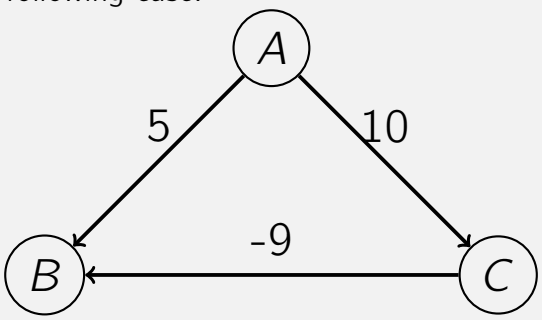
CS 161 (Stanford, Fall 2025) Section 5 Extra

1 True or False

1. Djikstra's can be used for successfully finding the shortest path from a source to all other vertices in a graph with negative edges, but not with negative cycles.

Solution

False. Consider the following case:



Once Dijkstra has marked node B as "I'm sure" with cost 5, it will not try to find another path to B. Thus, it will never update the cost to 1.

2. To find the shortest path from one vertex to another in an unweighted graph, you should use Dijkstra's algorithm as it is the most efficient solution.

Solution

False. For an unweighted graph, you should use BFS. It is more efficient because it does not need a data structure that can handle fast operations of type extractMin() and decreaseKey().

3. Adding a new positive edge to an undirected weighted graph with positive edges cannot lead to the output values of Dijkstra's increasing.

Solution

True. Adding a new edge to the graph can have the following possibilities:

(a) If the edge is in the new shortest path to a vertex v from source s, then the minimum cost will reduce.

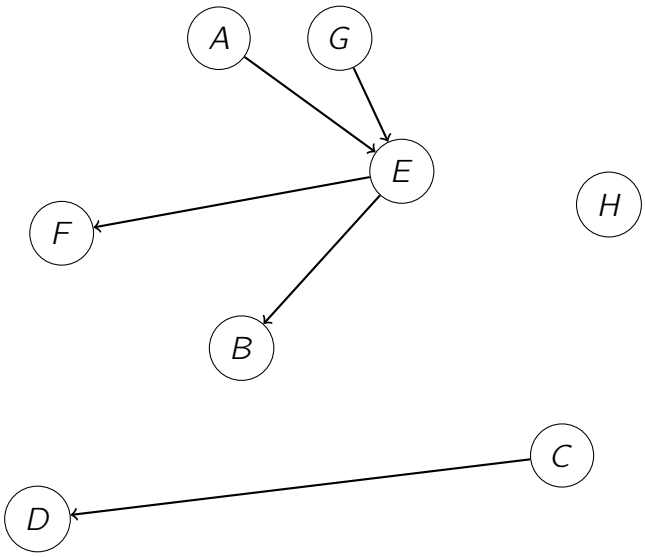
(b) If the edge is not in the new shortest path to a vertex v from source s, then the minimum cost will not be affected.

In either case, the cost of the shortest path can only decrease.

