COMP3121/9101
Algorithm Design

Problem Set 4 – Maximum Flow

[**K**] – key questions [**H**] – harder questions [**E**] – extended questions [**X**] – beyond the scope of this course

Contents

§ SECTION ONE: MAXIMUM FLOW

[K] Exercise 1. 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 not serve 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".

Solution. We begin by constructing a bipartite flow network as follows:

- the left hand side vertices represent families,
- the right hand side vertices represent tables,
- source s and sink t,
- connect s to each family vertex with capacity equal to the number of family members,
- connect each table vertex to t with capacity equal to the number of seats at that table,
- connect each family vertex with all tables compatible with the dietary preference with capacity 1.
  - The V vegetarian families are only connected to the v vegetarian tables.
  - The P_1 non-pork families are connected to the v vegetarian tables and the p non-pork tables.
  - The P_2 non-pork non-veg families are connected to the p non-pork tables only.
  - The R_1 families who eat anything are connected to all tables.
  - The R_2 non-veg families are connected to the p non-pork tables and the r pork tables.
  - The S Serbian tables are connected to the r pork tables only.
  - The edge capacity of 1 ensures that no two members of the same family go to the same table.

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. Applying *flow conservation* at each table ensures that the table capacities are respected.

The time complexity is $O(|V||E|^2)$ where $V = m+n+2$ and $E \leq m+n+mn$, so the algorithm runs in time polynomial in m and n as required. □

[K] Exercise 2. A band of m criminals has infiltrated a secure building, which is structured as an $n \times n$ square grid of rooms, each of which has a door on all of its sides. Thus,

- from an internal room, we can move to any of the four neighbouring rooms

- from a room on the side of the building (or edge room), we can move to three other rooms or leave the building
- from a corner room, we can move to two other rooms or leave the building

The criminals were able to shut down the building's security system before entering, but during their nefarious activities, the security system became operational again, so they decided to abort the mission and attempt to escape. The building has a sensor in each room, which becomes active when an intruder is detected, but only triggers the alarm if it is activated again. Thus, the criminals may be able to escape if they can all reach the outside of the building without any two of them passing through the same room.

Design an algorithm which runs in time polynomial in m and n and, given the m different rooms which the criminals occupy when the security system is reactivated, determines whether all m criminals can escape without triggering the alarm.

Solution. We start by creating the flow graph as follows:

- for each room, say with coordinates (i, j):
  - construct a pair of vertices $v_{i,j}^{\text{in}}$ and $v_{i,j}^{\text{out}}$, and
  - place an edge of capacity 1 from $v_{i,j}^{\text{in}}$ to $v_{i,j}^{\text{out}}$, representing a 'vertex capacity' of 1 for room (i, j).
- for each pair of neighbouring rooms (i, j) and (i', j'), place edges of capacity 1 from $v_{i,j}^{\text{out}}$ to $v_{i',j'}^{\text{in}}$ and from $v_{i',j'}^{\text{out}}$ to $v_{i,j}^{\text{in}}$.
- add a super source s, and place an edge of capacity 1 from s to the in-vertex of each of the m initially occupied rooms.
- add a super sink t, and place an edge of capacity 1 from the out-vertex of each edge room to t.

Now we can find maximum flow f in such a network via Edmonds-Karp algorithm. Then the problem has a solution if $f = m$. Note that our approach is optimal such that every vertex and every edge can belong to only one path, which corresponds to unit flows in our constructed flow graph. For the constraint that all paths must be disjoint, our augmentation of the vertices will ensure that each square can only be occupied by 1 path hence all such paths we consider will be disjoint. Therefore, the maximum flow will indicate the maximum number of paths that satisfies the constraint and hence if $f = m$ indicates if such a problem is possible.

In our flow graph $V = 2n^2 + 2$, and we will have:

- n^2 edges from an in-vertex to an out-vertex,
- $4n(n-1)$ edges between adjacent rooms,
- m edges from s and
- $4(n-1)$ edges to t.

Then the overall complexity of our solution is

$$O\left(2(n^2+1) \times (n^2 + 4n(n-1) + m + 4(n-1))^2\right) = O(n^6 m^2),$$

which is polynomial in m and n. □

[H] Exercise 3. 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 it's 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 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. ☐

[H] Exercise 4. Alice is the manager of a café 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 café and the ith 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 Alice refuses to serve them, or if all their favourite drinks or all their favourite desserts are unavailable, the customer will instead leave the café and provide a poor rating.

