Network Flows

1 Network flow problems

Problem 1: Blood supply (KT)

Statistically, the arrival of spring typically results in increased accidents and increased need for emergency medical treatment, which often requires blood transfusions. Consider the problem faced by a hospital that is trying to evaluate whether its blood supply is sufficient.

The basic rule for blood donation is the following. A person's own blood supply has certain *antigens* present (we can think of antigens as a kind of molecular signature); and a person cannot receive blood with a particular antigen if their own blood does not have this antigen present. Concretely, this principle underpins the division of blood into four types: A, B, AB, and O. Blood of type A has the A antigen, blood of type B has the B antigen, blood of type AB has both, and blood of type O has neither. Thus, patients with type A can receive only blood types A or O in a transfusion, patients with type B can receive only B or O, patients with type O can receive only O, and patients with type AB can receive any of the four types.[4]

1. Let s_O, s_A, s_B, and s_{AB} denote the supply in whole units of the different blood types on hand. Assume that the hospital knows the projected demand for each blood type d_O, d_A, d_B, and d_{AB} for the coming week. Give a polynomial-time algorithm to evaluate if the blood on hand would suffice for the projected need.

2. Consider the following example. Over the next week, they expect to need at most 100 units of blood. The typical distribution of blood types in U.S. patients is roughly 45 percent type O, 42 percent type A, 10 percent type B, and 3 percent type AB. The hospital wants to know if the blood supply it has on hand would be enough if 100 patients arrive with the expected type distribution. There is a total of 105 units of blood on hand. The table below gives these demands, and the supply on hand.

Is the 105 units of blood on hand enough to satisfy the 100 units of demand? Find an allocation that satisfies the maximum possible number of patients. Use an argument based on a minimum-capacity cut to show why not all patients can receive blood. Also, provide an explanation for this fact that would be understandable to the clinic administrators, who have not taken a course on algorithms. (So, for example, this explanation should not involve the words flow, cut, or graph in the sense we use them in this book.)

Problem 2: Bounds on augmentation (CLRS)

Show that a maximum flow in network $G = (V, E)$ can always be found by a sequence of at most $|E|$ augmenting paths. (*Hint*: Determine the paths *after* finding the maximum flow.)

Problem 3: Course scheduling

Each of a list of n courses want to use a large lecture room on campus, which has a list of $m \geq n$ weekly

blood type	supply	demand
O	50	45
A	36	42
B	11	8
AB	8	3

classtimes it is available. Each instructor I has given an ordering of their preferred classtimes, $T_{I,1}, ..T_{I,m}$, from best to worst. We need to assign each course a distinct class time. If instructor I is assigned the time $T_{I,K}$, we say the instructor has unhappiness K. We want to find an assignment that minimizes the maximum unhappiness of an instructor. Give an efficient algorithm for this problem. (Hint: Consider the related problem, "Is there an assignment with unhappiness at most K?" .

Problem 4: Database projections (KT 7.38)

You're working with a large database of employee records. For the purposes of this question, we'll picture the database as a two-dimensional table T with a set R of m rows and a set C of n columns; the rows correspond to individual employees, and the columns correspond to different attributes.

To take a simple example, we may have four columns labeled

`name, phone number, start date, manager's name`

and a table with five employees as shown here. Given a subset S of the columns, we can obtain a new, smaller

Table 1: Table with five employees.

name	**phone number**	**start date**	**manager's name**
Alanis	3-4563	6/13/95	Chelsea
Chelsea	3-2341	1/20/93	Lou
Elrond	3-2345	12/19/01	Chelsea
Hal	3-9000	1/12/97	Chelsea
Raj	3-3453	7/1/96	Chelsea

table by keeping only the entries that involve columns from S. We will call this new table the *projection* of T onto S, and denote it by $T[S]$. For example, if $S = \{$`name, start date`$\}$, then the projection $T[S]$ would be the table consisting of just the first and third columns.

There's a different operation on tables that is also useful, which is to *permute* the columns. Given a permutation p of the columns, we can obtain a new table of the same size as T by simply reordering the columns according to p. We will call this new table the *permutation* of T by p, and denote it by T_p.

All of this comes into play for your particular application, as follows. You have k different subsets of the columns $S_1, S_2, \ldots, S_k$ that you're going to be working with a lot, so you'd like to have them available in a readily accessible format. One choice would be to store the k projections $T[S_1], T[S_2], \ldots, T[S_k]$, but this would take up a lot of space. In considering alternatives to this, you learn that you may not need to explicitly project onto each subset, because the underlying database system can deal with a subset of the columns particularly efficiently if (in some order) the members of the subset constitute a *prefix* of the columns in left-to-right order. So, in our example, the subsets $\{$`name, phone number`$\}$ and $\{$`name, start date, phone number,`$\}$ constitute prefixes (they're the first two and first three columns from the left, respectively); and as such, they can be processed much more efficiently in this table than a subset such as $\{$`name, start date`$\}$, which does not constitute a prefix. (Again, note that a given subset S_i does not come with a specified order, and so we are interested in whether there is *some* order under which it forms a prefix of the columns.)

So here's the question: Given a parameter $\ell < k$, can you find ℓ permutations of the columns $p_1, p_2, \ldots, p_\ell$ so that for every one of the given subsets S_i (for $i = 1, 2, \ldots, k$), it's the case that the columns in S_i constitute a prefix of at least one of the permuted tables $T_{p_1}, T_{p_2}, \ldots, T_{p_\ell}$? We'll say that such a set of permutations constitutes a valid solution to the problem; if a valid solution exists, it means you only need to store the ℓ permuted tables rather than all k projections. Give a polynomial-time algorithm to solve this problem; for instances on which there is a valid solution, your algorithm should return an appropriate set of ℓ permutations.

Example. Suppose the table is as above, the given subsets are

$S_1 = \{$`name, phone number`$\}$,
$S_2 = \{$`name, start date`$\}$,
$S_3 = \{$`name, manager's name, start date`$\}$,

and $\ell = 2$. Then there is a valid solution to the instance, and it could be achieved by the two permutations

$p_1 = \{$`name, phone number, start date, manager's name`$\}$,
$p_2 = \{$`name, start date, manager's name, phone number`$\}$.

This way, S_1 constitutes a prefix of the permuted table T_{p_1}, and both S_2 and S_3 constitute prefixes of the permuted table T_{p_2}.

