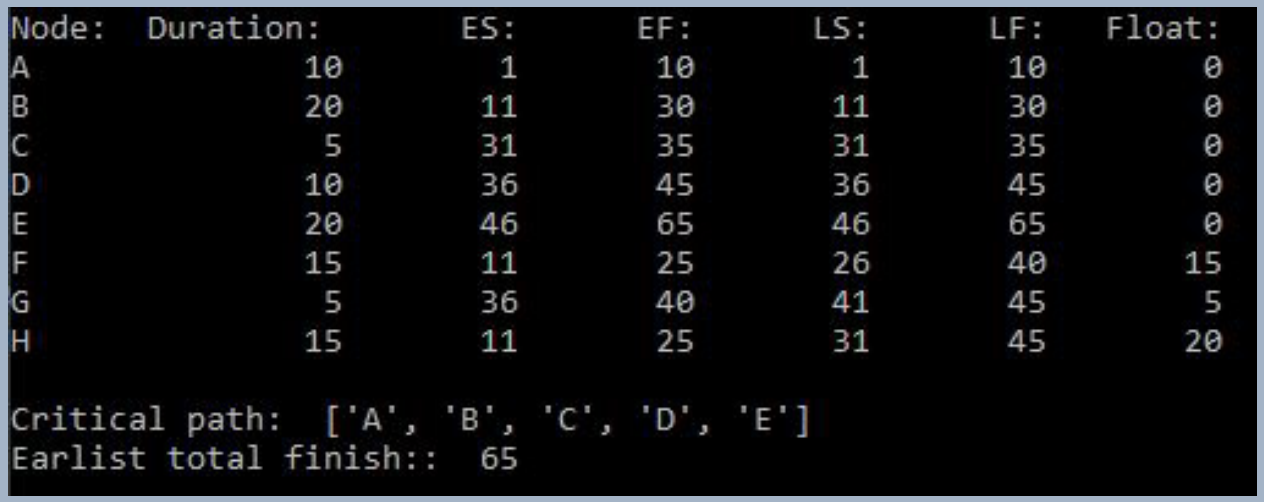
Critical Path

Assignment:
The picture from Wikipedia shows a graph with critical path and details shown. (By Agarcialw - Own work, CC BY-SA 4.0)

Make a general program, which set up the graph with the tasks (A-H) and their durations only.

Make the program find the critical path.



Node:	Duration:	ES:	EF:	LS:	LF:	Float:
A	10	1	10	1	10	0
B	20	11	30	11	30	0
C	5	31	35	31	35	0
D	10	36	45	36	45	0
E	20	46	65	46	65	0
F	15	11	25	26	40	15
G	5	36	40	41	45	5
H	15	11	25	31	45	20

Critical path: ['A', 'B', 'C', 'D', 'E']
Earlist total finish:: 65

Image 1:
A snippet of the console output from my Critical Path program based on the data from the diagram from Wikipedia.

Critical Path

Set up

For creating a CriticalPath object, (Image 1), there need to be an list with Nodes. (Image 2)

The only thing the program should call in the CriticalPath class is the calculate() and printSchema() functions.

The **calculate()** will call all the functions needed to calculate the project flow and critical path.

The **printSchema()** (Image 3) will just output the result in the console, and should be called after the calculate().

1

```
class CriticalPath():
    def __init__(self, graph):
        self._graph = graph
        self._numberofnodes = len(self._graph)
        self._finishnode = []
        self._minlenght = [100000] * self._numberofnodes
        self._minlenght[0] = 0
        self._predecessors = []
        self._theCriticalPath = []

    def calculate(self):
        self.findEarlist()
        self.findFinishNode()
        self.findLatest()
        self.findCiticalPath()
```

2

```
class Node():
    def __init__(self, name, duration, predecessors):
        self._name = name
        self._duration = duration
        self._earlieststart = 1
        self._earliestfinish = 0
        self._lateststart = 0
        self._latestfinish = 0
        self._totalfloat = None
        #self._totalDrag = None
        self._done = False
        self._predecessors = predecessors

projectGraph = [
        Node("A", 10, []),
        Node("B", 20, ["A"]),
        Node("C", 5, ["B"]),
        Node("D", 10, ["C"]),
        Node("E", 20, ["D", "G", "H"]),
        Node("F", 15, ["A"]),
        Node("G", 5, ["C", "F"]),
        Node("H", 15, ["A"])
        ]
```

