

Session 8

Corbet Elkins

October 23, 2026

Purdue University

Questions over lecture content?

See title

Graph Review

Recall the following algorithms from previous courses:

1. Depth-First Search (DFS) $O(n + m)$: used to determine if a node s can reach another node t in a graph.
2. Breadth-First Search (BFS) $O(n + m)$: used to determine the shortest path from a node s to all other nodes in an unweighted graph.
3. Dijkstra's Algorithm $O((n + m) \log n)$: used to determine the shortest path from a node s to all other nodes in a weighted graph with non-negative edge weights.
4. SCC (Strongly Connected Components) $O(n + m)$: used to determine the strongly connected components of a directed graph.

Graph Reductions

So these algorithms are pretty useful in their own, but what if I want to solve a slightly more complex algorithm.

Graph Reductions

So these algorithms are pretty useful in their own, but what if I want to solve a slightly more complex algorithm.

For example, what if I want to know if there exists an s - t path of even length in a graph?

Graph Reductions

So these algorithms are pretty useful in their own, but what if I want to solve a slightly more complex algorithm.

For example, what if I want to know if there exists an s - t path of even length in a graph?

I know what you might be thinking, what if I just alter DFS to keep track of the parity of the path length?

Graph Reductions

So these algorithms are pretty useful in their own, but what if I want to solve a slightly more complex algorithm.

For example, what if I want to know if there exists an s - t path of even length in a graph?

I know what you might be thinking, what if I just alter DFS to keep track of the parity of the path length? **NO! BAD! I WILL BE VERY SAD!**

Graph Reductions

So these algorithms are pretty useful in their own, but what if I want to solve a slightly more complex algorithm.

For example, what if I want to know if there exists an s - t path of even length in a graph?

I know what you might be thinking, what if I just alter DFS to keep track of the parity of the path length? **NO! BAD! I WILL BE VERY SAD!**

Instead, we will modify the graph and then feed that into the original DFS algorithm.

Graph Reduction Example

Consider a graph $G = (V, E)$. We want to determine if there exists an s - t path of even length.

Graph Reduction Example

Consider a graph $G = (V, E)$. We want to determine if there exists an s - t path of even length.

We construct a new graph $G' = (V', E')$ as follows:

- For each vertex $v \in V$, create two vertices v_0 and v_1 in V' .
- For each edge $(u, v) \in E$, add edges (u_0, v_1) and (u_1, v_0) to E' .

Graph Reduction Example

Consider a graph $G = (V, E)$. We want to determine if there exists an s - t path of even length.

We construct a new graph $G' = (V', E')$ as follows:

- For each vertex $v \in V$, create two vertices v_0 and v_1 in V' .
- For each edge $(u, v) \in E$, add edges (u_0, v_1) and (u_1, v_0) to E' .

Now, there exists an s - t path of even length in G if and only if there exists an s_0 - t_0 path in G' . (Why?)

Okay but why do I care?

Well, in using our original algorithm, we don't need to prove correctness of the algorithm, we only need to prove that our graph reduction is correct.

Okay but why do I care?

Well, in using our original algorithm, we don't need to prove correctness of the algorithm, we only need to prove that our graph reduction is correct.

It might seem easy to prove the correctness for this example, but for later examples it can become much more challenging.

Okay but why do I care?

Well, in using our original algorithm, we don't need to prove correctness of the algorithm, we only need to prove that our graph reduction is correct.

It might seem easy to prove the correctness for this example, but for later examples it can become much more challenging.

Doing it this way also makes it much cleaner to argue for slight changes in the algorithm, for example if I instead wanted to find an s - t path with a length that is a multiple of 3

Graph Reduction Tools

The primary method used in graph reductions is to encode the states into the new graph.

Graph Reduction Tools

The primary method used in graph reductions is to encode the states into the new graph.

For example, in the previous example, we encoded the parity of the path length into the new graph by creating two vertices for each original vertex, one representing an even distance and the other representing an odd distance.

Graph Reduction Tools

The primary method used in graph reductions is to encode the states into the new graph.

For example, in the previous example, we encoded the parity of the path length into the new graph by creating two vertices for each original vertex, one representing an even distance and the other representing an odd distance.

Another useful method is are super sources/super sinks, which are additional vertices added to the graph to simplify connectivity constraints. These are useful if there are multiple starting or accepting states in the graph.

