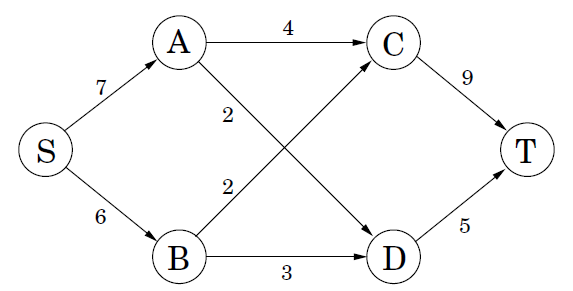
CSCI 570 - HW 6

Solutions

1. You are given the following graph *G*. Each edge is labeled with the capacity of that edge.



1. Find a max-flow in *G* using the Ford-Fulkerson algorithm. Draw the residual graph *Gf* corresponding to the max flow. You do not need to show all intermediate steps.



1. Find the max-flow value and a min-cut.

Solution: f = 11, cut: ({S, A, B}, {C, D, T })

1. Prove or disprove that increasing the capacity of an edge that belongs to a min cut will always result in increasing the maximum flow.

Solution: Disprove by a counterexample. Increasing (B,D) by one won’t increase the max-flow.

1. Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible *base stations.* We’ll suppose there are *n* clients, with the position of each client specified by its (*x*, *y*) coordinates in the plane. There are also *k* base stations; the position of each of these is specified by (*x*, *y*) coordinates as well. For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a *range parameter R* which means that aclient can only be connected to a base station that is within distance *R*. There is also a *load parameter L* which means that no more than *L* clients can be connected to any single base station. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station.

Solution: Introduce a vertex *bi* for every base station and a vertex *ci* for every client. Connect each client to a base station if that station is within the range R. Assign capacity 1 to those edges. Let *s* be a source vertex and *t* be a sink vertex. For every client vertex, add an edge (s, *ci*) of capacity 1. For every base station vertex, add an edge (*bi*, t) of capacity L.

Claim: The problem has a solution if and only if the network has a max-flow of value *n*.

Proof.

Assume that there is a solution. It means that every client is connected to a correspondent base station. So we can push a flow of 1from the source to each client. On the edges between clients and stations, we assign a flow of 1 or 0, depending whether a client is connected by a particular station or not. On the edges between stations and the sink, we assign a flow value equal to the number of clients covered by that dripper. This is possible, since we have a valid solution. It follows the max-flow is *n*.

Conversely, assume there is a max-flow of value *n*. This means that each client will get a flow of one unit. We also observe that no base station is overloaded due to the capacity condition L.

1. Consider a drip irrigation system, which is an irrigation method that saves water and fertilizer by allowing water to drip slowly to the roots of plants. Suppose that the locations of all drippers are given to us in terms of their coordinates (xd, yd). Also, we are given locations of plants specified by their coordinates (xp, yp). A dripper can only provide water to plants within distance *L.* A single dripper can provide water to no more than *n* plants. However, we recently got some funding to upgrade our system with which we bought *k* monster drippers, which can provide water supply to three times the number of plants compared to standard drippers. So, we now have *i* standard drippers and *k* monster drippers. Given the locations of the plants and drippers, as well as the parameters *L* and *n*, decide whether every plant can be watered simultaneously by a dripper, subject to the above mentioned constraints. Justify carefully that your algorithm is correct and can be obtained in polynomial time.

Solution: Introduce a vertex *pi* for every plant and a vertex *di* for every dripper. Connect each plant to a dripper if that plant is within the range *L*. Assign capacity 1 to those edges. Let *s* be a source vertex and *t* be a sink vertex. For every plant vertex, add an edge (*s*, *pi*) of capacity 1. For every dripper vertex, add an edge (*di*, *t*) of capacity n if that is a standard dripper and 3 n if that is a monster dripper.

Claim: The problem has a solution (every plant can be watered) if and only if the network has a max-flow of value *n*.

Proof.

Assume that there is a solution. It means that every plant can be watered. So we can push a flow of 1from the source to each plant. On the edges between plants and drippers, we assign a flow of 1 or 0, depending whether a plant is covered by a particular dripper or not. On the edges between drippers and the sink, we assign a flow value equal to the number of plants covered by that dripper. This is possible, since we have a valid solution. It follows the max-flow is *n*.

Conversely, assume there is a max-flow of value *n*.This means that each plant vertex will get a flow of 1. We also observe that no dripper is overloaded due to the capacity condition.

1. At a dinner party, there are *n* families *a*1, *a*2, …, *a*n and *m* tables *b*1, *b*2, …, *b*m. The *i*-th family *ai* has *gi* members and the *j*-th table *bj* has *hj* seats. Everyone is interested in making new friends and the dinner party planner wants to seat people such that no two members of the same family are seated at the same table. Design an algorithm that decides if there exists a seating assignment such that everyone is seated and no two members of the same family are seated at the same table. What would be a seating arrangement?

Solution: Introduce a vertex *ai* for every family and a vertex *bi* for every table. Connect each family to all tables. Assign capacity 1 to those edges. Let *s* be a source vertex and *t* be a sink vertex. For every family vertex, add an edge (*s*, *ai*) of capacity *gi*. For every table vertex, add an edge (*bi*, *t*) of capacity *hi*.

Claim: The problem has a solution (a valid seating assignment) if and only if the network has a max-flow of value *g*1+ *g*2+ …+ *g*n.

Proof.

Assume that there is a solution. It means that every family member is seated. So we can push a flow of *gi* from the source to each family. On the edges between families and tables, we assign a flow of 1 or 0. Since no two members of the same family are seated at the same table, each family vertex will have outgoing flow of value 1. On the edges between tables and the sink, we assign a flow value equal to the number of people seating at that table. This must be possible, since we have a valid assignment.

Conversely, assume there is a max-flow of value *g*1+ *g*2+ …+ *g*n. This means that each family vertex will get a flow of *gi*. Due to capacity constrain (each edge (*ai*, *bi*) has a unit capacity) no two members will seat at the same table. We also observe that no table is overloaded due to the capacity condition *hi*.

Seating assignment: run a network flow algorithm and pick edges (*ai*, *bi*) with a unit flow.

1. In the network below, the demand values are shown on vertices (supply value if negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. You need to show all your steps.

b:5

c:-4

(2,6)

(2,4)

(3,4)

a:9

(1,4)

(2,3)

(2,5)

d:-11

e:3

(2,5)

1. Turn the circulation with lower bounds problem into a circulation problem without lower bounds.

Solution: Push a flow on edge equal to the lower bound. Compute L(v) = fin(v) – fout(v).

L(a) = 4, L(b)=-2, L(c)=0, L(d)=-5, L(e)=3

Recompute the demands: d’(v) = d(v) – L(v)

d’(a) =5, d’(b) =7, d’(c) =-4, d’(d) =-6, d’(e) =0

1. Turn the circulation with demands problem into the maximum flow problem.

Solution: Create a source vertex s and connect it with c and d, with weights 4 and 6 respectively. Create a sink vertex t. Connect a and b with t, weights 5 and 7 respectively.

1. Does a feasible circulation exist? Explain your answer.

Solution: No, a feasible circulation doesn’t exist. In the given original network, the total demand value on the vertices doesn’t match the total supply value on all the vertices.