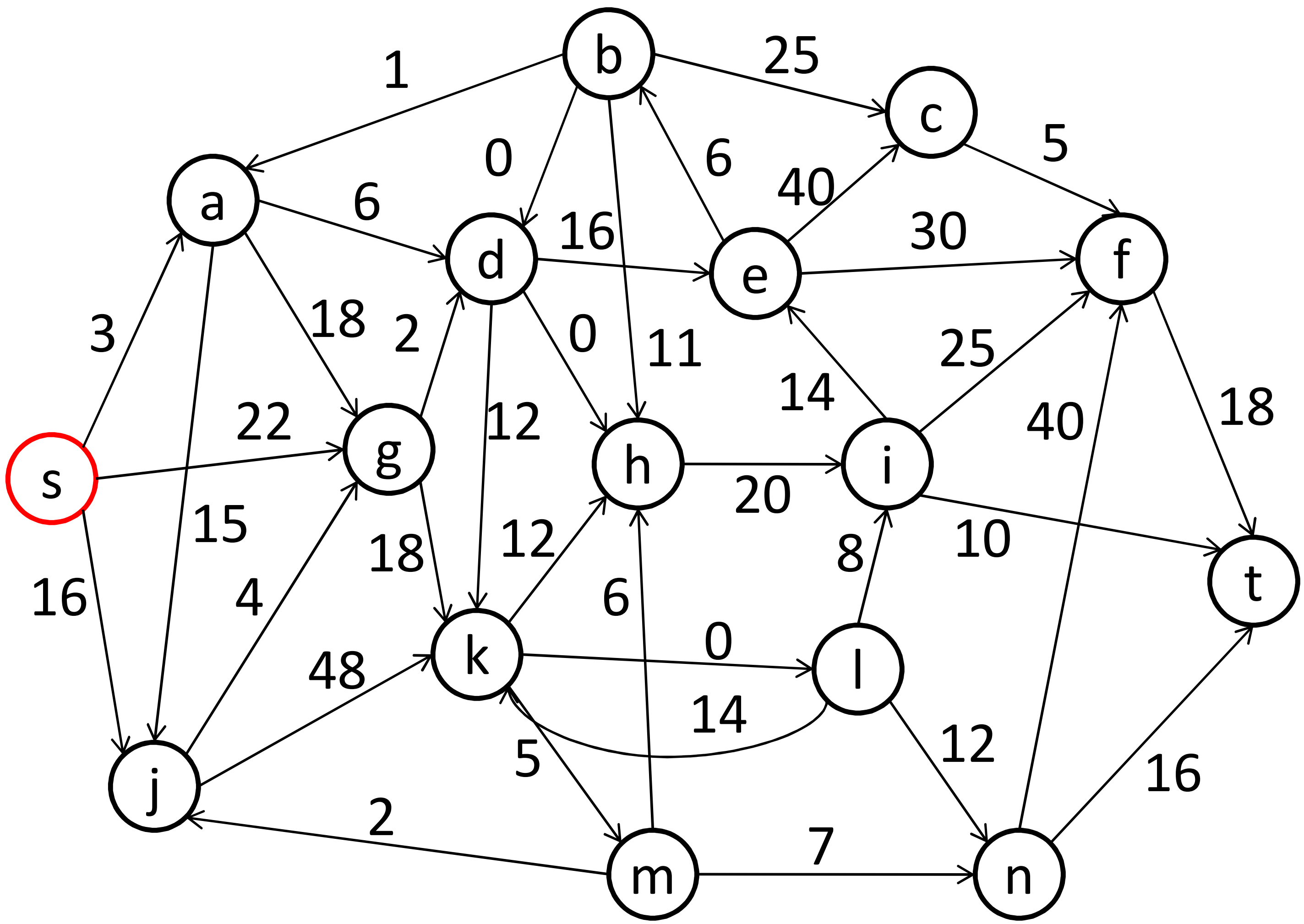
COS 423 Lecture 9
Shortest Paths II

Non-negative arc lengths

Use greedy method:

Shortest-first scanning (Dijkstra's algorithm):
Scan a labeled vertex v with $d(v)$ minimum, breaking a tie arbitrarily.

