Lab lll Report

Rishab Katta
CSCI-665 Foundations of Algorithms

April 26, 2019

Pretty Printing using Dynamic Programming

Given a set of words and limit on the number of characters you can print on every line, how do you arrange these words on multiple lines such that the number of empty spaces is evenly distributed. We will follow dynamic programming approach to solve this problem.

Algorithm

First we'll populate the matrix. Let's look at how to populate the matrix. We'll take the following set of words.
"Rishab" index 0
"Katta" index 1
"likes" index 2
"to" index 3
"code" index 4
And also the let's assume the width to be 10.

Now in the (0,0) position of the matrix - starting from 0th postion and ending at 0th position on line 1, we calculate the number of empty spaces- 10-6=4 and square of the value is 16. We store 16 in (0,0). Now in the matrix position (0,1) - starting from 0 ending at 1, that is "rishab" and "katta" on first line = 6+1+5 = 12 but the total width is 10. so we can't fit "rishab" and "katta" on 1 line. In such a case we'll store infinity(very large positive number). In such a way we'll fill out all the positions in the matrix.

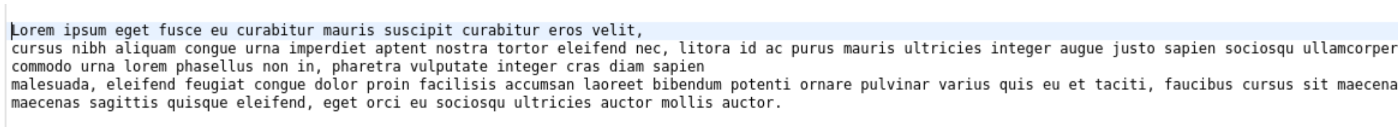
Next we'll use this matrix to decide at which point we'll put the line breakers.
We'll use two arrays - a min_cost array and fin_result array. For the min_cost array we'll keep two pointers i and j denoting the matrix positions(i,j) and the len of both the arrays = number of words and the pointers i,j will start at the end of min_cost array. When the pointers are at the end they'll point to (4,4) and we look at the matrix if there's value at matrix position (4,4) which is not infinity and if there is we populate that value in our min_cost array and we'll store the j+1 in our fin_result array. Now we decrement the i and j stays the same and populate the min_cost array. If at any point the matrix position has

infinity we'll have to decide at what point do we split the array. Now we decrement j until it's less than i and calculate the total cost for each arrangement and pick the least cost split. After we populate that particular position in the min_cost array, we'll put j back to last position in the array.

```
wrap(self, text, width):
cost =[[]]  Initialize the cost matrix

for i in range(0,len(text)):
    cost[i][i] = width - len(text[i])
    for j in range(i+1, len(text)):
        cost[i][j] = cost[i][j-1] - len(text[j])-1

for i in range(0,len(text)):
    for j in range(i,len(text)):
        if cost[i][j]<0:
            cost[i][j] = sys.maxsize
        else:
            cost[i][j] = cost[i][j]**2

min_cost=[0 for _ in range(len(text))]
fin_result=[0 for _ in range(len(text))]

for i in range(len(text)-1, -1, -1):
    min_cost[i] = (cost[i][len(text)-1])
    fin_result[i] = (len(text))
    for j in range(len(text)-1,i,-1):
        if cost[i][j-1] == sys.maxsize:
            continue
        if min_cost[i] > min_cost[j] + cost[i][j-1]:
            min_cost[i] = min_cost[j] + cost[i][j-1]
            fin_result[i]=j
fin_list=[]
i = 0
j = fin_result[i]
while True:
    if j>=len(text):
        break
    j = fin_result[i]
    for k in range(i,j):
        fin_list.append(str(text[k]) + " ")
    fin_list.append("\n")
    i=j
return ''.join(fin_list)
```

Results

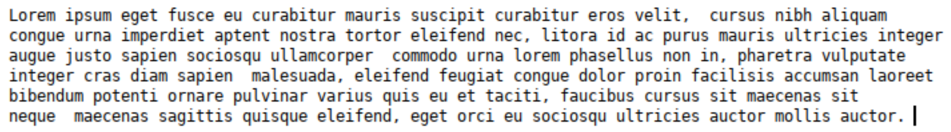
Input is the random text file with the text arranged in the below format.



```
Lorem ipsum eget fusce eu curabitur mauris suscipit curabitur eros velit,
cursus nibh aliquam congue urna imperdiet aptent nostra tortor eleifend nec, litora id ac purus mauris ultricies integer augue justo sapien sociosqu ullamcorper
commodo urna lorem phasellus non in, pharetra vulputate integer cras diam sapien
malesuada, eleifend feugiat congue dolor proin facilisis accumsan laoreet bibendum potenti ornare pulvinar varius quis eu et taciti, faucibus cursus sit maecena
maecenas sagittis quisque eleifend, eget orci eu sociosqu ultricies auctor mollis auctor.
```

The width input given is 100.

