

Homework 5

Homework05 > testHW-05-2.py > ...

```
1 class Student_information:
2     def __init__(self):
3         self.students = []
4
5     def add_student(self, name, score):
6         self.students.append((name, score))
7
8     def display_students(self):
9         print("\n--Unsorted Scores--")
10        for name, score in self.students:
11            print(f"{name}: {score}")
12
13    def sort_students(self):
14        for i in range(len(self.students)):
15            for j in range(len(self.students) - i - 1):
16                if self.students[j][1] < self.students[j + 1][1]:
17                    self.students[j], self.students[j + 1] = self.students[j + 1], self.students[j]
18
19    def display_sorted_students(self):
20        print("\n--Sorted Scores (Bubble Sort)--")
21        for name, score in self.students:
22            print(f"{name}: {score}")
23
24    def display_top_scores(self, top_n=3):
25        print(f"\n--Top {top_n} Highest Scores--")
26        for name, score in self.students[:top_n]:
27            print(f"{name}: {score}")
28
29    def display_bottom_scores(self):
30        print(f"\n--Top 3 Lowest Scores--")
31        for name, score in self.students[-1:-4:-1]:
32            print(f"{name}: {score}")
```

```
34     def search_by_score(self, score_to_search):
35         found_students = [name for name, score in self.students if score == score_to_search]
36         if found_students:
37             print(f"Found student(s) with score {score_to_search}")
38         else:
39             print(f"No students found with score {score_to_search}")
40
41
42 if __name__ == "__main__":
43     Student = Student_information()
44     Nstd = int(input("Enter the number of students: "))
45
46     for _ in range(Nstd):
47         name = input("Enter student name: ")
48         score = float(input(f"Enter student score for {name}: "))
49         Student.add_student(name, score)
50
51     Student.display_students()
52     Student.sort_students()
53     Student.display_sorted_students()
54     Student.display_top_scores()
55     Student.display_bottom_scores()
56
57     while True:
58         search_score = input("\nEnter the score to search (or type 'exit' to quit): ")
59         if search_score.lower() == 'exit':
60             break
61         Student.search_by_score(float(search_score))
```

Homework05 > HW-05_2.py > ...

```
1 #นายวรานนท์ ใจตรง
2 Nstd = int(input("Enter the number of students: "))
3 students = []
4 for _ in range(Nstd):
5     name = input("Enter student name: ")
6     score = float(input(f"Enter student score for {name}: "))
7     students.append((name, score))
8
9 print("\n--Unsorted Scores--")
10 for name, score in students:
11     print(f"{name}: {score}")
12
13 for i in range(len(students)):
14     for j in range(len(students) - i - 1):
15         if students[j][1] < students[j + 1][1]:
16             temp = students[j]
17             students[j] = students[j + 1]
18             students[j + 1] = temp
19
20 print("\n--Sorted Scores (Bubble Sort)--")
21 for name, score in students:
22     print(f"{name}: {score}")
```

```
24 print("\n--Top 3 Highest Scores--")
25 for name, score in students[:3]:
26     print(f"{name}: {score}")
27
28 print("\n--Top 3 Lowest Scores--")
29 for name, score in students[-1:-4:-1]:
30     print(f"{name}: {score}")
31
32 def search_by_score(score_to_search):
33     found_students = [name for name, score in students if score == score_to_search]
34     if found_students:
35         print(f"Found student with score {score_to_search}")
36     else:
37         print(f"No students found with score {score_to_search}")
38
39 while True:
40     search_score = input("\nEnter the score to search (or type 'exit' to quit): ")
41     if search_score.lower() == 'exit':
42         break
43     search_by_score(float(search_score))
```

