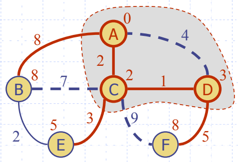
Shortest Path

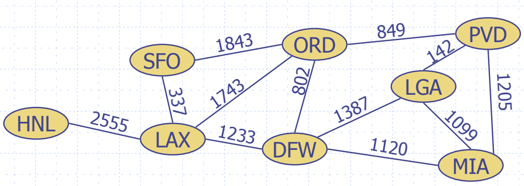


Outline and Reading

- Shortest path (§12.6)
  - Weighted graph
  - Shortest path problem
  - Shortest path properties
- Dijkstra's algorithm (§12.6.1)
  - Algorithm
  - Edge relaxation
  - Example
  - Analysis

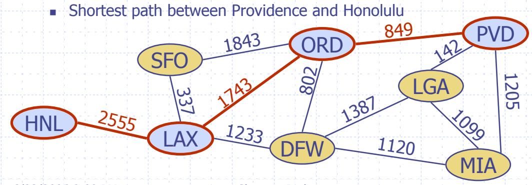
Weighted Graph

- In a weighted graph, each edge has an associated numerical value, called the weight of the edge
- Edge weights may represent, distances, costs, etc.
- Example:
  - In a flight route graph, the weight of an edge represents the distance in miles between the endpoint airports



Shortest Path Problem

- Given a weighted graph and two vertices u and v, we want to find a path of minimum total weight between u and v
- Applications
  - Flight reservations
  - Driving directions
  - Internet packet routing
- Example:
  - Shortest path between Providence and Honolulu



Shortest Path Properties

Property 1:

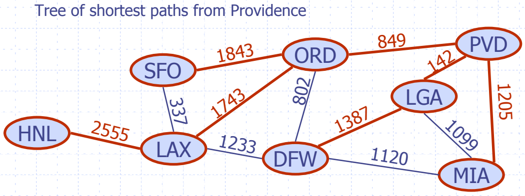
A subpath of a shortest path is itself a shortest path

Property 2:

There is a tree of shortest paths from a start vertex to all the other vertices

Example:

Tree of shortest paths from Providence

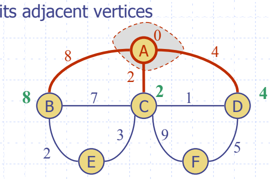


Dijkstra's Algorithm

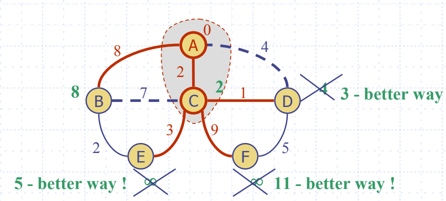
- The distance to a vertex v from a vertex s is the length of a shortest path between s and v
- Dijkstra's algorithm computes the distances to all the vertices from a given start vertex s
- Assumptions:
  - the graph is connected
  - the edges are undirected
  - the edge weights are nonnegative

- We grow a "cloud" of vertices, beginning with s and eventually covering all the vertices
- At each vertex v we store
  d(v) = distance to v from s in the subgraph consisting of the cloud and its adjacent vertices

Example



- At each step
  - We add to the cloud the vertex u outside the cloud with the smallest distance label
  - We update the labels of the vertices adjacent to u

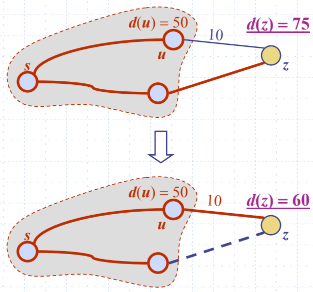


Update = Edge Relaxation

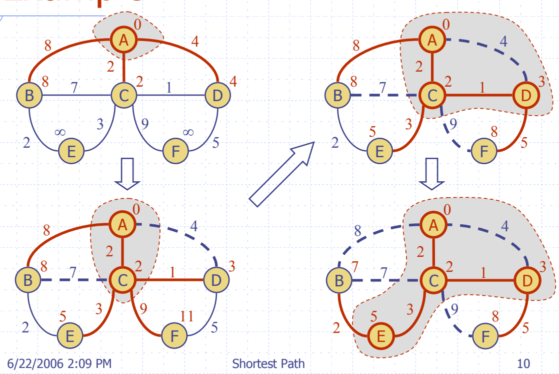
- Consider an edge $e = (u,z)$ such that
  - u is the vertex most recently added to the cloud
  - z is not in the cloud
- The relaxation of edge e updates distance $d(z)$ as follows

