Assignment 5

Vivek Bansal (Id : 112044493)
CSE 548 : Analysis of Algorithms

Due Date : December 05, 2018

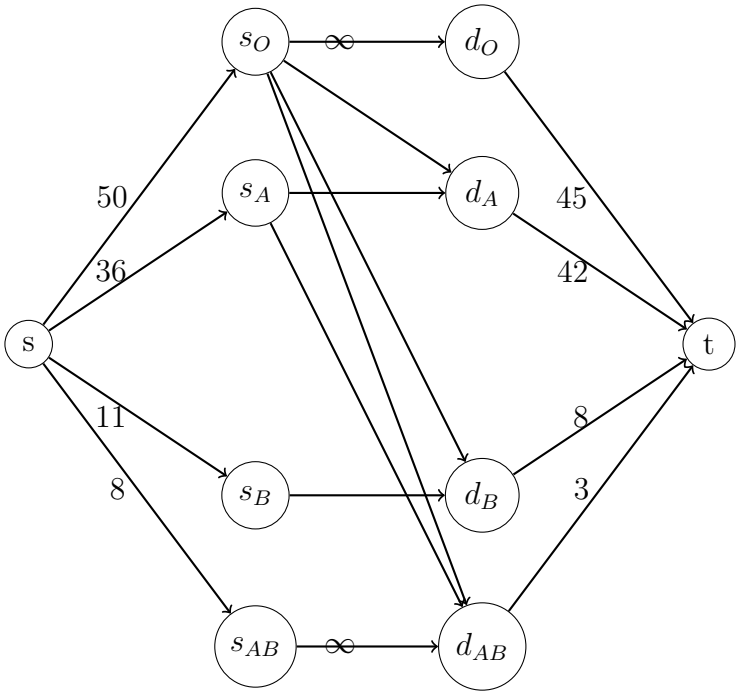
1 Blood Transfusions

a) Algorithm :

1) We create a graph G such that there are 2 nodes s_i and d_j for each blood type = A, AB, A, O. s_i nodes are for the supply nodes and d_j are for the demand nodes.
2) Create a source node s and a sink node t.
3) Create a directed edge from source s to each supply node s_i with capacity s_i and from each demand node d_j to t with capacity d_j.
4) Create a directed edge from s_i to d_j with capacity ∞ if patients of blood type j can receive blood of type i.
5) Run Ford-Fulkerson algorithm on the created graph. If the maximum flow f returned by the algorithm is greater than or equal to the required demand, then the given supply is sufficient to fulfill the demand.
6) Running multiple iterations of Ford-Fulkerson gives us a way of how to allocate the blood to the patients. If flow along edge (s_i, d_j) is f_{ij}, it means to use f_{ij} type-i blood for patients with type-j blood. Our capacity constraints on the edges (s, s_i) ensures that we can't use more than the available supply of type-i blood and capacity constraints on the edges (d_j, t) ensures that we don't allocate more blood to type-j patients than required.

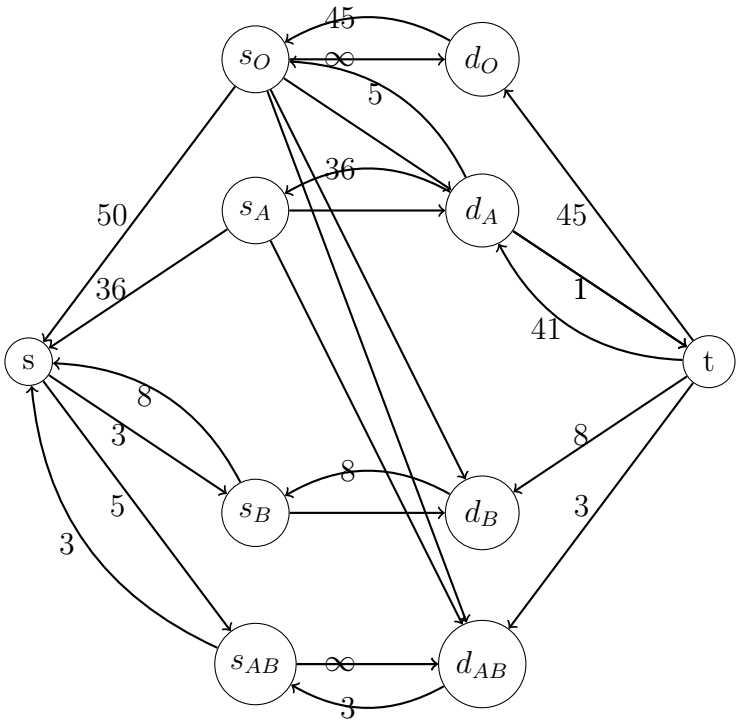
Running time:
The running time of the algorithm is O(Ef) where E are the number of edges in the graph and f is the maximum flow in the graph.