The output is printed as follows:



```
Lorem ipsum eget fusce eu curabitur mauris suscipit curabitur eros velit,  cursus nibh aliquam
congue urna imperdiet aptent nostra tortor eleifend nec, litora id ac purus mauris ultricies integer
augue justo sapien sociosqu ullamcorper  commodo urna lorem phasellus non in, pharetra vulputate
integer cras diam sapien  malesuada, eleifend feugiat congue dolor proin facilisis accumsan laoreet
bibendum potenti ornare pulvinar varius quis eu et taciti, faucibus cursus sit maecenas sit
neque  maecenas sagittis quisque eleifend, eget orci eu sociosqu ultricies auctor mollis auctor.
```

The time complexity and space complexity for this program is $\mathbf{O}(n^2)$

Instructions to run code

when "enter file path with uneven end-spacing without any quotation marks" pops up on the console, enter a file path with out any quotation marks.
/home/user/Documents/randomtext.txt like so and press enter

Blood Supply-Demand Network flow Problem.

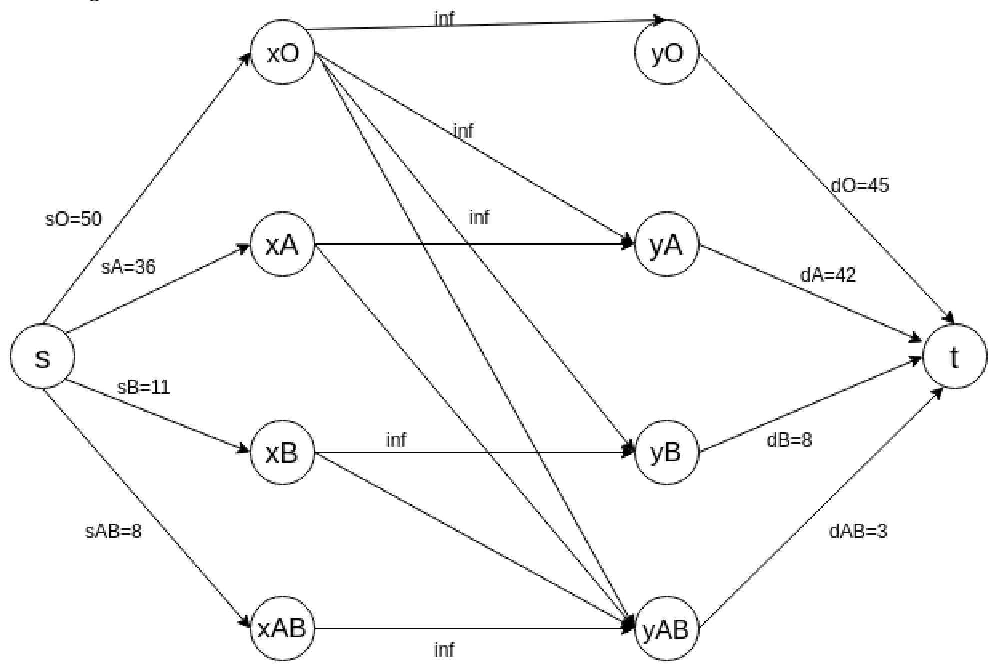
The problem here is to determine whether the current supply of blood will be enough to meet the projected demand over the next week for blood transfusions.

Algorithm

We are given a set of supplies available and a set of demands to meet. They are given below.

blood type	supply	demand
O	50	45
A	36	42
B	11	8
AB	8	3

When we have all the supply values as nodes on one side and all the demand values as nodes on the other side, this problem resembles a multisource-multisink network flow problem. In case of the multisource-multisink network flow, we add two extra nodes - artificial source(s) and artificial sink(t) and connect all the supply nodes to s with values as given in the question and all demand nodes to t with values given in the question. The interconnection between supply and demand nodes depends on the Antigen matching condition given in the textbook and the values for those edges will infinity because there's no constraint on the amount you can transfer between those nodes. The final diagram of the network looks something like this.



