20171248 안재형

CAUSWE 2021

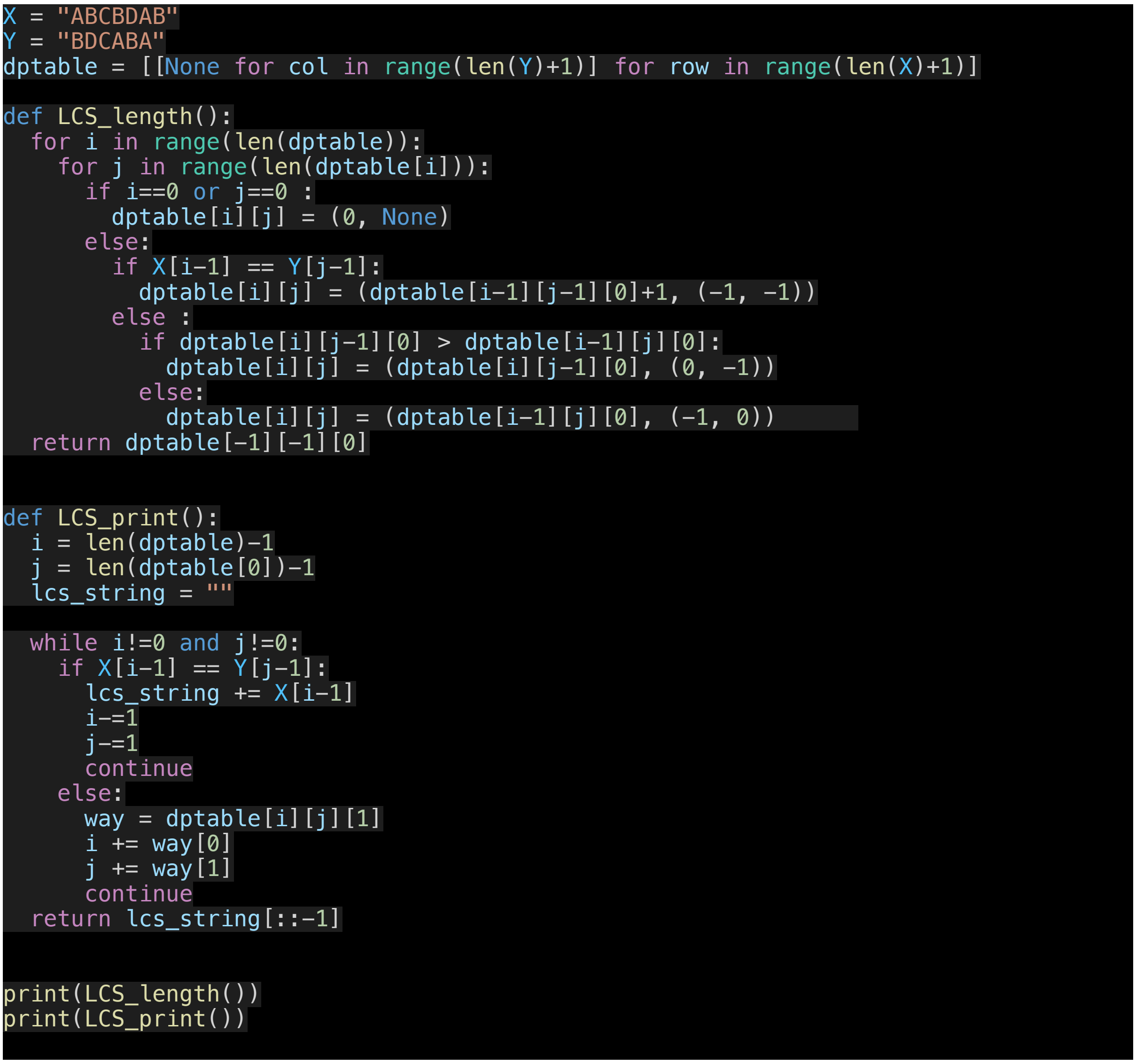
Algorithm Course - Class#2

(Prof.Eunwoo Kim)