Problem 5: Dilworth's theorem

Let G be an acyclic directed graph. A set $A \subseteq V$ is independent if no two elements of A are joined by a directed path in G. A path cover of G is a set $\{P_1, \dots, P_k\}$ of directed paths such that every node of G is a vertex of at least one P_i. Prove Dilworth's Theorem, that the maximum size of an independent set is equal to the minimum size of a path cover.

Problem 6: Domino Tiling

You are given a subset of the $n \times n$ grid of squares, described as an $n \times n$ boolean matrix. The question is whether it is possible to cover precisely this subset with 1×2 rectangular dominoes, subject to the following constraints: Each domino must be placed either horizontally or vertically, covering two adjacent squares of the grid. Both squares must be in the subset. No two dominoes can cover the same square. Each square in the subset must be covered by some domino.

Give an efficient algorithm for deciding whether it is possible, given as input the array of booleans describing the subset True meaning the square is in the subset, False meaning it is not.

Problem 7: Driving Schedule

Some of your friends with jobs out West decide they really need some extra time each day to sit in front of their laptops, and the morning commute from Woodside to Palo Alto seems likely the only option. So they decided to carpool to work.

Unfortunately, they all hate to drive, so they want to make sure that any carpool management they agree upon is fair and does not overload any individual with too much driving. Some sort of simple round-robin scheme is out, because none of them goes to work every day, and so the subset of them in the car varies from day to day.

Here is one way to define *fairness*. Let the people be labeled $S = \{p_1, \dots, p_k\}$. We say that the *total driving obligation* of p_j over a set of days is the expected number of times that p_j would have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, suppose the carpool plan lasts for d days, and on the i-th day a subset $S_i \subseteq S$ of the people go to work. Then the above definition of the total driving obligation Δ_j for p_j can be written as $\Delta_j = \sum_{i:p_j \in S_i} \frac{1}{|S_i|}$. Ideally, we would like to require that p_j drives at most Δ_j times. However, Δ_j may not be an integer.

So let us say that a *driving schedule* is a choice of a driver for each day, that is, a sequence $p_{i_1}, \dots, p_{i_d}$ with $p_{i_t} \in S_t$ and that a *fair driving schedule* is one in which each p_j is chosen as the driver on at most $\lceil \Delta_j \rceil$ days.

1. Prove that for any sequence of sets $S_1, \dots, S_d$, there exists a fair driving schedule.
2. Give an algorithm to compute a fair driving schedule with running time in polynomial in k and d.

Design an efficient polynomial-time algorithm.

Provide a high-level description of your algorithm, prove its correctness, and analyze its time complexity.

Problem 8: Edge connectivity (CLRS)

The ***edge connectivity*** of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how the edge connectivity of an undirected graph $G = (V, E)$ can be determined by running a maximum-flow algorithm on at most $|V|$ flow networks, each having $O(V)$ vertices and $O(E)$ edges.

Problem 9: Edmonds-Karp variation (CLRS)

Suppose that a flow network $G = (V, E)$ has symmetric edges, that is, $(u, v) \in E$ if and only if $(v, u) \in E$. Show that the Edmonds-Karp algorithm terminates after at most $|V||E|/4$ iterations. (*Hints*: For any edge (u, v), consider how both $\delta(s, u)$ and $\delta(v, t)$ change between times at which (u, v) is critical.)

Problem 10: Electronic message transmission systems

Consider the problem of selecting sites for an electronic message transmission system. Any number of sites can be chosen from a finite set of potential locations. We know the cost c_i of establishing site i and the revenue r_{ij} generated between sites i and j, if they are both selected. Both the costs and revenues are non-negative. Find an efficient algorithm to determine the subset of vertices such that the sum of the edge revenues less the vertex costs is as large as possible.

Design an efficient polynomial-time algorithm.

Provide a high-level description of your algorithm, prove its correctness, and analyze its time complexity.

Problem 11: Free Standing block of countries

Consider the following definition. We are given a set of n countries that are engaged in trade with one another. For each country i, we have the value s_i of its budget surplus; this number may be positive or negative, with a negative number indicating a deficit. For each pair of countries i, j, we have the total value e_{ij} of all exports from i to j; this number is always nonnegative. We say that a subset S of the countries is free-standing if the sum of the budget surpluses of the countries in S, minus the total value of all exports from countries in S to countries not in S, is nonnegative.

Give a polynomial-time algorithm that takes this data for a set of n countries and decides whether it contains a nonempty free-standing subset that is not equal to the full set.

Problem 12: Going to school (CLRS)

Professor Adam has two children who, unfortunately, dislike each other. The problem is so severe that not only do they refuse to walk to school together, but in fact each one refuses to walk on any block that the other child has stepped on that day. The children have no problem with their paths crossing at a corner. Fortunately both the professors house and the school are on corners, but beyond that he is not sure if it is going to be possible to send both of his children to the same school. The professor has a map of his town. Show how to formulate the problem of determining whether both his children can go to the same school as a maximum-flow problem.

Problem 13: Graph cohesiveness (KT 7.46)

In sociology, one often studies a graph G in which nodes represent people and edges represent those who are friends with each other. Let's assume for purposes of this question that friendship is symmetric, so we can consider an undirected graph.

Now suppose we want to study this graph G, looking for a "close-knit" group of people. One way to formalize this notion would be as follows. For a subset S of nodes, let $e(S)$ denote the number of edges in S-that is, the number of edges that have both ends in S. We define the *cohesiveness* of S as $e(S)/|S|$. A natural thing to search for would be a set S of people achieving the maximum cohesiveness.

1. Give a polynomial-time algorithm that takes a rational number α and determines whether there exists a set S with cohesiveness at least α.

2. Give a polynomial-time algorithm to find a set S of nodes with maximum cohesiveness.

Problem 14: Hall's theorem (CLRS)

A ***perfect matching*** is a matching in which every vertex is matched. Let $G = (V, E)$ be an undirected bipartite graph with vertex partition $V = L \cup R$, where $|L| = |R|$. For any $X \subseteq V$, define the ***neighborhood*** of X as

$$N(X) = \{y \in V : (x, y) \in E \text{ for some } x \in X\}$$

that is, the set of vertices adjacent to some member of X. Prove ***Hall's theorem***: there exists a perfect matching in G if and only if $|A| \leq |N(A)|$ for every subset $A \subseteq L$.

