CS 170 HW 8

Due **2021-10-25, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 (Max-Flow) A cohort of spies

A cohort of k spies resident in a certain country needs escape routes in case of an emergency. They will be travelling using the railway system which we can think of as a directed graph $G = (V, E)$ with V being the cities. Each spy i has a starting point $s_i \in V$, all s_i's are distinct. Every spy needs to reach the consulate of a friendly nation; these consulates are in a known set of cities $T \subseteq V$. In order to move undetected, the spies agree that at most c of them should ever pass through any one city. Our goal is to find a set of paths, one for each of the spies (or detect that the requirements cannot be met).

Model this problem as a flow network. Specify the vertices, edges and capacities, and show that a maximum flow in your network can be transformed into an optimal solution for the original problem. You do not need to explain how to solve the max-flow instance itself.

3 (Max-Flow LP) Min Cost Flow

In the max flow problem, we just wanted to see how much flow we could send between a source and a sink. But in general, we would like to model the fact that shipping flow takes money. More precisely, we are given a directed graph G with source s, sink t, costs l_e, capacities c_e, and a flow value F. We want to find a nonnegative flow f with minimum cost, that is $\sum_e l_e f_e$, that respects the capacities and ships F units of flow from s to t.

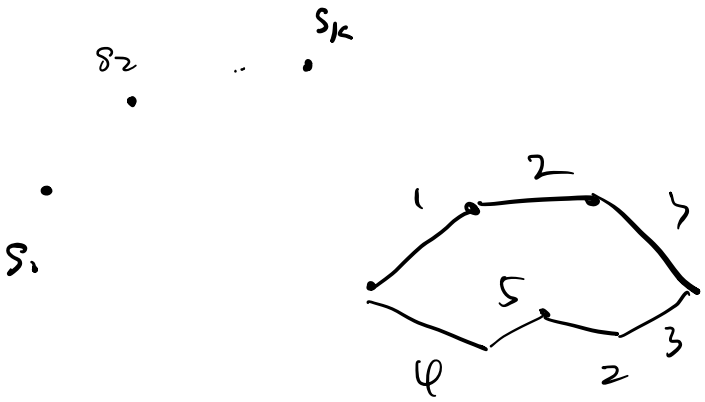
(a) Show that the minimum cost flow problem can be solved in polynomial time.

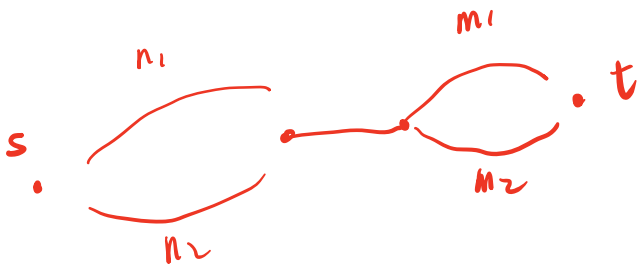
(b) Show that the shortest path problem can be solved using the minimum cost flow problem

(c) Show that the maximum flow problem can be solved using the minimum cost flow problem.

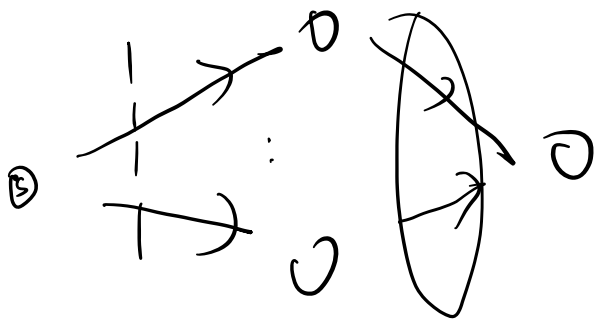
4 Faster Maximum Flow

In the class, we see that the Ford-Fulkerson algorithm computes the maximum flow in $O(mF)$ time, where F is the max flow value. The run-time can be very high for large F. In this question, we explore a polynomial-time algorithm whose run-time does not the depend on





min cut $O(n)$



after m iteration, it's exhausted

the flow value. Recall that Ford-Fulkerson provides a general recipe of designing max flow algorithm, based on the idea of an *augmenting path* in a residual graph:

Algorithm 1 Ford-Fulkerson

while there exists an augmenting path in G_f **do**
  Find an arbitrary augmenting path P from s to t
  Augment flow f along P
  Update G_f

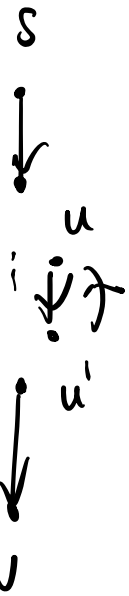
This problem asks you to consider a specific implementation of the algorithm above, where each iteration we find the augmenting path with the smallest number of edges and augment along it.

