Inf-2101 - Algoritmer

Graph Search

John Markus Bjørndalen

2010-09-02

Some foils are adapted from the book and the book's homepage.

- graph API
- maze exploration
- depth-first search
- **breadth-first search**
- connected components
- challenge

43

Breadth-first search

Depth-first search . Put unvisited vertices on a stack.

Breadth-first search . Put unvisited vertices on a queue.

Shortest path . Find a path from *s* to *t* that uses fewest number of edges.

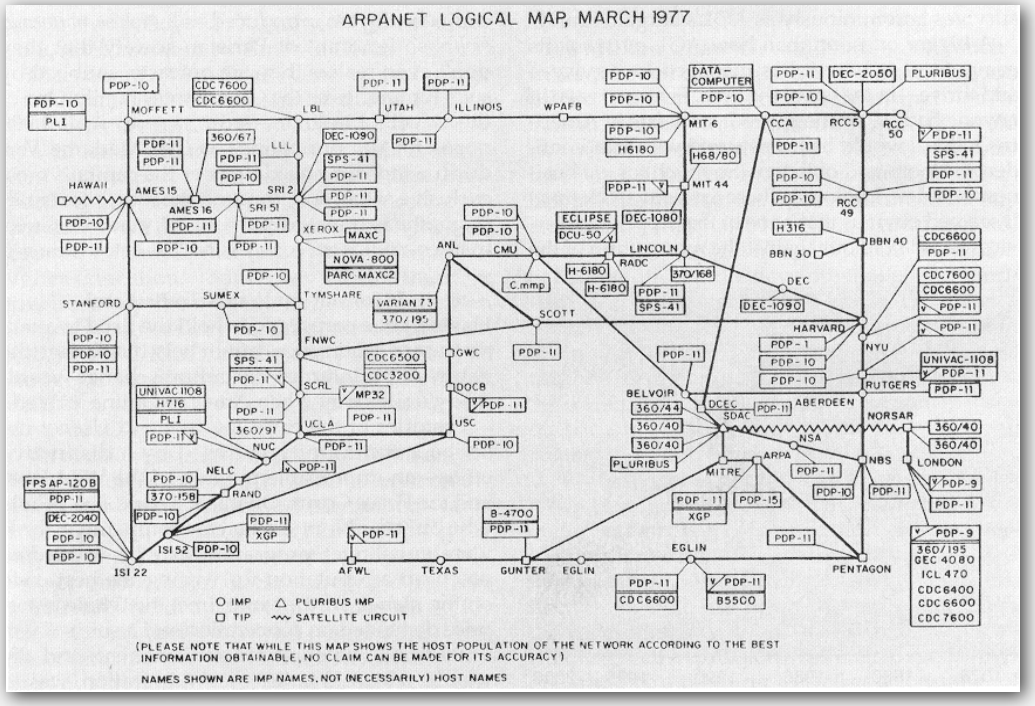
```
BFS(from source s to target t)
------------------------------
Put s onto a FIFO queue.
Repeat until the queue is empty;
* remove the least recently added vertex to v
* add each of v's unvisited neighbours to the queue,
  and mark them as visited.
```

Property . BFS examines vertices in increasing distance from *s*.