Homework 6

```
Homework06 > HW-06.py > main
1  #นายวรานนท์ ใจตรง 6706022510433
2  #นายวัชรกร ชูศรียิ่ง 67060227510051
3  > Thailand = {...}
17
18  def insert_data():
19      print(f"~" + "\n" + ".join(Thailand) + "\n")
20      region = input("Enter region name: ")
21      province = input("Enter province name: ")
22      if region.lower() in Thailand:
23          Thailand[region.lower()].append(province)
24      else:
25          Thailand[region] = [province]
26      print("Data added successfully!\n")
27
28  def update_data():
29      region = input("Enter the region name to update: ")
30      if region.lower() in Thailand:
31          print(f"Provinces in {region.capitalize()}: \n - " + "\n - ".join(Thailand[region]) + "\n")
32          old_province = input("Enter the province name to update: ")
33          if old_province in Thailand[region]:
34              new_province = input("Enter the new province name: ")
35              index = Thailand[region].index(old_province)
36              Thailand[region][index] = new_province
37              print("Data updated successfully!\n")
38      else:
39          print("Province not found!\n")
40      print("Region not found!\n")
41
42  def search_data():
43      print("=== Search Data ===")
44      print("1. Search by Region")
45      print("2. Search by Province")
46      choice = input("Please select a menu (1-2): ")
47      if choice == "1":
48          region = input("Enter the region name: ")
49          if region.lower() in Thailand:
50              print(f"Provinces in {region}: {Thailand[region]}\n")
51          else:
52              print("Region data not found!\n")
53      elif choice == "2":
54          province = input("Enter province name: ")
55          for region in Thailand:
56              if province in Thailand[region]:
57                  print(f'Province {province} is in {region} of Thailand\n')
```

```
60  def delete_data():
61      region = input("Enter the region name to delete data: ")
62      if region.lower() in Thailand:
63          print(f"Provinces in {region}: {Thailand[region]}")
64          province = input("Enter the province name to delete: ")
65          if province in Thailand[region]:
66              Thailand[region].remove(province)
67              if not Thailand[region]:
68                  del Thailand[region]
69              print("Data deleted successfully!\n")
70          else:
71              print("Province not found!\n")
72      else:
73          print("Region not found!\n")
74
75  def view_all_data():
76      if Thailand:
77          for region, provinces in Thailand.items():
78              print(f"{region.capitalize()}: \n - " + "\n - ".join(provinces) + "\n")
79              print("-----\n")
80      else:
81          print("No data available!\n")
82
83  def main():
84      while True:
85          print("=== Province Data Management Menu ===")
86          print("1. Insert Data")
87          print("2. Update Data")
88          print("3. Search Data")
89          print("4. Delete Data")
90          print("5. View All Data")
91          print("6. Exit")
92          print()
93
94          choice = input("Please select a menu (1-6): ")
95          print()
96
97          if choice == "1":
98              insert_data()
99          elif choice == "2":
100              update_data()
101          elif choice == "3":
102              search_data()
103          elif choice == "4":
104              delete_data()
105          elif choice == "5":
106              view_all_data()
107          elif choice == "6":
108              print("Exiting the program...")
109              break
110          else:
111              print("Please select a valid menu option (1-6)\n")
112
113  if __name__ == "__main__":
114      main()
```

```
Hash- > test.py > ...
1  data = [25,27,14,48,37,30,26,41,31,43,54,40]
2  for i in data:
3      print(i%len(data),end=" ")
4  print()
5
6  result = []
7  for i in range(len(data)):
8      res = []
9      for j in data:
10
11          if j%len(data) == i:
12              res.append(j)
13      result.append(res)
14
15  print(result)
```

