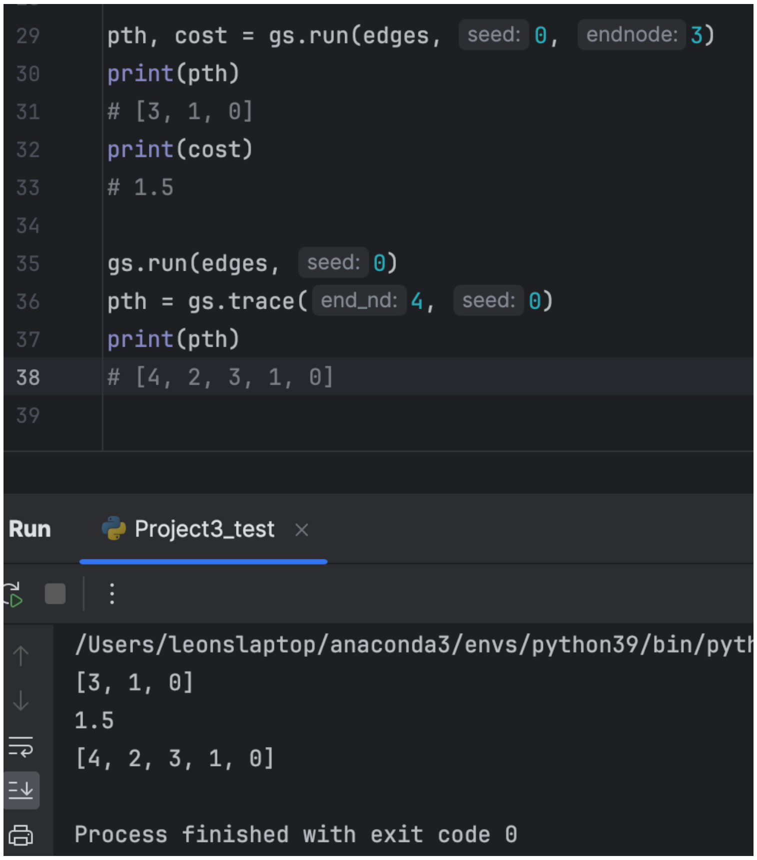
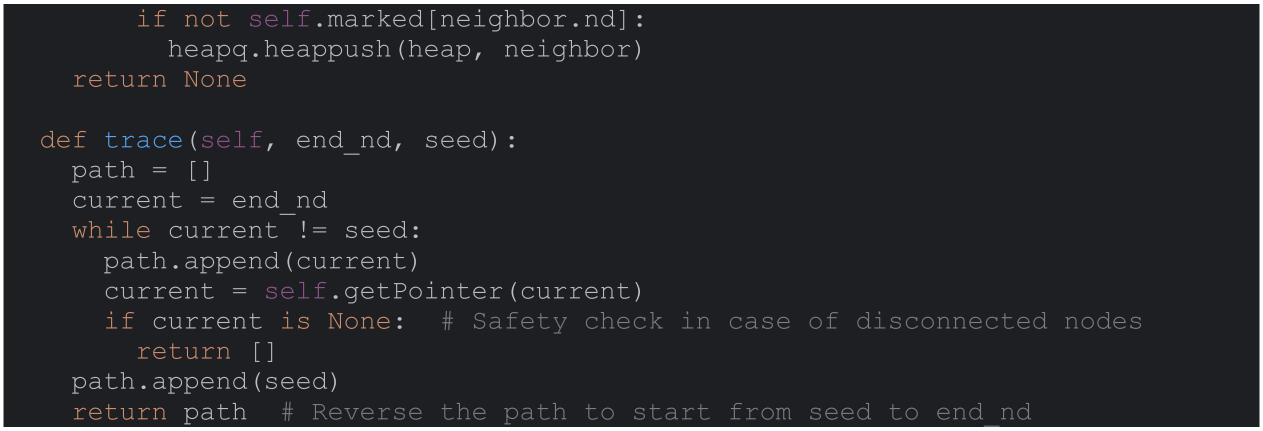


```
pth, cost = gs.run(edges, seed: 0, endnode: 3)
print(pth)
# [3, 1, 0]
print(cost)
# 1.5

gs.run(edges, seed: 0)
pth = gs.trace( end_nd: 4, seed: 0)
print(pth)
# [4, 2, 3, 1, 0]

```

```
/Users/leonslaptop/anaconda3/envs/python39/bin/pyth
[3, 1, 0]
1.5
[4, 2, 3, 1, 0]

Process finished with exit code 0
```

graphSearch.py

```
import copy
import heapq
import numpy as np
from Project3.lwnode import *

class graphSearch:
  def __init__(self, node_type):
    self.node_type = node_type
    # The heap for nodes to visit is kept here since it's a common component
across different
    # search strategies
    self.heap = []

  def run(self, seed, endnode=None):
    heap = []
    start_node = self.node_type(nd=seed)
    heapq.heappush(heap, start_node)
    while heap:
      node = heapq.heappop(heap)
      if self.marked[node.nd]:
        continue
      self.mark(node)
      self.setPointer(node, node.pr)
      if node.nd == endnode:
        return self.trace(node.nd, seed), node.cost
      for neighbor in self.findNeibs(node):
```



```
        if not self.marked[neighbor.nd]:
          heapq.heappush(heap, neighbor)
    return None

  def trace(self, end_nd, seed):
    path = []
    current = end_nd
    while current != seed:
      path.append(current)
      current = self.getPointer(current)
      if current is None:  # Safety check in case of disconnected nodes
        return []
    path.append(seed)
    return path  # Reverse the path to start from seed to end_nd
```