Alice wants to save the restaurant's 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:

- all p_i and all q_i are 1 (i.e. each customer has only one favourite drink and one favourite dessert),
- 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 $d_{i,1}$. The answer is k minus the maximum flow, found using Edmonds-Karp. The time complexity is $O(VE^2)$, where $V = 2 + n + m$ and $E = n + m + k$, so it is polynomial in n, m and k.

(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 $E = n + m + k + \sum_{i=1}^{k}(p_i + q_i) \leq n + m + k + (n + m)k$ edges, so the time complexity of $O(VE^2)$ is indeed polynomial in n, m and k.

$\square$

§ SECTION TWO: MINIMUM CUT

[K] Exercise 5. 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 cities as vertices $v_1, v_2, \ldots, v_n$
- 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 $m + 1$.

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.

With a total of n nodes and a maximum of m edges connecting each of possible pairs of cities, the complexity of our algorithm is then $O(nm^2)$. □

[K] Exercise 6. In the country of Pipelistan there are several oil wells, several oil refineries and many distribution hubs all connected by oil pipelines. To visualise Pipelistan's oil infrastructure, just imagine a undirected graph with k source vertices (the oil wells), m sinks (refineries) and n vertices which are distribution hubs linking (unidirectional) pipelines incoming to this vertex with the outgoing pipelines from that vertex.

You are given the graph and the capacity $C(i, j)$ of each pipeline joining a vertex i with vertex j. You want to install the smallest possible number of flow meters on some of these pipelines so that the total throughput of oil from all the wells to all refineries can be computed exactly from the readings of all of these meters. Each meter shows the direction of the flow and the quantity of flow per minute. Design an algorithm which runs in time polynomial in k, m and n and decides on which pipelines to place the flow meters.

Solution. We construct a flow network as follows:

- source s and sink t,
- a vertex w_i for each oil well;
  - for every i, we connect s to w_i of infinite edge capacity.

- a vertex r_j for each oil refinery;
  - for every j, we connect r_j to t of infinite edge capacity.
- a vertex d_ℓ for each distribution hub,
- each pipeline is represented by two directed edges of opposite direction, which represents bidirectional edges, of edge capacities 1 each.

Note. Note that we've ignored the *actual* capacity of each pipeline.

Once the flow network is constructed, we run Edmonds-Karp to find the maximal flow. We finally construct the last residual network flow, and look at all vertices to which there is a path from the source s. This defines a minimum cut, so we look at all edges crossing the minimum cut. The number of such edges (in the forward direction only!) determines the minimal number of metres needed to accurately compute the total flow from s to t.

The time complexity of our algorithm is given by the time complexity of Edmonds-Karp, which is given by $O(V \cdot E^2)$. The number of vertices given in our network is maximally given by $k+m+n+2$ (k oil wells, m refineries, n distribution hubs, as well as the source and sink). Maximally, we have k edges from the source to w_i, m edges from r_j to the sink, and $((k+m+n)(k+m+n-1))/2$ pipelines. To see why, note that we have $k+m+n$ vertices which excludes the source and sink. We can have edges between refineries, distribution hubs, or oil wells. Hence, the maximal number of edges is given by $\binom{k+m+n}{2}$. This, the overall time complexity is given by $O(V \cdot E^2)$, where $V = k+m+n+2$ and $E = (k+m+n)^2/2 - (k+m+n)/2$. □

[K] Exercise 7. 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 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.

Solution. Take the given flow graph, we construct via

- create the source s connect the vertex u with an edge of ∞ capacity
- create the sink t and connect vertex v with an edge of ∞ capacity

We then use Edmonds-Karp algorithm to find the maximum flow through such a network and then the corresponding minimal cut. The two added edges of infinite capacity cannot belong to the min-cut which ensures that u stays at the same side as s and v at the same side as t.

The total complexity of our algorithm is $O(nm^2)$. □

[K] Exercise 8. 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 returns a smallest capacity cut among all cuts for which vertices u and v are in the same side of the cut.