Problem 15: HopcroftKarp improvement (CLRS)

In this problem, we describe a faster algorithm, due to Hopcroft and Karp, for finding a maximum matching in a bipartite graph. The algorithm runs in $O(\sqrt{V}E)$ time. Given an undirected, bipartite graph $G = (V, E)$, where $V = L \cup R$ and all edges have exactly one endpoint in L, let M be a matching in G. We say that a simple path P in G is an ***augmenting path*** with respect to M if it starts at an unmatched vertex in L, ends at an unmatched vertex in R, and its edges belong alternately to M and $E - M$. (This definition of an augmenting path is related to, but different from, an augmenting path in a flow network.) In this problem, we treat a path as a sequence of edges, rather than as a sequence of vertices. A shortest augmenting path with respect to a matching M is an augmenting path with a minimum number of edges. Given two sets A and B, the ***symmetric difference*** $A \bigoplus B$ is defined as $(A - B) \cup (B - A)$, that is, the elements that are in exactly one of the two sets.

1. Show that if M is a matching and P is an augmenting path with respect to M, then the symmetric difference $M \bigoplus P$ is a matching and $|M \bigoplus P| = |M| + 1$. Show that if $P_1, P_2, \ldots, P_k$ are vertex-disjoint augmenting paths with respect to M, then the symmetric difference $M \bigoplus (P_1 \cup P_2 \cup \cdots \cup P_k)$ is a matching with cardinality $|M| + k$.

   The general structure of our algorithm is the following:

Algorithm 1: HOPCROFT-KARP (G)

```
1: $M \leftarrow \emptyset$
2: repeat
3:     let $\mathscr{P} \leftarrow \{P_1, P_2, \ldots, P_k\}$ be a maximum set of vertex-disjoint shortest augmenting paths with
       respect to $M$
4:     $M \leftarrow M \bigoplus (P_1 \cup P_2 \cup \cdots \cup P_k)$
5: until $\mathscr{P} = \emptyset$
6: return $M$
```

   The remainder of this problem asks you to analyze the number of iterations in the algorithm (that is, the number of iterations in the **repeat** loop) and to describe an implementation of line 3.

2. Given two matchings M and M^* in G, show that every vertex in the graph $G' = (V, M \bigoplus M^*)$ has degree at most 2. Conclude that G' is a disjoint union of simple paths or cycles. Argue that edges in each such simple path or cycle belong alternately to M or M^*. Prove that if $|M| \leq |M^*|$, then $M \bigoplus M^*$ contains at least $|M^*| - |M|$ vertex-disjoint augmenting paths with respect to M.

   Let l be the length of a shortest augmenting path with respect to a matching M, and let $P_1, P_2, \ldots, P_k$ be a maximum set of vertex-disjoint augmenting paths of length l with respect to M. Let $M' = M \bigoplus (P_1 \bigcup P_2 \bigcup \cdots \bigcup P_k)$, and suppose that P is a shortest augmenting path with respect to M'.

3. Show that if P is vertex-disjoint from $P_1, P_2, \ldots, P_k$, then P has more than l edges.

4. Now suppose that P is not vertex-disjoint from $P_1, P_2, \ldots, P_k$. Let A be the set of edges $(M \bigoplus M') \bigoplus P$. Show that $A = (P_1 \bigcup P_2 \bigcup \cdots \bigcup P_k) \bigoplus P$ and that $|A| \geq (k + 1)l$. Conclude that P has more than l edges.

5. Prove that if a shortest augmenting path for M has length l, the size of the maximum matching is at most $|M| + |V|/l$.

6. Show that the number of **repeat** loop iterations in the algorithm is at most $2\sqrt{V}$. (Hint: By how much can M grow after iteration number $\sqrt{V}$?)

7. Give an algorithm that runs in $O(E)$ time to find a maximum set of vertex-disjoint shortest augmenting paths $P_1, P_2, ..., P_k$ for a given matching M. Conclude that the total running time of HOPCROFT-KARP is $O(\sqrt{V}E)$.

Problem 16: Job Assignment

You are given a set of n jobs and a set of m machines. For each job you are given a list of machines capable of performing the job. An assignment specifies for each job one of the machines capable of performing it. The overhead for an assignment is the maximum number of jobs performed by the same machine. The job assignment problem is: given such lists and an integer $1 \leq k \leq n$, is there an assignment with overhead at most k?

Problem 17: Job scheduling (KT 7.41)

Suppose you're managing a collection of processors and must schedule a sequence of jobs over time.

The jobs have the following characteristics. Each job j has an arrival time a_j when it is first available for processing, a length ℓ_j which indicates how much processing time it needs, and a deadline d_j by which it must be finished. (We'll assume $0 < \ell_j \leq d_j - a_j$.) Each job can be run on any of the processors, but only on one at a time; it can also be preempted and resumed from where it left off (possibly after a delay) on another processor.

Moreover, the collection of processors is not entirely static either: You have an overall pool of k possible processors; but for each processor i, there is an interval of time $[t_i, t'_i]$ during which it is available; it is unavailable at all other times.

Given all this data about job requirements and processor availability, you'd like to decide whether the jobs can all be completed or not. Give a polynomial-time algorithm that either produces a schedule completing all jobs by their deadlines or reports (correctly) that no such schedule exists. You may assume that all the parameters associated with the problem are integers.

Example. Suppose we have two jobs J_1 and J_2. J_1 arrives at time 0, is due at time 4, and has length 3. J_2 arrives at time 1, is due at time 3, and has length 2. We also have two processors P_1 and P_2. P_1 is available between times 0 and 4; P_2 is available between times 2 and 3. In this case, there is a schedule that gets both jobs done.

- At time 0, we start job J_1 on processor P_1.
- At time 1, we preempt J_1 to start J_2 on P_1.
- At time 2, we resume J_1 on P_2. (J_2 continues processing on P_1.)
- At time 3, J_2 completes by its deadline. P_2 ceases to be available, so we move J_1 back to P_1 to finish its remaining one unit of processing there.
- At time 4, J_1 completes its processing on P_1.

Notice that there is no solution that does not involve preemption and moving of jobs.

Problem 18: Maximum flow by scaling (CLRS)

Let $G = (V, E)$ be a flow network with source s, sink t, and an integer capacity $c(u, v)$ on each edge $(u, v) \in E$. Let $C = \max_{(u,v)\in E} c(u, v)$.

1. Argue that a minimum cut of G has capacity at most $C|E|$.

2. For a given number K, show that an augmenting path of capacity at least K can be found in $O(E)$ time, if such a path exists.

