CSE 250: DFS+BFS; Dijkstra's Algorithm

Lecture 21

Oct 20, 2023

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Reminders

- PA2 released yesterday: Implement **Map Routing**
  1. Create an adjacency list (discussed today)
  2. Find a path from A to B with the fewest intersections
  3. Find a path from A to B with the shortest distance
- PA2 test cases due Sun, Oct 22 at 11:59 PM
- PA2 implementation due Sun, Nov 5 at 11:59 PM

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search

Start with some node A.

1. If we've explored all edges from A, return
2. Pick an unexplored edge
3. If the edge connects to an explored vertex, it's a back edge
4. Otherwise recur with A = the other vertex.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search

Mark Vertices `UNVISITED`	$O(N)$
Mark Edges `UNVISITED`	$O(M)$
`DFS` Vertex Loop	$O(N)$
All calls to `DFSOne`	$O(M)$
	$O(N+M)$

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Breadth First Search

Set up a todo list with your start node. Until the todo list is empty:

1. Take the next vertex on the todo list.
2. For each edge leaving the vertex node:
    1. If the edge is explored, leave it alone.
    2. If the edge connects to an explored vertex, mark it as a back edge.
    3. If the vertex is unexplored, mark it explored and add it to the todo list.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

To summarize...

Mark Vertices `UNVISITED`	$O(N)$
Mark Edges `UNVISITED`	$O(M)$
Add each vertex to the queue	$O(N)$
Process all vertices	$O(M)$
	$O(N + M)$

(But requires $O(N)$ memory for the queue)

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS vs BFS

DFS	**BFS**
Long Paths	Shortest Paths
Low Memory	Mid/High Memory

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Other questions

What if we want the path?

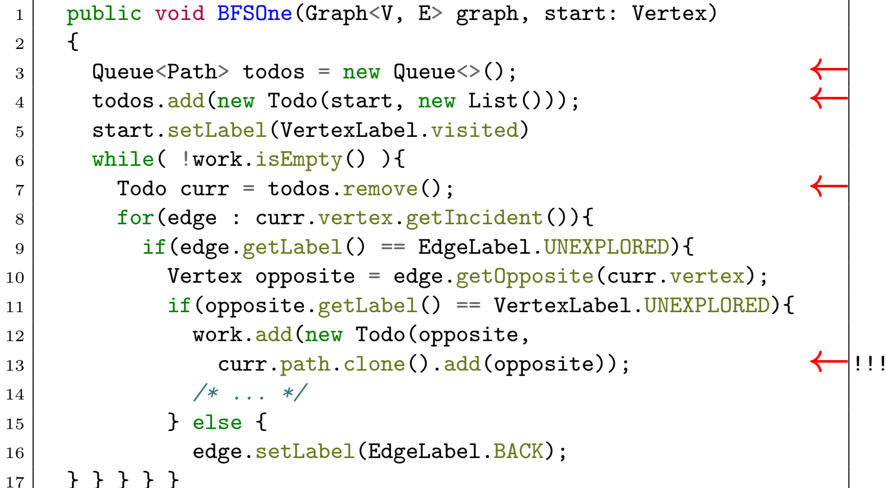
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Todo

```
1  public class Todo
2  {
3    Vertex vertex;
4    List<Vertex> path;
5  }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

BFSOne - Pathfinding



© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

BFSOne - Pathfinding

Problem: "Copying" the path can be $O(N)$

Idea: Store the 'previous hop' for each vertex.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

BFSOne - Pathfinding

```
1  public void BFSOne(Graph<V, E> graph, start: Vertex)
2  {
3    Queue<Vertex> todos = new Queue<>();
4    todos.add(start)
5    start.setLabel(VertexLabel.visited)
6    while( !work.isEmpty() ){
7      Vertex current = todos.remove();
8      for(edge : current.getIncident()){
9        if(edge.getLabel() == EdgeLabel.UNEXPLORED){
10         Vertex opposite = edge.getOpposite(current);
11         if(opposite.getLabel() == VertexLabel.UNEXPLORED){
12           /* ... */
13           prevHops.put(opposite, current);
14         } else {
15           edge.setLabel(EdgeLabel.BACK);
16 } } } } }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

