

Dynamic Programming Assignment

Name: Mohammad Muneer Ahmed

Roll No: 2303A51475

Date: February 06, 2026

Question 1: Teleport Path Problem

Given a 2D grid and the ability to either move right, down, or teleport to any previously visited cell, find the path from top-left to bottom-right that maximizes the sum of values collected.

Algorithm:

1. Initialize two DP tables: one without teleport (dp0) and one with teleport (dp1)
2. Set base values: $dp0[0][0] = dp1[0][0] = \text{grid}[0][0]$
3. Initialize row_best and col_best arrays to track maximum values
4. For each cell (i,j) :
 - a. Calculate $dp0[i][j]$ by taking max of coming from top or left (no teleport)
 - b. Calculate $dp1[i][j]$ by considering teleport from any previous cell
 - c. Track teleport source: $\max(\text{row_best}[i], \text{col_best}[j]) + \text{current value}$
 - d. Update row_best and col_best with current values
5. Return $dp1[n-1][m-1]$ for maximum path value

Pseudocode:

```
function teleportPath(grid, n, m):
    dp0 = 2D array of size n×m initialized with -infinity
    dp1 = 2D array of size n×m initialized with -infinity
    dp0[0][0] = dp1[0][0] = grid[0][0]

    row_best = array of size n initialized with -infinity
    col_best = array of size m initialized with -infinity
    row_best[0] = col_best[0] = grid[0][0]

    for i from 0 to n-1:
        for j from 0 to m-1:
            if i == 0 and j == 0:
                continue

            best_prev0 = -infinity
            if i > 0: best_prev0 = max(best_prev0, dp0[i-1][j])
            if j > 0: best_prev0 = max(best_prev0, dp0[i][j-1])
            dp0[i][j] = best_prev0 + grid[i][j]

            best_prev1 = -infinity
            if i > 0: best_prev1 = max(best_prev1, dp1[i-1][j])
            if j > 0: best_prev1 = max(best_prev1, dp1[i][j-1])

            teleport_here = max(row_best[i], col_best[j]) + grid[i][j]
            dp1[i][j] = max(best_prev1 + grid[i][j], teleport_here)

            row_best[i] = max(row_best[i], dp0[i][j])
            col_best[j] = max(col_best[j], dp0[i][j])

    return dp1[n-1][m-1]
```

Python Code:

```
import sys
input = sys.stdin.readline

n, m = map(int, input().split())
a = [list(map(int, input().split())) for _ in range(n)]

dp0 = [[-10**18]*m for _ in range(n)] # no teleport used
```

```

dp1 = [[-10**18]*m for _ in range(n)] # teleport already used

dp0[0][0] = a[0][0]
dp1[0][0] = a[0][0]

row_best = [-10**18]*n
col_best = [-10**18]*m
row_best[0] = a[0][0]
col_best[0] = a[0][0]

for i in range(n):
    for j in range(m):
        if i == 0 and j == 0:
            continue

        best_prev0 = -10**18
        if i > 0: best_prev0 = max(best_prev0, dp0[i-1][j])
        if j > 0: best_prev0 = max(best_prev0, dp0[i][j-1])
        dp0[i][j] = best_prev0 + a[i][j]

        best_prev1 = -10**18
        if i > 0: best_prev1 = max(best_prev1, dp1[i-1][j])
        if j > 0: best_prev1 = max(best_prev1, dp1[i][j-1])

        teleport_here = max(row_best[i], col_best[j]) + a[i][j]
        dp1[i][j] = max(best_prev1 + a[i][j], teleport_here)

        row_best[i] = max(row_best[i], dp0[i][j])
        col_best[j] = max(col_best[j], dp0[i][j])

print(dp1[n-1][m-1])

