# CS38
# Introduction to Algorithms

Lecture 5
April 15, 2014

## Outline

- review of Dijkstra's algorithm
- greedy algorithms: Huffman codes
- data structures for MST and Dijkstra's
  - union-find with $\log^*$ analysis
  - Fibonacci heaps with amortized analysis



## Dijkstra's algorithm

- given
  - directed graph G = (V,E) with non-negative edge weights
  - starting vertex $s \in V$
- find shortest paths from s to all nodes v
  - note: unweighted case solved by BFS
  - shortest paths from s form a tree rooted at s
  - "find" = fill in predecessor pointers for each vertex to define shortest-path tree from s



## Dijkstra's algorithm

- shortest paths exhibit "optimal substructure" property
  - optimal solution contains within it optimal solutions to subproblems
  - a shortest path from x to y via z contains a shortest path from x to z
- Main idea:
  - maintain set $S \subseteq V$ with correct distances
  - add nbr u with smallest "distance estimate"



## Dijkstra's algorithm

```
Dijkstra(G = (V,E), s)
1. S = ∅, s.dist = 0, build Min-Heap H from V, keys are distances
2. while H is not empty
3.     u = EXTRACT-MIN(H)          ← "greedy choice"
4.     S = S ∪ {u}
5.     for each neighbor v of u
6.         if v.dist > u.dist + weight(u,v) then
7.             v.dist = u.dist + weigth(u,v), DECREASE-KEY(H, v)
```

**Lemma**: can be implemented to run in O(m) time plus n EXTRACT-MIN and m DECREASE-KEY calls.

Proof?



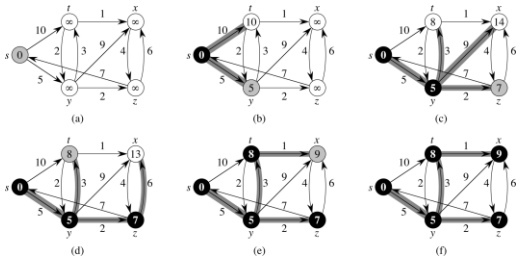
## Dijkstra's algorithm

```
Dijkstra(G = (V,E), s)
1. S = ∅, s.dist = 0, build Min-Heap H from V, keys are distances
2. while H is not empty
3.     u = EXTRACT-MIN(H)          ← "greedy choice"
4.     S = S ∪ {u}
5.     for each neighbor v of u
6.         if v.dist > u.dist + weight(u,v) then
7.             v.dist = u.dist + weigth(u,v), DECREASE-KEY(H, v)
```

**Lemma**: can be implemented to run in O(m) time plus n EXTRACT-MIN and m DECREASE-KEY calls.

Proof: each vertex added to H once, adj. list scanned once, O(1) work apart from min-heap calls

## Dijkstra's example from CLRS





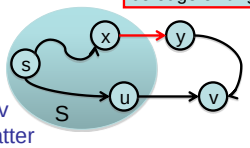
## Dijkstra's algorithm

**Lemma**: invariant of algorithm: for all $v \in S$ it holds that $v.dist = distance(s, v)$.

Proof: induction on size of $S$

- base case: $S = \emptyset$, trivially true
- case $|S| = k$:

consider any other $s - v$ path, let $(x,y)$ be edge exiting $S$




$x.dist$, $u.dist$ correct by induction, so $s - y$ path already longer than $s - v$ since algorithm chose latter



## Dijkstra's algorithm

- We proved:

**Theorem (Dijkstra)**: there is an $O(n^+ m \log n)$ time algorithm that is given
a directed graph with nonnegative weights
a starting vertex s
and finds
distances from s to every other vertex
(and produces a shortest path tree from s)

  - using binary heaps
  - later: Fibonacci heaps: $O(m + n \log n)$ time



## Variable-length encoding

- Scenario: large file with $n$ symbol types and frequencies $f_x$ for each symbol $x$
- Goal: compress the file (losslessly)
  - represent each symbol $x$ by a length $L_x$ string of binary digits ("variable-length")
  - prefix-free: no encoding string is a prefix of another; $\Rightarrow$ can write them one after another
- minimize total length: $\sum_x L_x f_x$



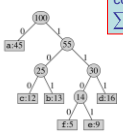
## Huffman codes

Example:

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| Frequency (in thousands) | 45 | 13 | 12 | 16 | 9 | 5 |
| Fixed-length codeword | 000 | 001 | 010 | 011 | 100 | 101 |
| Variable-length codeword | 0 | 101 | 100 | 111 | 1101 | 1100 |

