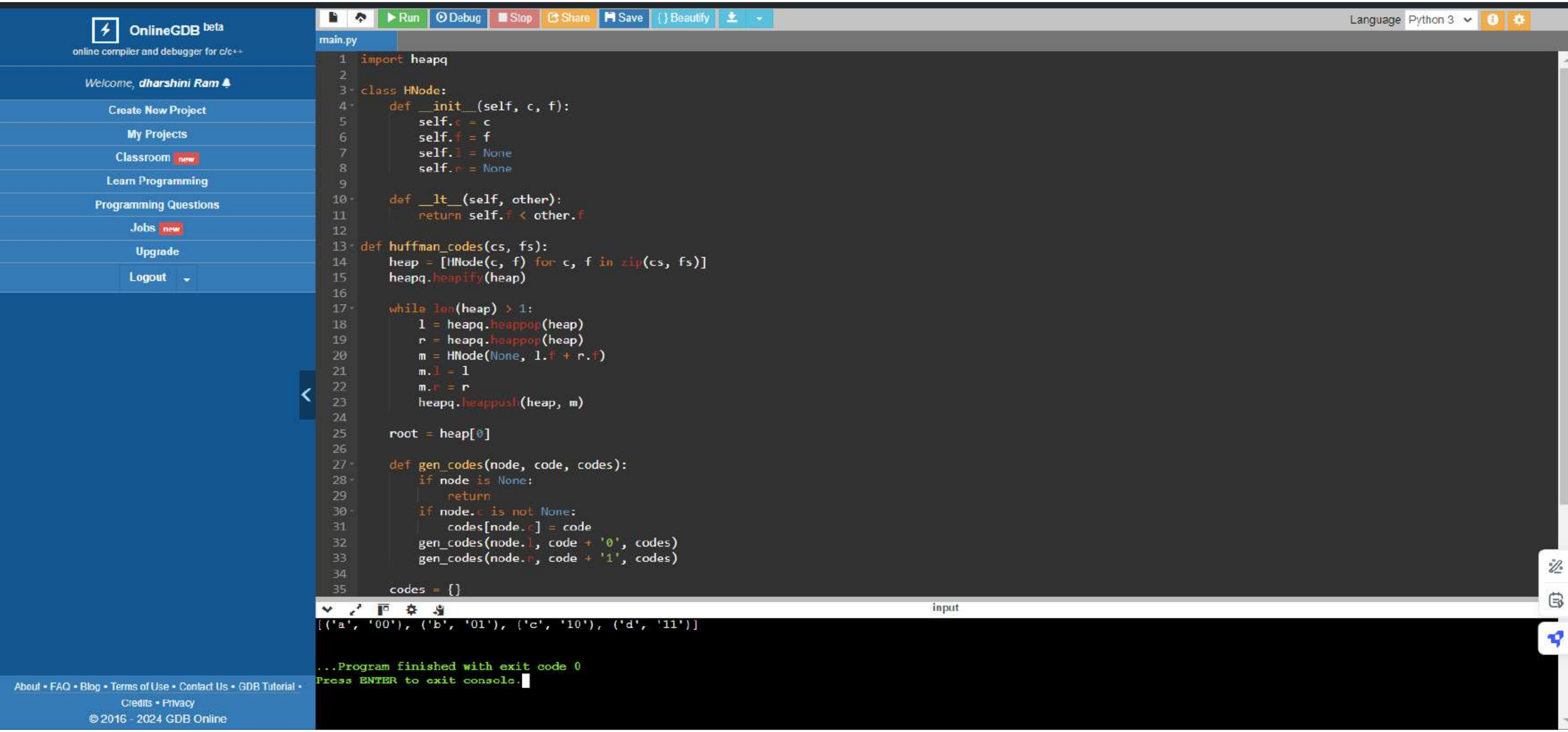
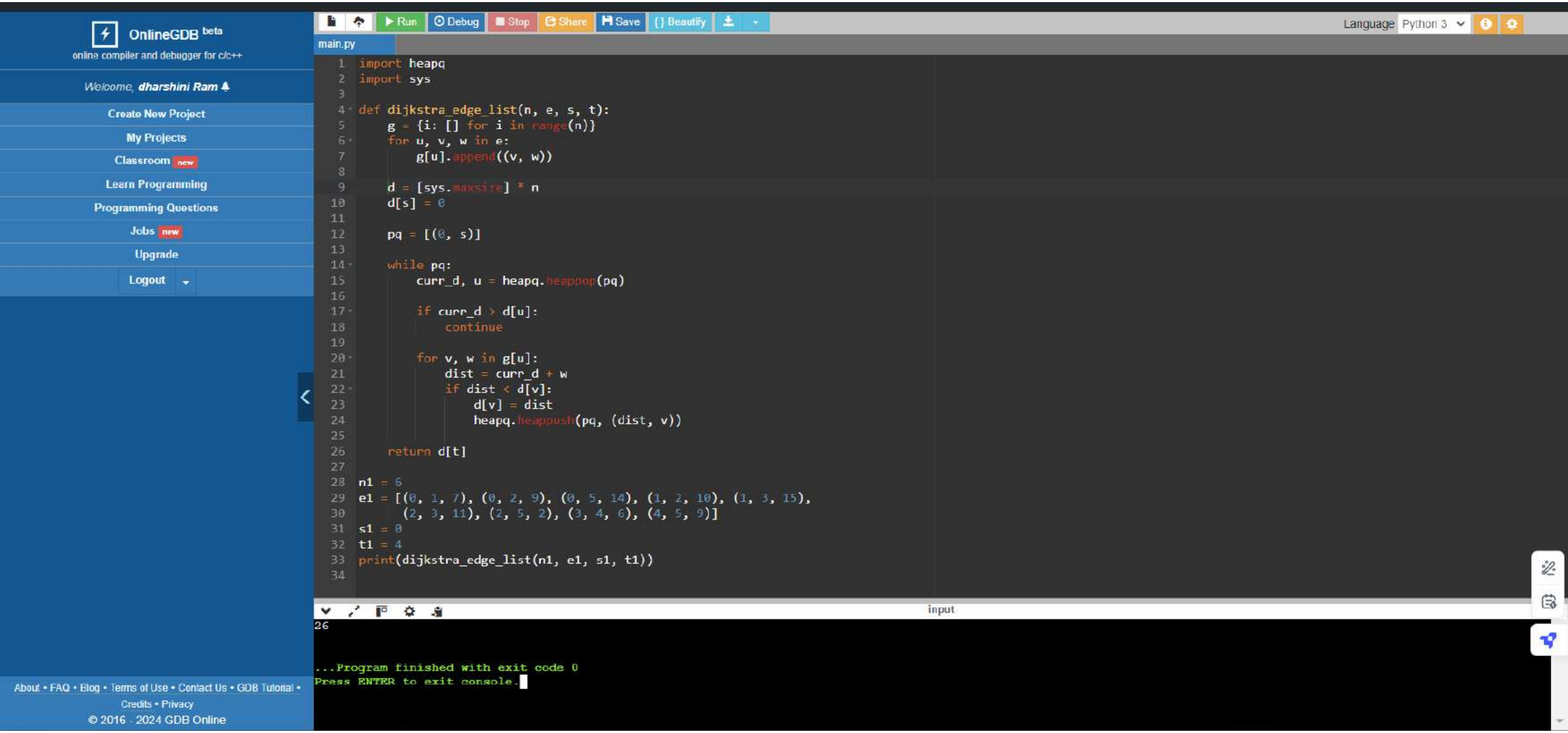


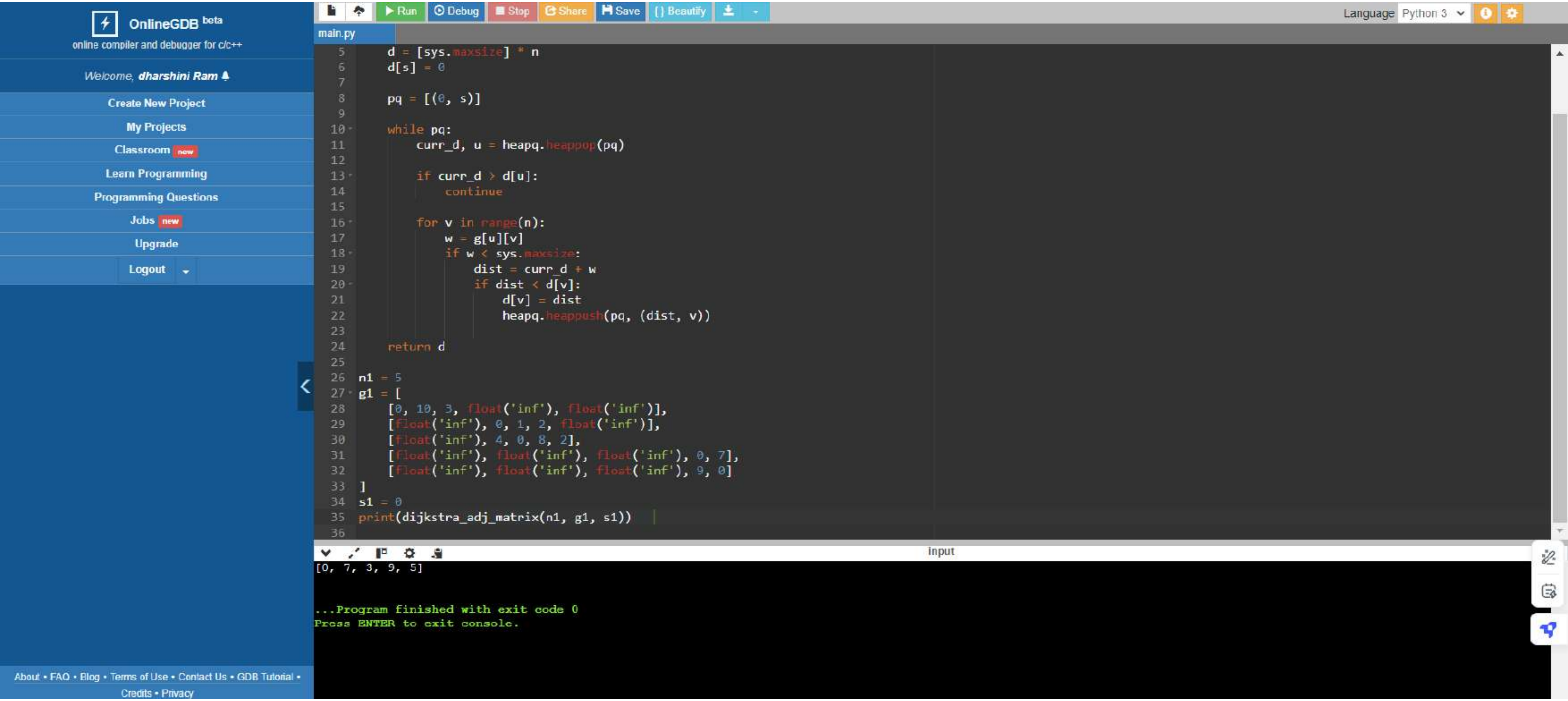
```python
import heapq

class HNode:
    def __init__(self, c, f):
        self.c = c
        self.f = f
        self.l = None
        self.r = None

    def __lt__(self, other):
        return self.f < other.f

def huffman_codes(cs, fs):
    heap = [HNode(c, f) for c, f in zip(cs, fs)]
    heapq.heapify(heap)

    while len(heap) > 1:
        l = heapq.heappop(heap)
        r = heapq.heappop(heap)
        m = HNode(None, l.f + r.f)
        m.l = l
        m.r = r
        heapq.heappush(heap, m)

    root = heap[0]

    def gen_codes(node, code, codes):
        if node is None:
            return
        if node.c is not None:
            codes[node.c] = code
        gen_codes(node.l, code + '0', codes)
        gen_codes(node.r, code + '1', codes)

    codes = {}
```

```
[('a', '00'), ('b', '01'), ('c', '10'), ('d', '11')]


...Program finished with exit code 0
Press ENTER to exit console.
```



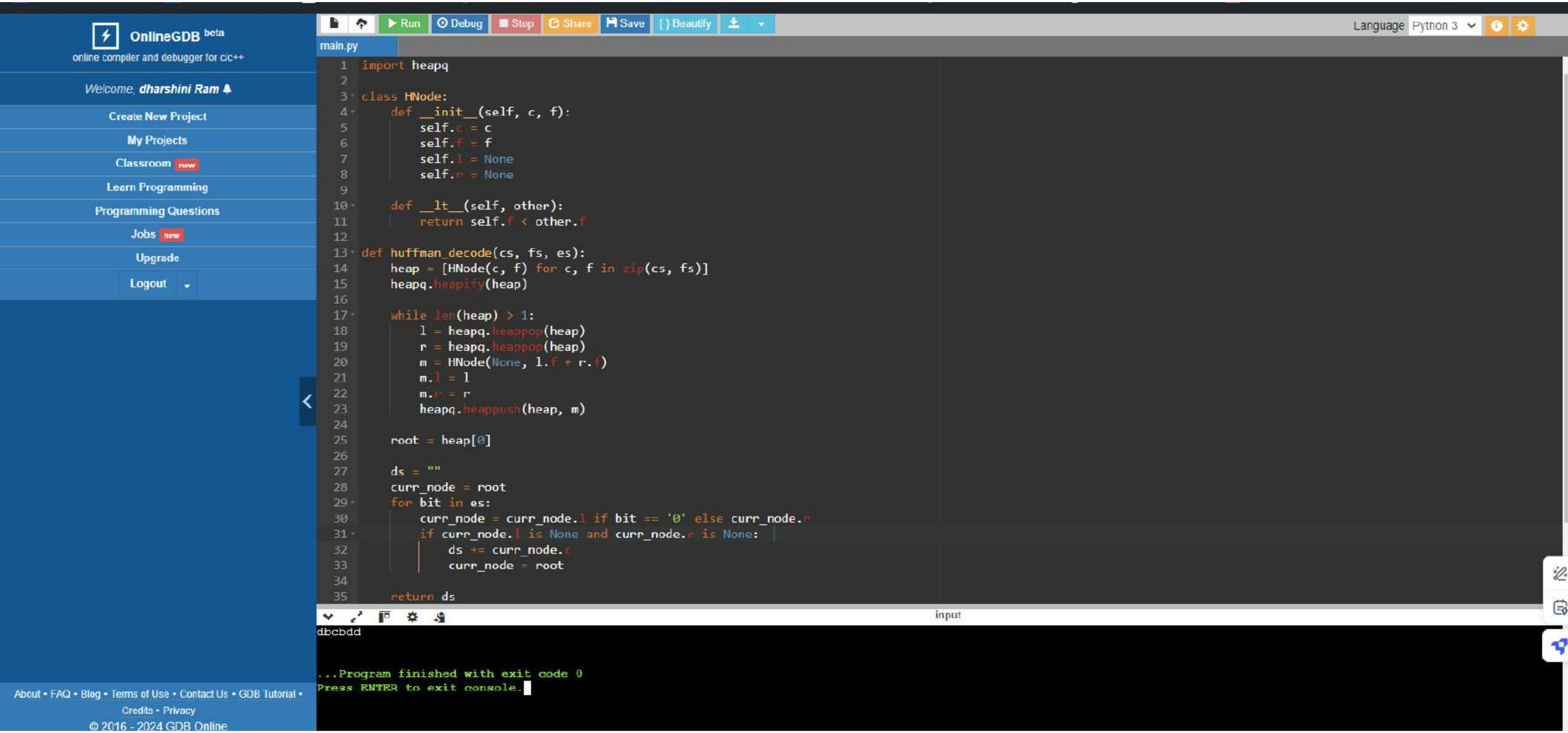
```python
import heapq
import sys

def dijkstra_edge_list(n, e, s, t):
    g = {i: [] for i in range(n)}
    for u, v, w in e:
        g[u].append((v, w))

    d = [sys.maxsize] * n
    d[s] = 0

    pq = [(0, s)]

    while pq:
        curr_d, u = heapq.heappop(pq)

        if curr_d > d[u]:
            continue

        for v, w in g[u]:
            dist = curr_d + w
            if dist < d[v]:
                d[v] = dist
                heapq.heappush(pq, (dist, v))

    return d[t]

n1 = 6
e1 = [(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15),
      (2, 3, 11), (2, 5, 2), (3, 4, 6), (4, 5, 9)]
s1 = 0
t1 = 4
print(dijkstra_edge_list(n1, e1, s1, t1))
