Graphs, Continued

CSC212 — Fall 2014

Dominique Thiébaut
dthiebaut@smith.edu
Detecting Cycles
Where are the cycles?
Detecting Cycles in Undirected Graphs
Detecting Cycles in Undirected Graphs
for every vertex $v$ in vertices:

\[ \text{num}(v) = 0; \]

\[ \text{cycleDetection}(v) \]

\[ \text{num}(v) = i++; \]

for all vertices $u$ adjacent to $v$

if \[ \text{num}(u) == 0 \]

record edge $(v,u)$ in edges;

\[ \text{cycleDetection}(u); \]

else if edge $(v,u)$ not in edges

\[ \text{cycle detected}; \]
Detecting Cycles in Directed Graphs
for every vertex $v$ in $vertices$:

\[
\text{num}(v) = 0;
\]

\[
digraphCycleDetection(v)
\]

\[
\text{num}(v) = i++; \\
\text{for all vertices } u \text{ adjacent to } v \\
\text{if } \text{num}(u) == 0 \\
\quad \text{digraphCycleDetection}(u); \\
\text{else if num}(u) != \infty \\
\quad \text{cycle detected}; \\
\text{num}(v) = \infty
\]
Network Algorithm
Union-Find

Good reference:
https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf
Are $u$ and $v$ connected?
Union-Find Idea:

- **Union**: links together vertices that are connected by a path.

- **Find**: Quickly figures out if two vertices are connected by a path.

- Union and Find operations are typically intertwined.

- Union and Find should be **fast** operations, as number of operations and number of vertices can be **huge**.
Python-Style Organization

• Number vertices 0 to V-1

• Create a list of all vertices: [0, 1, 2, 3, … 9, 10]

• Create list of vertices connected to each other:
  [ [0, 1, 10], 2, [3, 5], 4, [6, 7, 8, 9] ]

• **Find operation** should be able to easily answer: are 3 and 6 in same group

• **Union operation** should be able to easily join groups. E.g. connect 3 and 4:
  [ [0, 1, 10], 2, [3, 4, 5], [6, 7, 8, 9] ]
Remarkable Simple Implementation: Int Array!
Remarkable Simple Implementation: Int Array!
Remarkable Simple Implementation: Int Array!
Remarkable Simple Implementation: Int Array!

How do we know that the Id of 2 should change, and not that of 1?
Remarkable Simple Implementation: Int Array!

How do we know that the Ide of 3 should change, and not that of 2?
Remarkable Simple Implementation: Int Array!

How do we know that the Id of 4 should change, and not that of 3?
Remarkable Simple Implementation: Int Array!
Remarkable Simple Implementation: Int Array!
Questions to Address

• Which Id should we choose when doing a union operation on two vertices?

• Doing a Union operation on two large collections of vertices could be time consuming
Improvement 1: Nested Id Definition
Using Nested Ids

Id

0 1 2 3 4 5 6

0 1 2 3 4 5 6
Using Nested Ids
Using Nested Ids
Using Nested Ids
Using Nested Ids

![Diagram showing nodes and edges with nested ids]

The diagram illustrates nodes labeled 0, 1, 2, 3, 4, 5, and 6, connected by edges. The id matrix shows the nested id assignments.
Using Nested Ids
• Id of Vertex 5 = ?
  \text{Id}(5) = 6
  \text{Id}(6) = 6 \text{ (same)} \implies \text{Id of Vertex 5 is 6}

• Id of Vertex 0 = ?
  \text{Id}(0) = 0 \text{ (same)} \implies \text{Id of Vertex 0 is 0}

• Id of Vertex 4 = ?
  \text{Id}(4) = 3
  \text{Id}(3) = 2
  \text{Id}(2) = 1
  \text{Id}(1) = 0
  \text{Id}(0) = 0 \text{ (same)} \implies \text{Id of Vertex 4 is 0}
public static int idOf(int p) {
    while (id[p] != p)
        p = id[p];
    return p;
}
public static boolean quickFind( int p, int q ) {
    return idOf( p ) == idOf( q );
}
public static void quickUnite( int p, int q ) {
    int id_p = idOf( p );
    int id_q = idOf( q );
    if ( id_p == id_q )
        return;
    id[ id_p ] = id_q;
}
(UnionFindFast.java)
Some of these Operations could be Inefficient...
Perform Union 4-13

id of...

before  after
Solution: Keep Size of Trees!
public static void quickUnite( int p, int q ) {
    int id_p = idOf( p );
    int id_q = idOf( q );

    if ( id_p == id_q )
        return;

    if ( size[ id_p ] <= size[ id_q ] ) {
        id[id_p] = id_q;
        size[id_p] += size[id_q];
    }
    else {
        id[id_q] = id_p;
        size[id_q] += size[id_p];
    }
}
Improvement to Union-Find: Path-Compression
idOf(p)
idOf(p)

\[ \text{Id}(4) \]
idOf(p)

Id(4)?
(UnionFindFaster.java)
### Practical Complexity

- $M \text{ find}$ and/or $\text{union}$ operations on $V$ vertices

<table>
<thead>
<tr>
<th></th>
<th>Average Case Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init Array of Ids</td>
<td>$O(V)$</td>
</tr>
<tr>
<td>Find</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Union</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
LAB TIME!
Some Explanations…
Applet = setup() + draw()

Frequency = 30 draw/sec

Start Applet
Network: 50x50 grid of vertices
Location

Row 0

Row 1

Row 2

Row 3

Col 0

Col 1

Col 2

Col 3

Col: $\frac{103}{50} = 2$

Row: $103 \% 50 = 3$
\[ x = \text{deltaX} + \text{col} \times \text{deltaX} \]
\[ y = \text{deltaY} + \text{row} \times \text{deltaY} \]
Mouse Position

\( x = \text{deltaX} + \text{col} \times \text{deltaX} \)

\( y = \text{deltaY} + \text{row} \times \text{deltaY} \)
Finding Vertex Closest to Mouse Pointer

for ( int v=0; v<maxNoVertices; v++ ) {

    int vCol = deltaX + (int)( v / maxNoVerticesCols ) * deltaX;
    int vRow = deltaY + ( v % maxNoVerticesRows ) * deltaY;

    // is mouse pointer inside this vertex? If so, display some
    // information about this vertex
    if ( dist( vCol, vRow, mouseX, mouseY ) < 4 ) {
        fill( 255, 0, 0 );
        text( v" ", vCol, vRow );
        text( adjList( v ), 100, HEIGHT-10 );
    }
}

Generating
The same series of **random** numbers...

```java
import java.util.Random;

public class GenerateRandomInts {
    public static void main(String[] args) {
        Random random = new Random();
        random.setSeed(12345);
        for (int i = 0; i < 10; i++) {
            System.out.println(random.nextInt(100));
        }
    }
}
```

51 80 41 28 55 84 75 2 1 89