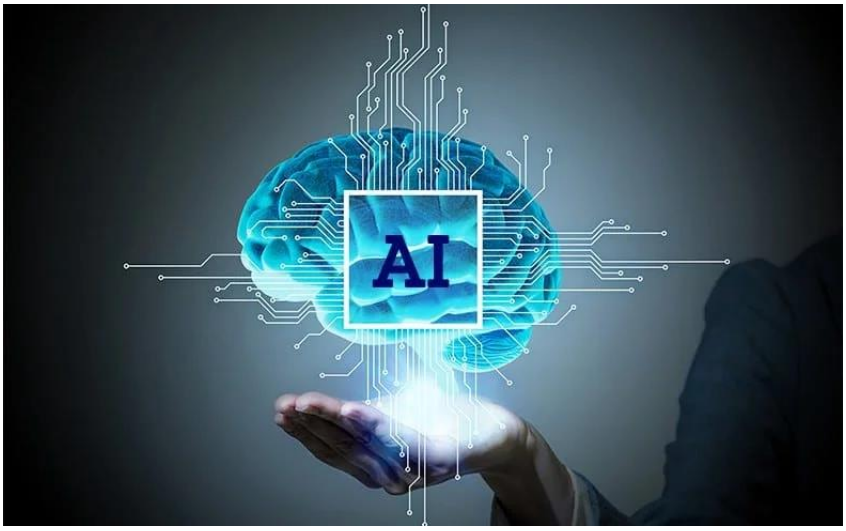
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

KHOA TOÁN - TIN HỌC

-----🕮-----

BÀI BÁO CÁO THỰC HÀNH TUẦN 6

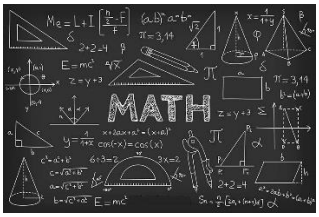


MÔN HỌC: **Phân Tích Thuật Toán**

Sinh Viên: **Trần Công Hiếu - 21110294**

Lớp: **21TTH**

TP.HCM, ngày 19 tháng 05 năm 2024



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA TOÁN – TIN HỌC

BÀI BÁO CÁO THỰC HÀNH TUẦN 5

HK2 - NĂM HỌC: 2024-2025

MÔN: PHÂN TÍCH THUẬT TOÁN

SINH VIÊN: TRẦN CÔNG HIẾU

MSSV: 21110294

LỚP: 21TTH

NHẬN XÉT

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp.HCM, Ngày……Tháng……Năm 2021

Giảng viên Bộ môn

Mục Lục

Bài 1. Quy hoạch động.

1. Ý tưởng bài toán.

Dựa trên ý tưởng được gợi ý trong bài. Thay vì tạo a_i có 2 giá trị w và v, thì ta tạo 2 mảng để chứa lần lượt các trọng số w_i và giá trị v_i tương ứng.

Giá trị $V[i,j]$ được tính dựa vào đoạn code sau:

$$if(j - w_i) < 0)$$

$$V[i,j] = V[i-1,j]$$

$$else\{$$

$$V[i,j] = max(V[i-1,j], v_i + V[i-1, j-w_i])$$

$$\}$$

Ta sẽ xét bảng với sức chứa chạy từ 0 cho đến W (hoặc S như trong bài toán). Sau đó, đoạn mã giả trên được hiểu là ta sẽ xét các item được bỏ vào túi có sức chứa j $\in [0, W]$, nếu không còn sức chứa (hay j $-$ w_i < 0) thì ta cập nhật giá trị $V[i,j] = V[i-1,j]$. Ngược lại, túi có sức chứa j vẫn có thể chứa tiếp được, ta sẽ lấy giá trị lớn nhất giữa giá trị vừa rồi của túi j là $V[i-1,j]$ so với giá trị khi j lấy item đó cộng với phần dư có thể lấy thêm item khác ($V[i-1, j-w_1) + v_i$).

Tìm lời giải của bài toán:

$i = n, \quad j = W$

while $\quad i, k > 0$

if $\quad V[i, j] \neq V[i-1, j] \quad$ *then*

Đánh dấu item thứ i như trong knapsack

$i = i - 1, \quad j = j - w_i$

else

$i = i - 1$

}

2. Trình bày đoạn mã.

(File: bai1.py)

Trong đoạn mã này, ta sẽ xét 1 trường hợp trong bài toán yêu cầu. Đó là có n = 50 item và S = 200 với trọng số w_i chạy trong đoạn [0, 500].

```
1   import numpy as np
2   n = 50
3   S = 200
4   V = np.zeros((n+1, S+1))
5
6
7   def print_matrix(matrix):
8       n = matrix.shape[0]
9       m = matrix.shape[1]
10      for i in range(0, n):
11          for j in range(0, m):
12              print(int(matrix[i][j]),end=" ")
13          print()
14
```

"import numpy as np": Gọi thư viện numpy để thao tác với mảng, tạo số ngẫu nhiên trong đoạn mã của chương trình và gọi

tắt là “np”, điều này cho phép gọi các hàm và đối tượng từ thư viện numpy bằng cách sử dụng tiền tố “np”.