Time for some code again

BFS application

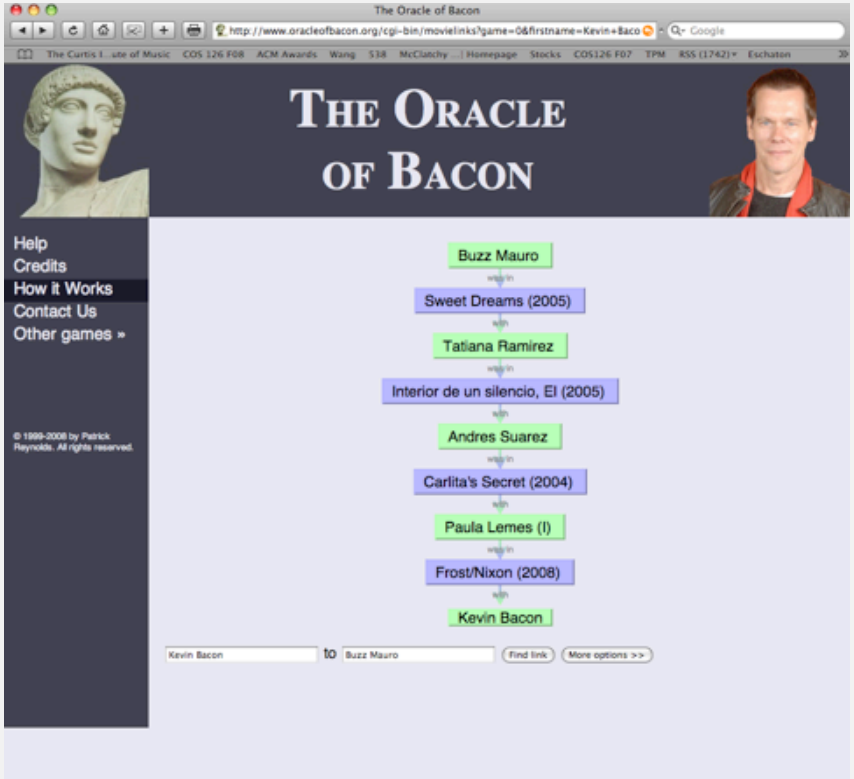
- Facebook.
- Kevin Bacon numbers.
- Fewest number of hops in a communication network.



ARPANET

BFS application

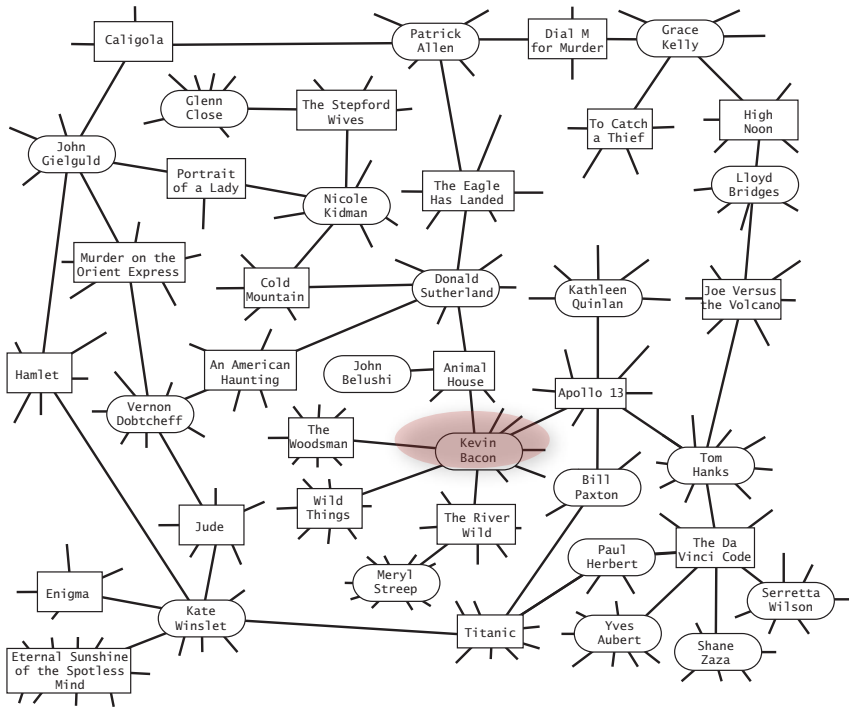
- Facebook.
- Kevin Bacon numbers.
- Fewest number of hops in a communication network.



48

Kevin Bacon graph

- Include vertex for each performer and movie.
- Connect movie to all performers that appear in movie.
- Compute shortest path from s = Kevin Bacon.



A tiny portion of the movie-performer relationship graph

Iterative Deepening Depth First Search

Breadth-first search returns a shortest path, but may require a lot of edges to be put on the queue for dense, large graphs.