โค้ดช่วย Hash

ใช้ mod ข้อมูลทีละตัว

AVL Tree

```
Homework07 > avl_tree.py > ...
1 class Node:
2     def __init__(self, key):
3         self.key = key
4         self.left = None
5         self.right = None
6     def avl_height(node):
7         return -1 if node is None else 1 + max(avl_height(node.left), avl_height(node.right))
8     def rotate_left(r):
9         u = r.right
10        r.right = u.left
11        u.left = r
12        return u
13    def rotate_right(r):
14        o = r.left
15        r.left = o.right
16        o.right = r
17        return o
18    def avl_adjust(r):
19        balance = avl_height(r.right) - avl_height(r.left)
20        if balance <= -2:
21            b1 = avl_height(r.left.right) - avl_height(r.left.left)
22            if b1 <= 0:
23                return rotate_right(r)
24            else:
25                r.left = rotate_left(r.left)
26                return rotate_right(r)
27        if balance >= 2:
28            br = avl_height(r.right.right) - avl_height(r.right.left)
29            if br >= 0:
30                return rotate_left(r)
31            else:
32                r.right = rotate_right(r.right)
33                return rotate_left(r)
34        return r
35    def avl_insert(r, key):
36        if r is None:
37            return Node(key)
38        if key < r.key:
39            r.left = avl_insert(r.left, key)
40        elif key > r.key:
41            r.right = avl_insert(r.right, key)
42        return avl_adjust(r)
43    def avl_minimum(r):
44        return float('inf') if r is None else r.key if r.left is None else avl_minimum(r.left)
```

```
45    def avl_delete(r, key):
46        if r is None:
47            return None
48        if key < r.key:
49            r.left = avl_delete(r.left, key)
50        elif key > r.key:
51            r.right = avl_delete(r.right, key)
52        else:
53            if r.left is None:
54                return r.right
55            elif r.right is None:
56                return r.left
57            min_key = avl_minimum(r.right)
58            r.key = min_key
59            r.right = avl_delete(r.right, min_key)
60        return avl_adjust(r)
61    def print_tree(r):
62        if r is None:
63            print("( )", end="")
64            return
65        if r.left is None and r.right is None:
66            print(f"({r.key})", end="")
67            return
68        print("(", end="")
69        print_tree(r.left)
70        print(f"({r.key})", end="")
71        print_tree(r.right)
72        print(")", end="")
73    # ตัวอย่างการใช้งาน
74    if __name__ == "__main__":
75        root = None
76        for i in range(7):
77            root = avl_insert(root, i)
78            print_tree(root)
79        print()
80        root = avl_delete(root, 0)
81        print_tree(root)
82        print()
83        root = avl_delete(root, 1)
84        print_tree(root)
85        print()
86        root = avl_delete(root, 2)
87        print_tree(root)
88        print()
```



Trees

```
Homework07 > trees.py > ...
1 class Node:
2     def __init__(self, data):
3         self.left = None
4         self.right = None
5         self.data = data
6     def insert(self, data):
7         if self.data:
8             if data < self.data:
9                 if self.left is None:
10                    self.left = Node(data)
11                else:
12                    self.left.insert(data)
13            elif data > self.data:
14                if self.right is None:
15                    self.right = Node(data)
16                else:
17                    self.right.insert(data)
18            else:
19                self.data = data
20    def PrintTree_In(self):
21        if self.left:
22            self.left.PrintTree_In()
23        print(self.data)
24        if self.right:
25            self.right.PrintTree_In()
26    def PrintTree_Pre(self):
27        print(self.data)
28        if self.left:
29            self.left.PrintTree_Pre()
30        if self.right:
31            self.right.PrintTree_Pre()
32    def PrintTree_Post(self):
33        if self.left:
34            self.left.PrintTree_Post()
35        if self.right:
36            self.right.PrintTree_Post()
37        print(self.data)
38    def findval(self, lkpval):
39        if lkpval < self.data:
40            if self.left is None:
41                return str(lkpval) + " Not Found"
42            return self.left.findval(lkpval)
43        elif lkpval > self.data:
44            if self.right is None:
45                return str(lkpval) + " Not Found"
46            return self.right.findval(lkpval)
47        else:
48            print(str(self.data) + ' is found')
49    def find_min(self):
50        current = self
51        while current.left is not None:
52            current = current.left
53        return current.data
```

```
def delete(self, lkpval):
    if lkpval < self.data:
        if self.left:
            self.left = self.left.delete(lkpval)
        else:
            print("Value not found")
    elif lkpval > self.data:
        if self.right:
            self.right = self.right.delete(lkpval)
        else:
            print("Value not found")
    else:
        if self.left is None and self.right is None:
            return None
        if self.left is None:
            return self.right
        if self.right is None:
            return self.left
        min_val = self.right.find_min()
        self.data = min_val
        self.right = self.right.delete(min_val)
    return self
def inorderTraversal(self, root):
    res = []
    if root:
        res = self.inorderTraversal(root.left)
        res.append(root.data)
        res = res + self.inorderTraversal(root.right)
    return res
```



```
def PreorderTraversal(self, root):
    res = []
    if root:
        res.append(root.data)
        res = res + self.PreorderTraversal(root.left)
        res = res + self.PreorderTraversal(root.right)
    return res
def PostorderTraversal(self, root):
    res = []
    if root:
        res = self.PostorderTraversal(root.left)
        res = res + self.PostorderTraversal(root.right)
        res.append(root.data)
    return res
93 root = Node(10)
94 root.insert(30)
95 root.insert(40)
96 root.insert(35)
97 root.insert(20)
98 root.insert(47)
99 root.insert(5)
100 print("Print InOrder")
101 root.PrintTree_In()
102 print("Print PreOrder")
103 root.PrintTree_Pre()
104 print("Print PostOrder")
105 root.PrintTree_Post()
106 print()
107 print(root.findval(7))
108 print(root.findval(35))
109 print(root.inorderTraversal(root))
110 print(root.PreorderTraversal(root))
111 print(root.PostorderTraversal(root))
```

