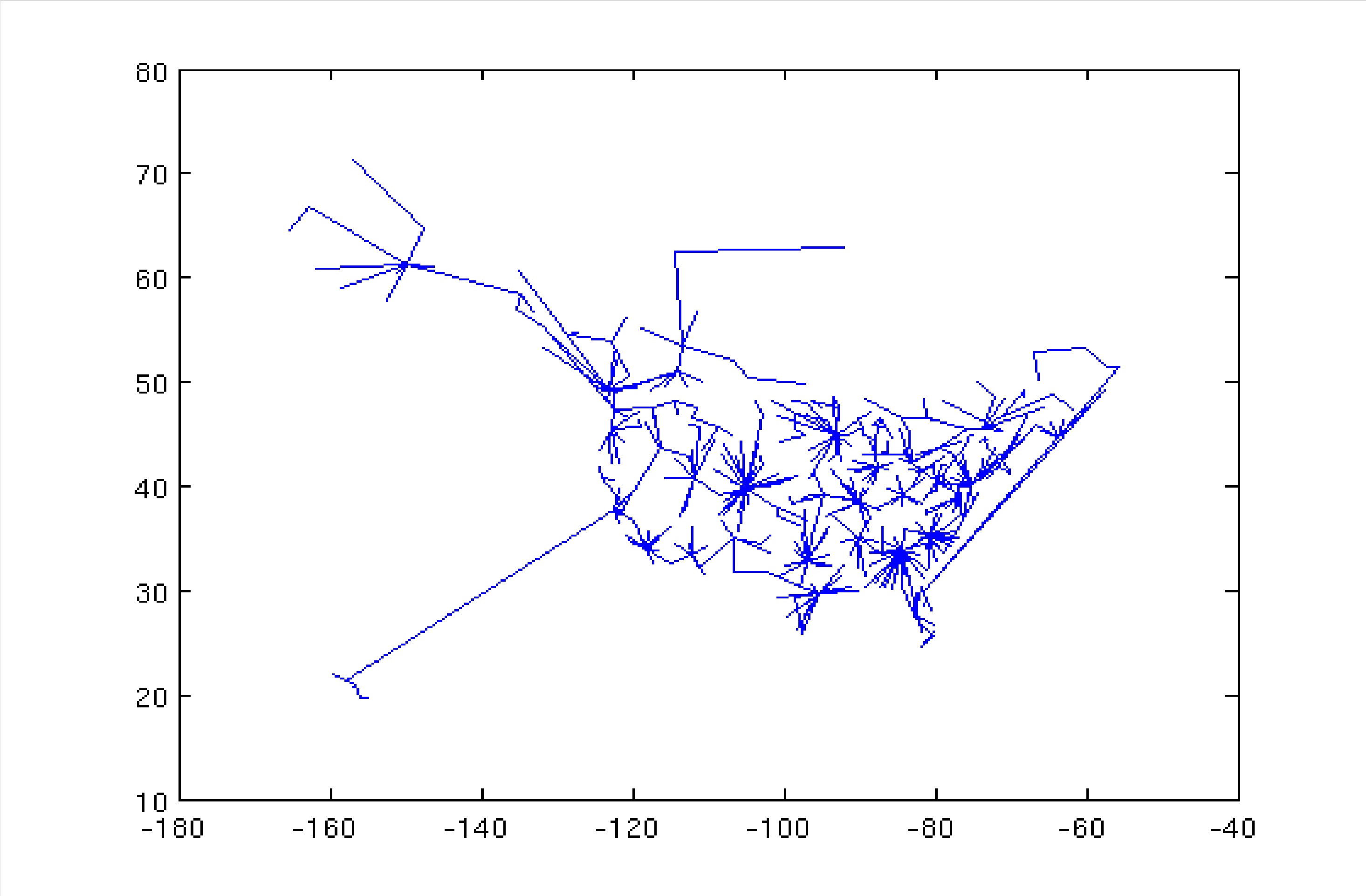
Minimum Spanning Tree (undirected graph)



Path tree vs. spanning tree

We have constructed trees in graphs for shortest path to anywhere else (from vertex is the root)

Minimum spanning trees instead want to connect every node with the least cost (undirected edges)

Path tree vs. spanning tree

Example: build the least costly road that allows cars to get from any start to any finish

Safe edges

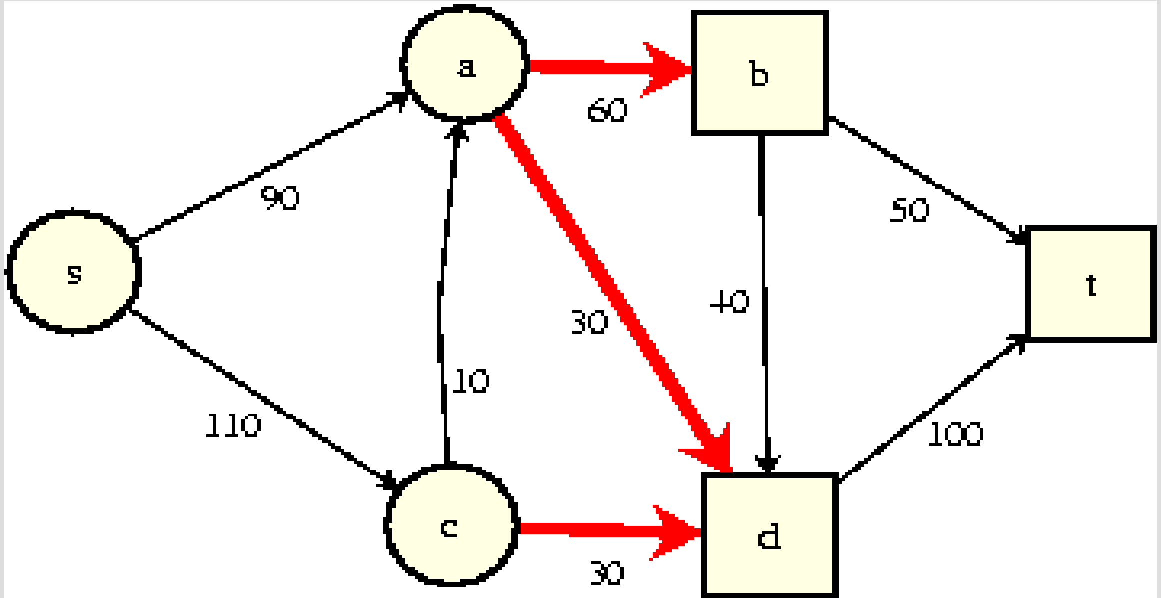
We an find (again) a greedy algorithm to solve MSTs

We can repeatedly add safe edges to an existing solution:

1. Find (u,v) as safe edge for *A*
2. Add (u,v) to *A* and repeat 1.

Safe edges

A <u>cut</u> S: (S, V-S) for any verticies S
Cut S <u>respects</u> *A*: no edge in *A* has one side in S and another in V-S

