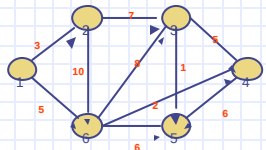
Graphs - Shortest Paths

- Application
  - In a graph in which edges have costs ..
  - Find the shortest path from a source to a destination
  - *Surprisingly ..*
    - While finding the shortest path from a source to one destination,
    - *we can find the shortest paths to all over destinations as well!*
  - Common algorithm for
    single-source shortest paths
    is due to Edsger Dijkstra

Dijkstra's Algorithm - Data Structures

- For a graph,
  $G = (V, E)$
- Dijkstra's algorithm keeps *two* sets of vertices:
  - S — Vertices whose shortest paths have already been determined
  - V-S — Remainder
- Also
  - d — Best estimates of shortest path to each vertex
  - π — Predecessors for each vertex

The Shortest Path: from vertex 1 to vertex 5



Adjacency matrix

-	1	2	3	4	5	6
1	0	3	∞	∞	∞	5
2	∞	0	7	∞	∞	10
3	∞	∞	0	3	1	∞
4	∞	∞	∞	0	6	∞
5	∞	∞	∞	∞	0	7
6	∞	∞	∞	6	2	0

Predecessor Sub-graph

- Array of vertex indices, π[j], j = 1 .. |V|
  - π[j] contains the predecessor for node j
  - π[j]'s predecessor is in π[π[j]], and so on ....
  - The edges in the predecessor sub-graph are
    (π[j], j)

Dijkstra's Algorithm - Operation

- Initialise d and π
  - For each vertex, j, in V
    - $d_j = \infty$
    - π_j = nil
  - Source distance, $d_s = 0$
- Set S to empty
- While V-S is not empty
  - Sort V-S based on d
  - Add u, the closest vertex in V-S, to S
  - Relax all the vertices still in V-S connected to u

No connections

Add s first!

Operation

- The Relaxation process

Relax the node v attached to node u

Edge cost matrix

```
relax( Node u, Node v, double w[][] )
    if (d[v] > d[u] + w[u][v]){
        d[v] = d[u] + w[u][v];
        pi[v] = u;
}
```

If the current best estimate to v is greater than the path through u ..

Update the estimate to v

Make v's predecessor point to u

Dijkstra's Algorithm - Full

- The Shortest Paths algorithm

Given a graph, g, and a source, s

```
shortest_paths( Graph g, Node s ){
    initialise_single_source( g, s );
    S = { 0 };          /* Make S empty */
    Q = Vertices(g); /* Put the vertices in a PQ */
    while (! Empty(Q)){
        u = removeMin( Q );
        AddNode( S, u ); /* Add u to S */
        for each vertex v in Adjacent( u )
            relax( u, v, w )
    }
}
```


Dijkstra's Algorithm - Initialise

- The Shortest Paths algorithm

Given a graph, g, and a source, s

Initialize d, p, S, vertex Q

```
shortest_paths( Graph g, Node s ){
    initialise_single_source( g, s );
    S = { 0 };          /* Make S empty */
    Q = Vertices(g) /* Put the vertices in a PQ */
    while (! Empty(Q)){
        u = removeMin( Q );
        AddNode( S, u ); /* Add u to S */
        for each vertex v in Adjacent( u )
            relax( u, v, w );
    }
}
```


Dijkstra's Algorithm - Loop

- The Shortest Paths algorithm

Given a graph, g, and a source, s

While there are still nodes in Q

Greedy!

```
shortest_paths( Graph g, Node s ){
    initialise_single_source( g, s );
    S = { 0 };          /* Make S empty */
    Q = Vertices( g ); /* Put the vertices in a PQ */
    while (! Empty(Q)){
        u = removeMin( Q );
        AddNode( S, u ); /* Add u to S */
        for each vertex v in Adjacent( u )
            relax( u, v, w );
    }
}
```


neighbours

- The Shortest Paths algorithm

Given a graph, g, and a source, s

Update the estimate of the shortest paths to all nodes attached to u

Greedy!

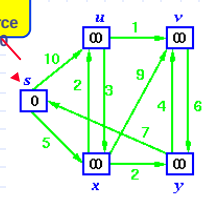
```
shortest_paths( Graph g, Node s ){
    initialise_single_source( g, s )
    S = { 0 }          /* Make S empty */
    Q = Vertices( g ) /* Put the vertices in a PQ */
    while (! Empty(Q)){
        u = removeMin( Q );
        AddNode( S, u ); /* Add u to S */
        for each vertex v in Adjacent( u )
            relax( u, v, w );
    }
}
```


