

2023학년도 1학기
Computer Algorithms



과 목 명	ComputerAlgorithms(01)
담 당 교 수	Se Eun Oh
학 과	컴퓨터공학과
학 번	2071035
이 름	이소민 (LeeSomin)
제 출 일	2023.05.03

ha2_1.py

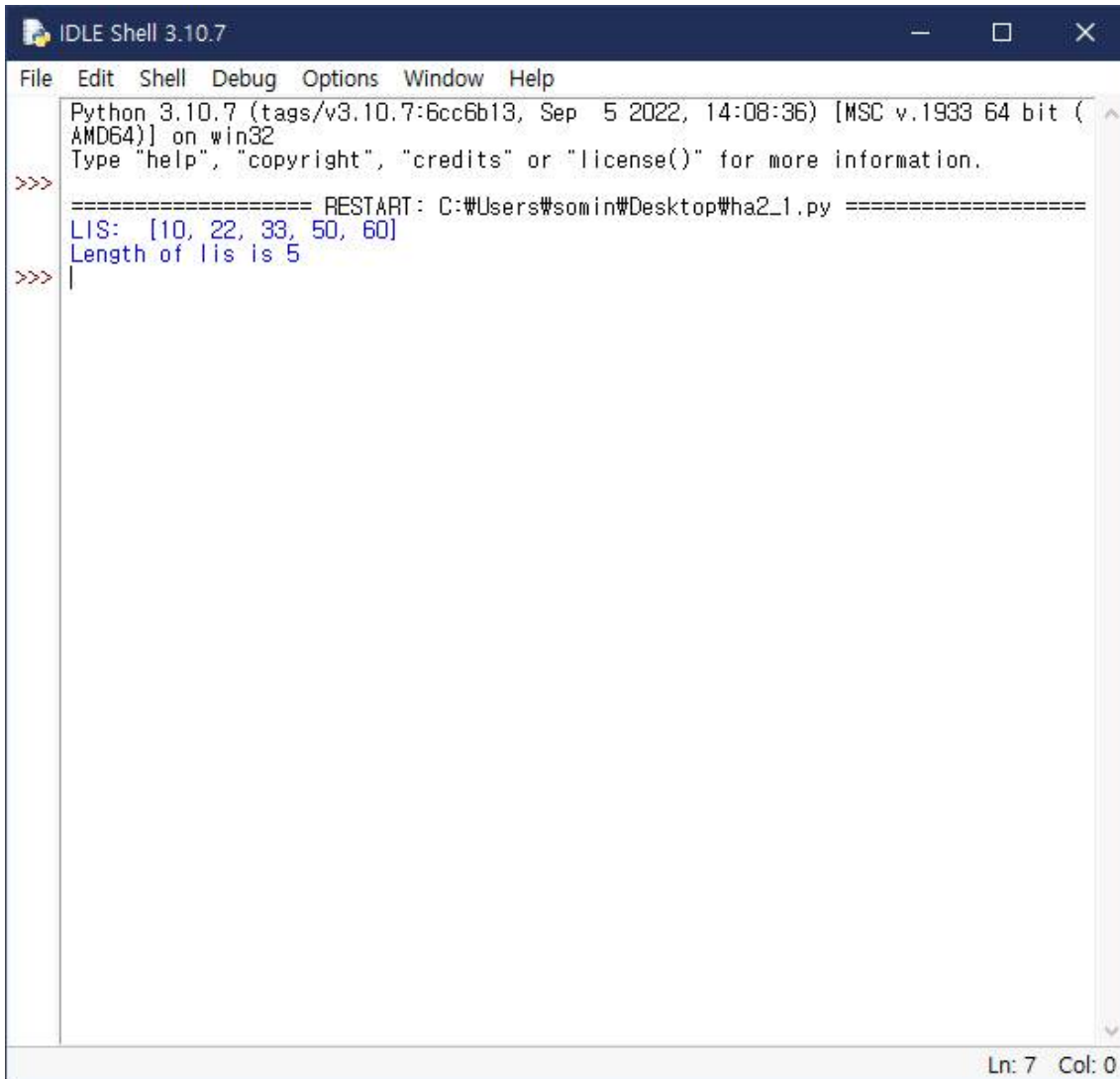
1. Code:

```
ha2_1.py - C:\Users\somin\Desktop\ha2_1.py (3.10.7)
File Edit Format Run Options Window Help
def lis(arr):
    n=len(arr)
    # h_i values for all indexes
    h=[1]*n
    # preceeding element for lis
    pre=[-1]*n
    # update preceeding element and lis
    # when meeting element smaller than present element
    # and its lis length+1 is bigger than present lis length
    for i in range(1,n):
        for j in range(0,i):
            if arr[i]>arr[j] and h[i] < h[j] + 1:
                h[i] = h[j]+1
                pre[i]=j
    # get the maximum length lis and the biggest element for lis
    max_len = 0
    for i in range (n):
        if (max_len<h[i]):
            max_len=h[i]
            max_val=i
    # get LIS array by visiting preceeding elements in lis
    lis=[0]*max_len
    lis[max_len-1]=arr[max_val]
    k = max_val
    pos = max_len-2
    while (k>0):
        lis[pos]=arr[pre[k]]
        k=pre[k]
        pos-=1
    # print lis and the length of the lis
    print("LIS: ",lis)
    return max_len

if __name__ == '__main__':
    arr = [10, 22, 9, 33, 21, 50, 41, 60]
    print("Length of lis is",lis(arr))

Ln: 1 Col: 0
```