“n = 50”: Khởi gián cho n bằng 50, điều này cho biết bài toán đang xét có 50 cặp trọng số w_i và giá trị v_i.

“S = 200”: Khởi gán cho S giá trị 200, đây là biến đại diện cho W như phát biểu trong ý tưởng bài toán. Nó cho biết tổng các trọng số w_i không vượt quá S.

“V = np.zeros((n+1, S+1))”: Thay vì như trong hướng dẫn, là duyệt dòng và cột đầu tiên (index = 0) và cho chúng là 0 như trong bảng Kcapsack. Thì ta có thể tạo ma trận 0 từ đầu với n+1 dòng (từ 0 đến n) và S+1 cột (từ 0 đến S).

“def print_matrix(matrix):”: Định nghĩa hàm print_matrix() với đối số truyền vào là matrix (trong bài chính là để truyền V vào) nhằm in ra bảng Kcapsack như trong bài. Trong hàm, ta sẽ lấy số dòng và cột gán lần lượt cho n, m thông qua phương thức shape[] (index 0 là lấy số dòng, index 1 là lấy số cột tương ứng với matrix). Khi đó, ta duyệt mọi phần tử trong ma trận để in ra được bảng Kcapsack. Các phần tử cách nhau bằng khoảng trắng (end = “ ”).

```
15    w = np.random.choice(range(501), size=n+1, replace=False)
16    w[0] = 0
17    print("-w:\n",w)
18    v = np.random.randint(0, 501, size=n+1)
19    v[0] = 0
20    print("-v:\n",v)
21
```

“w = np.random.choice()”: Với phương thức này, ta tạo được mảng w có các phần tử đôi một khác nhau có kích thước n+1 và giá trị nằm trong đoạn [0, 500]. Vì ta hiểu rằng các item cũng được xếp tương ứng bảng Kcapsack để xét nên do đó ta cần phải có phần tử đầu tiên của mảng w là 0 (w[0] = 0). Sau cùng có thể in ra để thuận tiện quan sát.

“v = np.random.randint ()”: Với phương thức này, ta tạo được mảng v có các phần tử có thể trùn nhau, bởi ta cần hiểu rằng tổng các giá trị a_{i_j} bằng S, tức các trọng số phải khác nhau theo yêu cầu còn giá trị có thể trùng nhau. Mảng có kích thước n+1 và giá trị nằm trong đoạn [0, 500]. Vì ta hiểu rằng các item cũng được xếp tương ứng bảng Kcapsack để xét nên do đó ta cần phải có phần tử đầu tiên của mảng w là 0 (w[0] = 0). Sau cùng có thể in ra để thuận tiện quan sát.

```
22    for i in range(1, n+1):
23        for j in range(1, S+1):
24            if(j < w[i]):
25                V[i][j] = V[i-1][j]
26            else:
27                V[i][j] = max(V[i-1][j], V[i-1][j-w[i]] + v[i])
28    #print("- Bang kcapsack:")
29    #print_matrix(V)
30    print("- Max = ",int(V[n][S]))
31
```

Các giá trị $V[i,j]$ được tính dựa vào đoạn mã trên. Như trong bảng có thể thấy, dòng và cột ứng với i = 0, j = 0 đều là 0 nên ta chỉ xét từ index 1 trở đi. Tạo 2 vòng lặp lồng nhau để xét các phần tử.

“if(j<w[i]):”: Điều kiện này cho biết sức chứa của “túi” tại j nhỏ hơn trọng số w, nên không thể “đựng” thêm nữa. Do đó, ta cập nhật lại $V[i,j]$ bằng $V[i-1,j]$.

Ngược lại, nếu “túi” tại j vẫn còn “sức chứa”, ta xét giá trị giữa 2 giá trị đó.

Có thể in bảng Kcapsack với 1 dòng thông báo in bảng và dùng hàm đã định nghĩa trên để in ma trận $V[i,j]$. Tuy nhiên, do bài toán có kích thước khá lớn nên việc quan sát bảng không “ổn” nên ta sẽ comment lại đoạn mã đó, khi với trường hợp kích thước bé hơn thì ta có thể in ra quan sát.

```
32    i = n
33    j = S
34    solution_items = []
35
36    print("- Loi giai toi uu cua bai toan Knapsack chua {",end="")
37    while ((i>0) and (j>0)):
38        if(V[i][j]!=V[i-1][j]):
39            solution_items.append(i)
40            i = i-1
41            j=j-w[i]
42        else:
43            i=i-1
44    solution_items.reverse()
45    print(", ".join(map(str, solution_items)), end="")
46    print("}")
```

Khởi tạo i = n, j = S như đoạn mã tìm lời giải của bài toán trong hướng dẫn. Tạo list solution_items để lưu lại item thứ i trong Knapsack.

