Lecture 11 – Dijkstra’s Algorithm

Sanjoy Dasgupta
Russell Impagliazzo
Ragesh Jaiswal
CSE101, Spring 2020, Week-03
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

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

Problem: efficiency

Simple trick: add dummy nodes

If edge lengths in $G$ are large:
(i) $G'$ is enormous
(ii) BFS wastes a lot of time computing distances to dummy nodes we don’t care about
Extending BFS

First 99 time steps: BFS (on G’) slowly advances along a—b and a—c. Boring!

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

Alarm for each real node: estimated time of arrival based on edges currently being traversed.

T = 0  set alarms for b (500), c (100)  snooze
T = 100 wake up, BFS is at c  set alarms for b (300), d (700)  snooze
T = 300 wake up, BFS is at b  set alarm for d (500)  snooze
T = 500 wake up, BFS is at d

\[
\begin{align*}
\text{dist}[c] &= 100 \\
\text{dist}[b] &= 300 \\
\text{dist}[d] &= 500
\end{align*}
\]
(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:
    distance[u] = T
for each edge (u,v) in E:
    if no alarm for v, set one for 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 (alarm times)
and supports these operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert(H,x)</td>
<td>insert new element into H</td>
</tr>
<tr>
<td>deletemin(H)</td>
<td>return element with smallest key value, remove from H</td>
</tr>
<tr>
<td>decreasekey(H,x)</td>
<td>allow x’s key value to be decreased</td>
</tr>
<tr>
<td>makequeue(S)</td>
<td>make a queue out of the elements in S (and their keys)</td>
</tr>
</tbody>
</table>

How to implement alarm?
Answer: priority queue (aka heap)

A priority queue H stores:
- a set of elements (our nodes)
- associated key values (alarm times)
and supports these operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert(H,x)</td>
<td>insert new element into H</td>
</tr>
<tr>
<td>deletemin(H)</td>
<td>return element with smallest key value, remove from H</td>
</tr>
<tr>
<td>decreasekey(H,x)</td>
<td>allow x’s key value to be decreased</td>
</tr>
<tr>
<td>makequeue(S)</td>
<td>make a queue out of the elements in S (and their keys)</td>
</tr>
</tbody>
</table>

Example:

```
G
\[\begin{array}{ccccc}
\text{a} & \text{b} & \text{c} & \text{d} \\
500 & 100 & 200 & 200 \\
200 & 600 & & \\
\end{array}\]
```
Dijkstra’s algorithm

procedure dijkstra(G,l,s)

input: graph G = (V,E); node s; positive edge lengths l_e
output: for each node u, dist[u] is set to its distance from s

for u in V:
    dist[u] = ∞
dist[s] = 0
H = makequeue(V)  // key = dist[

while H is not empty:
    u = deletemin(H)
    for each edge (u,v) in E:
        if dist[v] > dist[u] + l(u,v):
            dist[v] = dist[u] + l(u,v)
            decreasekey(H,v)
procedure dijkstra(G, l, s)

for u in V:
    dist[u] = ∞
    prev[u] = nil
dist[s] = 0
H = makequeue(V)  // key = dist[]

while H is not empty:
    u = deletemin(H)
    for each edge (u, v) in E:
        if dist[v] > dist[u] + l(u,v):
            dist[v] = dist[u] + l(u,v)
            prev[v] = u
            decreasekey(H, v)
Running time

procedure dijkstra(G,l,s)

for u in V:
    dist[u] = \infty

dist[s] = 0

H = makequeue(V)  // key = dist[]

while H is not empty:
    u = deletemin(H)
    for each edge (u,v) in E:
        if dist[v] > dist[u] + l(u,v):
            dist[v] = dist[u] + l(u,v)
            decreasekey(H,v)

Time:
O(V + E) + 
V x deletemin +  
V x insert +  
E x decreasekey