```

```
26


...Program finished with exit code 0
Press ENTER to exit console.
```



```
    d = [sys.maxsize] * n
    d[s] = 0

    pq = [(0, s)]

    while pq:
        curr_d, u = heapq.heappop(pq)

        if curr_d > d[u]:
            continue

        for v in range(n):
            w = g[u][v]
            if w < sys.maxsize:
                dist = curr_d + w
                if dist < d[v]:
                    d[v] = dist
                    heapq.heappush(pq, (dist, v))

    return d

n1 = 5
g1 = [
    [0, 10, 3, float('inf'), float('inf')],
    [float('inf'), 0, 1, 2, float('inf')],
    [float('inf'), 4, 0, 8, 2],
    [float('inf'), float('inf'), float('inf'), 0, 7],
    [float('inf'), float('inf'), float('inf'), 9, 0]
]
s1 = 0
print(dijkstra_adj_matrix(n1, g1, s1))

```

```
[0, 7, 3, 9, 5]


...Program finished with exit code 0
Press ENTER to exit console.
```



```
import heapq

class HNode:
    def __init__(self, c, f):
        self.c = c
        self.f = f
        self.l = None
        self.r = None

    def __lt__(self, other):
        return self.f < other.f

def huffman_decode(cs, fs, es):
    heap = [HNode(c, f) for c, f in zip(cs, fs)]
    heapq.heapify(heap)

    while len(heap) > 1:
        l = heapq.heappop(heap)
        r = heapq.heappop(heap)
        m = HNode(None, l.f + r.f)
        m.l = l
        m.r = r
        heapq.heappush(heap, m)

    root = heap[0]

    ds = ""
    curr_node = root
    for bit in es:
        curr_node = curr_node.l if bit == '0' else curr_node.r
        if curr_node.l is None and curr_node.r is None:
            ds += curr_node.c
            curr_node = root

    return ds
```

```
dbcbdd


...Program finished with exit code 0
Press ENTER to exit console.
```