3. Argue that MAX-FlOW-BY-SCALING return a maximum flow.
4. Show that the capacity of a minimum cut of the residual graph G_f us at most $2K|E|$ each time line 4 is executed.
5. Argue that inner **while** loop of line 5-6 executed $O(E)$ times for each value of K.
6. Conclude that MAX-FLOW-BY-SCALING can be implemented so that it runs in $O(E^2 lgC)$ time.

Problem 19: Maximum Augmenting Flow

An approach to improving the augmenting path algorithm is to choose at each step an augmenting path of maximum x-width. Give a good bound on the number of augmentations. How efficiently can an augmenting path of maximum x-width be found?

Problem 20: Maximum independent set in bipartite graphs

Let $G = (V, E)$ be an undirected graph. An *independent set* in G is a set of vertices no two of which are connected. Design an efficient algorithm to find a maximum-size independent set in a given bipartite graph.

Problem 21: NASA (CLRS)

Professor Spock is consulting for NASA, which is planning a series of space shuttle flights and must decide which commercial experiments to perform and which instruments to have on board each flight. For each flight, NASA considers a set $E = \{E_1, E_2, \ldots, E_m\}$ of experiments, and the commercial sponsor of experiments E_j has agreed to pay NASA p_j dollars for the results of the experiment. The experiments use a set $I = \{I_1, I_2, \ldots, I_n\}$ of instruments; each experiment E_j requires all the instruments in a subset $R_j \subseteq I$. The cost of carrying instrument I_k is c_k dollars. The professor's job is to find an efficient algorithm to determine which experiments to perform and which instruments to carry for a given flight in order to maximum the net revenue, which is the total income from experiments performed minus the total cost of all instruments carried.

Consider the following network G, The network contains a source vertex s, vertices $I_1, I_2, \ldots, I_n$, vertices $E_1, E_2, \ldots, E_m$, and a sink vertex t. For $k = 1, 2, \ldots, n$, there is an edge (s, I_k) of capacity c_k, and for $j = 1, 2, \ldots, m$, there is an edge (E_j, t) of capacity p_j. For $k = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$, if $I_k \in R_j$, then there is an edge (I_k, E_j) of infinite capacity.

1. Show that id $E_j \in T$ for a finite-capacity cur (S, T) of G, the $I_k \in T$ for each $I_k \in R_j$.
2. Show how to determine the maximum net revenue from the capacity of the minimum cut of G and the given p_j values.
3. Give an efficient algorithm to determine which experiments to perform and which instruments to carry. Analyze the running time of your algorithm in terms of m, n, and $r = \sum_{j=1}^{m} |R_j|$

Problem 22: Negative edge capacities (CLRS)

Suppose that we allow a flow network to have negative (as well as positive) edge capacities. In such a network, a feasible flow need not exist.

1. Consider an edge (u, v) in a flow network $G = (V, E)$ with $c(u, v) < 0$. Briefly explain what such a negative capacity means in terms the flow between u and v.

   Let $G = (V, E)$ be a flow network with negative edge capacities, and let s and t be the source and sink of G. Construct the ordinary flow network $G' = (V', E')$ with capacity function c', source s', and sink t', where

$$V' = V \cup \{s', t'\}$$

and

$$\begin{aligned} E' = E &\cup \{(u,v) : (u,v) \in E\} \\ &\cup \{(s',v) : v \in V\} \\ &\cup \{(u,t') : u \in V\} \\ &\cup \{(s,t) : (t,s)\} \end{aligned}$$

We assign capacities to edges as follows. For each edge $(u,v) \in E$, we set

$$c'(u,v) = c'(v,u) = (c(u,v) + c(v,u))/2.$$

For each vertex $v \in V$, we set

$$c'(s',u) = \max(0, c(V,u) - c(u,V)/2)$$

and

$$c'(u,t') = \max(c(u,V) - c(V,u)/2)$$

We also set $c'(s,t) = c'(t,s) = \infty$.

2. Prove that if a feasible flow exists in G, then all capacities in G' are nonnegative and a maximum flow exists in G' such that all edges into the sink t' are saturated.
3. Prove the converse of part(b). Your proof should be constructive, that is, given a flow in G' that saturates all the edges into t', your proof should show how to obtain a feasible flow in G.
4. Describe an algorithm that finds a maximum feasible flow in G. Denote by $MF(|V|,|E|)$ the worst-case running time of an ordinary maximum flow algorithm on a graph with $|V|$ vertices and $|E|$ edges. Analyze your algorithm for computing the maximum flow of a flow network with negative capacities in terms of MF.

Problem 23: No augmenting path

Suppose we have a feasible flow x such that, for some nonnegative number K, there is no x-augmenting path of x-width greater than K. Prove that $f_x(s)$ is within Km of the maximum value of a feasible flow. (Hint: How would you show this for $K = 0$?) m is the number of edges in the graph.

Problem 24: Number puzzle

You are trying to solve the following puzzle. You are given the sums for each row and column of an $n \times n$ matrix of integers in the range $1 \ldots, M$, and wish to reconstruct a matrix that is consistent. In other words, your input is M, $r_1, \ldots, r_n$, $c_1, \ldots, c_n$. Your output should be a matrix $a_{i,j}$ of integers between 1 and M so that $\sum_i a_{i,j} = r_j$ for $1 \le j \le n$ and $\sum_j a_{i,j} = c_i$ for $1 \le i \le n$; if no such matrix exists, you should output, "Impossible". Give an efficient algorithm for this problem.

Problem 25: Path cover (CLRS)

A ***path cover*** of a directed graph $G = (V,E)$ is a set P of vertex-disjoint paths such that every vertex in V is included in exactly one in P. Paths may start and end anywhere, and they may be of any length, including 0. A ***minimum path cover*** of G is a path cover containing the fewest possible paths.

1. Give an efficient algorithm to find a minimum path cover of a directed acyclic graph $G = (V,E)$. (*hint*: Assuming that $V = \{1,2,\ldots,n\}$, construct the graph $G' = (V',E)$, where

$$\begin{aligned} V' &= \{x_0, x_2, \ldots, x_n\} \cup \{y_0, y_1, \ldots, y_n\}, \\ E' &= \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i,j) \in E\}, \end{aligned}$$

and run a maximum-flow algorithm.)