Vòng lặp while xét điều kiện với cả i và j đồng thời lớn hơn 0. Khi đó, nếu giá trị tại vị trí đang xét khác với giá trị tại vị trí cùng cột nhưng trên 1 dòng thì khi đó ta lưu lại item thứ i đó bằng phương thức append() và cập nhật lại i, j. Ngược lại, nếu 2 giá trị đó bằng nhau, ta cập nhật lại i = i-1.

Cuối cùng ta dùng phương thức reverse() để đảo ngược list lại, vì list lưu item theo thứ tự giảm dần. Và xóa dấu phẩy đằng sau phần tử cuối cùng tìm được để in thông báo ra cho dễ quan sát.

*** Kết quả:**

```
C:\Users\PC-LENOVO\Downloads\Pycharm_code\venv\Scripts\python.exe C:\Users\PC-LENOVO\Downloads\Pycharm_code\test.py
-w:
 [  0 297 387  23 205 246  67 316 477 413 262  33  57   1 273 362 226 295
 383 355 122 352 389 331 336 269 348 253 381 499 120  83 457 450 136 259
 153 367  81 176 117 337 146 264 490 152 310 193 294 128 350]
-v:
 [  0   8 383 429  14 154 350 234 155 335  66  73 210 294 368 327 179 246
  58 374 255 200 445 225  45  59 261 188 497 412  78 336 278 258 492 241
 195 286 105  78 250 342  71 101  10 371 112 498  22 297 483]
- Max =  1409
- Loi giai toi uu cua bai toan Knapsack chua {3, 13, 31}

Process finished with exit code 0
```

2. Đồ thị thời gian trung bình t.

(File: dothi.py)

```
import numpy as np
import time
import matplotlib.pyplot as plt

```

Với đoạn mã trên, ta import thêm matplotlib.pyplot as plt để có công cụ vẽ đồ thị. Và cả time để tính thời gian chạy của đoạn mã ứng với từng n.

```
S = 200

def Kcapsack(n):
    start = time.time()
    V = np.zeros((n+1, S+1))
    w = np.random.choice(range(1001), size=n+1, replace=False)
    w[0] = 0
    v = np.random.randint(0, 501, size=n+1)
    v[0] = 0

    for i in range(1, n+1):
        for j in range(1, S+1):
            if(j < w[i]):
                V[i][j] = V[i-1][j]
            else:
                V[i][j] = max(V[i-1][j], V[i-1][j-w[i]] + v[i])

    i = n - 1
    j = S - 1
    solution_items = []

    #print("- Loi giai toi uu cua bai toan Knapsack chua {",end="")
    while ((i>0) and (j>0)):
        if(V[i][j]!=V[i-1][j]):
            solution_items.append(i)
            i = i-1
            j=j-w[i]
        else:
            i=i-1
    solution_items.reverse()
    #print(", ".join(map(str, solution_items)), end="")
    #print("}")
    return time.time() - start

```

Lúc này, gán S = 200 như yêu cầu test chương trình của bài toán. Tuy nhiên, ta cần hiểu rằng việc tạo ra các item a_i đôi một khác nhau tức là w_i và v_i thì có thể trùng nhau.

Đưa các phần chính của chương trình vào hàm Kcapsack(n). Và bỏ qua các dòng lệnh thông báo. Khởi gán ở đầu hàm start

bằng lệnh time.time() để lấy thời gian hiện tại và cuối cùng trả về time.time() - start để đưa về khoảng thời gian chạy đoạn mã.

Lưu ý, vì việc đôi một khác nhau nên nếu xét n từ 550 đến 1000 thì sẽ gây lỗi, bởi giá trị ngẫu nhiên nằm trong đoạn 0 đến 500 không thể tạo ra đôi một khác nhau cho n phần tử đó được. Vì thế ta xét đoạn [0, 1000] còn v thì vẫn là [0, 500].