Depth-first search is efficient memory-wise, but returns the first path found, not the shortest.

Iterative Deepening Depth First Search

Idea: use DFS, but limit the depth of the search.

A depth-limited search will stop adding edges to the fringe when it has reached a depth limit.

By starting with a low depth limit and increasing by 1 until we find a solution, we effectively end up with a shortest path.

Memory requirements are similar to DFS.

Iterative Deepening Depth First Search

Source code

- graph API
- maze exploration
- depth-first search
- breadth-first search
- **connected components**
- challenge

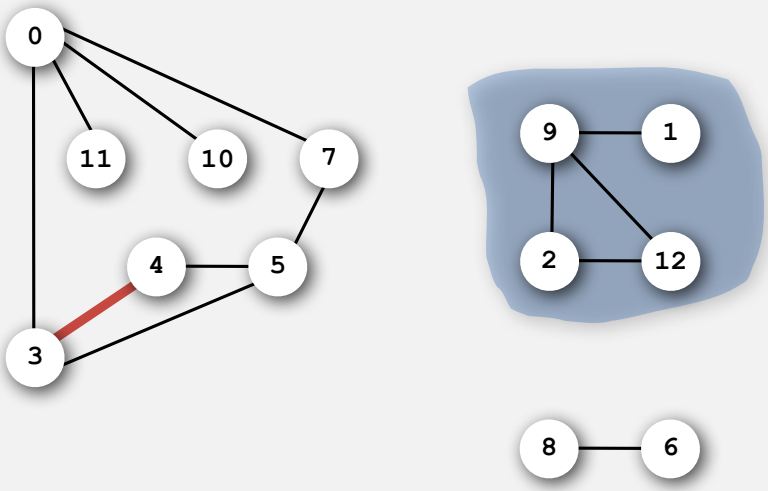
50

Connectivity queries

Def. Vertices v and w are connected if there is a path between them.

Def. A connected component is a maximal set of connected vertices.

Goal. Preprocess graph to answer queries: is `v` connected to `w`?
in constant time



Vertex	Component
0	0
1	1
2	1
3	0
4	0
5	0
6	2
7	0
8	2
9	1
10	0
11	0
12	1

Union-Find? Not quite.

51

Connected components

Goal: partition vertices into connected components.
Method:

- Initialize all vertices v as unmarked.
- For each unmarked vertex, v, run DFS to identify all vertices discovered as part of the same component.

preprocess time	query time	extra space
$E + V$	1	V

Depth-first search for connected components

```
public class CCFinder
{
   private final static int UNMARKED = -1;
   private int components;
   private int[] cc;

   public CCFinder(Graph G)
   {  /* see next slide */  }

   public int connected(int v, int w)
   {  return cc[v] == cc[w];  }

}
```

