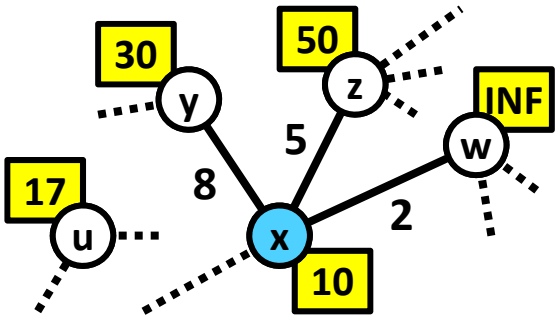
Notes on priority queue management

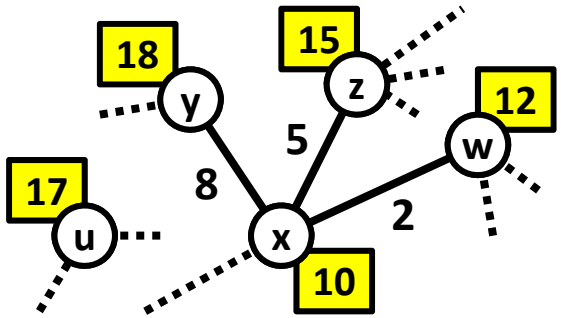
A Priority queue trouble

Usual theoretical and conceptual descriptions of Prim's (Dijkstra's, etc.) algorithm say:
"... store nodes in a priority queue".
Technically, this is nearly impossible to do.
The graph and the nodes are defined separately and stored elsewhere in the memory.
The node has no reference to its position in the queue.
The programmer does not know where is the node in the queue.
So, how to move a node inside the queue according to the algorithm demands?
Standard solution:
Do not move a node, enqueue a "copy of a node", possibly more times.

When a copy of the node with the smallest value (=highest priority) among all its copies appears at the top of the queue
it does its job exactly according to the algorithm prescription.
From that moment on, all other copies of the node which are still in the queue become useless and must be ignored. The easiest way to ignore a copy is to check it when it later appears at the top of the queue: If the node is already closed, ignore the copy, pop it and process the next top of the queue. If the node is still open, process it according to the algorithm.



Easy



top priority queue

x	..	...	u	...	y	...	z	...	w	...
10	..	...	17	...	30	...	50	...	INF	...

Difficult

top priority queue

...	..	w	...	z	...	u	...	y	...
...	..	12	...	15	...	17	...	18	...

q.insert(y);
q.insert(z);
q.insert(w);
// push the nodes
// once more to the queue.

Easy

top priority queue

..	w	...	z	...	u	...	y	...	y	...	z	...	w	...
..	12	...	15	...	17	...	18	...	30	...	50	...	INF	...

new copies old copies

The older copies of nodes will get to the top of the queue later than the new copies (which have higher priority) . The older copy gets to the top when the node had been processed and closed earlier. Thus:
If the node at the top of the queue is closed just pop it and do not process it any more.