2. Does your algorithm work for directed graphs that contain cycle? Explain.

Problem 26: Maximum likelihood points of failure

A network is described as a directed graph (not necessarily acyclic), with a specified *source* s and *destination* t. A set of nodes (not including s or t) is a *failure point* if deleting those nodes disconnects t from s. For each node of the graph, i, a *failure probability* $0 \leq p_i \leq 1$ is given. It is assumed that nodes fail independently, so the failure probability for a set $F \subseteq V$ is $\prod_{i \in F} p_i$. Give an algorithm which, given G and $p_i, i \in V$, finds the failure point F with the maximum failure probability.

Problem 27: Porting applications (KT 7.29)

Some of your friends have recently graduated and started a small company, which they are currently running out of their parents' garages in Santa Clara. They're in the process of porting all their software from an old system to a new, revved-up system; and they're facing the following problem.

They have a collection of n software applications, $\{1, 2, \ldots, n\}$, running on their old system; and they'd like to port some of these to the new system. If they move application i to the new system, they expect a net (monetary) benefit of $b_i \geq 0$. The different software applications interact with one another; if applications i and j have extensive interaction, then the company will incur an expense if they move one of i or j to the new system but not both; let's denote this expense by $x_{ij} \geq 0$.

So, if the situation were really this simple, your friends would just port all n applications, achieving a total benefit of $\sum_i b_i$. Unfortunately, there's a problem. $\cdots$

Due to small but fundamental incompatibilities between the two systems, there's no way to port application 1 to the new system; it will have to remain on the old system. Nevertheless, it might still pay off to port some of the other applications, accruing the associated benefit and incurring the expense of the interaction between applications on different systems.

So this is the question they pose to you: Which of the remaining applications, if any, should be moved? Give a polynomial-time algorithm to find a set $S \subseteq \{2, 3, \ldots, n\}$ for which the sum of the benefits minus the expenses of moving the applications in S to the new system is maximized.

Problem 28: Prescribed flows (CLRS)

Suppose that each source s_i in a multiple source, multisink problem produces exactly p_i units of flow, so that $f(s_i, V) = p_i$. Suppose also that each sink t_j consumes exactly q_j units, so that $f(V, t_i) = q_j$, where $\sum_i P_i = \sum_j q_j$. Show how to convert the problem of finding a flow f that obeys these additional constraints into the problem of finding a maximum flow in a single-source, single-sink flow network.

Problem 29: Projects

Projects $1, 2, \ldots, k$ are available to be undertaken. With each project i is associated a positive revenue r_i. Each project i requires a set S_i of resources to be available, and each resource j, $1 \leq j \leq l$, has an associated cost c_j. However if j is purchased, it is available for any projects for which it is required. Give an algorithm to choose a set of projects so that the associated revenue minus the cost of the required resources is maximized.

Problem 30: Regular graphs (CLRS)

We say that bipartite graph $G = (V, E)$, where $V = L \cup R$, is ***d*-regular** if every vertex $v \in V$ has degree exactly d. Every d-regular bipartite graph has $|L| = |R|$. Prove that every d-regular bipartite graph has a matching of cardinality $|L|$ by arguing that a minimum cut of the corresponding flow network has capacity $|L|$.

Problem 31: Remote Sensors

Devise as efficient as possible algorithm for the following problem. You have n remote sensors s_i, each with a bandwidth requirement r_i and a two-dimensional location (x_i, y_i), that need to be assigned bandwidth on $m < n$ base stations B_j, each with a location (x_j, y_j) and an identical bandwidth capacity C. You are given that no two base-stations are less than 1 km apart (in standard Euclidean distance, $\sqrt{((x_j - x_k)^2 + (y_j - y_k)^2)}$). The assignments can allot sensors bandwidth on multiple base stations. Let $b_{i,j}$ be the amount of bandwidth assigned to sensor s_i on base station B_j. The assignment must meet the following constraints:

- No sensor may be assigned any bandwidth on a base station more than 2 km distance from it, i.e., if the distance from s_i to B_j is greater than 2, $b_{i,j} = 0$.
- The sum of all the bandwidth assigned to any remote sensor s_i must be at least r_i: for each $1 \le i \le n$, $\sum_j b_{i,j} \ge r_i$.
- The sum of all bandwidth assigned on base station B_j must be at most C: for each $1 \le j \le m$, $\sum_i b_{i,j} \le C$.

Your algorithm should find a solution meeting the above constraints if possible, and otherwise output a message saying "No solution exists". (There is an $O(n^3)$ algorithm.)

Problem 32: Rounding (KT 7.39)

You are consulting for an environmental statistics firm. They collect statistics and publish the collected data in a book. The statistics are about populations of different regions in the world and are recorded in multiples of one million. Examples of such statistics would look like the Table 2. We will assume here for

Table 2: Examples of census statistics.

Country	**A**	**B**	**C**	**Total**
grown-up men	11.998	9.083	2.919	24.000
grown-up women	12.983	10.872	3.145	27.000
children	1.019	2.045	0.936	4.000
Total	26.000	22.000	7.000	55.000

simplicity that our data is such that all row and column sums are integers. The Census Rounding Problem is to round all data to integers without changing any row or column sum. Each fractional number can be rounded either up or down. For example, a good rounding for our table data would be as Table 3.

Table 3: Rounding results for census statistics.

Country	**A**	**B**	**C**	**Total**
grown-up men	11.000	10.000	3.000	24.000
grown-up women	13.000	10.000	4.000	27.000
children	2.000	2.000	0.000	4.000
Total	26.000	22.000	7.000	55.000

1. Consider first the special case when all data are between 0 and 1. So you have a matrix of fractional numbers between 0 and 1, and your problem is to round each fraction that is between 0 and 1 to either 0 or 1 without changing the row or column sums. Use a flow computation to check if the desired rounding is possible.
2. Consider the Census Rounding Problem as defined above, where row and column sums are integers, and you want to round each fractional number α to either $\lfloor \alpha \rfloor$ or $\lceil \alpha \rceil$. Use a flow computation to check if the desired rounding is possible.
3. Prove that the rounding we are looking for in (a) and (b) always exists.

