6.006 Recitation
Build 2008.25
6.006 Proudly Presents

- Dijkstra: minimum-cost paths on crack
- Algorithm
- Concepts
- Implementation
- Data structures come back from the dead (not talking about the quiz)
Minimum-Path Problem

- Given: graph G, source vertex s, edge costs
- Want: paths from s to everything else with minimum costs (sum of edge costs)
- Approach: let $d[v]$ be upper bounds for the real minimum costs, $\delta[v]
- Start out easy: $d[v] = \infty$, $d[s] = 0$
- Relax until values in $d$ converge to $\delta$
Good Dijkstra

• Generic initialization

• $U = V$

• Choose $v = \text{argmin } d[v]$ in $U$, remove $v$ from $U$

• Notice $d[v] = \delta[v]$

• Relax $v$’s outgoing edges

• Rinse, repeat
Bad Dijkstra

- Generic initialization
- $U = V$
- Choose $v = \text{argmin } d[v \text{ in } U]$, remove $v$ from $U$
- Notice $d[v] = \delta[v]$
- Relax $v$'s outgoing edges
- Rinse, repeat
Dijkstra Overview

- Nice and fast (that’s why it’s on crack)
- With limitations (crack impacts judgement)
  - Doesn’t handle negative-cost edges
  - DOES handle 0-cost edges
- Harder to code than Bellman-Ford
Dijkstra Works:

Intuition
Dijkstra Works: Formal
Making Dijkstra Fast
(it's crack dealer)

- Generic initialization:
  \[ d[v] \leftarrow \infty, \quad d[S] = 0 \]

- Choose \( v = \arg\min d[v] \),
  by now \( d[v] = \delta[v] \)

- Relax all edges going out of \( v \)

- Rinse, repeat

- Computing argmin
  - \( V \) times
  - Relaxing
  - \( E \) times

- Looks like we need a Data Structure
Min-Priority Queues

- **Data Structure**
  - `insert(key)` : adds to the queue
  - `min()` : returns the minimum key
  - `delete-min()` : deletes the min key
  - `delete(key)` : deletes the given key
- optional (only needed in some apps)
Priority Queues with Min-Heaps

- Costs (see above line for explanations)
  - insert: $O(\log(N))$
  - min: $O(1)$
  - delete-min: $O(\log(N))$
  - delete: $O(\log(N))$ - only if given the index of the node containing the key
Priority Queues with PS3

- Is this priority queue monotone?
- Profit
Cool Python: Generators

1. Iterators
   - used in for loops
   - objects implementing `next()`

2. Generators
   - express iterator functionality in a cooler way

```python
1 def counter():
2     i = 0
3     while True:
4         yield i
5     i += 1

----
c = counter()
c.next()
c.next() >> 0
c.next() >> 1
d = counter()
d.next()
>> 0
c.next()
>> 2
d.next()
>> 1
c.next()
>> 3
```
class heap_id:
    def __init__(self):
        self.A = [None]
        self.heapsize = 0
        self.ID_to_index = {}
        self._ID = self._ID_generator()
    def insert(self, key):
        """Returns an ID that is associated with the item."""
        self.heapsize += 1
        ID = self._ID.next()
        self.ID_to_index[ID] = self.heapsize
        self.A.append([positive_infinity(), ID])
        self.decrease_key(self.heapsize, key)
        return ID
    def _ID_generator(self):
        ID = 0
        while True:
            yield ID
            ID += 1
class heap_id:

def decrease_key_using_id(self, ID, key):
    """Decrease key given ID."""
    self.decrease_key(self.ID_to_index[ID], key)

def extract_min(self):
    """Extracts min and returns key."""
    return self.extract_min_with_id()[0]

def extract_min_with_id(self):
    """Extracts min and returns (key,ID) pair.""
    if self.heapsize < 1:
        print "error: empty heap"
        return
    self._swap(1, self.heapsize)
    self.heapsize -= 1
    min_pair = self.A.pop()
    del self.ID_to_index[min_pair[1]]
    self.min_heapify(1)
    return tuple(min_pair)
And we’re done!

- costan@mit.edu
- IRC: #algos-lite on irc.us.gamesurge.net
- (617) 230-9694, no voicemail
- 32G-8th Floor
v. Next

• Drawings to help Dijkstra intuition, and make it go faster (or not fail completely)

• Consider eliminating intuition if the formal method is taught

• The Dijkstra run was boring, add spice to it (don’t remove, we needed to go through both runs to clarify negative cycle issues)