Solution. 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. Once the flow network is constructed, we then run Edmonds-Karp to find the maximum flow and its corresponding minimal cut. From the discussion above, the two directed edges will ensure that u and v remain in the same side of the cut. The time complexity is again $O(nm^2)$. □

[K] Exercise 9. Given an undirected graph with vertices numbered $1, 2, \ldots, n$ and m edges, design an algorithm which runs in time polynomial in n and m and partitions the vertices into two disjoint subsets such that:

- vertex 1 and n are in different subsets, and
- the number of edges with both ends in the same subset is maximised.

Solution. We take the given undirected graph and construct a flow network by

- designating vertex 1 as the source and vertex n as the sink, and
- replacing every undirected edge with 2 directed edges of capacity 1.

Our problem is now equivalent to minimising the number of edges between the two subsets. Note that all the original undirected edges appear in pairs of edges with the same endpoints but in opposite directions. Also, whenever there is an edge from the sink side to the source side, there will also be an edge in the opposite direction which the min-cut will take into account.

Therefore we can then simply apply the Edmonds-Karp algorithm to find the maximum flow through such a network and then the corresponding minimal cut. The total complexity of our algorithm is $O(nm)$ as the flow $f \leq m$. $\square$

[K] Exercise 10. You know that $n + 2$ spies S, $s_1, s_2, \ldots, s_n$ 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. We proceed as below:

(a) We construct a flow network with:

- each vertex v_i representing each spy i
- 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 as we have $E = m$ and $f \leq m$, the total complexity of our algorithm is then $O(E|f|) = O(m^2)$ time.

(b) We approach this very similarly; we replicate the graph via part (a), then

- change all capacities of the edges to ∞
- for each non-source and non-sink vertex v_i, we split v_i into two vertices $v_i^{(\text{in})}$ and $v_i^{(\text{out})}$. For each edge incoming to v_i, we connect them to $v_i^{(\text{in})}$; for each edge outgoing from v, we modify it to instead outgoing from $v_i^{(\text{out})}$. We lastly connect $v_i^{(\text{in})}$ and $v_i^{(\text{out})}$ with capacity of 1.

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 slight different but still polynomial in m and n with $O(E|f|) = n(m + n)$.

□

§ SECTION THREE: BIPARTITE MATCHING

[K] Exercise 11. 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. We proceed by constructing a graph with:

- a vertex $v_{i,j}$ for each non-defective cell, and
- an edge between each pair of neighbouring non-defective cells.

This graph is bipartite, as each edge joins a vertex where $i + j$ is odd with a vertex where $i + j$ is even (*think about why*).

Placing a circuit corresponds to selecting an edge in this graph. No cell can be part of more than one circuit, so no two edges can share a vertex, i.e. our selection of edges must constitute a matching. Therefore, the maximum number of circuits that can be placed is exactly the size of the maximum matching in this bipartite graph. This can be found using the Edmonds-Karp algorithm in $O(VE)$ time (*why* VE *?*), and since $V \leq mn$ and $E \leq 4mn$ the runtime is clearly polynomial in m and n. □

[K] Exercise 12. You are hosting a game festival where n players may participate in m games. During the festival event, each individual player has a preference of k candidates (k is the same for all players) – the i-th player has candidates $a_{i1}, a_{i2}, \ldots, a_{ik}$. One player may choose at most one game (he may choose not to play, though) among those candidates.

For every game, the player having the highest score would receive a prize. It is guaranteed that no two players would share the same highest score (so you wouldn't have to worry about causing conflict between winners!). As the host of this game festival, you would like to know how many prizes should you prepare to ensure no player would end up receiving no prizes. That is, calculate the maximum number of distinct games chosen by players.

Solution. Construct the graph $G = \langle V, E \rangle$ as follows:

- Define V_1 to be the set of vertices that consist of the n players;
- Define V_2 to be the set of vertices that consist of the m games;
- The graph will then consist of $V = V_1 \cup V_2$ vertices, which is a disjoint union of V_1 and V_2.
- For each of the k preferences, we draw an edge from player $i \in V_1$ to game $a_{ij} \in V_2$ to denote that player i likes game a_{ij}.

