CSE 322: Shortest Paths

Richard Anderson, Steve Seitz
Winter 2014

Announcements (3/5/14)

- HW 7 due today
- HW 8 out today
- Reading for this lecture: Chapter 9.

Wrapping up concurrency

Locking a Hashtable

- Consider a hashtable with
  - many simultaneous **`lookup`** operations
  - rare **`insert`** operations
- What's the right locking strategy?

Read vs. Write Locks

- Recall race conditions
  - two simultaneous write to same location
  - one write, one simultaneous read
- But two simultaneous reads OK
- Synchronize is too strict
  - blocks simultaneous reads

Readers/Writer Locks

A new synchronization ADT: The readers/writer lock

- A lock's states fall into three categories:
  - "not held"
  - "held for writing" by one thread
  - "held for reading" by *one or more* threads

$0 \leq$ writers ≤ 1
$0 \leq$ readers
writers*readers==0

- **`new:`** make a new lock, initially "not held"
- **`acquire_write:`** block if currently "held for reading" or "held for writing", else make "held for writing"
- **`release_write:`** make "not held"
- **`acquire_read:`** block if currently "held for writing", else make/keep "held for reading" and increment *readers count*
- **`release_read:`** decrement readers count, if 0, make "not held"

In Java

- Java's `synchronized` statement does not support readers/writer
- Instead, library
- `java.util.concurrent.locks.ReentrantReadWriteLock`
- Different interface: methods **`readLock`** and **`writeLock`** return objects that themselves have **`lock`** and **`unlock`** methods

7

Concurrency Summary

- Parallelism is powerful, but introduces new concurrency issues:
  - Data races
  - Interleaving
  - Deadlocks
- Requires synchronization
  - Locks for mutual exclusion
- Guidelines for correct use help avoid common pitfalls

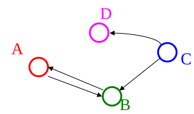
8

Back to graph theory

Graphs

- A formalism for representing relationships between objects

  – Graph **G = (V,E)**
  – *Set of vertices*:
  V = {v$_1$**,v**$_2$**,…,v**$_n$**}**
  – *Set of edges*:
  E = {e$_1$**,e**$_2$**,…,e**$_m$**}**
  where each **e**$_i$ connects one
  – vertex to another **(v**$_j$**,v**$_k$**)**




```
V = {A, B, C, D}
E = {(C, B),
     (A, B),
     (B, A),
     (C, D)}
```

- For *directed edges*, **(v**$_j$**,v**$_k$**)** and **(v**$_k$**,v**$_j$**)** are distinct. (More on this later…)

10

Paths and connectivity

11

The Shortest Path Problem

Given a graph *G*, and vertices *s* and *t* in *G*, find the shortest path from *s* to *t*.

Two cases: weighted and unweighted.

For a path $p = v_0\ v_1\ v_2\ \ldots\ v_k$

– *unweighted length* of path $p = k$ (a.k.a. *length*)

– *weighted length* of path $p = \sum_{i=0..k-1} c_{i,i+1}$ (a.k.a. *cost*)

12

<u>Single Source</u> Shortest Paths (SSSP)

Given a graph *G* and vertex *s*, find the shortest paths from *s* to <u>all</u> vertices in G.

- How much harder is this than finding single shortest path from s to t?

13

Variations of SSSP

- Weighted vs. unweighted
- Directed vs undirected
- Cyclic vs. acyclic
- Positive weights only vs. negative weights allowed
- Shortest path vs. longest path
- …

14

Applications

- Network routing
- Driving directions
- Cheap flight tickets
- Critical paths in project management (see textbook)
- …

15

SSSP: Unweighted Version

16

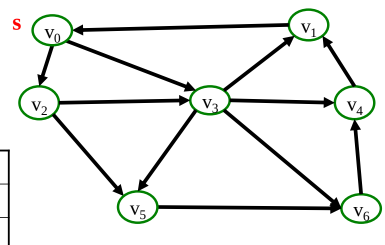
```
void Graph::unweighted (Vertex s){
  Queue q(NUM_VERTICES);
  Vertex v, w;
  q.enqueue(s);
  s.dist = 0;

  while (!q.isEmpty()){
    v = q.dequeue();
    for each w adjacent to v
      if (w.dist == INFINITY){
        w.dist = v.dist + 1;
        w.prev = v;
        q.enqueue(w);
      }
    }
  }
```

each edge examined at most once – if adjacency lists are used

each vertex enqueued at most once

total running time: O(   )

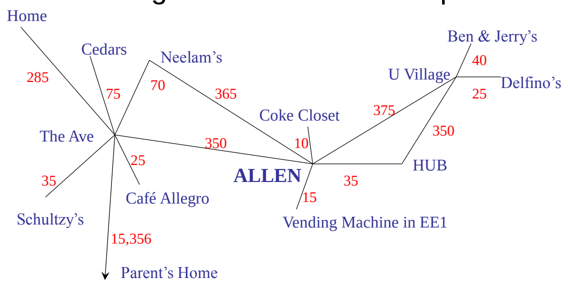
17



V	Dist	prev
v0		
v1		
v2		
v3		
v4		
v5		
v6		

18

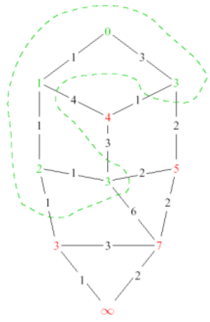
Weighted SSSP:
All edges are not created equal



Can we calculate shortest distance to all vertices from Allen Center?

