Paths in graphs

The classic 15-puzzle

Graph $G = (V, E)$
$V$ = {configurations of puzzle}
$E$: edges between neighboring configurations

$explore(G, a)$: Finds a path from $a$ to $i$. But this isn't the shortest possible path!

Distances in graphs

Distance between two nodes = length of shortest path between them

Physical model:
Vertex = ping-pong ball
Edge = piece of string

Suppose we want to compute distances from some starting node $s$ to all other nodes in $G$.
Strategy: layer-by-layer
First, nodes at distance 0
Then, nodes at distance 1
Then, nodes at distance 2, etc.

Breadth-first search

Suppose we have seen all nodes at distance $d$.
How to get the next layer?
Solution:
A node is at distance $d+1$ if:
it is adjacent to some node at distance $d$
it hasn't been seen yet

Why does BFS work?

Two search strategies

Depth-first

Breadth-first

Running time: $O(V + E)$, like DFS
Edge lengths

BFS treats all edges as having the same length. This is rarely true in applications.

Denote the length of edge $e = (u, v)$ by $l(e)$ or $L_e$ or $l(u, v)$.

Extending BFS

Suppose $G$ has positive integral edge lengths.

- $G'$ has unit-length edges.
- For the "real" nodes, distance in $G = \text{distance in } G'$.
- So run BFS on $G'$.

**Problem: efficiency**

If edge lengths in $G$ are large:
- $G'$ is enormous.
- BFS wastes a lot of time computing distances to dummy nodes we don't care about.

Extend BFS

**First 99 time steps:** BFS (on $G$) slowly advances along $a \rightarrow b$ and $a \rightarrow c$. Boring!

Can we snooze and have an alarm wake us whenever BFS reaches a real node?

**Alarm clock algorithm**

**Given graph $G$ and starting node $s$**

- Set an alarm for node $s$ at time $0$.
- If the next alarm goes off at time $T$, for node $u$:
  - $\text{distance}(u) = T$
  - For each edge $(u, v) \in E$:
    - If no alarm for $v$, set one at $T + l(u, v)$.
    - If there is an alarm for $v$, but later than $T + l(u, v)$, then reset to this earlier time.

Exactly simulates BFS on $G$, we no longer need to construct $G'$.

**How to implement alarm?**

**Answer:** priority queue (aka heap)

A priority queue $H$ stores:
- A set of elements (our nodes)
- Associated key values (alarms times)

and supports these operations:

- `insert(H, x)`
  - Insert a new element into $H$.
  - Set a new alarm.
- `deletemin(H)`
  - Return element with smallest key.
  - Remove from $H$ which alarm is going off next?
- `decreasekey(H, x)`
  - Allow $x$'s key value to be decreased.
  - Allow alarm to be reset to an earlier time.
- `makequeue(S)`
  - Make a queue out of the elements in $S$ (and their keys).

**Alarm clock algorithm**

```
G

T = 0
set alarms for b (300), c (100)

T = 100
wake up BFS in a, c
set alarm for b (300), d (700)

T = 300
wake up, BFS is at b
set alarm for c (100)

T = 500
wake up, BFS is at d

dist[c] = 100
dist[b] = 300
dist[d] = 500
```

**Alarm:** estimated time of arrival based on edges currently being traversed.

<table>
<thead>
<tr>
<th>Alarm</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>300</td>
</tr>
<tr>
<td>c</td>
<td>100</td>
</tr>
<tr>
<td>d</td>
<td>700</td>
</tr>
</tbody>
</table>

**Extending BFS**

1. Suppose $G'$ has unit-length edges.
2. For the "real" nodes, distance in $G = \text{distance in } G'$.
3. So run BFS on $G'$.

**Problem: efficiency**

If edge lengths in $G$ are large:
- $G'$ is enormous.
- BFS wastes a lot of time computing distances to dummy nodes we don't care about.

Extend BFS

**First 99 time steps:** BFS (on $G'$) slowly advances along $a \rightarrow b$ and $a \rightarrow c$. Boring!

Can we snooze and have an alarm wake us whenever BFS reaches a real node?

