

**알고리즘 과제 3**

**과목명 알고리즘**

**담당교수 김희철교수님**

**제출일 20211110**

**전공 컴퓨터전자시스템**

**학번 201904458**

**이름 이준용**

* **문제 1-1번**

1-1) n(1 이상 10,000 이하 정수)개 계단을 바닥에서 위로 올라가려고 한다. 계단을 올라갈 때 한 번에 1개, 2개의 계단만 오를 수 있으며, 각 계단은 밟을 때 비용이 있다. 바닥에서 가장 위의 계단으로 올라갈 때 밟는 계단의 비용 합이 최소가 되도록 하면서 올라가고자 한다. 이때의 비용합(최소 비용)을 구하는 프로그램을 작성하시오.

입력:

첫 번째 줄에 양의 정수 n이 주어진다. 다음 줄에 가장 아래 계단부터 위로 차례대로 n개 각 계단을 밟을 때 비용이 양의 정수로 주어진다.

출력:

바닥에서 가장 위의 계단으로 올라갈 때 밟는 계단의 비용 합의 최소값을 출력한다.

입력 예 1 출력 예 1

7

2 6 2 4 11 10 2 17

* **알고리즘 및 자료구조**

1, 2번째 계단까지의 비용 값을 구하는 부분은 직접 구했습니다.

3번째부터 n번째까지의 for문으로 구하고 마지막 계단은 밟아야 되므로 cost[i] 추가했습니다.

비용의 최솟값을 재귀적으로 구했습니다.

* **시간복잡도 분석**

For 구문에서 최소값에 i번째 계단에 오르는 비용을 더하는 부분이므로 1 ~ n까지인 시간복잡도는 O(n)입니다.

* **느낀 점**

n번째 계단 최소비용을 구하는 점화식을 for구문으로 변환하는데 시간이 걸렸습니다.

* **프로그램 코드**
* # 1-1) n(1 이상 10,000 이하 정수)개 계단을 바닥에서 위로 올라가려고 한다.  
  # 계단을 올라갈 때 한 번에 1개, 2개의 계단만 오를 수 있으며, 각 계단은 밟을 때 비용이 있다.  
  # 바닥에서 가장 위의 계단으로 올라갈 때 밟는 계단의 비용 합이 최소가 되도록 하면서 올라가고자 한다.  
  # 이때의 비용합(최소 비용)을 구하는 프로그램을 작성하시오.  
  import sys  
    
  def min\_cost\_step(n, cost):  
   if 1 <= n <= 10000:  
   cost.insert(0, 0) # 0번째 계단 비용  
   arr = [0 for i in range(n+1)] # arr에는 최소값이 들어감  
   arr[0], arr[1], arr[2] = cost[0], cost[1], cost[2]  
   # 4번째까지는 최소값 구함  
   for i in range(3, n+1): # 3번째부터 n까지 for문을 돌림.  
   arr[i] = min(arr[i-1], arr[i-2]) + cost[i]  
   return arr[n] # n번째 계단 비용의 최소값을 리턴  
    
  n = int(sys.stdin.readline())  
  price = list(map(int, sys.stdin.readline().split()))  
  print(min\_cost\_step(n, price))
* **문제 1-2번**

1-2) n(1이상 10,000이하 정수)개 계단을 바닥에서 위로 올라가려고 한다. 계단을 올라갈 때 한 번에 **s개 이하**의 계단만 오를 수 있으며, 각 계단은 밟을 때 비용이 있다. 바닥에서 가장 위의 계단으로 올라갈 때 밟는 계단의 비용 합이 최소가 되도록 하면서 올라가고자 한다. 이때의 비용합(최소 비용)을 구하는 프로그램을 작성하시오.

입력:

첫 번째 줄에 양의 정수 n과 s가 주어진다. 다음 줄에 가장 아래 계단부터 위로 차례대로 n개 각 계단을 밟을 때 비용이 양의 정수로 주어진다.

출력:

바닥에서 가장 위의 계단으로 올라갈 때 밟는 계단의 비용 합의 최소값을 출력한다.

입력 예 1 출력 예 1

7 3

3 7 9 9 2 5 4 13

* **알고리즘 및 자료구조**

입력 받은 n개의 계단이 1부터 10000사이의 값인지 확인합니다.

