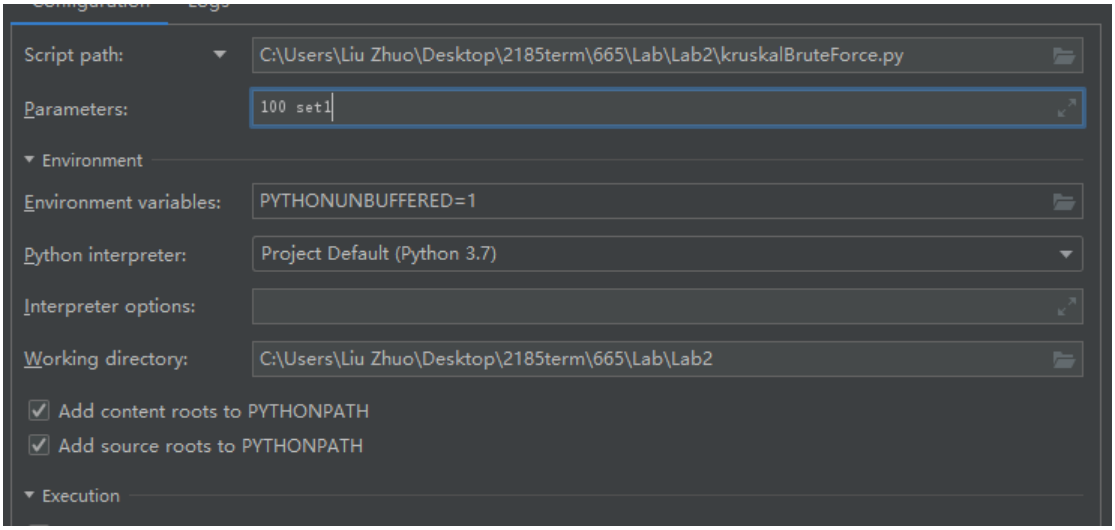
Lab 2 Report

CS665
Zhuo Liu
zl9901

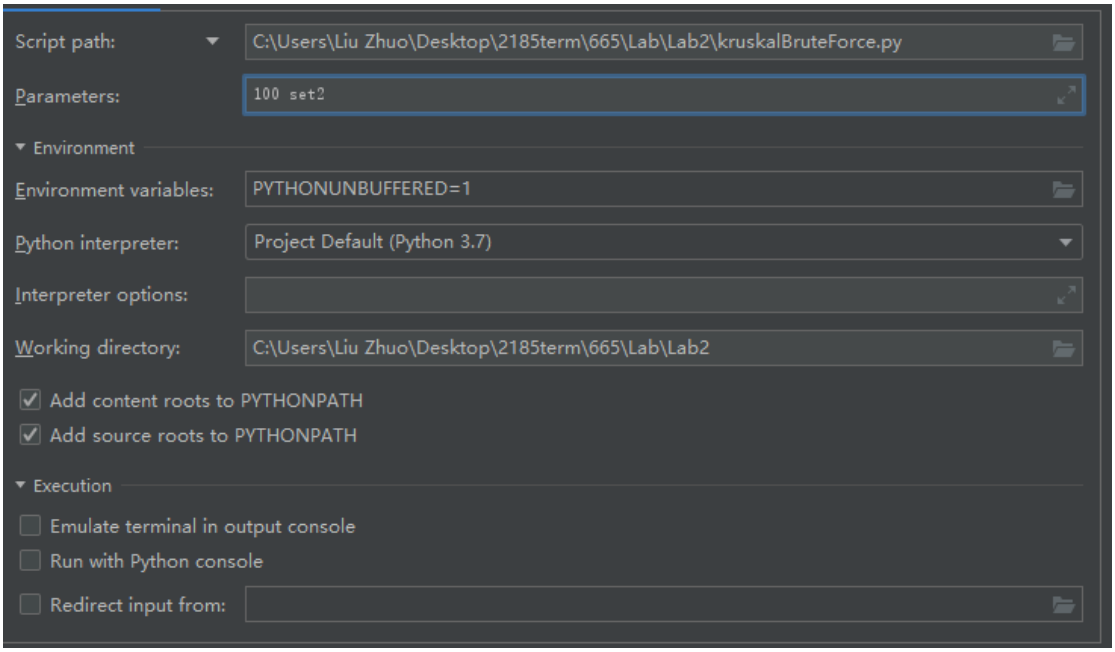
In both of my code **kruskalBruteForce.py and GraphInputGenerator.py,** there are two parameters in command line.
The first is the number of vertices and the second is the option whether you choose set1, set2 or set3.

