**Homework 3**

Wenjun Zhang

A53218995

**Problem 1: Job scheduling (KT 7.41)**

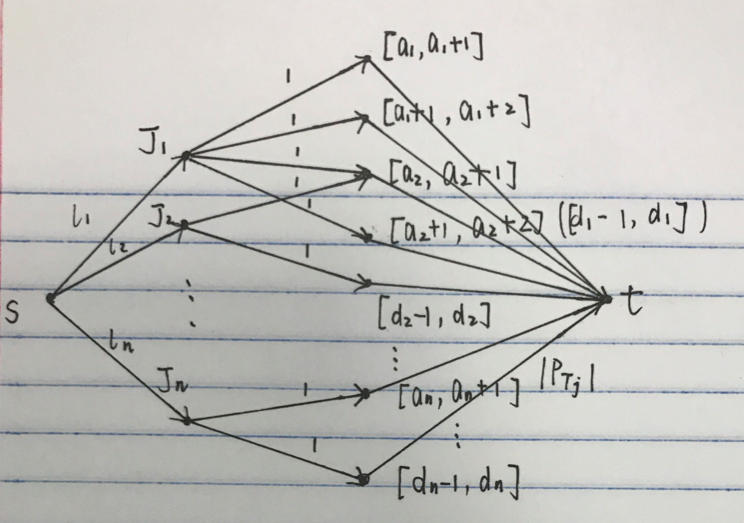
**Algorithm description:**

We are given a set of jobs and their corresponding arrival times, lengths and deadlines, and we also have a collection of processors which are not available all the time. We are asked to manage a job scheduling to check whether the jobs can all be completed or not.

We can change this problem into a max flow problem. Suppose the jobs set we have is . For each , its corresponding arrival time is , deadline is and length is . We also have a set of processors and for each processor it is available during . We also define a set , where the element in set *T* is unit length interval such as .

We can construct a network , where and . in are the source and sink, and for the edge set, we link the source *s* with every job and link every job with their available processing time , and we also link all the available time interval of every job to the sink *t*.

For the capacity of each edge, we define , and , where is number of available processors during . The network we construct is shown as below:



The capacities are all integer, so we can use the Labeling algorithm to find the max flow. If the max flow with value exists, we can manage a job scheduling that the jobs can all be completed before their deadlines. Otherwise, no such scheduling exists.

**Proof of correctness:**

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem.

If there is a flow with value , then the capacities of all edges between the source *s* and every job are used up, meaning that every job is assigned time intervals to process. Next, since every job only links to the source *s*, or links to their available process time intervals with 1 capacity respectively. So every job is finished during their available process time duration. For each time interval, they link to the sink *t* with the capacity equal to the number of available processors during that interval, so during every time interval the number of working processors is bounded by the number of available processors.

If no such flow exists, it means the flow value is smaller than . Since the capacity of edge is , and the flow value of the network cannot exceed . This means that there will be a part of the jobs that cannot be finished, so there isn’t such job scheduling.

Therefore, if and only if such max flow with value exists, there will be a job scheduling ensuring all jobs be completed before their deadlines.

**Time complexity:**

We first need to construct the network, which has at most vertices and edges . So the construction takes . We are using Labeling algorithm to find the max flow, and the time complexity is , where the max flow has the value no greater than . So the total complexity of the algorithm is

**Problem 2: Graph cohesiveness (KT 7.46)**

**Question 1**

**Algorithm description:**

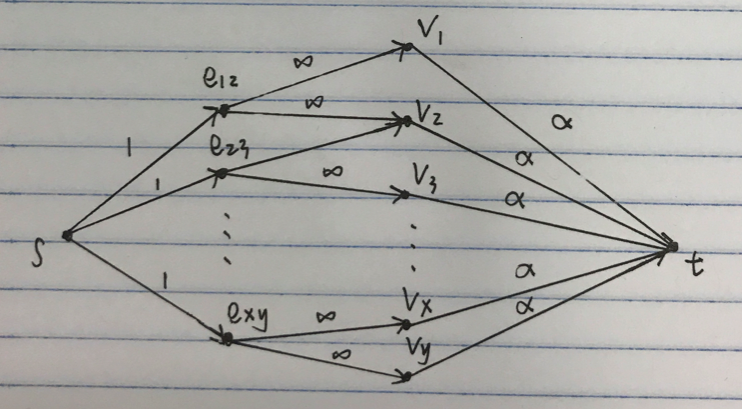
We are given a graph where nodes present people and edges represent friendship and we are asked to give a polynomial-time algorithm that takes a rational number and determines whether there exists a set S with cohesiveness at lease .