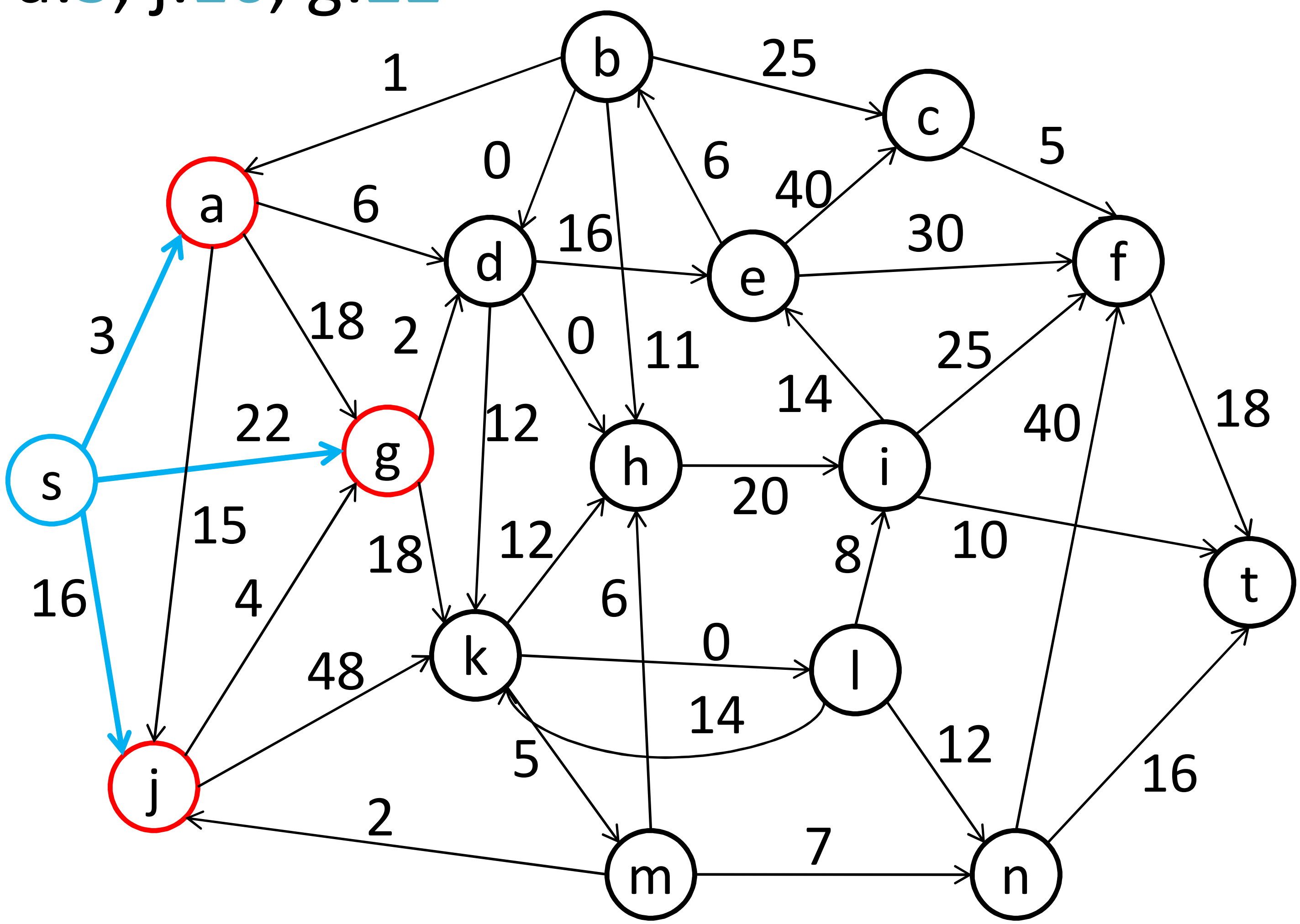
L = s:0   *scan s*



s
a
b
c
d
e
f
g
h
i
j
k
l
m
n
t
1
25
0
6
40
5
6
16
30
3
18
2
0
11
25
14
22
12
18
40
15
18
12
20
10
16
4
48
12
6
8
0
14
5
12
16
2
7