Algorithm 2 Fast Max Flow

while there exists an augmenting path in G_f **do**
  Find the augmenting path P from s to t with the smallest number of edges
  Augment flow f along P
  Update G_f

We first show that each iteration can be implemented efficiently. Then we analyze the number of iterations required to terminate. Throughout, we define the shortest path from u to v as the path from u to v with the smallest number of edges (instead of the smallest sum of edge capacities). Consequently, we define distance $d(u, v)$ from u to v as the number of edges in the shortest path from u to v. construct a graph with weight =1

(a) Show that given G_f, the augmenting path P from s to t with the smallest number of edges can be found in $O(m + n)$ time.
Hint: Use the most basic graph algorithm you know. BFS



(b) Show that the distance from s to v in G_f never decreases throughout the algorithm, for any v (including t). BFS layer
Hint: Consider what augmentation does to the "forward edges" going from u to u', where $d(s, u') = d(s, u) + 1$. shortest path add back edge

(c) Show that with every m flow augmentations, the distance from s to t must increase by (at least) 1. edges
Hint: Observe that each flow augmentation removes at least one forward edge.

(d) Conclude by proving that the total number of flow augmentations this algorithm performs is at most $O(mn)$. Analyze the total run-time of the algorithm.
Hint: You may observe that the distance from s to t can increase at most $O(n)$ times throughout the algorithm. If you use this fact, explain why it holds.

5 Jumbled Sequences

You are measuring temperatures in a few different places, and receiving temperatures remotely. You have received a sequence of n temperatures, from a total of k thermometers;

unfortunately, they are all jumbled together, and you don't know which temperatures are from which thermometer.

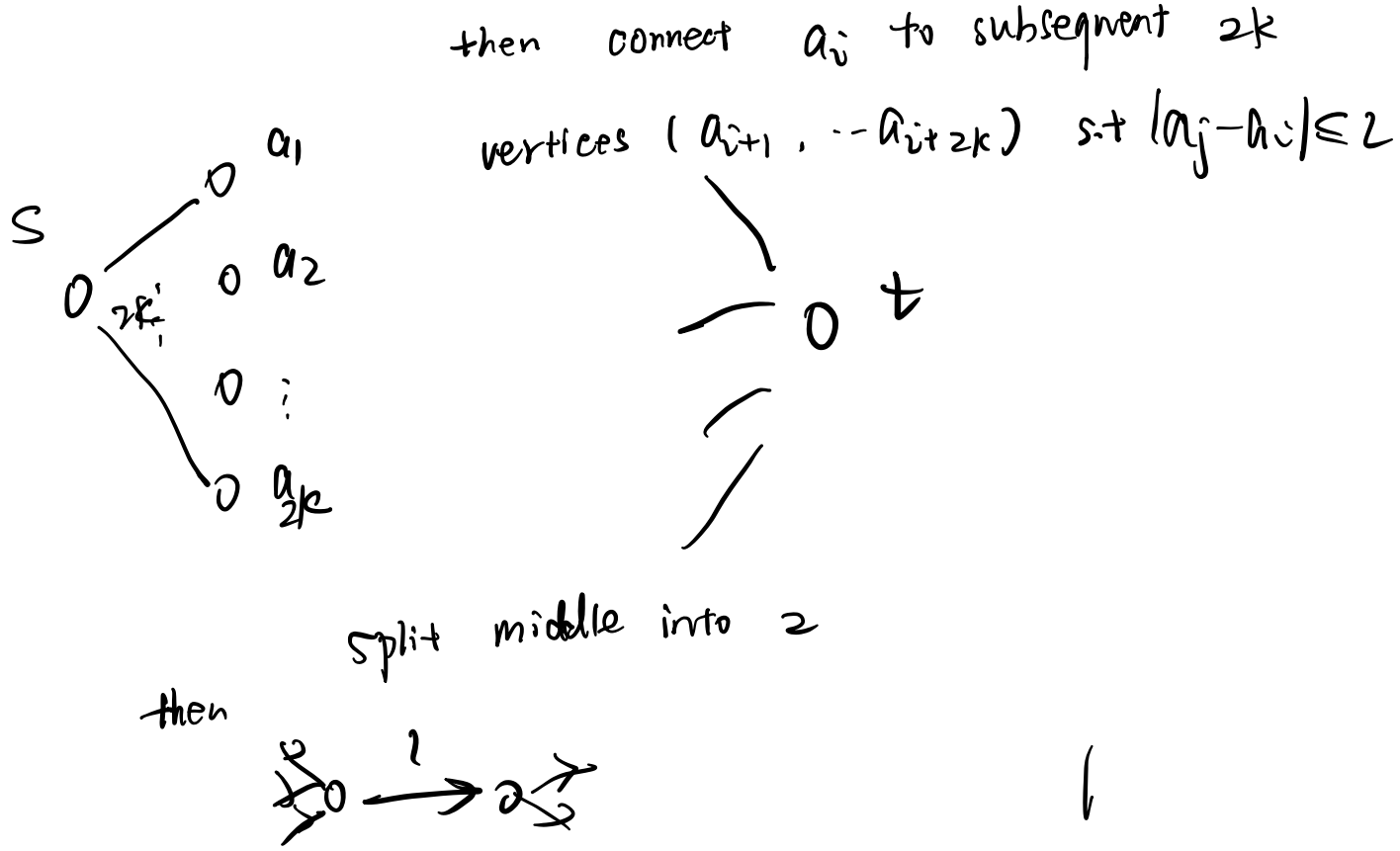
Thankfully, you don't really care, and so are content to simply extract k disjoint 'plausible' subsequences from your original sequence of measurements (disjoint meaning each element in the original sequence belongs to at most 1 subsequence). A subsequence is 'plausible' if it satisfies 3 conditions:

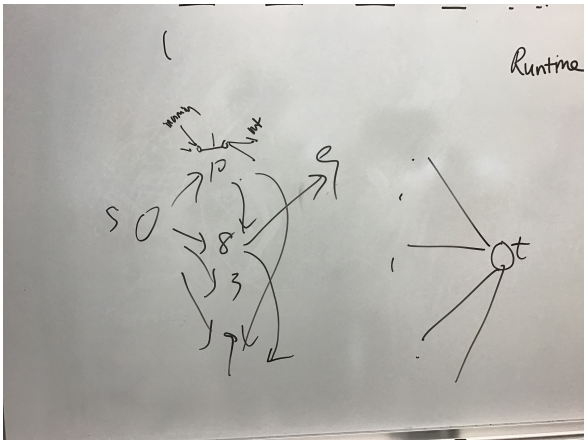
1. it consists of elements from the original sequence in their original order
2. any two consecutive elements in the subsequence have no more than $2k$ elements between them
3. the first element in the subsequence has no more than $2k$ elements before it in the original sequence, and the last element has no more than $2k$ elements after it
4. any two consecutive elements in the subsequence differ in value by no more than 2

As an example, if you receive the sequence $(10, 8, 3, 9, 4, 9, 5, 6)$, you can extract 2 disjoint plausible subsequences from it: $(10, 8, 9, 9)$ and $(3, 4, 5, 6)$. Note that $(10, 8, 6)$ is not a plausible subsequence, because there are 5 elements between 8 and 6 in the original sequence.

Give an algorithm which, given a *specific* k, determines in $O(nk^2)$ time whether it is possible to find k disjoint plausible subsequences.

Describe your algorithm and give a runtime analysis; no proof of correctness is necessary.





2k outgoing edges

↑

$O(mF)$

↓

$N(2k+1)$

s

s

t

t

s

Study Group

Shenghan Zheng 3037534676

2 (Max-Flow) a cohort of spies

Main Idea

Set all cities as vertices

First, denote each spy's path as a unit flow. We want a set of paths for each spy to go from source s to any vertex in T.

To achieve this, we add a source s and add (s,s_i) to the edge set and set their capacity to be 1. Likewise, we add a sink vertex t and add (p,t) to the edge set for every vertex in T(denoted by p) and assign their value to be positive infinity. Then, we split each vertex into v_in and v_out. v_in has all incoming edges of v while v_out has all outgoing edges of v. Assign the capacity of (v_in,v_out) to be c.

Since the source has k outgoing edges with same capacity of 1. The maximum flow should be less than k. Run Ford-Fulkerson to solve this problem. If the max flow is less than k. It indicates that the requirement can't be met. If the max flow is k, then we find a set of paths for each spy that satisfies the requirement.

