CS 170 Homework 4

Due 2021-09-27, 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  Running Errands

You need to run a set of \( k \) errands in Berkeley. Berkeley is represented as a directed weighted graph \( G \), where each vertex \( v \) is a location in Berkeley, and there is an edge \((u, v)\) with weight \( w_{uv} \) if it takes \( w_{uv} \) minutes to go from \( u \) to \( v \). The errands must be completed in order, we’ll assume the \( i \)th errand can be completed immediately upon visiting any vertex in the set \( S_i \) (for example, if you need to buy snacks, you could do it at any grocery store). Your home in Berkeley is the vertex \( h \).

Given \( G, h, \) and all \( S_i \) as input, given an efficient algorithm that computes the time needed to complete all the errands starting at \( h \). That is, find the shortest path in \( G \) that starts at \( h \) and passes through a vertex in \( S_1 \), then a vertex in \( S_2 \), then in \( S_3 \), etc.

Given \( G, h, \) and all \( S_i \) as input, given an efficient algorithm that computes the time needed to complete all the errands starting at \( h \). That is, find the shortest path in \( G \) that starts at \( h \) and passes through a vertex in \( S_1 \), then a vertex in \( S_2 \), then in \( S_3 \), etc.

Give a 3-part solution.

Solution: Main idea Create \( k + 1 \) copies of \( G \), called \( G_0, G_1, ...G_k \), to form \( G' \). Let the copy of \( v \) in \( G_i \) be \( v_i \). For every \( v \) in \( S_i \), we add an edge from \( v_{i-1} \) to \( v_i \) with weight 0. We run Dijkstra’s starting from \( h_0 \) in \( G' \), and output the shortest path length to any vertex in \( G_k \).

Correctness Any path in \( G' \) from \( h_0 \) to a vertex \( G_k \) can be mapped to a path in \( G \) of the same length passing through vertices, by taking each edge \((u_i, v_i)\) and replacing it with the edge \((u, v)\) in \( G \), ignoring edges of the form \((v_{i-1}, v_i)\). These paths must also complete the errands in order, since they must contain edges of the forms \((v_0, v_1), (v_1, v_2), ... \) in that order.

Runtime analysis This takes time \( O(k(|V| + |E|) \log k|V|) \) since the new graph is \( k \) times the size of the original graph.

3  MST Variant

Give an undirected graph \( G = (V, E \cup S) \) with edge weight \( c(e) \). Note that \( S \) is disjoint with \( E \). Design an algorithm to find a minimum one among all spanning trees having at most one edge from \( S \) and others from \( E \).

Input: A graph \( G = (V, E \cup S) \), and a cost function \( c(e) \) defined for every \( e \in E \cup S \).

Output: A tree \( T = (V, E') \) such that \( T \) is connected (there is a path in \( T \) between any two vertices in \( V \)), \( E' \subseteq E \cup S \), \( \sum_{e \in E'} c(e) \) is minimized, and \( |E' \cap S| \leq 1 \).
Give a 3-part solution.

Solution:

Main Idea: Use Prim’s (or Kruskal’s) algorithm to find the MST $T'$ of $G$. Then for each potential $s \in S$, try adding $s$ to $T'$ and keep track of the difference between $c(s)$ and $c(e)$ where $e$ is the edge of highest weight along that cycle. If $c(s) < c(e)$, then adding $s$ to $T'$ and removing $e$ from $T'$ creates a new spanning tree with lower total weight than $T'$. Doing this for each $s \in S$ and taking the smallest resulting spanning tree gives us $T$.

Pseudocode:

```plaintext
procedure MSTVariant(graph G, additional edges S, cost function c)
    Use Prim’s to find $T' = (V, E')$, the MST of $G$
    bestDifference ← 0
    bestEdges ← $E'$
    for $s \in S$ do
        Add $s$ to $T'$ and use DFS to find the cycle containing $s$. Let $e$ be the most costly edge in that cycle.
        if $c(e) - c(s) > bestDifference$ then
            bestDifference ← $c(e) - c(s)$
            bestEdges ← $E' \cup \{s\} - \{e\}$
    return $(V, bestEdges)$
```

Running time analysis: Prim’s takes $O(|E| \log |E|)$ to find the MST. For each edge in $S$, we run a DFS on $T'$, which has $|V|$ vertices and $O(|V|)$ edges. Therefore, this takes $O(|S||V|)$ time. Thus, the overall running time is $O(|E| \log |E| + |S||V|)$.

Proof of correctness: The optimal solution containing exactly one edge from $S$ cannot be better than the optimal solution containing exactly one edge from $S$ and exactly $|V| - 2$ edges from $T'$, the MST of $G$. To see this, assume towards contradiction that the optimal solution $T$ contains some $e \in E$ such that $e$ is not part of $T'$. Consider the cut in $T$ defined by $e$, i.e. if $e = (u, v)$, then divide the vertices into those reachable from $u$ and those reachable from $v$ without using $e$. Let $e'$ be the edge of lowest weight in $T'$ that spans that cut. Adding $e'$ to $T$ creates a cycle with $e$. We know that $c(e') \leq c(e)$ because otherwise $e$ would be in $T'$ instead of $e'$. Therefore, removing $e$ from $T$ and adding $e'$ yields a spanning tree that costs no more than $T$. Since we can do this for any edge in $T$ but not in $T'$, we can effectively transform $T$ into a solution that is at least as good and contains no edges from $E$ that are not in $T'$.