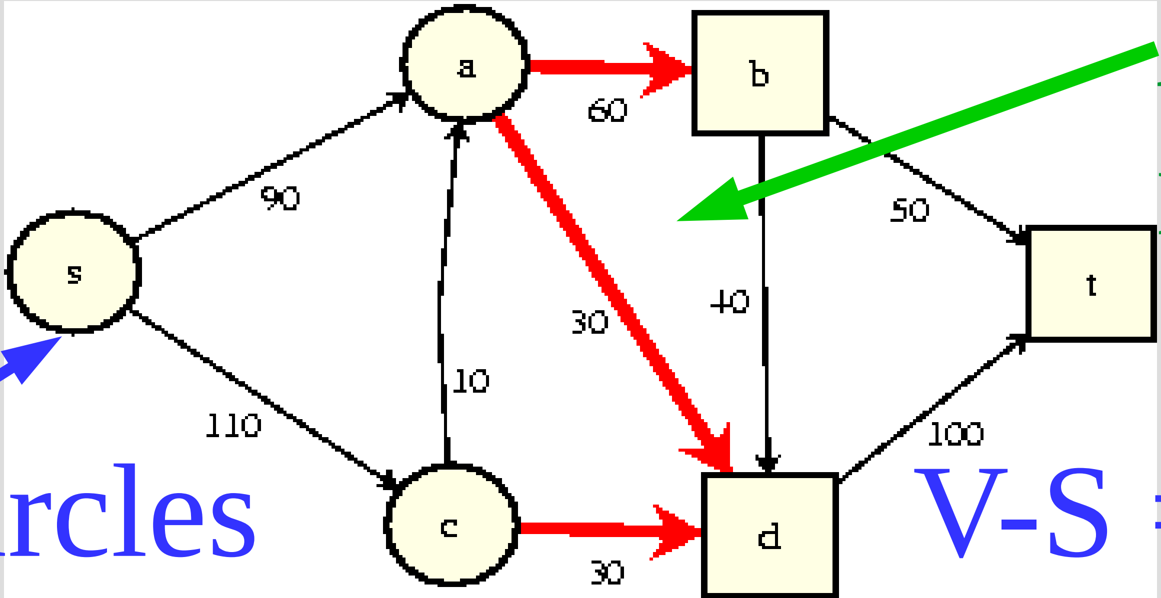


Safe edges

A <u>cut</u> S: (S, V-S) for any verticies S

Cut S <u>respects</u> A: no edge in A has one side in S and another in V-S

S respects A if no red edges



S = circles

V-S = squares

Safe edges

Theorem 23.1:
Let A be a set of edges that is included in some MST

Let S be a cut that respects A

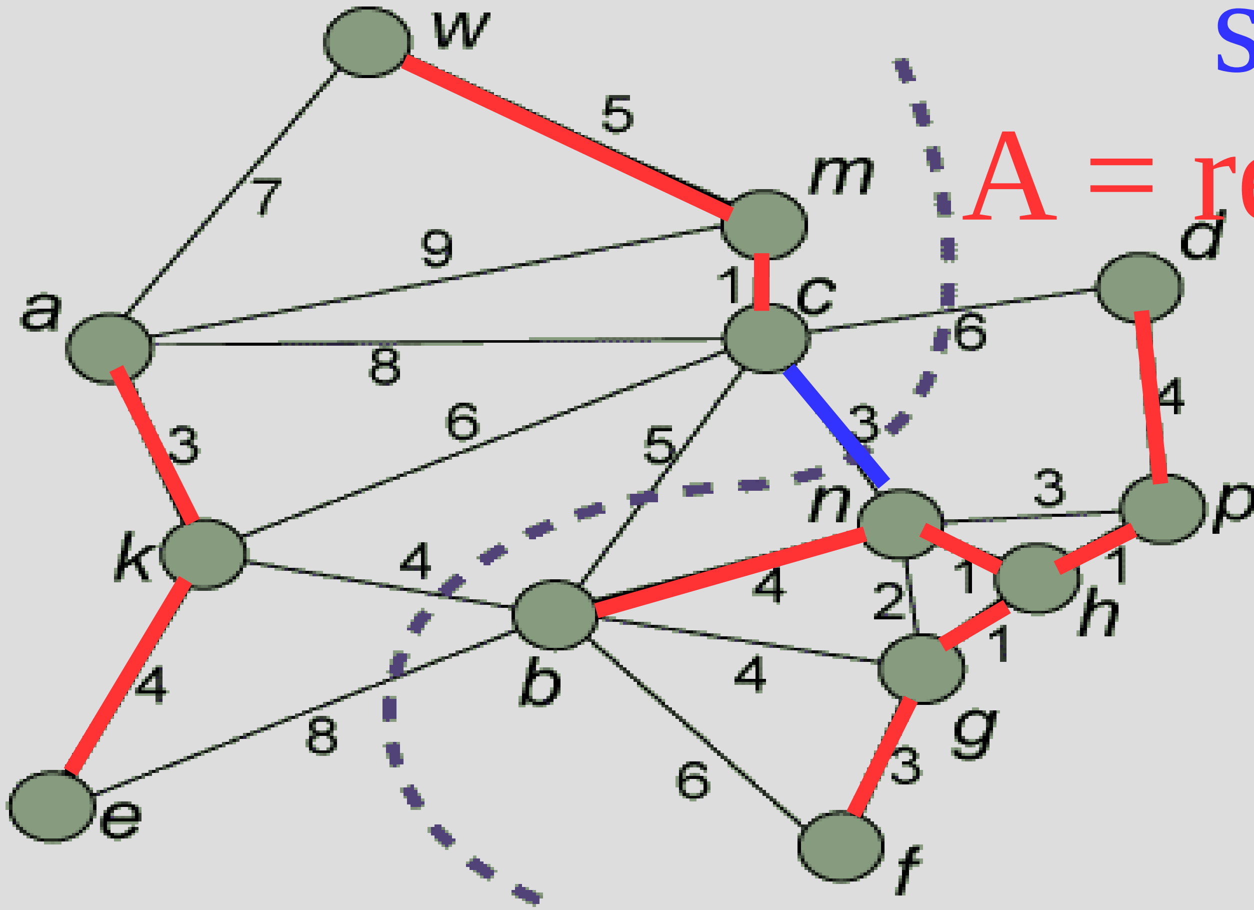
Then the minimum edge that crosses S and V-S is a safe edge for A

Safe edges

Theorem 23.1:

blue = minimum safe edge

A = red edges



LHS = S

RHS = V-S

Safe edges

Proof:

Let T be a MST that includes A

Add minimum safe edge (u,v)

Let (x,y) be the other edge on the cut

Remove (x,y), and call this T' thus:

