COS 423 Lecture 10
Minimum Spanning Trees

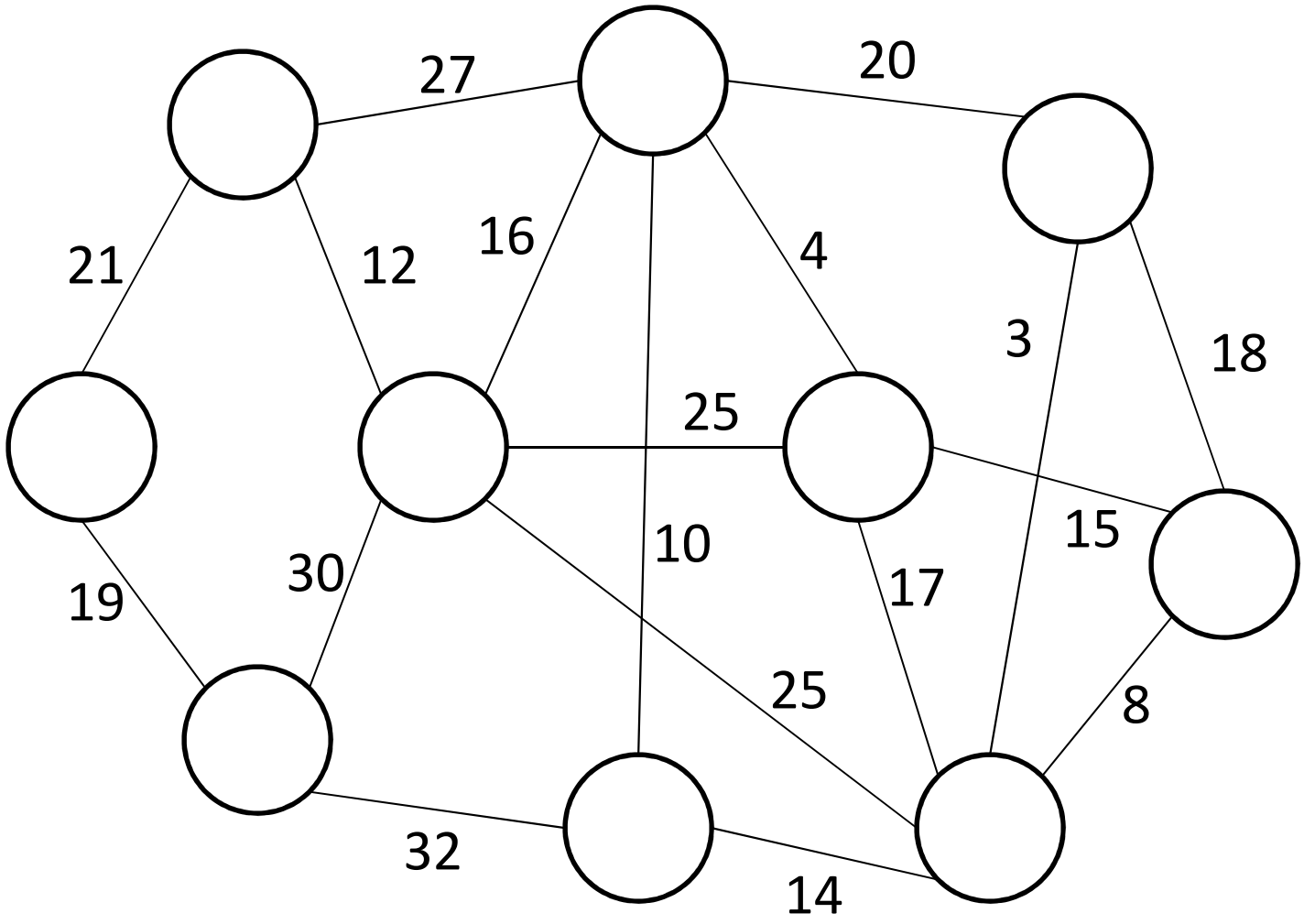
An undirected graph is *connected* if there is a path from any vertex to any other

A (*free* or *unrooted*) *tree* is an acyclic connected graph

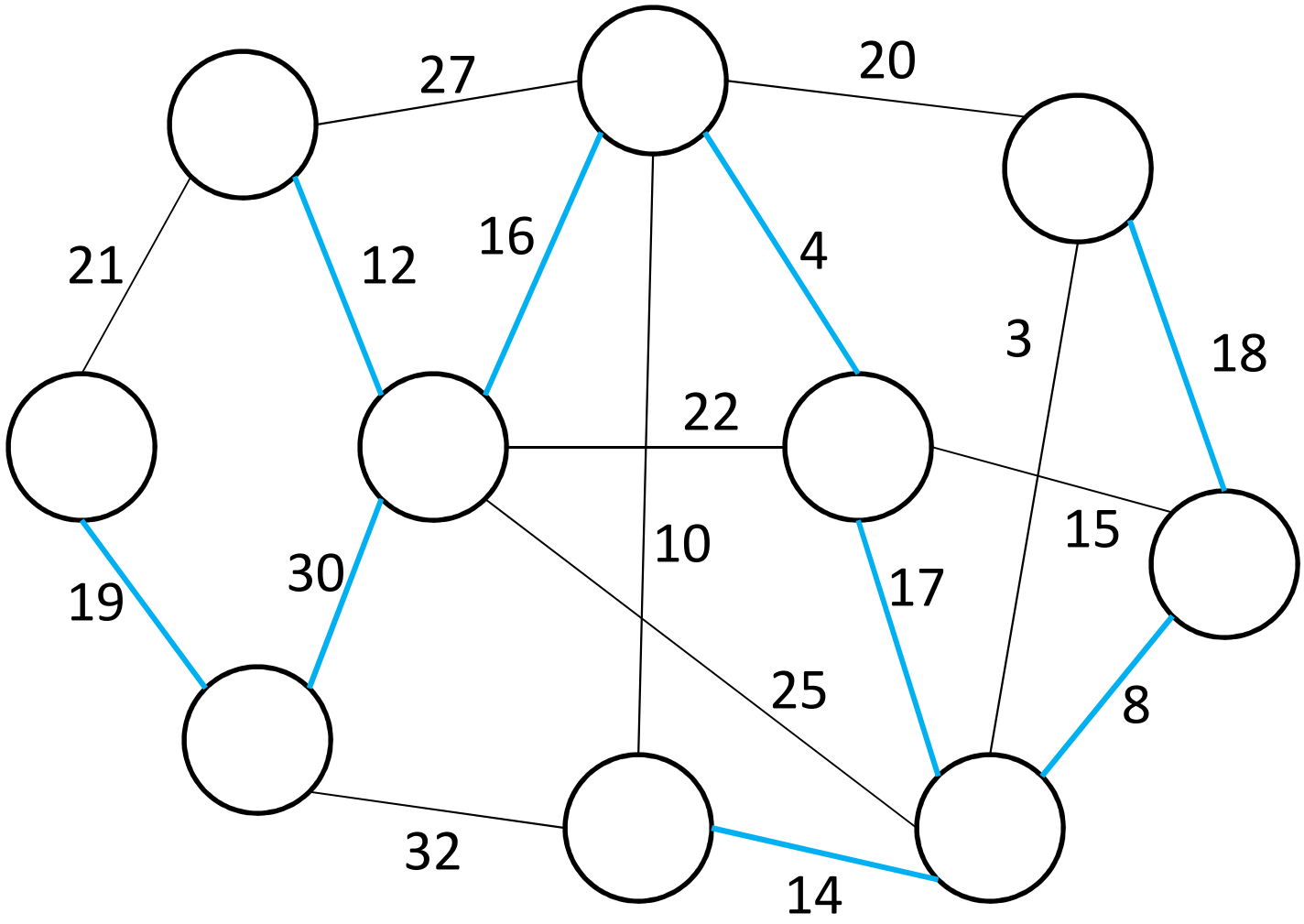
Given a connected graph, a *spanning tree* is a subgraph that is a tree and that contains all the vertices

Problem: Given a connected graph with edge weights, find a spanning tree of minimum total edge cost: a *minimum spanning tree* (*MST*)

A connected undirected graph with edge weights



Minimum spanning tree?



Applications

Clustering (single-linkage): 1909, skull classification in anthropology

Network design: 1920's, Moravian electrical network

Traveling salesperson problem lower bounds: 1971, Held & Karp

When can we conclude that an edge is in the MST?

When can we conclude that an edge is not in the MST?

(tie-breaking by edge numbering)

Build an MST by edge coloring

blue = accepted (added to MST)

red = rejected (no longer a candidate)

cut: a partition of the vertex set into two non-empty parts *X*, *Y*. An edge with one end in *X* and one in *Y crosses* the cut.

Blue rule: Given a cut, color blue the minimum-weight edge crossing it (*good edge*)

Red rule: Given a cycle, color red its maximum-weight edge (*bad edge*)

Generalized greedy method

Begin with all edges uncolored. Repeatedly apply the blue and red rules, in any order, until all edges are colored.

At all times, the blue edges form a set of trees, called the *blue trees*. Initially, no blue edges: each vertex forms a one-vertex blue tree.

Once there is only one blue tree, containing all the vertices, it is an MST.

Classical algorithms

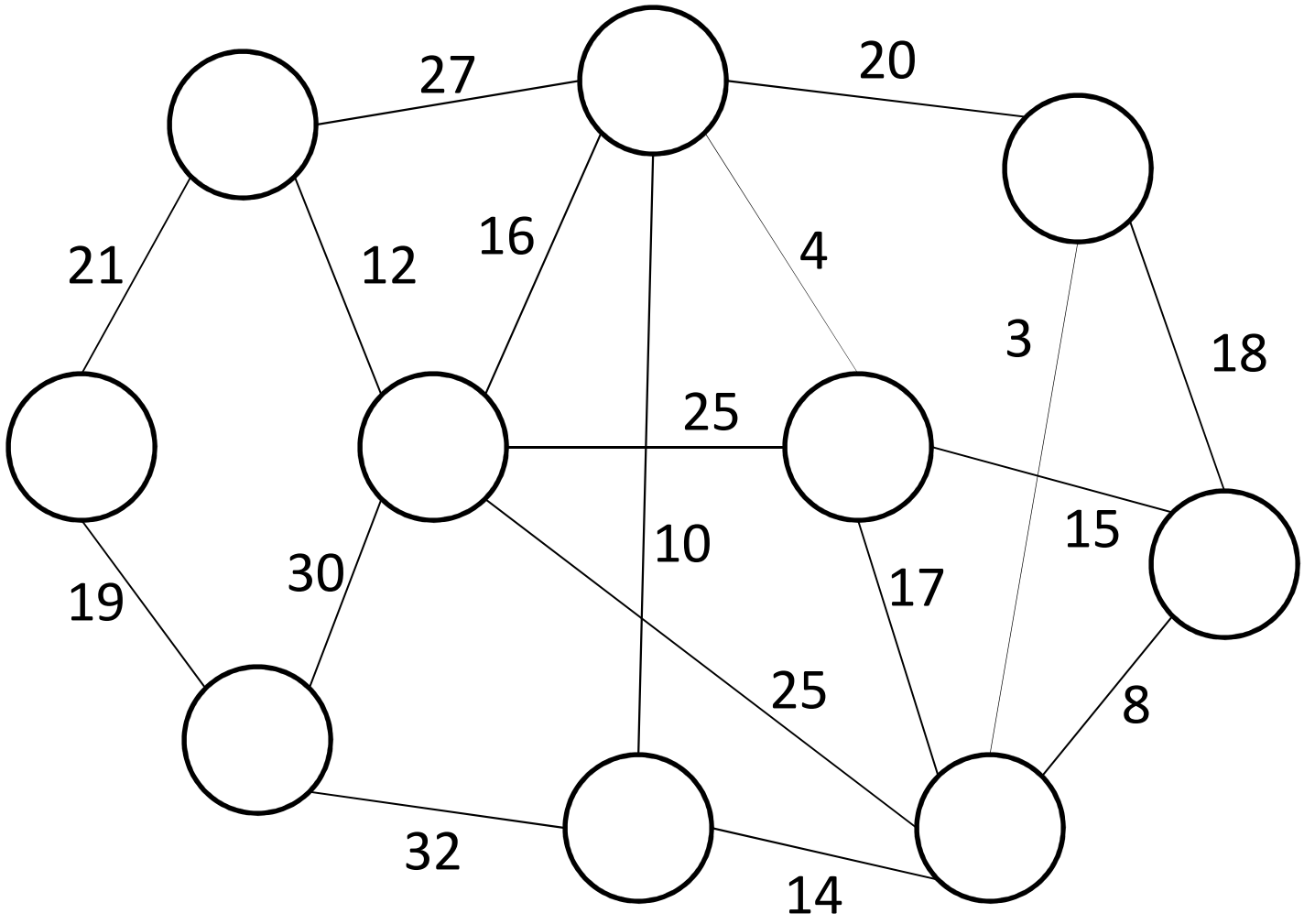
Global greedy (Kruskal, 1956): Process the edges in increasing order by weight. To process an edge, if its ends are in different blue trees color it blue; otherwise, color it red.