2 Hyperlinks can go backward?

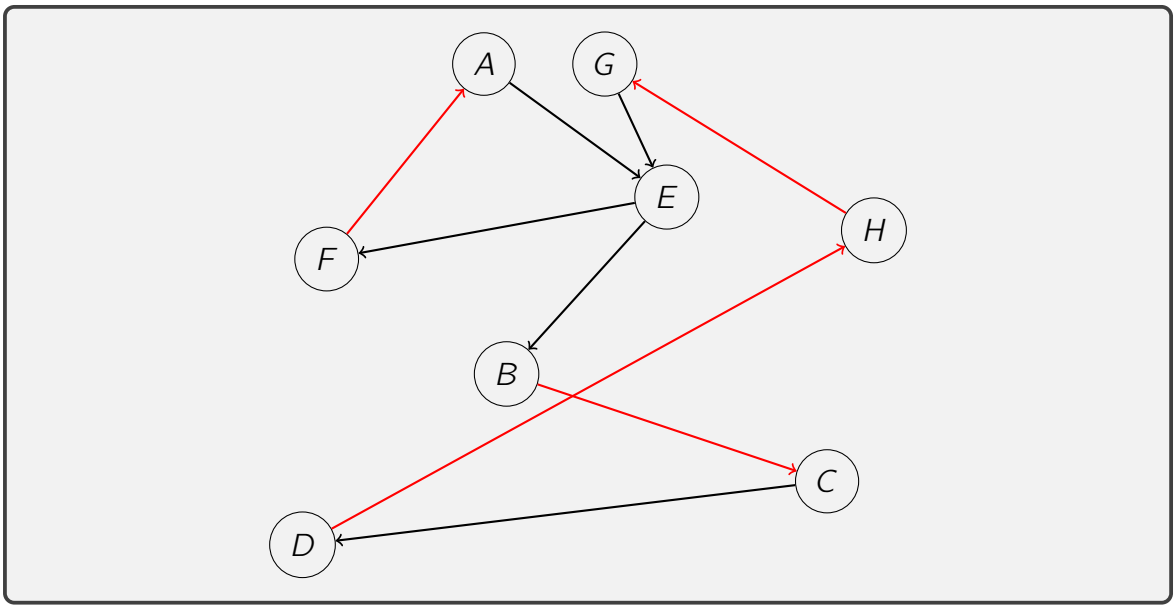
On the internet, many pages have links pointing to other pages, but sometimes it's not possible to reach a site you were on previously without clicking the "back" button in your browser. Elgoog can model their website as a directed graph G with n pages, and each page has some number of links to other pages. There are a total of m links over all these pages. Currently, it's not possible to get from some pages to some other pages without clicking the back button, and sometimes not possible at all! They want your help in designing an algorithm which can output the **minimum** total number of extra links they need to add so that every page is reachable from every other page.

1. In the given graph, find the minimum number of links that Elgoog must add.



Solution

The answer is 4. To see this, first suppose we just had the AGEFB subgraph. Here, we can connect F to A and B to G to get 2. Now, add in H: connect B to H and H to G instead of just having B to G. Finally, replace the B to H edge with B to C and D to H edges. So, the total edges added are F to A, B to C, D to H, H to G. Here's the new graph:



2. Suppose G is a directed, acyclic graph (DAG) where every sink is reachable from every source. Define $S \subseteq V$ as the set of *source* nodes: those vertices with no incoming edges and $T \subseteq V$ as the set of *sink* nodes: those vertices with no outgoing edges. **Prove** that the minimum number of links which have to be added is $\max(|S|, |T|)$.

Solution

Note that each source vertex must have at least one incoming edge added, and each sink vertex must have at least one outgoing edge added. Hence, we have a lower bound of $\max(|S|, |T|)$. To achieve this, assume without loss of generality that $|S| \geq |T|$ (there are more sources than sinks). Add one edge to each source from some sink, making sure that each sink gets at least one outgoing edge and each source gets exactly one incoming edge.
Since in the original graph there is a path from each source to each sink, adding all such edges implies that there is a path from any sink $u \in T$ to any other sink $v \in T$ and hence (by construction) to any source $w \in S$. Therefore, there is a path from every sink to every other node in the graph. Then, since from every node in the graph there is a path to a sink, this implies that there is a path from every node to every other node.

3 Currency conversion

Suppose the various economies of the world use a set of currencies $C_1, C_2, \ldots, C_n$ – think of these as dollars, pounds, bitcoins, etc. Your bank allows you to trade each currency C_i for any other currency C_j, and finds some way to charge you for this service (in a manner to be elaborated in the subparts below). We will devise algorithms to trade currencies to maximize the amount we end up with.

3.1 Flat fees

Suppose that for each ordered pair of currencies (C_i, C_j) the bank charges a flat fee of $f_{ij} > 0$ dollars to exchange C_i for C_j regardless of the quantity of currency being exchanged). Devise an efficient algorithm which, given a starting currency C_s, a target currency C_t, and a list of fees f_{ij} for all $i, j \in \{1, 2, \ldots, n\}$, computes the cheapest way (that is, incurring the least in fees) to exchange all of our currency in C_s to currency C_t. Justify the correctness of your algorithm and its runtime.

Solution

Build the complete graph on the currencies with weights equal to the corresponding fees; i.e. $G = (V, E, w)$ where $V = \{C_1, ..., C_n\}$, $E = \{(C_i, C_j) : i \neq j\}$, and $w(C_i, C_j) = f_{ij}$. Run Dijkstra's algorithm from C_s and return a shortest path $C_s \to C_t$.