4 Blackout in the Igloos

It’s the year 3077, exactly 1000 years after Cyberpunk happened. PNPenguins live in igloos that are equipped with one-directional portals, allowing them to instantly travel from one igloo to another. It is polar night now, so lights are necessary for PNPenguins to navigate inside their igloos – without light, igloos are completely dark inside. There are $n$ igloos in total, and in each igloo, there are $k$ one-directional portals numbered $\{1 \ldots k\}$. Each portal can teleport PNPenguins to a certain igloo (including the one that this portal is located).

Recently, PG&E (Penguin Gaming and Electric Company - yes, gaming is considered a utility in 3077) issued a notice indicating power could be shut off at any time. PNPenguins
want to buy an emergency power generator for one igloo, and in the case of a power shut off, all PNPenguins need to rendezvous there. PNPenguins are terrible at memorizing which igloo they are in, but they all have a very good sense of direction. When the power is shut off at an igloo, they still can choose any portal and travel through it to another igloo.

In the event of a power shut off, PNPenguins need a way to know how to get to the igloo with the power generator. Thus, there has to be a fixed sequence of numbers $p_1, ..., p_l$ such that starting from any igloo, if a PNPenguin goes into portals $p_1, ..., p_l$, it ends up at the igloo with the power generator.

PNPenguins are wondering, if there exists any fixed sequence of portal numbers $p_1, ..., p_l$, such that regardless of the igloo they are currently in, after going through these portals, they end up in the same igloo? They only need to know if such a sequence of portals exist without knowing the sequence itself.

More formally, there are $k$ functions $f_1, ..., f_k$ and each function maps each igloo to an igloo (possible the same). Your goal is to verify if there exists a sequence of numbers $p_1, ..., p_l$ such that $f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}$ outputs the same igloo regardless of the igloo a PNPenguin starts from, i.e.,

$$\forall x, y \in \{1 \ldots n\} f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}(x) = f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}(y)$$

$\circ$ is function composition:

$$f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}(x) = f_{p_1}(f_{p_2}(...(f_{p_{l-1}}(f_{p_l}(x))))))$$

(a) Provide a 3-part solution for the described problem.

(b) Go to the online contest system at [https://hellfire.ocf.berkeley.edu/](https://hellfire.ocf.berkeley.edu/), access the HW4 contest and provide a program to solve this problem.

**Hint** First, think about just two igloos. Can you find an algorithm that will find if there is a sequence of doors that will map these two igloos to the same igloo? Then, try to generalize this to all igloos by “merging” together pairs of igloos. Based on this, try to find an "if and only if" condition for the existence of a sequence of doors that will map all igloos to the same igloo. Can you develop an algorithm to verify if this "if and only if" condition holds?

**Note** that there are several ways to solve this problem with different time complexities, ranging from a very slow to a very fast solution. This is a hard problem and we do not expect everyone to get to the best possible solution. Because of this, we will be awarding partial credit for slower solution, even for a solution that is exponential in the input size. Try to do your best. We will be providing Office Hours support for this question, but note that we will emphasize conceptual help and we do not guarantee that a staff member will be able to debug your code.

**Solution:**

### 4.1 Slow Solution $O(2^n nk)$

**Main Idea:**

We construct a graph of nodes that represent all the possible subsets of igloos. This can be represented as the graph of all binary strings of size $n$, where a binary string has a 1 on position $i$ if igloo $i$ is in the subset and it has a 0 on that position otherwise.
Each node has $k$ outgoing edges, one edge for each portal. Edge $E_i^A$ connects a node $A$ to another node $B$ such that $B$ is the subset of igloos to which we can get from the subset of igloos represented by node $A$ using portal $i$. More formally, $j \in A \iff f_k(j) \in B$.

By this construction, we get a directed graph on $2^n$ nodes where each node has $k$ outgoing edges. Then, we run BFS (or DFS) from the node that represents the set of all igloos (in the binary string representation, it would be the binary string '111...111') and check if during the traversal, we reach a node that represents a subset of exactly one igloo (as a binary string, it would be a string that contains exactly one 1 and all other 0's). If we reached such a node, output YES, otherwise, output NO.

**Proof of Correctness:**

If there is a path on the constructed graph from node '111...111' to a subset that corresponds to one igloo, then an escape sequence of portals to a single igloo must exist. We can see this by composing the functions corresponding to each edge of the path: the final composition of functions will be the escape sequence we are looking for. More formally, if the path is $(E_i^{A_1}, ..., E_i^{A_p})$, then $f_{i_p} \circ \cdots \circ f_{i_1}$ is the sequence of portals that penguins can always take to reach a single igloo.

On the other hand, if there is such a sequence of portals that leads from any igloo to one specific igloo, then the set of edges corresponding to these portals is a path from the node '111...111' to the node corresponding to exactly one igloo. Hence, there exists some sequence of portals to a single igloo if and only if we reach a node representing a single igloo in the graph.