Single-source greedy (Jarník 1929, Kruskal 1956, Prim 1957, Dijkstra 1958): Choose a start vertex *s*. While the blue tree *B* containing *s* is not spanning, color blue the minimum-weight edge with exactly one end in *B*.

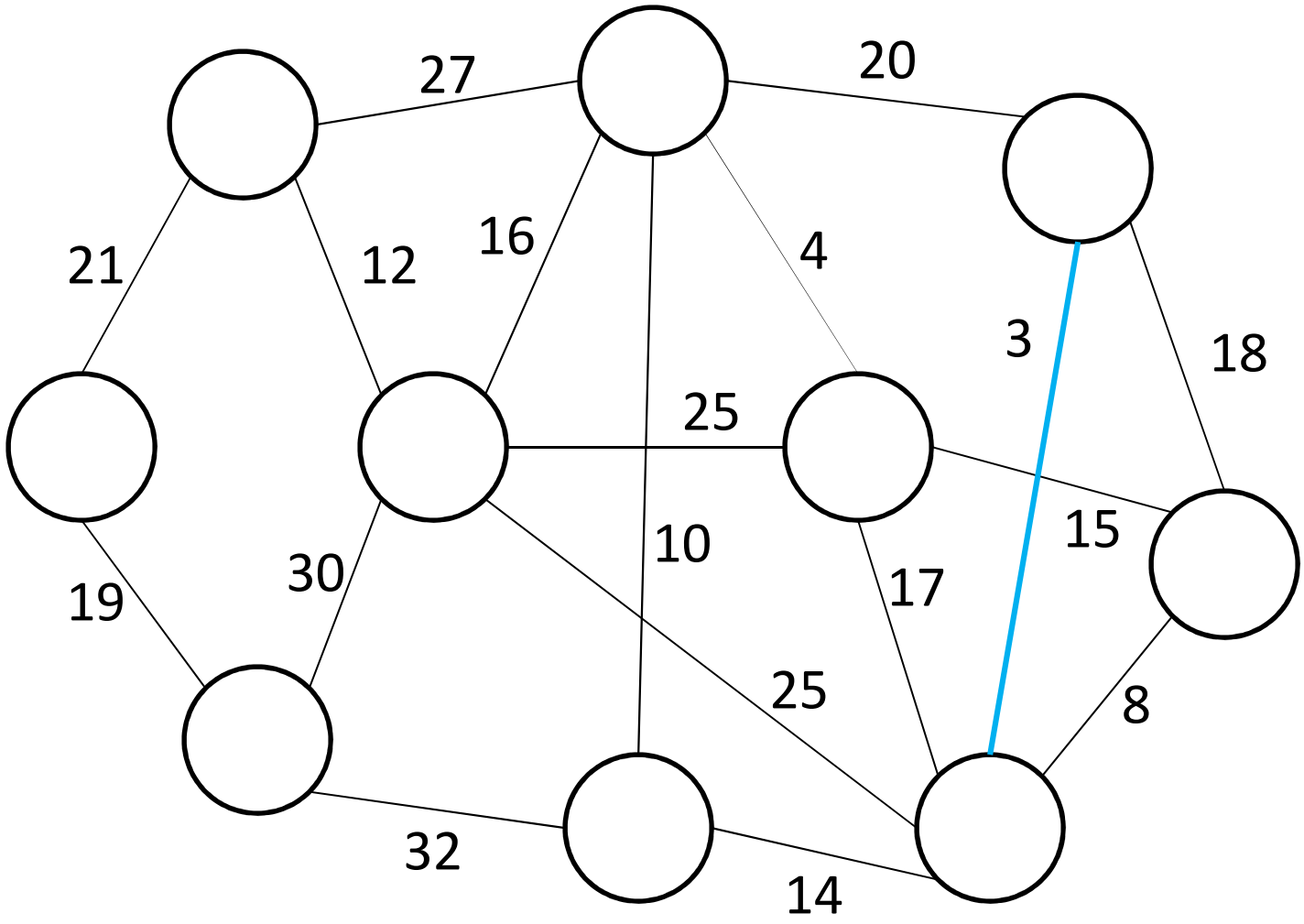
Concurrent greedy (Borůvka, 1926): For each blue tree, choose the minimum-weight edge with one end in the tree. Color all the chosen edges blue. Repeat until there is only one blue tree.

All three of these algorithms are special cases of the generalized greedy algorithm.

