Algorithms: COMP3121/9101

Aleks Ignjatović, ignjat@cse.unsw.edu.au
office: 504 (CSE building K 17)
Admin: Song Fang, cs3121@cse.unsw.edu.au

School of Computer Science and Engineering
University of New South Wales Sydney

Flow Networks Practice Problems

Several families are coming to a birthday celebration in a restaurant. You have arranged that v many tables will serve only vegetarian dishes, p many tables will serve dishes with meat but without pork and r many remaining tables will serve food with pork. You know that V many families are all vegetarians, P_1 many families do not eat pork but do not mind eating vegetarian dishes, P_2 many families do not eat pork but hate vegetarian dishes. Also R_1 many families have no dietary restrictions and would also not mind eating vegetarian dishes or food without pork, R_2 many families have no dietary restrictions but hate vegetarian dishes but can eat food without pork. Finally, S many families are from Serbia and cannot imagine not eating pork. You are also given the number of family members in each family and the number of seats at each table. In total, there are m families and n tables. You must place the guests at the tables so that their food preferences are respected and no two members from the same family sit at the same table. Your algorithm must run in time polynomial in m and n, and in case the problem has no solutions, your algorithm should output no solution.

We begin by constructing a bipartite flow network as follows:

- the left hand side n vertices represent n families,
- the right hand side m vertices represent m tables,
- a source s and a sink t,
- connect s to each family vertex with an edge of capacity equal to the number of family members,
- connect each table vertex to t with an edge of capacity equal to the number of seats at that table,
- connect each family vertex with all tables compatible with the dietary preference with an edge of capacity 1

From this flow network construction, we run Edmonds-Karp to find the maximum flow. If the maximum flow is less than the total number of guests, then we output no solution. Otherwise, if the maximum flow equals the total number of guests, then a placement is possible. Further, we can deduce a placement of guests at tables by examining which of the (family,table) edges carry flow: if there is 1 unit of flow from family i to table j then we seat a member of family i at table j. The time complexity is $O(|V||E|^2)$ where $|V| = m + n + 2$ and $|E| \leq m + n + mn$, so algorithm runs in time polynomial in m and n.

You have been told of the wonder and beauty of a very famous painting. It is painted in the hyper-modern style, and so it is simply an $n \times n$ grid of squares, with each square coloured either black or white. You have never seen this picture for yourself but have been told some details of it by a friend. Your friend has told you the value of n and the number of white squares in each row and each column. Additionally, your friend has also been kind enough to tell you the specific colour of some squares: some squares are black, some are white, and the rest they simply could not remember.
The more details they tell you, the more amazing this painting becomes but you begin to wonder that perhaps its simply too good to be true. Thus, you wish to design an algorithm which runs in time polynomial in n and determines whether or not such a painting can exist.

Solution: This problem can be viewed as a (bipartite) network flow problem in disguise. We begin by adjusting the count of the number of unknown white squares in each row and column based on the location of the known (white) squares. Note. The sum of the count in all rows must equal the sum of the count in all columns. Let this sum be S so we can infer that there are precisely S white squares among the squares of unknown colour. We then consider the bipartite graph where every row is a vertex on the left side of the graph and every column is a vertex on the right side, making $2n$ vertices in total. Every square in the grid which is of unknown colour forms a directed edge from its corresponding row to its corresponding column. We can then convert this graph into a flow network as follows:

- each edge has capacity of 1.
- we add a source s and sink t to this graph.
- we add a directed edge from s to each row with capacity equal to the adjusted number of white squares in that row
- we add a directed edge from each column to t with capacity equal to the adjusted number of white squares in that column

It is evident that the saturated edges of the bipartite graph in any integer-valued flow from the source to the sink describe a possible colouring of the grid.

Therefore we can now find the maximum flow f via the Edmonds-Karp algorithm and as any such flow has a capacity at most S, a painting exists if and only if $f = S$.

With $2n$ vertices in total, $2n$ edges for s and t and n^2 edges for the bipartite flow graph, we can conclude that our solution runs in a time complexity of $O(n^5)$, which is clearly polynomial in n.