We can construct a flow network based on the graph provided in the problem. The vertices and edges of the network are described as below:

For the vertices set, we first include a source and sink as well as the vertices of the provided graph, meaning . Then for every edge , we add a node into .

For the edges set, we link the source *s* to every , and then we link every node to its corresponding vertices and . Besides, we also link every to the sink

For the capacities of edges, we define , and . The network we construct is shown as below:



Given is a rational number, we can apply the Preflow-push method to find max flow. If there exists max flow with value smaller than , which is the number of edges linked to the source *s*, then there is a set *S* with cohesiveness at lease . Otherwise, no such set S exists.

**Proof of correctness:**

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem.

We assume the min cut where and . We first prove that , and must be included in the same set. Since , if we do not include and in the set which belongs to, it will lead to infinite capacity. Similarly, if and are in a particular set, then including will only decrease the cut capacity.

Based on the truth mentioned above, the capacities only consist of 1 and , which can be expressed as , where there are kinds of , so we can also write it as or .

Under the max-flow min-cut theorem, when max flow with value smaller than , , which means

and thus,

On the other hand, we can prove that when max flow with value (since there are only edges linked to the source, the max flow is bounded by ), there won’t exist such a set.

If the set S is still the one of min cut , from the previous description we will have

If the set S is not the one of min cut , let’s assume it is , to achieve

we will have to make , which disobey the fact that being the min cut.

Therefore, if and only if max flow with value smaller than exists, there will be a set *S* with cohesiveness at lease . Otherwise, there won’t.

**Time complexity:**

We first need to construct the network, which has vertices and edges . So the construction takes . The time complexity for FIFO Preflow-push algorithm is So the total complexity of the algorithm is

**Question 2**

**Algorithm description:**

We are asked to give a polynomial time algorithm to find a set S of nodes with maximum cohesiveness.

From the expression of cohesiveness , for the denominator there are choices of value and for the numerator there are choices. So there are choices of rational number in total.

For every rational number derived above, we can start from the largest one to the smallest one and apply the algorithm mentioned in Question 1. Then we can find the maximum cohesiveness.

**Proof of correctness:**

The rational numbers we derive all are the candidate maximum cohesiveness. Since we start from the larger one to the smaller one, we go through all the possible maximum cohesiveness and we can stop after we obtain the maximum cohesiveness.

**Time complexity:**

There are choices of cohesiveness number and time complexity for the algorithm in Question 1 is . So the total complexity of this algorithm is

**Problem 3: Number puzzle**

**Algorithm description:**

We are given sums for each row and column of a matrix and we are asked to reconstruct the matrix.

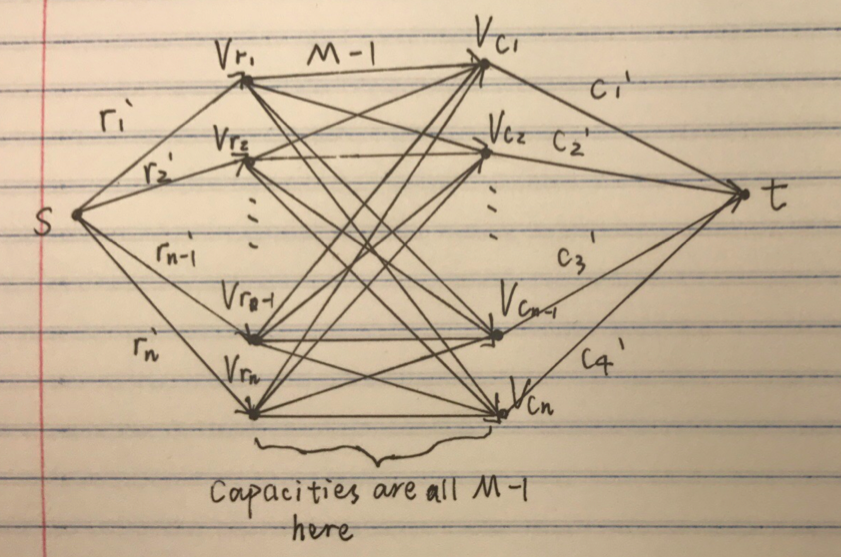
We can convert this problem into a max flow problem. But to begin with, we need to check if and and for .