$d(z) \leftarrow$

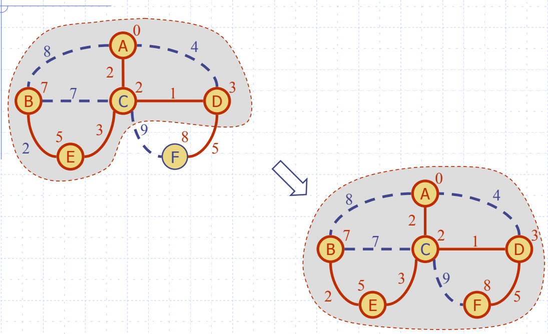
$\min(d(z), d(u) + weight(e))$



Example



Example (cont.)



Dijkstra's Algorithm

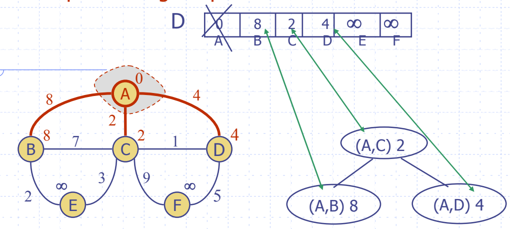
We use a priority queue Q to store
the vertices <u>not</u> in the cloud,
where D[v] is the key of a vertex v in Q

Algorithm ShortestPath(G, v):

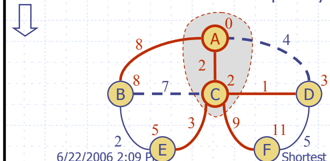
Input: A weighted graph G and a distinguished vertex v of G.
Output: A label D[u], for each vertex u of G, such that D[u] is the length of a shortest path from v to u in G.

initialize D[v] ← 0 and D[u] ← ∞ for each vertex v ≠ u
let Q be a priority queue that contains all of the vertices of G using the D labels as keys.
while Q ≠ ∅ do {pull u into the cloud C}
    u ← Q.removeMinElement()
    for each vertex z adjacent to u such that z is in Q do
        {perform the relaxation operation on edge (u, z) }
        if D[u] + w((u, z)) < D[z] then
            D[z] ←D[u] + w((u, z))
            change the key value of z in Q to D[z]
return the label D[u] of each vertex u.

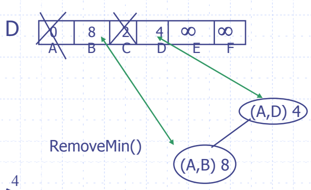
Same Example – Using heap

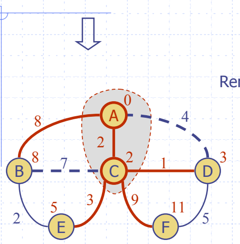


In the book: location-aware priority queue



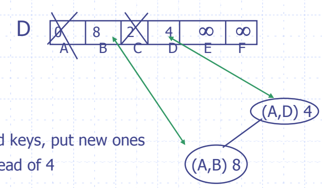
RemoveMin() and Update





Relaxation:
Update:

(C,D) 3	YES (3 < 4)
(C,E) 5	YES (5 < ∞)
(C,F) 11	YES (11 < ∞)
(C,B)	NO (9>8)



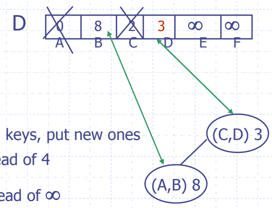
Update means: remove old keys, put new ones