2. Result:



```
IDLE Shell 3.10.7
File Edit Shell Debug Options Window Help
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\somin\Desktop\ha2_1.py =====
lis: [10, 22, 33, 50, 60]
Length of lis is 5
>>> |
```

3. Space Complexity

LIS를 출력하는 프로그램을 구현하기 위해 추가로 사용한 변수는 int형 원소 n 개를 갖는 배열 `pre[]`와 LIS의 길이만큼의 int형 원소를 갖는 배열 `lis[]`이다. 각각의 Space Complexity와 강의자료에 의한 기존 코드의 Space Complexity인 $O(n)$ 중 더 큰 값이 프로그램의 Space Complexity가 된다.

- `pre[]`: $O(n)$
 - `lis[]`: $O(n)$
- (배열이 이미 오름차순으로 정렬되어있을 때 `lis[]`는 최악 경우 n 개의 원소를 갖는다)
따라서 LIS를 출력하는 프로그램의 Space Complexity는 $O(n)$ 이다.

ha2_2.py

1. Code:

```
ha2_2.py - C:\Users\wsomin\Desktop\ha2_2.py (3.10.7)
File Edit Format Run Options Window Help
1 # Find minimum number of US postage stamps_Dynamic Programing
2
3 import sys
4
5 def minStamp(p_stamp, q_stamp, cost):
6     # store the minimum number of postage for i value.
7     dp = [0]*(cost+1)
8     # Base case (If given cost is 0)
9     dp[0] = 0
10    # initiate the number of total stamp for each cost as max
11    for i in range(1, cost+1):
12        dp[i] = sys.maxsize
13    # find minimum number of stamp for each cost
14    for i in range(1, cost+1): # all values up to cost
15        for j in range(q_stamp): # all stamps in array
16            if(p_stamp[j] <= i):
17                sub_res = dp[i-p_stamp[j]] # subproblem result
18                # if the result of subproblem+1 is smaller than
19                # current number of stamp, update the number
20                if (sub_res != sys.maxsize and sub_res + 1 < dp[i]):
21                    dp[i] = sub_res + 1
22    return dp[cost]
23
24 if __name__ == '__main__':
25     p_stamp = [1, 10, 21, 34, 70, 100, 350, 1225, 1500]
26     q_stamp = len(p_stamp)
27     cost = 140
28     print("Minimum Stamp required is ", minStamp(p_stamp, q_stamp, cost))
29
Ln: 16 Col: 23
```

2. Result:

```
IDLE Shell 3.10.7
File Edit Shell Debug Options Window Help
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\wsomin\Desktop\ha2_2.py =====
>>> Minimum Stamp required is 2
>>> |
```