Arr 0부터 n까지의 배열을 생성

s\_li s개 이하의 계단을 오를 수 있는 값을 저장

arr를 10000수로 채우고, i가 k 보다 크거나 같은 경우

arr[i] = min(arr[i], arr[i-k] + cost[i])

i번째 arr 배열에 최소값 저장

arr[n]값이 초기값인 10000인 경우는 배열의 끝이다. 이러면 -1을 리턴

아닌 경우는 arr[n]리턴합니다.

* **시간복잡도 분석**

두개의 for 구문중에 첫번째 for구문i는 0부터 n까지 계단 검사하는데 s개 이하의 계단을 오른다고 했으므로 두번째 for 구문 x는 0부터 s까지 하나씩 검사합니다. 그러므로 시간복잡도는 O(ns) n개계단\*s개이하 계단오르는 값입니다.

* **느낀 점**

수요일 수업 끝나고 교수님께 질문해서 제 코드의 문제점을 이해할 수 있었습니다. 문제점을 알기 전에는arr배열에 [10001 for i in range(n+1)]로 입력 받은 n 개수만큼 10001로 초기화 했습니다.

하지만 예를 들어 입력 값에

7 1

4 10000 10000 666666666666 6666666666 444444444 44444

위와 같은 경우에는 오를 수 있는 계단수는 1이지만 계단의 비용이 1보다 크므로

Arr 배열을 갱신하지를 못해서 arr배열에는 [10001,10001,…,10001]이 그대로 남게 된다는 문제점을 발견했습니다. 그 때 n의 값이 1부터 10000이하여서 입력 받을 수 있는 계단 비용도 1부터 10000사이만 가능한 줄 알고 잘못 이해하고 있었습니다. 해결책은 arr배열에 엄청 큰 수를 초기화 시켜서 문제점을 해결 했습니다.

* **프로그램 코드**
* #1-2) n(1이상 10,000이하 정수)개 계단을 바닥에서 위로 올라가려고 한다.  
  # 계단을 올라갈 때 한 번에 s개 이하의 계단만 오를 수 있으며,  
  # 각 계단은 밟을 때 비용이 있다.  
  # 바닥에서 가장 위의 계단으로 올라갈 때 밟는 계단의 비용 합이 최소가 되도록 하면서 올라가고자 한다.  
  # 이때의 비용합(최소 비용)을 구하는 프로그램을 작성하시오.  
  def min\_costs\_step(n, s\_stair, cost):  
   if 1 <= n <= 10000:  
   cost.insert(0, 0) # 0번째 계단 비용를 추가함  
   arr = [100000000000 for i in range(n+1)] # 0부터 n까지 큰 수 를 담은 arr배열  
   # n번째 계단까지 최소비용을 출력하는 배열을 만듦  
   arr[0] = 0 # 0번째 계단은 비용이 0임  
   s\_li=list((range(1, s\_stair+1))) # s개 이하의 계단오르는 값을 배열에 저장  
   for i in range(n+1): # 1-2문제에서는 s개 이하의 계단만 오른다고 했으므로 몇개 오로는지 입력받아야 알 수 있음  
   for k in s\_li: # s\_li배열의 값들을 하나씩 검사  
   if i >=k: # arr배열에 저장할 인덱스 값과 s개 계단을 하나씩 검사해서  
   arr[i] = min(arr[i], arr[i-k]+ cost[i]) # arr에 저장된 최소비용과 전 계단과 다음 밟을 계단의 비용을  
   # 더한 값중에 최소인 값을 i번째 arr배열에 갱신해 나아감.  
   # 계단 최소비용중에 최소값을 arr배열에 저장  
   else:  
   break # 위의 조건을 만족하지 않은면 break  
   if arr[n] == 100000000000: # 초기화된 큰 수가 계속 남아있다면 return -1  
   return -1  
   else: # 정상적으로 최소비용이 출력될 경우  
   return arr[n]  
  n, s\_stair = list(map(int, input().split())) # n개의 계단과 s개 이하의 계단을 오르는 값  
  cost = list(map(int, input().split())) # 계단 오르는데 비용  
  print(min\_costs\_step(n, s\_stair, cost))
* **문제 2-1번**

