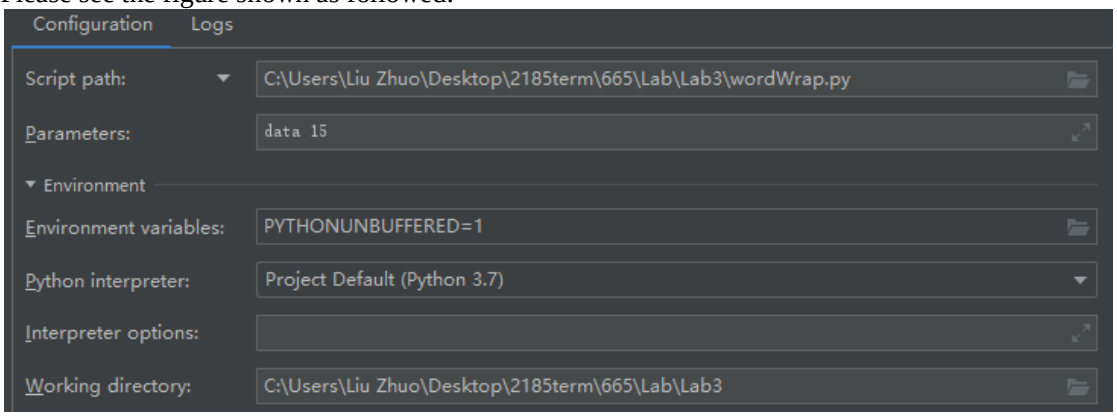
Lab 3 Report

CS665
Zhuo Liu
zl9901

First assignment is considered to be **wordWrap.py**, there are two parameters in the command line.
First is the filename of one page paragraph.
Second is the width you want this page of text to be.
Please see the figure shown as followed:




```
def solveWordWrap(fulltxt,wordLength, n, width):
    val=[[0 for j in range(1,n+1)] for i in range(1,n+1)]

    for i in range(0,n):
        for j in range(i,n):
            if i==j:
                val[i][j]=width-wordLength[j]
            else:
                val[i][j]=val[i][j-1]-wordLength[j]-1

    cost = [[0 for j in range(1, n + 1)] for i in range(1, n + 1)]
```

solveWordWrap function is to use dynamic programming method to print neatly.

```
def showResult(fulltxt,pointer,index):
    if pointer[index]==1:
        # print((pointer[index],index))
        if pointer[index]==index:
            print(fulltxt[index-1])
        else:
            for i in range(pointer[index],index+1):
                print(fulltxt[i-1],end="")
                if i==index:
                    print(" ")
                else:
                    print(" ",end="")
        return
```

The purpose of showResult function is to print the one page of text in a neat order.

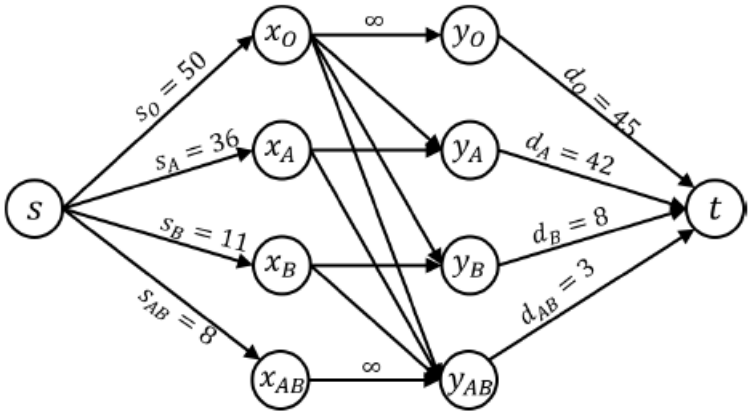
```
def processData():
    fulltxt=[]

    """
    give the file name which needs to be printed
    """
    filename = sys.argv[1] + '.txt'
    with open(filename) as f:
        for line in f:
            word1 = " ".join(re.findall("[a-zA-Z]+", line))
            data=word1.split()
            fulltxt.extend(data)
    count=[]
    for i in range(0,len(fulltxt)):
        count.append(len(fulltxt[i]))
```

The purpose of processData is to remove some “noises” such as comma in the one page of text by using regular expression. Since we only want the words themselves.

Time complexity of dynamic programming: O(n^2).
Space complexity of dynamic programming: O(n^2).
n is the number of words in the input text.

Please note:
Since the complexity of this algorithm is O(n^2), the command line parameter width should not be too small, my suggestion is to set this parameter greater than 15. Because if the width of a word is longer than 15, there will never be a position for this word to place.

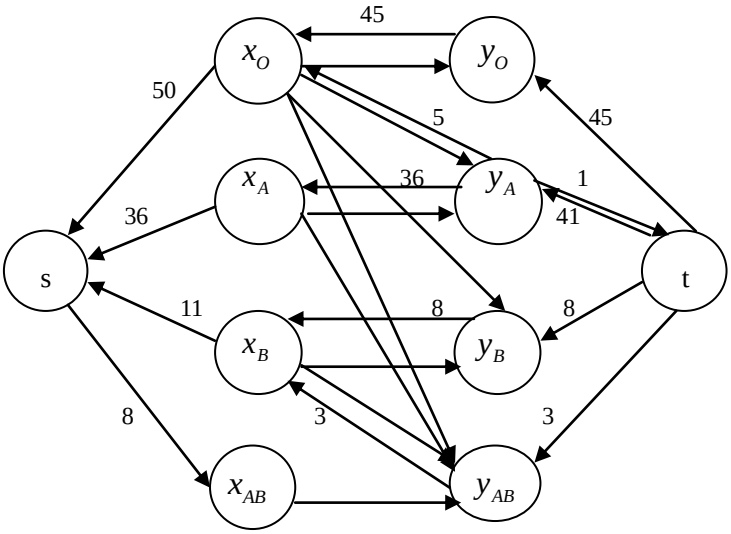
Second assignment is considered to be **bloodType.py**, there is no parameter in the command line.



Here is the graph I create.
In my code, I use an adjacent list to represent the above graph accordingly.

```
graph = [[0, 50, 36, 11, 8, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, math.inf, math.inf, math.inf, math.inf, 0],
         [0, 0, 0, 0, 0, 0, math.inf, 0, math.inf, 0],
         [0, 0, 0, 0, 0, 0, 0, math.inf, math.inf, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, math.inf, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 45],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 42],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 8],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 3],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         ]
```

As the picture shown above, 0 means there is no path between two nodes.



We can see there are no paths from s to t in the residue graph, so our run of Ford-Fulkerson is complete. **The max-flow we found has value 97.**
As we can see, it is not possible to meet the full requirement in this graph. The **total demand is 98** and the max-flow which I calculate is 97. It is a known fact that max-flow is equal to min-cut.
One cut that has value 97 is S={s, x_B, y_B, x_{AB}, y_{AB} }, T={ x_O, y_O, x_A, y_A }.
Min-cut=50+36+8+3=97 is shown above.
Furthermore, if we look at the subset of O and A, the total demand for these blood types is

45+42=87. But the total supply that can be used in transfusions to either blood type is only 50+36=86, still we **can not** meet the full requirement.