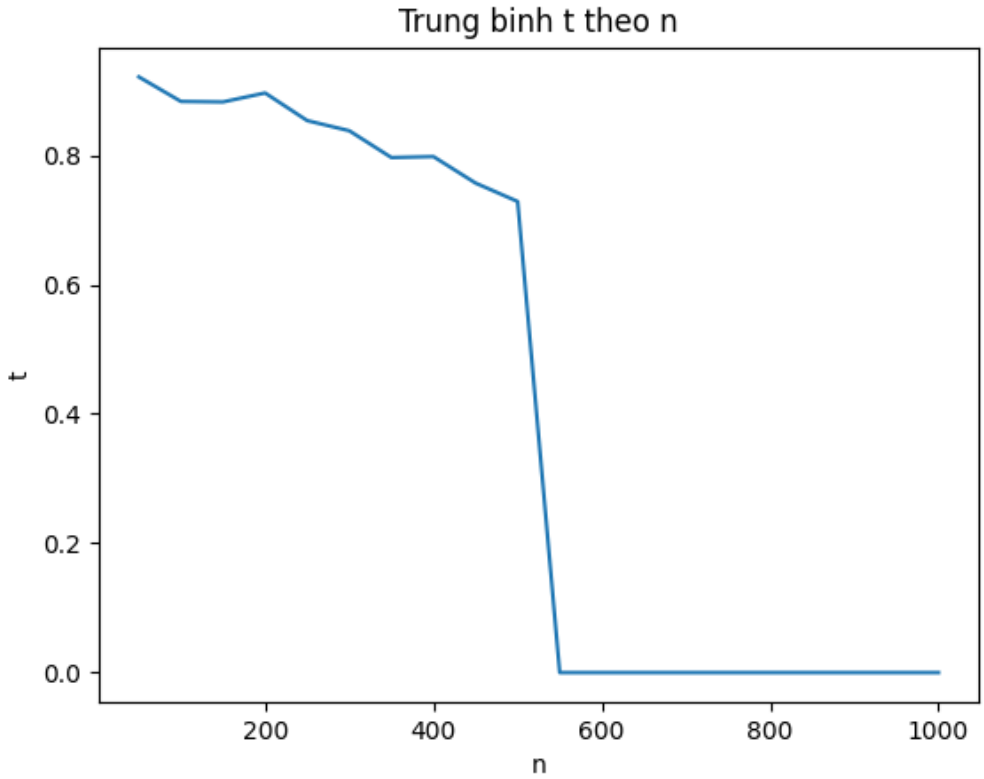
```
40    time_list = np.zeros(20)
41
42    list_n = []
43    for n in range(50, 1050, 50):
44        list_n.append(n)
45        for i in range(0, 10):
46            time_list[i] += Kcapsack(n)
47        k = int((n-50)/50)
48        time_list[k] /= 10
49        print("- Voi n =",n,"thi t = ",time_list[k])
50
51    # Vẽ đồ thị
52    plt.plot(list_n,time_list)
53
54    # Thêm tiêu đề và nhãn
55    plt.title('Trung binh t theo n')
56    plt.xlabel('n')
57    plt.ylabel('t')
58
59    # Hiển thị đồ thị
60    plt.show()
```

Tạo list time_list để lưu lại các thời gian chạy đoạn mã ứng với n chạy từ 50 đến 1000, bước nhảy 50. Với mỗi n ta sẽ chạy 10 lần để lấy trung bình 10 lần chạy với n cố định đó. Và cuối cùng vẽ đồ thị.

*** Kết quả:**

```
C:\Users\PC-LENOVO\Downloads\Pycharm_code\venv\Scripts\python.exe C:\Users\PC-LENOVO\Downloads\Pycharm_code\test.py
- Voi n = 50 thi t =  0.0004278421401977539
- Voi n = 100 thi t =  0.001316976547241211
- Voi n = 150 thi t =  0.0025092601776123048
- Voi n = 200 thi t =  0.004419851303100586
- Voi n = 250 thi t =  0.006679272651672364
- Voi n = 300 thi t =  0.009249091148376465
- Voi n = 350 thi t =  0.011984562873840332
- Voi n = 400 thi t =  0.0158023765563965
- Voi n = 450 thi t =  0.019901108741760255
- Voi n = 500 thi t =  0.02342538833618164
- Voi n = 550 thi t =  0.0
- Voi n = 600 thi t =  0.0
- Voi n = 650 thi t =  0.0
- Voi n = 700 thi t =  0.0
- Voi n = 750 thi t =  0.0
- Voi n = 800 thi t =  0.0
- Voi n = 850 thi t =  0.0
- Voi n = 900 thi t =  0.0
- Voi n = 950 thi t =  0.0
- Voi n = 1000 thi t =  0.0

Process finished with exit code 0
```



*** Nhận xét:**

Khi n tăng từ 50 đến 500 thì thời gian đoạn mã tìm được lời giải bài toán tăng dần. Tuy nhiên, n tăng tiếp lên 550 đến 1000 thì

thời gian chạy chỉ là xấp xỉ 0.0. Điều này chứng tỏ thuật toán phù hợp với việc giải quyết tốt các bài toán có kích thước đầu vào lớn.