TEXT JUSTIFICATION

We use dynamic programming to print a given text in a neat manner. “Neat manner” is defined as the utilizing as much space as possible for each line. Here we ensure that the sum of left-over spaces in the entire text is as low as possible.

We first define the character limit for each line in our printing. Here I have chosen this to be 100 characters.

Limit = 100

We define left-over space for each line as:

Left-over space = Limit – sum of character lengths of all words on the line – number of words -1

We also define left over space from word number i to j as:

Left-over space[i,j] = (Limit – j + i + $\sum X_k$) | k => i to j)

(Each line has a limit of 100 characters, each word occupies the length of each word and an extra character to ensure there is a space between two words. The minus one at the end is because the last word on the line does not need an extra space).

We use memoization technique to find out all the Left-over space for all combinations of i and j. For example we take i to be 1 and then we iterate j from 2 to the length of the text to find out the different Left-Over space values for each combination.

There are three possible values we could get for a combination of i,j:

Left-over(i,j) =

1. Negative (when the length of the characters and the spaces in between them exceed line limit).
2. 0 (for the last line and when sum of character lengths of all words on the line – number of words -1 = 100)
3. Positive (when the words and the space between them are lesser than 100).

Once we calculate the Left-Over value for a combination we save that value in a dictionary. The key value for the dictionary is a tuple that contains (i,j) and its value is the Left-over value.

As we do not need negative values (we cannot print words beyond the word limit). We utilize the following procedure while saving Left-over values to the memoization table:

```
If Left-over<0:
        Break the loop, do not save the value (this helps save space and reduces number of
        computations)
```

```
If Left-over>=0:
        Save the value in the memoization table as (i,j):Left-over
If Left-over>=0 and j = last word in the text:
        Save the value as 0 (We do this as the last line's extra characters do not impact the overall Left-over spaces as this is the end.)
```

We are trying to arrange the words from text in an optimal manner. To do this, we can start with the last line and try and find the best way to arrange the words from i to j. While doing this we also believe that the lines before the last line are also optimally arranged (words 1 to i-1). Thus, if Cost[j] is the cost it takes to arrange words from 1 to j (where cost is the least sum of left-over values over all the lines), we can define Cost[j] recursively as:

```
Cost[j] = {0 if j = 0,
           Min(Cost[i-1] + Left-Over[i,j]) if j>0 | 1<=i<=j}
```

Once we calculate the lowest cost of j, we look at the i for the least cost and save it another dictionary called parent pointers. We take this dictionary to track the last and first words for a line and then print the same. Here:

The parent pointer of j will be i-1 and the parent pointer of i-1 is tracked and so on. Eventually leading to the i=1.

The pseudocode described above is a modified version and is referenced from:

CLRS Solutions[1]

Here is the unprocessed text:

There was more noise than ever over at the house. The main building was called "the house," to distinguish it from the cottages. The chattering and whistling birds were still at it.

Two young girls, the Farival twins, were playing a duet from "Zampa" upon the piano. Madame Lebrun was bustling in and out, giving orders in a high key to a yard-boy whenever she got inside the house, and directions in an equally high voice to a dining-room servant whenever she got outside.

She was a fresh, pretty woman, clad always in white with elbow sleeves.

Her starched skirts crinkled as she came and went. Farther down, before one of the cottages, a lady in black was walking demurely up and down, telling her beads. A good many persons of the pension had gone over to the Cheniere Caminada in Beaudelet's lugger to hear mass. Some young people were out under the wateroaks playing croquet. Mr. Pontellier's two children were there--sturdy little fellows of four and five.

[1] https://walkccc.github.io/CLRS/Chap15/Problems/15-4/

A quadroon nurse followed them about with a faraway, meditative air. Mr. Pontellier finally lit a cigar and began to smoke, letting the paper drag idly from his hand. He fixed his gaze upon a white sunshade that was advancing at snail's pace from the beach.

He could see it plainly between the gaunt trunks of the water-oaks and across the stretch of yellow camomile. The gulf looked far away, melting hazily into the blue of the horizon. The sunshade continued to approach slowly. Beneath its pink-lined shelter were his wife, Mrs. Pontellier, and young Robert Lebrun.