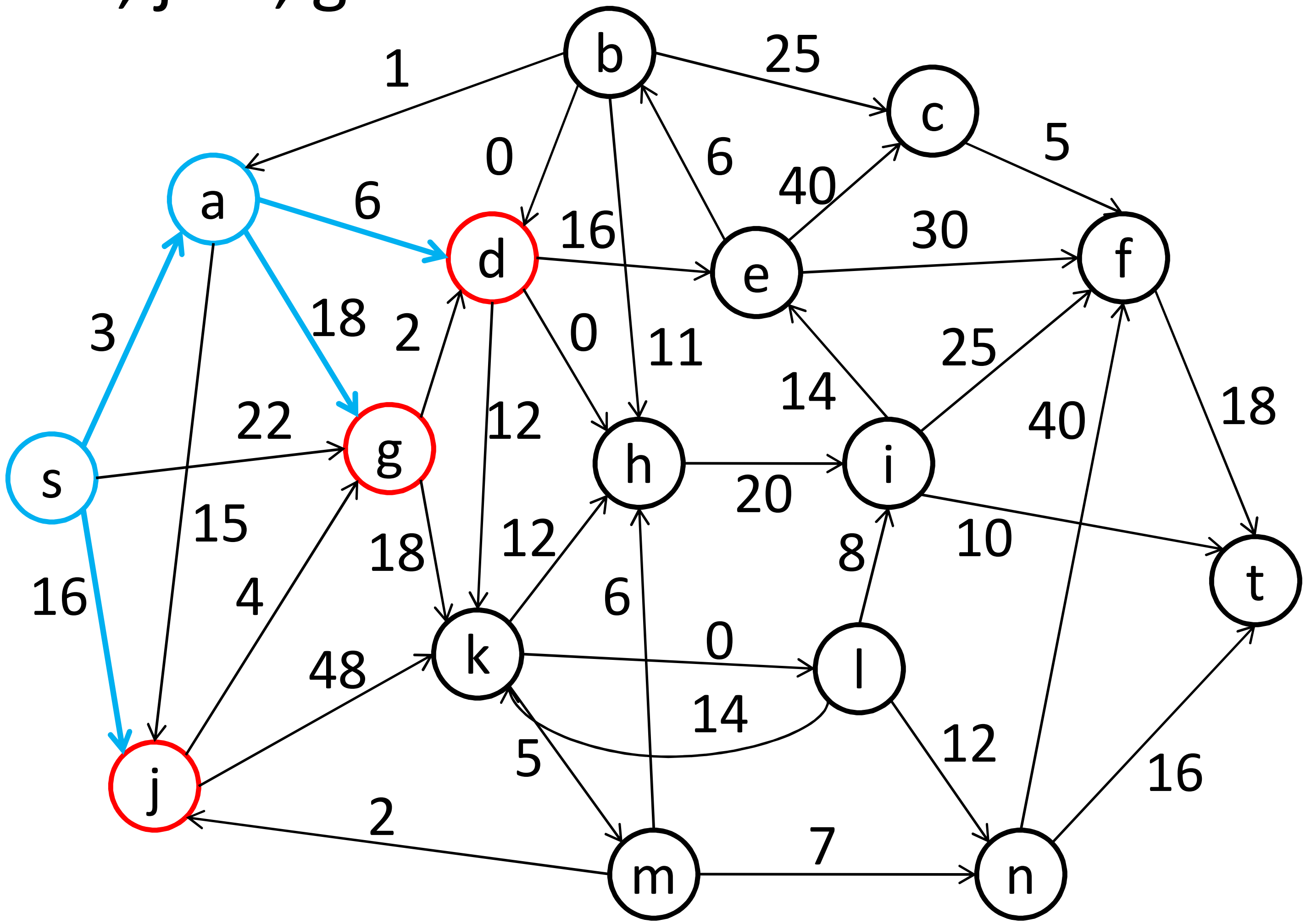
S = s:0 *scan* a

L = a:3, j:16, g:22



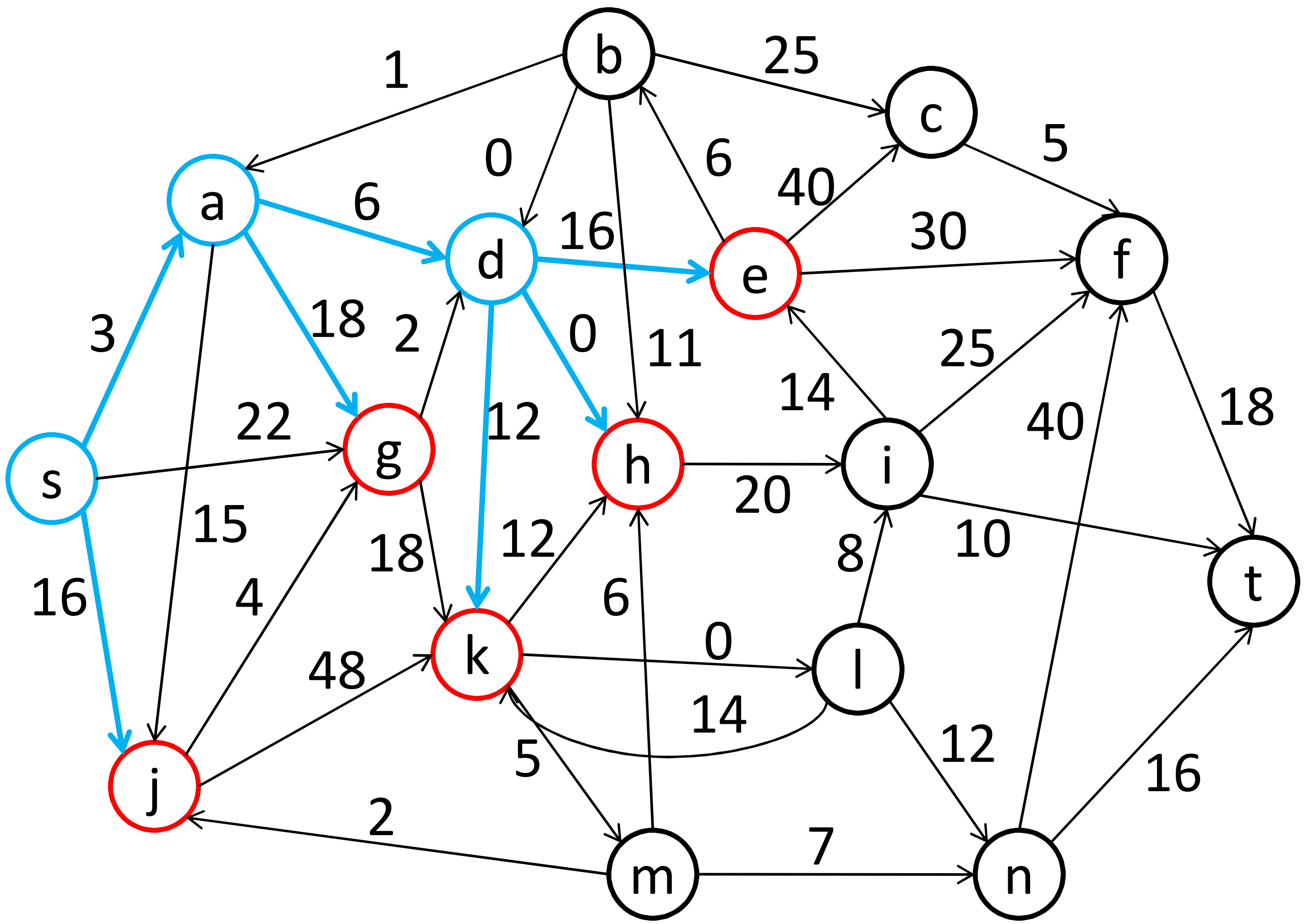
S = s:0, a:3 *scan* d

L = d:9, j:16, g:21



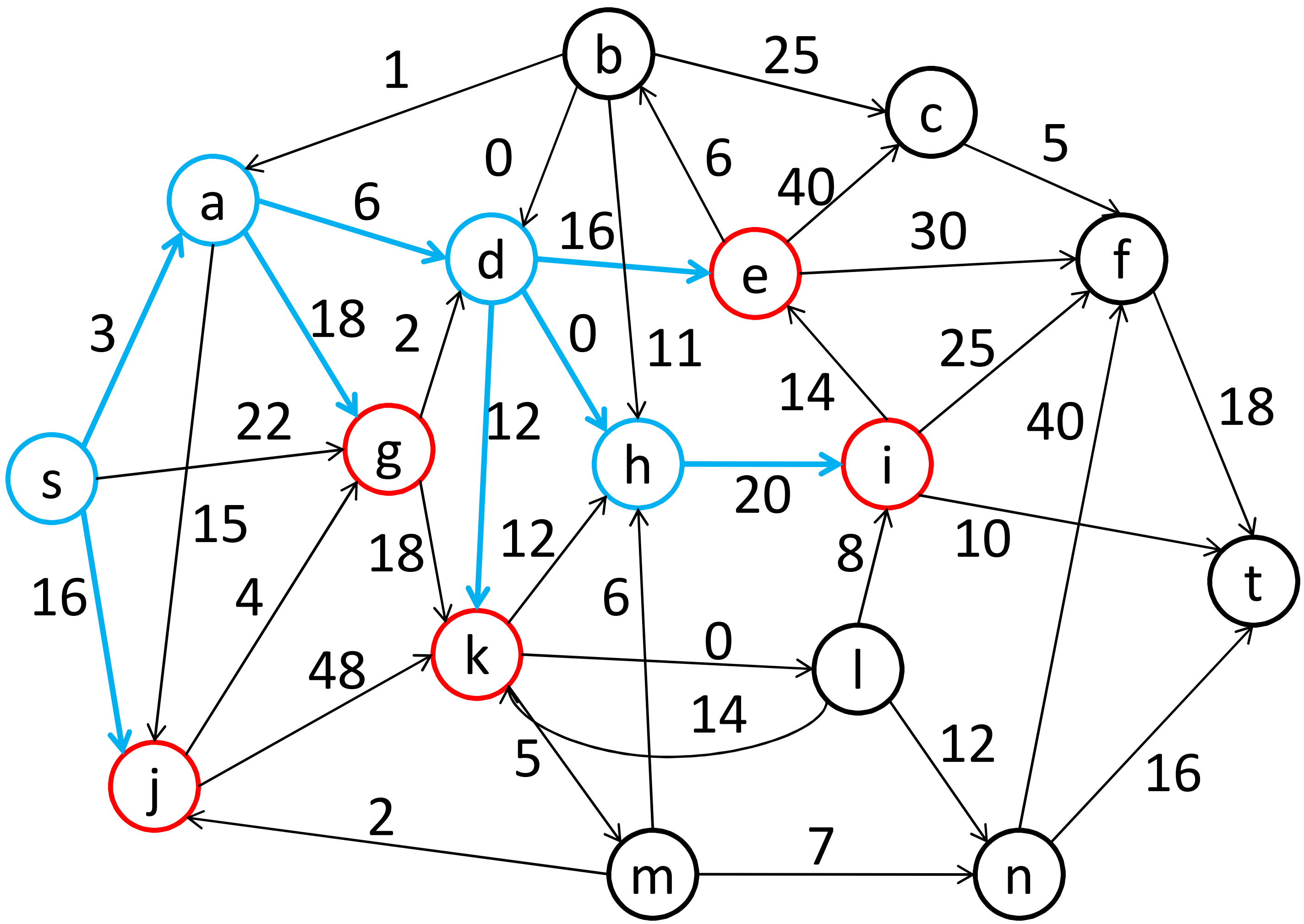
S = s:0, a:3, d:9 *scan* h

L = h:9, j:16, g:21, k:21, e:25