BFSOne - Pathfinding

To compute the path from A to B:

1. B is the last element of the path.
2. `prevHops`$[B]$ is the second-to last element of the path.
3. `prevHops`[`prevHops`$[B]$] is the third-to last element of the path.
4. ...
5. A is the first element of the path.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

How about

What happens if we use a `Stack` instead of a `Queue`?

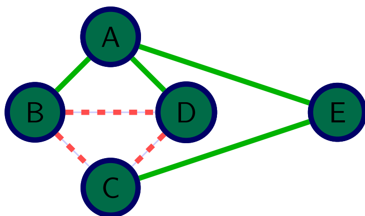
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

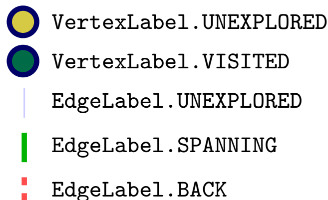
Not Quite BFS

```
1  public void NotQuiteBFSOne(Graph<V, E> graph, start: Vertex)
2  {
3    Stack<Vertex> todos = new Stack<>();
4    todos.add(start)
5    start.setLabel(VertexLabel.visited)
6    while( !work.isEmpty() ){
7      Vertex curr = todos.remove();
8      for(edge : curr.getIncident()){
9        if(edge.getLabel() == EdgeLabel.UNEXPLORED){
10         Vertex opposite = edge.getOpposite(curr);
11         if(opposite.getLabel() == VertexLabel.UNEXPLORED){
12           work.add(opposite);
13           opposite.setLabel(VertexLabel.VISITED);
14           edge.setLabel(EdgeLabel.SPANNING);
15         } else {
16           edge.setLabel(EdgeLabel.BACK);
17 } } } } }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Not quite BFS





Call Stack

`NotQuiteBFS(G)`
`NotQuiteBFSOne(G, A)`

Work Stack

A
B
D
E
C

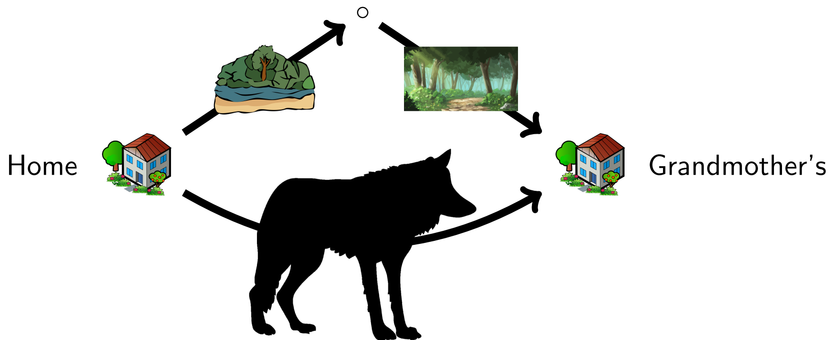
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS

DFS is BFS with a Stack

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Shortest Path



openclipart.org

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Shortest Path

Problem: BFS always finds the path with the *Fewest Edges*

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Weighted Graphs

A **weighted graph** is:

- A graph $G = (V, E)$
- A weight function $\omega(e)$ that assigns a real (called a **edge weight**) to each edge $e \in E$.

Examples of Weighted Graphs

- Latency of a network connection
- Distance between two cities
- Time between two subway/bus stops
- Flow capacity between two points in a series of tubes.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Shortest Path

Given

- A weighted graph $G = (V, E, \omega)$
- A start vertex `start` $\in V$
- A end vertex `end` $\in V$

Goal

- Produce a simple path P from `start` to `end`
- ... that minimizes the sum of edge weights in P.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

BFS with Level

Set up a todo list with [start node, level 0]. Until the todo list is empty:

1. Take the next [vertex, level] on the todo list.