**Runtime Analysis:**

The graph will contain $2^n$ nodes and $2^n k$ edges. Constructing each edge takes $O(n)$ time since to construct the edge from a node $A$, we have to apply some function $f_i$ to all igloos in $A$, and there are $O(n)$ igloos in $A$. Thus, constructing the graph will take $O(2^n nk)$ time. Then, running BFS or DFS on it is just $O(2^n k)$ time, so the whole algorithm takes $O(2^n nk)$ time.

**4.2 Faster Solution $O(n^3 k)$**

**Main Idea:**

This has the same main idea as the fastest solution, but we run DFS on each $(s, s)$ node instead of running DFS once on a dummy node.

**Proof of Correctness:**

See the proof of correctness below.

**Runtime Analysis:**

We run DFS $n$ on a graph with $O(n^2)$ vertices and $O(n^2 k)$ edges. Constructing the graph takes us linear time or $O(n^2 + n^2 k)$. This gives us a final runtime of $n \ast O(n^2(k + 1)) = O(n^3 k)$.

**4.3 Fastest Solution $O(n^2 k)$**

**Main Idea:**

Let $v = f_i(u)$ represent the igloo that penguins reach after going through the $i$-th portal in igloo $u$. 
We create a graph where the vertices of the graph represent pairs of igloos. There exists an edge from \((a^{(1)}, a^{(2)})\) to \((b^{(1)}, b^{(2)})\) if there exists some portal numbered \(i\) such that \(f_i(b^{(1)}) = a^{(1)}\) and \(f_i(b^{(2)}) = a^{(2)}\). We also attach a dummy node \(q\) to all nodes \((s, s)\) where \(s = s\).

Starting from \(q\), we now run DFS on this graph. If all vertices have been visited, a fixed sequence of portal numbers to some igloo exists. Otherwise, such a sequence does not exist in our graph.

**Proof of Correctness:**

Recall that if we can reach \((s, s)\) from every pair of vertices \((a_i^{(1)}, a_i^{(2)})\), that means there must exist some edge from \((a_k^{(1)}, a_k^{(2)})\) to \((s, s)\); in other words, there exists a single portal \(i\) such that \(f_i(a_k^{(1)}) = s\) and \(f_i(a_k^{(2)}) = s\) for some igloos \(a_k^{(1)}\) and \(a_k^{(2)}\).

There must exist some sequence of portals from \((a_k^{(1)}, a_k^{(2)})\) to \((s, s)\).

We now show that it sees that if all pairs of igloos can resolve to the same value \((s, s)\) (i.e., has a path from \((s, s)\)), we can create a escape sequence of portals to a single igloo. Let \(F_1\) be some sequence of portal that resolves igloos \(s_1, s_2\) into the same state \(F_1(s_1)\). Let \(F_2\) be the function that resolves \(F_1(s_1)\) and \(F_1(s_3)\) into the same igloo \(F_2 * F_1(s_1)\). Let \(F_j\) be the function that resolves \(F_{j-1} \circ ... \circ F_1(s_1), F_{j-1} \circ ... \circ F_1(s_{j+1})\). We can see that the composition of functions \(F_{j-1} \circ ... \circ F_1\) resolves the igloos \(s_1\) to \(s_{j+1}\) to the same igloo.

Now, with this construction, \(F_x \circ ... \circ F_1\), which is the sequence of portals that we get when \(F_x \circ ... \circ F_1(s_j) = F_x \circ ... \circ F_1(s_1)\) for all \(j\), is some escape sequence if \(a_i^{(1)}, a_i^{(2)}\) resolve to the same value for all \(i\). Note that if there exists some path from a vertex \((s, s)\) to all other vertices, \(F_i\) must exist for \(i \in [1, x]\).

Hence, if we can reach the set of vertices \(\{(s, s)\forall s \in V\}\) from every pair of vertices \((a_i^{(1)}, a_i^{(2)})\), an escape sequence must exist.

The other direction also holds as follows. Say that an escape sequence does not exist. Then, there must exist at least two igloos \(a^{(1)}\) and \(a^{(2)}\) that cannot be resolved to some igloo \(s\)–in other words, it is not possible for a penguin in \(a^{(1)}\) and a penguin in \(a^{(2)}\) to meet if they take the same portals in the same sequence from their igloos. That means we can not resolve the vertex representing that pair of igloos \((a^{(1)}, a^{(2)})\) to any vertex of the form \((x, x)\) in our graph, since a sequence of portals that would bring penguins in \(a^{(1)}\) and \(a^{(2)}\) to the same igloo does not exist.

That means that our algorithm would find that the vertex \((a^{(1)}, a^{(2)})\) has not been reached and hence conclude that no universal escape sequence exists.

**Runtime Analysis:**

We run DFS on a graph with \(O(n^2)\) vertices and \(O(n^2k)\) edges. Constructing the graph takes us linear time or \(O(n^2 + n^2k)\). This gives us a final runtime of \(O(n^2(k + 1)) = O(n^2k)\).