HOMEWORK 5

Name: Tianyu Gu   Collaborator: Angli

Question 1:

In this question, I choose the DFS for searching in the Residual Graph. The data structure to represent the Graph is simply the 2D adjacency matrix. The values of this matrix represent the capacity.

According to my program, there is no perfect match in the given file. However, the size of the largest bipartite matching is 99, quite close to 100. The exact pairs are:

```
69  101
70  102
20  103
58  104
3  105
26  106
43  107
97  108
55  109
6  110
56  111
87  112
33  113
28  114
66  115
25  116
80  117
14  118
92  119
64  120
30  121
67  122
99  123
38  124
74  125
89  126
82  127
37  128
12  129
4  130
49  131
93  132
1  133
60  134
45  135
```

```
76  136
41  137
98  138
19  139
13  140
54  141
32  142
11  143
27  144
5  145
35  146
48  147
18  148
51  149
71  150
96  151
65  152
7  153
42  154
61  155
68  156
44  157
75  158
79  159
34  161
15  162
84  163
9  164
24  165
50  166
46  167
95  168
52  169
47  170
39  171
16  172
10  173
78  174
85  175
8  176
53  177
36  178
2  179
94  180
```

```
91   181
77   182
29   183
72   184
21   185
31   186
57   187
59   188
88   189
40   190
73   191
83   192
86   193
81   194
17   195
62   196
23   197
90   198
63   199
22   200
```

The exact implementation can be found as attachment with name: solution_1. There are some comments in the files, which is used to explain the program.

Question 2:

(a).

Here, we can solve this problem by a categorical solution.

1. First, we will check if type O is sufficient or not, since if type O is not enough, then there is no way that the total supply is sufficient.
2. Secondly, if Type AB is enough and need not other types supply, then we can just check if the extra type O is enough for Type A and Type B.
3. If Type AB is not sufficient, but type O is enough for Type A and Type B, we can check if the total extra Type A, B, O is enough for Type AB.

The Pseudo code is like:

```
/* the main function*/
main()
    ex_o = S_o - D_o        // get the difference
    ex_a = S_a - D_a
    ex_b = S_b - D_b
    ex_ab = S_ab - D_ab
// when type ab is sufficient, we can only consider the type o,a,b
    if (ex_ab > 0)
        return is_satisfied(ex_o,ex_a,ex_b)
// when type ab is not sufficient, we should consider the if type o,a,b are sufficient for the type ab
    else:
 // first, check if type o,a,b is sufficient or not
        if(is_satisfied(ex_o,ex_a,ex_b))
            if(ex_o + ex_a +ex_b +ex_ab < 0)
                return false
            else
                return true
        else
            return false
/* this function used to judge if Type O is sufficient for type a and type b
 */
boolean is_satisfied(ex_o,ex_a,ex_b)
    if(ex_o < 0)
            return false
        else
                if(ex_o + ex_a < 0)
                    return false
                if(ex_o + ex_b < 0)
                    return false
                if(ex_a < 0 and ex_b < 0)
                    if(ex_o + ex_a + ex_b < 0)
                        return false
                return Ture
```

The total running time of this algorithm will be linear to n, where the n represents the numbers of the blood type. However, in this case, the n is always equal to 4, so there are multiple ways for writing the is_satisfied() function. If n can be larger, and let’s suppose the new blood type is similar to type A and B, which can only be accepted by themselves and type AB. Still we can solve this in linear time, basically, you follow the greedy rules: checking if the extra O can cover the insufficiency of type A, then update the amount of extra O and check if it still enough for type B, and so on so forth.

(b).