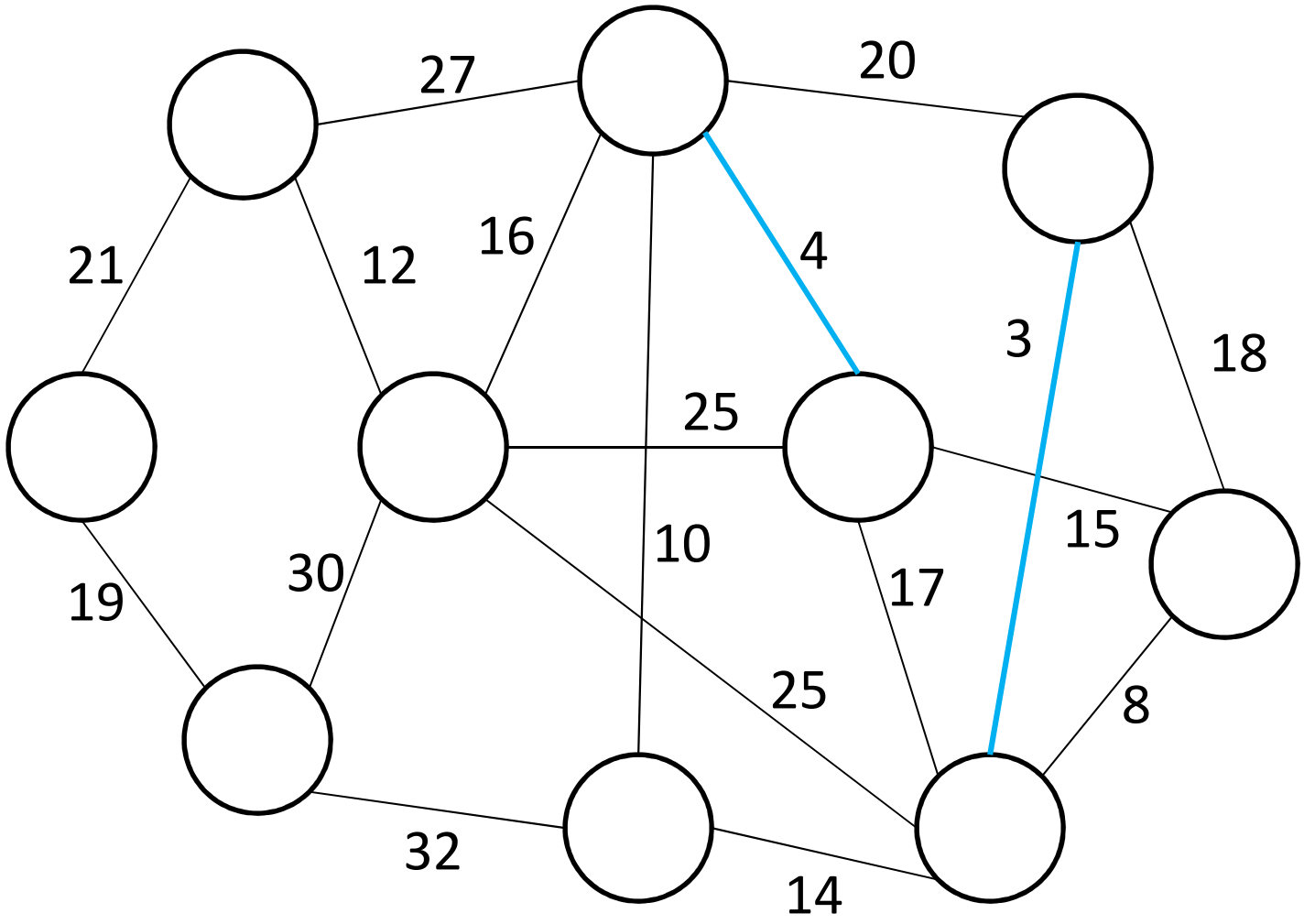
Global greedy



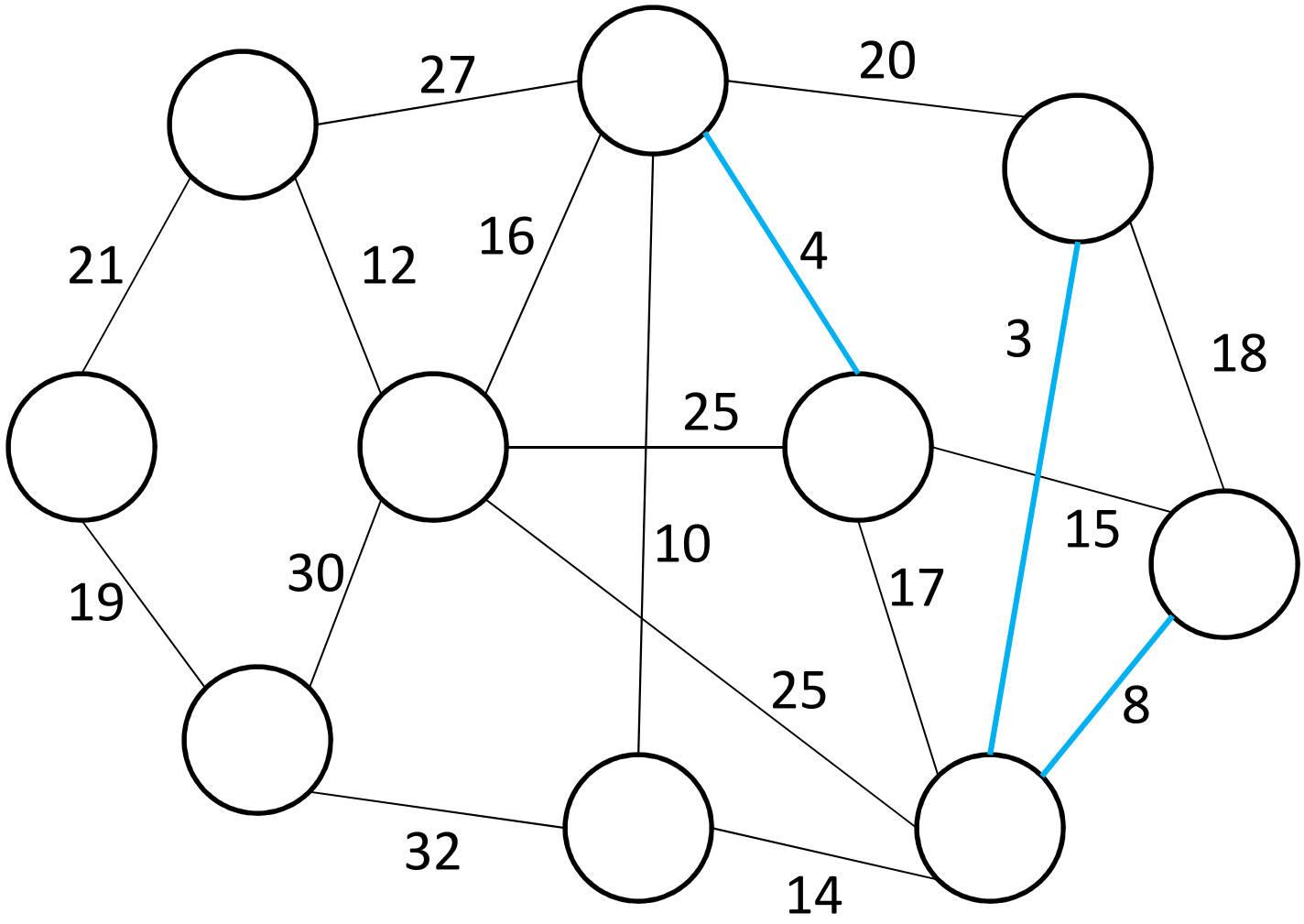
Global greedy



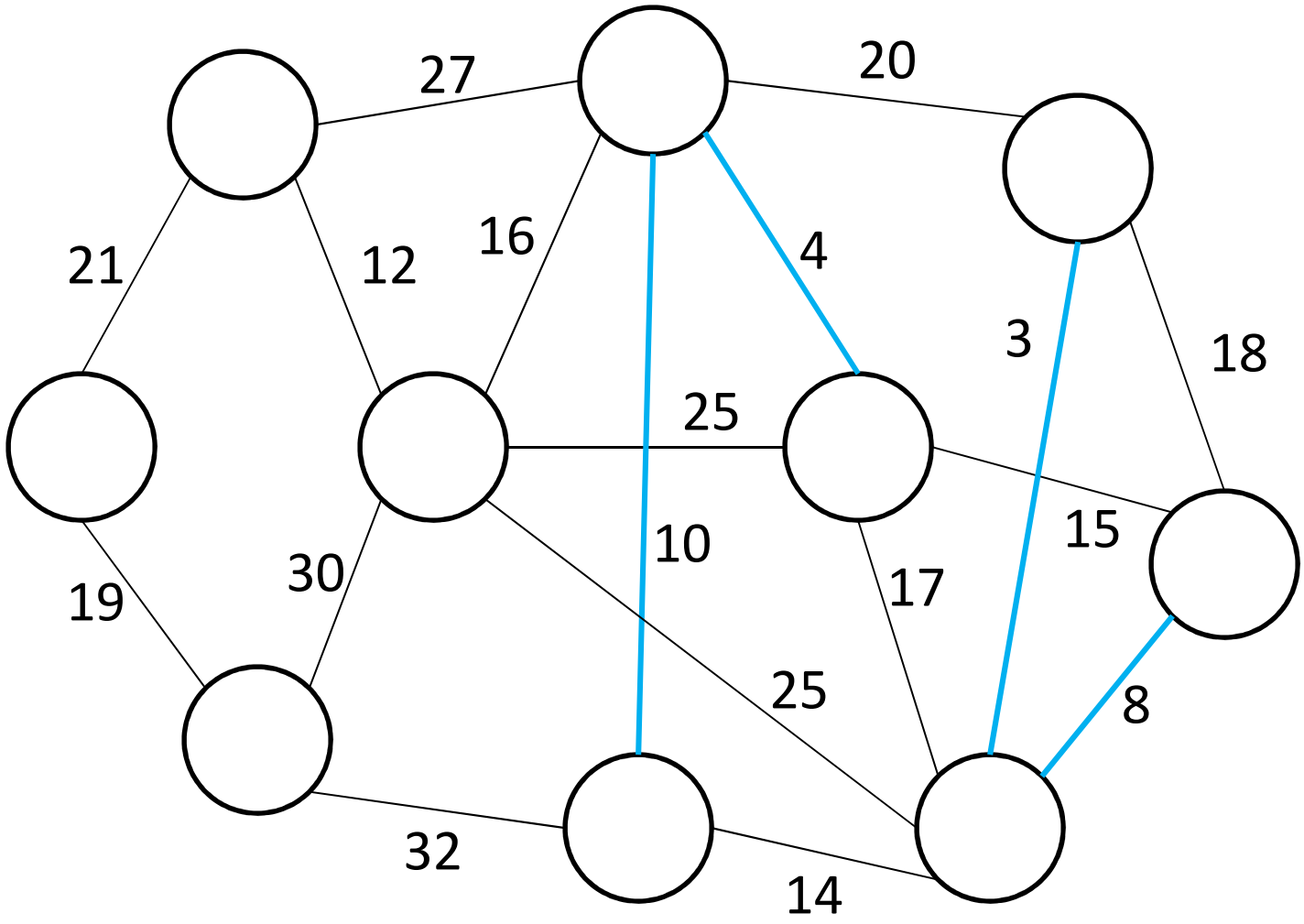
Global greedy



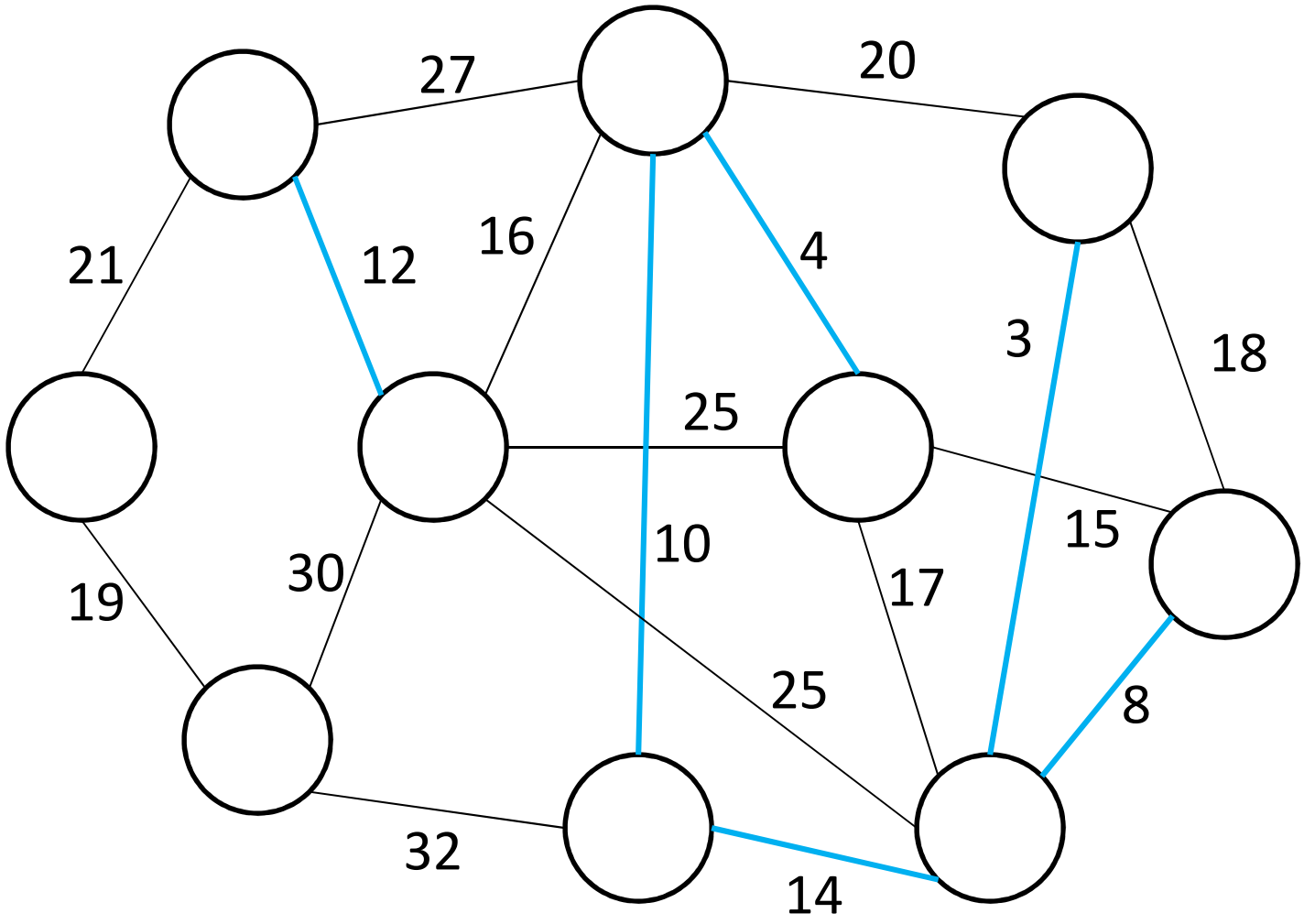
Global greedy



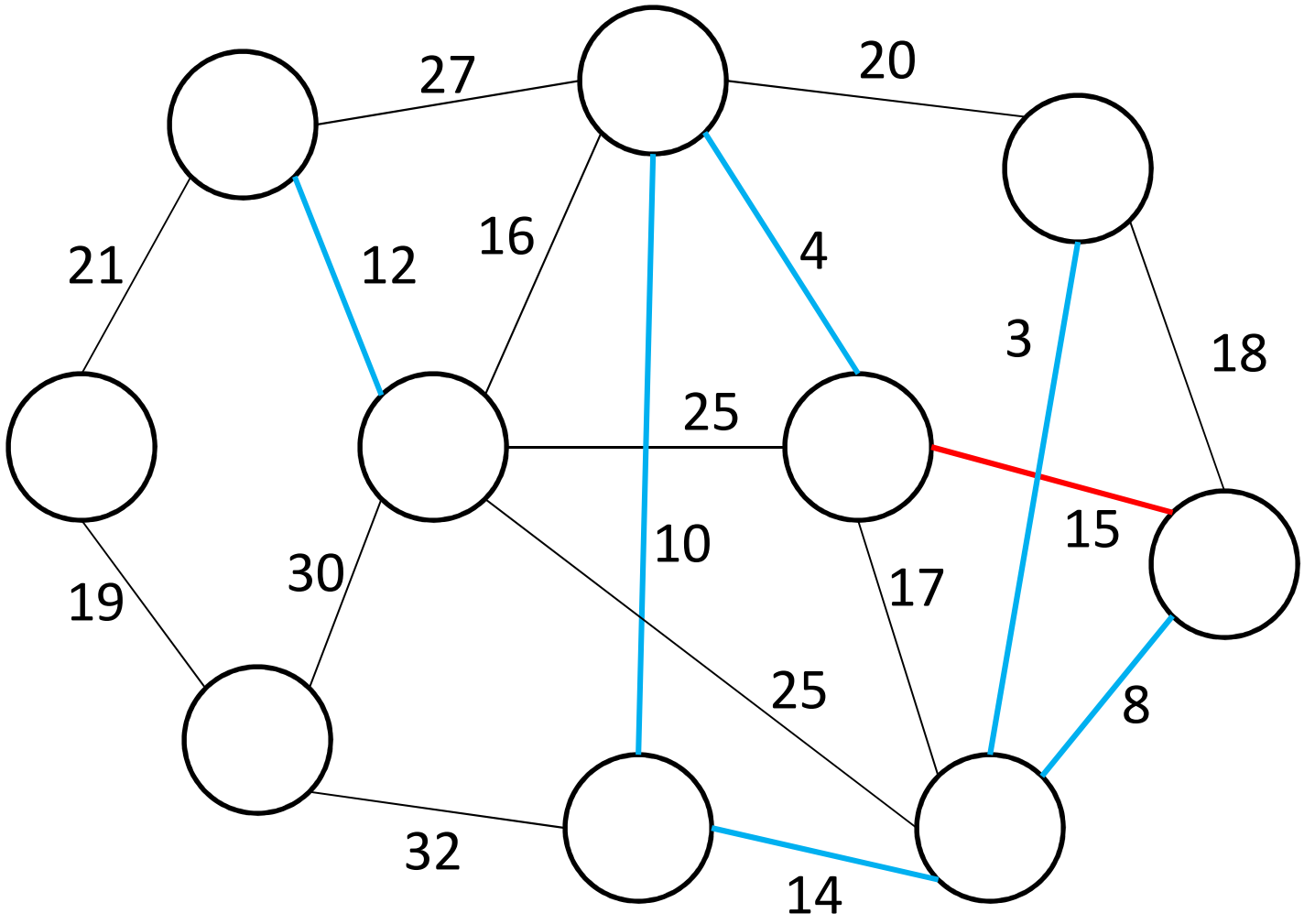
Global greedy



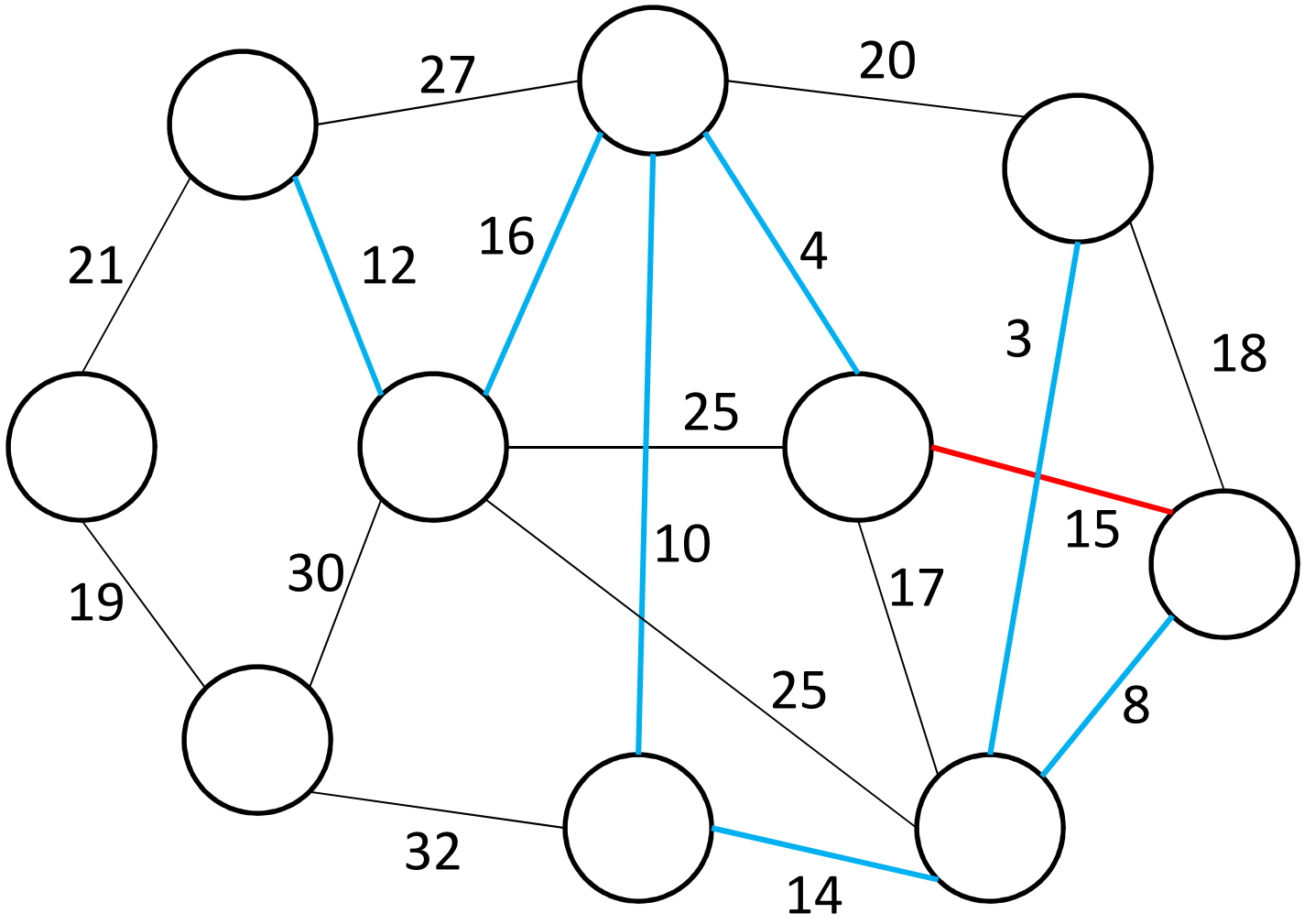
Global greedy



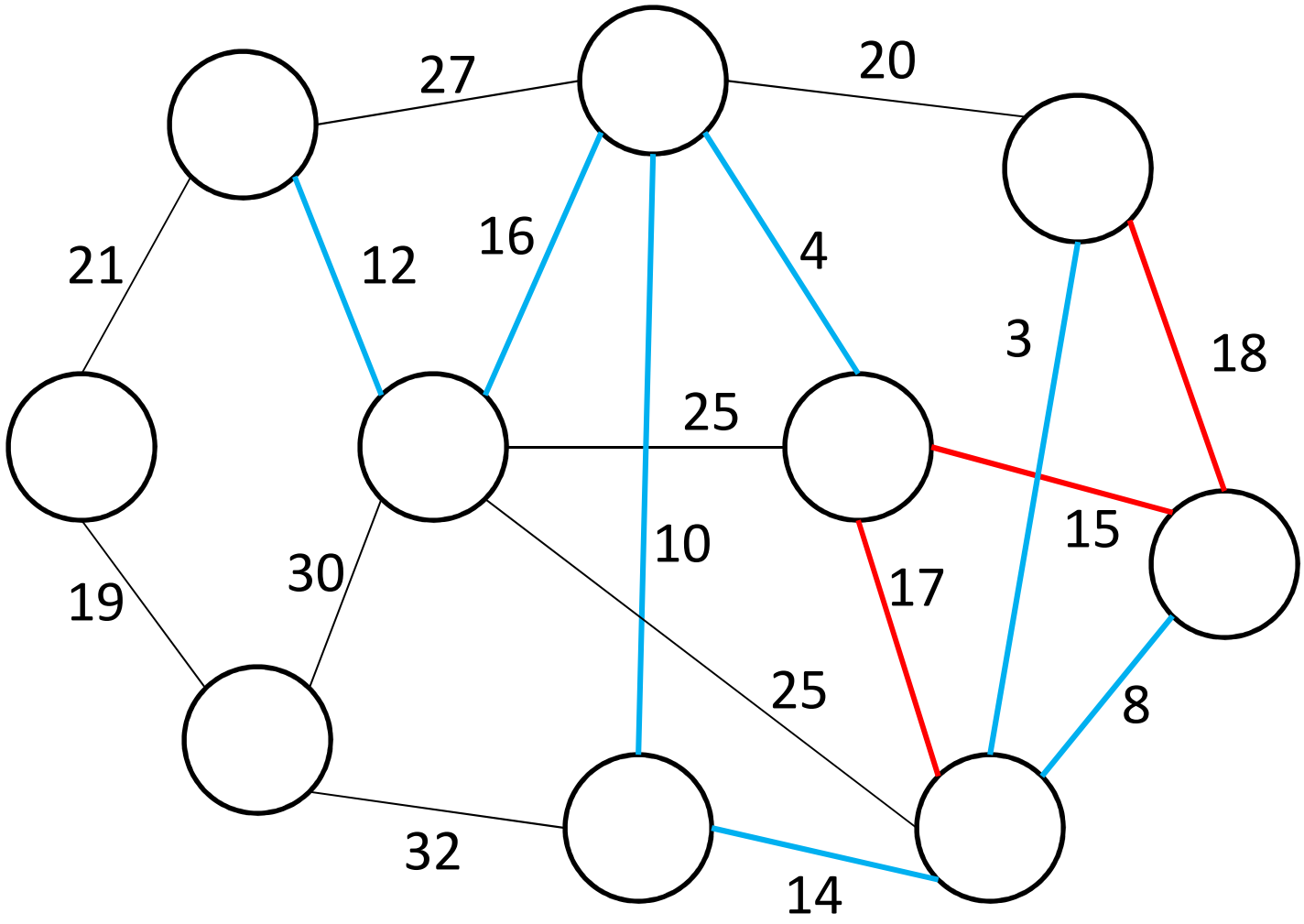
Global greedy



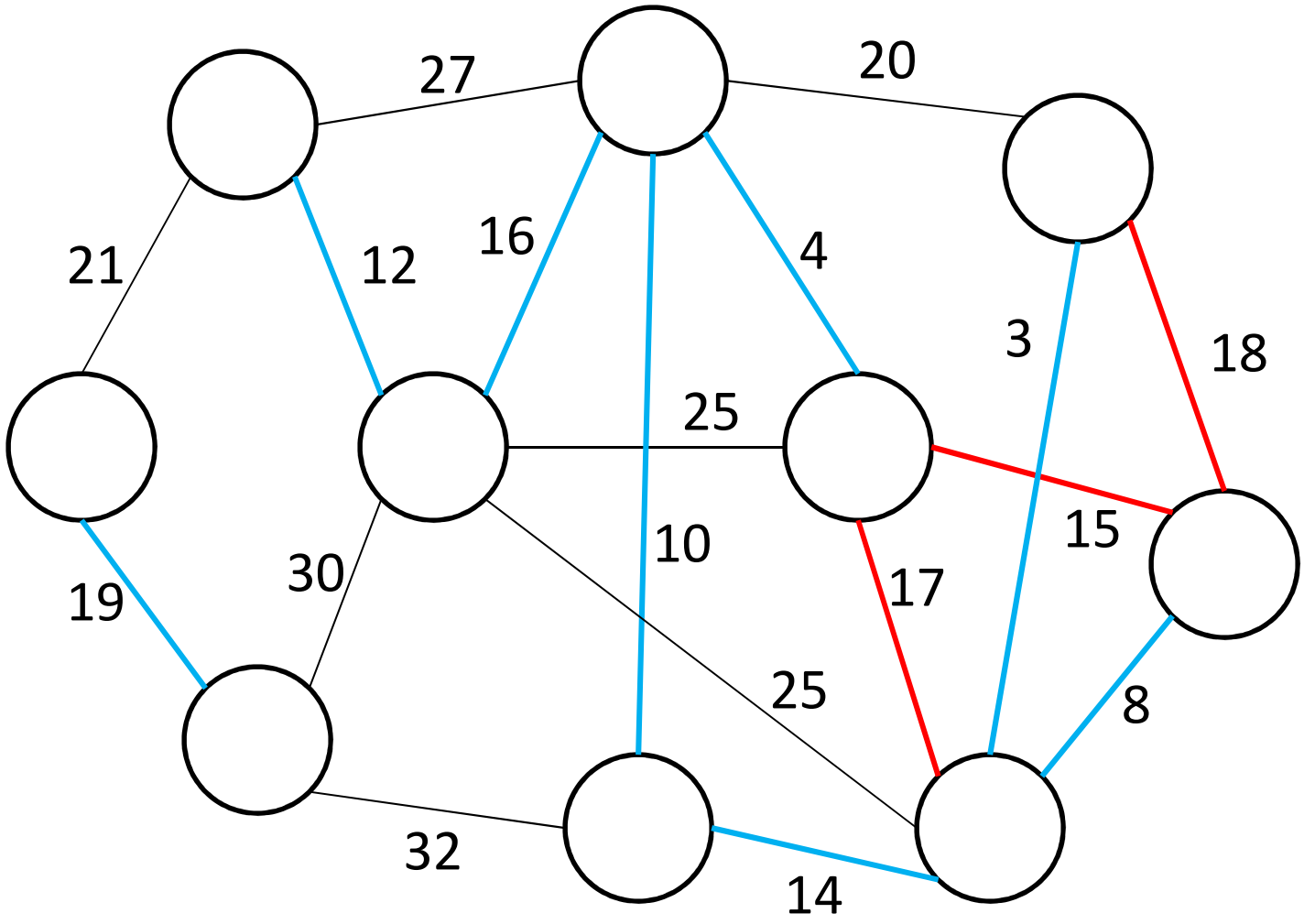
Global greedy



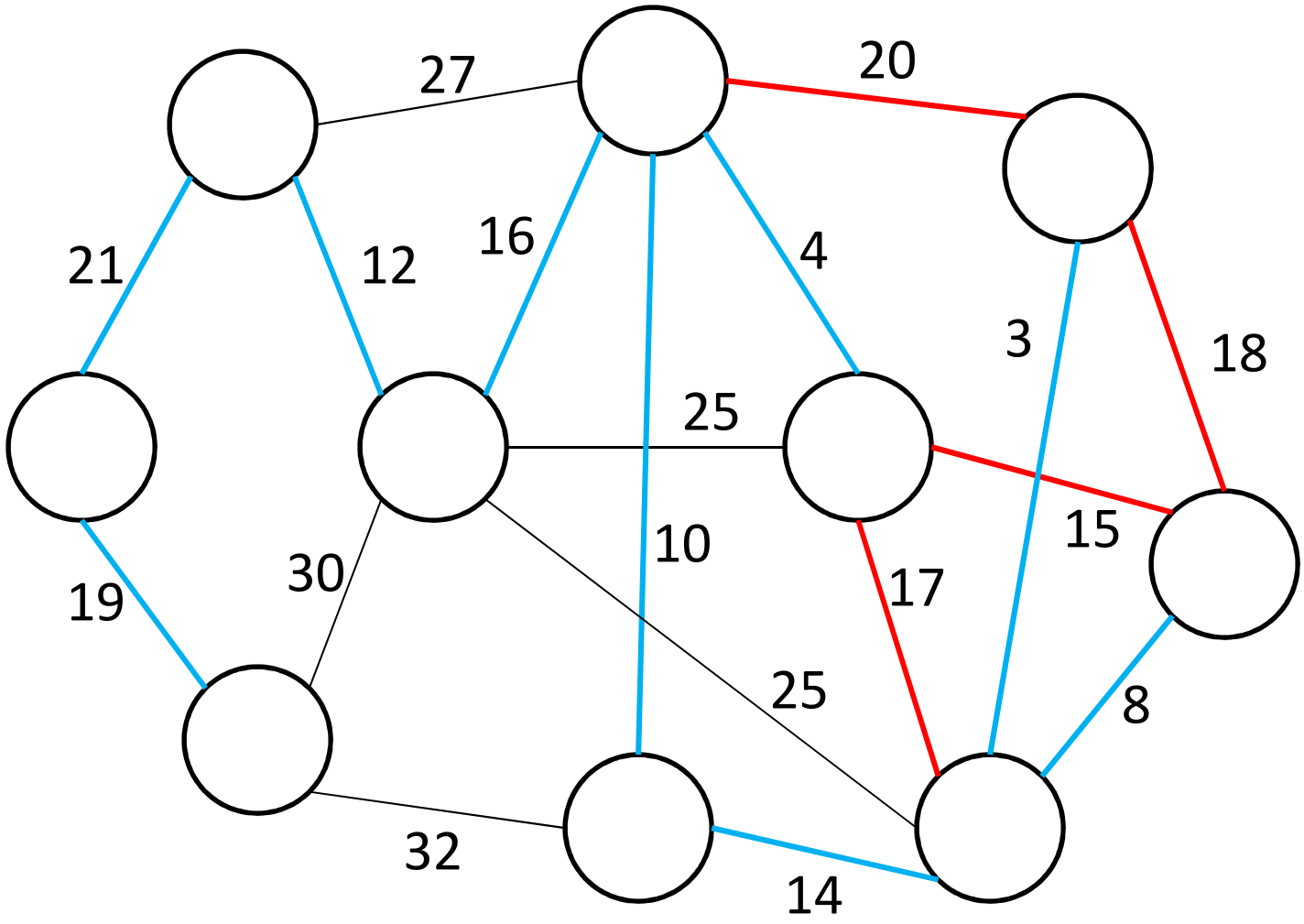
Global greedy



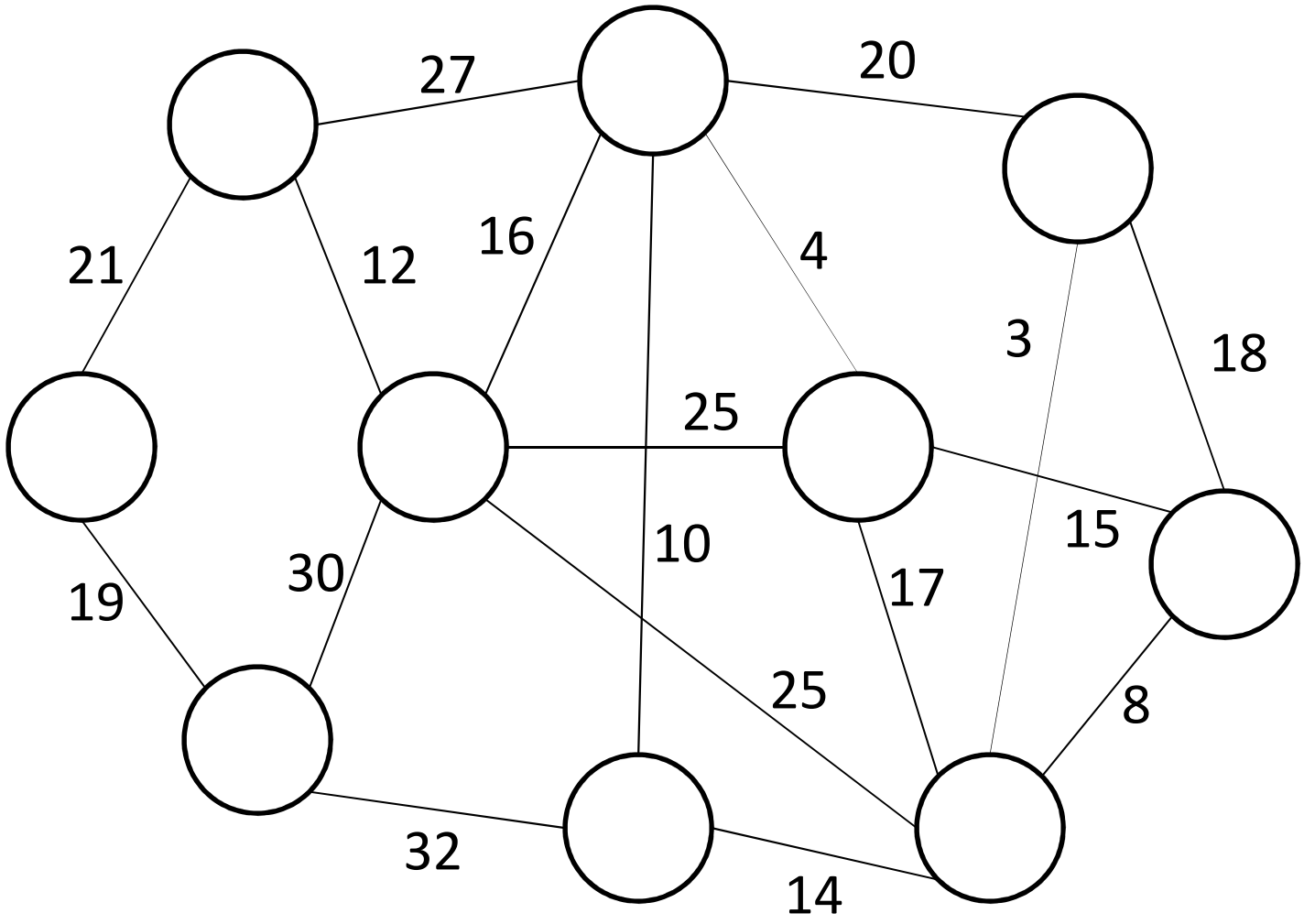
Global greedy



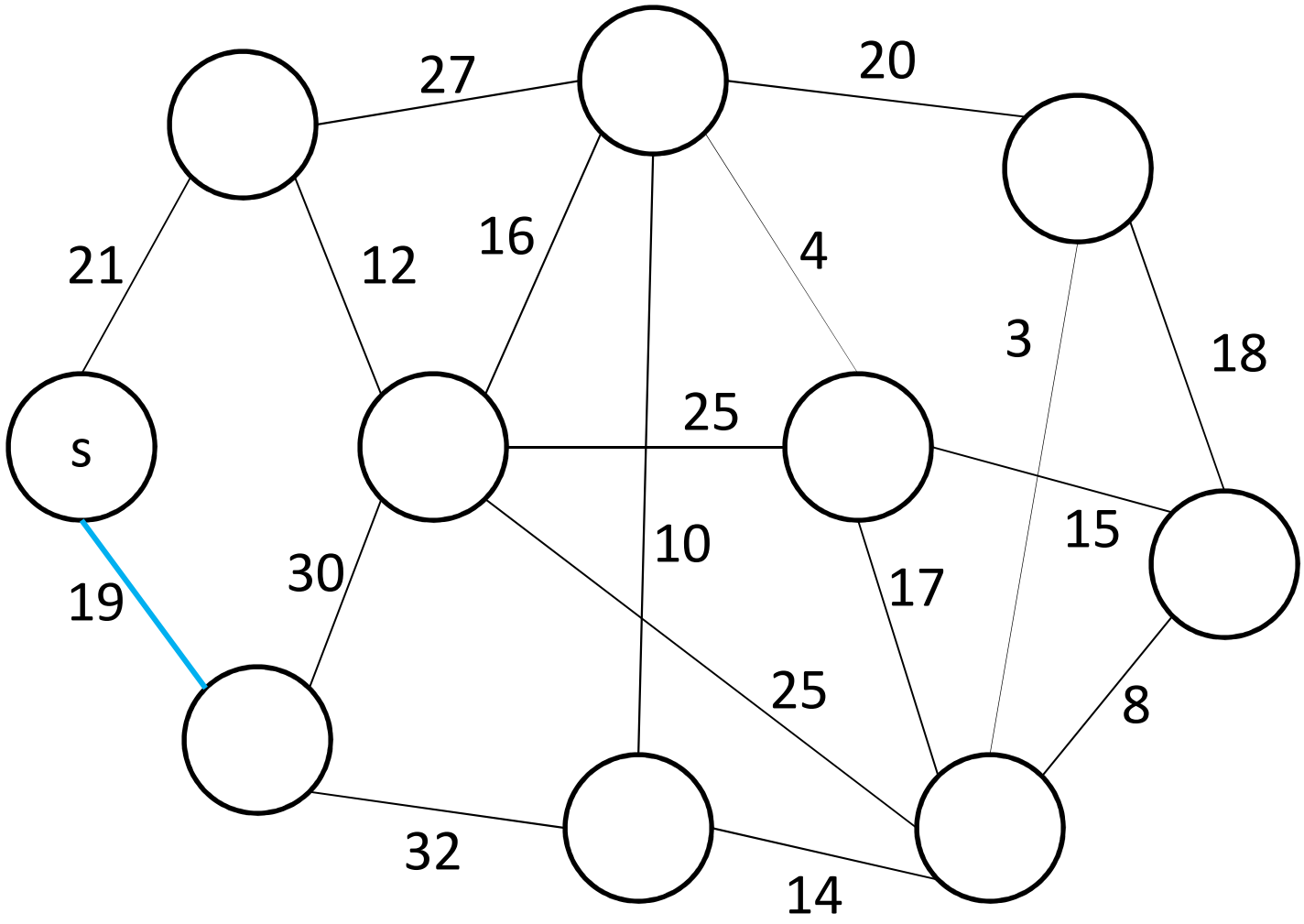
Global greedy



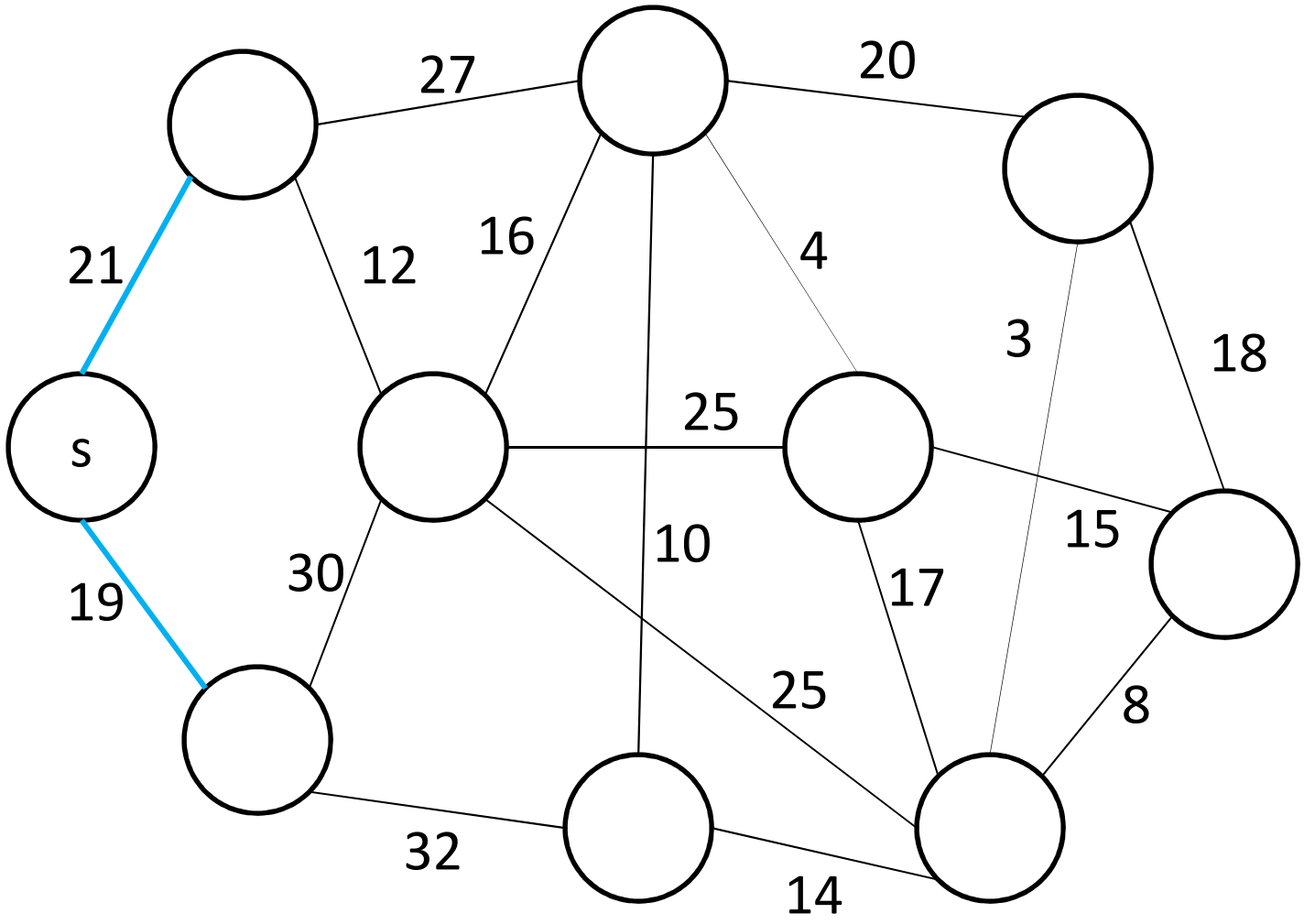
Single-source greedy



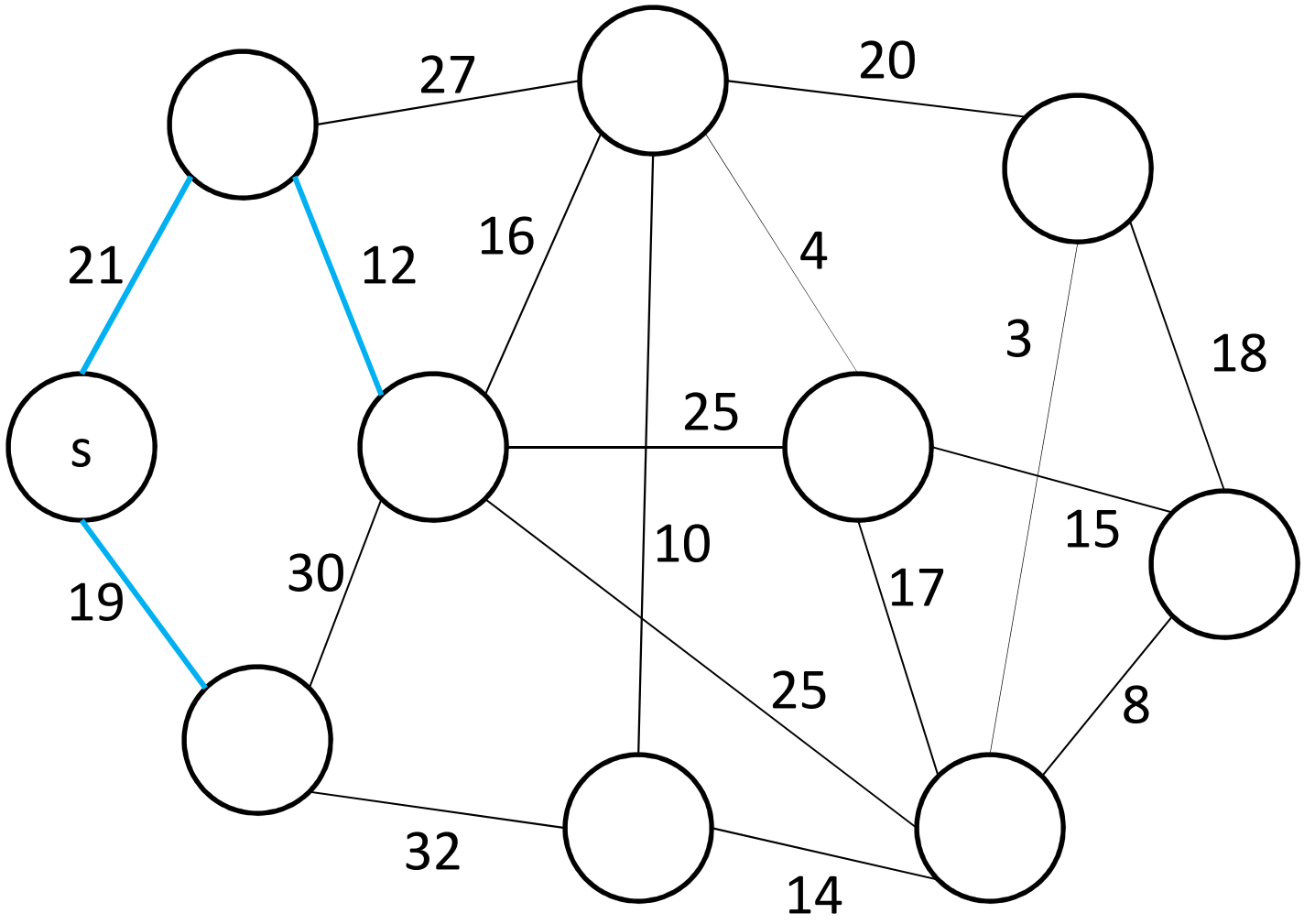
Single-source greedy



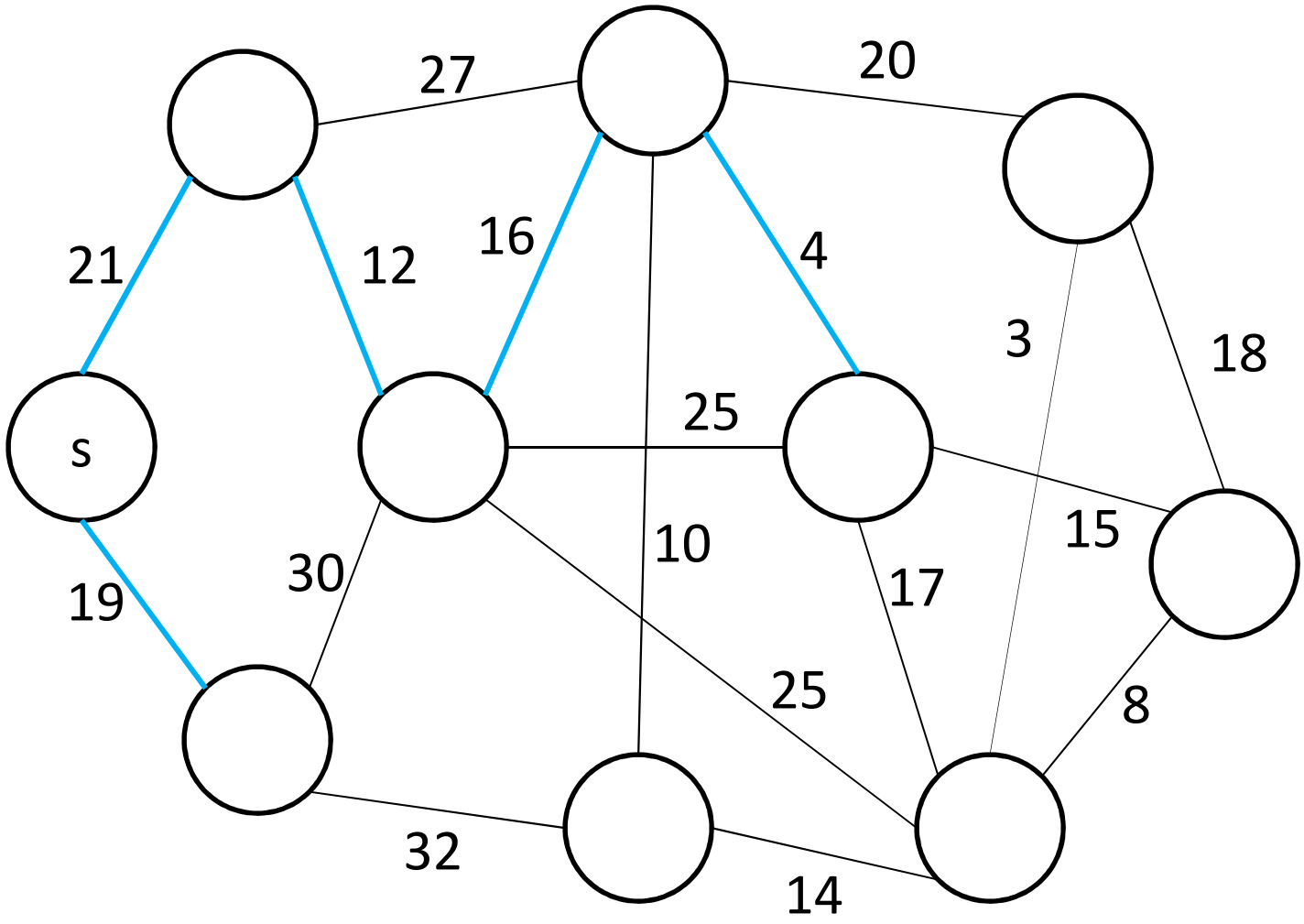
Single-source greedy



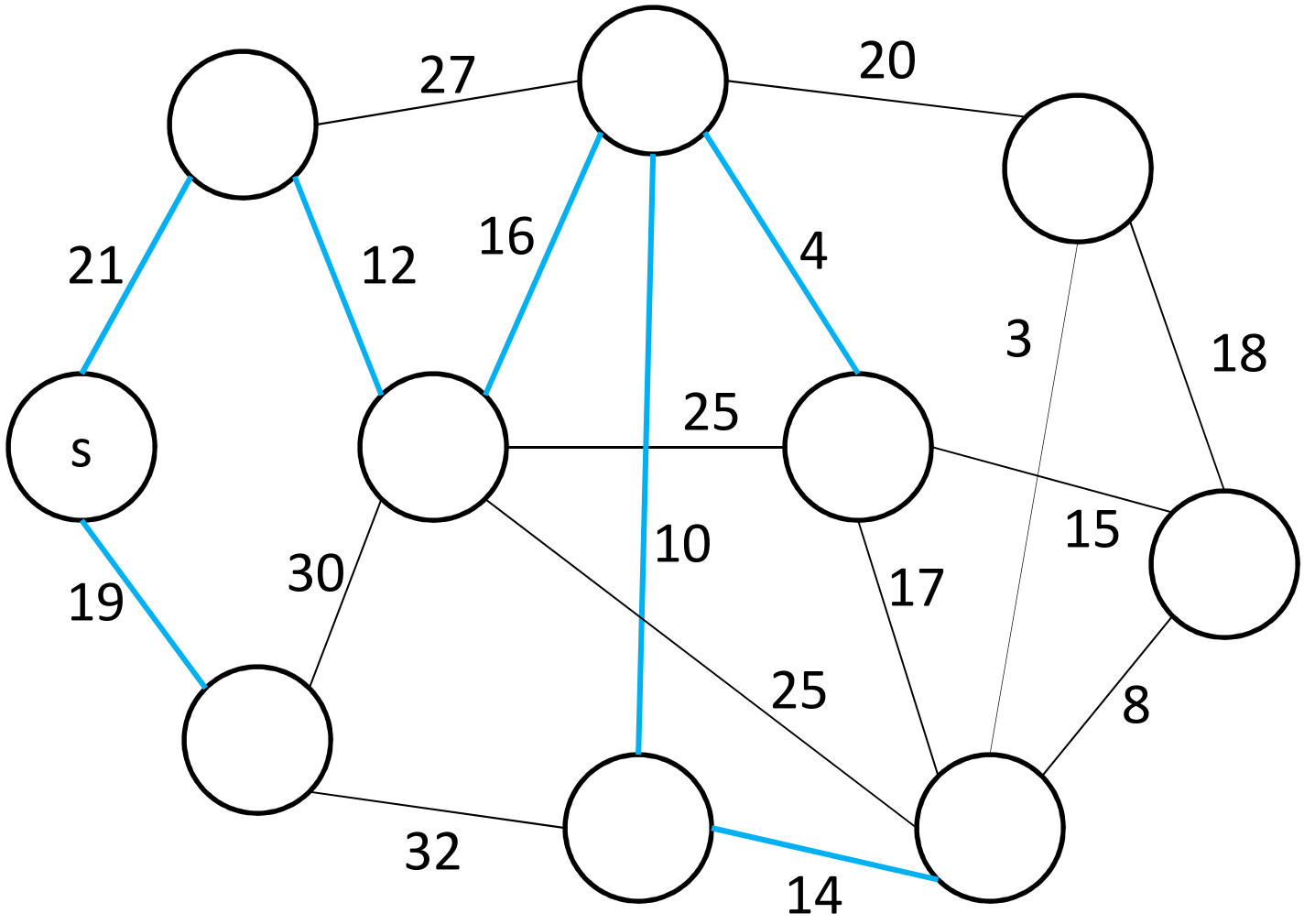