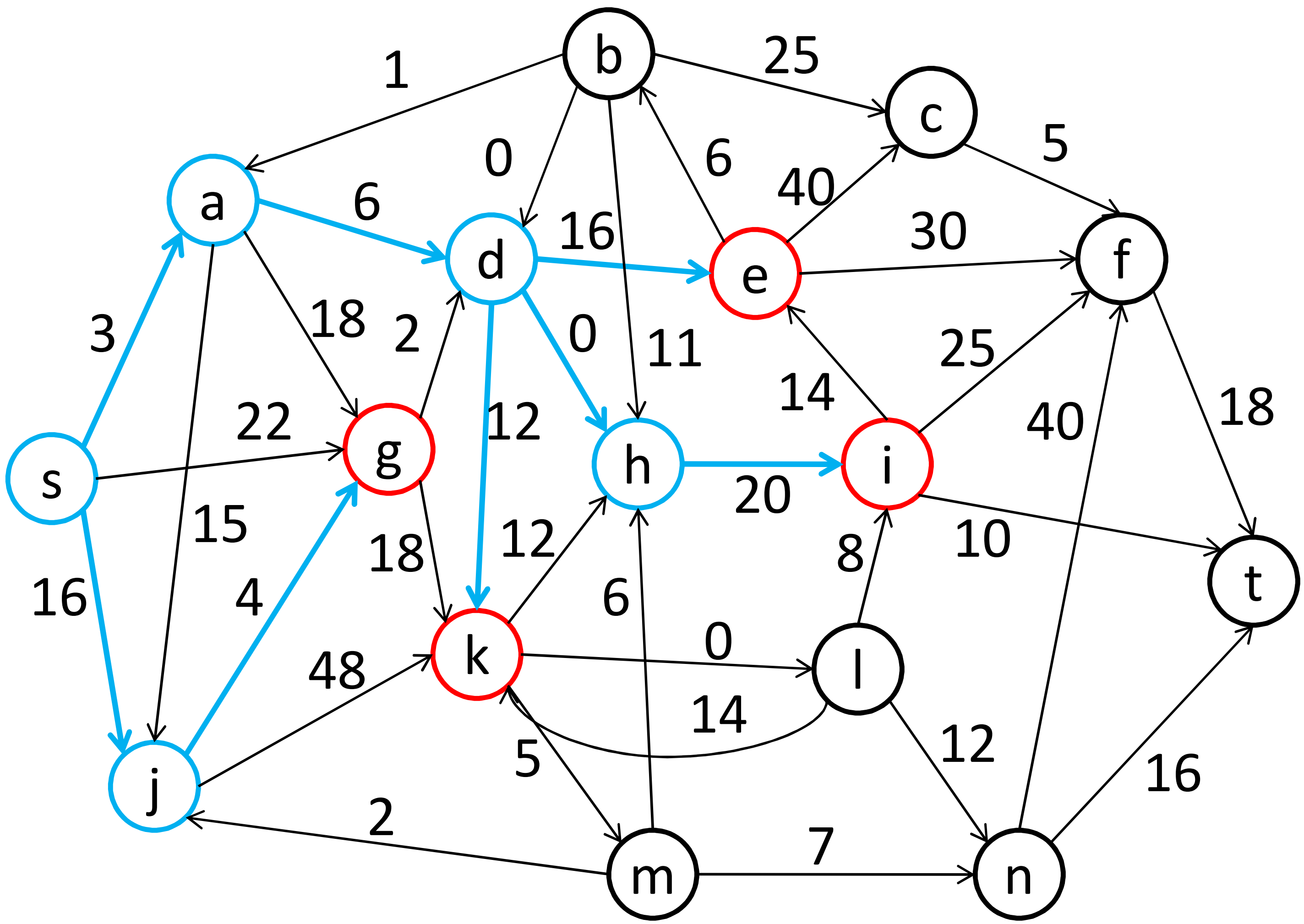
S = s:0, a:3, d:9, h:9 *scan* j

L = j:16, g:21, k:21, e:25, i:29



S = s:0, a:3, d:9, h:9, j:16 *scan* g...

L = g:20, k:21, e:25, i:29



Lemma 1: If arc lengths are non-negative, shortest-first scanning maintains the invariant that $d(x) \leq d(y)$ if x is scanned and y is labeled.

Proof: By induction on number of scans. True initially. Suppose true before scan of v. If $d(v) + c(v, w) < d(w)$, $d(v) < d(w)$ since $c(v, w) \geq 0$. By the choice of v to scan, w must be unlabeled or labeled. After $d(w)$ is decreased to $d(v) + c(v, w)$, $d(v) \leq d(w)$. Thus the scan preserves the invariant that if x is labeled, $d(v) \leq d(x)$, and the lemma remains true when v is moved to S.

Theorem 1: If arc lengths are non-negative, shortest-first scanning scans each vertex at most once.