3

```
def printSchema(self):
    print("{:6} {:10} {:8} {:8} {:8} {:8} {:8}".format("Node: ", "Duration:", "
    ES:", "     EF:", "     LS:", "     LF:", "  Float:"))
    for i in range(0, self._numberofnodes):
        print("{:6} {:10} {:8} {:8} {:8} {:8} {:8}".format(self._graph[i]._name,
        self._graph[i]._duration, self._graph[i]._earlieststart,
        self._graph[i]._earliestfinish, self._graph[i]._lateststart,
        self._graph[i]._latestfinish, self._graph[i]._totalfloat))
    print()
    print("Critical path: ", self._theCriticalPath)
    print("Earlist total finish:: ", self._finishnode._earliestfinish)
```

4

```
def findEarlist(self):
    # Calculating Earlist start
    for i in range(0, self._numberofnodes):
        currentnode  = self._graph[i]
        for j in range(0, self._numberofnodes):
            presessorNode = self._graph[j]
            if presessorNode._name in currentnode._predecessors :
                currentES = presessorNode._duration + presessorNode._earlieststart
                if(currentES > currentnode._earlieststart):
                    currentnode._earlieststart = currentES
    # Calculating Earlist finish
    for i in range(0, self._numberofnodes):
        currentnode  = self._graph[i]
        currentnode._earliestfinish = currentnode._earlieststart +
        currentnode._duration -1
```

5

```
def findFinishNode(self):
    # Find finish node
    self._finishnode = ""
    finishnodedur = 0
    for i in range(0, self._numberofnodes):
        if self._graph[i]._earlieststart > finishnodedur:
            self._finishnode = self._graph[i]
            finishnodedur = self._graph[i]._earlieststart
    self._finishnode._latestfinish = self._finishnode._earliestfinish
    self._finishnode._lateststart = self._finishnode._earlieststart
    return self._finishnode
```

Critical Path

Calculate step 1

When the calculate() function is called the program will first calculate **Earliest start** and **Earliest finish** for each node inside the list . (Image 4)

Afterwards, the program will find the **finish node** and set the latest start and finish for that node. (Image 5)

Critical Path

Calculate step 2

After it has found the **Latest start** and **Lastest finish** for the last node, the program calculate it for every node by using the **bubble sort** algorithm.

```
def findLatest(self):
    orderedList = self._graph
    # Order list with bubble sort algorithms
    for i in range(0, self._numberofnodes):
        for j in range(0, self._numberofnodes - i - 1):
            if orderedList[j]._earliestfinish < orderedList[j+1]._earliestfinish :
                swap = orderedList[j]
                orderedList[j] = orderedList[j+1]
                orderedList[j+1] = swap
    #Find Latest finish and start
    for i in range(0, self._numberofnodes):
        currentNode = orderedList[i]
        for j in range(0, self._numberofnodes):
            presessorNode = self._graph[j]
            if presessorNode._name in currentNode._predecessors:
                latestfinish = currentNode._lateststart
                if latestfinish < presessorNode._latestfinish or
                presessorNode._latestfinish == 0:
                    presessorNode._latestfinish = latestfinish - 1
                    presessorNode._lateststart = latestfinish -
                    presessorNode._duration
    # Order the list back
    for i in range(0, self._numberofnodes):
        for j in range(0, self._numberofnodes - i - 1):
            if orderedList[j]._name > orderedList[j + 1]._name:
                swap = orderedList[j]
                orderedList[j] = orderedList[j+1]
                orderedList[j+1] = swap
```

```
def findFloat(self):
    # find float for the critical path
    for i in range(0, self._numberofnodes):
        tFloat = self._graph[i]._latestfinish - self._graph[i]._earliestfinish
        self._graph[i]._totalfloat = tFloat

def findCiticalPath(self):
    self.findFloat()
    # The finish node has to be found before this function
    currentNode = self._finishnode
    for i in range(0, self._numberofnodes):
        if self._graph[i] == currentNode:
            self._theCriticalPath.append(currentNode._name)
    thepredecessors = currentNode._predecessors
    while len(thepredecessors) != 0:
        for i in range( 0, len(thepredecessors)):
            currentName = thepredecessors[i]
            for j in range(0, self._numberofnodes):
                if self._graph[j]._name == currentName:
                    currentNode = self._graph[j]
            if currentNode._totalfloat == 0:
                self._theCriticalPath.append(currentNode._name)
                for h in range(0, len(currentNode._predecessors)):
                    thepredecessors.append(currentNode._predecessors[h])
            thepredecessors.pop(0)
            break
    self._theCriticalPath.sort()
```

Use of API

Calculate step 3

In the end, the calculate() calls for the **findCriticalPath()** function, which finds the critical path by calling the **findFloat()** function at a start.