3 (Max Flow LP) Min Cost Flow

(a)

Construct linear program

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j)\in E} c_{ij} f_{ij} \\ \text{subject to} & \sum_{\{k|(j,k)\in E\}} f_{jk} - \sum_{\{i|(i,j)\in E\}} f_{ij} = b_j, \qquad \forall j \in V \\ & 0 \leq f_{ij} \leq u_{ij}, \qquad \forall (i,j) \in E. \end{array}$$

Since the linear program can be solved in polynomial time, the minimum cost flow problem can be solved in polynomial time

(b)

Let’s say we want to find the shortest path from p to q

Set capacity to each edge to be positive infinity(ignore capacity)

Set each edge's cost to be its distance(or weight)

Add a source s and add (s,p) to E and the capacity and cost of (s,p) is 1. Add a sink t and add (q,t) to E and the capacity and cost of (q,t) is 1

Now run Ford-Folkerson for (s,t), the result is the minimum path from p to q

Proof

After modifying, the objective function becomes: sum of weight + 2

If we find a flow with the minimum value, we have the minimum path from p to q. Because the capacity for the source is only 1 and min flow means min distance.

(c)

Set cost for all edges to be 0

Add (t,s) to E with positive infinite capacity and -1 cost

Now run Ford-Folkerson for (s,t)

Proof:

The augmented path will be continuously added to the flow, since going through (t,s) is the only way to reduce the cost from s to t. The algorithm will end when there is no augmented path which is the end of the max-flow problem

4 Faster Maximum Flow

(a)

Main Idea

1. Add a new weight 1 to each edge
2. Run BFS according to the new edge with root s

Correctness

BFS will guarantee that the height of t is minimum(blackbox). Thus, it's the shortest distance path by definition

Runtime

add new weight O(m)

Run BFS O(m+n)

total runtime O(m+n)

(b)

Proof

After running each augmentation in the graph. We will remove at least one forward edge from s to t and add a back edge. Since the back edge only go up 1 layer, it will not decrease the distance from s to t. Otherwise, suppose we get a smaller distance by using this back edge(named (p,q)), then q has a smaller distance compared to the vertex used in the minimum path at the same layer. Contradiction!

(c)

Proof

Notice that every time we do the flow augmentation, one forward edge will be removed and we will replace it with a reverse edge.

Lemma: If we add previous deleted edge from after one augmentation and include it in the shortest path, then our distance add 1.

Let's say edge (u,v) is critical if it's the edge to be removed in the residual graph. u, v are in the layer i, i+1 respectively of BFS tree. To make (u, v) add to the augmenting path again, u must appear in a level higher then v since (v, u) should be the saturated edge included in the augmenting path to make it reverse.

Since the distance from s to v never decreases, the level of v will be equal or greater than i+1. Thus, u must rise to level at least i+2. Thus, after adding (u, v) again, our distance add at least 1

Back to the problem, if none of the edges will be added again in the original graph, it indicates that we remove all m edges one time and reverse them. After m-1 iterations, we have 1 outgoing edge from s that isn't reversed. Otherwise we can't reach t.That means the last path is directly from s to t but its distance is 1 which shows that the distance of previous m-1 paths is 1 which is impossible. Thus, we must add a edge that is previously deleted in the graph. According to the lemma, the statement holds

(d)

From c) we know that the distance of u will increase at least 2 since its level changes from i to at least i+2. Notice that the BFS tree has no levels numbered above n. Thus, the total time that one edge occur as a critical edge is at most n/2. There are at most 2m edges(including the reverse edge) will appear during the iterations. Since we handle at least one critical edge for 1 iteration, we will do O(mn) augmentations

5 Jumbled Sequences

Main Idea

1. store the sequence as vertices with their values
2. Add a source s and connect it with the first 2k vertices. Add a sink t and add (p,t) in E(p are all last 2k vertices). All edges with capacity 1
3. For the n vertices(except s,t),

   For i in range(n):

   For j in range(2k):

   if |a_i - a_(i+j)|<=2:

   add (a_i, a_(i+j)) to E
4. For the n vertices(except s,t), split each one of them into 2 vertices v_in, v_out such that v_in has all the incoming edges while v_out has all the outgoing edges. Add (v_in, v_out) to E with capacity 1.
5. Run Ford-Fulkerson

Runtime

Construct the graph is O(nk)

s has 2k outgoing edges. t has 2k incoming edges. For any of the n vertices, each of them has at most 2k edges. The split of vertices will add n edges. Thus, the total number of edges is at most 4k+2kn+n

The max flow is less than 2k since s only has 2k outgoing edges with each capacity equals 1.

Total runtime = O(2k(4k+2kn+n)+nk) = O(nk^2)