- can represent prefix-free encoding scheme by binary tree T:
- Problem: given frequencies, construct optimal tree (prefix-free encoding scheme)

cost $C(T) = \sum_c L(T)_c f_c$



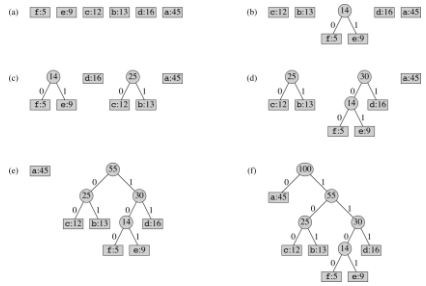



## Huffman codes

- Idea: build tree bottom-up and make greedy choice (what is it?)
  - joining two symbols commits to a bit to distinguish them; which should we choose?

```
1. build heap H with frequencies as keys
2. for i = 1 to n-1
3.       allocate new node z
4.       z.left = x = EXTRACT-MIN(H)
5.       z.right = y = EXTRACT-MIN(H)
6.       z. freq = x.freq + y.freq; INSERT(H, z)
7.  return EXTRACT-MIN(H)
```

## Huffman example from CLRS

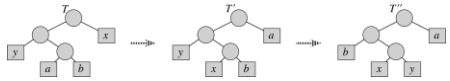




## Huffman codes

**Lemma**: there exists an optimal tree where the 2 lowest frequency symbols are siblings.

**Proof**: let T be an optimal tree; a,b lowest siblings



change from exchanging x, a:

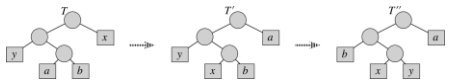
$\sum_c L(T)_c f_c - \sum_c L(T')_c f_c = (f_a - f_x)(L(T)_a - L(T)_x) \geq 0$



## Huffman codes

**Lemma**: there exists an optimal tree where the 2 lowest frequency symbols are siblings.

**Proof**: let T be an optimal tree; a,b lowest siblings



change from exchanging y, b:

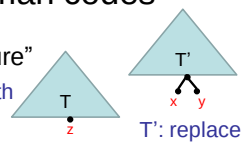
$\sum_c L(T')_c f_c - \sum_c L(T'')_c f_c = (f_b - f_y)(L(T')_b - L(T'')_y) \geq 0$



## Huffman codes

"optimal substructure"

T: optimal tree with merged symbol




T': replace z

**Lemma**: T' is an optimal tree.

**Proof**: consider another tree S', w.l.o.g. having x, y as siblings. Let S be merged version. Optimality of T gives:

$C(S') = C(S) + f_x + f_y \geq C(T) + f_x + f_y = C(T')$



## Greedy algorithms review

- Coin changing: choose largest coin
- Interval scheduling: choose job w/ first finish
- Dijkstra's: extend shortest paths tree by vertex with smallest distance estimate
- Prim's: extend tree by lowest weight edge
- Kruskal's: extend forest by lowest weight edge
- Huffman : build tree by merging lowest freq. pair

Safe choice: can always extend to an optimal solution



## Role of data structures

- Running times: using **heaps**
  - Huffman: O(n) INSERT, O(n) EXTRACT-MIN
  - Dijkstra: O(n) INSERT + EXTRACT-MIN; O(m) DECREASE-KEY
  - Prim: O(n) INSERT + EXTRACT-MIN, O(m) DECREASE-KEY

… and using **union-find**

- Kruskal: time to sort m items and then O(m) FIND + UNION

## Coming up: two data structures

- amortized analysis:
  - each operation has an amortized cost
  - any sequence of operations has cost bounded by sum of amortized costs

1. Fibonacci heap amortized vs. binary heap
   - EXTRACT-MIN O(log n) vs. O(log n)
   - INSERT, DECREASE-KEY O(1) vs. O(log n)



## Coming up: two data structures

- ammortized analysis:
  - each operation has an amortized cost
  - any sequence of operations has cost bounded by sum of amortized costs

2. Union find (amortized)
   - any sequence of n UNIONs and m FINDs: O(n log* m)



## Improvements to running times

- using Fibonacci heaps (vs. binary heaps):
  - Dijkstra: O(n) INSERT + EXTRACT-MIN; O(m) DECREASE-KEY
  - Prim: O(n) INSERT + EXTRACT-MIN, O(m) DECREASE-KEY

  O(m + n log n) vs. O(m log n)
- using union-find
  - Kruskal: time to sort m items and then O(m) FIND + UNION

  O(m log m + m log* n)



## Union-find data structure

Kevin Wayne's slides
based on Kleinberg + Tardos text and
Dasgupta, Papadimitriou,Vazirani text