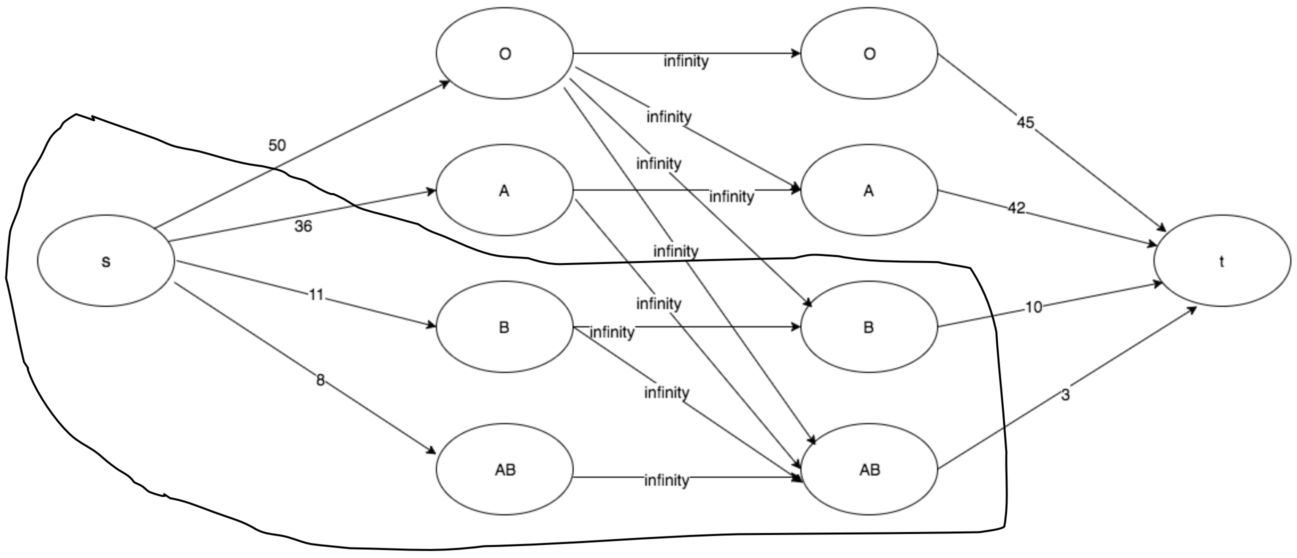
i.

Minimum capacity cut:

This problem can also be solved by maximum flow. Similar to the question 1, we could construct the graph with following steps:

1. List all the supply nodes and demand nodes in two straight lines
2. For all nodes, connect the node m from supply list to n in demand list, if blood type m can be received by blood type n. for example, A can be received by AB, so connect A to AB
3. Set all the edges to infinity.
4. Add two node, source and sink. Then, add edges from the source node to all the nodes in supply list. Set the capacity to the amount of the supply of that type of blood.
5. Similar to source, add edges from the nodes in demand list to the sink node, set the capacity to the amount of demand of that blood type

The graph is shown as below:



We can find that, the above partition with vertex: S, B in supply, AB in supply, B in demand, AB in demand, will give us the minimum cut: 99.

Therefore, the maximum flow in this graph will be 99, which is smaller than the total demand

100.Therefore, the supply is not sufficient.

ii.

The maximum flow on this Graph can be computed with the implementation in question 1, with slightly changes. Since the redundancy is large, the exact implementation can be found as attachment.

As the result, the maximum flow is 99, which is the same as the result of minimum cuts.

iii.

Explanation:

According to the figure above, we can see from the minimum cut that, currently the supply for type AB and type B is sufficient. However, the supply for type O and A is totally 86, which is smaller than the total demand of O and A, which is 45+42 = 87. Therefore, the total blood is not sufficient.

Another explanation will be:

Since type O can only receive type O, therefore, we only have 5unit of type O remain. However, we need 6 unit amount of type A to cover the insufficiency. Meanwhile, since we can only use type O to cover, apparently there is no way to make type A sufficient.

Question 3:

a) With the capacities of vertices in a network flow, we can reconstruct this network to a ordinary flow network with following rules:
   i. With all the vertices, including the sources and sinks, we could divide each vertex, for example V, into two vertices, V and V'.
   ii. For each pair of the new vertices, add a new edge from V to V'. The capacity of this new edge will be the same as the capacity of this node.
   iii. For all the edges $\{u, v\}$, reconnect them into $\{u', v\}$.

By following these rules, we can transfer the vertex capacity into the edge capacity. Since there is at most that much flows passing through that edge, the total flow passing the vertex is bounded by the capacity of this edge, which is identical to the capacity of this vertex.
The new network will have $|V'| = |2V|$ and $|E'| = |E| + |V|$. This is quite comparable to the original network flow.

b) This escape problem can be solved by the flow network with both the vertices and edges capacities set to 1. Therefore, we can further simplify this flow network into an ordinary flow network with only edge capacity, with the rules in part a.
   The exact implementation for this problem will be:
   1) Introduce the source node S and the sink node T. Connect the S to every point in m starting points and connect every point in 4n – 4 boundary points to the T.
   2) For each edges (i, j) in the original grid, we can add two directed edges (i, j) and (j, i) in the network.
   3) Assign the capacity of all the edges to 1, which means it can only be passed through once.

By following the above rules, we can construct the flow network. Since the capacity is one, and there is only one edge between the duplicate vertices for each node in the original grid. The node can only be passed through for one time.
Therefore, we can calculate the maxflow of this network, if it is less than m, there is no escape method for m points. On the other hand, if it is great or equal to m, there is an escape method for m points.
The running time of computing the maxflow will be $O(C * |V|)$. In this case, the maxflow C will be bounded to 4n, and $|V| = O(2n^2)$. Therefore, the total running time will be $O(n^2)$.