Single source shortest paths

- Given a directed graph G=<V, E> and a source node, *s*, find the shortest path to each node from the source, *s*
- Dijkstra's algorithm

A greedy algorithm (Dijkstra's)

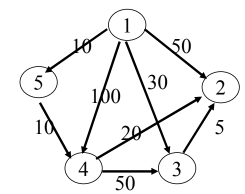
- S – partial solution set, a set of nodes whose shortest paths have been found
  - We use $\delta(s, v)$ to denote the length of shortest path from *s* to *v*
- *Special* path for node except the source node *s*
  - A path from the source node *s* where all nodes except the endpoint must belong to S
- Greedy algorithm
  - At each step, add the node with the shortest *special* path to S

Dijkstra's algorithm

S : partial solution set
d[v]: length of the shortest special path for v.
π[v]: the previous node of v along its shortest (special) path.

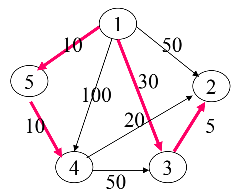
```
Dijkstra(G, w, s)
{
   /* initialization */
   for each node v {
      d[v] = ∞;
      π[v] = null;
   }
   d[s] = 0;

   S = ∅;
   Q.build(V); // a priority Q use d[] as keys
   while (!Q.empty())
      u = Q.extractMin();
      S = S ∪ {u};
      for each v adjacent to u {
            if (v ∈Q && d[u] + w(u,v) < d(v)) {
               d[v] = d[u] + w(u,v);
               Q.decreaseKey(v, d(v));
               π[v] = u;
            }
      }
   }
}
```

Example



Step	u	S	d 1	d 2	d 3	d 4	d 5	π 1	π 2	π 3	π 4	π 5
Init	-	∅	**0**	∞	∞	∞	∞	-	-	-	-	-
1	1	{1}	0	50	30	100	**10**	-	1	1	1	1
2	5	{1, 5}	0	50	30	**20**	10	-	1	1	5	1
3	4	{1,5,4}	0	40	**30**	20	10	-	4	1	5	1
4	3	{1,5,4,3}	0	35	30	20	10	-	3	1	5	1
5	2	{1,5,4,3,2}	0	35	30	20	10	-	3	1	5	1

Example



Step	u	S	d					π				
Init	-	∅	**0**	∞	∞	∞	∞	-	-	-	-	-
1	1	{1}	0	50	30	100	**10**	-	1	1	1	1
2	5	{1, 5}	0	50	30	**20**	10	-	1	1	5	1
3	4	{1,5,4}	0	40	**30**	20	10	-	4	1	5	1
4	3	{1,5,4,3}	0	35	30	20	10	-	3	1	5	1
5	2	{1,5,4,3,2}	0	35	30	20	10	-	3	1	5	1

Proof using loop invariant

- We prove the following loop invariant
  - At the start of each while loop iteration, for any node v in S, d[v] = δ(s, v)

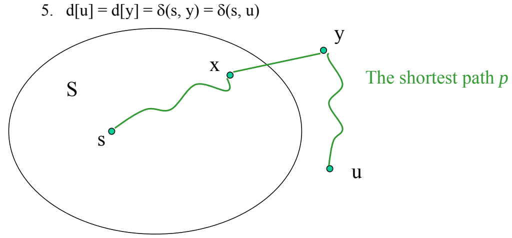
Proof.
Initilaiztion: Initially S = ∅, trivially true.

Maintenance: Assume that at the start of a while loop iteration, for any node v in S, d[v] = δ(s, v). We like to show that d[u] = δ(s, u) when u is added. (next slide)

Termination: All nodes are added to S, so all the shortest paths are found

Proof of the maintenance step

- Assume by contradiction that d[u] ≠ δ(s, u). Let *p* be the shortest path from s to u
  - Because s ∈ S and u ∈V-S, let y be the first node along p such that y ∈V-S, and x is y's predecessor along p
  - We show that
    1. d[x] = δ(s, x) by loop invariant assumption
    2. d[y] = δ(s, y) by the convergence property
    3. d[y] <= d[u] by the path structure
    4. d[u] <= d[y] by the algorithm
    5. d[u] = d[y] = δ(s, y) = δ(s, u)



Analysis of the heap implementation

```
Dijkstra(G, w, s)
{
  /* initialization */
  for each node v {
     d[v] = ∞;
     π[v] = null;
  }
  d[s] = 0;

  S = ∅;
  Q.build(V); // a priority Q use d[] as keys
  while (!Q.empty())
     u = Q.extractMin();
     S = S ∪ {u};
     for each v adjacent to u {
          if (v ∈Q && d[u] + w(u,v) < d(v)) {
             d[v] = d[u] + w(u,v);
             Q.decreaseKey(v, d(v));
             π[v] = u;
          }
     }
  }
}
```

Θ(V)

execute V times

O(VlogV)

execute E times
overall: O(ElogV)

Total: O(ElogV)

An implementation using adjacency matrix

```
Dijkstra(Weight L[][]) // n = |V|
{
   /* initialization */
   Q = {i | 1 <= i <= n};   // S = {1}; 1 is the source
   for (i=1; i<=n; i++) {            ← Θ(V)
      d[i] = ∞;
      π[i] = 1;
   }
   d[1] = 0;
   for (i=1; i<=n-1; i++) { // repeat n-1 times
      u = some element of Q minimizing d[u];       ← O(V)
      Q = Q – {u};  // S = S ∪ {u}
      for (each v ) {                               ← Θ(V) If go through L[u, 1..n]
         if ( d[u]+L[u,v]  < d[v]) {
            d[v] = d[u]+L[u,v];
            π[v] = u;
         }
      }
   }
}
```

Total: $\Theta(V^2)$