Alice is the manager of a cafe which supplies n different kinds of drink and m different kinds of dessert. One day the materials are in short supply, so she can only make a_i cups of each drink type i and b_j servings of each dessert type j. On this day, k customers come to the cafe and the i^{th} of them has p_i favourite drinks $(c_{i,1}, c_{i,2}, \ldots, c_{i,p_i})$ and q_i favourite desserts $(d_{i,1}, d_{i,2}, \ldots, d_{i,q_i})$. Each customer wants to order one cup of any one of their favourite drinks and one serving of any one of their favourite desserts. If all their favourite drinks or all their favourite desserts are unavailable, the customer will instead leave the cafe and provide a poor rating. Alice wants to save the restaurants rating. From her extensive experience with these k customers, she has listed out the favourite drinks and desserts of each customer, and she wants your help to decide which customers orders should be fulfilled. Design an algorithm which runs in time polynomial in n, m and k and determines the smallest possible number of poor ratings that Alice can receive, given that:

(a) all p_i and all q_i are 1 (i.e. each customer has only one favourite drink and one favourite dessert),

(b) there is no restriction on the p_i and q_i.

Solution: (a) Construct a flow graph with a vertex A_i for each drink, a vertex B_j for each dessert, then two extra vertices for a source S and a sink T. For each drink i, add an edge with capacity a_i from S to A_i. For each dessert j, add an edge with capacity b_j from B_j to T. Finally, for each customer, add an edge of capacity 1 from $c_{i,1}$ to di,1. The answer is k minus the maximum flow, found using Edmonds-Karp. The time complexity is $O(|V||E|^2)$, where $|V| = 2 + n + m$ and $|E| = n + m + k$, so it is polynomial in n, m and k.

Solution: (b) We start by constructing a flow graph with:

- Two vertices, S and T, for the source and the sink.
- A vertex A_i for each drink i, with an edge of capacity a_i from S to A_i, to restrict the number of available cups of this drink.
- A vertex B_j for each dessert j, with an edge of capacity b_j from B_j to T, to restrict the number of available servings of this dessert.
- Two vertices C_i and D_i for each customer, with an edge of capacity 1 from C_i to D_i to ensure that each customer either has both their drink and dessert, or has neither. Note that we ignore serving them only one, as that is equivalent to serving them nothing in terms of ratings.
- For each favourite drink $c_{i,j}$ of customer i, an edge of capacity 1 from $A_{c_{i,j}}$ to C_i for any drink they would accept.
- For each favourite dessert $d_{i,j}$ of customer i, an edge of capacity 1 from D_i to $B_{c_{i,j}}$ for any dessert they would accept.

Each unit of flow through this graph assigns a different customer one of their favourite drinks and one of their favourite desserts.

Running the Edmonds-Karp algorithm on this graph then gives us the maximum flow, i.e. the maximum number of customers that we can satisfy, and so k minus this value is the minimum number of poor ratings. Our flow graph has $|V| = 2 + n + m + 2k$ vertices and $|E| \leq n + m + k + (n + m)k$ edges, so the time complexity of $O(|V|\,|E|^2)$ is indeed polynomial in n, m and k.

There are n cities (labelled $1, 2, \ldots, n$), connected by m bidirectional roads. Each road connects two different cities. A pair of cities may be connected by multiple roads. A well-known criminal is currently in city 1 and wishes to get to the city n via road. To catch them, the police have decided to block the minimum number of roads possible to make it impossible to get from city 1 to city n. However, some roads are major roads. In order to avoid disruption, the police cannot close any major roads. Your goal is to find the minimum number of roads to block to prevent the criminal from going from city 1 to city n, or report that the police cannot stop the criminal. Design an algorithm which achieves this goal and runs in time polynomial in n and m.

Solution: We construct a flow network as follows:

- create vertices $v_1, v_2, \ldots, v_n$ corresponding to n cities;
- make v_1 as the source and v_n as the sink;
- suppose there are k roads between two cities i and j.
  - if none of the roads is a major road, we connect v_i and v_j with two directed edges in opposite directions and of capacity each equal to k.
  - if one of the roads is a major road the capacity of the two directed edges is set to $m + 1$ (m is the total number of roads).
- We can now find the maximum flow f in such a network using the Edmonds-Karp algorithm, and recall that the value of this flow equals the capacity of the min cut.
- If $|f| > m$ then at least one edge of capacity $m + 1$ has been crossed, which indicates that every cut is crossed by a major road which cannot be blocked and thus we cannot catch the criminal.
- On the other hand, if $|f| \leq m$, then there is a cut which is crossed only by minor roads, so the criminal can be caught.
- To block the fewest number of roads, we block those roads which cross the min cut in the forward direction, i.e. those which go from a vertex reachable from v_1 in the final residual graph to a vertex without this property.

Assume that you are given a network flow graph with n vertices, including a source s, a sink t and two other distinct vertices u and v, and m edges. Design an algorithm which runs in time polynomial in n and m and returns the smallest capacity-cut among all cuts for which:

(a) the vertex u is on the same side of the cut as the source s and vertex v is on the same side as the sink t.

