Prim’s Algorithm.

We have a set $B$ of nodes that are not in the component of $s$, so the component of $s$ is $V - B$.

For each $u \in B$ we have the least cost edge $\{u, v\}$ that connects it to $V - B$; we have $Cost[u] = c(u, v)$ and $Neighbor[u] = v$. If there is no such edge, we set $Cost[u] = \infty$.

// Initialization sets priorities
for (every node $u$)
    Cost[$u$] = $\infty$;
Cost[$s$] = 0;
$B = V$; // organize $B$ as a priority queue
// Iteration
$v = \text{DeleteMin}(B)$; // remove $v \in B$ with the minimum $Cost[v]$ add $\{v, Neighbor[v]\}$ to the tree
for (every edge $\{v, w\}$) {
    if (w not in $B$)
        continue;
    if ($Cost[w] > (\text{new} = c(v, w))$)
        $Cost[w] = \text{new}$; // decrease of priority
        $Neighbor[w] = v$;
}

If $|V| = n$, $|E| = m$, we perform $n - 1$ DeleteMin operations and up to $m$ operations of decrease a priority.

Changes needed to compute shortest paths from node $s$ instead: before, when $u$ was added to the tree, $\{u, v\}$ was an edge and $v$ was not in a tree, we had to consider new possibility: $\{u, v\}$ can be a better edge to join $v$ with the tree, because the cost $c(u, v)$ can be lower.

Now we consider a directed edge $(u, v)$, and we have to check if we see a better path from $s$ to $v$: first to $u$ and then with edge $(u, v)$, because $Cost[u] + c(u, v)$ can be lower. Thus we need two changes:

i. rather than computing the spanning tree, we compute predecessors on paths from $s$ to $u$; $Pred[u]$;
ii. when we check edge $(u, v)$, we compare $Cost[v]$ with $Cost[u] + c(u, v)$. 
Dijkstra’s Algorithm.

We have a set $B$ of nodes for which we did not yet compute shortest paths from node $s$.

For each $u \in B$ we have predecessor on the best path we considered so far, $Pred[u]$. This means that on the best path computed so far $(Pred[u], u)$ is the last edge. We also know $Cost[u]$, the cost of the best path computed so far.

If we do not know any path to node $u$ yet, $Cost[u] = \infty$ and $Pred[u]$ makes no sense (and $Pred[s]$ makes no sense either).

    // Initialization sets priorities
    for (every node u)
        Cost[u] = \infty;
    Cost[s] = 0;
    B = V; // organize B as a priority queue

    // Iteration
    v = DeleteMin(B); // remove v\in B with the minimum Cost[v]
    add {v, Neighbor[v]} to the tree
    for (every edge {v, w}) {
        if (w not in B)
            continue;
        if (Cost[w] > (new = Cost[v] + c(v, w)))
            Cost[w] = new; // decrease of priority
            Pred[w] = v;
    }