Final Assignment

Problem #1

Code



```
X = "ABCBDAB"
Y = "BDCABA"
dptable = [[None for col in range(len(Y)+1)] for row in range(len(X)+1)]

def LCS_length():
  for i in range(len(dptable)):
    for j in range(len(dptable[i])):
      if i==0 or j==0 :
        dptable[i][j] = (0, None)
      else:
        if X[i-1] == Y[j-1]:
          dptable[i][j] = (dptable[i-1][j-1][0]+1, (-1, -1))
        else :
          if dptable[i][j-1][0] > dptable[i-1][j][0]:
            dptable[i][j] = (dptable[i][j-1][0], (0, -1))
          else:
            dptable[i][j] = (dptable[i-1][j][0], (-1, 0))
  return dptable[-1][-1][0]


def LCS_print():
  i = len(dptable)-1
  j = len(dptable[0])-1
  lcs_string = ""

  while i!=0 and j!=0:
    if X[i-1] == Y[j-1]:
      lcs_string += X[i-1]
      i-=1
      j-=1
      continue
    else:
      way = dptable[i][j][1]
      i += way[0]
      j += way[1]
      continue
  return lcs_string[::-1]


print(LCS_length())
print(LCS_print())
```

Result(Output)

```
4
BCBA
```

Problem #2

Code

```
from enum import Enum
class Color(Enum):
  white = 1
  gray = 2
  black = 3

class Vertex:
  def __init__(self):
    self.predecessor = None
    self.d = None
    self.f = None
    self.color = Color.white

class DFS:
  def __init__(self, graph):
    self.graph = graph
    self.graph_dfs = dict(map(lambda x: (x[0], Vertex()), graph))
    self.time = 0

  def visit(self, u):
    self.time += 1
    self.graph_dfs[u].d = self.time
    self.graph_dfs[u].color = Color.gray
    for i in self.graph[u]:
      if self.graph_dfs[i].color == Color.white:
        self.graph_dfs[i].predecessor = u
        self.visit(i)
    self.graph_dfs[u].color = Color.black
    self.time += 1
    self.graph_dfs[u].f = self.time

dfs = DFS({
  'r': ['v', 's'],
  's': ['r', 'w'],
  't': ['u', 'w', 'x'],
  'u': ['t', 'x', 'y'],
  'v': ['r'],
  'w': ['s', 't', 'x'],
  'x': ['t', 'u', 'w', 'y'],
  'y': ['u', 'x'],
})
dfs.visit('s')

for key, value in dfs.graph_dfs.items():
  print(f"{key} - pi : {value.predecessor}, d : {value.d}")
```

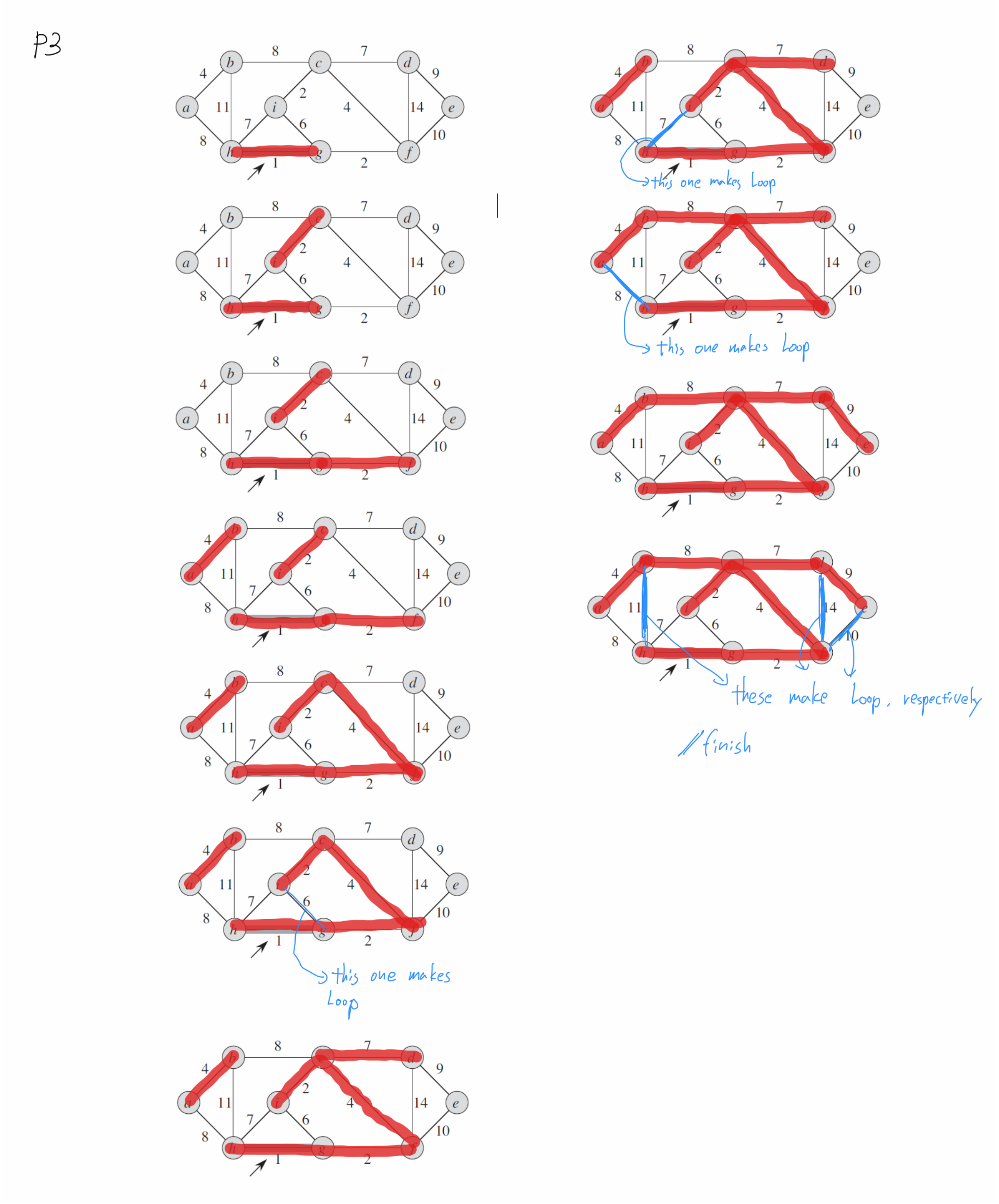
Result(Output)

```
r - pi : s, d : 2
s - pi : None, d : 1
t - pi : w, d : 7
u - pi : t, d : 8
v - pi : r, d : 3
w - pi : s, d : 6
```