Single-source greedy



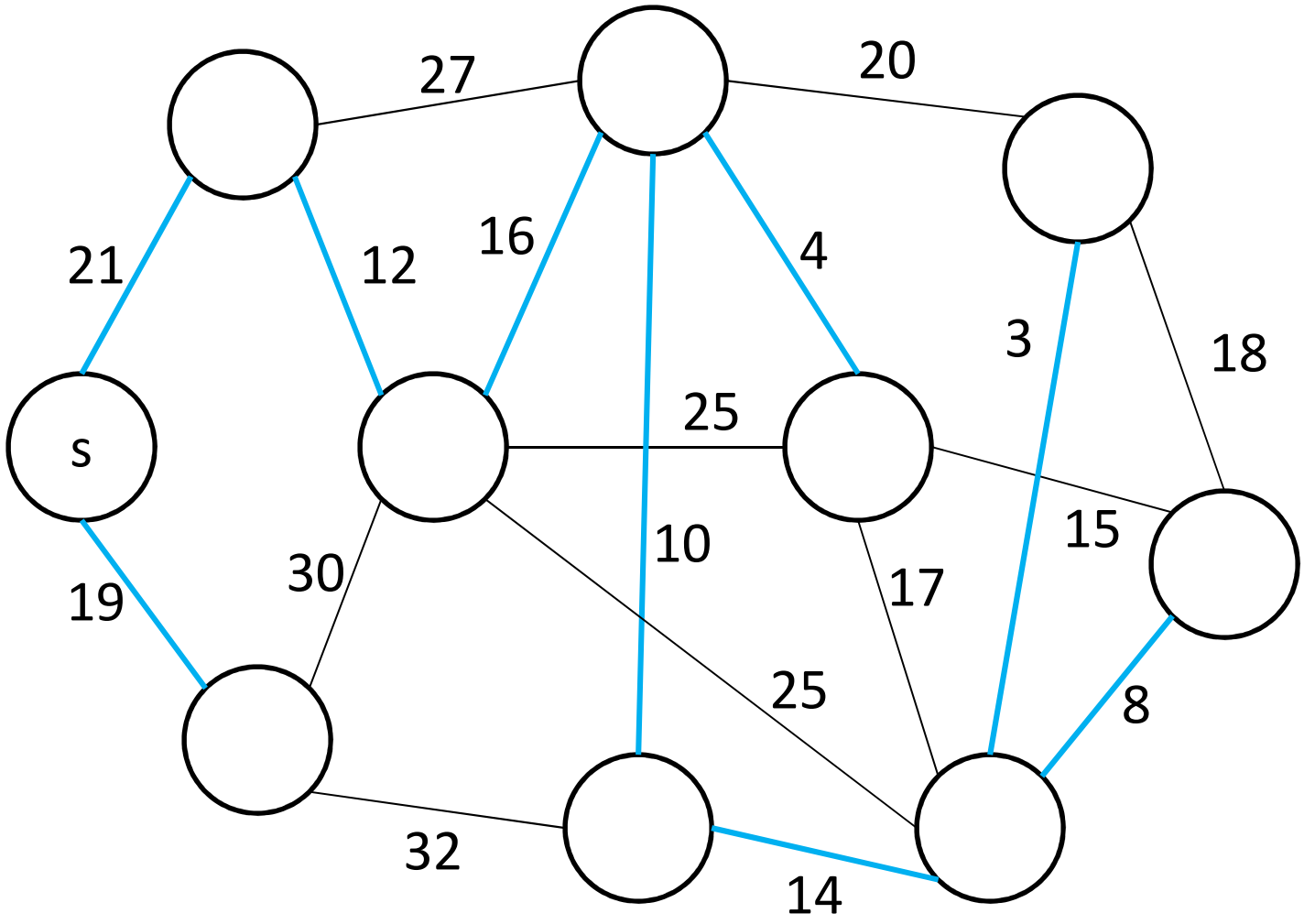
Single-source greedy



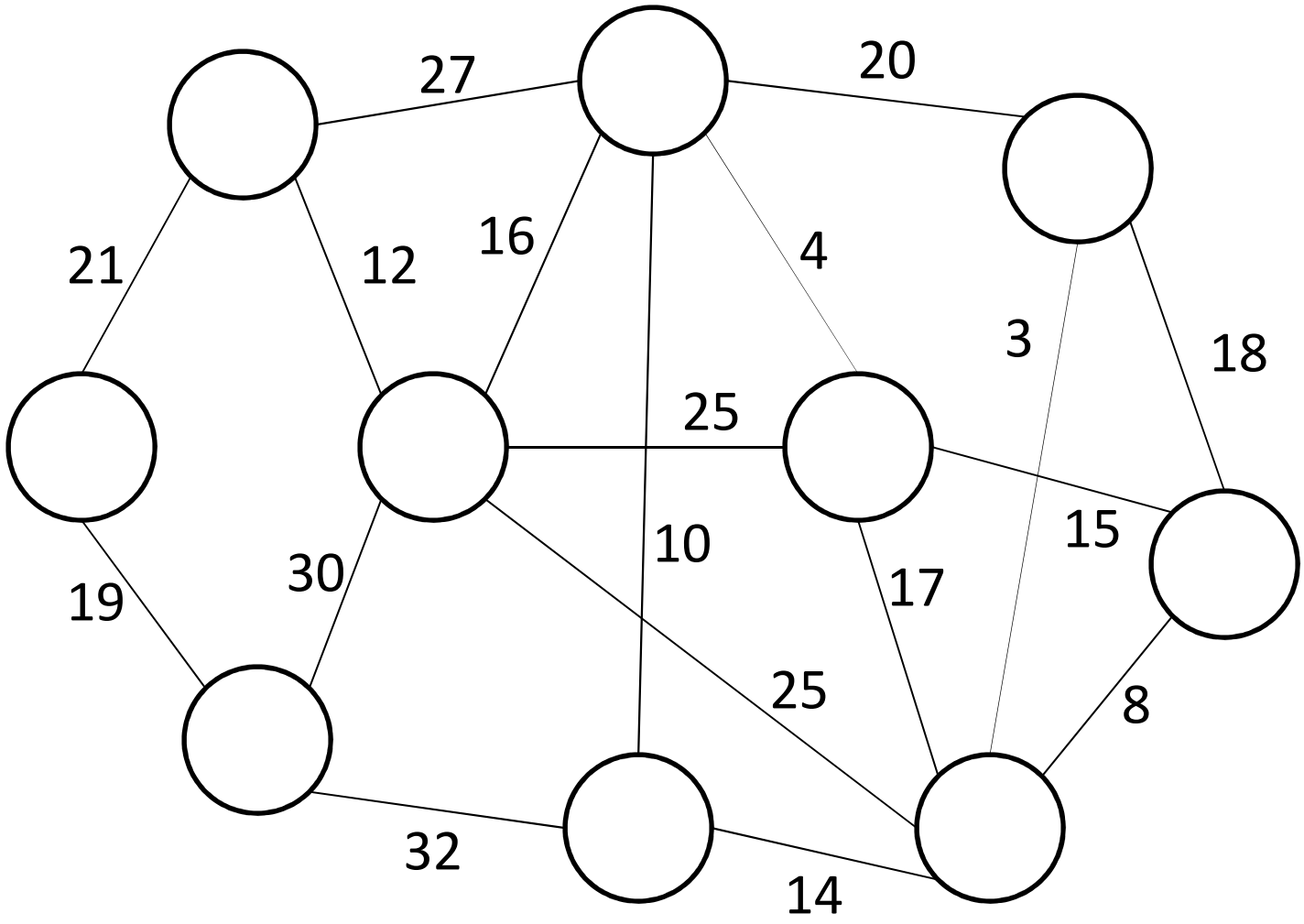
Single-source greedy



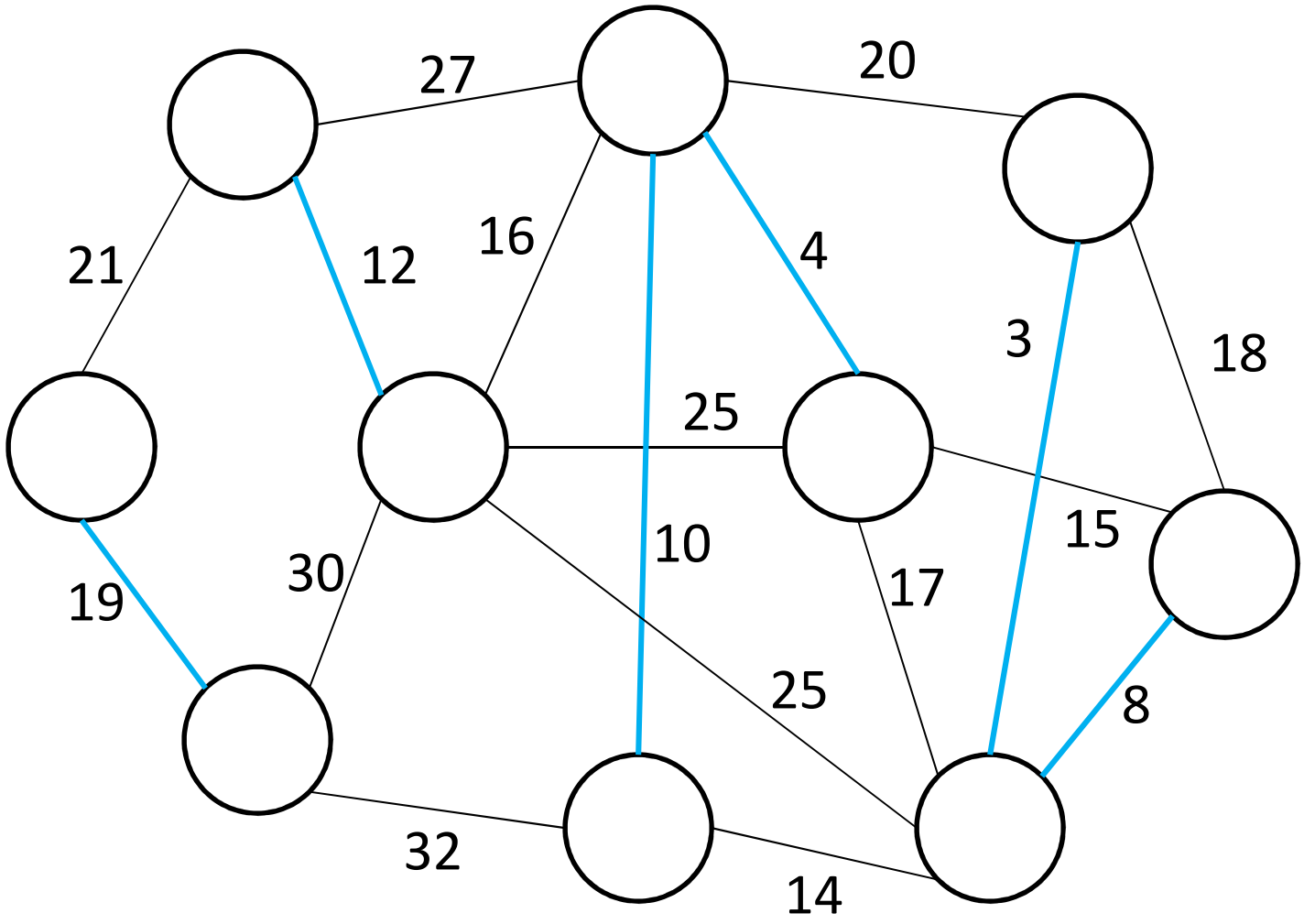
Single-source greedy



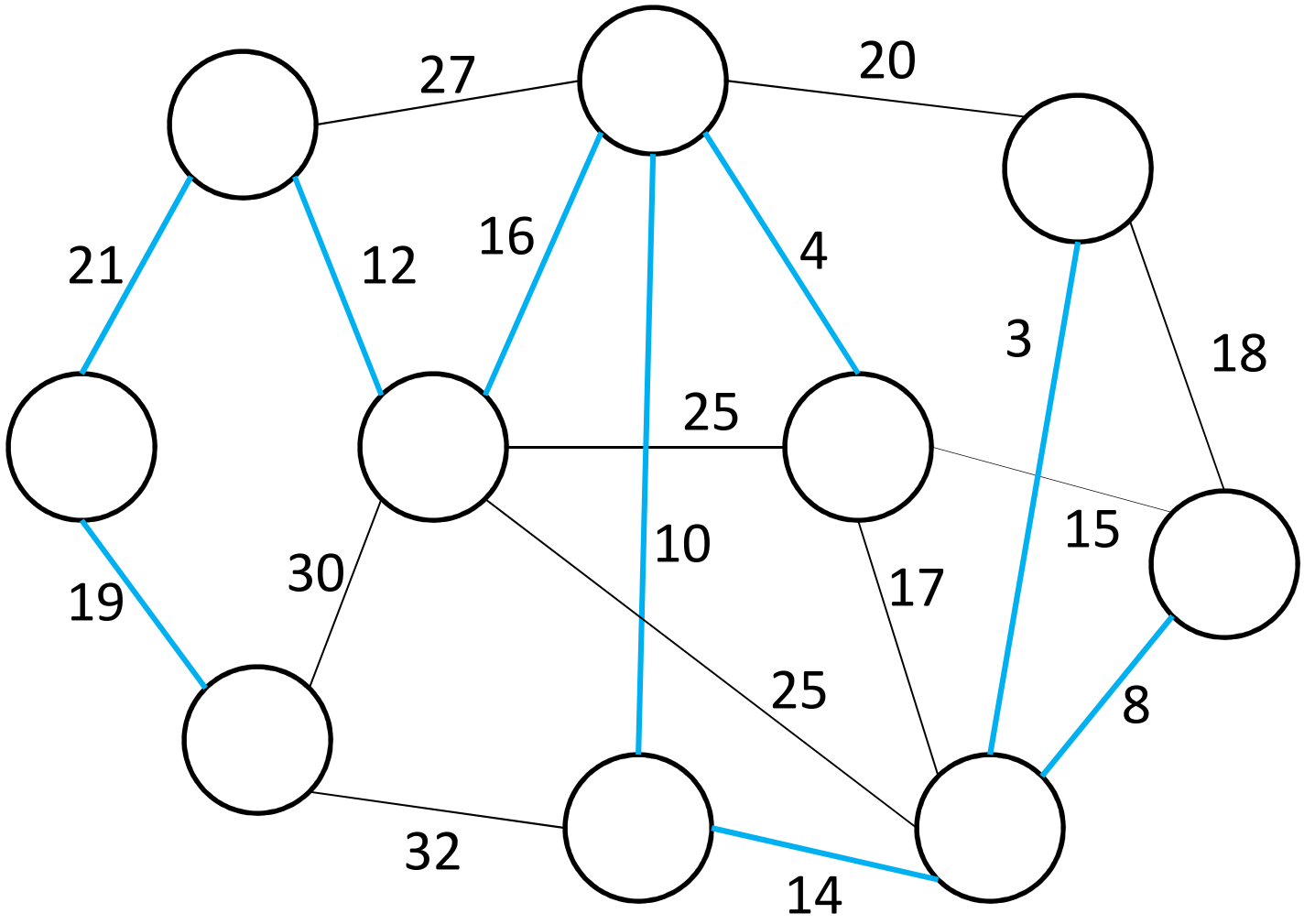
Concurrent greedy



Concurrent greedy



Concurrent greedy



Correctness?

Implementation and efficiency?

Fastest version?

No edge can be colored both blue and red.

Proof: Suppose (v, w) is minimum crossing X, Y and maximum on C. Some other edge (x, y) on C must cross X, Y. $c(v, w) < c(x, y) < c(v, w)$: contradiction.

→ there is no cycle of blue edges: the blue edges always form a set of trees

Let T be an MST. Any edge in T can be colored blue; any edge not in T can be colored red.