b)



Running Ford-Fulkerson on the above graph, we get :
1) Send flow = 45 along path s, s_O, d_O, t.(type-O blood to type-O patients)
2) Send flow = 36 along path s, s_A, d_A, t.(type-A blood to type-A patients)
3) Send flow = 8 along path s, s_B, d_B, t.(type-B blood to type-B patients)
4) Send flow = 3 along path s, s_{AB}, d_{AB}, t.(type-AB blood to type-AB patients)
5) Send flow = 5 along path s, s_O, d_A, t.(type-O blood to type-A patients)

Residual graph :



As we can see that the max flow from the algorithm is 97 while the total demand is 98. So it will not be possible to meet the full demand. So only 97 patients can be allocated proper blood. Also the minimum value cut will be equal to the max flow, and the cut which has flow 97 is A = [s, s_B, s_{AB}, d_B, d_{AB}] and B = [t, s_O, s_A, d_O, d_A]. The edges crossing the cut are [(s, s_O), (s, s_A), (d_B, t), (d_{AB}, t)].
The reason for not meeting the full demand which can be understood by the clinic administrators is that we see the subset of blood types A and O. The total demand for them is 45 + 42 = 87, but the supply for them is only 86. So we are short by 1 unit.

2 Patients Hospital Problem

Algorithm:

We have n number of patients and k number of hospitals to serve these patients. Also, there are $\lceil \frac{n}{k} \rceil$ patients which can be admitted to each hospital to maintain the balance on each hospital. Now, we have to find out whether all the patients can be served in the hospitals maintaining the balance constraint of each hospital.

1. We will create a graph G = (V,E) where the patients and hospitals are the nodes of the graph.
2. Create 2 nodes s and t, the source and sink nodes respectively.
3. Connect the source node s with all the patients nodes in the graph. Give the edge weight of 1 unit to these edges.
4. Connect the hospitals nodes with the sink node t in the graph. Give the edge weight of $\lceil \frac{n}{k} \rceil$ to these edges since each hospital can serve only $\lceil \frac{n}{k} \rceil$ number of patients to maintain the balance constraint.
5. Connect the patients nodes with all those hospitals nodes which are within 30 mins distance from the patients in the graph G. Give the edge weight of 1 unit to these edges since 1 patient can be served by only 1 hospital.
6. Now, calculate the maximum flow in the graph by using Edmond-Karp's algorithm. If the maximum flow value comes out to be n, then all the patients can be served in the hospitals following the balance constraint of the hospitals. If the flow value is less than n, then it will not be possible to serve all patients.

Running time:

Construction of the graph takes O(nk) time. Running Edmond-Karp algorithm takes O(E^2.V) time where E are the number of edges in graph and V are the number of vertices in graph. For our problem, we have E = nk and V = (n + k), thus overall complexity is O(n^3k^2).

3 Network Attack Problem

Algorithm :

1. Since the attacker attacks on the min-cut whose value is k and remove only those edges, we are sure that these paths which we got after applying Ford-Fulkerson are disjoint paths P_1, P_2, P_3,..., P_k. We can find these paths after applying Ford-Fulkerson. These disjoint paths don't share an arc in common because there will be no s-t paths after removing the edges from these paths.
2. Now, the attacker has removed exactly k edges each of them will lie on the above paths. The length of paths are n.
3. For each path P_i, apply binary search for the vertices on the path. Get the middle vertex lying on path P_i. Ping for that vertex and check if this is reachable or not. If this is reachable, we apply binary search on right, if not reachable we apply binary search on the left of the path. So, we will be able to find first and last nodes in the path which are not reachable using binary search algorithm.
4. In this way, we will be able to find all unreachable nodes in logn pings for each path. But since we have k paths, we are doing kLogn pings in total.