Since the integers are in the range , to convert this into a max flow network, we first need to reduce every and by for , which means and for . Then we can start to construct the network.

The vertices set of the network can be expressed as , where are the source and sink, represents the *n* reduced row sums and represents the *n* reduced column sums.

For the edges set, . We first link the source with every row sum, and every row sum links to every column sum since there is always a mutual element between them. And we also link every column sum to the sink.

For the capacities of edges, we define , and . The network we construct is shown as below:



The capacities are all integer, so we can use the Labeling algorithm to find the max flow. If the max flow with value exists, we can reconstruct the matrix whose element is equal to the flow on edge plus one. Otherwise, we cannot reconstruct the matrix.

**Proof of correctness:**

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem. But to begin with, we need to prove that making and for in this problem will not influence the correctness of the solution.

The elements of the matrix are in the range . But as we all know in the max flow problem the flow on an edge can be zero, so we need to make each element in the matrix minus 1 to make them in the range . If a matrix whose elements are in the range exists, then by adding 1 to each of its element will lead to the result we want for this problem.

If there is a flow with value , then the capacities of all edges between the source *s* and are used up, meaning that for every row , the sum of row is exactly . Next, since every links to every , it means there will always be a non-negative value passing through to , and it’s the mutual element of row and column, and we can write it as . And since the capacity of edge is , we guarantee that the value of is within the range . Every links to the sink *t* and the capacity for every edge is , because , to achieve a flow with value , the capacity of every edge is used up as well, meaning the column sum of column is exactly .

If no such flow exists, it means the flow value is smaller than , since the capacity of edge is , and the flow value of the network cannot exceed . This means that there will be a part of the rows that don’t sum up to their corresponding , the required matrix cannot be constructed.

Finally, we make , which can ensure that , and since for and for , we have for and for .

Therefore, if and only if such max flow with value exists, we can reconstruct the matrix.

**Time complexity:**

We first need to construct the network, which has vertices and edges . So the construction takes . We are using Ford-Fulkerson algorithm to find the max flow, and the time complexity is , where the max flow has the value no greater than , so it takes . Adding one to every element in the matrix takes . So the total complexity of the algorithm is

**Problem 4: Database projections (KT 7.38)**

**Algorithm description:**

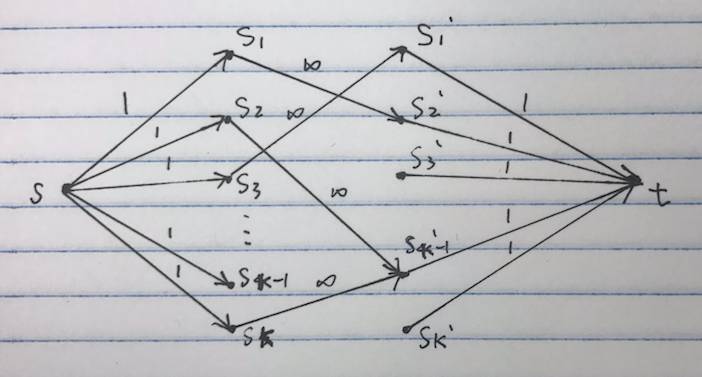
We are given subsets and a parameter and we are asked to find permutations of the columns that for every one of the given subsets, it’s the case that the column in constitute a prefix of at least one of the permuted tables.

We can convert this problem to a max flow problem. We first duplicate a set of size , where for . Then we can construct a network described as below:

For the vertices set, , where are the source and sink, and the and their replicas are included so as to restore the prefix relation.

For the edges set, we first link the source *s* to all subsets and link all the replicas to the sink *t*. Next, if for and , we add an edge to the edges set .