(C,D) **3** Instead of 4

(C,E) 5 Instead of ∞

(C,F) 11 Instead of ∞

Replace (A,D) 4 with (C,D) 3
Insert (C,E) 5
Insert (C,F) 11



Update means: remove old keys, put new ones

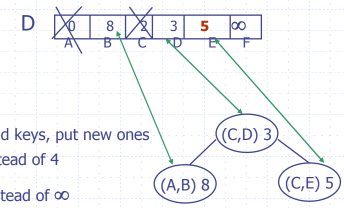
(C,D) **3** Instead of 4

(C,E) 5 Instead of ∞

(C,F) 11 Instead of ∞

Replace (A,D) 4 with (C,D) 3
Insert (C,E) 5
Insert (C,F) 11

When replacing you might need to rearrange the heap (not in this example).



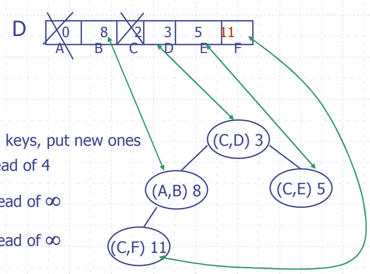
Update means: remove old keys, put new ones

(C,D) **3** Instead of 4

(C,E) 5 Instead of ∞

(C,F) 11 Instead of ∞

Replace (A,D) 4 with (C,D) 3
Insert (C,E) 5
Insert (C,F) 11



Update means: remove old keys, put new ones

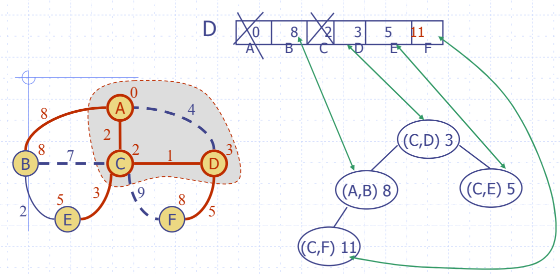
(C,D) **3** Instead of 4

(C,E) 5 Instead of ∞

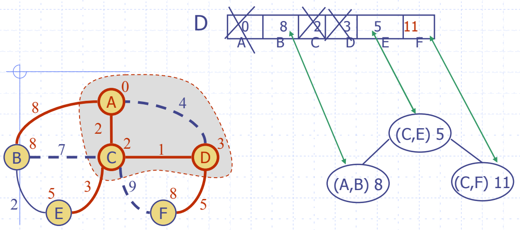
(C,F) 11 Instead of ∞

Replace (A,D) 4 with (C,D) 3

Insert (C,E) 5
Insert (C,F) 11

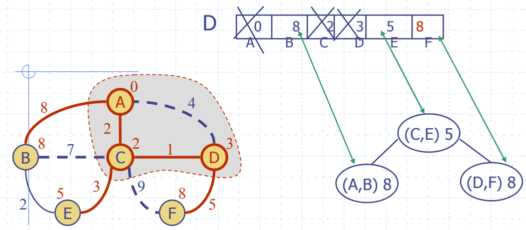


RemoveMin()
Update



RemoveMin()
Update (D,F) 8 ? Yes 8 < 11

Replace (C,F) 11
with (D,F) 8



RemoveMin()
Update (D,F) 8 ? Yes 8 < 11

Replace (C,F) 11
with (D,F) 8

Running Time

If we represent G with an adjacency list. We can then step through all the vertices adjacent to u in time proportional to deg(u)

- The priority queue Q
  - A Heap:

while Q ≠ ∅ do {pull u into the cloud C}

at each iteration:

- extraction of vertices with the smallest D-label: O(log n).
- key updates: O(log n) for each update (replace and insert keys).

After each extraction: O(deg(u) log n)

in total: $\sum_{u \in G}$ (1 + deg(u)) log n = O((n+m) log n) = **O(m log n)**

worst case: O(n^2 log n)

- An Unsorted Sequence:

  O(n) when we extract minimum elements, but fast key updates (O(1)).

There are only n-1 extractions and m updates.

The running time is O(n^2+m) = **O(n^2)**

Heap	Sequence
O(m log n)	**O(n^2)**