19

Dijkstra's Algorithm: Idea

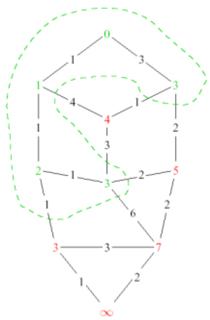


Adapt BFS to handle weighted graphs

Two kinds of vertices:
- Known
  - shortest distance is already known
- Unknown
  - Have tentative distance

20

Dijkstra's Algorithm: Idea



At each step:
1) Pick closest unknown vertex
2) Add it to known vertices
3) Update distances

21

Dijkstra's Algorithm: Pseudocode

Initialize the cost of each node to ∞
Initialize the cost of the source to 0

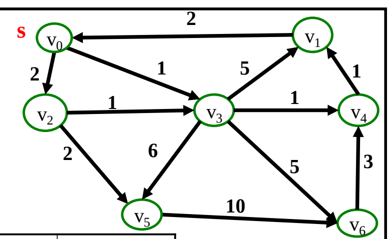
While there are unknown vertices left in the graph
  Select an unknown vertex ***a*** with the lowest cost
  Mark ***a*** as known
  For each vertex ***b*** adjacent to ***a***
    newcost = cost(**a**) + cost(**a**,**b**)
    if (newcost < cost(**b**))
      cost(**b**) = newcost
      previous(**b**) = **a**

22

Important Features

- Once a vertex is known, the cost of the shortest path to that vertex is known
- While a vertex is still unknown, another shorter path to it might still be found
- The shortest path can found by following the previous pointers stored at each vertex

23



V	Known?	Cost	Previous
v0			
v1			
v2			
v3			
v4			
v5			
v6			

24

Dijkstra's Alg: Implementation

Initialize the cost of each vertex to ∞
Initialize the cost of the source to 0
While there are unknown vertices left in the graph
  Select the unknown vertex ***a*** with the lowest cost
  Mark ***a*** as known
  For each vertex ***b*** adjacent to ***a***
    newcost = min(cost(***b***), cost(***a***) + cost(***a***, ***b***))
    if newcost < cost(**b**)
      cost(***b***) = newcost
      previous(***b***) = *a*

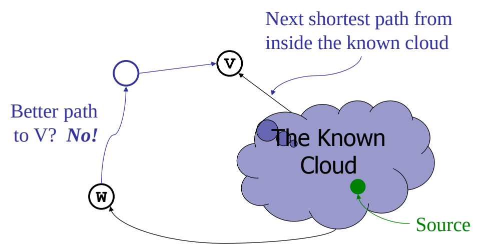
What data structures should we use?

Running time?

Dijkstra's Algorithm: Summary

- Classic algorithm for solving SSSP in weighted graphs *without negative weights*
- A *greedy* algorithm (irrevocably makes decisions without considering future consequences)
- Why does it work?

Correctness: The Cloud Proof



How does Dijkstra's decide which vertex to add to the Known set next?

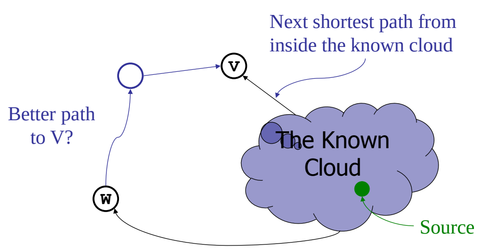
- If path to **V** is shortest, path to **W** must be *at least as long*
  (or else we would have picked **W** *as the next vertex)*
- So the path through **W** to **V** cannot be any shorter!

Correctness: Inside the Cloud

Prove by induction on # of nodes in the cloud:
  Initial cloud is just the source with shortest path 0
  Assume: Everything inside the cloud has the correct shortest path
  Inductive step: by argument on previous slide, we can safely add min-cost vertex to cloud

When does Dijkstra's algorithm not work?

Negative Weights?



How does Dijkstra's decide which vertex to add to the Known set next?

- If path to **V** is shortest, path to **W** must be *at least as long*
  (or else we would have picked **W** *as the next vertex)*
- So the path through **W** to **V** cannot be any shorter!

Dijkstra for BFS

- You can use Dijkstra's algorithm for BFS
- Is this a good idea?