4 Bipartite Matching Problem

a) Approach :

To solve this problem, we will create a maximum flow from the given coverage expansion. The given bipartite graph is divided into 2 sets X and Y where X belongs to set of vertices at one end and Y belongs to set of vertices at other end. The solution steps are mentioned below:

1) Create a source node s and a sink node t.
2) Add directed edges from s to each node in X.
3) Add directed edges from each node in Y to t.
4) Add all the edges from X to Y.
5) Set the capacity of all the edges given in matching M to 0 and all other edges to 1.

Algorithm : Coverage-Expansion(G, M, K)

1) if K $<$ 0
    return false
2) if K == 0
    return M
3) G' = Create a network flow from G as mentioned in the approach above.
4) f = Ford-Fulkerson(G')
5) if |f| $\geq$ |M| + K
    M' = disjoint edges such that edges are of the form e = (x, y) where x $\in$ X and y $\in$ Y.
    M' = M $\cup$ M'
    return M'
6) else
    return false

We have to prove that |f| ≥ |M| + K if there are atleast K edges which are disjoint between X and Y and are also disjoint to M. Let's say we assume that |f| ≥ |M| + K is true. Since all edges have the capacity of 1 excluding the edges from M, there must be atleast |f| disjoint edges from s to t and there should be atleast K disjoint edges from X to Y. These K disjoint edges are also disjoint to the edges in M. From these edges of M, we can get a flow of |M| + K which satisfies |f| ≥ |M| + K.

Running time:
The running time of the algorithm is $O(|E|.n)$ where E are the number of edges in network flow G' and n are the number of iterations.

b)

In the above bipartite graph we have X = [v1, v2] and Y = [v3, v4]. Also we have edges as E = [(v1, v3), (v2, v4)].

Let K = 1 and matching M is [(v2, v3)] as marked in red.
We have |M| = 1 which implies |M'| = |M| + k = 2.

Let's say we have M' = [(v1, v3), (v2, v4)] as marked in blue.
Thus, as we can see M' has more edges than M and every node covered in M is also covered in M'. Also the edges in M which are [(v2, v3)] are not the subset of edges of M' which are [(v1, v3), (v2, v4)].

c) Input: We have a bipartite graph G with a given matching M. Also, we have matching M' which covers all the nodes of matching M. The maximum size of largest matching in M' is k1. We have matching M" which considers all possible matchings of graph G. The maximum size of largest matching in M" is k2. Now, we have to prove that k1 = k2.

Proof by Contradiction: Let M1 be a matching in M' which is of largest size, i.e. k1. If M1 is not a maximum matching in G, there must exist an M1-augmented path P. Thus M1ΔP is a matching with one more edge and should covered all the vertices of M1. So, it must contain all the vertices covered by M and therefore M1ΔP is in M' as well. This contradicts our choice of M1 and it should be the maximum matching in graph G. Thus its length k1 should be equal to the length of maximum matching in graph G which is k2. So, we have k1 = k2.

5 Rearrangeable Matrix problem

a) Below is an example of a matrix which is not rearrangeable and atleast one of its entry in each row and column is 1. The matrix is not rearrangeable since there doesn't exist any way to swap its

rows and columns to make all entries in a diagonal equal to 1.

$$M = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

b) Algorithm:
The matrix is rearrangeable if there exists a permutation in the matrix which will make every entry of the diagonal equal to 1. To find this permutation we have to check whether there exists a perfect matching in the bipartite graph. So, we will create a bipartite graph with vertex for each row as one set and vertex for each column as another set and there exists an edge between vertices i and j (vertex i belong to row set and vertex j belong to column set) if M[i][j] = 1. We will use Edmond-Karp algorithm to find the maximum flow in this bipartite graph and if this maximum flow is a perfect matching (if value of flow is n) in this graph, then the matrix is rearrangeable.

Running time:
The running time of the algorithm is O(n^3) since Edmond-Karp algorithm takes O($E^2.V$) time. We have E = n^2 since we have n^2 entries in the matrix of size n x n. Also we have V = n since we have vertices corresponding to number of rows or number of columns in our bipartite graph.