Problem 33: Row-Column sums

You are asked to fill the entries of an $n \times n$ matrix by integers between 0 and a bound k, so that the sum of all entries in each row, and each column, comes to one of $2n$ numbers given in advance. For example, the following instance

$$\begin{array}{c c} & \begin{array}{ccc} 17 & 5 & 4 \end{array} \\ \begin{array}{c} 6 \\ 9 \\ 11 \end{array} & \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix} \end{array}$$

with $k = 9$ has solution

$$\begin{array}{c c} & \begin{array}{ccc} 17 & 5 & 4 \end{array} \\ \begin{array}{c} 6 \\ 9 \\ 11 \end{array} & \begin{pmatrix} 6 & 0 & 0 \\ 2 & 3 & 4 \\ 9 & 2 & 0 \end{pmatrix} \end{array}$$

Formulate and solve this problem as a flow problem.

Problem 34: Scheduling in a medical consulting firm (KT 7.19)

You've periodically helped the medical consulting firm Doctors Without Weekends on various hospital scheduling issues, and they've just come to you with a new problem. For each of the next n days, the hospital has determined the number of doctors they want on hand; thus, on day i, they have a requirement that exactly p_i doctors be present.

There are k doctors, and each is asked to provide a list of days on which he or she is willing to work. Thus doctor j provides a set L_j of days on which he or she is willing to work.

The system produced by the consulting firm should take these lists and try to return to each doctor j a list L'_j with the following properties.

(A) L'_j is a subset of L_j, so that doctor j only works on days he or she finds acceptable.

(B) If we consider the whole set of lists $L'_1, \dots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \dots, n$.

1. Describe a polynomial-time algorithm that implements this system. Specifically, give a polynomial-time algorithm that takes the numbers $p_1, p_2 \dots, p_n$, and the lists $L_1, \dots, L_k$, and does one of the following two things.

   - Return lists $L'_1, L'_2, \dots, L'_k$ satisfying properties (A) and (B); or
   - Report (correctly) that there is no set of lists $L'_1, L'_2, \dots, L'_k$ that satisfies both properties (A) and (B).

2. The hospital finds that the doctors tend to submit lists that are much too restrictive, and so it often happens that the system reports (correctly, but unfortunately) that no acceptable set of lists $L'_1, L'_2, \dots, L'_k$ exists.

   Thus the hospital relaxes the requirements as follows. They add a new parameter $c > 0$, and the system now should try to return to each doctor j a list L'_j with the following properties.

   (A^*) L'_j contains at most c days that do not appear on the list L_j.

   (B) (Same as before) If we consider the whole set of lists $L'_1, \dots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \dots, n$.

   Describe a polynomial-time algorithm that implements this revised system. It should take the numbers $p_1, p_2, \dots, p_n$, the lists $L_1, \dots, L_k$, and the parameter $c > 0$, and do one of the following two things.

   - Return lists $L'_1, L'_2, \dots, L'_k$ satisfying properties (A^*) and (B); or
   - Report (correctly) that there is no set of lists $L'_1, L'_2, \dots, L'_k$ that satisfies both properties (A^*) and (B).

Problem 35: Scheduling with deadlines

Suppose you have one machine and a set of n jobs $a_1, a_2, \ldots, a_n$ to process on that machine. Each job a_j has a processing time t_j, a profit p_j, and a deadline d_j. The machine can process only one job at a time, and job a_j must run uninterruptedly for t_j consecutive time units. If job a_j is completed by its deadline d_j, you receive a profit p_j, but if it is completed after its deadline, you receive a profit of 0. Give an efficient algorithm to find a schedule that obtains the maximum amount of profit, assuming that all processing times are integers between 1 and n. Your algorithm must in time polynomial in n.

Problem 36: Shortest augmenting path heuristic

Suppose that the augmenting path algorithm always chooses an augmenting path having as few reverse edges as possible. Prove that the number of augmentations will be $O(mn)$. Give an $O(m)$ algorithm for finding such augmentation.

Problem 37: Spanning subgraph

Given a bipartite graph $G = (V, E)$ and an integer d_v for each node v, does there exist a spanning subgraph H of G such that each node has degree d_v in H. Give an efficient algorithm to answer this question, and also necessary and sufficient conditions for the existence of such a subgraph. A spanning subgraph of $G = (V, E)$ is a subgraph whose vertex set is V and whose edge set is a subset of E.

Problem 38: Sub-trees

One rooted tree T is a *subtree* of another T' if there is a 1-1 mapping from $f : T \leftarrow T'$ so that $f(r_T) = r_{T'}$ and $p(f(x)) = f(p(x))$ for every $x \in T$, where $r_T, r_{T'}$ are the roots of the two trees, and $p(x)$ is the parent of x. In other words, if we delete some nodes in T' and then possibly reorder the children of some nodes, we can get a copy of T. Give an algorithm for deciding whether T is a subtree of T', given two arbitrary rooted trees. (T and T' need not be binary trees, and could have large numbers of children at many nodes.)

Problem 39: Maximizing the benefit of unreachable nodes

We are given a directed graph G, a special node r, and for each directed edge e a non-negative cost c_e of destroying e. If an attacker destroys a set A of edges, she receives a non-negative benefit b_v for each node v that can no longer be reached from r by a directed path. The attacker wants to choose A to maximize the benefit minus the cost. Devise a polynomial-time algorithm, prove its correctness, and determine its complexity.

Problem 40: Updating maximum flow (CLRS)

Let $G = (V, E)$ be a flow network with source s, sink t, and integer capacities. Suppose that we are given a maximum flow in G.

1. Suppose that the capacity of a single edge $(u, v) \in E$ is increased by 1. Give an $O(V + E)$-time algorithm to update the maximum flow.
2. Suppose that capacity of a single edfe $(u, v) \in E$ is decreased by 1. Give an $O(V + E)$-time algorithm to update the maximum flow.

Problem 41: Vertex disjoint paths (CLRS)

An $n \times n$ grid is an undirected graph consisting of n rows and n columns of vertices, as shown in Figure 1. We denote the vertex in the ith row and the jth column by (i, j). All vertices in a grid have exactly four neighbors, except for the boundary vertices,which are the points (i, j) for which $i = 1, i = n, j = 1,$ or $j = n$.

Given $m \leq n^2$ starting points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ in the grid, the ***escape problem*** is to determine whether or not there are m vertex-disjoint paths from the starting points to any m different points on the boundary. For example, the grid in Figure 1(a) has an escape, but the grid in Figure 1(b) does not.