문 2) 다음과 같은 개의 행과 개의 열의 셀(cell)들로 이루어진 격자가 있다. 여기서 가장 위의 행은 행 1이고 가장 아래의 행은 행 이며, 가장 왼쪽 열은 열 1이고 가장 오른쪽 열은 열 이다. 셀 는 번째 행과 번째 열의 셀을 나타낸다. 각 셀 에는 비용 이 주어진다. 아래 그림은 격자 예를 보여준다. 여기서 = 2, ..., = 8이다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 열 1 | 열 2 | 열 3 | 열 4 | 열 5 |
| 행 1 | 2 | 8 | 9 | 5 | 8 |
| 행 2 | 4 | 9 | 6 | 5 | 3 |
| 행 3 | 6 | 7 | 5 | 2 | 1 |
| 행 4 | 3 | 2 | 5 | 4 | 8 |

2-1) 셀 (1,1)에서 오른쪽 방향 혹은 아래쪽 방향으로만 가면서 셀 까지 가는 서로 다른 경로의 개수를 구하는 프로그램을 동적계획법을 이용하여 작성하시오. **경로의 개수를 9999999로 나눈 나머지를 출력한다.**

입력: 첫 번째 줄에 과 이 주어진다. 다음 각 줄에 각 행(위로부터 아래로)의 셀의 비용이 주어진다. 문 2-2)는 마지막 줄에 p, q가 주어진다.

출력:

2-1) 위의 문 2-1) 경로의 개수를 9999999로 나눈 나머지를 출력한다.

* **알고리즘 및 자료구조**

입력 받은 m, n으로 각 셀에 0으로 구성된 m\*n격자를 생성

For 구문 2개를 돌려서 if 구문으로 1행1열의 셀 일 경우 셀의 값은 1

행과 열이 1이 아닐 경우 부분문제 최적해 구하는 함수로

Grd[i][j] = grd[i-1][j] + grd[i][j-1]

행과 열의 끝 셀에 경로의 개수의 값이 있으므로 9999999로 나눈 값 출력

* **시간복잡도 분석**

입력 받은 행과 열의 값에 시간복잡도가 결정됩니다. For 구문을 m과 n 인자로 검사하게 되므로

행이 m, 열이 n일 경우 시간 복잡도는 O(mn)입니다.

* **느낀 점**

이론만으로는 격자상에서 최소비용 구하는 과정은 처음에 바로 이해하기 어려웠습니다. 하지만 동적계획법으로 부분 문제들을 세워서 코딩함으로써 구하는 과정이 쉽게 구해지는 것을 확인하며 이해할 수 있었습니다.

* **프로그램 코드**
* #2) 다음과 같은 개의 행과 개의 열의 셀(cell)들로 이루어진 격자가 있다.  
  # 여기서 가장 위의 행은 행 1이고 가장 아래의 행은 행 이며,  
  # 가장 왼쪽 열은 열 1이고 가장 오른쪽 열은 열 이다. 셀 는 번째 행과 번째 열의 셀을 나타낸다.  
  # 각 셀 에는 비용 이 주어진다. 아래 그림은 격자 예를 보여준다. 여기서 C(1,1)= 2, ..., C(4,5) = 8이다.  
    
  # 2-1) 셀 (1,1)에서 오른쪽 방향 혹은 아래쪽 방향으로만 가면서  
  # 셀 까지 가는 서로 다른 경로의 개수를 구하는 프로그램을 동적계획법을 이용하여 작성하시오.  
  # 경로의 개수를 9999999로 나눈 나머지를 출력한다.  
  def mngrid\_solution(m, n): # m\*n격자 경로 개수 출력 함수  
   if 1<= n and m <=2000:  
   grd = [[0 for \_ in range(n + 1)] for \_ in range(m + 1)] # m\*n격자를 생성함 각 셀의 값은 0  
   for i in range(1, m + 1): # for 구문을 m 과 n을 검사함  
   for j in range(1, n + 1):  
   if i == 1 and j == 1: # i = 1과 j = 1 일 경우 격자[1][1] 의 값은 1  
   grd[i][j] = 1  
   else: # 행과 열이 1이 아닐 경우  
   grd[i][j] = grd[i - 1][j] + grd[i][j - 1] # 부분문제 최적 해 값(목적함수)의 재귀적 정의  
   return grd[m][n] % 9999999 # 경로의 개수를 9999999로 나눈 나머지 출력  
    
    
  m, n = map(int, input().split()) # m\*n격자 행과 열을 입력받음  
  print(mngrid\_solution(m, n))
* **문제 2-2번**