Set1 represents that node connectivity less than 10
Set2 represents that node connectivity more than n/2
Set3 represents that randomly generated Graphs
For example, you just need to input as followed:

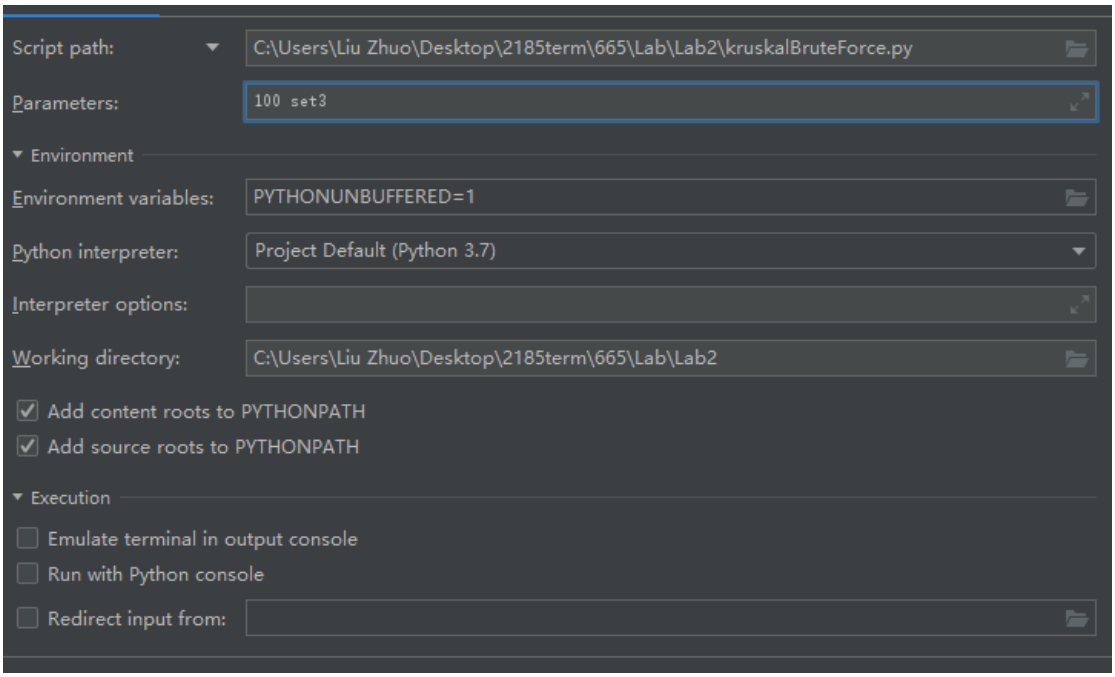
In the command line then you can get the output by input:



Or


Script path:
C:\Users\Liu Zhuo\Desktop\2185term\665\Lab\Lab2\kruskalBruteForce.py
Parameters:
100 set2
Environment
Environment variables:
PYTHONUNBUFFERED=1
Python interpreter:
Project Default (Python 3.7)
Interpreter options:
Working directory:
C:\Users\Liu Zhuo\Desktop\2185term\665\Lab\Lab2
Add content roots to PYTHONPATH
Add source roots to PYTHONPATH
Execution
Emulate terminal in output console
Run with Python console
Redirect input from:

Or


Script path:
C:\Users\Liu Zhuo\Desktop\2185term\665\Lab\Lab2\kruskalBruteForce.py
Parameters:
100 set3
Environment
Environment variables:
PYTHONUNBUFFERED=1
Python interpreter:
Project Default (Python 3.7)
Interpreter options:
Working directory:
C:\Users\Liu Zhuo\Desktop\2185term\665\Lab\Lab2
Add content roots to PYTHONPATH
Add source roots to PYTHONPATH
Execution
Emulate terminal in output console
Run with Python console
Redirect input from:

Algorithm

```
def unionAndFind(self):
    index=0
    e=0
    for i in range(0,self.vertices):
        self.parent.append(i)
    for j in range(0,self.vertices):
        self.rank.append(0)
    while e<self.vertices and index<=self.vertices-1:
        u=self.graph[index][0]
        v=self.graph[index][1]
        x=self.find(u)
        y=self.find(v)
        if x!=y:
            e+=1
            xroot=self.find(x)
```

```
def find(self,i):
    if self.parent[i]==i:
        return i
    else:
        return self.find(self.parent[i])


def union(self,x,y):
    if self.rank[x]==self.rank[y]:
        self.parent[y]=x
        self.rank[x]+=1
    elif self.rank[x]>self.rank[y]:
        self.parent[y]=x
    elif self.rank[y]>self.rank[x]:
```