From this construction, it is easy to see that such a graph G is *bipartite*. Therefore, the problem reduces to a maximum bipartite matching problem. Using the standard reduction to a maximum flow problem (construct a super source vertex s and a super sink vertex t; construct an edge from s to every vertex in V_1 and, for each vertex in V_2, construct an edge to t; each edge in the new graph has capacity 1), run Edmonds-Karp to obtain the maximum flow. The maximum flow corresponds to the maximum matching of the original problem. □

[K] Exercise 13. 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 bishops. Recall that bishops attack *diagonally*.

Solution. To solve this problem 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. □

[H] Exercise 14. You are the head of n spies, who are all wandering in a city. On one day you received a secret message that the bad guys in this city are going to arrest all your spies, so you'll have to arrange for your spies to run away and hide in strongholds. You have T minutes before the bad guys arrive. Your n spies are currently located at

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

and your m strongholds are located at

$$(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m).$$

The ith spy can move v_i units per minute, and each stronghold can hold only one spy.

Design an algorithm which runs in time polynomial in n and m determines which spies should be sent to which strongholds so that you have the maximum number of spies hiding from the bad guys.

Solution. First, for each spy i check which strongholds j are reachable in T minutes. Each of mn pairs can be checked in constant time by directly comparing the distance between spy i and stronghold j to v_iT, the distance that spy i can travel.

Then, we observe that matching spies with strongholds they can go to is a problem of finding the maximum bipartite matching, since no two spies can go to the same stronghold. We therefore construct a flow graph, with each spy i represented by a vertex p_i and each stronghold j represented by a vertex q_j, as well as a source s and a sink t. We place edges of capacity 1 from s to each p_i, and edges of capacity 1 from each q_j to t, and finally for each pair (i, j) such that spy i can reach stronghold j, we place an edge of capacity 1 from p_i to q_j. Running the Ford-Fulkerson algorithm on this graph finds a maximum flow, and hence the size of the maximum bipartite matching. We inspect the flow function found by this algorithm to recover the actual maximum matching – for each edge (p_i, q_j) carrying flow, we send spy i to stronghold j.

There are $n + m$ total nodes and up to $nm + n + m = O(nm)$ edges, so constructing the graph takes $O(nm)$ time. Further, the value of a maximum flow is bounded above by $\min(n, m)$, so finding the maximum flow and hence the maximum matching takes $O(nm \times \min(n, m))$ time. □

[H] Exercise 15. 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.

Hint: Combine binary search with a max flow.

Solution. First, sort time travel distances d_i in increasing order; for the rest of the solution, we may now assume that $d_{i+1} \geq d_i$.

Consider a value d_i for some i and construct a bipartite graph G_i with warehouses w_j as the left side of the partition and with shops s_j ($1 \leq j \leq n$). Connect all warehouses with all shops which are within travel distance times d_i. Use max flow to see if such bipartite graph has a perfect maximum matching of size n. Use a binary search to find the smallest i such that graph G_i has a matching of size n, i.e. a matching in which every warehouse has been matched with a shop, so that different warehouses are assigned different shops. □

[H] Exercise 16. There are n boys and n girls at a party. Whenever a song starts, they will form exactly n pairs to dance and no boy will dance with the same girl twice.

Some pairs of boys and girls like each other, and all other pairs of boys and girls dislike each other. Every boy will dance with at most k girls that he dislikes, and each girl will dance with at most k boys that she dislikes where $k < n$.

As the DJ, it is your job to determine the maximum number of songs to play such that it is possible for pairs to be formed that satisfy the above requirement. Design a $O(n^4 \log n)$ algorithm that achieves this task.

Hint: Start with the case where $k = 0$ and fix a capacity x for edges between each boy and girl. How can you generalise this to arbitrary k?

Solution. We first take care of the case where $k = 0$. Label the boys $b_1, b_2, \cdots, b_n$, and the girls $g_1, g_2, \cdots, g_n$, and fix $x \leq n$. Construct a flow network with:

- a vertex b_i for each boy, and a vertex g_j for each girl
- vertices s and t, the super source and super sink
- for each boy, an edge from s to b_i with capacity x
- for each girl, an edge from g_j to t with capacity x
- for each boy-girl pair who like each other, an edge from b_i to g_j with capacity 1.