Proof: Let (v, w) be an edge in T. Deleting (v, w) from T forms two trees. Let X be the vertex set of one tree and Y that of the other. Then (v, w) is the minimum-weight edge crossing X, Y: if some other edge (x, y) crossing X, Y has smaller weight, the tree formed from T by deleting (v, w) and adding (x, y) has smaller weight than T, a contradiction. Thus (v, w) can be colored blue.

Proof (cont.): Let (v, w) be an edge not in T. The path in T connecting v and w forms a cycle with (v, w). Edge (v, w) must have maximum weight on this cycle: if some edge (x, y) on the cycle has larger weight, the tree formed by deleting (x, y) from T and adding (v, w) has less weight than T, a contradiction.

→ generalized greedy algorithm is correct, MST is unique

Implementations

Single-source scanning: Use a heap *H* of vertices not in the blue tree *B* but connected to it by at least one edge

k(*v*) = minimum weight of an arc connecting v to blue tree

(*p*(*v*), *v*) = connecting arc of minimum weight

E = edge set of graph

for $v \in V$ **do** $k(v) \leftarrow \infty$;

$H \leftarrow$ *make-heap*; initialize B to contain s and no edges;

for $(s, v) \in E$ **do**

{$k(v) \leftarrow c(s, v)$; $p(v) \leftarrow s$; *insert*(v, H)};

while $H \neq \{\ \}$ **do**

{$v \leftarrow$ *delete-min*(H); add $(p(v), v)$ to B;

for $(v, w) \in E$ **do if** $c(v, w) < k(w)$ **and** w not in B

then {$k(w) \leftarrow c(v, w)$; $p(w) \leftarrow v$;

if w not in H then *insert*(w, H)

else *decrease-key*$(w, k(v), H)$}

Graph representation: for each vertex, set of incident edges. Each edge is in two such sets.

$n - 1$ *insertions*, $m - n + 1$ *decrease-keys*

→$O(m \lg n)$ time (implicit heap or pairing heap)

$O(m + n \lg n)$ time (rank-pairing heap)

(vs. $O(n^2)$ original implementation:

heap = unordered set)

