Dijkstra's Algorithm & Hashing

CS 62 - Spring 2016
Michael Bannister

This Week

No more assignments!

- You should have grades for 1-10 (BooOS report grade/ec soon)
- email from: no.reply.pomonagrading@gmail.com
- Please privately repost any grading concerns to Piazza with subject: "Assignment Grade Question" ASAP

Extra Credit Lab

- Probably recursion coding exercises, as practice for final

Reading: Continue with Bailey Chapter 16 (graphs), start on Chapter 15 (hashing).

Dijkstra's Algorithm

Variables:

```
graph G               //  The graph
vertex_t s            // The starting vertex
double length[n][n]   // Edge length
double dist[n]        // Current best distance
pqueue Q              // PQ (vertex_t, double)
```

Dijkstra'a Algorithm

Set dist[v] to ∞ for all v, except dist[s] = 0
Add s to Q with priority 0.0
loop while Q is not empty
    get vertex cur with min priority d
    if d ≤ dist[curr]:
        for each out going edge cur → v:
            if dist[cur] + length[cur][v] < dist[v]:
                dist[v] = dist[cur] + length[cur][v]
                parent[v] = cur
                Add v to Q with new priority dist[v]

Dijkstra Example

(On Board)

Maps / Dictionaries

- Maintain an association between keys and values
- For every key there is at most one value in the dictionary, i.e., it defines a function
- Generalizes arrays
- Many implementations (including using a BST)

Implementation Performance

Data Structure	**get**	**set**	**remove**
list	$O(n)$	$O(1)$	$O(1)$
sorted list	$O(\log n)$	$O(n)$	$O(n)$
Balanced BST	$O(\log n)$	$O(\log n)$	$O(\log n)$
Array["key range"]	$O(1)$	$O(1)$	$O(1)$

Keys must be comparable!

Keys must be unsigned ints and of small range

Goal

Want: Array like performance for all types of keys

Problems:

- Keys are not unsigned ints and no easy conversion
- Keys range is large or even infinite

Hashing

Pice a function $H : \{\texttt{key\_type}\} \to \{\texttt{size\_t}\}$ uniformly at random from all such functions.

- Such a function should spread keys uniformly over the indexes of an array
- Should be fast to compute
- Probably is not really random, but close..
- There are common tricks for picking good random functions

Hash Collisions

- A **hash collisions** is when
  $$H(k_1) = H(k_2) \text{ and } k_1 \neq k_2$$
  that is, both keys want the same array cell.
- Has collisions will happen! see the b-day paradox
- Hash collisions will be rare with good hash func

External Chaining

- Replace each array cell with a list!
- On collision add new item to the end of the list
- Performance will be good if lists are kept short, i.e., few hash collisions.