19CSE302 Design and Analysis of Algorithms

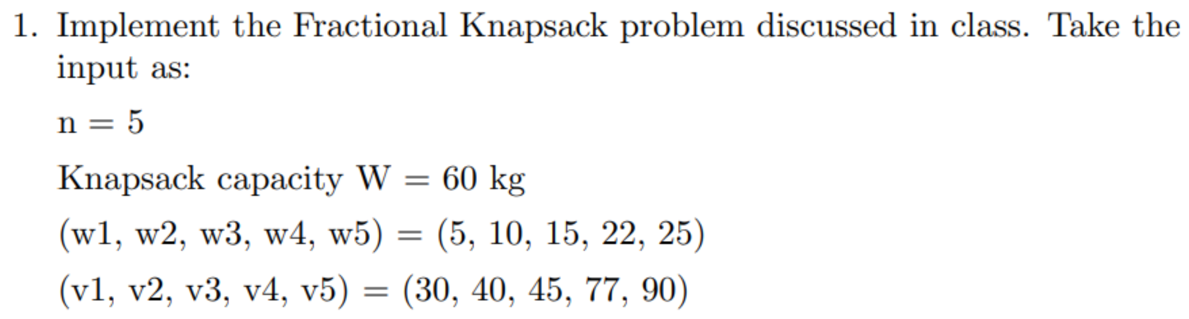
Lab Sheet 9

S Abhishek

AM.EN.U4CSE19147

Colab

Knapsack



1. Implement the Fractional Knapsack problem discussed in class. Take the input as:

   n = 5

   Knapsack capacity W = 60 kg

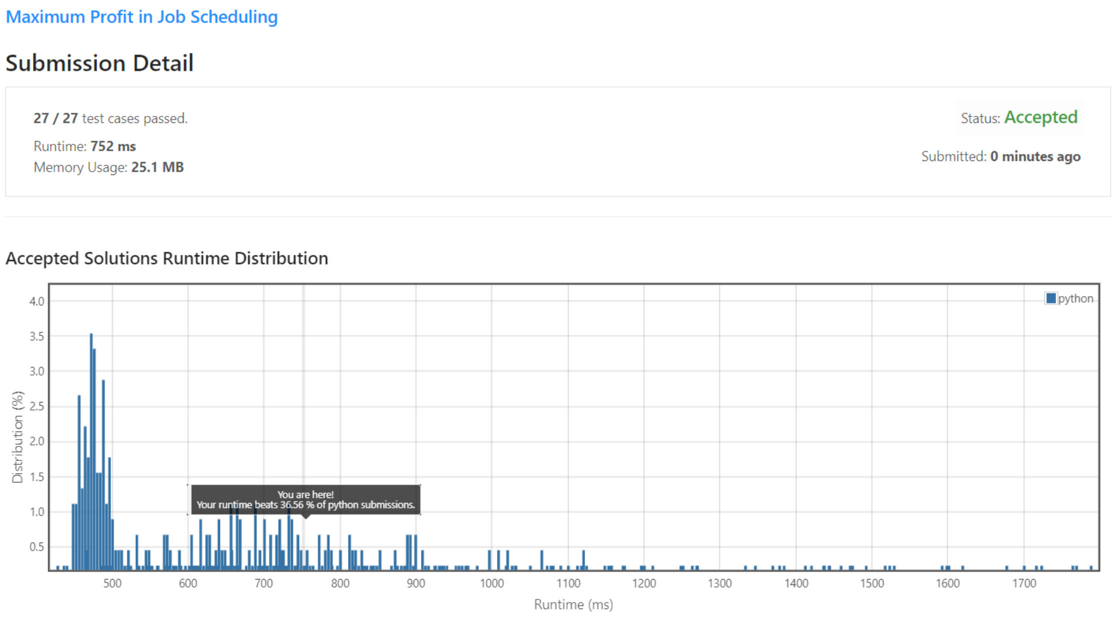
   (w1, w2, w3, w4, w5) = (5, 10, 15, 22, 25)

   (v1, v2, v3, v4, v5) = (30, 40, 45, 77, 90)

```
def KnapSack(W, wt, val):
    n = len(val)
    table = [[0 for x in range(W + 1)] for x in range(n + 1)]
    for i in range(n + 1):
        for j in range(W + 1):
          if i == 0 or j == 0:
            table[i][j] = 0
          elif wt[i-1] <= j:
            table[i][j] = max(val[i-1] + table[i-1][j-wt[i-1]], table[i-1][j])
          else:
            table[i][j] = table[i-1][j]
    return table[n][W]


Capacity = 60

Weight = [5, 10, 15, 22, 25]

Value = [30, 40, 45, 77, 90]

print("The Maximum Value that can be put in a Knapsack of Capacity",
Capacity, "is", KnapSack(Capacity, Weight, Value))
```

```
The Maximum Value that can be put in a Knapsack of Capacity 60 is 207
```

2. Leetcode Problem no. 1235. Maximum Profit in Job Scheduling



Maximum Profit in Job Scheduling

Submission Detail

27 / 27 test cases passed.