For the capacities of the edges, we define , and . The network we construct is shown as below:



The capacities are all integer, so we can use the Labeling algorithm to find the max flow. If the max flow value is greater than or equal to , there ia a set of valid permutations . Otherwise, no such permutations exist.

**Proof of correctness:**

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem.

Let’s first assume the max flow of the network is . Based on the network we construct, there are edges , meaning there are pairs of where with and . Since , for every pair, we can use the permutation of to represent both of the subsets and . Note that in the example network, we have , there are 3 pairs and we can use one permutation of to represent all of them, which is exactly four minus three. There are subsets in total, to represent all the subsets we will require permutations. Given we can construct a permuted table of size , if the max flow value is greater than or equal to , this means we can represent all the subsets using no more than permutations, which is the same as the requirements of the problem.

If the max flow value is smaller than , let’s assume the max flow is again and we have . We want to construct a valid permuted table with less than permutations, suppose we can use permutations where . To do this we will need pairs of such that with and . If we have that many pairs, the flow of the network will be , which contradicts that fact that the max flow value is . So it’s impossible to use less than permutations to represent the all the subsets, let alone permutations.

Therefore, if and only if the max flow value is greater than or equal to , there are a set of valid permutations .

**Time complexity:**

We first need to construct the network, which has vertices and edges at worst . So the construction takes . We are using Ford-Fulkerson algorithm to find the max flow, and the time complexity is , where the max flow has the value no greater than , so it takes . So the total complexity of the algorithm is

**Problem 5: Maximum likelihood points of failure**

**Algorithm description:**

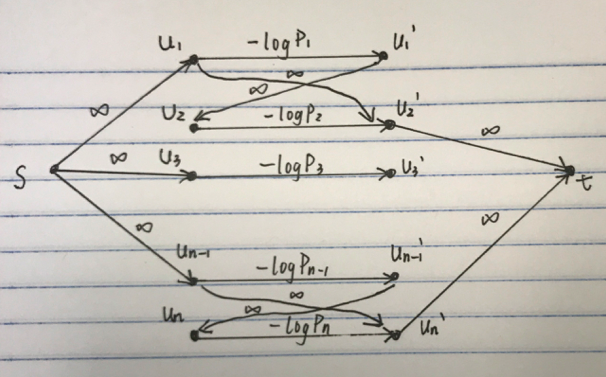
We are given a network and we are asked to find a subset of failure point with the maximum failure probability.

We can convert this problem into a flow problem. We first define a vertices set , which includes the same vertices of except for and , meaning and . Then we duplicate and make . Then we can construct a network described as below:

For the vertices set, , where is the vertices set of the provided network. And we add a replica of the vertices set except for the source and sink to the vertices set.

For the edges set, we first link the source *s* to and link to the destination *t* if there is an edge or for and in the provided edges set . Next, we link to and link to if there is an edge or for and , in the provided edges set . In addition, we also link every for .

For the capacities of the edges, we define every , , and for and , . Since we are looking for a maximum of , to convert it into a min cut problem, we can take the log form. So we assign for . The network we construct is shown as below:



The capacities are probably irrational, so we need to apply Preflow-push method to find the min cut. Suppose the min cut where the source *s* belongs to and the destination *t* belongs to , we assume the set of endpoints of the cutting border edges is . The nodes included in corresponds to a set of original nodes in the provided vertices set V, which is the subset of failure point with the maximum failure probability.

**Proof of correctness:**

To prove the correctness of the algorithm, we need to show that the min cut solution corresponds to the solution of the original problem.

First we need to prove that the min cut disconnect the *t* from *s*. This is actually quite obvious due to the definition of a cut, which is a set of edges whose removal will divide the network into two separate parts.

Next, since only the capacities of edges have finite value, the cut only includes edges among . Otherwise, the cut will have value infinity. Let’s assume the min cut value is , and this means the capacity of the min cut is , which can be expressed as

Since is the minimum and log function is monotonically increasing, term is the maximum, and is the subset of failure point with the maximum failure probability.

**Time complexity:**

We first need to construct the network, which has vertices and edges . So the construction takes . We use FIFO preflow-push algorithm, whose time complexity is , so it takes . Thus, the total complexity of the algorithm is