Assessed Task 1: Selection Sort

      Selection sort works with two subarrays: the sorted section, and the unsorted section.

It works by taking the minimum value in the unsorted array, putting it at the end of the sorted array, and continuing until there is no unsorted array.

For example, starting with the unsorted array A:

A = [11, 22, 14, 67, 2, 9]

We find the minimum value, and move it to the end of the unsorted array – which, first time round, is just the first element of A:

A = [**2,** 11, 22, 14, 67, 9]

Now 2 is the sorted subarray, and the unsorted subarray is everything else.

Next we find the smallest value in the unsorted subarray, and move it to the end of the sorted subarray:

A = [2, 9, 11, 22, 14, 67]

Now 2 and 9 are the sorted subarray, and again we look for the smallest value in the unsorted subarray and move that to the end of the sorted subarray:

A = [**2, 9, 11,** 22, 14, 67]

… and so on, until we have a sorted array:

A = [**2, 9, 11, 14, 22, 67**]

      Here is the pseudocode for selection sort:

**SELECTION\_SORT(A)**

**FOR i TO length(A)-1**

**min ← i**

**FOR j ← i + 1 TO length(A)**

**IF A[j] < A[min]**

**min ← j**

**SWAP (A, i, min)**

**RETURN A**

Implement this pseudocode in Python, or C++, or (if you like), both, adding comments as necessary. Be aware that min may be a protected term so you may need a different variable name e.g. minn. Note that a swap needs to be performed, and this, in the pseudocode, is the subfunction SWAP. You need to work out how to implement the swap. This does not have to be done using a subfunction: the swap could be implemented directly – the choice is yours.

**Assessed Standard Task 3/5**: Implement Graph as Adjacency Matrix  

where the vertices consist of positive integers (note that 0 is not a positive integer). The program should have methods for the following:

Adding a vertex to the graph.

Adding an edge to the graph.

This should check whether the edge already exists

Removing an edge from the graph.

Checking that there is an edge to remove

Printing the graph as a matrix.

Which should look something like this:

**1   2   3   4   5   6**

**---------------------**

**1 |   0   1   1   0   0   0**

**2 |   1   0   1   0   0   0**

**3 |   1   1   0   1   0   0**

**4 |   0   0   1   0   0   0**

**5 |   0   0   0   0   0   0**

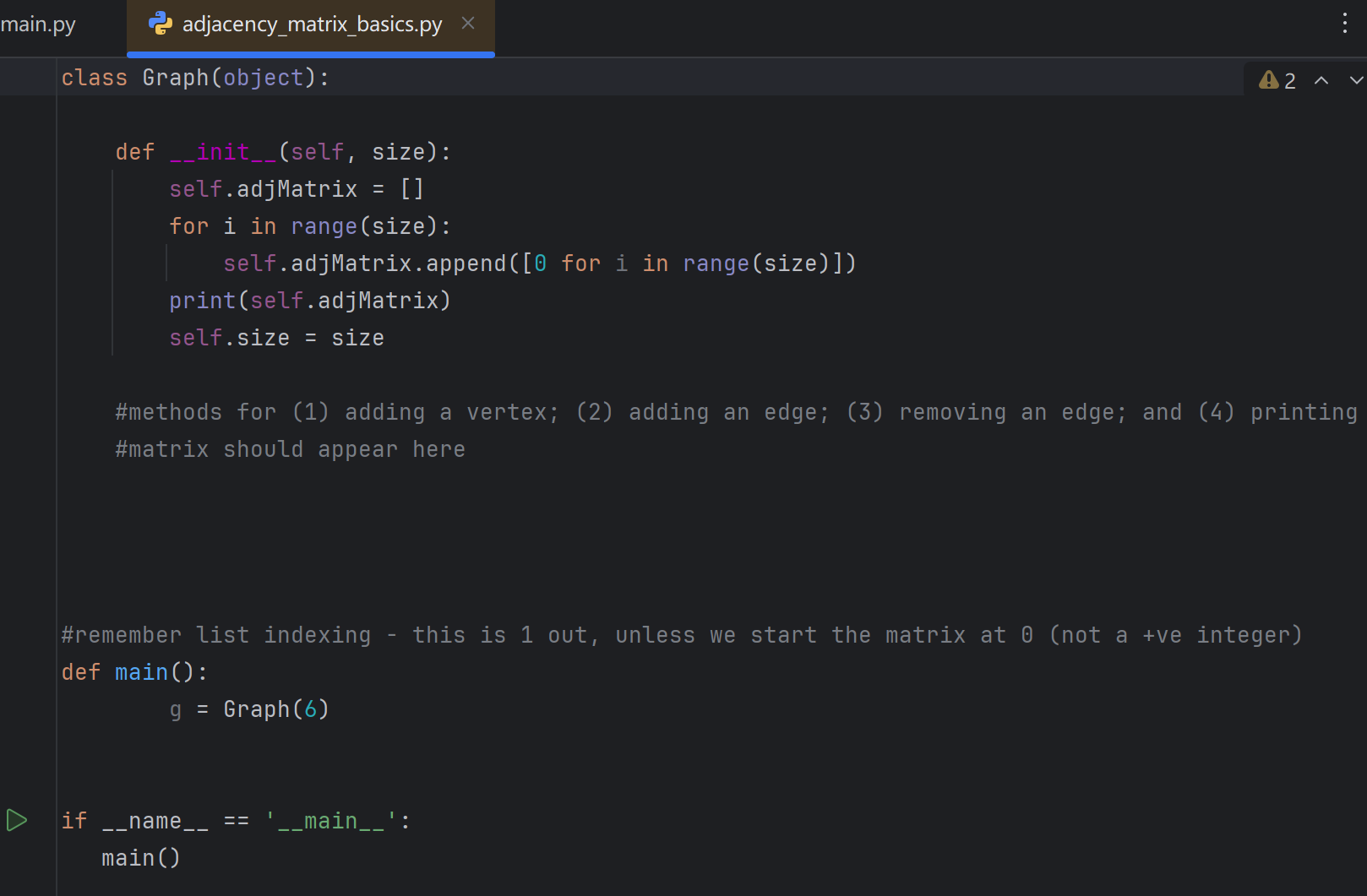
**6 |   0   0   0   0   0   0**

IMPORTANT:

The code as given may need altering to allow the new methods to be introduced.

Edges can be removed with no need to worry about vertices.

NOTE: If you wish, you could implement the entire code as C++ instead of, or in addition to, python. This is not mandatory for this task.



**Assessed Standard Task 5:** Implement an Insert Method for a Linked List

For this task:

You will need to have studied the lecture Data Structures 2, on Aula, Week 5, Slides 52 to 83.

Use the code in the folder on Aula called LINKED-LIST-STD-TASK-5. Add an insert method to correctly add the day of the week that is missing (NB the week runs from Monday to Sunday).

The code currently outputs the following:

**Mon**

**Tue**

**Thur**

**Fri**

**Sat**

**Sun**

So we need to add ‘Weds’.

To do this, create an insert method. Here is some code to get you started. You’ll need to add code after the else:

**def Insert(self,val\_before,newdata):**

**if val\_before is None:**

**print("No node to insert after")**

**return**

**else:**

**# your code here**

You will need to add a new command to insert the missing value, and this should appear before the existing command to print the list.

The result should be:

**Mon**

**Tue**

**Weds**

**Thur**

**Fri**

**Sat**

**Sun**

**Assessed Advanced Task 1/3:** Remove method for Binary Tree

class 

From the partial pseudocode given below (one case is omitted), implement an iterative method called remove which deletes a node and reorganises the tree. There are indications where the pseudocode is missing. NB the pseudocode crosses pages.

Add comments to show your understanding.

Implement your solution into the python Binary Tree class given on this week’s Aula in the zip folder, ‘BST-class’.

Make sure that remove works correctly; that is, not only is the target node deleted, but the tree is also correctly re-organised. If not, it may mean the pseudocode needs some detail adding.