3. Time Complexity: $O(\text{cost} \cdot \text{q_stamp})$

- cost는 우편의 가격, q_stamp는 우표 종류 수
 - line 11-12 (for문): $O(\text{cost})$
cost만큼 $O(1)$ 명령 반복
 - line 14-21 (2중첩 for문): $O(\text{cost} \cdot \text{q_stamp})$
최악 경우 제일 비싼 우표의 가격이 우편의 가격보다 작을 때,
안쪽 for문에서 $O(1)$ 인 명령이 q_stamp번 반복되며 안쪽 for문은 cost번 반복
따라서 $O(1)$ 의 명령이 총 $\text{cost} \cdot \text{q_stamp}$ 만큼 반복된다.
 - 나머지 line들은 각 $O(1)$ 의 time complexity를 가지므로 생략한다.
- q_stamp>0이므로 이 코드는 최악 경우 $O(\text{cost} \cdot \text{q_stamp})$ 의 time complexity를 가진다.

4. Space Complexity: $O(\max(\text{cost}, \text{q_stamp}))$

- cost는 우편의 가격, q_stamp는 우표 종류 수
 - dp[]: $O(\text{cost})$
이 코드는 모든 cost값에 대한 최소 우표 수를 저장하므로
dp는 cost+1의 크기 배열이다.
 - p_stamp[]: $O(\text{q_stamp})$
p_stamp는 우표의 종류에 따른 가격들을 저장한 배열이므로
우표의 종류 수인 q_stamp크기의 배열이다.
 - 나머지 변수는 각 $O(1)$ 의 space complexity를 가지므로 생략한다.
- 따라서 이 코드는 최악 경우 $O(\max(\text{cost}, \text{q_stamp}))$ 의 space complexity를 가진다.

5. Optimal Solution

우편의 가격이 0원일 때 최소우표 갯수의 최적해는 0개이다. 이후의 우편 가격에 대해 dynamic programming은 더해질 우표의 가격을 뺀 우편 가격의 최적해에 +1을 한 값과 기존 값을 비교하여 더 작은 값을 dp[]에 저장함으로 각 가격에 대한 최적해를 점화적으로 구한다. 따라서 0cent 우편의 dp[]값이 최적해일 때, 모든 우편 가격에 대해 해당 코드는 최적해를 구한다. 따라서 dp[0]의 값이 정해진 우표 문제에 대해 dynamic programming은 모든 cost에 대한 최적해를 구한다. 반면, greedy알고리즘은 다른 우편 가격에 대한 고려를 하지 않고 사용할 수 있는 가장 비싼 우표를 최대 개수로 사용하며 가격을 줄여 간다. 하지만 이는 해당 가격의 우표 m개를 사용할 수 있을 때 m개의 우표보다 적은 개수의 우표 n 개를 사용하고 남은 가격이 다른 우표의 가격으로 나누어 떨어지는 경우 최소 우표 개수에 대해 최적해를 반환하지 못한다.

코드의 실행 결과를 확인하면 수기로 구한 최적해 70cent짜리 우표 두 개와 같은 값인 2를 반환하는 것을 확인할 수 있다.

ha2_3.py

1. Code:

```
*ha2_3.py - C:\Users\somin\Desktop\ha2_3.py (3.10.7)*
File Edit Format Run Options Window Help
1 INF = int(1e9+7)
2
3 def floyd_warshall(dist, length):
4     # path array stores the intermediate node in between source and destination
5     path = [[INF for i in range(length)] for j in range(length)]
6     # for evry nodes, check if the path having the node as
7     # intermediate node is shorter than current distance
8     # if it is, update distance array and add intermediate node to path array
9     for r in range(length):
10         for p in range(length):
11             for q in range(length):
12                 if(dist[p][q]>dist[p][r]+dist[r][q]):
13                     dist[p][q] = dist[p][r]+dist[r][q]
14                     path[p][q] = r
15     print_dist(dist, length)
16     return path
17
18 def print_dist(dist, length):
19     # print the shortest distance between source and destination
20     for p in range(length):
21         temp=""
22         for q in range(length):
23             if(dist[p][q]==INF):
24                 temp+="INF "
25             else:
26                 temp+=str(dist[p][q])+ " "
27         print(temp+" ")
28
29 def print_path(printPath, path, src, dst):
30     # print the shortest path by calling print_path() recursively
31     if(path[src][dst]==INF):
32         printPath.append(src)
33         return
34     # print path from source to intermediate node
35     print_path(printPath, path, src, path[src][dst])
36     # print path from intermediate node to destination
37     print_path(printPath, path, path[src][dst], dst)
38     printPath.append(dst)
39     return
40
41 if __name__ == '__main__':
42     W = [[0, 3, INF, 7],
43          [8, 0, 2, INF],
44          [5, INF, 0, 1],
45          [2, INF, INF, 0]]
46     # get intermediate node for every source and destination
47     path = floyd_warshall(W, len(W[0]))
48     # print the shortest path
49     printPath = []
50     print_path(printPath, path, 1, 3)
51     print("path from 1 to 3:", printPath)
52     printPath.clear()
53     print_path(printPath, path, 0, 2)
54     print("path from 0 to 2:", printPath)
55
```

