**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:

1. 69  101
2. 70  102
3. 20  103
4. 58  104
5. 3  105
6. 26  106
7. 43  107
8. 97  108
9. 55  109
10. 6  110
11. 56  111
12. 87  112
13. 33  113
14. 28  114
15. 66  115
16. 25  116
17. 80  117
18. 14  118
19. 92  119
20. 64  120
21. 30  121
22. 67  122
23. 99  123
24. 38  124
25. 74  125
26. 89  126
27. 82  127
28. 37  128
29. 12  129
30. 4  130
31. 49  131
32. 93  132
33. 1  133
34. 60  134
35. 45  135
36. 76  136
37. 41  137
38. 98  138
39. 19  139
40. 13  140
41. 54  141
42. 32  142
43. 11  143
44. 27  144
45. 5  145
46. 35  146
47. 48  147
48. 18  148
49. 51  149
50. 71  150
51. 96  151
52. 65  152
53. 7  153
54. 42  154
55. 61  155
56. 68  156
57. 44  157
58. 75  158
59. 79  159
60. 34  161
61. 15  162
62. 84  163
63. 9  164
64. 24  165
65. 50  166
66. 46  167
67. 95  168
68. 52  169
69. 47  170
70. 39  171
71. 16  172
72. 10  173
73. 78  174
74. 85  175
75. 8  176
76. 53  177
77. 36  178
78. 2  179
79. 94  180
80. 91  181
81. 77  182
82. 29  183
83. 72  184
84. 21  185
85. 31  186
86. 57  187
87. 59  188
88. 88  189
89. 40  190
90. 73  191
91. 83  192
92. 86  193
93. 81  194
94. 17  195
95. 62  196
96. 23  197
97. 90  198
98. 63  199
99. 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:

1. /\* the main function\*/
2. main()
3. ex\_o = S\_o - D\_o        // get the difference
4. ex\_a = S\_a - D\_a
5. ex\_b = S\_b - D\_b
6. ex\_ab = S\_ab - D\_ab
7. // when type ab is sufficient, we can only consider the type o,a,b
8. **if** (ex\_ab > 0)
9. **return** is\_satisfied(ex\_o,ex\_a,ex\_b)
10. // when type ab is not sufficient, we should consider the if type o,a,b are sufficient for the type ab
11. **else**:
12. // first, check if type o,a,b is sufficient or not
13. **if**(is\_satisfied(ex\_o,ex\_a,ex\_b))
14. **if**(ex\_o + ex\_a +ex\_b +ex\_ab < 0)
15. **return** **false**
16. **else**
17. **return** **true**
18. **else**
19. **return** **false**
20. /\* this function used to judge if Type O is sufficient for type a and type b \*/
21. boolean is\_satisfied(ex\_o,ex\_a,ex\_b)
22. **if**(ex\_o < 0)
23. **return** **false**
24. **else**
25. **if**(ex\_o + ex\_a < 0)
26. **return** **false**
27. **if**(ex\_o + ex\_b < 0)
28. **return** **false**
29. **if**(ex\_a < 0 and ex\_b < 0)
30. **if**(ex\_o + ex\_a + ex\_b < 0)
31. **return** **false**
32. **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).

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:

D:\Downloads\Untitled Diagram.png

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.

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.

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:**

1. With the capacities of vertices in a network flow, we can reconstruct this network to a ordinary flow network with following rules:
   1. With all the vertices, including the sources and sinks, we could divide each vertex, for example V, into two vertices, V and.
   2. For each pair of the new vertices, add a new edge from V to. The capacity of this new edge will be the same as the capacity of this node.
   3. For all the edges {u, v}, reconnect them into {, 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. This is quite comparable to the original network flow.

1. 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. In this case, the maxflow C will be bounded to 4n, and. Therefore, the total running time will be