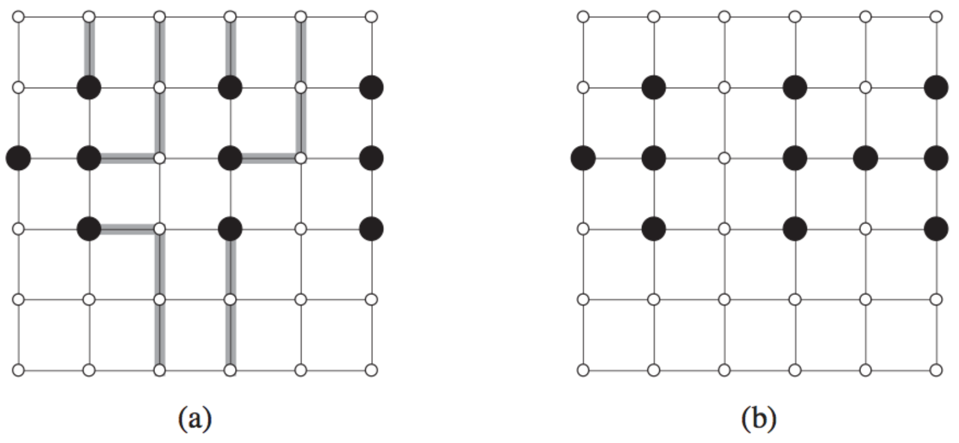

Figure 1: Grids for the escape problem. Starting points are black, and other grid vertices are white. **(a)** A grid with an escape, shown by shaded paths. **(b)** A grid with no escape.

2 Solved problems

Problem 42: Free Standing block of countries

Consider the following definition. We are given a set of n countries that are engaged in trade with one another. For each country i, we have the value s_i of its budget surplus; this number may be positive or negative, with a negative number indicating a deficit. For each pair of countries i, j, we have the total value e_{ij} of all exports from i to j; this number is always nonnegative. We say that a subset S of the countries is free-standing if the sum of the budget surpluses of the countries in S, minus the total value of all exports from countries in S to countries not in S, is nonnegative.

Give a polynomial-time algorithm that takes this data for a set of n countries and decides whether it contains a nonempty free-standing subset that is not equal to the full set.

Solution: Free Standing block of countries

Let $[n]$ represent the list of countries. We are looking for a nontrivial block of countries $C^{'}$ (i.e. $\emptyset \subset C^{'} \subset [n]$) s.t. the cost $\displaystyle\sum_{i\in C',j\notin C'} e_{i,j} - \sum_{i\in C'} s_i$ of the block is ≤ 0. Let $G = (V, E)$ where

$$\begin{aligned} V &= \{s,t\} \cup [n] \\ E &= s \times [n] \cup [n] \times [n] \cup [n] \times t \end{aligned}$$

with edge capacities

$$\begin{aligned} c(s,i) &= \begin{cases} s_i & \text{if } s_i \geq 0 \\ 0 & \text{else} \end{cases} \\ c(i,t) &= \begin{cases} -s_i & \text{if } s_i < 0 \\ 0 & \text{else} \end{cases} \\ c(i,j) &= e_{i,j}. \end{aligned}$$

Let $S = \displaystyle\sum_{i\ni s_i\geq 0} s_i, T = \sum_{i\ni s_i<0} -s_i$, and $C \subseteq V$ be a cut in G. Then $C^{'} = C - \{s\}$ is a set of countries. The capacity of C is

$$\begin{aligned} & \sum_{i\in C',j\notin C'} e_{i,j} + \sum_{i\in C'\ni s_i<0} -s_i + \sum_{i\notin C'\ni s_i\geq 0} s_i \\ = & \sum_{i\in C',j\notin C'} e_{i,j} - \sum_{i\in C'} s_i + S, \end{aligned}$$

which is the cost of the block $C^{'}$ plus the constant S. So there is a block $C^{'}$ with cost ≤ 0 iff G has a cut C with capacity $\leq S$. Using the relabel-to-front algorithm, we can discover whether G has such a cut in time $O(n^3)$.

The only problem is that we have so far ignored the requirement that $C^{'}$ be nontrivial. A naive way to x that would be to, for each $i, j \in [n], i \neq j$, (change the capacity on (s, i) and (j, t) to ∞ and then resolve the above problem). This would force i into the cut and forbid j. But this would take time $O(n^5)$. We can do better.

Let us rst assume that the capacities are rational, which is reasonable since a country usually will not have an irrational surplus nor irrational exports. By multiplying the capacities by the LCM of their denominators, we get an equivalent problem with integer capacities. So from now on, we assume G has integer capacities.

Dene G^j to be G but with the capacity of (j, t) changed to ∞ and increasing the capacity on each (s, i) edge by $\epsilon = \frac{1}{2n}$. This will forbid j from being in any min cut of G^j and provide a small incentive for a min cut to include some $i \in [n]$.

Say a cut C is trivial iff $C = \{s\}$ or $C = V - \{t\}$. We are looking for a nontrivial cut of G with capacity $\leq S$.

Lemma 1. *G has a nontrival cut C with $j \notin C$ and of capacity $\leq S$ iff maxflow(G^j) $< S + \frac{1}{2}$.*

Proof. ($\Rightarrow$) Since C is nontrivial, $C \neq \{s\}$. This and $j \notin C$ imply capacity$_{G^j}(C)$ is capacity$_G(C)$ plus some number $x \in (0, \frac{1}{2})$. So $maxflow(G^j) \leq capacity_{G^j}(C) = capacity_G(C) + x < S + \frac{1}{2}$.

($\Leftarrow$) Let C be a min cut in G^j of capacity $< S + \frac{1}{2}$. Then $C \neq s$ since that cut has capacity $S + \frac{1}{2}$, and $j \notin C$ since otherwise C would have infinit capacity. The fractional part of the capacity of C comes exclusively from edges of the form (s, i), and so $capacity_G(C) \leq S$.

The above lemma implies the correctness of the following $O(n^4)$ time algorithm: for each $j \in [n]$ compute maxflow(G^j). There is a nontrivial free standing block of countries iff for some j, maxflow(G^j) $< S + \frac{1}{2}$. □

Even Better

If $S \leq T$, we can solve the problem in time $O(n^3)$. Let f be a max ow in G and let C be the corresponding min cut. If value(f) $< S$, then C is nontrivial(since capacity$_G(\{s\}) = S$, capacity$_G(V - \{t\}) = T \geq S$), and we are done. If value(f) $> S$, then there is no solution. The only remaining case is value(f) $= S$. For $i, j \in [n], i \neq j$, dene $G^{i,j}$ to be G but with the capacities of (s, i) and (j, t) changed to ∞. The residual graph $G^{i,j}_f$ is the same as G_f except with edges $(s, i), (j, t)$ added (each with capacity ∞).