component labels

constant-time connectivity query

53

Depth-first search for connected components

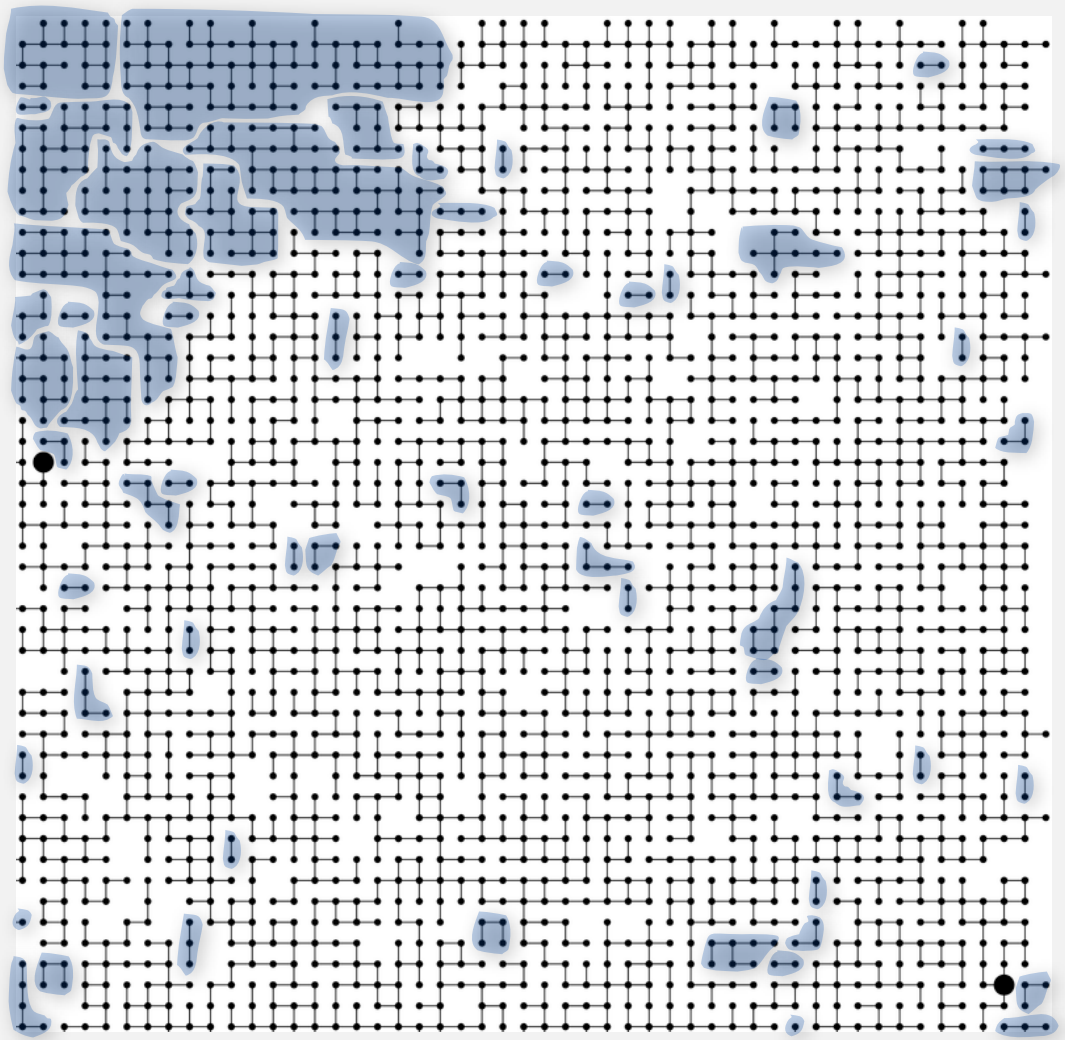
```
public CCFinder(Graph G)
{
   cc = new int[G.V()];
   for (int v = 0; v < G.V(); v++)
      cc[v] = UNMARKED;
   for (int v = 0; v < G.V(); v++)
      if (cc[v] == UNMARKED)
      {
         dfs(G, v);
         components++;
      }
}
```

DFS for each component

```
private void dfs(Graph G, int v)
{
   cc[v] = components;
   for (int w : G.adj(v))
      if (cc[w] == UNMARKED) dfs(G, w);
}
```