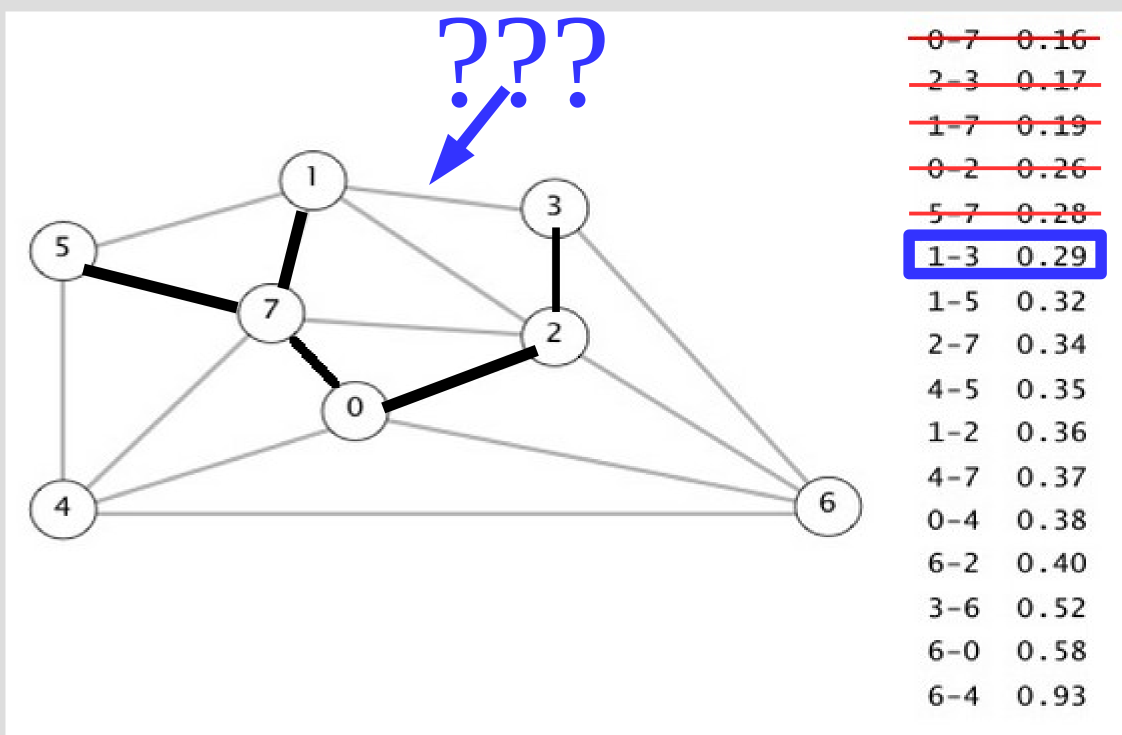
$w(T') = w(T) + w(u,v) - w(x,y)$

But (u,v) min, so $w(u,v) \leq w(x,y)$

Thus, $w(T') \leq w(T)$ and we done

Safe edges

No-cycle theorem: There is no cut through edge (u,v) that respects A if adding (u,v) creates a cycle



Safe edges

Proof: (contradiction)

Suppose cut exists (u in S, v in V-S)

Adding (u,v) creates a cycle

Thus A has path from u to v

Must exist some edge (x,y) with x in S and y in V-S

S cuts this edge and thus cannot respect A

Kruskal

Idea:

1. Sort all edges into a list

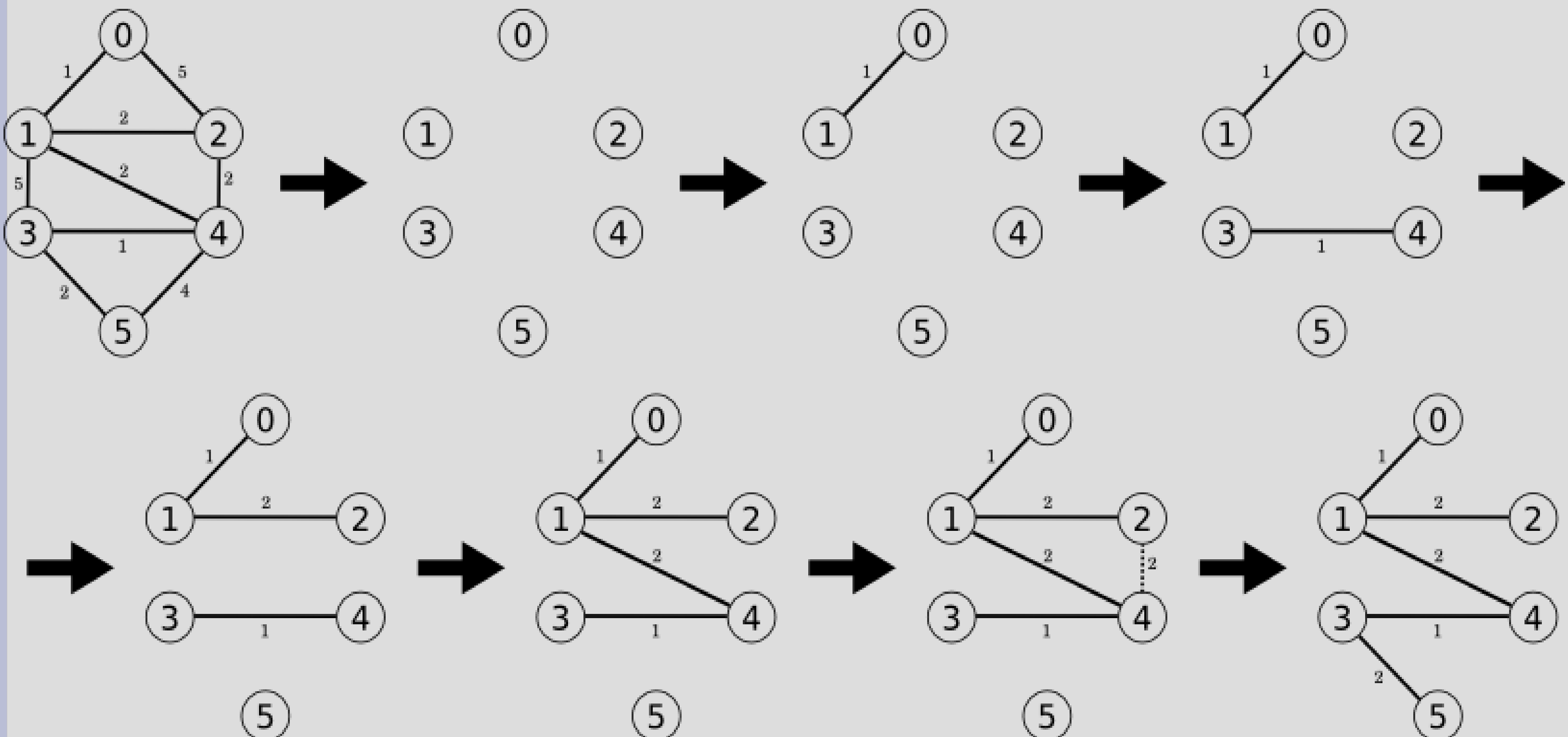
2. If the minimum edge in the list does not create a cycle, add it to A