(b) vertices u and v are at the same side of the cut.

Solution:

(a) Add to the flow network one edge of infinite capacity from s to u and one edge of infinite capacity from v to t. Find the min cut in such a network. Run Edmonds-Karp to find the maximum flow and its corresponding minimal cut. Time complexity is $O(nm^2)$.

(b) Given the flow network, we construct two directed edges, one from u to v and the other from v to u, both of which having infinite edge capacities respectively. We note that, if only one infinite edge is constructed from u to v, then v can still end up on the source side and u can still end up on the sink side, so the edge will not belong in the cut. From the discussion above, the two directed edges will ensure that u and v remain in the same side of the cut.

Assume that you are given a network flow graph with n vertices, including a source s, a sink t and two other distinct vertices u and v, and m edges. Design an algorithm which runs in time polynomial in n and m and returns the smallest capacity-cut among all cuts for which:

(a) the vertex u is on the same side of the cut as the source s and vertex v is on the same side as the sink t.

(b) vertices u and v are at the same side of the cut.

Solution:

(a) Add to the flow network one edge of infinite capacity from s to u and one edge of infinite capacity from v to t. Find the min cut in such a network. Run Edmonds-Karp to find the maximum flow and its corresponding minimal cut. Time complexity is $O(nm^2)$.

(b) Given the flow network, we construct two directed edges, one from u to v and the other from v to u, both of which having infinite edge capacities respectively. We note that, if only one infinite edge is constructed from u to v, then v can still end up on the source side and u can still end up on the sink side, so the edge will not belong in the cut. From the discussion above, the two directed edges will ensure that u and v remain in the same side of the cut.

Assume that you are given a network flow graph with n vertices, including a source s, a sink t and two other distinct vertices u and v, and m edges. Design an algorithm which runs in time polynomial in n and m and returns the smallest capacity-cut among all cuts for which:

(a) the vertex u is on the same side of the cut as the source s and vertex v is on the same side as the sink t.

(b) vertices u and v are at the same side of the cut.

Solution:

(a) Add to the flow network one edge of infinite capacity from s to u and one edge of infinite capacity from v to t. Find the min cut in such a network. Run Edmonds-Karp to find the maximum flow and its corresponding minimal cut. Time complexity is $O(nm^2)$.

(b) Given the flow network, we construct two directed edges, one from u to v and the other from v to u, both of which having infinite edge capacities respectively. We note that, if only one infinite edge is constructed from u to v, then v can still end up on the source side and u can still end up on the sink side, so the edge will not belong in the cut. From the discussion above, the two directed edges will ensure that u and v remain in the same side of the cut.

You know that $n+2$ spies $S, s_1, s_2, ..., s_n$ and T are communicating through m communication channels; in fact, for each i and each j you know if there is a channel through which spy s_i can send a secret message to spy s_j or if there is no such a channel (i.e., you know what the graph with spies as vertices and communication channels as edges looks like). Design an algorithm which runs in time polynomial in n and m that prevents spy S from sending a message to spy T by:

(a) compromising as few channels as possible;

(b) bribing as few of the other spies as possible.

Solution:

(a) We construct a flow network with each vertex v_i representing each spy i and with edges of capacity 1 from v_i to v_j if there is a communication link from i to j. We then run Edmonds-Karp on the flow network to find the maximum flow and the corresponding minimum cut. As any cut of the graph represents a valid set of channels (or edges) for our problem, then the edges that cross the minimum cut forwards are the ones that have to be compromised. This runs in polynomial time.

(b) We replicate the graph of part (a), but change all capacities of the edges to ∞. Then for each non-source and non-sink vertex v_i, we split v_i into two vertices v_i^{in} and and v_i^{out} and connect v_i^{in} and v_i^{out} with an edge of capacity 1. For each edge incoming to v_i, we connect it to v_i^{in}; for each edge outgoing from v_i, we modify it to instead be outgoing from v_i^{out} i . Using this flow network graph, we repeat the same process as part (a) by running Edmonds-Karp and computing the minimum cut to find all the edges that represent the corresponding spies to bribe. As we have $E = m + n$ and $f \leq n$, our total complexity is slightly different but still polynomial in m and n with $O(E|f|) = n(m + n)$.

You are manufacturing integrated circuits from a rectangular silicon board that is divided into an $m \times n$ grid of squares. Each integrated circuit requires two adjacent squares, either vertically or horizontally, that are cut out from this board. However, some squares in the silicon board are defective and cannot be used for integrated circuits. For each pair of coordinates (i, j), you are given a boolean $d_{i,j}$ representing whether the square in row i and column j is defective or not. Design an algorithm which runs in time polynomial in m and n and determines the maximum number of integrated circuits that can be cut out from this board.