   - **Consequence:** Dequeue reads vertices in ascending order of level.
   - **Therefore:** The first time we reach a vertex, it is via the fewest number of edges.
2. For each edge leaving the vertex node:
   1. If the edge is explored, leave it alone.
   2. If the edge connects to an explored vertex, mark it as a back edge.
   3. If the other vertex is unexplored, mark it explored and add [other, level+1] to the todo list.
      - BFS always adds 1 to the level when exploring a new node; One edge adds 1 to the level.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

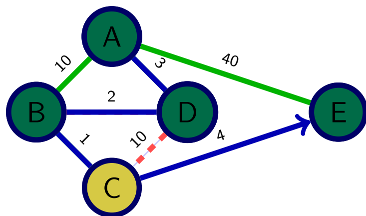
Shortest Path

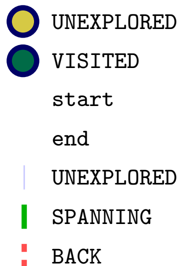
In this example, the 'distance' is based on the number of edges. Each edge has a 'weight' of one. ($\omega(e) = 1$ for all $e \in E$)

What if we let ω vary??

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

BFS with Varying ω





How do we find this path?

Call Stack

BFS(G)
BFSOne(G, A)

Work Queue

A
B
D
E

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Shortest Path

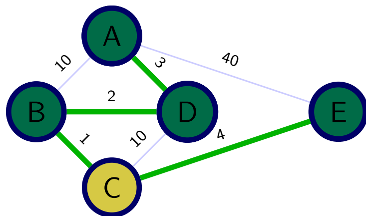
Question: How do we find the shortest path, when not every edge is created equal?

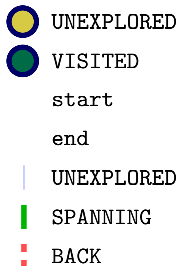
At any given point, what vertex should we explore next?

Idea: Explore the smallest edge available.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Attempt 1: Explore the Shortest Edge





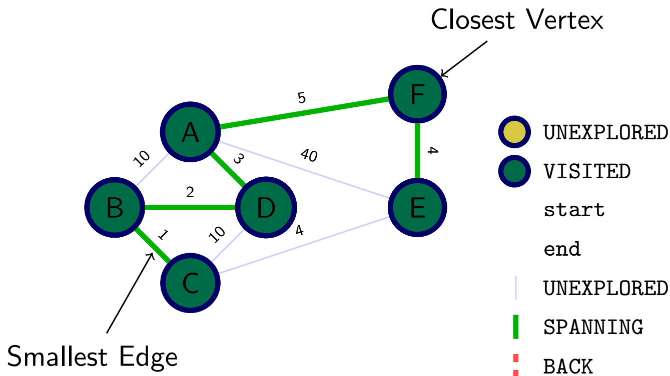
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Attempt 1: Explore the Shortest Edge

Question: Will exploring the shortest available edge always work?

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Attempt 1: Explore the Shortest Edge



© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Dijkstra's Algorithm

Set up a todo list with [start node, level 0]. Until the todo list is empty:

1. Take the next [vertex, level] on the todo list in ascending order of level... but how?.
2. For each edge leaving the vertex node:
    1. If the edge is explored, leave it alone.
    2. If the edge connects to an explored vertex, mark it as a back edge.
    3. If the other vertex is unexplored, mark it explored and add [other, level+1ω(edge)] to the todo list.

This is **Dijkstra's Algorithm**.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

New ADT: Priority Queue

`PriorityQueue<E>` (`E` must be `Comparable``Comparable`)

- `public void add(E e)`: Add e to the queue.
- `public E peek()`: Return the *leastleast* element added.
- `public E remove()`: Remove and return the *leastleast* element added.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY