Chapter 4. Section 1

Dijkstra's Minimum-Weight Path Algorithm

Recall that the **length** of a walk or path in a graph G is the number of edges it contains. If each edge e has a weight $w(e)$ then the **weight** of a path P, is the sum of the edge weights of P, i.e. $\sum_{e\in E} w(e)$.

For any two vertices x and y of G we define the distance from x to y, denoted by $\mathrm{d}(x,y)$ to be the minimum weight of any path joining x and y. If we need to specify which graph we mean then we write $\mathrm{d}_G(x,y)$.

Dijkstra's Algorithm constructs a tree, starting at a **root vertex** v in which the (unique) path from v to any vertex x is precisely $\mathrm{d}_G(v,x)$. It operates in almost exactly the same way as Prim's algorithm:

Prim's algorithm labels a vertex x with *the quickest way to get to x by leaving the current version of the tree.*

Choose a root vertex v. Dijkstra's algorithm labels x with: *the quickest way to get to x* **from the root v** *by leaving the current version of the tree.*

Specifically when the algorithm has constructed a subtree T of G, the vertex label $l(x)$ of any vertex x will be given by:

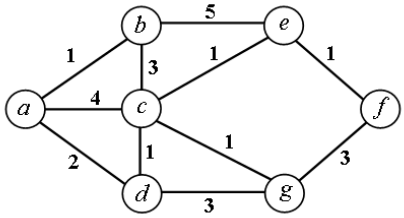
$$l(x) = \begin{cases} \mathrm{d}_G(v,x) & \text{if } x \text{ is in the current tree} \\ (p(x),q(x)) & \text{where } p(x) = \text{the vertex of } T \text{ adjacent to } x \text{ whose distance from } v \text{ is least;} \\ & \quad q(x) = \mathrm{d}(v,p(x)) + w(p(x)x) \quad \text{(so, the least distance to } x \text{ from } v \text{ via } p(x)\text{).} \end{cases}$$

There is another important difference: the input must be in $\mathbb{R}^{\geq 0}$, the nonnegative reals. Things go wrong otherwise as well shall see in the next lecture.

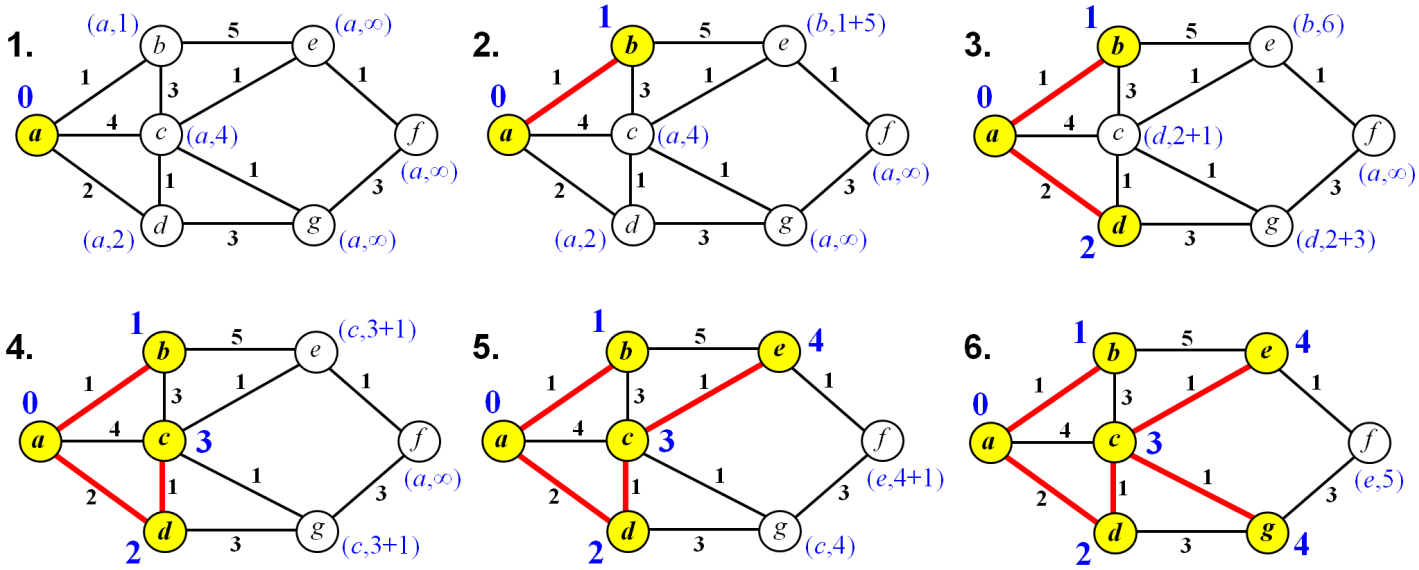
Algorithm **Dijkstra**
Input: connected graph $G = (V(G), E(G))$ with edge weights in $\mathbb{R}^{\geq 0}$, root vertex v
Output: a spanning tree T of G such that, for all $x \in V$, $d_T(v,x) = d_G(v,x)$

```
    # Initialise tree:
1.  Choose v ∈ V(G), T := ({v}, ∅)
    # Initialise vertex label pairs:
2.  for x ∈ V(G)\{v} do label x with pair l(x) = (p(x), q(x)), where p(x) := v and q(x) := ∞
3.  u := v; l(u):=0      # u records the vertex most recently added to T
4.  while |V(T)| < |V(G)| do
        # Update vertex labels:
5.      for each edge ux leaving T from vertex u do
6.          if d(v, u) + w(ux) < q(x) then p(x) := u; q(x) := d(v, u) + w(ux)
        # Extend the tree:
7.      u := vertex x ∈ V(G)\V(T) with minimum value of q(x)      # u becomes the new T vertex
8.      V(T) := V(T) ∪ {u}; E(T) := E(T) ∪ {p(u)u}
9.      l(u) := q(u)
    return(T)
end
```

Example: Consider the following graph:



With vertex a chosen as the root vertex (v in the algorithm) the action of Dijkstra's algorithm is illustrated in a series of diagrams:



The final step will be the output of the algorithm:

The steps are set on in tabular form below. Notice the close similarity to Prim's algorithm. The big difference is highlighted by writing new vertex labels as, for example, $(b,\ 1+5)$ to emphasise the fact that not only the cost of the edge from b is used, but also the distance of b from the root vertex v.

	a	b	c	d	e	f	g	u	tree edges
Initialise:		$(a,\ \infty)$	$(a,\ \infty)$	$(a,\ \infty)$	$(a,\ \infty)$	$(a,\ \infty)$	$(a,\ \infty)$	a	$\emptyset$
Iteration									
1 Update labels:		$(a,\ 1)$	$(a,\ 4)$	$(a,\ 2)$	$(a,\ \infty)$	$(a,\ \infty)$	$(a,\ \infty)$		
Extend tree:		**1**						b	$\{ab\}$
2 Update labels:			$(a,\ 4)$	$(a,\ 2)$	$(b,\ 1+5)$	$(a,\ \infty)$	$(a,\ \infty)$		
Extend tree:				**2**				d	$\{ab,\ ad\}$
3 Update labels:			$(d,\ 2+1)$		$(b,\ 6)$	$(a,\ \infty)$	$(d,\ 2+3)$		
Extend tree:			**3**					c	$\{ab,\ ad,\ dc\}$
4 Update labels:					$(c,\ 3+1)$	$(a,\ \infty)$	$(c,\ 3+1)$		
Extend tree:					**4**			e	$\{ab,\ ad,\ dc,\ ce\}$
5 Update labels:						$(e,\ 4+1)$	$(c,\ 4)$		
Extend tree:							**4**	g	$\{ab,\ ad,\ dc,\ ce,\ cg\}$
6 Update labels:						$(e,\ 5)$			
Extend tree:						**5**		f	$\{ab,\ ad,\ dc,\ ce,\ cg,\ ef\}$