## (Single Source) Shortest Paths

Given: weighted graph, *G*, and source vertex, *v*

Compute: shortest path to every other vertex in *G*

- Path length is sum of edge weights along path
- Shortest path has smallest length among all possible paths



## Dijkstra's Algorithm

Grow a collection of vertices for which shortest path is known

- paths contain only vertices in the set
- add as new vertex the one with the smallest distance to the source
- shortest path to an outside vertex must contain a current shortest path as a prefix

Use a greedy algorithm



## Edge Relaxation

Maintain value D[*u*] for each vertex

- Each starts at infinity, and decreases as we find out about a shorter path from *v* to *u* (D[*v*] = 0)

Maintain priority queue, *Q*, of vertices to be relaxed

- use D[*u*] as key for each vertex
- remove min vertex from *Q*, and relax its neighbors

Relaxation for each neighbor of *u*:

if D[*u*] + w(*u*, *z*) < D[*z*] then
    D[*z*] = D[*u*] + w(*u*, *z*)



## Dijkstra Pseudocode

```
ShortestPath(G, v)
  init D array entries to infinity
  D[v]=0
  add all vertices to priority queue Q
  while Q not empty do
    u = Q.removeMin()
    for each neighbor, z, of u in Q do
      if D[u] + w(u,z) < D[z] then
         D[z] = D[u] + w(u,z)
         Change key of z in Q to D[z]
  return D as shortest path lengths
```



## Dijkstra Analysis

$O(n\log n)$ time to build priority queue

$O(n\log n)$ time removing vertices from queue

$O(m\log n)$ time relaxing edges

- Changing key can be done in $O(\log n)$ time

Total time: $O((n + m)\log n)$

- which can be $O(n^2\log n)$ for dense graph



## Minimum Spanning Trees

Given: connected, undirected, weighted graph

Compute: spanning tree with minimum sum of edge weights

- Spanning tree contains all *n* vertices and subset of edges (*n*-1)
- minimize $w(T) = \Sigma_{(v,u) \text{ in } T}\, w((v,u))$
  - —if edge weights are not unique, there may be multiple MSTs for a graph

## Applications of MST

Think of edge weight as a *cost* of some sort

- MST minimizes total cost associated with connecting the vertices by edges

Some applications:

- Telephone, electrical, plumbing
- Computer networks



## Minimum Bridge Principle

Consider all vertices partitioned into two sets, $V_1$ and $V_2$

- Spanning tree must have at least one edge to "bridge the gap" between the partitions

Consider all bridge edges (with one vertex in $V_1$ and one in $V_2$)

- The minimum weight bridge, *e*, is part of some minimum spanning tree
  - —For an MST without *e*, insert *e*, and remove another bridge from the cycle
  - —This creates an MST with the same or smaller weight (if smaller, the original was not an MST



## Greedy Algorithms for MST

Kruskal's Algorithm

- Start with many small clusters
- Add minimum bridges, merging clusters as we go

Prim-Jarnik Algorithm

- Start with a root (arbitrary)
  - —partition into "root cluster" and "other cluster"
- Find minimum bridge, and transfer node from other cluster to root cluster
  - —proceeds much like Dijkstra's shortest paths algorithm



## Kruskal's Algorithm

```
Kruskal(G)
  for each vertex in G do
    define cluster C(v)={v}
  insert edges into priority queue, Q
  Initialize empty tree graph, T
  while T.numEdges() < n - 1 do
    (u,v) = Q.removeMin()
    if C(u) != C(v) then
        add edge (u,v) to T
        Merge C(u) and C(v)
  return T
```



## Analyzing Kruskal's Algorithm

Building Q (and extracting edges) takes $O(m\log m) == O(m\log n^2) == O(m\log n)$

Each merge takes linear time

- Each vertex is involved in only log*n* merges
  - —always merge the smaller cluster into the larger one
  - —cluster of a vertex always at least doubles in size

Total time is $O(\,(n + m)\log n\,)$



## Prim-Jarnik Algorithm

```
PrimJarnik(G)
  pick vertex v as root
  Initialize tree graph, T, to contain v
  Initialize priority queue, Q, to contain
   incident edges of v
  while T.numEdges() < n-1 do
   e = Q.removeMin()
   if only one vertex of e is in T do
    v = vertex of e not in T
    T.insert(v), T.insert(e)
    Q.insert(all incident edges of v)
  return T
```

## Prim-Jarnik Analysis

**Inserting and removing into edge queue takes $O(m \log m) = O(m \log n)$**