REMOVE(tree, target)

IF tree.root IS None //if no tree

RETURN False

ELSE IF tree.root.data = target //if tree root is target

IF tree.root.left IS None AND tree.root.right IS None

tree.root ← None

ELSE IF tree.root.left AND tree.root.right IS None

tree.root ← tree.root.left

ELSE IF tree.root.left IS None AND tree.root.right

tree.root ← tree.root.right

ELSE IF tree.root.left AND tree.root.right

IF\_LEFT\_AND\_RIGHT(tree.root)

(continues over)

//if root is not target

parent ← None

node ← tree.root

WHILE node and node.data != target

parent ← node

IF target < node.data

node ← node.left

ELSE IF target > node.data

node ← node.right

IF node IS None OR node.data != target //CASE 1: Target not found

RETURN False //for info only (we could not find it)

ELSE IF node.left IS None AND node.right IS None //CASE 2: Target has no children

IF target < parent.data

parent.left ← None

ELSE

parent.right ← None

RETURN True //info only

ELSE IF node.left AND node.right IS None //CASE 3: Target has left child only

IF target < parent.data

parent.left ← node.left

ELSE

parent.right ← node.left

RETURN True //info only

NOT IMPLEMENTED //CASE 4: Target has right child only

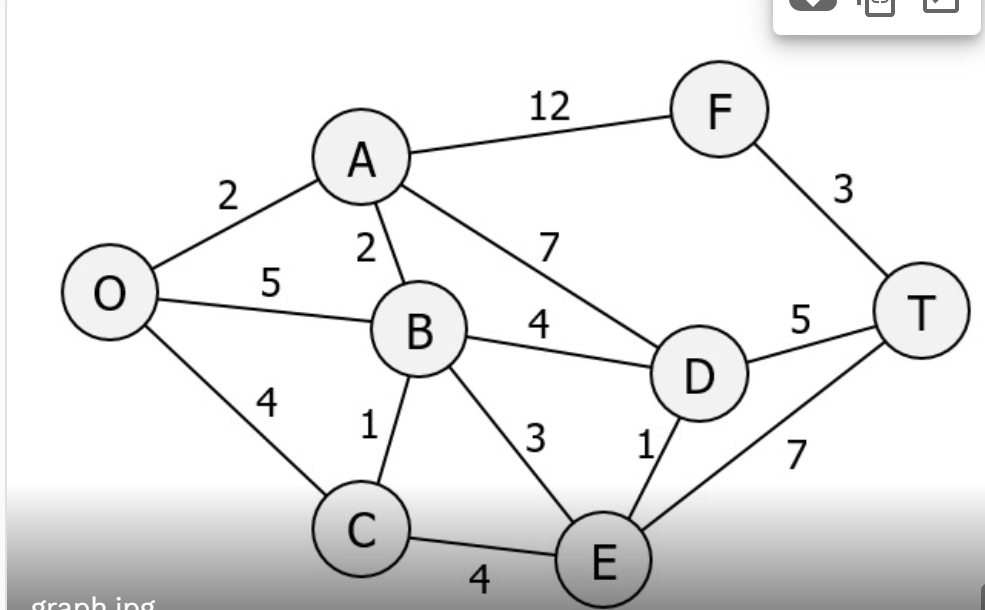
ELSE //CASE 5: Target has left and right children

IF\_LEFT\_AND\_RIGHT(node)

**Assessed Advanced Task 2: Implement Dijkstra’s Algorithm**

Using the given code on Aula, in the folder DIJKSTRA-CODE-ADV-TASK-2, complete the existing Python code to produce code that works for the example below (same as Week 6 lecture), plus any other (come up with alternative graphs for testing purposes). You need to complete the code where indicated.

The code, if working, will output the correct solution for the graph below.



The output from your code should look something like this – a list of the nodes which is the lowest cost path, as well as the cost itself:

**(['O', 'A', 'B', 'D', 'T'], 13)**