Search

Searching >  sequentialsearch.py >  sequentialSearch

```
1 def sequentialSearch(alist, item):
2     pos = 0
3     found = False
4
5     while pos < len(alist) and not found:
6         if alist[pos] == item:
7             found = True
8         else:
9             pos += 1
10
11     return found
12
```

Sequential Search

Searching >  binarysearch.py >  binarySearch



```
1 def binarySearch(alist, item):
2     first = 0
3     last = len(alist) - 1
4     found = False
5
6     while first <= last and not found:
7         midpoint = (first + last)//2
8         if alist[midpoint] == item:
9             found = True
10        else:
11            if item < alist[midpoint]:
12                last = midpoint - 1
13            else:
14                first = midpoint + 1
15
16    return found
```

Binary Search

Sorting

1. Bubble Sort

วิธีเปรียบเทียบข้อมูลใกล้เคียงกันทีละคู่ และสลับตำแหน่งถ้าจำเป็น ซ้ำไปเรื่อย ๆ จนกว่าข้อมูลทั้งหมดจะเรียงลำดับ

```
python   แก้ไข
```

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# ตัวอย่างการใช้งาน
data = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(data)
print("Bubble Sort:", data) # Output: [11, 12, 22, 25, 34, 64, 90]
```

3. Insertion Sort

แทรกแต่ละค่าจากส่วนที่ยังไม่ได้เรียงไปยังตำแหน่งที่เหมาะสมในส่วนที่เรียงแล้ว

```
python   แก้ไข
```

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# ตัวอย่างการใช้งาน
data = [12, 11, 13, 5, 6]
insertion_sort(data)
print("Insertion Sort:", data) # Output: [5, 6, 11, 12, 13]
```

2. Selection Sort

เลือกค่าที่เล็กที่สุดจากส่วนที่ยังไม่เรียงและสลับกับตำแหน่งแรกในส่วนที่ยังไม่เรียง

```
python   แก้ไข
```

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

# ตัวอย่างการใช้งาน
data = [64, 25, 12, 22, 11]
selection_sort(data)
print("Selection Sort:", data) # Output: [11, 12, 22, 25, 64]
```

5. Quick Sort

เลือกค่าหนึ่งเป็น pivot แล้วแบ่งข้อมูลตามค่าที่เล็กกว่าหรือใหญ่กว่า pivot

```
python   แก้ไข
```

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# ตัวอย่างการใช้งาน
data = [10, 7, 8, 9, 1, 5]
sorted_data = quick_sort(data)
print("Quick Sort:", sorted_data) # Output: [1, 5, 7, 8, 9, 10]
```