The total capacity leaving s (and entering t) is nx, so the value of a maximum flow is at most nx. If x songs can be played, then we can construct a flow of value nx by flowing:

- all edges (s, b_i) with x units of flow
- all edges (g_j, t) with x units of flow
- for each pair (b_i, g_j) who dance together, the edge (b_i, g_j) with one unit of flow.

Proving the converse (that a flow of exactly nx allows x songs to be played) is more subtle.

Proof: Consider a maximum flow in the network. For each edge (b_i, g_j) which carries flow, we will record that boy i and girl j dance together for one song. If the maximum flow is nx, we will thus find x partners for each attendee, while enforcing that:

- no pair who dislike each other dances together, and
- no pair dances together more than once.

However, we have yet to confirm whether the nx boy-girl pairings can be grouped into x songs. The proof of this fact was not required to achieve full marks, but it is included here for completeness.

Construct a graph with $2n$ vertices corresponding to the boys and girls, with nx edges corresponding to the matched pairs. This graph is clearly bipartite, with parts $B = \{b_1, \dots, b_n\}$ and $G = \{g_1, \dots, g_n\}$, and x-regular (every vertex is incident to exactly x edges). Now, we use Hall's marriage theorem.

Definition: For a subset W of B, let $N(W) \subseteq G$ be the set of vertices adjacent to at least one vertex in W.

Theorem*: Suppose for all subsets W of B that $|W| \leq |N(W)|$. Then the graph has a perfect matching, i.e. a matching of size n.

For a contradiction, suppose there is a set W of p boys to which only $q < p$ girls are adjacent. There are exactly px edges between W and $N(W)$, since each boy is matched with exactly x girls. However, each of these edges is also incident to exactly one of the q girls, and since $p > q$ it is impossible for all q of these girls to have fewer than x edges each. Therefore, we have the required contradiction, and it follows that a perfect matching exists (corresponding to the pairs who dance together in the first song). Furthermore, upon removing these edges, the remaining graph is $(x-1)$-regular, so the same property applies. We can thus make n pairs for each of the x songs, as required. $\square$

Thus, we have a test for whether x songs can be played or not. We can then apply this test for $x = 0, 1, 2, \ldots$ until we find the largest value of x for which it is possible, which is the answer.

We run the Edmonds-Karp algorithm up to $n + 1$ times. In each iteration, the number of edges is $O(n^2)$ and the maximum flow is $O(n^2)$, so the total time complexity is $O(n^5)$. Note that this can be improved to $O(n^4 \log n)$ by binary searching for the largest allowable value of x. This is clearly polynomial in n - running max flow on $O(n)$ vertices, $O(n^2)$ edges at most n times.

To generalise to arbitrary k, we consider the following construction.

The fundamental structure is as above: for each x from 0 to n, run the Edmonds-Karp algorithm once to determine whether x songs can be played, and return the largest x for which it was possible. However, the graph construction is more intricate. For a particular value x, construct a flow graph with:

- vertices b_i, b_i^L and b_i^D for each boy
- vertices g_j, g_j^L and g_j^D for each girl
- vertices s and t, the super source and super sink
- for each boy:
  - an edge from s to b_i with capacity x
  - an edge from b_i to b_i^L with capacity x
  - an edge from b_i to b_i^D with capacity k
- for each girl:
  - an edge from g_j to t with capacity x
  - an edge from g_j^L to g_j with capacity x
  - an edge from g_j^D to g_j with capacity k
- for each boy-girl pair:
  - an edge from b_i^L to g_j^L with capacity 1 if they like each other, or
  - an edge from b_i^D to g_j^D with capacity 1 if they don't like each other.

As above, we will find a maximum flow in this graph, and record each boy-girl edge carrying flow as a pair who dance together. This guarantees that:

- no pair dances together more than once, since for each pair (i, j), either
  - they like each other, so $c(b_i^L, g_j^L) = 1$ and $c(b_i^D, g_j^D) = 0$, or
  - they don't like each other, so $c(b_i^L, g_j^L) = 0$ and $c(b_i^D, g_j^D) = 1$.
- each boy and each girl has exactly x partners
- of these partners, at most k are not liked.

The asymptotic time complexity is unaffected, since the number of edges is still $O(n^2)$.

Note that setting $k = 0$ in this graph recovers the construction from before. $\square$