Status: Accepted

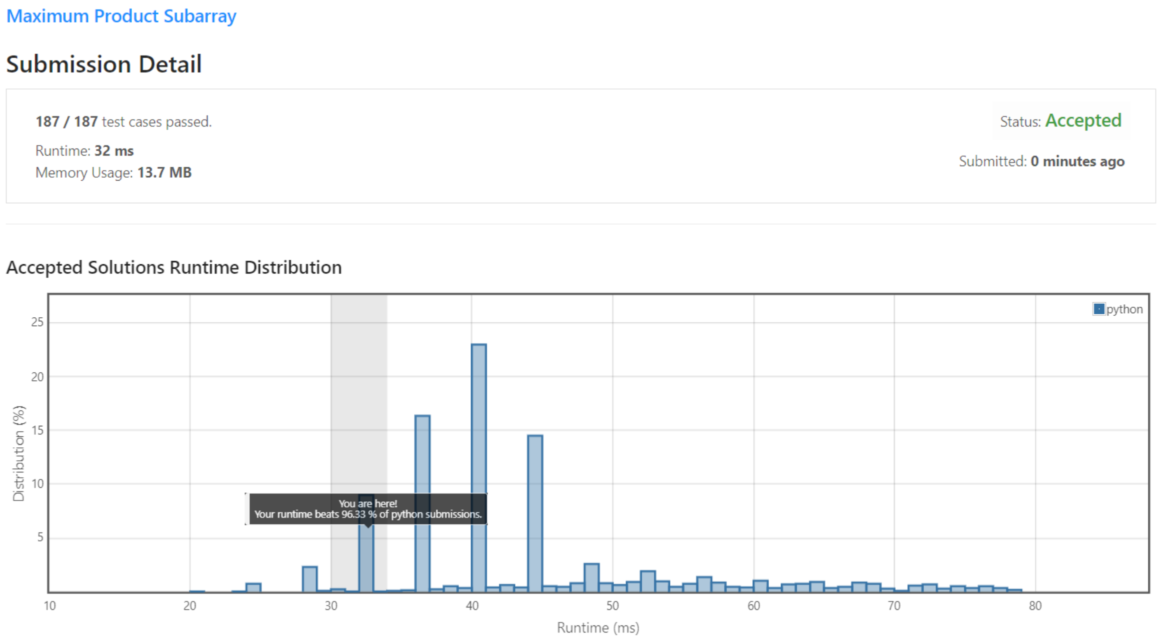
Runtime: 752 ms

Memory Usage: 25.1 MB

Submitted: 0 minutes ago

Accepted Solutions Runtime Distribution

3. Leetcode Problem no.152. Maximum Product Subarray



Maximum Product Subarray

Submission Detail

187 / 187 test cases passed.

Status: Accepted

Runtime: 32 ms

Memory Usage: 13.7 MB

Submitted: 0 minutes ago

Accepted Solutions Runtime Distribution

4. Implement the All pair Shortest Path algorithm

All Pairs Shortest Paths

- Floyd Warshall Algorithm

```
def Path(path, v, u, route):
  if path[v][u] == v:
    return
  Path(path, v, path[v][u], route)
  route.append(path[v][u])


def Display(path, n):
  for v in range(n):
    for u in range(n):
      if u != v and path[v][u] != -1:
        route = [v]
        Path(path, v, u, route)
        route.append(u)
        print(f'The shortest path from {v} —> {u} is', route)

def FloydWarshall(adjMatrix):
  if not adjMatrix:
    return
  n = len(adjMatrix)
```

```
 cost = adjMatrix.copy()
 path = [[None for x in range(n)] for y in range(n)]
 for v in range(n):
  for u in range(n):
   if v == u:
    path[v][u] = 0
   elif cost[v][u] != float('inf'):
    path[v][u] = v
   else:
    path[v][u] = -1

 for k in range(n):
  for v in range(n):
   for u in range(n):
    if cost[v][k] != float('inf') and cost[k][u] != float('inf') and (cost[v][k] + cost[k][u] <
cost[v][u]):
     cost[v][u] = cost[v][k] + cost[k][u]
     path[v][u] = path[k][u]
   if cost[v][v] < 0:
    print('Negative-weight cycle found')
    return
 Display(path, n)


if __name__ == '__main__':
 I = float('inf')
```

```
adjMatrix = [
    [0, I, -2, I],
    [4, 0, 3, I],
    [I, I, 0, 2],
    [I, -1, I, 0]
]

FloydWarshall(adjMatrix)
```

```
The shortest path from 0 —> 1 is [0, 2, 3, 1]

The shortest path from 0 —> 2 is [0, 2]

The shortest path from 0 —> 3 is [0, 2, 3]

The shortest path from 1 —> 0 is [1, 0]

The shortest path from 1 —> 2 is [1, 0, 2]

The shortest path from 1 —> 3 is [1, 0, 2, 3]

The shortest path from 2 —> 0 is [2, 3, 1, 0]

The shortest path from 2 —> 1 is [2, 3, 1]

The shortest path from 2 —> 3 is [2, 3]

The shortest path from 3 —> 0 is [3, 1, 0]

The shortest path from 3 —> 1 is [3, 1]

The shortest path from 3 —> 2 is [3, 1, 0, 2]
```

Thankyou!!