Here is the processed text after running the program:

```
There was more noise than ever over at the house. The main building was called
“the house,” to distinguish it from the cottages. The chattering and whistling birds were still
at it. Two young girls, the Farival twins, were playing a duet from “Zampa” upon the piano.
Madame Lebrun was bustling in and out, giving orders in a high key to a yard-boy whenever she
got inside the house, and directions in an equally high voice to a dining-room servant whenever
she got outside. She was a fresh, pretty woman, clad always in white with elbow sleeves.
Her starched skirts crinkled as she came and went. Farther down, before one of the cottages,
a lady in black was walking demurely up and down, telling her beads. A good many persons
of the pension had gone over to the Cheniere Caminada in Beaudelet's lugger to hear mass.
Some young people were out under the wateroaks playing croquet. Mr. Pontellier's two children
were there--sturdy little fellows of four and five. A quadroon nurse followed them about with
a faraway, meditative air. Mr. Pontellier finally lit a cigar and began to smoke, letting the
paper drag idly from his hand. He fixed his gaze upon a white sunshade that was advancing at
snail's pace from the beach. He could see it plainly between the gaunt trunks of the water-oaks
and across the stretch of yellow camomile. The gulf looked far away, melting hazily into the
blue of the horizon. The sunshade continued to approach slowly. Beneath its pink-lined shelter
were his wife, Mrs. Pontellier, and young Robert Lebrun.
```

Hospital Blood Stock

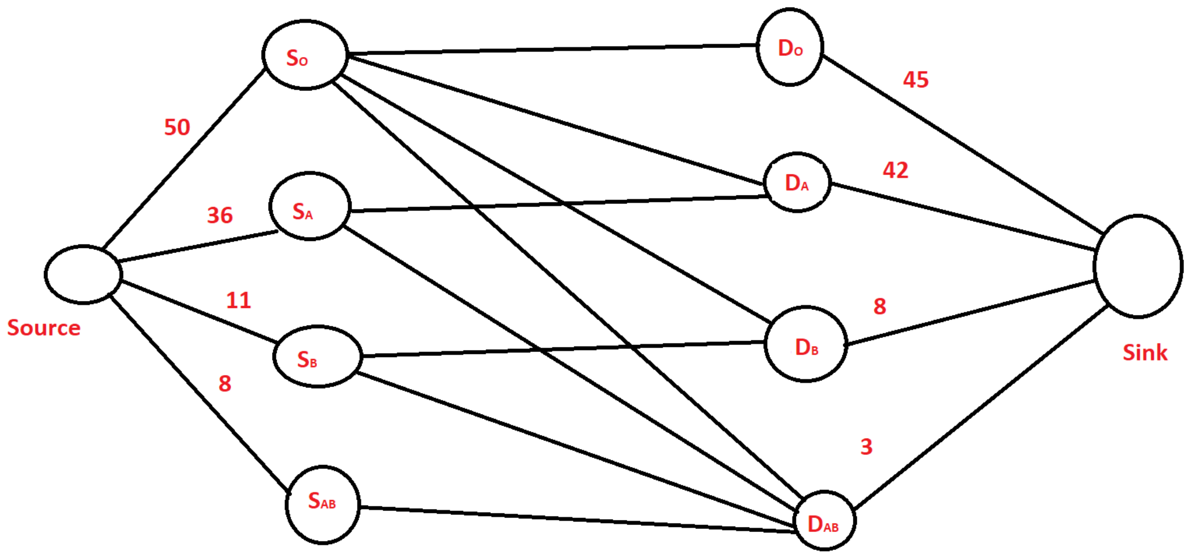
Given the following supply demand table for different blood types. We need to predict if a hospital will be able to meet the demand.

Blood Type	Supply	Demand
O	50	45
A	36	42
B	11	8
AB	8	3

We can solve this problem using the max flow problem analogy. We create a set of nodes that represent the supply and a set of nodes for demand. We then create a dummy source node and a dummy sink node. The flow lines from the Source to the supply set of nodes are the supply amounts given above and the flow lines from the demand set of nodes to the sink is given by the demand amounts above.

We then draw lines linking the blood type that can donate to a blood type that can receive from the blood type.

Here is a graph representing the above table as described:



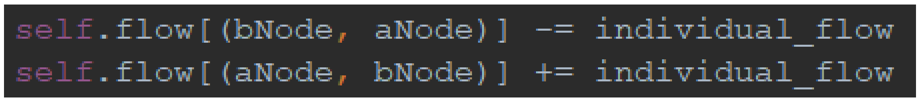
The lines connecting the supply set of nodes and the demand set of nodes are given a value of infinity.

We then apply the Ford-Fulkerson method to calculate the max flow across the graph (I have chosen to use the Edmonds-Karp implementation of the Ford-Fulkerson method, using Breadth First Search to augment flow).

Algorithm

1. Create a graph object and store the edges with the values shown above. Create a separate dictionary to maintain the flow between nodes.