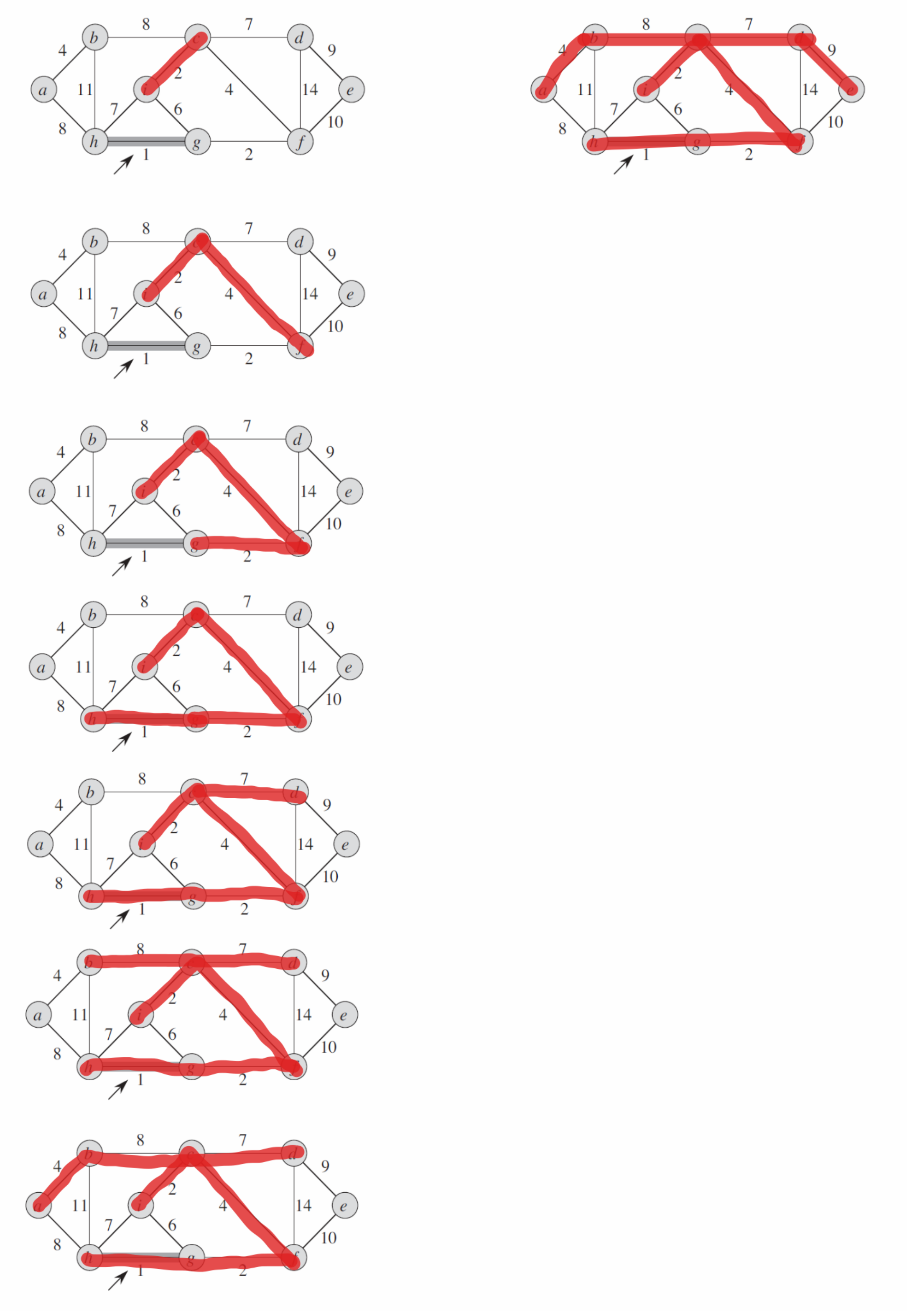
```
x - pi : u, d : 9
y - pi : x, d : 10
```