Global greedy: Sort edges by weight or store in a heap with key = weight

Need a data structure to keep track of the vertex sets of the blue trees. Initially each vertex is in its own tree. Operations (first cut):

find(*x*): return the set containing *x*

unite(*A, B*): unite the sets *A* and *B*

Process the edges in sorted order.

process(v, w): **if** *find*(v) $\neq$ *find*(w) **then**

{*unite*(*find*(v), *find*(w)), color (v, w) blue}

Stop after $n - 1$ edges are colored blue.

The running time is dominated by the sorting time, or by the heap operations if a heap is used: $O(m \lg n)$

Concurrent greedy: after k passes, each blue tree contains at least 2^k vertices → $\leq \lg n$ passes

To do a pass: For each blue tree, keep track of the minimum-weight connecting edge, initially null. For each arc, find the blue trees containing its ends; update the minimum-weight connecting arcs of these blue trees appropriately. (If both ends in the same tree, can delete the edge.)

To find blue trees: either use disjoint set data structure or find connected components of the blue edges by graph search and number each component.

O(m) time per pass → O($m$lg$n$) time total

Cleanup: Before a pass, number the blue trees. Assign to each edge the tree numbers of its ends. Sort the edges lexicographically by edge number. Delete all edges with both ends numbered the same, and of each group of edges with the same pair of numbers, delete all but the minimum-weight arc.

After cleanup, at most $b^2/2$ edges, if b blue trees

With cleanup, after pass k, $b \le n/2^k$

→ total time for concurrent greedy with cleanup is $O(\min\{n^2, m\lg n\})$

Is $O(n\lg n + m)$ fastest? (Is sorting inherent in MST-finding?)