CS 312: Algorithm Design & Analysis

Lecture #20: Correctness and Efficiency of Dijkstra’s Algorithm

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.

Slides by: Eric Ringger, adapting figures from Dasgupta et al.

Objectives

- Discuss correctness of Dijkstra’s algorithm
- Examine possible Priority Queue implementations and their impact on efficiency of Dijkstra’s algorithm
- Update Dijkstra’s to explore on the fly

3 Questions

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

Correctness of Dijkstra’s

Context: Graphs with non-negative edge weights

Question: once a vertex $u$ has been settled, is it possible that we might later find some vertex $w$ that leads to $u$ and via which an even shorter path to $u$ can be found?

Correctness of Dijkstra’s

Context: Graphs with non-negative edge weights

Assume $3 u \in V$

$s \neq R \text{ and } \text{len}(w \rightarrow u) < \text{dist}(u)$

Since $\text{len}(w \rightarrow u) \geq 0$, $\text{dist}(w) < \text{dist}(u)$

Thus, $w \in R$.

Analysis:

- makequeue
- deletemin
- decreasekey

Dijkstra’s

Figure 4.8 Dijkstra’s shortest-path algorithm

procedure dijkstra($G$, $s$)
Input: Graph $G = (V, E)$, directed or undirected, positive edge lengths $\text{len} : E \rightarrow \mathbb{R}$, vertex $s \in V$
Output: For all vertices $v$ reachable from $s$, dist(v) is set to the distance from $s$ to $v$.

for all $v \in V$

$\text{prev}(v) = s$

$\text{dist}(v) = 0$

$M = \text{makequeue}(\{s\})$ (using dist-values as keys)

while $M$ is not empty

$u = \text{deletemin}(M)$

for all edges $(u, v) \in E$

if $\text{dist}(u) < \text{dist}(v) + \text{len}(u, v)$

$\text{dist}(v) = \text{dist}(u) + \text{len}(u, v)$

$\text{prev}(v) = u$

$\text{decreasekey}(v, \text{dist}(v))$
Summary of Cost

- Dijkstra’s requires:
  - \(1 \text{ makequeue } \leq |V| \text{ insert}\)
  - \(|V| \text{ deletemin}\)
  - \(|E| \text{ decreasekey operations}\)
- Insert and decreasekey operations take the same time in priority queues we examine.
  - \(|V| \text{ deletemin}\)
  - \(|V| + |E| \text{ insert (or decreasekey) operations}\)
- How much time do those operations require?

Priority Queue Implementations

| Implementation          | deletemin | insert/decreasekey | \(|V| \times \text{deletemin} + (|V| + |E|) \times \text{ insert}) |
|------------------------|-----------|--------------------|---------------------------------------------------------------|
| Unsorted Array         | \(O(|V|)\) | \(O(1)\)           | \(O(|V| + |E|) \times \text{ insert}) \)                      |
| Binary heap            | \(O(\log |V|)\) | \(O(\log |V|)\) | \(O(|V| \log |V|) + |E| \log |V|) \)                          |
| d-ary heap             | \(O(\log |V|)\) | \(O(\log |V|)\) | \(O(|V| \log |V|) + |E| \log |V|) \)                          |
| Fibonacci heap         | \(O(1)\) | \(O(1)\)           | \(O(|V| \log |V|) + |E| \log |V|) \)                          |

Implications for Dijkstra’s Algorithm

- Dense Graph: \(|E| = O(|V|^2)\)
  - Unsorted Array: \(O(|V|^2)\)
  - Binary heap: \(O(|V|^2 \log |V|)\)
  - Fibonacci heap: \(O(|V|^2)\)
- Sparse Graph: \(|E| = O(|V|)\)
  - Unsorted Array: \(O(|V| \log |V|)\)
  - Binary heap: \(O(|V| \log |V|) \checkmark\)
  - Fibonacci heap: \(O(|V| \log |V|) \checkmark\)
- Punch-line: data structures matter; input matters

Project #4

- What does the graph in proj. #4 look like?
- Is it sparse or dense? \(|E| = 4 \times |V| = O(|V|)\)
- Your answer should inform your choice of priority queue.