Correctness: Notice that any path from C_s to C_t in G indicates a sequence of exchanges, and that its total weight is precisely the sum of the fees needed to perform those exchanges. Thus it suffices to find a $C_s \to C_t$ path in G of minimum weight. Note that the f_{ij}'s are all positive, so this is a valid input to Dijkstra's algorithm.

Running time: $O(n^2)$ total. Since all exchange pairs are possible, we have $m = \binom{n}{2} = \Theta(n^2)$ edges. This is also how long it takes to build G. Dijkstra's algorithm therefore takes $O((n^2 + n) \log n) = O(n^2 log n)$ time.

3.2 Exchange rates

Consider the more realistic setting where the bank does not charge flat fees, but instead uses exchange rates. In particular, for each ordered pair (C_i, C_j), the bank lets you trade one unit of C_i for r_{ij} units of C_j, i.e. you receive r_{ij} units of C_j in exchange of one unit of C_i. Devise an efficient algorithm which, given starting currency C_s, target currency C_t, and a list of rates r_{ij}, computes a sequence of exchanges that results in the greatest amount of C_t. Justify the correctness of your algorithm and its runtime.

Solution

Build $G = (V, E, w)$ as in part (a), but with weights $w(C_i, C_j) = -\log(r_{ij})$. Run Bellman-Ford from C_s and return a shortest path $C_s \to C_t$.

Correctness: As in part (a), a sequence of exchanges corresponds to a path in G. However, we want a path $C_{i_1} = C_s, C_{i_2}, ..., C_{i_k} = C_t$ here that maximizes $\prod_{l=1}^{k-1} r_{i_l, i_{l+1}}$. Since log is a monotonically increasing function (i.e. if $a \geq b$ then $\log(a) \geq \log(b)$), this is the same as maximizing $\log(\prod_{l=1}^{k-1} r_{i_l, i_{l+1}}) = \sum_{l=1}^{k-1} \log(r_{i_l, i_{l+1}})$. Finally, this is equivalent to minimizing $\sum_{l=1}^{k-1} -\log(r_{i_l, i_{l+1}}) = \sum_{l=1}^{k-1} w(C_{i_l}, C_{i_{l+1}})$, which is the shortest path objective. Note that we must use Bellman-Ford rather than Dijkstra's algorithm,

since these weights may be negative.

Running time: $O(n^3)$ total. G can be built in time $O(n^2)$ time, and Bellman-Ford takes $O(n^3)$ time since we have $\Theta(n^2)$ edges.

3.3 Making money

Due to fluctuations in the markets, it is occasionally possible to find a sequence of exchanges that lets you start with currency A, change into currencies, B, C, D, etc., and then end up changing back to currency A in such a way that you end up with more money than you started with—that is, there are currencies $C_{i_1}, \ldots, C_{i_k}$ such that

$$r_{i_1 i_2} \times r_{i_2 i_3} \times \ldots \times r_{i_{k-1} i_k} \times r_{i_k i_1} > 1.$$

Devise an efficient algorithm that finds such an anomaly if one exists. Justify the correctness of your algorithm and its runtime.

Solution

This problem is essentially trying to find a negative cycle in the graph. We can do this using the same graph in part (b) and run Bellman-Ford to check if in any iteration of the Bellman-Ford algorithm whether there is a negative cycle in G. If there is, the cycle is the anomaly, which means trading in the cycle will result in a profit!

Correctness: A currency anomaly $\prod_{l=1}^{k-1} r_{i_l, i_{l+1}} > 1$ implies (by the same log manipulations we did in part (b)) that $\sum_{l=1}^{k-1} w(C_{i_l}, C_{i_{l+1}}) = \sum_{l=1}^{k-1} -\log(r_{i_l, i_{l+1}}) < 0$. Thus there is a negative cycle in G, which can be found by an extra iteration of Bellman-Ford.

Running time: $O(n^3)$ total. We are still using the same algorithm in part (b) but having a different objective. Once we realize that there is a negative cycle, we must have one more iteration to find the exact cycle which takes $O(|E|) = O(n^2)$ time.