standard DFS

Connected components

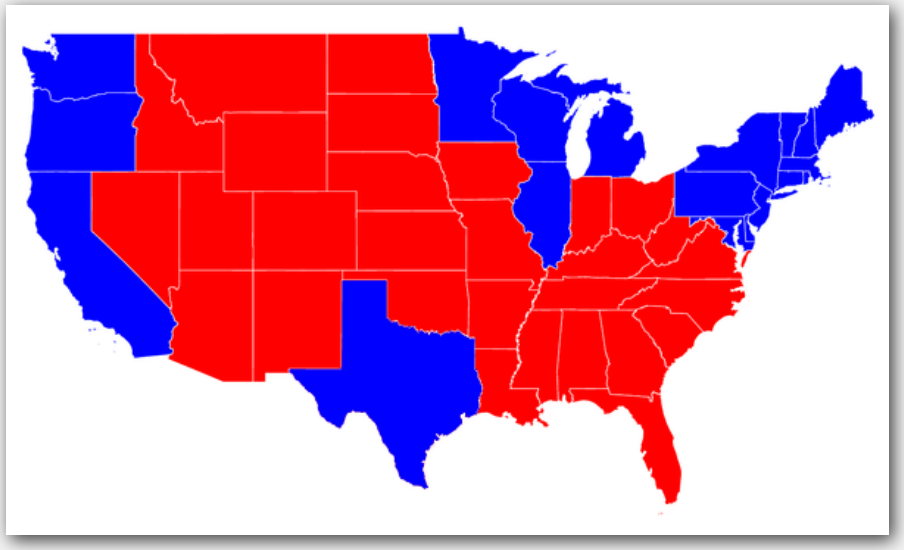


63 components

55

Connected components application: image processing

Goal. Read in a 2D color image and find regions of connected pixels that have the same color.



assuming contiguous states

Input. Scanned image.

Output. Number of red and blue states.

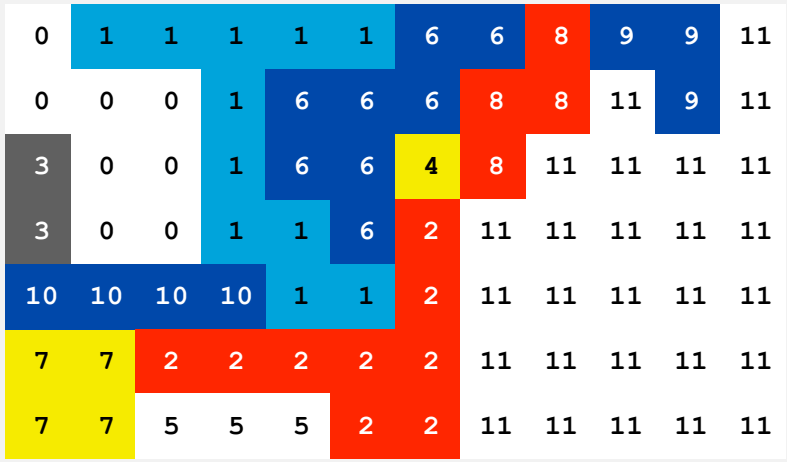
56

Connected components application: image processing

Goal. Read in a 2D color image and find regions of connected pixels that have the same color.

Efficient algorithm.

- Create grid graph.
- Connect each pixel to neighboring pixel if same color.
- Find connected components in resulting graph.

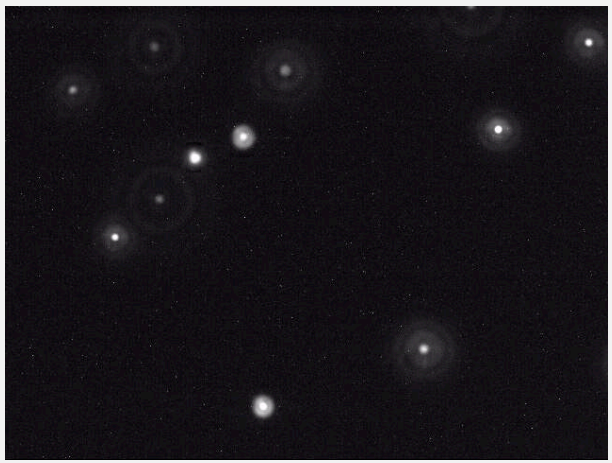


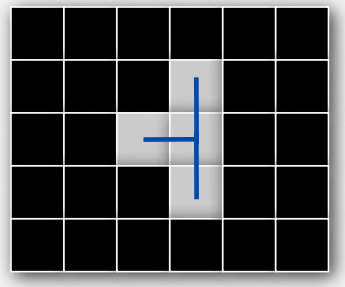
Connected components application: particle detection

Particle detection. Given grayscale image of particles, identify "blobs."

- Vertex: pixel.
- Edge: between two adjacent pixels with grayscale value ≥ 70.
- Blob: connected component of 20-30 pixels.

black = 0
white = 255





Particle tracking. Track moving particles over time.