Problem #3

Kruscal Algorithm

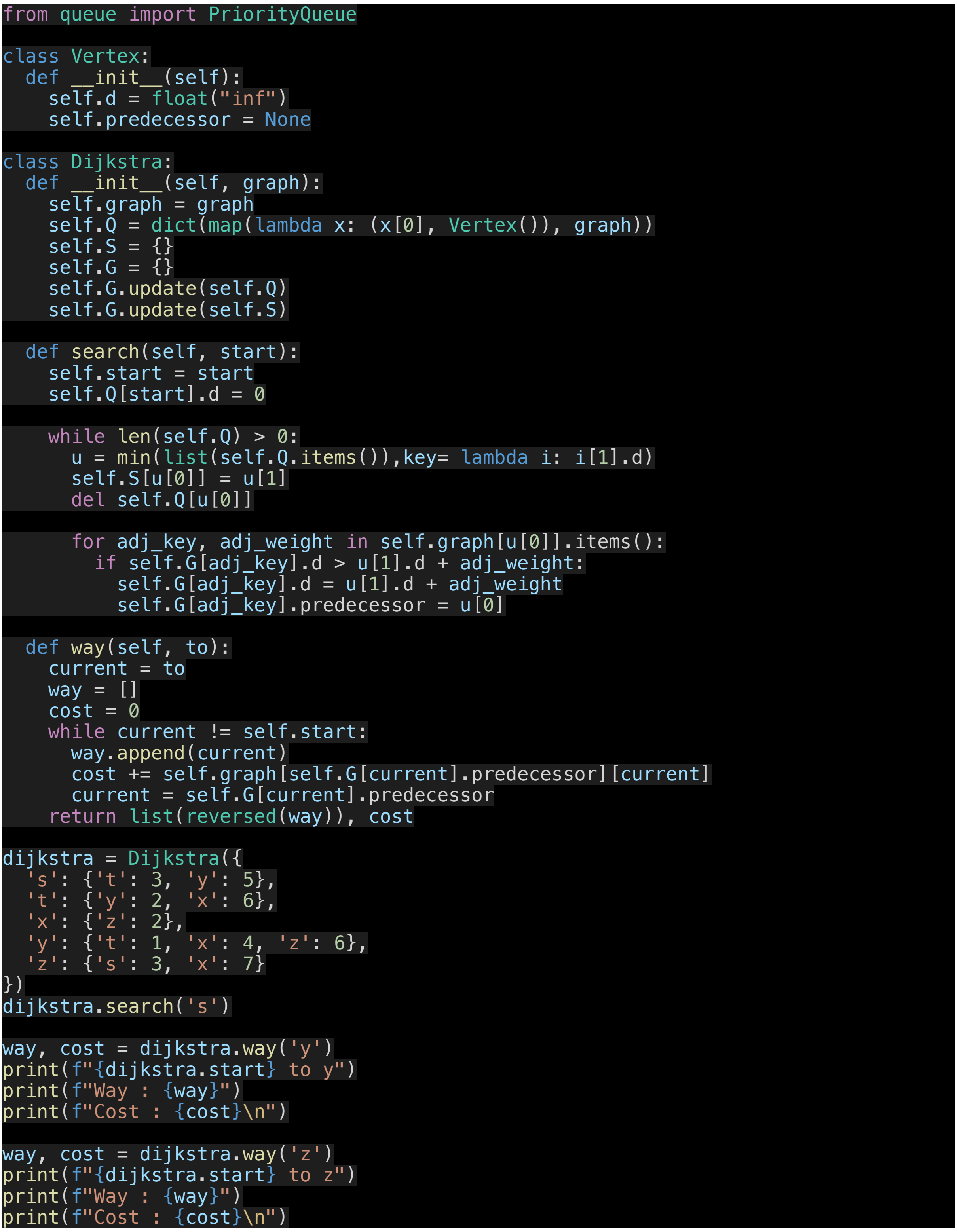


Prim's Algorithm



Problem #4

Code

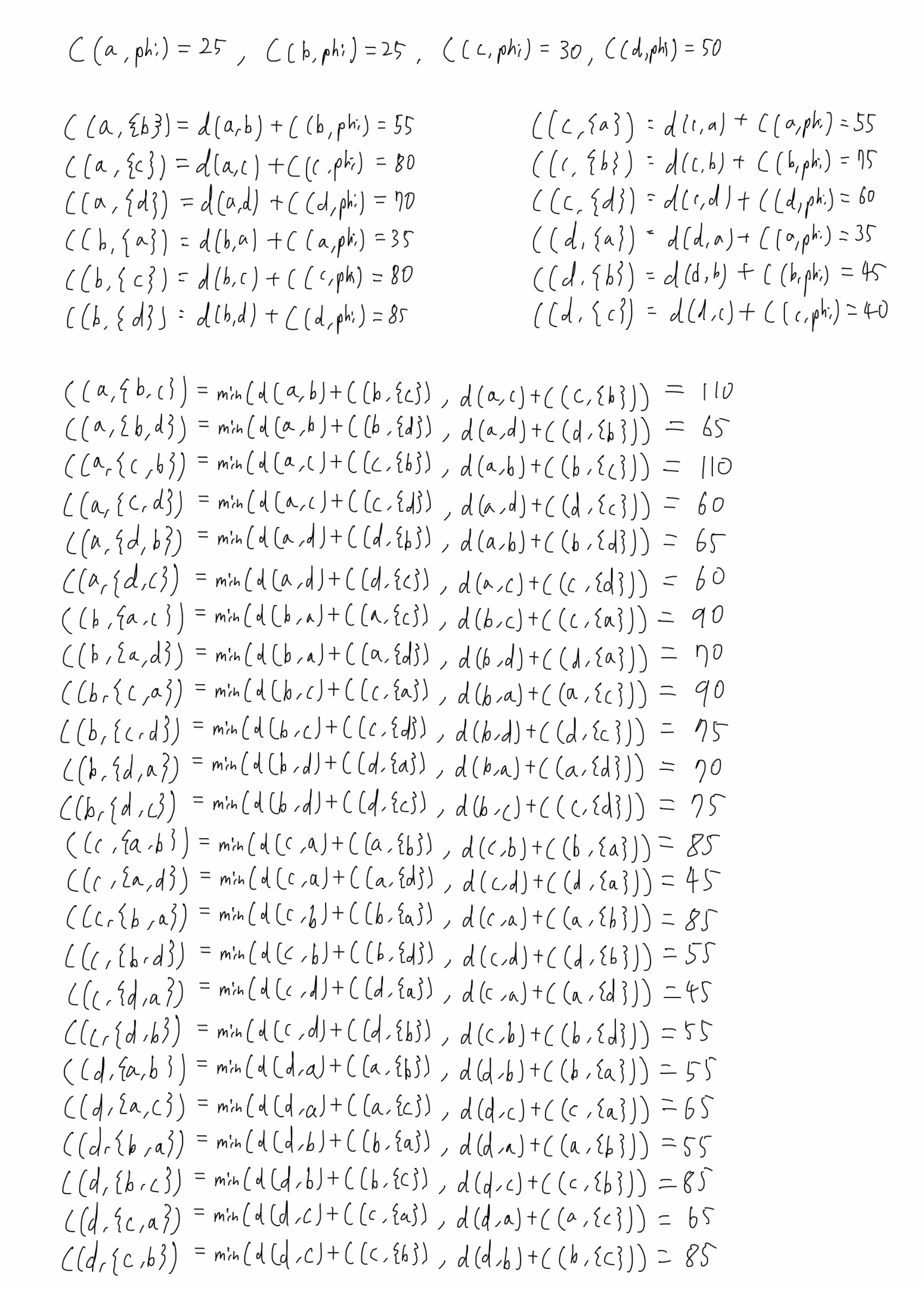


```
from queue import PriorityQueue

class Vertex:
  def __init__(self):
    self.d = float("inf")
    self.predecessor = None

class Dijkstra:
  def __init__(self, graph):
    self.graph = graph
    self.Q = dict(map(lambda x: (x[0], Vertex()), graph))
    self.S = {}
    self.G = {}
    self.G.update(self.Q)
    self.G.update(self.S)

  def search(self, start):
    self.start = start
    self.Q[start].d = 0

    while len(self.Q) > 0:
      u = min(list(self.Q.items()),key= lambda i: i[1].d)
      self.S[u[0]] = u[1]
      del self.Q[u[0]]

      for adj_key, adj_weight in self.graph[u[0]].items():
        if self.G[adj_key].d > u[1].d + adj_weight:
          self.G[adj_key].d = u[1].d + adj_weight
          self.G[adj_key].predecessor = u[0]

  def way(self, to):
    current = to
    way = []
    cost = 0
    while current != self.start:
      way.append(current)
      cost += self.graph[self.G[current].predecessor][current]
      current = self.G[current].predecessor
    return list(reversed(way)), cost

dijkstra = Dijkstra({
  's': {'t': 3, 'y': 5},
  't': {'y': 2, 'x': 6},
  'x': {'z': 2},
  'y': {'t': 1, 'x': 4, 'z': 6},
  'z': {'s': 3, 'x': 7}
})
dijkstra.search('s')

way, cost = dijkstra.way('y')
print(f"{dijkstra.start} to y")
print(f"Way : {way}")
print(f"Cost : {cost}\n")

way, cost = dijkstra.way('z')
print(f"{dijkstra.start} to z")
print(f"Way : {way}")
print(f"Cost : {cost}\n")
```