Dijkstra's Algorithm - Operation

- Initial Graph

Source
Mark 0

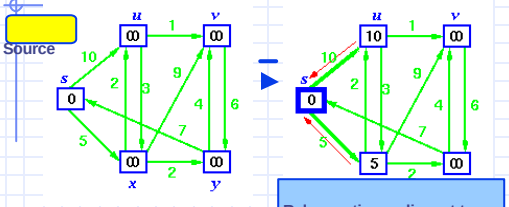
Distance to all nodes marked ∞



Dijkstra's Algorithm - Operation

- Initial Graph

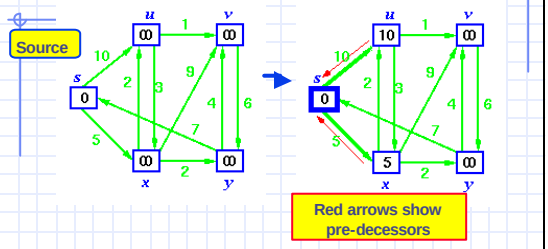
Source



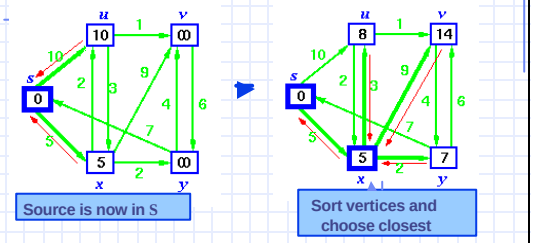
Relax vertices adjacent to source

Dijkstra's Algorithm - Operation

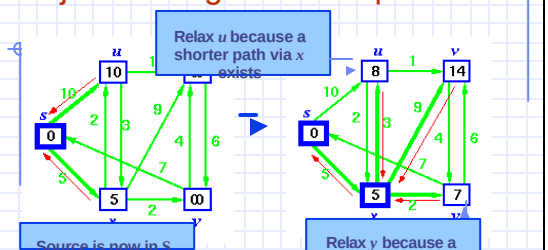
- Initial Graph



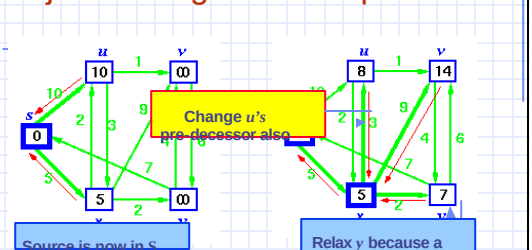
Dijkstra's Algorithm - Operation



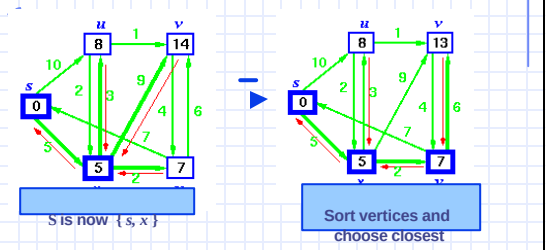
Dijkstra's Algorithm - Operation



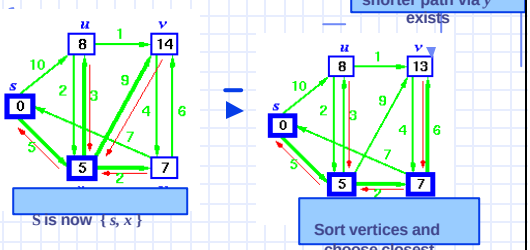
Dijkstra's Algorithm - Operation



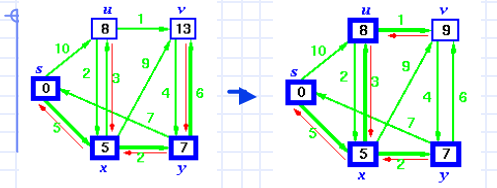
Dijkstra's Algorithm - Operation



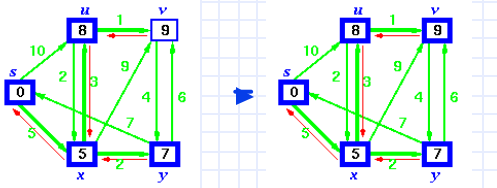
Dijkstra's Algorithm - C



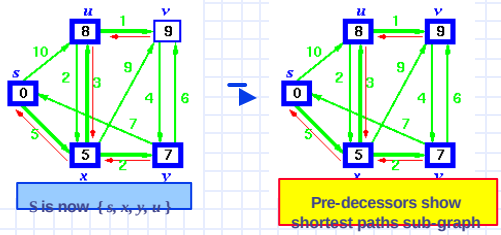
Dijkstra's Algorithm - Operation



Dijkstra's Algorithm - Operation



Dijkstra's Algorithm - Operation



Dijkstra's Algorithm - Time Complexity

- Dijkstra's Algorithm
  - Similar to MST algorithms
  - Key step is sort on the edges
  - Complexity is
    - $O((|E|+|V|)\log|V|)$ ***or***
    - $O(n^2 \log n)$

    for a dense graph with $n = |V|$ **and** $|E| \gg |V|^2$