Implementation: $L = heap$, key of $v = d(v)$

find v to scan: *delete-min*(L)

label(w): after decreasing $d(w)$,

if $w \in U$ **then** *insert*(w, L)

else ($w \in L$) *decrease-key*(w, $d(w)$, L)

$\le n$ *inserts*, $\le n$ *delete-mins*, $\le m$ *decrease-keys*

L = implicit heap or pairing heap: $O(m \lg n)$

L = rank-pairing heap: $O(m + n \lg n)$

No cycles

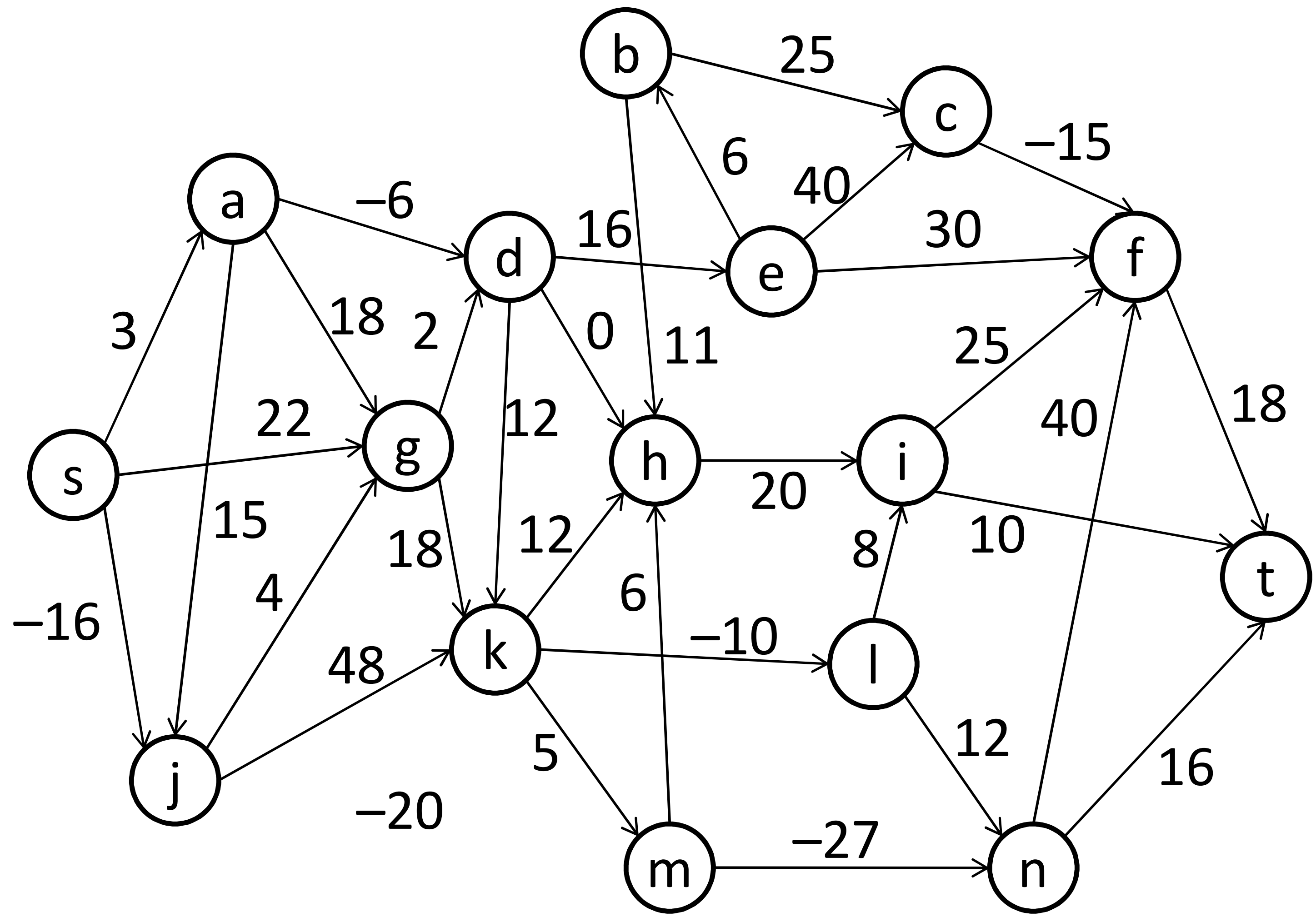
topological order of vertices: $(v, w) \in A$
$\rightarrow$ v before w

graph is acyclic $\leftrightarrow$ $\exists$ topological order

Can find a topological order or a cycle in $O(m)$ time: graph search, later lecture