Output

```
s to y
Way : ['y']
Cost : 5

s to z
Way : ['y', 'z']
Cost : 11
```

Problem #5



C(a, phi) = 25 , C(b, phi) = 25 , C(c, phi) = 30 , C(d, phi) = 50

C(a, {b}) = d(a,b) + C(b, phi) = 55

C(a, {c}) = d(a,c) + C(c, phi) = 80

C(a, {d}) = d(a,d) + C(d, phi) = 70

C(b, {a}) = d(b,a) + C(a, phi) = 35

C(b, {c}) = d(b,c) + C(c, phi) = 80

C(b, {d}) = d(b,d) + C(d, phi) = 85

C(c, {a}) = d(c,a) + C(a, phi) = 55

C(c, {b}) = d(c,b) + C(b, phi) = 75

C(c, {d}) = d(c,d) + C(d, phi) = 60

C(d, {a}) = d(d,a) + C(a, phi) = 35

C(d, {b}) = d(d,b) + C(b, phi) = 45

C(d, {c}) = d(1,c) + C(c, phi) = 40

C(a, {b, c}) = min(d(a,b) + C(b, {c}) , d(a,c) + C(c, {b})) = 110

C(a, {b, d}) = min(d(a,b) + C(b, {d}) , d(a,d) + C(d, {b})) = 65

C(a, {c, b}) = min(d(a,c) + C(c, {b}) , d(a,b) + C(b, {c})) = 110

C(a, {c, d}) = min(d(a,c) + C(c, {d}) , d(a,d) + C(d, {c})) = 60

C(a, {d, b}) = min(d(a,d) + C(d, {b}) , d(a,b) + C(b, {d})) = 65

C(a, {d, c}) = min(d(a,d) + C(d, {c}) , d(a,c) + C(c, {d})) = 60

C(b, {a, c}) = min(d(b,a) + C(a, {c}) , d(b,c) + C(c, {a})) = 90

C(b, {a, d}) = min(d(b,a) + C(a, {d}) , d(b,d) + C(d, {a})) = 70

C(b, {c, a}) = min(d(b,c) + C(c, {a}) , d(b,a) + C(a, {c})) = 90

C(b, {c, d}) = min(d(b,c) + C(c, {d}) , d(b,d) + C(d, {c})) = 75

C(b, {d, a}) = min(d(b,d) + C(d, {a}) , d(b,a) + C(a, {d})) = 70

C(b, {d, c}) = min(d(b,d) + C(d, {c}) , d(b,c) + C(c, {d})) = 75

C(c, {a, b}) = min(d(c,a) + C(a, {b}) , d(c,b) + C(b, {a})) = 85