2-2) 셀 (1,1)에서 오른쪽 방향 혹은 아래쪽 방향으로만 가면서 셀 까지 가는 경로의 최소비용을 구하는 프로그램을 동적계획법을 이용하여 작성하시오. 경로의 비용이란 지나가는 셀의 비용의 총합이다. **단, 특정 셀 (p,q)를 반드시 지나가야 한다.**

입력: 위의 입력 마지막 줄에 p q가 주어진다. 단, (p,q)는 (1,1)과 (m,n)은 아니다.

출력:

2-2) 위의 문 2-2) 경로의 최소 비용을 출력한다.

* **알고리즘 및 자료구조**

경로의 최소비용으로 구하는 과정을 총 2가지 부분으로 나눠서 설명할 수 있습니다.

첫 번째 부분은 (1,1)에서 (p,q)까지의 최소비용을 구하는 과정입니다.

0행일 경우 부분 문제의 해의 값을 저장하는 테이블을 만드는 함수를 작성하고

0행이 아닐 경우 부분문제의 최소비용을 구해서 새로운 배열에 저장합니다.

행을 검사하면서 구한 최소비용 배열에 갱신합니다.

(p,q)까지 구한 최소비용을 (p,q)셀에 값을 대입합니다.

두 번째 부분은 (p,q)에서 (m,n)까지의 최소비용을 구하는 과정입니다.

q-1행일 경우 부분 문제의 해의 값을 저장하는 테이블을 만드는 함수를 작성하고

q-1행이 아닐 경우 부분문제의 최소비용을 구해서 새로운 배열에 저장합니다.

행을 검사하면서 구한 최소비용 배열에 갱신합니다.

이렇게 구한 부분 문제의 테이블 배열에서 제일 마지막 최소 비용의 셀 값을 출력

* **시간복잡도 분석**

(1,1) 에서 (p,q)까지 또한 (p,q)에서 (m,n)까지 최소비용을 구하는 for구문이 2개씩 들어가므로

입력 받은 m,n값과 p,q값에 따라 결정됩니다. 그러므로 시간복잡도는 O(mn+pq)입니다.

* **느낀 점**

Case를 나눠서 설계하는 과정에서 아이디어가 바로 떠오르지 않아 시간을 제일 잡아먹은 문제입니다. 특히 p,q에서 m,n까지의 최소비용을 구하는 for구문에서 인자 값의 시작 위치를 잡아주는 부분에서 힘들었습니다.

* **프로그램 코드**
* # 2-2) 셀 (1,1)에서 오른쪽 방향 혹은 아래쪽 방향으로만 가면서 셀 까지 가는 경로의 최소비용을  
  # 구하는 프로그램을 동적계획법을 이용하여 작성하시오. 경로의 비용이란 지나가는 셀의 비용의 총합이다.  
  # 단, 특정 셀 (p,q)를 반드시 지나가야 한다.  
    
  # 입력: 위의 입력 마지막 줄에 p q가 주어진다. 단, (p,q)는 (1,1)과 (m,n)은 아니다.  
  def mngrid\_solution(m, n):  
   if 1<= m and n<= 2000:  
   AA\_pq = [0] \* n # (1,1)부터 (p,q)까지의 경로의 최소비용 총합 배열  
   pq\_mn = [0] \* n # (p,q)부터 (m,n)까지의 경로의 최소 비용 총합 배열  
   A\_copy = [0] \* n # AA\_pq에서 copy()을 이용해서 부분문제의 해의 값을 저장하는 테이블  
   B\_copy = [0] \* n # pq\_mn에서 copy()을 이용해서 부분문제의 해의 값을 저장하는 테이블  
   row = [0] \* m # 입력 받은 셀 비용 배열  
   n\_row = [0] \* m # 새로운 입력받은 셀 비용 배열  
   for i in range(m - 1, -1, -1): # 입력받은 셀 비용값을 정리함  
   row[i] = list(map(int, input().split())) # 입력된 행기준으로 배열에 대입  
   for i in range(0, m):  
   n\_row[i] = row[m - i - 1]  
   p, q = map(int, input().split()) # p,q 입력 받기  
    