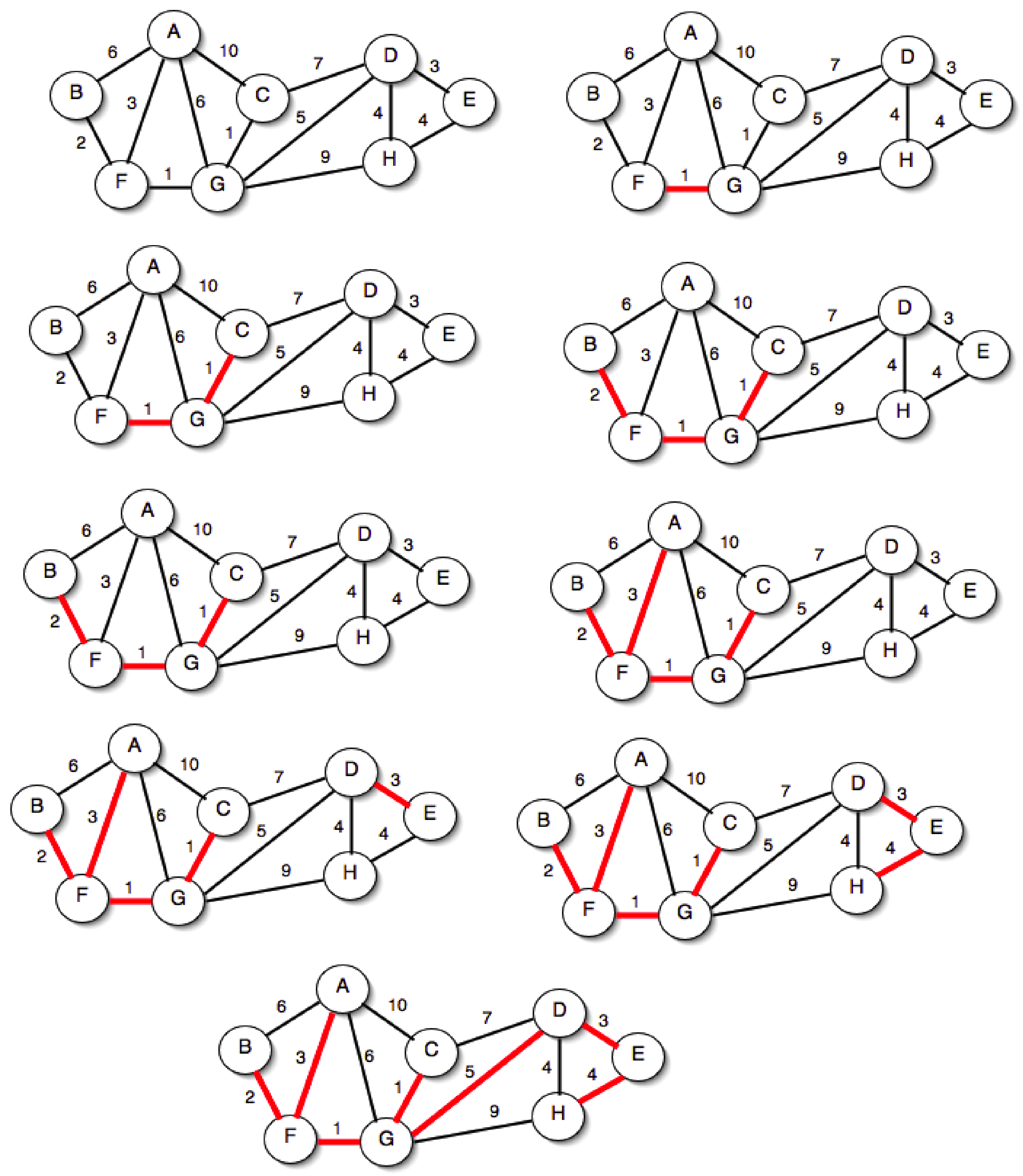
3. Remove the edge and repeat 2 until no more edges

Kruskal

```
MST-Kruskal(G,w)
A = { }
for each v in G.V: Make-Set(V)
sort(G.E)
for (u,v) in G.E (w(u,v) increasing)
   if Find-Set(u) ≠ Find-Set(v)
      A= A U {(u,v)}
      Union(u,v)
```

Kruskal




A B C D E F G H
6 10 7 3 3 6 1 5 4 4 2 1 9

Kruskal

Runtime:
Find-Set takes about $O(\lg |V|)$ time (Ch. 21)

Thus overall is about $O(|E| \lg |V|)$

Prim

Idea:

1. Select any vertex (as the root)
2. Find the shortest edge from a vertex in the tree to a vertex outside
3. Add this edge (and the connected vertex) to the tree
4. Goto 2.

Like Dijkstra, but different relaxation

Prim

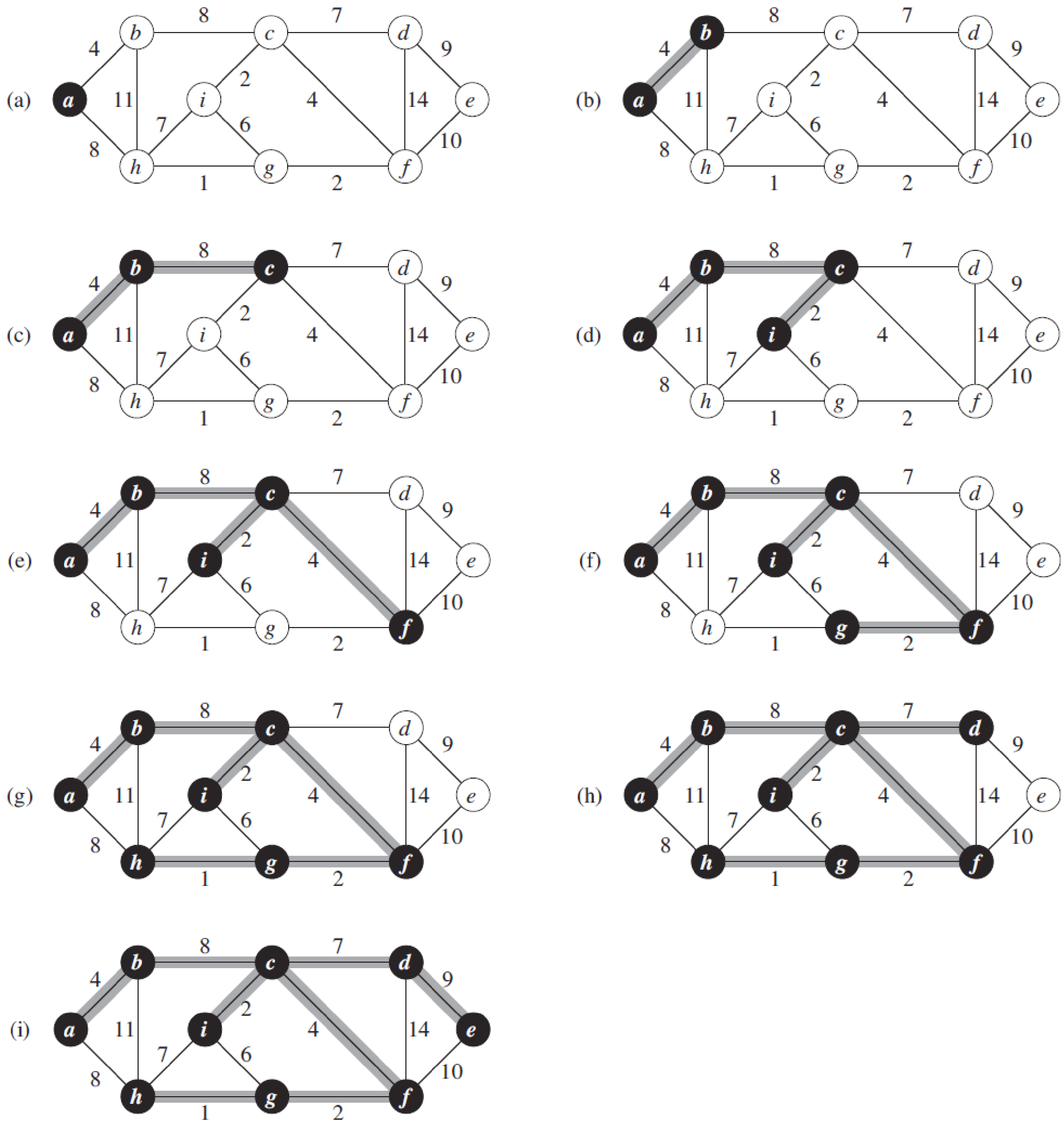
MST-Prim(G, w, r) // r is root
for each u in G.V: u.key=∞, u.π=NIL
r.key = 0, Q = G.V
while Q not empty
    u = Extract-Min(Q)
    for each v in G.Adj[u]
        if v in Q and <u>w(u,v) < v.key</u>
            v.key=w(u,v), v.π=u

