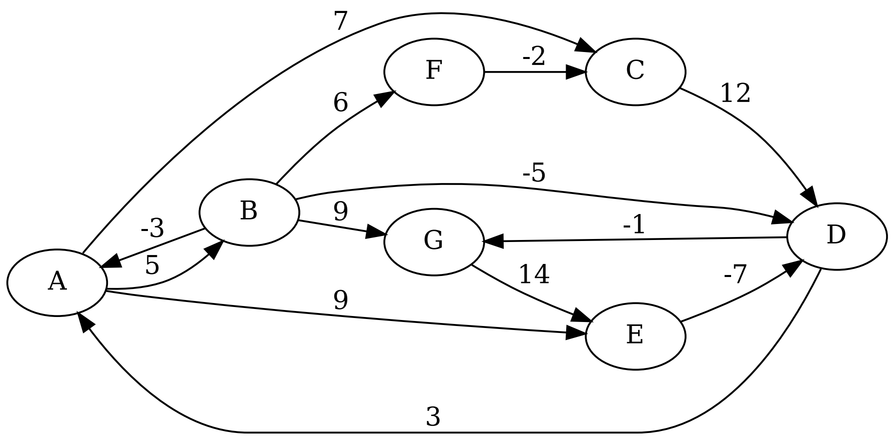
TD 6: APSP

1 Execute Johnson's algorithm

Execute Johnson's algorithm on the graph below.



2 Shortest Cycles

We are given as input a directed graph $G = (V, A)$ with positive weights on the arcs in adjacency list form (that is, you can assume that for each vertex you have a list of outgoing arcs and their corresponding weights).

- Give an efficient algorithm that takes as input an arc $ij \in A$ and outputs the shortest cycle that contains ij (or correctly reports that none exists).
- Give an efficient algorithm that outputs the shortest cycle in G.

How does your answer change if the graph may have negative weights? (you are promised that no negative cycles exist)

3 Many distances

We are given as input a directed graph $G = (V, A)$ with positive arc weights in adjacency list form. Furthermore, we are given a set of k special vertices $S \subseteq V$. We define the S-distance of a vertex $x \in V$ as $\mathrm{dist}_S(x) = \min_{s \in S} \mathrm{dist}(x, s)$. Give an efficient algorithm that computes the S-distances of all vertices.

To motivate the problem, consider the following scenario: G represents the road network between different cities, arc weights represent distances, and S represents a set of cities that contain an important facility (e.g. a large hospital). For other cities, we want to calculate what is the minimum distance we need to traverse to reach *some* important facility.

4 (Min,+)-Product

4.1 The basics

Suppose we are given two matrices A, B with dimensions $n_1 \times n_2$ and $n_2 \times n_3$ respectively. Then, the $(\min, +)$-product of A, B, denoted $A \otimes B$ is defined as the $n_1 \times n_3$ matrix C satisfying the following:

$$C[i,j] = \min_{k \in \{1,\ldots,n_2\}} \{A[i,k] + B[k,j]\}, \ \ \forall i \in \{1,\ldots,n_1\}, j \in \{1,\ldots,n_3\}$$

Recall the following facts, discussed in class:

- Given two $n \times n$ matrices A, B, $A \otimes B$ can be computed in $O(n^3)$ time (assuming arithmetic operations take time $O(1)$).
- Assume we have an algorithm that can calculate $A \otimes A$, where A is an $n \times n$ matrix, in time $O(T)$. Then, we can solve APSP on weighted directed graphs without negative cycles in time $O(T \log n)$.

4.2 (min,+) and APSP equivalence

As discussed in class, the question of whether APSP can be solved in time $O(n^{3-\varepsilon})$ is one of the most important open problems in theoretical computer science. The above indicate that one approach to obtain such an algorithm would be to design a faster algorithm for $(\min, +)$-product. This approach, however, presents some serious challenges (discussed again below). One could, then, hope, that perhaps we can obtain a faster APSP algorithm through some other means.

- Prove that it is in fact impossible to obtain a sub-cubic APSP algorithm without improving upon the best algorithm for $(\min, +)$-product. More precisely, show that if there is an algorithm solving APSP in time $O(n^{3-\varepsilon})$, then such an algorithm also exists for computing the $(\min, +)$-product of two $n \times n$ matrices. (Hint: given two matrices, construct a graph such that the result is hidden in the APSP matrix of the graph).

4.3 (min,+) squares

So far we have tried (and failed) to improve upon the fastest APSP algorithm by using $(\min, +)$-matrix multiplication. One could, however, object that there is an angle we have neglected: in order to speed up the best APSP algorithm it is not necessary to speed up $(\min, +)$-multiplication in general; rather, it is sufficient to have a faster algorithm for computing **squares**. In other words, we can focus on the special case of the problem where $A = B$.

- Show that this idea does not help. That is, there is an algorithm which can compute $A \times B$, for two $n \times n$ matrices, in time $O(T)$, if and only if there is such an algorithm for the case $A = B$.

4.4 (min,+) to normal products

Recall that, as we discussed in class, the usual matrix multiplication operations (where $\min$ is replaced by $+$ and $+$ is replaced by $\times$) does have sub-cubic algorithms (notably, Strassen's algorithm). Unfortunately, it does not seem possible to use such algorithms to compute the $(\min, +)$-product faster than time $O(n^3)$, because such algorithms rely on the ability to perform subtractions. In the $(\min, +)$ case we would therefore need a function that is the inverse of $\min$. Let us explore an idea that may allow us to do this.

We are given two $n \times n$ matrices A, B, and let W be the largest absolute value of any entry in A, B. Consider the following algorithm to compute $A \otimes B$:

1. Compute that matrix A' with $A'[i,j] = (n+1)^{W - A[i,j]}$. Similarly, compute B'.
2. Compute $C' = A' \cdot B'$ (using the normal matrix product).

3. For each $i, j \in \{1, \ldots, n\}$ compute $C[i, j]$ as follows: let x be the maximum integer such that $(n+1)^x \leq C'[i, j]$; then set $C[i, j] := 2W - x$.

- Prove that the algorithm above correctly calculates $A \otimes B$.
- Explain why the algorithm above still does not lead to a faster APSP algorithm (even though step 2 can be performed using Strassen's algorithm).