The graph representation of the above diagram in the form of adjacency matrix is given as follows:

```
                 s   x0  xA  xB  xAB y0 yA yB yAB t
adj_matrix = s[[0, 50, 36, 11,  8,  0, 0, 0, 0, 0],
            x0   [0, 0, 0, 0, 0, i, i, i, i, 0],
             xA  [0, 0, 0, 0, 0, 0, i, 0, i, 0],
             xB  [0, 0, 0, 0, 0, 0, 0, i, i, 0],
            xAB  [0, 0, 0, 0, 0, 0, 0, 0, i, 0],
             y0  [0, 0, 0, 0, 0, 0, 0, 0, 0, 45],
             yA  [0, 0, 0, 0, 0, 0, 0, 0, 0, 42],
             yB  [0, 0, 0, 0, 0, 0, 0, 0, 0, 8],
             yAB [0, 0, 0, 0, 0, 0, 0, 0, 0, 3],
              t[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Then we run max flow algorithm on the following graph to determine if the supply is enough to meet the demand.We use BFS to determine if there is a path from source to sink.

The Algorithm is as follows:

```
Breadth_First_Search(source, sink, parent_list):

    visited_set = [False]*self.matrix_row #initially all vertices are not visited.
    bfs_queue=[]
    visited_set[source] = True
    bfs_queue.append(source)
    while len(bfs_queue)!=0:
        node = bfs_queue.pop(0)
        for index, value in enumerate(self.dir_graph[node]):
            if value>0 and visited_set[index]==False:
                bfs_queue.append(index)
                visited_set[index] = True
                parent_list[index] = node
    if visited_set[sink]==True:
        return True
    return False

Ford_Fulkerson(source, sink):

    parent_list=[-1]*self.matrix_row
    max_flow = 0                    #initialize the max flow to 0.
    while self.Breadth_First_Search(source,sink,parent_list):
        current_flow = sys.maxsize
        s=sink
        while(s!=source):
            current_flow = min(current_flow, self.dir_graph[parent_list[s]][s])
```

