Priority Queues with Decreasing Keys

Gerth Stølting Brodal
Aarhus University
Background

- Bachelor project = shortest paths on Open Street Map graphs
- Students have trouble implementing Dijkstra's algorithm in Java™
Dijkstra’s algorithm

- Non-negative edge weights
- Visits nodes in increasing distance from source

```
proc Dijkstra_1(V, E, δ, s)
    dist[v] = +∞ for all v ∈ V \ {s}
    dist[s] = 0
    Insert(Q, ⟨dist[s], s⟩)
    while Q ≠ ∅ do
        ⟨d, u⟩ = ExtractMin(Q)
        for (u, v) ∈ E ∩ ({u} × V) do
            if dist[u] + δ(u, v) < dist[v] then
                dist[v] = dist[u] + δ(u, v)
                if v ∈ Q then
                    DecreaseKey(Q, v, dist[v])
                else
                    Insert(Q, ⟨v, dist[v]⟩)
        return dist

proc Dijkstra_2(V, E, δ, s)
    dist[v] = +∞ for all v ∈ V \ {s}
    dist[s] = 0
    Insert(Q, ⟨dist[s], s⟩)
    while Q ≠ ∅ do
        ⟨d, u⟩ = ExtractMin(Q)
        for (u, v) ∈ E ∩ ({u} × V) do
            if dist[u] + δ(u, v) < dist[v] then
                dist[v] = dist[u] + δ(u, v)
                if v ∈ Q then
                    DecreaseKey(Q, v, dist[v])
                else
                    Insert(Q, ⟨v, dist[v]⟩)
        return dist
```

Fibonacci heaps ⇒ O(m + n \cdot \log n)

relax

⇒ O(log n) Remove ⇒ O(m \cdot \log n)
The Challenge - Java's builtin binary heap

- no `decreasekey`
- `remove` $O(n)$ time
  \[\Rightarrow\text{ Dijkstra } O(m \cdot n)\]

- comparator function
Repeated insertions

- Relax inserts new copies of item
- Skip outdated items

```
proc Dijkstra_3(V, E, δ, s)
    dist[v] = +∞ for all v ∈ V \ {s}
    dist[s] = 0
    Insert(Q, ⟨dist[s], s⟩)
    while Q ≠ ∅ do
        ⟨d, u⟩ = ExtractMin(Q)
        if d = dist[u] then
            for (u, v) ∈ E ∩ (u × V) do
                if dist[u] + δ(u, v) < dist[v] then
                    dist[v] = dist[u] + δ(u, v)
                    Insert(Q, ⟨dist[v], v⟩)
    return dist
```
Using a visited set

```
proc Dijkstra_4(V, E, δ, s)
    dist[v] = +∞ for all v ∈ V \ {s}
    dist[s] = 0
    visited = ∅
    Insert(Q, ⟨dist[s], s⟩)
    while Q ≠ ∅ do
        ⟨d, u⟩ = ExtractMin(Q)
        if u ∉ visited then
            visited = visited ∪ {u}
            for (u, v) ∈ E ∩ (∅ × V) do
                if dist[u] + δ(u, v) < dist[v] then
                    dist[v] = dist[u] + δ(u, v)
                    Insert(Q, ⟨dist[v], v⟩)
        return dist
```

use bitvector →

![Graph diagram]
A shaky idea...

```plaintext
proc Dijkstra_{4}(V, E, \delta, s)
   dist[v] = +\infty for all v \in V \setminus \{s\}
   dist[s] = 0
   visited = \emptyset
   Insert(Q, \langle dist[s], s \rangle)
   while Q \neq \emptyset do
      u = ExtractMin(Q)
      if u \notin visited then
         visited = visited \cup \{u\}
         for (u, v) \in E \cap (\{u\} \times V) do
            if dist[u] + \delta(u, v) < dist[v] then
               dist[v] = dist[u] + \delta(u, v)
               Insert(Q, \langle dist[v], v \rangle)
      return dist
```

- Only store nodes in \(Q\) (save space)
- Comparator
- Key = current distance \(dist\)

Heap invariants break
Experimental study

- Implemented Dijkstra\textsubscript{\textdagger} in Python
- Stress test on random cliques
- Binary heaps  \textcolor{red}{failed} (default priority queue in Java and Python)
- Skew heaps  \textcolor{green}{worked}
- Leftist heaps  \textcolor{green}{worked}
- Pairing heaps  \textcolor{green}{worked}
- Binomial queues  \textcolor{green}{worked}
- Post-order heaps  \textcolor{green}{worked}
- Binary heaps with top-down insertions  \textcolor{green}{worked}

\begin{verbatim}
visited = set()
Q = Queue()
Q.insert(Item(0, source))
while not Q.empty():
    u = Q.extract_min().value
    if u not in visited:
        visited.add(u)
        for v in G.out[u]:
            dist_v = dist[u] + G.weights[(u, v)]
            if dist_v < dist[v]:
                dist[v] = dist_v
                parent[v] = u
                Q.insert(Item(dist[v], v))
\end{verbatim}
Binary heap insertions – bottom-up vs top-down

Insert(7)

Top-down

Bottom-up
Binary heaps using $\textit{dist}$ in a comparator fails

1. Incorrect placement of a new key.
2. Outdated key.
3. Wrong placement of a new key.
4. Ignored key since visited.

Diagram:
- $s_0$: extract $s_0$, insert $a_1$, insert $b_8$, insert $c_7$, insert $d_5$, insert $e_6$.
- $a_1$: extract $a_1$, insert $c_7$.
- $b_2$: extract $b_2$, insert $c_3$.
- $d_5$: extract $d_5$, insert $d_4$.
- $b_2$: not smallest key.
- $d_5$: ignored since visited.
Definition

Priority Queues with Decreasing Keys

- Items = \langle key, value \rangle
- Over time keys can decrease – priority queue is not informed
- Items are compared w.r.t. their current keys
- The original key of an item is the key when it was inserted

Insert (item)

ExtractMin() returns an item with current key less than or equal to all original keys in the priority queue
Theorem 1

Dijkstra$_4$ correctly computes shortest paths when using $dist$ as current key and a priority queue supporting decreasing keys.

Theorem 2

The following priority queues support decreasing keys (out of the box):

- binary heaps with top-down insertions
- skew heaps
- leftist heaps
- pairing heaps
- binomial queues
- post-order heaps
Proof of Theorem 2 - Basic idea

- Decreased heap order
  \[ u \text{ ancestor of } v \Rightarrow \text{current key } u \leq \text{original key } v \]

- Root valid item to extract

- Top-down merging two paths preserves decreased heap order

  \Rightarrow \text{skew heaps and leftist heaps support decreasing keys}
Experimental evaluation of various heaps

- Cliques with uniform random weights
- With decreasing keys less comparisons (outdated items removed earlier)
Reduction in comparisons

comparisons decreasing keys / comparisons (key, value) pairs
Postorder heap [Harvey and Zatloukal, FUN 2004]

- Insert amortized $O(1)$, ExtractMin amortized $O(\log n)$
- Implicit (space efficient)
- Best implicit comparison performance (and good time performance)
Conclusion

- Introduced notion of **priority queues with decreasing keys**
  ... as an approach to deal with outdated items in Dijkstra’s algorithm
- Experiments identified priority queues supporting decreasing keys
  ... just had to prove it
- Built-in priority queues in Java and Python are binary heaps
  ... do not support decreasing keys
- **Binary heaps with top-down insertions** do support decreasing keys
  ... and also
  skew heaps, leftist heaps, pairing heaps, binomial queues, post-order heaps