modified "relax"
from Dijkstra

Prim

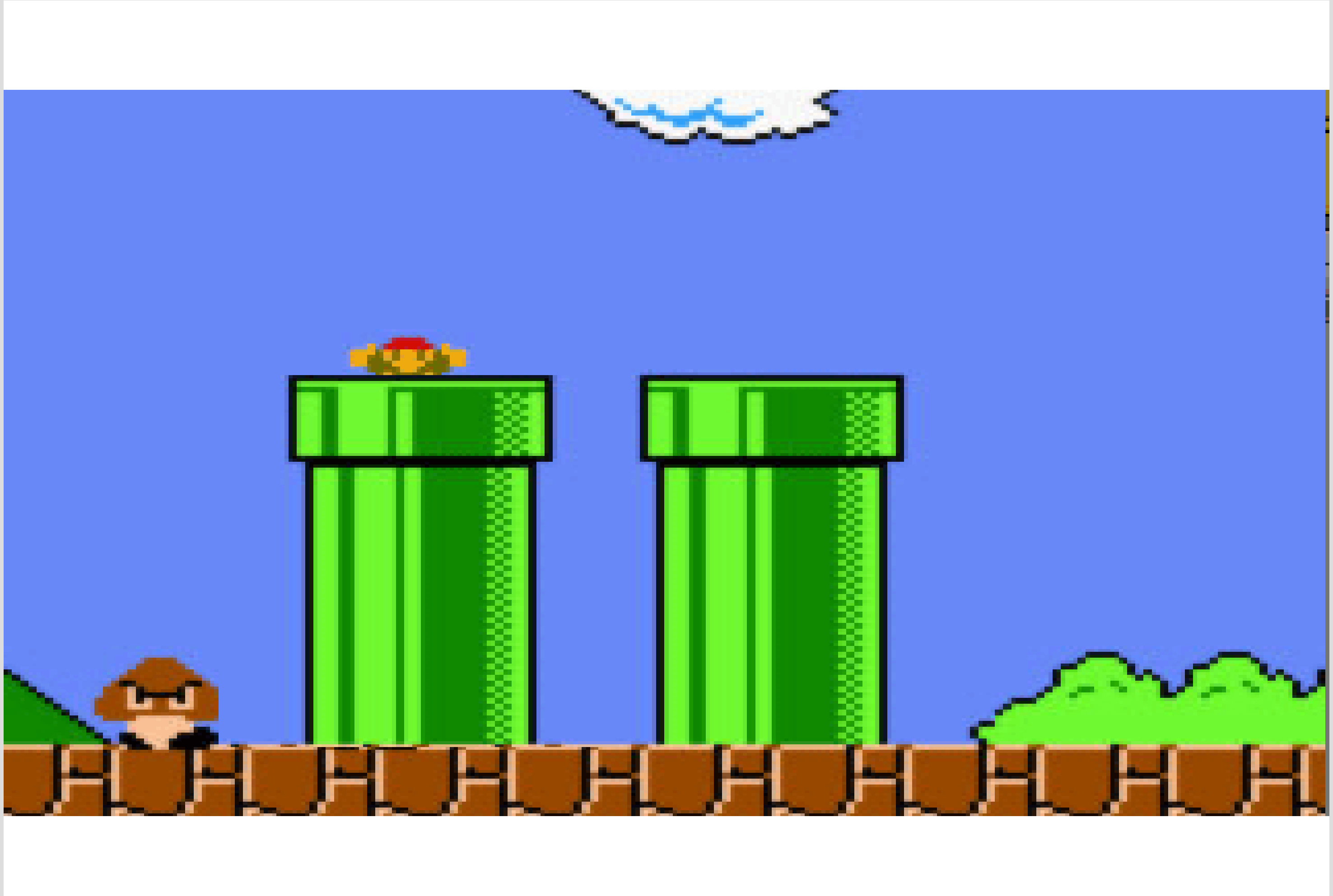
Runtime:
Extract-Min(V) is O(lg |V|), run |V| times is O(|V| lg |V|)

for loop runs over each edge twice, minimizing (i.e. Decrease-Key())...
O((|V|+|E|) lg |V|) = O(|E| lg |V|)
(Fibonacci heaps O(|E| + |V| lg |V|))



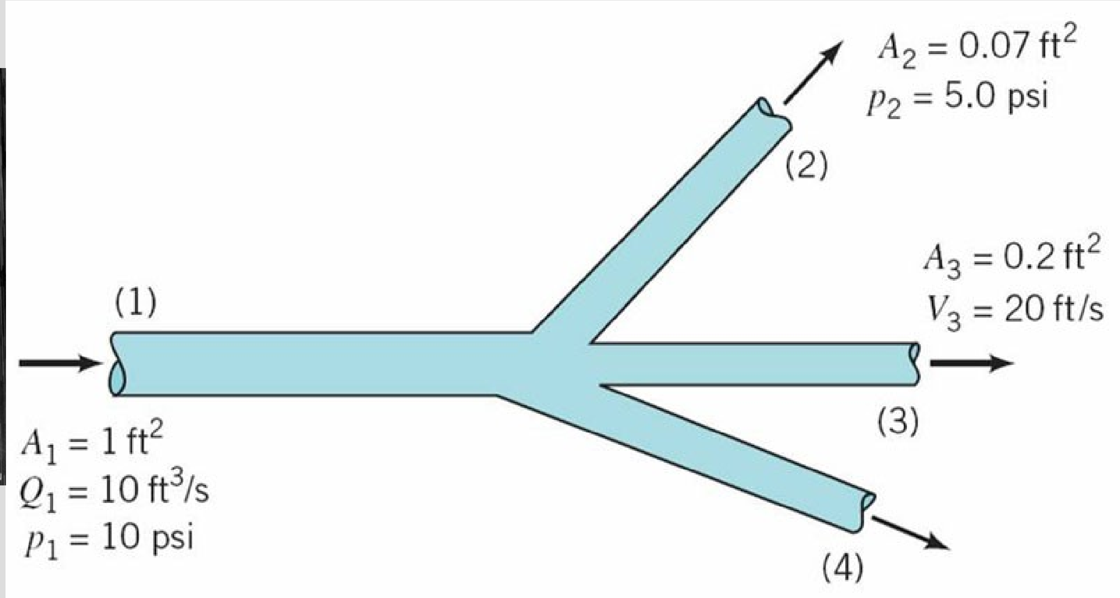
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2

Network Flow



Network Flow terminology

Network flow is similar to finding how much water we can bring from a “source” to a “sink” (infinite) (intermediates cannot “hold” water)



Network Flow terminology

Definitions:

$c(u,v)$: edge <u>capacity</u>, $c(u,v) \geq 0$

$f(u,v)$: flow from u to v s.t.