   for a in range(q): # 부분문제의 해의 값 저장하는 테이블  
   if a == 0:  
   AA\_pq[a] = n\_row[0][a] # 1행 밑에 0행의 부분문제를 구하기 위한 배열 생성  
   else:  
   AA\_pq[a] = AA\_pq[a - 1] + n\_row[0][a]  
   A\_copy = AA\_pq.copy()  
   for i in range(1, p): # (1,1)에서 (p,q)까지 최소비용 구함  
   for j in range(q): # 문제에서 p,q 좌표가 1,1 m,n일 경우는 없다고 했으므로 q=1일 경우는 생략합니다.  
   if j == 0: # 0열 일 경우  
   AA\_pq[j] = AA\_pq[j] + n\_row[i][j]  
   else: # 0열이 아닐경우  
   AA\_pq[j] = n\_row[i][j] + min(AA\_pq[j - 1], A\_copy[j]) # 점화식  
   A\_copy = AA\_pq.copy()  
   n\_row[p - 1][q - 1] = AA\_pq[q - 1] # (p,q)에 방금 구한 최소비용 대입  
   for a in range(q - 1, n): # (p,q)에서 (m,n)까지 구하기 위한 부분문제의 해의 값을 저장하는 테이블  
   if a == q - 1:  
   pq\_mn[a] = n\_row[p - 1][a]  
   else:  
   pq\_mn[a] = pq\_mn[a - 1] + n\_row[p - 1][a]  
   B\_copy = pq\_mn.copy()  
   for i in range(p, m): # (p,q)에서 (m,n)까지 최소비용 구함  
   for j in range(q - 1, n): # 문제에서 p,q 좌표가 1,1 m,n일 경우는 없다고 했으므로 j=q일 경우는 생략합니다.  
   if j == q - 1: # 비용계산하는 첫 열일 경우  
   pq\_mn[j] = pq\_mn[j] + n\_row[i][j]  
   else: # 첫 열이 아닐 경우  
   pq\_mn[j] = n\_row[i][j] + min(pq\_mn[j - 1], B\_copy[j])  
   B\_copy = pq\_mn.copy()  
   print(pq\_mn[-1]) # 부분 문제의 테이블 배열에서 제일 마지막 최소비용을 출력  
    
  m, n = map(int, input().split()) # m\*n격자 테이블 행과 열값 입력  
  mngrid\_solution(m, n)
* **문제 2-3번**

2-3) 가장 위의 행의 어떤 셀로부터 왼**쪽 아래 대각선 방향 혹은 아래쪽 방향 혹은 오른쪽 아래 대각선 방향으로만 가면서** 가장 아래 행의 어떤 셀까지 가는 경로의 최소비용을 구하는 프로그램을 동적계획법을 이용하여 작성하시오. 경로의 비용이란 지나가는 셀의 비용의 총합이다.

**요구조건: 부분문제의 해의 값를 저장하는 테이블로 1차원 배열(리스트)를 사용해야 하고, 이 배열의 크기는 O(n)이어야 함.**

입력: 첫 번째 줄에 과 이 주어진다. 다음 각 줄에 각 행(위로부터 아래로)의 셀의 비용이 주어진다. 문 2-2)는 마지막 줄에 p, q가 주어진다.

출력:

2-2) 위의 문 2-3) 경로의 최소 비용을 출력한다.

* **알고리즘 및 자료구조**

부분 문제 해결을 위한 테이블 1행 비용 값을 새로운 배열에 옮김

첫번째 for구문은 1행일 때 와 2행 이상일때를 나눠서 검사합니다.

두번째 for구문은 열을 검사합니다.

이때 열은 1열의 경우 왼쪽열은 없으므로 1열일 경우와 1열이 아닐 경우로 나눠서 검사합니다.

1열이 아닐 경우 부분 문제를 오른쪽 대각선, 왼쪽 대각선, 아래쪽방향으로 나눠서 검사합니다.

이렇게 구한 최소비용 마지막 배열에서 최소값을 출력합니다.

* **시간복잡도 분석**

입력 받은 m, n값에 따라 for구문의 인자 값으로 시간 복잡도는 결정되므로 시간 복잡도는 O(mn)입니다.