Solution: Form a bipartite graph with the left side consisting of all cells (x, y) which are non-defective and which satisfy that $x + y$ is even and the right side consisting of all non-defective cells which satisfy that $x + y$ is odd. Connect each cell with (at most four) adjacent non-defective cells. Now the problem reduces to finding a maximal matching in this graph which ensures that each cell is used in at most one IC.

You are manufacturing integrated circuits from a rectangular silicon board that is divided into an $m \times n$ grid of squares. Each integrated circuit requires two adjacent squares, either vertically or horizontally, that are cut out from this board. However, some squares in the silicon board are defective and cannot be used for integrated circuits. For each pair of coordinates (i, j), you are given a boolean $d_{i,j}$ representing whether the square in row i and column j is defective or not. Design an algorithm which runs in time polynomial in m and n and determines the maximum number of integrated circuits that can be cut out from this board.

Solution: Form a bipartite graph with the left side consisting of all cells (x, y) which are non-defective and which satisfy that $x + y$ is even and the right side consisting of all non-defective cells which satisfy that $x + y$ is odd. Connect each cell with (at most four) adjacent non-defective cells. Now the problem reduces to finding a maximal matching in this graph which ensures that each cell is used in at most one IC.

You are given an $n \times n$ chessboard with k white bishops on the board at the given cells (a_i, b_i), where $1 \leq a_i, b_i \leq n$ for each $1 \leq i \leq k$. You have to determine the largest number of black rooks which you can place on the board so that no two rooks are in the same row or in the same column or are under the attack of any of the k white bishops. Recall that bishops attack diagonally.

Solution: We construct a bipartite graph with n left vertices r_i representing n rows of the board and n right vertices c_j representing n columns of the board. We construct edges in such a graph so that vertex r_i is connected with a vertex r_j just in case the cell (i, j) on the board is not under attack of any of the bishops. We add a super source s and connect it with all vertices r_i with edges of capacity 1; we also add a super sink and connect all vertices c_j also with edges of capacity 1. The maximal number of rooks that meet the conditions is equal to the max flow in this flow network, with rooks placed in the cells corresponding to the occupied edges from r_i to c_j .

You are given an $n \times n$ chessboard with k white bishops on the board at the given cells (a_i, b_i), where $1 \leq a_i, b_i \leq n$ for each $1 \leq i \leq k$. You have to determine the largest number of black rooks which you can place on the board so that no two rooks are in the same row or in the same column or are under the attack of any of the k white bishops. Recall that bishops attack diagonally.

Solution: We construct a bipartite graph with n left vertices r_i representing n rows of the board and n right vertices c_j representing n columns of the board. We construct edges in such a graph so that vertex r_i is connected with a vertex r_j just in case the cell (i, j) on the board is not under attack of any of the bishops. We add a super source s and connect it with all vertices r_i with edges of capacity 1; we also add a super sink and connect all vertices c_j also with edges of capacity 1. The maximal number of rooks that meet the conditions is equal to the max flow in this flow network, with rooks placed in the cells corresponding to the occupied edges from r_i to c_j .

You have n warehouses and n shops. At each warehouse, a truck is loaded with enough goods to supply one shop. There are m roads, each going from a warehouse to a shop, and driving along the i-th road takes d_i hours, where d_i is an integer. Design a polynomial time algorithm to send the trucks to the shops, minimising the time until all shops are supplied.

Solution: We combine a binary search with a maximum matching problem. We sort the values d_i in an increasing order. Represent each of the warehouse with a vertex and each shop also with a vertex. For any value of d_i we can connect all warehouses with all shops which are at most d_i away from each other, and then use a max-flow algorithm to determine if such a graph has a perfect matching of size n. Use binary search to find the smallest d_i for which such a perfect matching exists.

You have n warehouses and n shops. At each warehouse, a truck is loaded with enough goods to supply one shop. There are m roads, each going from a warehouse to a shop, and driving along the i-th road takes d_i hours, where d_i is an integer. Design a polynomial time algorithm to send the trucks to the shops, minimising the time until all shops are supplied.

Solution: We combine a binary search with a maximum matching problem. We sort the values d_i in an increasing order. Represent each of the warehouse with a vertex and each shop also with a vertex. For any value of d_i we can connect all warehouses with all shops which are at most d_i away from each other, and then use a max-flow algorithm to determine if such a graph has a perfect matching of size n. Use binary search to find the smallest d_i for which such a perfect matching exists.