1. $0 \leq f(u,v) \leq c(u,v)$
2. $\sum_v f(u,v) = \sum_v f(v,u)$

s : a <u>source</u>, $\sum_v f(s,v) \geq \sum_v f(v,s)$

t : a <u>sink</u>, $\sum_v f(t,v) \leq \sum_v f(v,t)$

Network Flow terminology

Definitions (part 2):

$|f| = \sum_v f(s,v) - \sum_v f(v,s)$

^ amount of flow from source

Want to maximize $|f|$ for the <u>maximum-flow problem</u>

Network Flow terminology

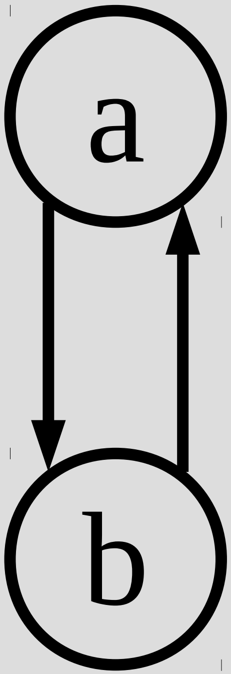
Graph restrictions:

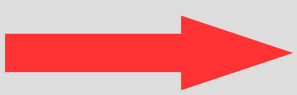
1. If there is an edge (u,v), then there cannot be edge (v,u)
2. Every edge is on a path from source to sink
3. One sink and one source

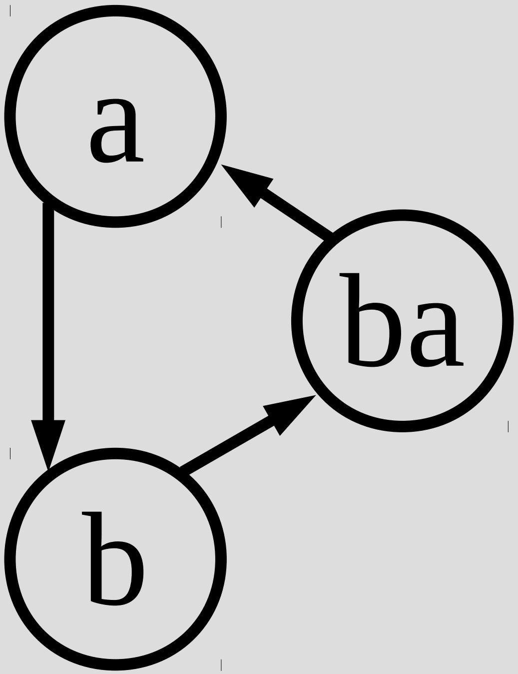
(None are really restrictions)

Network Flow terminology

1. If there is an edge (u,v), then there cannot be edge (v,u)





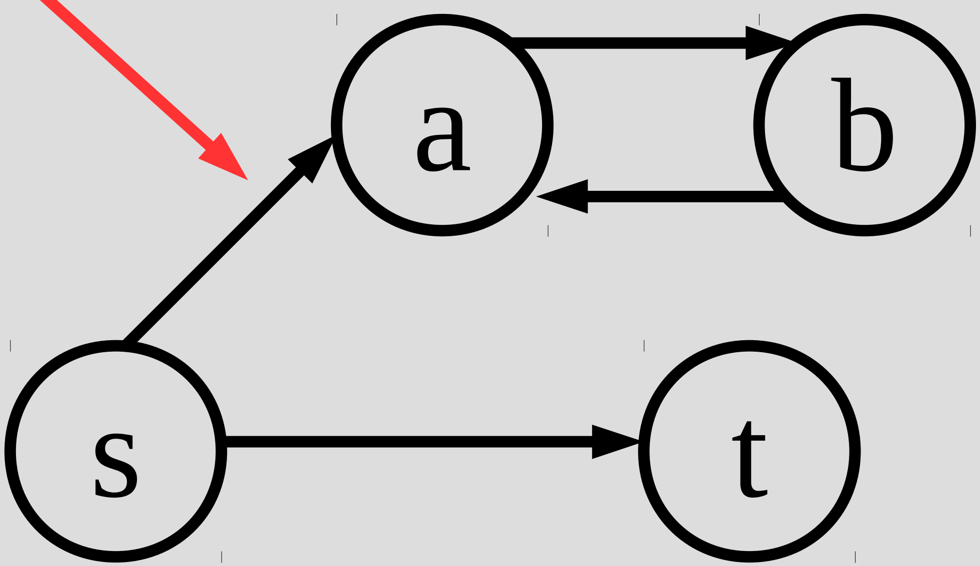


Network Flow terminology

2. Every edge is on a path from source to sink

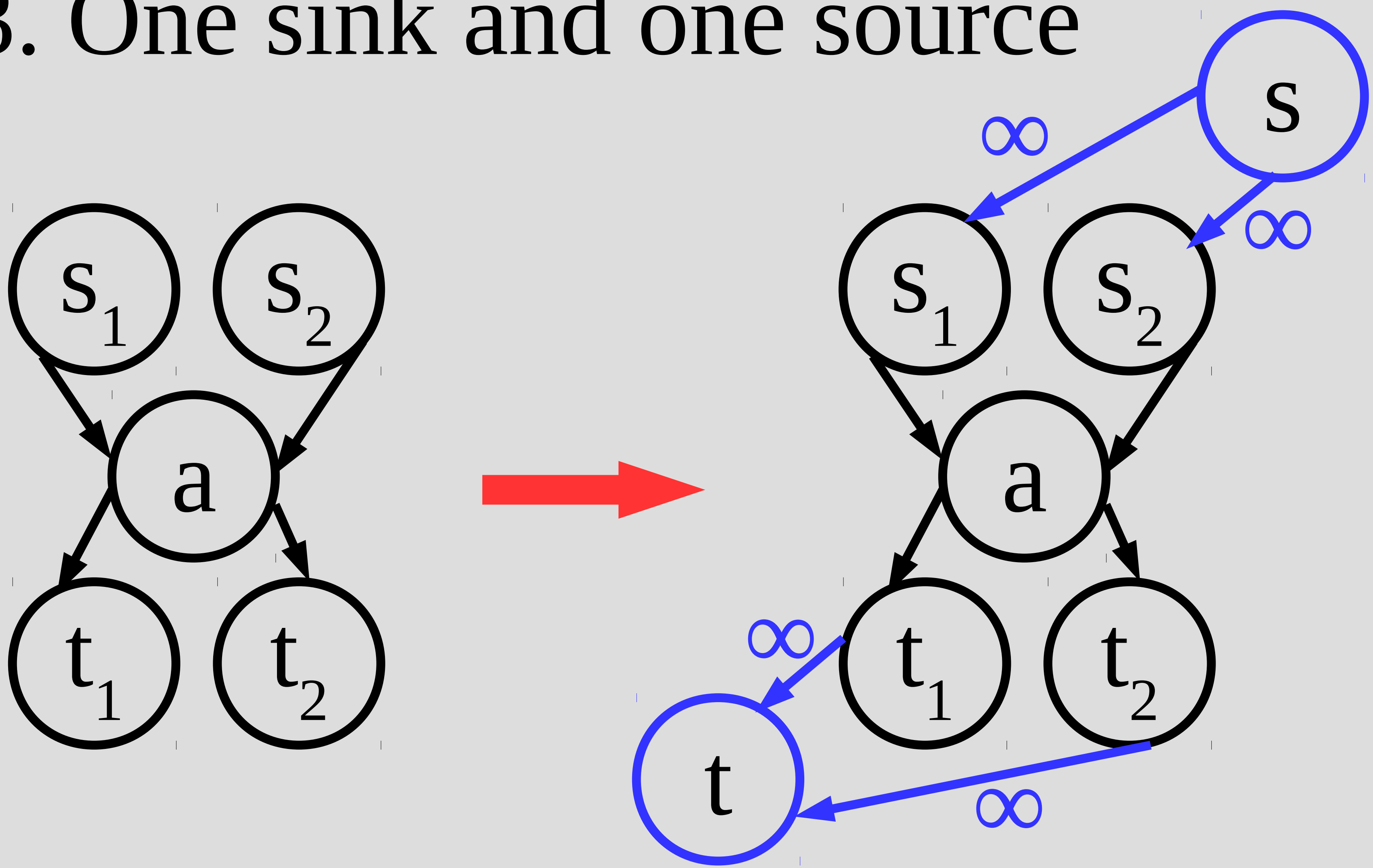
flow in = flow out,
only possible
flow in is 0