Topological scanning: scan vertices in topological order: each vertex scanned once, O(m) time

Scan s, a, j, g, d, e, b, c, k, l, m, h, i, n, f, t



Arc-length transform

Let π be any real-valued function on vertices

Reduced length of (v, w):

$c_\pi(v, w) = c(v, w) + \pi(v) - \pi(w)$

If P is a path, $c_\pi(P)$ = sum of c_π on arcs of P

If P is from v to w, $c_\pi(P) = c(P) + \pi(v) - \pi(w)$

→ replacing c by c_π preserves shortest paths, although in general it changes their lengths

If C is a cycle, $c_\pi(C) = c(C)$

→ replacing c by c_π preserves cycle lengths

Goal: find a π that makes all arc lengths non-negative

Solution: Single-source shortest distances via breadth-first scanning

Create a dummy source s with arcs of length 0 to all other vertices (to guarantee reachability)

Find shortest distances from s to all vertices

If no negative cycle, let $\pi = d$:

$c_\pi(v, w) = c(v, w) + d(v) - d(w) \geq 0$

Special case of linear programming duality

All-pairs shortest paths

Transform arc lengths so non-negative via breadth-first scanning

For each source, solve a single-source problem with non-negative arc lengths via shortest-first scanning

Undo arc length transform

Time = $O(n(n\lg n + m))$

Alternative all-pairs algorithm (Floyd-Williams)

Dynamic programming:

For each vertex pair u, w maintain

$d(u, w)$ = length of shortest path from u to w found so far

Process vertices one-by-one. To process v, consider paths containing v.

for $u \in V$ **do for** $w \in V$ **do**

  if $u = w$ **then** $d(u, w) \leftarrow 0$

  else if $(u, w) \in A$ **then** $d(\mathrm{u}, w) \leftarrow c(u, w)$

  else $d(u, w) \leftarrow \infty$

for $v \in V$ **do for** $u \in V$ **do for** $w \in V$ **do**

  $d(u, w) = \min\{d(u, w), d(u, v) + d(v, w)\}$

When the algorithm stops, if $d(v, v) < 0$ for some v, there is a negative cycle containing v; otherwise, $d(u, w)$ is the length of a shortest path from u to w.

The algorithm is very simple but its running time is $\Theta(n^3)$: takes no advantage of sparsity (few arcs)

To exploit sparsity, only iterate over finite entries. Resembles *Gaussian elimination*; indeed, Gaussian elimination can be used to find shortest paths.

Single-pair shortest paths

Goal: find a shortest path from *s* to *t*

Assume no non-negative arc lengths

One-way search: Do shortest-first scanning forward from s until t is deleted from L, or do shortest-first scanning backward from t until s is deleted from L

Two-way search: Do shortest-first scanning forward from *s* and backward from *t* concurrently.

Stopping condition for two-way search

Let $d(v)$ and $d'(v)$ be computed distance of v from s and to t, respectively

Stop when some v has been deleted from both heaps (both $d(v)$ and $d'(v)$ are exact). The length of a shortest path from s to t is

$$\min\{d(x) + d'(x) \mid x \in V\}$$

(not necessarily $d(v) + d'(v)$)

Exercise: prove this

One-way heuristic search

Goal: avoid examining vertices of the graph other than those on a shortest path from s to t

Method: use an easy-to-compute estimate $e(v)$ of the distance from v to t to help guide shortest-first scanning from s: key of v in heap is $d(v) + e(v)$. ($e(v) = 0$ is Dijkstra's algorithm)

Normalization: $e(t) = 0$

Examples

15 puzzle: *e* = number of misplaced tiles

more accurate: *e* = sum of manhattan

distances of tiles to correct positions

Road network, lengths are distances

e = straight-line distance ("as the crow flies")

What must hold of e so that one scan per vertex suffices?

safety: $e(v) \leq c(v, w) + e(w)$ for $(v, w) \in A$

If e is safe and $e(t) = 0$, $e(v) \leq c(P)$ for any path P from v to t

Exercise: prove this

If e is safe, $d(v)$ when v is first scanned is the length of a shortest path from s to v

Exercise: prove this