Back to the algorithm:

```
\text{procedure dijkstra}(G,v)\;
\text{Input:} \ G \text{- a graph, } v \text{- a vertex; } G \text{- connected or not connected;}
\text{Output:} \ \text{compute distances from } v \text{ to } G \text{- reachable vertices.}
\text{dist}(v) = 0; \ \text{prev}(v) = v; \ \text{for all } u \notin V, \ \text{dist}(u) = \infty; \ \text{prev}(u) = \text{null};
\text{while } N \text{ is not empty} \{
    n = \text{deletemin}(N); \ 
    \text{for all edges } (v,n) \in E, \text{ if dist}(v) > \text{dist}(n) + w(n,v); \ 
    \text{dist}(v) = \text{dist}(n) + w(n,v); \ 
    \text{prev}(v) = n; \ 
    \text{decreasekey}(N,v)\}
```
On-the-Fly Dijkstra’s

- Should we add all vertices at once?
- What if we encounter a vertex again that has already been settled?
- Same inputs and outputs:
  - Input: Graph $G = (V,E)$, directed or undirected; non-negative edge lengths $l(e): e \in E$; starting vertex $s \in V$
  - Output: For all vertices $u$ reachable from $s$, $\text{dist}(u)$ is set to the shortest distance from $s$ to $u$.

**On-the-Fly Dijkstra’s**

**procedure** `dijkstraonthefly` ($G$, $l$, $s$)

for all $u \in V$:
  dist($u$) = $\infty$; prev($u$) = nil
H.insert($s$)

while !H.empty():
  $u$ = H.deletemin()
SettledSet.insert($u$)
  for all edges $(u,v) \in E$:
    if !SettledSet.contains($v$):
      if !H.contains($v$):
        dist($v$) = dist($u$) + $l(u,v)$; prev($v$) = $u$; H.insert($v$)
      else if dist($v$) > dist($u$) + $l(u,v)$:
        dist($v$) = dist($u$) + $l(u,v)$; prev($v$) = $u$; H.decreasekey($v$)

Goal(s)?

- For any one starting point, how many goal points do we need the shortest path to?

Special Graph!

Single Goal Dijkstra’s

**procedure** `dijkstraonesecond` ($G$, $l$, $s$, goal)

for all $u \in V$:
  dist($u$) = $\infty$; prev($u$) = nil
dist($s$) = 0
H.insert($s$)

foundgoal = false

while !H.empty() and !foundgoal:
  $u$ = H.deletemin()
if $u$ == goal:
  foundgoal = true
else
  SettledSet.insert($u$)
  for all edges $(u,v) \in E$:
    if !SettledSet.contains($v$):
      if !H.contains($v$):
        dist($v$) = dist($u$) + $l(u,v)$; prev($v$) = $u$; H.insert($v$)
      else if dist($v$) > dist($u$) + $l(u,v)$:
        dist($v$) = dist($u$) + $l(u,v)$; prev($v$) = $u$; H.decreasekey($v$)

In Project #4, you get to prove that this condition is never met when running on our peculiar graph!
**Single Goal Dijkstra’s v3**

procedure dijkstrasinglegoal (G, l, s, goal)

for all u ∈ V:
    dist(u) = ∞; prev(u) = nil
    H.insert(u)
foundgoal = false

while !H.empty() and !foundgoal:
    u = H.deletemin()
    if u == goal:
        foundgoal = true
    else
        SettledSet.insert(u)
        for all edges (u,v) ∈ E:
            if !SettledSet.contains(v) and !H.contains(v):
                dist(v) = dist(u) + l(u,v); prev(v) = u; H.insert(v)

**Single Goal Dijkstra’s v4**

procedure dijkstrasinglegoal (G, l, s, goal)

for all u ∈ V:
    dist(u) = ∞; prev(u) = nil
    H.insert(u)
foundgoal = false

while !H.empty() and !foundgoal:
    u = H.deletemin()
    SettledSet.insert(u)
    for all edges (u,v) ∈ E:
        if v == goal:
            foundgoal = true
        else if !SettledSet.contains(v) and !H.contains(v):
            dist(v) = dist(u) + l(u,v); prev(v) = u; H.insert(v)

At this point, we probably shouldn’t call it Dijkstra’s anymore!

**Extracting the Path**

- Once you’ve found the goal, you want to draw the path back to the starting vertex. How?
- Back-trace from the goal to the start vertex by following the prev() pointers!

**End**