```
                s=parent_list[s]
            max_flow +=current_flow
            v=sink
            while(v!=source):
                u = parent_list[v]
                self.dir_graph[u][v] = self.dir_graph[u][v] - current_flow
                self.dir_graph[v][u] = self.dir_graph[v][u] + current_flow
                v=parent_list[v]
        return max_flow
```

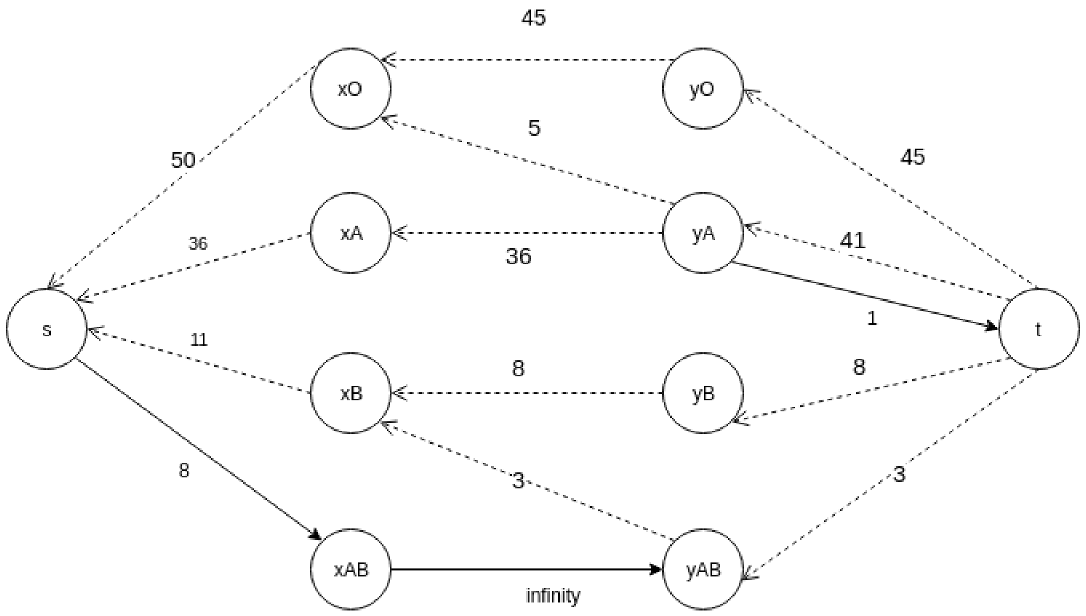
Results

When we run the max flow algorithm on the given Blood supply demand network for the given input, We get the max flow in the network as 97. If we sum up all the demand in our input, we get the total demand of 98.

The adjacency matrix after running the max flow algorithm gives the residual capacities. The residual capacities are given by the below adjacency matrix.

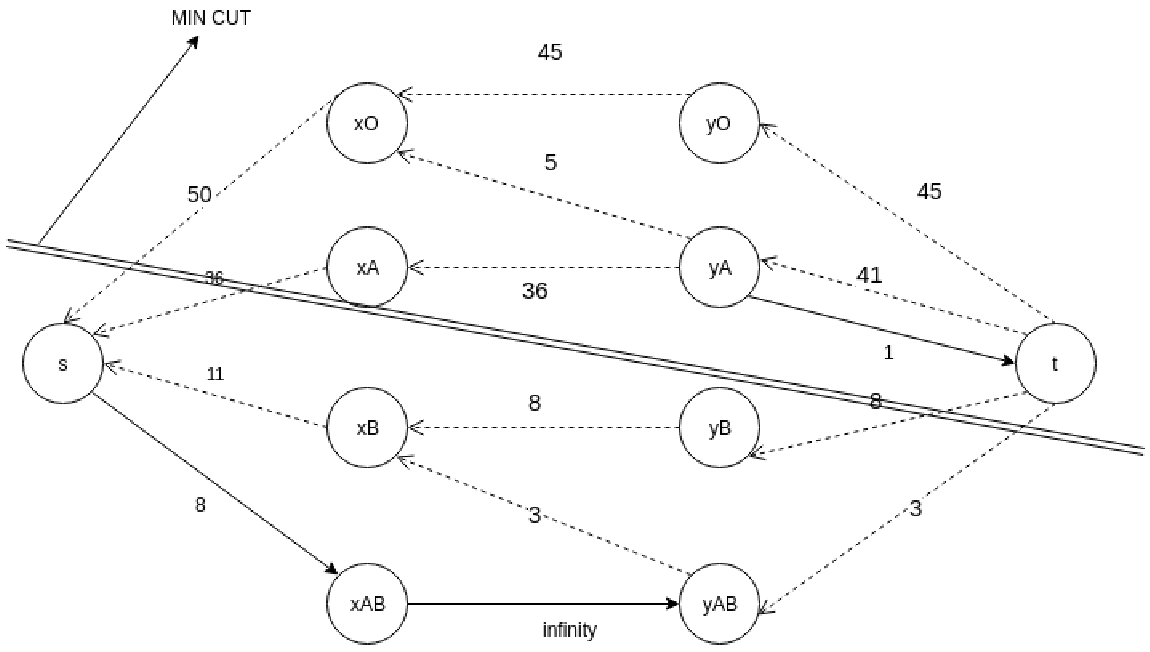
```
residual graph =[0, 0, 0, 0, 8, 0, 0, 0, 0, 0]
                [50, 0, 0, 0, 0, i, i, i, i, 0]
                [36, 0, 0, 0, 0, 0, i, 0, i, 0]
                [11, 0, 0, 0, 0, 0, 0, i, i, 0]
                [0, 0, 0, 0, 0, 0, 0, 0, i, 0]
                [0, 45, 0, 0, 0, 0, 0, 0, 0, 0]
                [0, 5, 36, 0, 0, 0, 0, 0, 0, 1]
                [0, 0, 0, 8, 0, 0, 0, 0, 0, 0]
                [0, 0, 0, 3, 0, 0, 0, 0, 0, 0]
                [0, 0, 0, 0, 0, 45, 41, 8, 3, 0]
```

After drawing a graph with back edges(with infinity edges intentionally left out to make residual capacities more clear.), **We find that the demand for the Blood Type "A" has not been met by the supply of the hospital by 1 unit.**



Also from the residual graph we get certain insights like, the blood supply for type "AB" has not been used at all. The 3 units of blood demand for "AB" has been met by type "B".Also like, the extra demand for type "A" has been met by 5 units of type "O".

The min cut is shown below.



Flow across the min-cut is 50+36+8+3 = 97. we know that the value of edges across min-cut is the max flow in the graph. The total demand is 98 and total supply possible is 97. So we could not meet the demand of A type blood by 1 unit.

The time complexity for this algorithm is **O(E*f)** E is the number of edges in the graph and f is the maximum flow in the graph, which is equivalent to O(E)