C(c, {a, d}) = min(d(c,a) + C(a, {d}) , d(c,d) + C(d, {a})) = 45

C(c, {b, a}) = min(d(c,b) + C(b, {a}) , d(c,a) + C(a, {b})) = 85

C(c, {b, d}) = min(d(c,b) + C(b, {d}) , d(c,d) + C(d, {b})) = 55

C(c, {d, a}) = min(d(c,d) + C(d, {a}) , d(c,a) + C(a, {d})) = 45

C(c, {d, b}) = min(d(c,d) + C(d, {b}) , d(c,b) + C(b, {d})) = 55

C(d, {a, b}) = min(d(d,a) + C(a, {b}) , d(d,b) + C(b, {a})) = 55

C(d, {a, c}) = min(d(d,a) + C(a, {c}) , d(d,c) + C(c, {a})) = 65

C(d, {b, a}) = min(d(d,b) + C(b, {a}) , d(d,a) + C(a, {b})) = 55

C(d, {b, c}) = min(d(d,b) + C(b, {c}) , d(d,c) + C(c, {b})) = 85

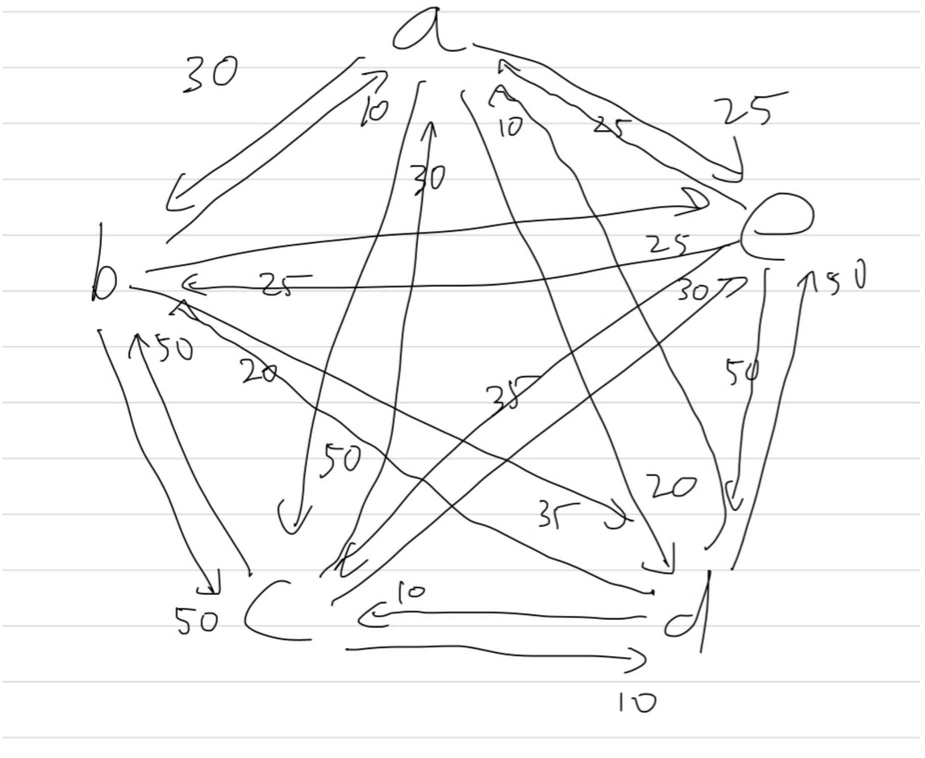
C(d, {c, a}) = min(d(d,c) + C(c, {a}) , d(d,a) + C(a, {c})) = 65

C(d, {c, b}) = min(d(d,c) + C(c, {b}) , d(d,b) + C(b, {c})) = 85

$$C(a,\{b,c,d\}) = min(d(a,b)+C(b,\{c,d\}), d(a,c)+C(c,\{b,d\}), d(a,d)+C(d,\{b,c\})$$

$$C(a,\{b,c,d\}) = min(d(a,b)+C(b,\{c,d\}), d(a,c)+C(c,\{b,d\}), d(a,d)+C(d,\{b,c\}) = 105$$
$$C(b,\{a,c,d\}) = min(d(b,a)+C(a,\{c,d\}), d(b,c)+C(c,\{a,d\}), d(b,d)+C(d,\{a,c\}) = 70$$
$$C(c,\{a,b,d\}) = min(d(c,a)+C(a,\{b,d\}), d(c,b)+C(b,\{a,d\}), d(c,d)+C(d,\{a,b\}) = 65$$
$$C(d,\{a,b,c\}) = min(d(d,a)+C(a,\{b,c\}), d(d,b)+C(b,\{a,c\})\ d(d,c)+C(C,\{a,b\}) = 95$$

$$\begin{aligned}C(e,\{a,b,c,d\}) = min(\ & d(e,a)+C(a,\{b,c,d\}),\\ & d(e,b)+C(b,\{a,c,d\}),\\ & d(e,c)+C(c,\{a,b,d\}),\\ & d(e,d)+C(d,\{a,b,c\}))\\ = min(\ & 25+105, 25+70, 35+65, 50+95) = 95\end{aligned}$$