2. Implement Breadth first search to find if there is a path from Source to Sink. However, implement flows along with breadth first search as described below:
    a. Start with the source, add it to a queue.
    b. Maintain a separate dictionary called visited, store the source node as "visited" in the dictionary.
    c. Run a loop till the queue is empty:
        i. Pop the first node from the queue.
        ii. Obtain a list of all its neighbours.
        iii. Add the neighbours to the queue if and only if the flow between node and its neighbor is greater than 0.
        iv. Mark the node as visited.
    d. If there is no more flow possible between the source and sink the sink does not get marked as visited.
    e. We return the status of the sink node (visited or unvisited) and return a dictionary that maintains the path from the source to the sink.
3. Run the max flow algorithm using the above implementation of Breadth First Search:
    a. Repeatedly run the above BFS algorithm. Till the above code returns no path between source and sink.
    b. When a path is returned. Find the edge with the least capacity and then augment the max flow with the same. Change the flow along an edge and back edge as shown:



```
self.flow[(bNode, aNode)] -= individual_flow
self.flow[(aNode, bNode)] += individual_flow
```

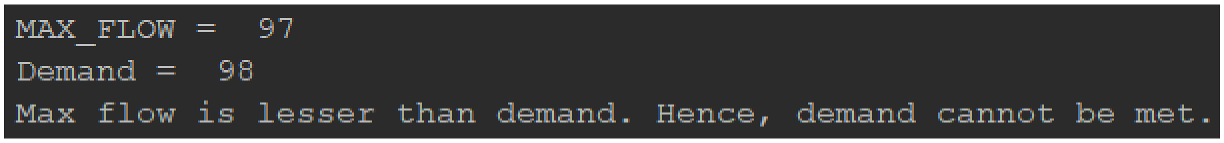
Return the max flow once the above algorithm has run.

The pseudocode for the above algorithm was referenced from:

Wikipedia/Ford-Fulkerson_algorithm[2]

Results

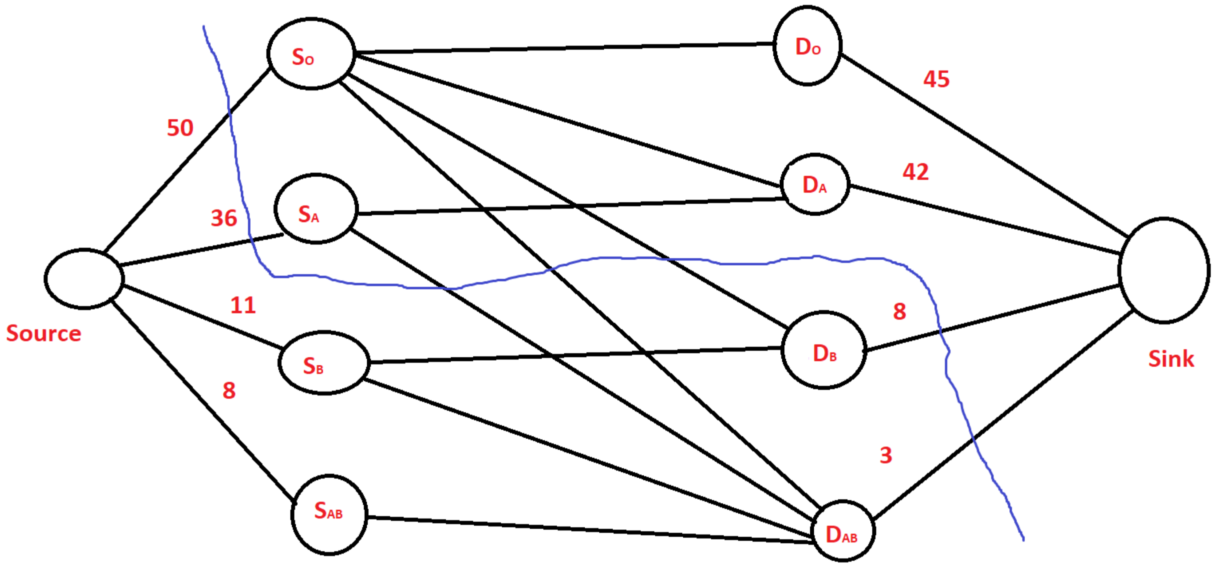
Upon creating the graph and running the above code, we get the max flow as 97:



```
MAX_FLOW =  97
Demand =  98
Max flow is lesser than demand. Hence, demand cannot be met.
```

The minimum cut across the graph is shown below. From the Ford-Fulkerson Method we find that the minimum cut is 97. As max-flow is equal to min cut. The minimum cut across the graph tells us that the most amount of blood that can be serviced across the flow network is 97. Whereas the requirement is 98.

[2] https://en.wikipedia.org/wiki/Ford–Fulkerson_algorithm



From the above diagram we can say that:

The demand for A and O is 45+42 = 87. However, the total supply for O and A is 50+36 = 86. If we increase this supply by 1. We can sustain the demand and supply.