* **느낀 점**

최소 비용을 구하는 과정에서 1열의 경우와 1열이 아닐 경우로 case를 나눠서 설계해야 된다는 것과 오른쪽, 왼쪽, 아래쪽 부분문제를 따로 코딩해야 된다는 것을 알고 있었지만 막상 코드화 하는 부분에서 애를 먹었습니다.

* **프로그램 코드**
* # 2-3) 가장 위의 행의 어떤 셀로부터 왼쪽 아래 대각선 방향 혹은 아래쪽 방향  
  # 혹은 오른쪽 아래 대각선 방향으로만 가면서 가장 아래 행의 어떤 셀까지  
  # 가는 경로의 최소비용을 구하는 프로그램을 동적계획법을 이용하여 작성하시오.  
  # 경로의 비용이란 지나가는 셀의 비용의 총합이다.  
    
  def mngrid\_solution(m, n):  
   AA\_mn = [0] \* n # 1행부터 마지막행 까지의 경로의 최소비용 총합 배열  
   AA\_result = [0] \* n # AA\_mn으로 부터 최소비용값 copy()값  
   row = [0] \* m # 입력받은 셀 비용 배열  
   n\_row = [0] \* m # 새로운 입력받은 셀 비용 배열  
   for i in range(m - 1, -1, -1): # 입력받은 셀 비용값을 정리함  
   row[i] = list(map(int, input().split()))  
   for i in range(0, m): # 입력받은 셀 비용값을 n\_row배열에 옮김  
   n\_row[i] = row[m - i - 1]  
   # base case  
   for a in range(n): # 부분문제 해결을 위한 테이블 1행 비용값 옮김  
   AA\_mn[a] = n\_row[0][a]  
    
   for i in range(1, m):  
   if i == 1: # 1행 일때  
   for j in range(n): # 열비용 검사 - > 오른쪽 왼쪽 아래쪽 방향 정하는 부분  
   if n == 1: # 1열 일경우 왼쪽열은 없음  
   AA\_mn[j] = n\_row[i - 1][j] + n\_row[i][j]  
   else: # 1열이 아닌경우 오른쪽왼쪽아래쪽 선택 가능  
   if j == 0: # 아래쪽 방향 선택시 부분문제  
   AA\_mn[j] = min(n\_row[i - 1][j], n\_row[i - 1][j + 1])+n\_row[i][j]  
   elif j == n - 1: # 왼쪽 방향 선택시 부분 문제  
   AA\_mn[j] = min(n\_row[i - 1][j - 1], n\_row[i - 1][j]) +n\_row[i][j]  
   else: # 오른쪽 방향 선택시 부분 문제  
   AA\_mn[j] = min(n\_row[i - 1][j],n\_row[i - 1][j - 1], n\_row[i - 1][j + 1])+n\_row[i][j]  
   AA\_result = AA\_mn.copy() # 구한 최소비용 배열을 새로운 배열에 저장  
   elif i >= 2: # 2행 이상일 때  
   for j in range(n): # 열비용 검사 - > 오른쪽 왼쪽 아래쪽 방향 정하는 부분  
   if n == 1:  
   AA\_mn[j] = n\_row[i][j] + AA\_result[j] # 1열 일경우 왼쪽열은 없음  
   else: # 1열이 아닌경우 오른쪽왼쪽아래쪽 선택 가능  
   if j == 0: # 아래쪽 방향 선택시 부분문제  
   AA\_mn[j] = min(AA\_result[j], AA\_result[j + 1])+n\_row[i][j]  
   elif j == n - 1: # 왼쪽 방향 선택시 부분 문제  
   AA\_mn[j] = min(AA\_result[j - 1], AA\_result[j])+n\_row[i][j]  
   else: # 오른쪽 방향 선택시 부분 문제  
   AA\_mn[j] = min(AA\_result[j - 1], AA\_result[j], AA\_result[j + 1])+ n\_row[i][j]  
   AA\_result = AA\_mn.copy() # 구한 최소비용 배열을 새로운 배열에 저장  
   print(min(AA\_mn)) # 모든 최소 비용값중에 최소값 출력  
    
  m, n = map(int, input().split()) # m\*n격자 행과 열 값 입력  
  mngrid\_solution(m, n)