**Alarm clock algorithm**

**Given graph $G$ and starting node $s$**

- Set an alarm for node $s$ at time $0$.
- If the next alarm goes off at time $T$, for node $u$:
  - $\text{distance}(u) = T$
  - For each edge $(u, v) \in E$:
    - If no alarm for $v$, set one at $T + l(u, v)$.
    - If there is an alarm for $v$, but later than $T + l(u, v)$, then reset to this earlier time.

Exactly simulates BFS on $G'$, we no longer need to construct $G'$.

**How to implement alarm?**

**Answer:** priority queue (aka heap)

A priority queue $H$ stores:
- A set of elements (our nodes)
- Associated key values (alarms times)

and supports these operations:

- `insert(H, x)`
  - Insert a new element into $H$.
  - Set a new alarm.
- `deletemin(H)`
  - Return element with smallest key.
  - Remove from $H$ which alarm is going off next?
- `decreasekey(H, x)`
  - Allow $x$'s key value to be decreased.
  - Allow alarm to be reset to an earlier time.
- `makequeue(S)`
  - Make a queue out of the elements in $S$ (and their keys).

**Alarm clock algorithm**

```
G

T = 0
set alarms for b (300), c (100)

T = 100
wake up, BFS is at b
set alarm for c (100)

T = 200
wake up, BFS is at b
set alarm for d (700)

T = 300
wake up, BFS is at c
set alarm for b (300)

T = 500
wake up, BFS is at d

dist[c] = 100
dist[b] = 300
dist[d] = 500
```

**Alarm:** estimated time of arrival based on edges currently being traversed.

<table>
<thead>
<tr>
<th>Alarm</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>300</td>
</tr>
<tr>
<td>c</td>
<td>100</td>
</tr>
<tr>
<td>d</td>
<td>700</td>
</tr>
</tbody>
</table>

**Extending BFS**

**First 99 time steps:** BFS (on $G'$) slowly advances along $a \rightarrow b$ and $a \rightarrow c$. Boring!

Can we snooze and have an alarm wake us whenever BFS reaches a real node?

**Alarm clock algorithm**

**Given graph $G$ and starting node $s$**

- Set an alarm for node $s$ at time $0$.
- If the next alarm goes off at time $T$, for node $u$:
  - $\text{distance}(u) = T$
  - For each edge $(u, v) \in E$:
    - If no alarm for $v$, set one at $T + l(u, v)$.
    - If there is an alarm for $v$, but later than $T + l(u, v)$, then reset to this earlier time.

Exactly simulates BFS on $G'$, we no longer need to construct $G'$.

**How to implement alarm?**

**Answer:** priority queue (aka heap)

A priority queue $H$ stores:
- A set of elements (our nodes)
- Associated key values (alarms times)

and supports these operations:

- `insert(H, x)`
  - Insert a new element into $H$.
  - Set a new alarm.
- `deletemin(H)`
  - Return element with smallest key.
  - Remove from $H$ which alarm is going off next?
- `decreasekey(H, x)`
  - Allow $x$'s key value to be decreased.
  - Allow alarm to be reset to an earlier time.
- `makequeue(S)`
  - Make a queue out of the elements in $S$ (and their keys).

**Alarm clock algorithm**

```
G

T = 0
set alarms for b (300), c (100)

T = 100
wake up, BFS is at b
set alarm for c (100)

T = 200
wake up, BFS is at b
set alarm for d (700)

T = 300
wake up, BFS is at c
set alarm for b (300)

T = 500
wake up, BFS is at d

dist[c] = 100
dist[b] = 300
dist[d] = 500
```

**Alarm:** estimated time of arrival based on edges currently being traversed.

<table>
<thead>
<tr>
<th>Alarm</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>300</td>
</tr>
<tr>
<td>c</td>
<td>100</td>
</tr>
<tr>
<td>d</td>
<td>700</td>
</tr>
</tbody>
</table>