```

Input/Output:

```

# day5.py > ...
1 import sys
2 input = sys.stdin.readline
3
4 n, m = map(int, input().split())
5 a = [list(map(int, input().split())) for _ in range(n)]
6
7 dp0 = [[-10**18]*m for _ in range(n)] # no teleport used
8 dp1 = [[-10**18]*m for _ in range(n)] # teleport already used
9
10 dp0[0][0] = a[0][0]
11 dp1[0][0] = a[0][0]
12
13 row_best = [-10**18]*n
14 col_best = [-10**18]*m
15 row_best[0] = a[0][0]
16 col_best[0] = a[0][0]
17
18 for i in range(n):
19     for j in range(m):
20         if i == 0 and j == 0:
21             continue
22
23         best_prev0 = -10**18
24         if i > 0: best_prev0 = max(best_prev0, dp0[i-1][j])
25         if j > 0: best_prev0 = max(best_prev0, dp0[i][j-1])
26         dp0[i][j] = best_prev0 + a[i][j]
27
28         best_prev1 = -10**18
29         if i > 0: best_prev1 = max(best_prev1, dp1[i-1][j])
30         if j > 0: best_prev1 = max(best_prev1, dp1[i][j-1])
31
32         teleport_here = max(row_best[i], col_best[j]) + a[i][j]
33         dp1[i][j] = max(best_prev1 + a[i][j], teleport_here)
34
35         row_best[i] = max(row_best[i], dp0[i][j])
36         col_best[j] = max(col_best[j], dp0[i][j])
37
38 print(dp1[n-1][m-1])
39

```

PROMPTS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

python -u "/Users/mohammadneerahmed/Documents/training2.py/day5.py"
● mohammadneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadneerahmed/Documents/training2.py/day5.py"
3 3
1 2 3
4 5 6
7 8 9
29

```

Question 2: Longest Common Subsequence

Find the length of the longest common subsequence between two strings where matching characters can be selected from either string.

Algorithm:

1. Initialize DP array of size n where $dp[i]$ represents the LCS ending at position i
2. For each position i in string s:
 - a. Initialize $dp[i]$ to $i+1$ (diagonal value)
3. For each window length from 2 to $n+1$:
 - a. For each position l in the window:
 - Calculate $r = l + \text{length} - 1$
 - $dp[l][r] = \max(dp[l+1][r], dp[l][r-1])$
 - If $s[l] == s[r]$: add matching pair contribution
 - If $l+1 \leq r-1$: add $dp[l+1][r-1]$
 - Update $dp[l][r]$ with maximum value
4. Return $dp[0][n-1]$

Pseudocode:

```
function longestPalindromeSubseq(s, n):
    dp = 2D array of size nxn

    // Base case: single characters
    for i from 0 to n-1:
        dp[i][i] = i + 1

    // Fill DP table for increasing lengths
    for length from 2 to n+1:
        for l from 0 to n-length:
            r = l + length - 1
            dp[l][r] = max(dp[l+1][r], dp[l][r-1])

            if s[l] == s[r]:
                add = (l+1) + (r+1)
                if l+1 <= r-1:
                    add += dp[l+1][r-1]
                dp[l][r] = max(dp[l][r], add)

    return dp[0][n-1]
```

Python Code:

```
import sys
input = sys.stdin.readline

n = int(input())
s = input().strip()

dp = [[0]*n for _ in range(n)]

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

for length in range(2, n+1):
    for l in range(n-length+1):
        r = l + length - 1
        dp[l][r] = max(dp[l+1][r], dp[l][r-1])
```

```

if s[l] == s[r]:
    add = (l+1) + (r+1)
    if l+1 <= r-1:
        add += dp[l+1][r-1]
    dp[l][r] = max(dp[l][r], add)

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

```

Input/Output:

```

day5.py > ...
1  import sys
2  input = sys.stdin.readline
3
4  n = int(input().strip())
5  s = input().strip()
6
7  dp = [[0]*n for _ in range(n)]
8
9  for i in range(n):
10     dp[i][i] = i + 1
11
12 for length in range(2, n+1):
13     for l in range(n-length+1):
14         r = l + length - 1
15         dp[l][r] = max(dp[l+1][r], dp[l][r-1])
16         if s[l] == s[r]:
17             add = (l+1) + (r+1)
18             if l+1 <= r-1:
19                 add += dp[l+1][r-1]
20             dp[l][r] = max(dp[l][r], add)
21
22 print(dp[0][n-1])
23