Problem #6

Problem #6

```
import os
import time
from functools import reduce
stack = []

class Puzzle:
  def __init__(self, puzzle, from_root):
    self.puzzle = puzzle
    self.from_root = from_root

  def empty_cell_index(self):
    for i in range(len(self.puzzle)):
      for j in range(len(self.puzzle[i])):
        if self.puzzle[i][j] is None:
          return i, j

  def duplicated(self):
    result = []
    for i in self.puzzle:
      result_row = []
      for j in i:
        result_row.append(j)
      result.append(result_row)
    return Puzzle(result, self.from_root)

  def all_movables(self):
    result = []
    i = self.empty_cell_index()
    if i[0] != 0:
      dup = self.duplicated()
      dup.from_root += 1
      dup.puzzle[i[0]][i[1]] = dup.puzzle[i[0]-1][i[1]]
      dup.puzzle[i[0]-1][i[1]] = None
      result.append(dup)

    if i[1] != 0:
      dup = self.duplicated()
      dup.from_root += 1
      dup.puzzle[i[0]][i[1]] = dup.puzzle[i[0]][i[1]-1]
      dup.puzzle[i[0]][i[1]-1] = None
      result.append(dup)

    if i[0] != len(self.puzzle)-1:
      dup = self.duplicated()
      dup.from_root += 1
      dup.puzzle[i[0]][i[1]] = dup.puzzle[i[0]+1][i[1]]
      dup.puzzle[i[0]+1][i[1]] = None
      result.append(dup)

    if i[1] != len(self.puzzle[0])-1:
      dup = self.duplicated()
      dup.from_root += 1
      dup.puzzle[i[0]][i[1]] = dup.puzzle[i[0]][i[1]+1]
      dup.puzzle[i[0]][i[1]+1] = None
      result.append(dup)

    return result

  def cost(self, goal):
    misplaced = 0
    for i in range(len(self.puzzle)):
      for j in range(len(self.puzzle[i])):
        if self.puzzle[i][j] is None:
          continue
        if self.puzzle[i][j] != goal[i][j]:
```

```
     misplaced += 1
  return self.from_root + misplaced

 def is_same(self, puzzle):
  for i in range(len(self.puzzle)):
   for j in range(len(self.puzzle[i])):
    if self.puzzle[i][j] != puzzle[i][j]:
     return False
  return True




goal = [
 [1,2,3,4],
 [5,6,7,8],
 [9,10,11,12],
 [13,14,15,None],
]

class Tree:
 def __init__(self, data):
  self.data = data
  self.parent = None
  self.children = []

 def is_goal_achieved(self):
  achieved_result = list(map(lambda x: x.is_goal_achieved(), self.children))
  achieved_result.append(self.data.is_same(goal))
  return reduce(lambda x, y: x or y, achieved_result)

 def lowest_cost_node(self):
  result_candidates = list(map(lambda x: x.lowest_cost_node(), self.children))
  if len(result_candidates) == 0:
   return self
  else:
   return reduce(lambda x, y: x if x.data.cost(goal)<y.data.cost(goal) else y, 
result_candidates)

 def root(self):
  current = self
  while current.parent != None:
   current = current.parent
  return current

 def tree_span(self):
  children = list(map(lambda x: Tree(x), self.data.all_movables()))
  # print(children)
  children = list(filter(lambda x: not self.root().does_exist(x.data.puzzle), 
children))
  # print(children)
  for i in self.children:
   i.parent = self
  self.children = children

 def does_exist(self, puzzle):
  result = list(map(lambda x: x.does_exist(puzzle), self.children))
  result.append(self.data.is_same(puzzle))
  return reduce(lambda x, y: x or y, result)

 def search(self):
  while not self.is_goal_achieved():
   current_node = self.lowest_cost_node()
   # print(current_node.data.puzzle, current_node.data.cost(goal), 
current_node.data.from_root)
   # os.system("clear")
   print("----------")
   for i in current_node.data.puzzle:
    print(i)
```

```
      current_node.tree_span()
      # print(self.children)

root = Tree(Puzzle([
  [10, 7, 3, 4],
  [5 ,9 ,None ,11],
  [6 ,1 ,2 ,8],
  [13, 14, 15, 12]
], 0))

print(root.search())
```

Output(result)

```
----------
[10, 7, 3, 4]
[5, 9, None, 11]
[6, 1, 2, 8]
[13, 14, 15, 12]
----------
[10, 7, 3, 4]
[5, 9, 11, None]
[6, 1, 2, 8]
[13, 14, 15, 12]
...(Infinite Loop)...
```

Explanation

- It seems that my code doesn't work with checking already implemented puzzle. That means, Infinite Loop can be occurred. I tried hard to fix the problem, but the time for this assignment was too little.
- If the checking system works well, infinite loop won't be occurred and could work well.