Graph Reduction Problem

Problem

Given a positive weighted graph $G = (V, E)$, and two vertices $s, t \in V$, determine the length of the shortest s - t path whose number of edges is either a multiple of 3 or a multiple of 5, but not a multiple of 15

Graph Reduction Problem

Problem

Given a positive weighted graph $G = (V, E)$, and two vertices $s, t \in V$, determine the length of the shortest s - t path whose number of edges is either a multiple of 3 or a multiple of 5, but not a multiple of 15

So for the graph construction, we somehow want to encode if a path length meets these criterion.

Graph Reduction Problem

Problem

Given a positive weighted graph $G = (V, E)$, and two vertices $s, t \in V$, determine the length of the shortest s - t path whose number of edges is either a multiple of 3 or a multiple of 5, but not a multiple of 15

So for the graph construction, we somehow want to encode if a path length meets these criterion.

Thoughts?

Graph Reduction Problem

So we need to create a new graph where each vertex encodes the remainder of the path length modulo 15. This way, we can track whether the path length is a multiple of 3, a multiple of 5, or a multiple of 15.

Graph Reduction Problem

So we need to create a new graph where each vertex encodes the remainder of the path length modulo 15. This way, we can track whether the path length is a multiple of 3, a multiple of 5, or a multiple of 15.

Specifically, for each vertex $v \in V$ and each remainder $r \in \{0, 1, \dots, 14\}$, we create a vertex v_r in the new graph. For each edge $(u, v) \in E$ and each remainder r , we add an edge from u_r to $v_{(r+1) \bmod 15}$. Let G' be the new graph.

Graph Reduction Problem

So we need to create a new graph where each vertex encodes the remainder of the path length modulo 15. This way, we can track whether the path length is a multiple of 3, a multiple of 5, or a multiple of 15.

Specifically, for each vertex $v \in V$ and each remainder $r \in \{0, 1, \dots, 14\}$, we create a vertex v_r in the new graph. For each edge $(u, v) \in E$ and each remainder r , we add an edge from u_r to $v_{(r+1) \bmod 15}$. Let G' be the new graph.

Here, we can use a supersink, T , to connect to all of these vertices t_r where r is a multiple of 3 or 5 but not 15, and with edge weight 0. Then we just ask for the shortest path from s_0 to T in the new graph.

Proof of Correctness

Let ℓ be the algorithm's output and ℓ^* be the solution.

Proof of Correctness

Let ℓ be the algorithm's output and ℓ^* be the solution.

We first show $\ell \leq \ell^*$. To see this, we take the path from G' and turn it into a path for G , as follows

$s_0 \rightarrow v_1^1 \rightarrow v_2^2 \rightarrow \cdots \rightarrow v_{k \bmod 15}^k \rightarrow T \rightarrow t$. Each edge in G' corresponds to an edge in G , and the total length of the path in G is the same as the length of the path in G' minus 1 for the super source. Therefore, $\ell \leq \ell^*$.

Proof of Correctness

Let ℓ be the algorithm's output and ℓ^* be the solution.

We first show $\ell \leq \ell^*$. To see this, we take the path from G' and turn it into a path for G , as follows

$s_0 \rightarrow v_1^1 \rightarrow v_2^2 \rightarrow \cdots \rightarrow v_{k \bmod 15}^k \rightarrow T \rightarrow t$. Each edge in G' corresponds to an edge in G , and the total length of the path in G is the same as the length of the path in G' minus 1 for the super source. Therefore, $\ell \leq \ell^*$.

We show $\ell \geq \ell^*$. By similarly taking a path in G and turning it into a path in G' .

Proof of Correctness

Let ℓ be the algorithm's output and ℓ^* be the solution.

We first show $\ell \leq \ell^*$. To see this, we take the path from G' and turn it into a path for G , as follows

$s_0 \rightarrow v_1^1 \rightarrow v_2^2 \rightarrow \cdots \rightarrow v_{k \bmod 15}^k \rightarrow T \rightarrow t$. Each edge in G' corresponds to an edge in G , and the total length of the path in G is the same as the length of the path in G' minus 1 for the super source. Therefore, $\ell \leq \ell^*$.

We show $\ell \geq \ell^*$. By similarly taking a path in G and turning it into a path in G' .

See isn't that so much easier than reproving Dijkstra's.