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
5
abca
10
$ mohammadmuneerahmed@Muneers-MacBook-Air training2.py %

```

Question 3: Fenwick Tree with Range Queries

Implement a Fenwick Tree (Binary Indexed Tree) data structure that supports point updates and range sum queries efficiently.

Algorithm:

1. Initialize Fenwick Tree with size $n+1$
2. For $\text{add}(i, v)$ operation:
 - a. Increment index by 1 (1-indexed)
 - b. While $i \leq n$:
 - Add v to $\text{bit}[i]$
 - Move to next position: $i += (i \& -i)$
3. For $\text{sum}(i)$ operation:
 - a. Initialize sum = 0, set $i = i+1$
 - b. While $i > 0$:
 - Add $\text{bit}[i]$ to sum
 - Move to parent: $i -= (i \& -i)$
 - c. Return sum
4. For $\text{range}(l, r)$ query:
 - Return $\text{sum}(r) - \text{sum}(l-1)$ if $l > 0$ else $\text{sum}(r)$

Pseudocode:

```
class Fenwick:  
    function __init__(n):  
        self.n = n  
        self.bit = array of size n+1 initialized with 0  
  
    function add(i, v):  
        i += 1 // Convert to 1-indexed  
        while i <= self.n:  
            self.bit[i] += v  
            i += (i & -i) // Move to next position  
  
    function sum(i):  
        s = 0  
        i += 1  
        while i > 0:  
            s += self.bit[i]  
            i -= (i & -i) // Move to parent  
        return s  
  
    function range(l, r):  
        if l > r: return 0  
        return self.sum(r) - (self.sum(l-1) if l > 0 else 0)
```

Python Code:

```
import sys  
input = sys.stdin.readline  
  
n, q = map(int, input().split())  
arr = list(map(int, input().split()))  
  
class Fenwick:  
    def __init__(self, n):  
        self.n = n
```

```

        self.bit = [0]*(n+1)

    def add(self, i, v):
        i += 1
        while i <= self.n:
            self.bit[i] += v
            i += i & -i

    def sum(self, i):
        s = 0
        i += 1
        while i > 0:
            s += self.bit[i]
            i -= i & -i
        return s

    def range(self, l, r):
        if l > r: return 0
        return self.sum(r) - (self.sum(l-1) if l > 0 else 0)

fw = Fenwick(n)
for i, v in enumerate(arr):
    fw.add(i, v)

for _ in range(q):
    t = input().split()
    if t[0] == 'U':
        i, x = map(int, t[1:])
        fw.add(i, x - arr[i])
        arr[i] = x
    else:
        l, r, L = map(int, t[1:])
        l = max(0, i-K+1)
        best = -10**30
        for start in range(l, r-L+2):
            s = fw.range(start, start+L-1)
            best = max(best, s)
        print(f"{best/L:.6f}")

```

Input/Output:

```

day5.py > ...
1 import sys
2 input = sys.stdin.readline
3
4 n, q = map(int, input().split())
5 arr = list(map(int, input().split()))
6
7 class Fenwick:
8     def __init__(self, n):
9         self.n = n
10        self.bit = [0]*(n+1)
11    def add(self, i, v):
12        i += 1
13        while i <= self.n:
14            self.bit[i] += v
15            i += i & -i
16    def sum(self, i):
17        s = 0
18        i += 1
19        while i > 0:
20            s += self.bit[i]
21            i -= i & -i
22        return s
23    def range(self, l, r):
24        if l > r: return 0
25        return self.sum(r) - (self.sum(l-1) if l > 0 else 0)
26
27 fw = Fenwick(n)
28 for i,v in enumerate(arr):
29     fw.add(i, v)
30
31 for _ in range(q):
32     t = input().split()
33     if t[0] == 'U':
34         i, x = map(int, t[1:])
35         fw.add(i, x - arr[i])
36         arr[i] = x
37     else:
38         l, r, L = map(int, t[1:])
39         best = -10**30
40         for start in range(l, r-L+2):
41             s = fw.range(start, start+L-1)
42             best = max(best, s)
43         print(f"{best/L:.6f}")
44
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@Munes-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
5 2 3 4 5
q 0 4 3
4.000000

```

Question 4: Job Scheduling Problem

Given n jobs with deadlines and profits, schedule jobs to maximize total profit where each job takes one time slot and must be completed before its deadline.

Algorithm:

1. Sort jobs in descending order by profit
2. Initialize Union-Find structure for time slots (0 to n)
3. Initialize parent array where $\text{parent}[i] = i$
4. Define $\text{find}(x)$ function with path compression
5. Define $\text{union}(a,b)$ function to merge time slots
6. For each job (profit, deadline, index):
 - a. Find available slot = $\text{find}(\min(\text{deadline}, n))$
 - b. If slot > 0:
 - Add profit to total
 - Add job index to chosen list
 - Union current slot with previous slot (slot-1)
7. Return total profit and scheduled jobs

Pseudocode:

```
function jobScheduling(jobs, n):
    sort jobs by profit in descending order
    parent = array from 0 to n

    function find(x):
        if parent[x] != x:
            parent[x] = find(parent[x])
        return parent[x]

    function union(a, b):
        a = find(a)
        b = find(b)
        parent[a] = b

    profit = 0
    chosen = empty list

    for each (p, d, i) in jobs:
        slot = find(min(d, n))
        if slot > 0:
            profit += p
            chosen.append(i)
            union(slot, slot-1)

    return profit, chosen
```

Python Code:

```
import sys
input = sys.stdin.readline

n = int(input())
jobs = []
for idx in range(n):
    d, p = map(int, input().split())
    jobs.append((p, d, idx+1))
```

```

jobs.sort(reverse=True)

parent = list(range(n+1))

def find(x):
    if parent[x] != x:
        parent[x] = find(parent[x])
    return parent[x]

def union(a, b):
    a = find(a)
    b = find(b)
    parent[a] = b

profit = 0
chosen = []

for p, d, i in jobs:
    slot = find(min(d, n))
    if slot > 0:
        profit += p
        chosen.append(i)
        union(slot, slot-1)

print(profit)
print(*chosen)

```

Input/Output:

```

day5.py > ...
1   import sys
2   input = sys.stdin.readline
3
4   n, Q = map(int, input().split())
5   a = list(map(int, input().split()))
6
7   class Fenwick:
8       def __init__(self, n):
9           self.n=n
10          self.bit=[0]*(n+1)
11      def add(self,i,v):
12          i+=1
13          while i<=self.n:
14              self.bit[i]+=v
15              i+=i&-i
16      def sum(self,i):
17          s=0; i+=1
18          while i>0:
19              s+=self.bit[i]
20              i-=i&-i
21          return s
22      def range(self,l,r):
23          return self.sum(r)-(self.sum(l-1) if l>0 else 0)
24
25  fw = Fenwick(n)
26  for i,v in enumerate(a):
27      fw.add(i,v)
28
29  for _ in range(Q):
30      t = input().split()
31      if t[0] == 'U':
32          i,x = map(int,t[1:])
33          fw.add(i, x-a[i])
34          a[i]=x
35      else:
36          i,K = map(int,t[1:])
37          l = max(0, i-K+1)
38          print(fw.range(l,i))
39