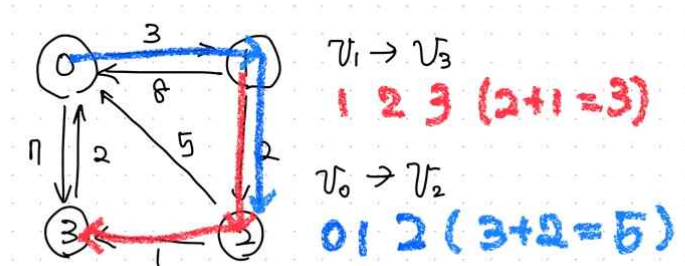
Ln: 8 Col: 6

2. Result:

```

IDLE Shell 3.10.7
File Edit Shell Debug Options Window Help
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\somin\Desktop\ha2_3.py =====
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0
path from 1 to 3: [1, 2, 3]
path from 0 to 2: [0, 1, 2]
>>>

```



3. Code Analysis:

1) floyd_warshall(dist, length)

이 함수는 dynamic programming을 통해 중간노드 r 의 경유할 때의 경로 길이가 기존 경로의 길이보다 작으면 경로 길이를 갱신한다. 경로의 길이가 갱신될 때, `path[][]`에 중간 노드 r 이 저장된다. 모든 노드에 대한 확인이 끝나면 `print_dist()`를 불러 최소경로의 2차원 배열(`dist[][]`)을 출력하고 각 노드의 모든 노드로 갈 때 최소 경로에서의 중간노드를 저장한 배열인 `path[][]`를 반환한다.

2) print_dist()

이 함수는 중첩 for문으로 이차원배열 `dist[][]`를 문자열의 형태로 출력한다. 이때, `dist[p][q]`의 값이 변수 `INF`의 값과 같으면 문자열 "INF"를 출력한다.

3) print_path()

이 함수는 재귀적 호출을 통해 최소 경로에서 들르는 노드들을 출력한다. 출발 노드와 도착 노드가 주어지면, 함수는 `path[][]`를 조회해 출발 노드에서 중간 노드까지의 경로 출력을 위해 재귀호출을 하고, 이가 끝나면 중간 노드에서 도착 노드까지의 경로 출력을 위해 다시 재귀호출을 한다. 노드들이 중간 노드 없이 직접 연결되어있을 경우, `path[][]`의 값이 `INF`이므로 이때 출발 노드를 출력 배열에 추가하고 반환한다. 마지막 자식 재귀호출이 반환되면, 마지막 도착 노드를 출력하고 최종반환한다.

4. Time Complexity: $O(n^3)$

- n 은 int length값과 동일하고, 주어진 그래프의 노드 개수를 의미한다.

- line 9-14 (3중첩 for문): $O(n^3)$

- line 20-26 (중첩 for문): $O(n^2)$

- line 29-39 (재귀함수): $O(n)$

최악 경우 모든 노드를 지나는 경로가 최적경로일 때 `print_path`는 n 번 호출된다.

따라서, 이 코드의 **Time Complexity**는 $O(n^3)$ 으로 경로를 출력하지 않을 때와 동일하다.

5. Space Complexity: $O(n^2)$

추가로 사용한 배열은 dist와 동일한 크기의 path배열과 최적경로 노드를 저장하는 printPath 배열인데, printPath 배열은 최악 경우 그래프의 총 노드의 개수와 같으므로 **Space Complexity** 또한 $O(n^2)$ 로 경로를 출력하지 않을 때와 동일하다.