58

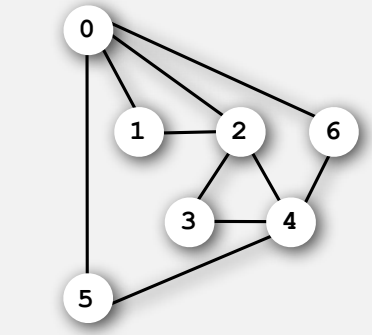
- graph API
- maze exploration
- depth-first search
- breadth-first search
- connected components
- **challenges**

59

Graph-processing challenge 3

Problem. Find a cycle that uses every edge.

Assumption. Need to use each edge exactly once.



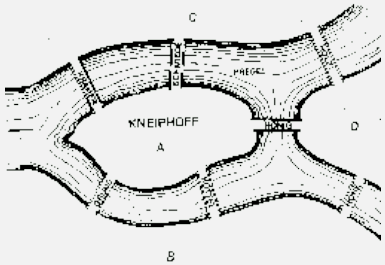
How difficult?

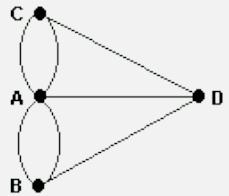
- Any COS 126 student could do it.
- Need to be a typical diligent COS 226 student.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

Bridges of Königsberg

The Seven Bridges of Königsberg. [Leonhard Euler 1736]

> *" … in Königsberg in Prussia, there is an island A, called the Kneiphof; the river which surrounds it is divided into two branches ... and these branches are crossed by seven bridges. Concerning these bridges, it was asked whether anyone could arrange a route in such a way that he could cross each bridge once and only once."*





Euler tour. Is there a cyclic path that uses each edge exactly once?

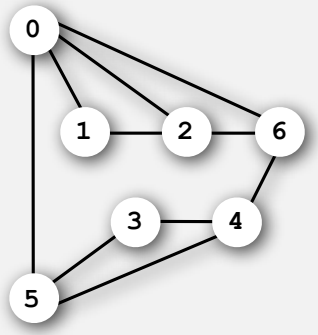
Answer. Yes iff connected and all vertices have even degree.

To find path. DFS-based algorithm (see Algs in Java).

Graph-processing challenge 4

Problem. Find a cycle that visits every vertex.

Assumption. Need to visit each vertex exactly once.



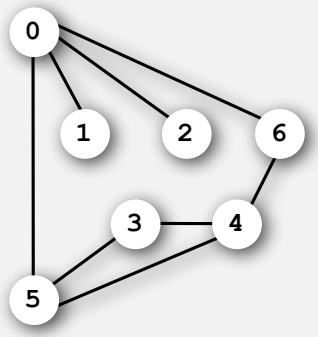
How difficult?

- Any COS 126 student could do it.
- Need to be a typical diligent COS 226 student.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

62

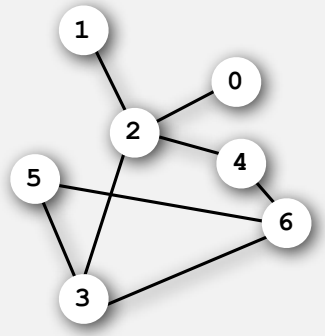
Graph-processing challenge 5

Problem. Are two graphs identical except for vertex names?



How difficult?

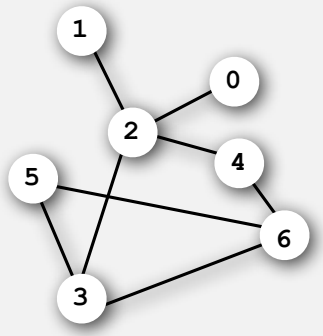
- Any COS 126 student could do it.
- Need to be a typical diligent COS 226 student.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.



63

Graph-processing challenge 6

Problem. Lay out a graph in the plane without crossing edges?



0-2
1-2
2-3
2-4
3-5
3-6
4-6
5-6

How difficult?

- Any COS 126 student could do it.
- Need to be a typical diligent COS 226 student.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

64