Lab week 6

C++ Essentials

2024-03-19

Phillip G. Bradford

Lab Exercise

Write a C++ program that computes reachability for a graph starting with -1, 0, and +1 labeled edges. This is the same problem as in the last lab.

However, we will use only **one** matrix. This is done by using the following technique.

Given the three $n \times n$ matrices $D^{(0)}$, $D^{(-1)}$ and $D^{(1)}$ from the last lab, then compute the matrix C with elements

$$c_{i,j}^{g} \leftarrow (3(n+1))^{d_{i,j}^{g}}$$

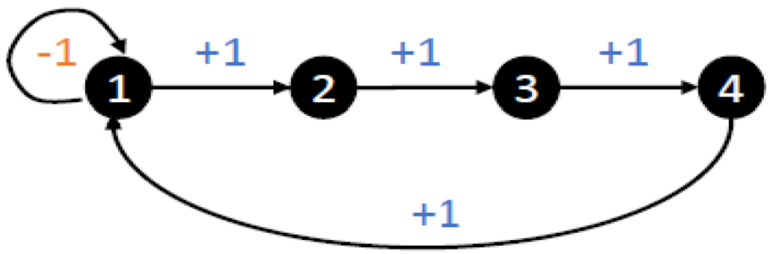
where $g \in \{-1,0,+1\}$.

This means
A 0 edge is represented as $\big(3(n+1)\big)^0 = 1$
A 1 edge is represented as $\big(3(n+1)\big)^1 = 3(n+1)$
A -1 edge is represented as $\big(3(n+1)\big)^{-1} = \frac{1}{3(n+1)}$

That is, a -1 edge followed by a +1 edge is the product $\big(3(n+1)\big)^{-1}\big(3(n+1)\big)^1 = 1$ since the exponents add together. Try the other examples. Use 0 for infinity which means there is no edge.

This means we can use matrix multiplication to find exact zero cost paths.

For example



The **old representation** with matrix $D^{(0)}$ is

$D^{(0)}$	$D^{(-1)}$	$D^{(1)}$
0 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0	-1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	2 1 2 2 2 2 1 2 2 2 2 1 1 2 2 2

Our new representation gives a **new matrix** A:

0.066667	15	0	0
0	0	15	0
0	0	0	15
15	0	0	0

One matrix product $A \times A$ gives

0.004444	1	225	0
0	0	0	225
225	0	0	0
1	225	0	0

Note the two 1 values indicate we found the exact 0 path from node 1 to note 2 in cell A[1][2]. Similarly, we found an exact 0 path from node 4 to node 1 in A[4][1].

Squaring this last matrix $(A \times A)^2 = A \times A \times A \times A$ gives

50625	0.004444	1	225
225	50625	0	0
1	225	50625	0
0.004444	1	225	50625

But some of the other numbers are a little challenging.

So, to start we must implement the following:

1. A matrix multiplication algorithm
2. Initialization algorithm to start
3. Normalization algorithm to remove some of the “strange numbers"

Initialization algorithm

For all $i, j : n \geq i, j \geq 1$, let

$$
\begin{aligned}
c_{i,j}^{-1} &\leftarrow \sum_{k=1}^{n} \left((3(n+1))^{d_{i,k}^{0} + d_{k,j}^{-1}} + (3(n+1))^{d_{i,k}^{-1} + d_{k,j}^{0}} \right) \\
c_{i,j}^{0} &\leftarrow \sum_{k=1}^{n} \left((3(n+1))^{d_{i,k}^{-1} + d_{k,j}^{+1}} + (3(n+1))^{d_{i,k}^{+1} + d_{k,j}^{-1}} \right) \\
c_{i,j}^{+1} &\leftarrow \sum_{k=1}^{n} \left((3(n+1))^{d_{i,k}^{0} + d_{k,j}^{+1}} + (3(n+1))^{d_{i,k}^{+1} + d_{k,j}^{0}} \right)
\end{aligned}
$$

Normalization algorithm is based on the functions

1. $\mathit{detectNegativeOneEdge}(\text{edge\_cost}, n)$
2. check $\leftarrow 3(n+1) \times$ fractional_part(edge_cost)
3. **if** $2n \geq$ check ≥ 1 **then return** True
4. **else return** False

1. $\mathit{detectPositiveOneEdge}(\text{edge\_cost}, n)$
2. check $\leftarrow$ truncate(edge_cost$/3(n+1)$)
3. **if** $2n \geq$ check ≥ 1 **then return** True
4. **else return** False

1. $\mathit{detectZeroEdge}(\text{edge\_cost}, n)$
2. check $\leftarrow$ truncate(edge_cost) **mod** $3(n+1)$
3. **if** $3n \geq$ check > 0 **then return** True
4. **else return** False

What to hand in

A single page document containing:

1. **Your name and the Lab name: matrix based reachability**
2. **A github repo link**
3. **The github repo must be cloneable by the graders**
4. **The cloned code must run its unit tests after a compile**
5. **Any compiling instructions must be in your README file**

You must include declaration files (H-files) and definition (CPP-files) as well as a **unit testing**.