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@luneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
0
1 2 3 4 5 6
s 5 3
15

```

Question 5: Coin Change Problem (Greedy)

Given a target amount T and C types of coins, find the minimum number of coins needed to make the amount using a greedy approach.

Algorithm:

1. Read target amount T and number of coin types C
2. Read coin values and counts for each type
3. For each coin type (value, count):
 - a. Set k = 1 (multiplier for coin value)
 - b. While count > 0:
 - Calculate how many of this coin type to take: min(k, count)
 - Store (value*take, take) in items list
 - Decrease count by take
 - Double k for next iteration (k <= 1)
4. Initialize DP array with size T+1, set dp[0] = 0
5. For each (value, count) in items:
 - a. For each amount from T down to value:
 - dp[amount] = min(dp[amount], dp[amount-value] + count)
6. Return dp[T] if valid, else -1

Pseudocode:

```
function coinChange(T, coins):
    items = empty list

    for (value, count) in coins:
        k = 1
        while count > 0:
            take = min(k, count)
            items.append((value * take, take))
            count -= take
            k <= 1

    INF = 10**18
    dp = array of size T+1 filled with INF
    dp[0] = 0

    for (val, cnt) in items:
        for s from T down to val:
            dp[s] = min(dp[s], dp[s-val] + cnt)

    return dp[T] if dp[T] != INF else -1
```

Python Code:

```
import sys
input = sys.stdin.readline

T = int(input())
C = int(input())
coins = [tuple(map(int, input().split())) for _ in range(C)]

items = []
for v, c in coins:
    k = 1
    while c > 0:
```

```

take = min(k, c)
items.append((v*take, take))
c -= take
k <= 1

INF = 10**18
dp = [INF]*(T+1)
dp[0] = 0

for val, cnt in items:
    for s in range(T, val-1, -1):
        dp[s] = min(dp[s], dp[s-val] + cnt)

print(dp[T] if dp[T] != INF else -1)

```

Input/Output:

```

day5.py > ...
1  import sys
2  input = sys.stdin.readline
3
4  n = int(input())
5  jobs = []
6  for idx in range(n):
7      d,p = map(int,input().split())
8      jobs.append((p,d,idx+1))
9
10 jobs.sort(reverse=True)
11
12 parent = list(range(n+1))
13
14 def find(x):
15     if parent[x]==x:
16         return parent[x]
17     parent[x]=find(parent[x])
18     return parent[x]
19
20 def union(a,b):
21     a=find(a); b=find(b)
22     parent[a]=b
23
24 profit = 0
25 chosen = []
26
27 for p,d,i in jobs:
28     slot = find(min(d,n))
29     if slot>p:
30         profit += p
31         chosen.append(i)
32         union(slot, slot-1)
33
34 print(profit)
35 print(*chosen)
36

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
4
2 100
1 19
2 7
1 25
127
13
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %

```

Question 6: Running Median with Two Heaps

Maintain a running median of numbers using two heaps (max heap for lower half, min heap for upper half) to support efficient insertion and median queries.

Algorithm:

1. Initialize two heaps: low (max heap) and high (min heap)
2. Define add(x) function:
 - a. If low is empty or $x \leq \text{max of low}$:
 - Push x to low (negated for max heap)
 - b. Else:
 - Push x to high
 - c. Call balance() to maintain heap sizes
3. Define balance() function:
 - a. If $\text{len}(\text{low}) > \text{len}(\text{high}) + 1$:
 - Pop from low and push to high (negate)
 - b. If $\text{len}(\text{high}) > \text{len}(\text{low})$:
 - Pop from high and push to low (negate)
4. Define median() function:
 - Return top of low heap (negated)
5. Process k insertions, computing median after each

Pseudocode:

```
function runningMedian(n, k, array):
    low = max heap (empty)
    high = min heap (empty)

    function add(x):
        if not low or x <= -low[0]:
            heappush(low, -x)
        else:
            heappush(high, x)
        balance()

    function balance():
        if len(low) > len(high) + 1:
            heappush(high, -heappop(low))
        if len(high) > len(low):
            heappush(low, -heappop(high))

    function median():
        return -low[0]

    ans = empty list
    for i from 0 to k-1:
        add(array[i])

    ans.append(median())

    for i from k to n-1:
        window = sorted(array[i-k+1:i+1])
        ans.append(window[(k-1)//2])

    return ans
```

Python Code:

```

import heapq, sys
input = sys.stdin.readline

n, k = map(int, input().split())
a = list(map(int, input().split()))

low = [] # max heap (negated)
high = [] # min heap

def add(x):
    if not low or x <= -low[0]:
        heapq.heappush(low, -x)
    else:
        heapq.heappush(high, x)
balance()

def balance():
    if len(low) > len(high) + 1:
        heapq.heappush(high, -heapq.heappop(low))
    if len(high) > len(low):
        heapq.heappush(low, -heapq.heappop(high))

def median():
    return -low[0]

for i in range(k):
    add(a[i])

ans = [median()]

for i in range(k, n):
    window = sorted(a[i-k+1:i+1])
    ans.append(window[(k-1)//2])

print(*ans)

```

Input/Output:

```

day5.py > ...
1 import sys
2 input = sys.stdin.readline
3
4 T = int(input())
5 C = int(input())
6 coins = [tuple(map(int, input().split())) for _ in range(C)]
7
8 items = []
9 for v,c in coins:
10     k = 1
11     while c>0:
12         take = min(k,c)
13         items.append((v*take, take))
14         c -= take
15         k <= 1
16
17 INF = 10**18
18 dp = [INF]*(T+1)
19 dp[0]=0
20
21 for val,cnt in items:
22     for s in range(T, val-1, -1):
23         dp[s] = min(dp[s], dp[s-val] + cnt)
24
25 print(dp[T] if dp[T]!=INF else -1)
26

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
• mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
11
3
1 5
2
7 1
3
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %

```

Question 7: Array Product Problem

Given two arrays A and B of size n, compute array C where $C[i]$ equals the sum of products $A[j] * B[i-j]$ for all valid j from 0 to i.

Algorithm:

1. Read n, array A, and array B
2. Initialize array C of size n with all zeros
3. For each position i from 0 to n-1:
 - a. Initialize sum s = 0
 - b. For each position j from 0 to i:
 - Add $A[j] * B[i-j]$ to s
 - c. Set $C[i] = s$
4. Output all elements of array C

Pseudocode:

```
function arrayProduct(n, A, B):
    C = array of size n initialized with 0

    for i from 0 to n-1:
        s = 0
        for j from 0 to i:
            s += A[j] * B[i - j]
        C[i] = s

    return C
```

Python Code:

```
import sys
input = sys.stdin.readline

n = int(input())
A = list(map(int, input().split()))
B = list(map(int, input().split()))

C = [0] * n

for i in range(n):
    s = 0
    for j in range(i + 1):
        s += A[j] * B[i - j]
    C[i] = s

print(*C)
```

Input/Output:

```
day5.py > ...
1  import heapq, sys
2  input = sys.stdin.readline
3
4  n,k = map(int,input().split())
5  a = list(map(int,input().split()))
6
7  low = [] # max heap (neg)
8  high = [] # min heap
9
10 def add(x):
11     if not low or x <= -low[0]:
12         heapq.heappush(low,-x)
13     else:
14         heapq.heappush(high,x)
15     balance()
16
17 def balance():
18     if len(low) > len(high)+1:
19         heapq.heappush(high, -heapq.heappop(low))
20     if len(high) > len(low):
21         heapq.heappush(low, -heapq.heappop(high))
22
23 def median():
24     return -low[0]
25
26 for i in range(k):
27     add(a[i])
28
29 ans = [median()]
30
31 for i in range(k,n):
32     # rebuild window (lazy removal omitted for brevity)
33     window = sorted(a[i-k+1:i+1])
34     ans.append(window[(k-1)//2])
35
36 print(*ans)
37
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
7 3
1 5 2 6 3 7 4
2 5 3 6 4
↳ mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```