Objectives

- Discuss iterative formulation of the explore() sub-routine for DFS
- Find shortest paths
- Formulate problems as graph problems on weighted graphs
- Introduce Dijkstra’s algorithm

Recall: Chapter 3 vs. Chapter 4

- Chapter 3 is all about connectivity in graphs
  - Formulating problems as graphs
  - Can I get from a to b?
- Chapter 4 is all about paths in graphs
  - Formulating problems as weighted graphs
  - What’s the cheapest (e.g., shortest) path from a to b?

Graph Exploration

```
procedure explore (G, v)
Input: Graph G = (V, E), directed or undirected; vertex v ∈ V
Output: For all vertices u reachable from v, visited(u) is set to true.

visited(v) = true
previsit(v)
for each edge (v,u) in E
  if not visited(u) then explore(u)
postvisit(v)
```

How to make iterative?
- Use a stack
  - Maintaining stack as we go
What property must you preserve?
- Graph traversal order
  - Some pre- & post-

Stack
Iterative Explore() for DFS

procedure exploreIter (G, v)
Input: Graph G = (V, E), directed or undirected; vertex v ∈ V
Output: For all vertices u reachable from v, visited(u) is set to true.

Let K be an empty stack
visited(v) = true
K.push(v) // K contains only v

while K is not empty:

while there exists node w adjacent to u=top(K) // i.e., an edge (u,w)
(in ascending order) such that visited(w) do
visited(w) = true
K.push(w)

x = pop(K)
postvisit(x)

DFS for Shortest Paths?

Edges as Strings

DFS for Shortest Paths?

Edges as Strings

Figure 4.8 Breadth-first search.

procedure bfs(G, x)
Input: Graph G = (V, E), directed or undirected; vertex x ∈ V
Output: For all vertices u reachable from x, dist(u) is set to the distance from x to u.

for all u ∈ V:
dist(u) = ∞
prev(u) = nil

dist(x) = 0
Q = [x] (queue containing just x)
while Q is not empty:
    u = dequeue(Q) // dequeue
    for all edges (u,v) ∈ E:
        if dist(v) = ∞:
            inject(Q, v) // enqueue
            dist(v) = dist(u) + 1
            prev(v) = u

BFS
BFS Example

Additionally, track the prev / parent attribute

Analysis: BFS

Figure 4.3: Breadth-first search.

Correct?

- Idea: layers
- For each distance $d = 0, 1, 2, ..., \text{there is a moment at which}$
  1. all nodes at distance $\leq d$ from $s$ have their distances correctly set
  2. the queue contains exactly the nodes at distance $d$
  3. all other nodes have their distances set to infinity
- How to prove?

3 Questions

- Is it correct?
- How long does it take? (done)
- Can we do better?

BFS

procedure bfs(G; s)
Input: Graph $G = (V; E)$, directed or undirected; vertex $s \in V$
Output: For all vertices $u$ reachable from $s$, dist($u$) is set to the distance from $s$ to $u$.

for all $u \in V$:
  dist($u$) = $\infty$
  prev($u$) = nil

dist(s) = 0
Q = [s] \// queue containing just s

while Q is not empty:
  $u = \text{eject}(Q)$
  for all edges $(u; v) \in E$:
    if dist($v$) == $\infty$:
      $\text{inject}(Q; v)$
      dist($v$) = dist($u$) + 1
      prev($v$) = $u$

The “Agenda” is a Queue

Similarity to Iterative Explore() for DFS

procedure explore(G; s)
Input: Graph $G = (V; E)$, directed or undirected; vertex $s \in V$
Output: For all vertices $u$ reachable from $s$, visited($u$) is set to true.

for all $u \in V$:
  visited($u$) = false

previsit(s)
visited(s) = true
K = [s] \// stack containing just s

while K is not empty:
  while there exists node $v$ adjacent to top(K) in ascending order such that visited($v$) do
    prev($v$) = true
    push(K; v)
  end
  v = pop(K)
  postvisit(v)

The “Agenda” is a Stack
Formulating problems as problems on graphs

- In designing a graph-based formulation of a problem, there are three main decisions to be made:
  - What is stored in each node?
  - What does it mean for two nodes to be connected by an edge?
  - What property should be extracted from the graph?

Formulating problems as problems on weighted graphs

- In designing a graph-based formulation of a problem, there are four main decisions to be made:
  - What is stored in each node?
  - What does it mean for two nodes to be connected by an edge?
  - What does the weight on an edge represent?
  - What property should be extracted from the graph?
- What kinds of problems make sense as weighted graphs?

Distance as Weighted Edges

Use BFS?

Dummy Vertices

Good Idea?
“Alarm Clocks”

- Set an alarm clock for node $x$ at time $t$.
- Repeat until there are no more alarms:
  - Say the next alarm goes off at time $t$, for node $x$. Then:
    - The distance from $s$ to $x$ is $t$.
    - For each neighbor $y$ of $x$ in $G$:
      - If $y$ is already set for some time $t' < t$, then reset it to this earlier time.
  Do we really need the ticking clock $T$?

### Implementation

- We just care about the “alarm clocks” going off in order.
- What do we need to manage them?
- Priority Queue

#### Operation | Description
--- | ---
Insert | Add a new element to the set (also: “Enqueue”, “Inject”)
Decrease-key | Accommodate the decrease in key value of a particular element
Delete-min | Return the element with the smallest key, and remove it from the set (also: “Dequeue”, “Eject”)
Top | Return the element with the smallest key
Make-queue | Build a priority queue out of the given elements, with the given key values

*In many implementations, this is significantly faster than inserting the elements one by one*

### Dijkstra’s

**Figure 9.9 Dijkstra’s shortest path algorithm.**

- **Input:** Graph $G = (V,E)$, directed or undirected; positive edge lengths $|e| = |(u,v)|$, vertex $v \in V$
- **Output:** For all vertices $u$ reachable from $s$, $dist(u)$ is set to the distance from $s$ to $u$.

```
for all u \in V:
    dist(u) = \infty
    pred(u) = nil
H = \text{makeheap}(\{ (u, \text{dist}(u)) \}) [using \text{dist-values as keys}]
while H is not empty:
    u = \text{decreasekey}(H)
    for all edges (u, v) \in E:
        if dist(u) + |(u,v)| < dist(v):
            dist(v) = dist(u) + |(u,v)|
            pred(v) = u
            \text{decreasekey}(H, v)
```

#### Efficiency?

See next lecture!

### Example

**The “Agenda” is a Priority Queue**

<table>
<thead>
<tr>
<th>Priority Queue</th>
<th>A:0 B:∞ C:∞ D:∞ E:∞</th>
</tr>
</thead>
</table>

**Length of shortest paths:**

<table>
<thead>
<tr>
<th>Priority Queue</th>
<th>A:0 B:4 C:2</th>
</tr>
</thead>
</table>

**Example**

<table>
<thead>
<tr>
<th>Priority Queue</th>
<th>A:0 B:3 C:7 D:6 E:7</th>
</tr>
</thead>
</table>

**Length of shortest paths:**

<table>
<thead>
<tr>
<th>Priority Queue</th>
<th>A:0 B:3 C:7 D:6 E:7</th>
</tr>
</thead>
</table>
Example

Length of shortest paths:

| Priority Queue | E:6 |

Example

Length of shortest paths:

| Priority Queue | E:6 |

Example

Length of shortest paths:

| Priority Queue | empty |

Next:

- Alternate formulation of Dijkstra’s
- Discussion of possible Priority Queue implementations

Assignment

- HW #12.5