4. Merge Sort

ใช้การแบ่งและรวม (Divide and Conquer)

```
python 🔗 คัดลอก 🗑 แก้ไข

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]

        merge_sort(left)
        merge_sort(right)

        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1

        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1

        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1

# ตัวอย่างการใช้งาน
data = [12, 11, 13, 5, 6, 7]
merge_sort(data)
print("Merge Sort:", data) # Output: [5, 6, 7, 11, 12, 13]
```

Graph

```
test.py > ...
1 import matplotlib.pyplot as plt
2 x = ['Mon', 'The', 'Wed', 'Thu', 'Fri']
3 y = [10,20,30,40,50]
4
5 plt.bar(x,y,color = 'y',alpha = 0.8)
6 plt.show()
```

```
graph > graph.py > ...
1 from matplotlib.font_manager import weight_dict
2 import networkx as nx
3 import matplotlib.pyplot as plt
4 network = nx.Graph()
5 network.add_nodes_from([1,2,3,4,5,6,7])
6 color_list = ['gold','red','violet','pink','brown','yellow','gray']
7 plt.figure(figsize=(6,6))
8 plt.title("Example of Graoh Representation",size=10)
9 network.add_edge(6,7,weight= 2)
10 network.add_edge(6,5)
11 network.add_edge(5,3)
12 network.add_edge(7,3)
13 network.add_edge(3,1)
14 network.add_edge(5,1,weight= 4)
15 network.add_edge(1,4)
16 network.add_edge(1,2)
17 network.add_edge(4,2)
18
19 print(f"This network has now {network.number_of_nodes()} nodes.")
20
21 nx.draw_networkx(network,node_color = color_list, with_labels=True)
22 plt.show()
```

6. Heap Sort

ใช้โครงสร้างข้อมูล heap

```
python 🔗 คัดลอก 🗑 แก้ไข

import heapq

def heap_sort(arr):
    heap = []
    for value in arr:
        heapq.heappush(heap, value)
    sorted_arr = [heapq.heappop(heap) for _ in range(len(heap))]
    return sorted_arr

# ตัวอย่างการใช้งาน
data = [4, 10, 3, 5, 1]
sorted_data = heap_sort(data)
print("Heap Sort:", sorted_data) # Output: [1, 3, 4, 5, 10]
```

```
Homework05 > radixSort.py > ...
1 def counting_sort(arr, div):
2     n = len(arr)
3     output = [0] * n
4     count = [0] * 10
5
6     for i in arr:
7         index = (i // div) % 10
8         count[index] += 1
9
10    for i in range(1, 10):
11        count[i] += count[i - 1]
12
13    for i in reversed(range(n)):
14        index = (arr[i] // div) % 10
15        output[count[index] - 1] = arr[i]
16        count[index] -= 1
17
18    for i in range(n):
19        arr[i] = output[i]
20    print(output)
21
22 def radix_sort(arr):
23     max_num = max(arr)
24     div = 1
25     while max_num // div > 0:
26         counting_sort(arr, div)
27         div *= 10
28
29 data = [171, 45, 75, 91, 802, 24, 2, 66]
30 radix_sort(data)
31 print("Sorted Array:", data)
```


Dictionary

ตัวอย่างการใช้ Dict

Dictionary > create.py > [0] my_dict

```
1 my_dict = {'Dave': '001', 'Ava': '002', 'Joe': '003'}
2 print(my_dict)
3 print(type(my_dict))
4
5 my_dict = dict()
6 print(my_dict)
7 print(type(my_dict))
8
9 my_dict = dict(name = "John", age = 36, country = "Norway")
10 print(my_dict)
11
12 my_dict = dict(name = "John", age = 36, country = "Norway")
13 print(my_dict)
14 x = my_dict["country"]
15 print(x)
16
17 my_dict = dict(name = "John", age = 36, country = "Norway")
18 print(my_dict)
19 x = my_dict.get("country")
20 print(x)
21
22 my_dict = dict(name = "John", age = 36, country = "Norway")
23 print(my_dict)
24 x = my_dict.keys()
25 print(x)
26
27 my_dict = dict(name = "John", age = 36, country = "Norway")
28 print(my_dict)
29 x = my_dict.keys()
30 print(x) #before the change
31 my_dict["color-like"] = "white"
32 print(my_dict)
33 print(x) #after the change
34
35 my_dict = dict(name = "John", age = 36, country = "Norway")
36 print(my_dict)
37 if "country" in my_dict:
38     print("Yes, 'country' is one of the keys in the my_dict dictionary")
39
40 my_dict = dict(name = "John", age = 36, country = "Norway")
41 print(my_dict)
42 my_dict["name"] = "Ford"
43 my_dict["color-like"] = "Blue"
44 print(my_dict)
```

```
46 my_dict = dict(name = "John", age = 36, country = "Norway")
47 print(my_dict)
48 my_dict.update({"country": "Thailand"})
49 my_dict.update({"color-like": "Green"})
50 print(my_dict)
51
52 my_dict = dict(name = "John", age = 36, country = "Norway")
53 print(my_dict)
54 my_dict.pop("age")
55 print(my_dict)
56
57 my_dict = dict(name = "John", age = 36, country = "Norway")
58 print(my_dict)
59 my_dict.popitem()
60 print(my_dict)
61
62 my_dict = dict(name = "John", age = 36, country = "Norway")
63 print(my_dict)
64 my_dict.clear()
65 print(my_dict)
66
67 my_dict = dict(name = "John", age = 36, country = "Norway")
68 for x in my_dict:
69     print(x)
70
71 my_dict = dict(name = "John", age = 36, country = "Norway")
72 for x in my_dict:
73     print(my_dict[x])
74 my_dict = dict(name = "John", age = 36, country = "Norway")
75 for x in my_dict.values():
76     print(x)
77 for x in my_dict.keys():
78     print(x)
79 for x, y in my_dict.items():
80     print(x, y)
81
82 my_dict = dict(name = "John", age = 36, country = "Norway")
83 print(my_dict)
84 new_dict1 = my_dict.copy()
85 print(new_dict1)
86 new_dict2 = dict(my_dict)
87 print(new_dict2)
```

Dictionary > nd.py > ...

```
1 # Level 1 Dictionary
2 university_data = {
3     'name': 'XYZ University',
4     'location': 'Technology City',
5     'faculties': {
6         # Level 2 Dictionary
7         'information_technology': {
8             'departments': {
9                 # Level 3 Dictionary
10                'software_engineering': {'courses': ['Software Development', 'Database Systems']},
11                'networking': {'courses': ['Computer Networks', 'Network Security']}
12            }
13        },
14        'business': {
15            'departments': {
16                'management_systems': {'courses': ['Business Intelligence', 'IT Management']},
17                'finance_technology': {'courses': ['Fintech', 'Blockchain']}
18            }
19        }
20    }
21 }
22 # Displaying data from Nested Dictionary
23 print("University Name:", university_data['name'])
24 print("Location:", university_data['location'])
25 # Displaying data from Level 2 Dictionary
26 print("\nFaculty of Information Technology:")
27 print("Departments:", university_data['faculties']['information_technology'])
28 # Displaying data from Level 3 Dictionary
29 print("\nInformation Technology Branch - Software Engineering:")
30 print("Courses:", university_data['faculties']['information_technology']['departments']['software_engineering']['courses'])
```

Assignment Graph

```
4 import networkx as nx
5 import matplotlib.pyplot as plt
6
7 filename = "Data_Graph"
8
9 def create_graph():
10     G = nx.Graph()
11     locations = {}
12     edges = []
13     G, locations, edges = load_graph(filename, G, locations, edges)
14     return G, locations
15
16 def draw_graph(G, locations, shortest_path=None):
17     pos = nx.get_node_attributes(G, 'pos')
18
19     plt.figure(figsize=(12, 8))
20     nx.draw(G, pos, with_labels=True, node_size=2000, node_color="lightblue", edge_color="gray", font_size=8)
21
22     edge_labels = {(u, v): d['weight'] for u, v, d in G.edges(data=True)}
23     nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8)
24
25     if shortest_path:
26         path_edges = list(zip(shortest_path, shortest_path[1:]))
27         nx.draw(G, pos, edgelist=path_edges, edge_color="red", width=3)
28
29     plt.show()
30
31 def find_shortest_path(G, start, end):
32     try:
33         path = nx.shortest_path(G, source=start, target=end, weight='weight')
34         distance = nx.shortest_path_length(G, source=start, target=end, weight='weight')
35         return path, distance
36     except nx.NetworkXNoPath:
37         return None
38
39 def insert_node(G, locations, name, pos, locate_add_edge, distance):
40     if name in locations:
41         print("Node already exists.")
42         return
43     locations[name] = pos
44     G.add_node(name, pos=pos)
45     G.add_edge(name, locate_add_edge, weight=distance)
46     save_graph(filename, G, locations)
47     print(f"Location {name} inserted successfully.")
48
49 def delete_node(G, locations, name):
50     if name not in locations:
51         print("Node does not exist.")
52         return
53     del locations[name]
54     G.remove_node(name)
55     save_graph(filename, G, locations)
56     print(f"Location {name} deleted successfully.")
57
58 def save_graph(filename, G, locations):
59     data = {
60         "locations": locations,
61         "edges": [(u, v, d['weight']) for u, v, d in G.edges(data=True)]
62     }
63     with open(filename, 'w') as file:
64         file.write(str(data))
65     print(f"Graph saved to {filename}")
66
67 def load_graph(filename, G, locations, edges):
68     with open(filename, 'r') as file:
69         data = eval(file.read())
70
71     G = nx.Graph()
72     locations = data["locations"]
73     edges = data["edges"]
74     for location, pos in locations.items():
75         G.add_node(location, pos=pos)
76     for u, v, weight in edges:
77         G.add_edge(u, v, weight=weight)
78     print(f"Graph loaded from {filename}")
79     return G, locations, edges
```

Data เก็บข้อมูลใน Assign

Assignment > Data_Graph

```

1 {'Locations': {'Modern One Dorm': (-3, 5), 'Thipai Dorm': (-4, 4.5), 'Baan Kasem Dorm': (-5, 4),
2 'Baan Puen Apartment': (-2.7, 3.7), 'The Brick Place': (1.4, 2.9),
3 'Khao Yai Modern Place': (3.9, 2.8), 'Gray Dorm': (2.8, 3.4),
4 'White Lion Dorm': (3.6, 3.7), 'Wannaporn Dorm': (2, 4), 'Saengtawan Dorm': (0.6, 4),
5 'Wanasaya Grand': (-3.5, 1.4), 'Baan Nicha Prachinburi': (-2.5, 1.1),
6 'Mee Suk House': (-2.5, 1.7), 'Baan Thanomkhan': (-1.5, 1.3),
7 'Mangkonthong Mansion': (-0.8, 2), 'Waramon Grand Place': (0.8, 1.5),
8 'Chanchao Mansion': (3.5, 0.5), 'Saowalak Dorm': (-1.6, 0.5),
9 'Buakhao Dorm': (-0.1, 0.5), 'Chamnonngjit Dorm': (2, 0.5),
10 'KMUTNB Male Dorm': (-2.9, 2.1), 'KMUTNB Female Dorm': (-3.5, 2.6), 'University': (-1.6, 3)},
11 'edges': [(('Modern One Dorm', 'University', 1.4), ('Modern One Dorm', 'Thipai Dorm', 0.45),
12 ('Thipai Dorm', 'Baan Puen Apartment', 0.3), ('Thipai Dorm', 'Baan Kasem Dorm', 0.45),
13 ('Baan Kasem Dorm', 'Baan Puen Apartment', 0.4), ('Baan Puen Apartment', 'University', 1),
14 ('The Brick Place', 'University', 0.7), ('The Brick Place', 'Khao Yai Modern Place', 0.25),
15 ('Khao Yai Modern Place', 'White Lion Dorm', 0.12), ('Khao Yai Modern Place', 'Gray Dorm', 0.12),
16 ('Gray Dorm', 'Wannaporn Dorm', 0.13), ('Gray Dorm', 'White Lion Dorm', 0.013),
17 ('White Lion Dorm', 'Wannaporn Dorm', 0.13), ('Wannaporn Dorm', 'Saengtawan Dorm', 0.05),
18 ('Saengtawan Dorm', 'University', 0.4), ('Wanasaya Grand', 'Mee Suk House', 0.2),
19 ('Wanasaya Grand', 'Baan Nicha Prachinburi', 0.2), ('Baan Nicha Prachinburi', 'Baan Thanomkhan', 0.11),
20 ('Baan Nicha Prachinburi', 'Mee Suk House', 0), ('Mee Suk House', 'Baan Thanomkhan', 0.11),
21 ('Baan Thanomkhan', 'Mangkonthong Mansion', 0.18), ('Mangkonthong Mansion', 'University', 0.95),
22 ('Mangkonthong Mansion', 'Waramon Grand Place', 0.65), ('Waramon Grand Place', 'Chamnonngjit Dorm', 1.1),
23 ('Waramon Grand Place', 'Buakhao Dorm', 1.1), ('Chanchao Mansion', 'Chamnonngjit Dorm', 0.26),
24 ('Saowalak Dorm', 'Buakhao Dorm', 0.13), ('Buakhao Dorm', 'Chamnonngjit Dorm', 0.45),
25 ('KMUTNB Male Dorm', 'University', 0), ('KMUTNB Female Dorm', 'University', 0)]}
```

```

81 if __name__ == "__main__":
82     G, locations = create_graph()
83
84     print("Shortest route search system\n1.Show Graph\n2.Find Shortest Path",
85           "\n3.Insert Location\n4.Delete Location\n5.Save Graph\n0.Exit")
86     choice = int(input("Enter your choice: "))
87     if choice == 1:
88         draw_graph(G, locations)
89
90     elif choice == 2:
91         print("Available Locations:")
92         for idx, loc in enumerate(locations.keys(), 1):
93             print(f"{idx}. {loc}")
94
95         location_start = int(input("Select location start: "))
96         location_list = list(locations.keys())
97         selected_location_start = location_list[location_start - 1]
98
99         location_end = int(input("Select location end: "))
100        selected_location_end = location_list[location_end - 1]
101
102        if selected_location_start and selected_location_end in locations:
103            path, distance = find_shortest_path(G, selected_location_start, selected_location_end)
104            if path:
105                print(f"Shortest path to {selected_location_end}:")
106                print(" -> ".join(path))
107                print(f"Total distance: {format(distance, '.2f')} km")
108                draw_graph(G, locations, path)
109            else:
110                print("No path found.")
111        else:
112            print("Invalid location selection.")
113
114    elif choice == 3:
115        name = input("Enter location name: ")
116        pos = tuple(map(float, input("Enter location position (x, y): ").split(", ")))
117        print("Available Locations:")
118        for idx, loc in enumerate(locations.keys(), 1):
119            print(f"{idx}. {loc}")
120
121        location = int(input("Select location to insert distance : "))
122        location_list = list(locations.keys())
123        selected_location = location_list[location - 1]
124        distance = float(input("Enter distance to selected location: "))
125        insert_node(G, locations, name, pos, selected_location, distance)
126        draw_graph(G, locations)
127
128    elif choice == 4:
129        name = input("Enter location name to delete: ")
130        delete_node(G, locations, name)
131        draw_graph(G, locations)
132
133    elif choice == 5:
134        save_graph(filename, G, locations)
135
136    elif choice == 0:
137        exit()
```