Let reachable$_G(a, b)$ mean that there is a path in G from a to b. Since f is a max ow in G, $\neg$reachable$_{G_f}(s, t)$. So

$$reachable_{G^{i,j}_f}(s, t) = reachable_{G_f}(s, j) \vee reachable_{G_f}(i, t) \vee reachable_{G_f}(i, j).$$

Given f, the reachable$_{G_f}$ relation can be computed in time $O(n^3)$. Also, maxow($G^{i,j}$) $>$ maxow(G) iff reachable$_{G^{i,j}}(s, t)$.

If maxflow($G^{i,j}$) $=$ maxflow(G), then $G^{i,j}_f$ denes a cut C containing i and not j (and is therefore nontrivial) with capacity$_G(C) = S$ and we are done. Otherwise, we claim that no cut C with capacity$_G(C) \leq S$ contains i and not j. For suppose not. Then capacity$_G(C) = S =$ maxflow(G), and we have the contradiction

$$maxflow(G) < maxflow(G^{i,j}) \leq capacity_{G^{i,j}}(C) = capacity_G(C) = maxflow(G).$$

So

$\exists$ nontrivial cut C with capacity$_G(C) \leq S$

iff $\exists i, j \in [n], i \neq j$ s.t. maxflow($G^{i,j}$) $=$ maxflow(G)

iff $\exists i, j \in [n], i \neq j$ s.t. $\neg$reachable$_{G^{i,j}_f}(s, t)$

iff $\exists i, j \in [n], i \neq j$ s.t. $\neg reachable_{G_f}(s, j) \vee reachable_{G_f}(i, t) \vee reachable_{G_f}(i, j)$, and we can test this last condition in time $O(n^2)$ once the reachable$_{G_f}$ relation is computed. The whole algorithm takes time $O(n^3)$.

This leaves us tantalizingly close to a general $O(n^3)$ time solution. Can the assumption $S \leq T$ be removed, perhaps by articially inating T?

Problem 43: Spanning subgraph

Given a bipartite graph $G = (V, E)$ and an integer d_v for each node v, does there exist a spanning subgraph H of G such that each node has degree d_v in H. Give an efficient algorithm to answer this question, and also necessary and sufficient conditions for the existence of such a subgraph. A spanning subgraph of $G = (V, E)$ is a subgraph whose vertex set is V and whose edge set is a subset of E.

Solution: Spanning subgraph

G is a bipartite graph, thus, we can divide V into two sets, X and Y such that every edge connects a vertice in X to one in Y. We denote the nodes in X as x and the node in Y as y.

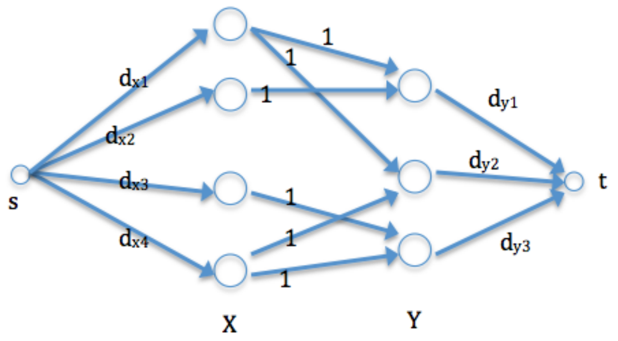

Figure 2: Flow network

Construct the graph $G' = (V', E')$ in Figure 2 where

$$V' = \{s, t\} \cup X \cup Y$$
$$E' = \{s\} \times X \cup E \cup Y \times \{t\}.$$

and with edge capacities

$$c(s, x) = d_x \quad c(x, y) = 1 \quad c(y, t) = d_y.$$

Two conditions must be satisfied in order to get such a spanning subgraph that each node has degree d_v.

1. In the spanning subgraph, sum of the degree of nodes in set X must equal to the sum of the degree of the nodes in set Y, s.t. $\sum_{x \in X} d_x = \sum_{y \in Y} d_y$. Otherwise, there is no such spanning subgraph H, that each node in H has degree d_v.

   This is because in bipartite graph, all the edges come from set X will end at set Y. Consider every edge, it will add one degree to each set X and Y, thus the sum of the degree of nodes in set X equal to the sum of the degree of the nodes in set Y.

2. The maximum-flow of the graph in Figure 2 need to be equal to $\sum_{x \in X} d_x$.

   To prove it is an necessary and sufficient condition for the existence of such a subgraph, we need to prove two aspects.

   - If the maximum-flow equals $\sum_{x \in X} d_x$, we can construct a spanning subgraph H, such that each node in H has degree d_v.

     Proof. For the first statement, because there are $\sum_{x \in X} d_x$ flow coming from source node s, and the maximum-flow equals $\sum_{x \in X} d_x$, thus each node has saturate flow d_x, and according to the conservation property of maximum-flow, d_x flow will output from node x. Plus the capacity of edges out of x is 1, thus, it has d_x edges out of x, which will be the edges in the spanning subgraph. For nodes in set Y, easy to prove it applying the same arguments. □

- If there is a spanning subgraph H, with each node having degree d_v, then we can have a maximum-flow of $\sum_{x\in X} d_x$.

  Proof. First, we construct the same flow graph in Figure 2. We can prove the graph have maximum-flow $\sum_{x\in X} d_x$ using min-cut.

  For each cut $c(A, B)$, where $s \in A$ and $t \in B$, assume there are k nodes of X in the set A, thus, we have $|X| - k$ nodes from set X in the set B, where $0 \leq k \leq |X|$. For the x not in A, the cut will cross the edges between them and s, thus, the capacity of the cut of this part is $\sum_{x\notin A} d_x$. For those x in A, the flow will cross the cut or go to set Y. For those flow go to set Y will finally flow out and cross the cut. The capacity of the nodes not is A, we can have at least $\sum_{x\in A} d_x$ capacity. Thus, we can have the total capacity of the cut $\leq \sum x \in X d_x$. For the base case $k = 0$ or $k = |X|$, we can use the same idea to prove it. Besides, we have only $\sum_{x\in A} d_x$ flow out from s, which is the min-cut.

  Because the capacity of min-cut equals maximum-flow, thus, if there exists a spanning subgraph H, with degree of each node to be d_v, we can construct a maximum-flow $\sum_{x\in X} d_x$. □