These three figures shown above is kruskal's union and find algorithm which are in my **kruskalBruteForce.py**

```
def BruteForce(self):
    e=0
    index=0
    while e<self.vertices and index<=self.vertices-1:
        if index==0:
            array=[]
            u = self.graph[index][0]
            v = self.graph[index][1]
            array.append(u)
            array.append(v)
            self.visited.append(array)
            self.res.append(self.graph[index])
            index+=1
```

```
            self.mark2=i
if self.mark1==-1 and self.mark2!=-1:
    self.visited[self.mark2].append(u)
    self.res.append(self.graph[index])
elif self.mark1!=-1 and self.mark2==-1:
    self.visited[self.mark1].append(v)
    self.res.append(self.graph[index])
elif self.mark1==-1 and self.mark2==-1:
    temp=[]
    temp.append(u)
    temp.append(v)
    self.visited.append(temp)
    self.res.append(self.graph[index])
elif self.mark1!=-1 and self.mark2!=-1:
    if self.mark1!=self.mark2:
        self.visited[self.mark1].extend(self.visited[self.mark2])
        self.visited.pop(self.mark2)
        self.res.append(self.graph[index])
```

BruteForce function is the algorithm without using union and find data structure.

As the picture shown above , I have a visited array in my class, there are four situations,

The two vertices of one edge,

1) Both of two vertices are in the visited, then we discard it
2) First vertex is in the visited array while the second is not, we append this vertex in the visited and then adopt this edge
3) Second vertex is in the visited list while the first one is not, we append second vertex in the visited list and then adopt this edge
4) Both of two vertices are not in the visited, we add these two in the visited list

Because of the time complexity is high, it is time consuming.

And I modified the GraphInputGenerator.java provided by Professor, the codes which I modified are both in **kruskalBruteForce.py and GraphInputGenerator.py** , I list the algorithm independently which I can use it for my sample data, you can follow the instruction which is stated at the beginning of this report to see my sample data **in GraphInputGenerator.py**

The followed picture is the modified code in my **kruskalBruteForce.py**

```
def fillAdj(self):
    if self.set == 'set1':
        for v1 in range(self.vertices):
            count = 0
            while count < 9:
                v2 = v1
                while v2 == v1:
                    v2 = randint(0, self.vertices - 1)
                if self.connectivity_check(v1) < 9 and self.connectivity_check(v2) < 9:
                    self.addEdge(v1, v2)
                count += 1
    elif self.set == 'set2':
        for v1 in range(self.vertices):
```

```
def connectivity_check(self, v):
    count = 0
    for i in range(len(self.adj)):
        if i != v and self.adj[v][i] != 0:
            count += 1
    return count


def addEdge(self,v1,v2):
    if self.adj[v1][v2]==0:
        e=1
        if self.isWeighted:
            e+=randint(0,int(self.edges*self.edges/2))
        self.adj[v1][v2]=e
        if not self.isDirected:
```

The connectivity check is used to make sure the connectivity is correct

```
def addEdge(self,v1,v2):
    if self.adj[v1][v2]==0:
        e=1
        if self.isWeighted:
            e+=randint(0,int(self.edges*self.edges/2))
        self.adj[v1][v2]=e
        if not self.isDirected:
            self.adj[v2][v1]=e
        return True
    return False

def genGraphInput(self):
    # print(str(self.vertices)+" "+str(self.edges))
    print("number of vertices are "+str(self.vertices))
    print()
    for i in range(0,len(self.adj)):
        for j in range(0 if self.isDirected else i,len(self.adj)):
```

Graph size 40(the result is the average of 10 reruns of the execution)

	Set_1 less than 10	Set_2 more than n/2	Set_3 Random Graphs
BruteForce	0.0008023	0.0007012	0.0007027
UnionAndFind	0.0005087	0.0005147	0.0005018

Unit: CPU time(second)

Graph size 60(the result is the average of 10 reruns of the execution)

	Set_1 less than 10	Set_2 more than n/2	Set_3 Random Graphs
BruteForce	0.000995635986328125	0.001497030258178711	0.0014972686767578125
UnionAndFind	0.0004992485046386719	0.0004992485046386719	0.0004990100860595703

Unit: CPU time(second)

Graph size 100(the result is the average of 10 reruns of the execution)

	Set_1 less than 10	Set_2 more than n/2	Set_3 Random Graphs
BruteForce	0.000995635986328125	0.0014812946319580	0.0014834403991699219
Union and Find	0.0004992485046386719	0.0004992485046386	0.000499725341796875

Unit: CPU time(second)

As the table shown above, we can see the Union and Find data structure tends to be more efficient than my BruteForce algorithm.

Please notice:
Because the size of the graph is somewhat a little bit small, sometimes the CPU time might be 0.0s
It is because the program runs so fast