(worthless
edge)



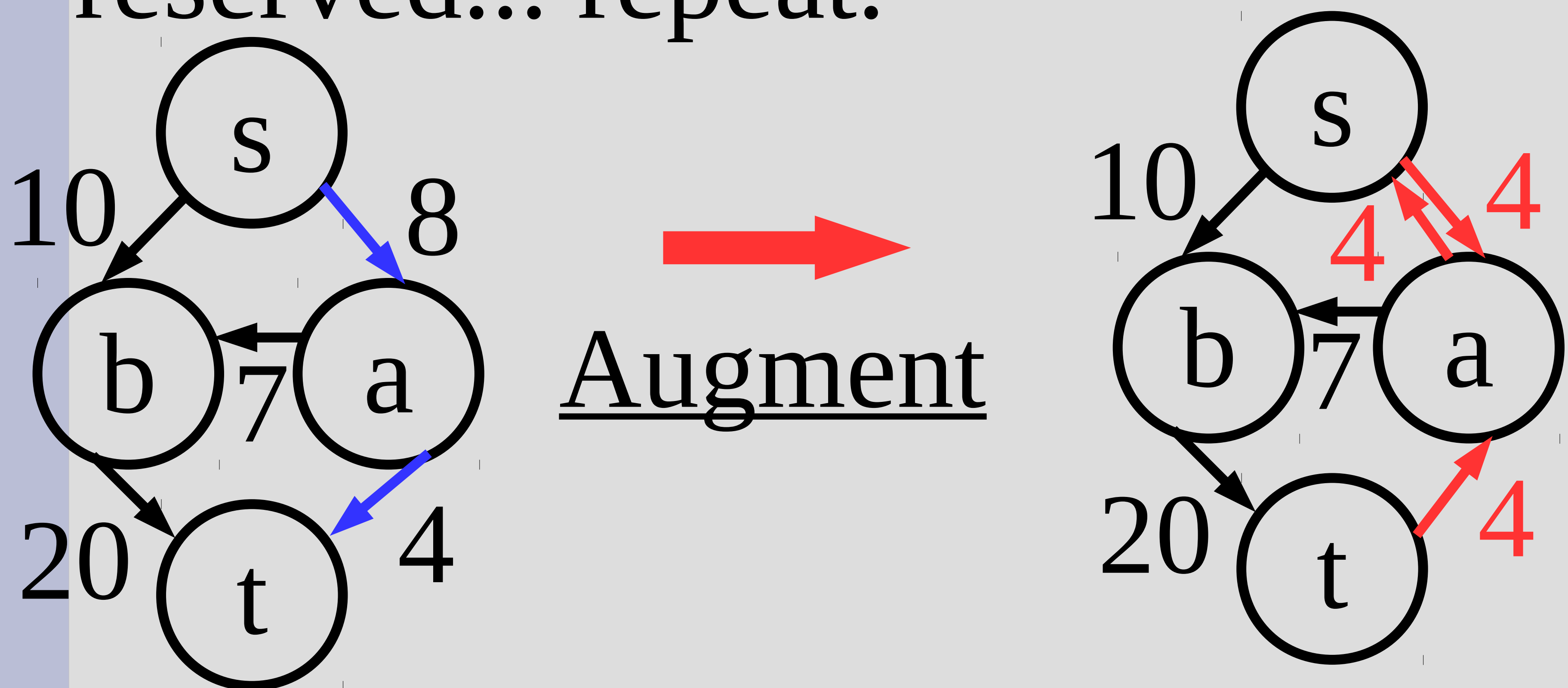
Network Flow terminology

3. One sink and one source



Ford-Fulkerson

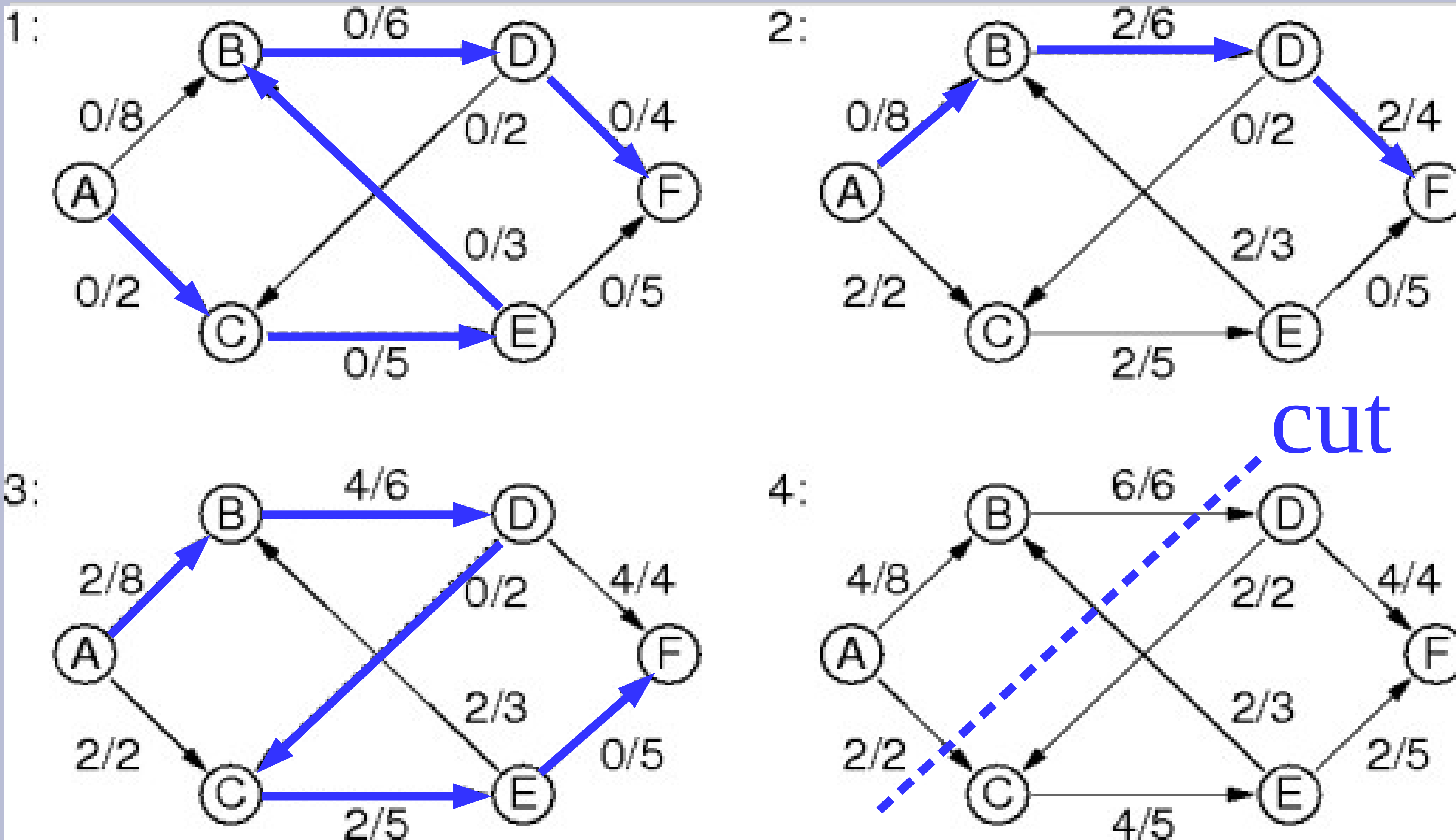
Idea: Find a way to add some flow, modify graph to show this flow reserved... repeat.



Ford-Fulkerson

```
Ford-Fulkerson(G, s, t)
initialize network flow to 0
while (exists path from s to t)
   augment flow, f, in G along path
return f
```

Ford-Fulkerson



Ford-Fulkerson

Subscript “f” denotes residual (or modified graph)

G_f = residual graph

E_f = residual edges

c_f = residual capacity

$c_f(u,v) = c(u,v) - f(u,v)$

$c_f(v,u) = f(v,u)$

Ford-Fulkerson

$(f \uparrow f')(u,v)$ = flow f augmented by f'

$(f \uparrow f')(u,v) = f(u,v) + f'(u,v) - f'(v,u)$

Lemma 26.1: Let f be the flow in G, and f' be a flow in G_f, then $(f \uparrow f')$ is a flow in G with total amount:

$|f \uparrow f'| = |f| + |f'|$

Proof: pages 718-719

Ford-Fulkerson

For some path p:

$c_f(p) = \min(c_f(u,v) : (u,v) \text{ on } p)$

^^ (capacity of path is smallest edge)

Claim 26.3:

Let $f_p = f_p(u,v) = c_f(p)$, then

$|f \uparrow f_p| = |f| + |f_p|$

Ford-Fulkerson

```
Ford-Fulkerson(G, s, t)
for: each edge (u,v) in G.E: (u,v).f=0
while: exists path from s to t in G_f
  find c_f(p) // minimum edge cap.
  for: each edge (u,v) in p
    if(u,v) in E: (u,v).f=(u,v).f + c_f(p)
    else: (u,v).f=(u,v).f - c_f(p)
```

Ford-Fulkerson

Runtime:

How hard is it to find a path?

How many possible paths could you find?

Ford-Fulkerson

Runtime:

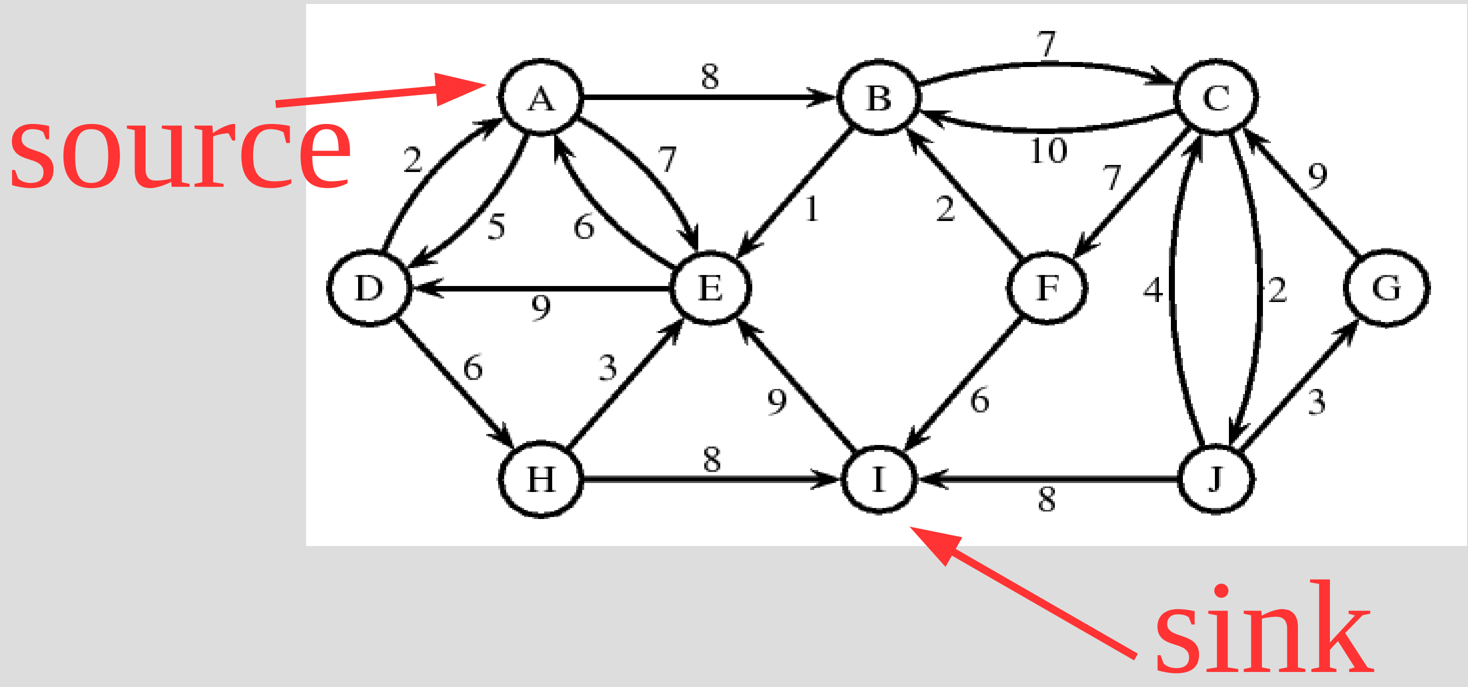
How hard is it to find a path?
-O(E) (via BFS or DFS)
How many possible paths could you find?
- |f*| (paths might use only 1 flow)
.... so, O(E |f*|)

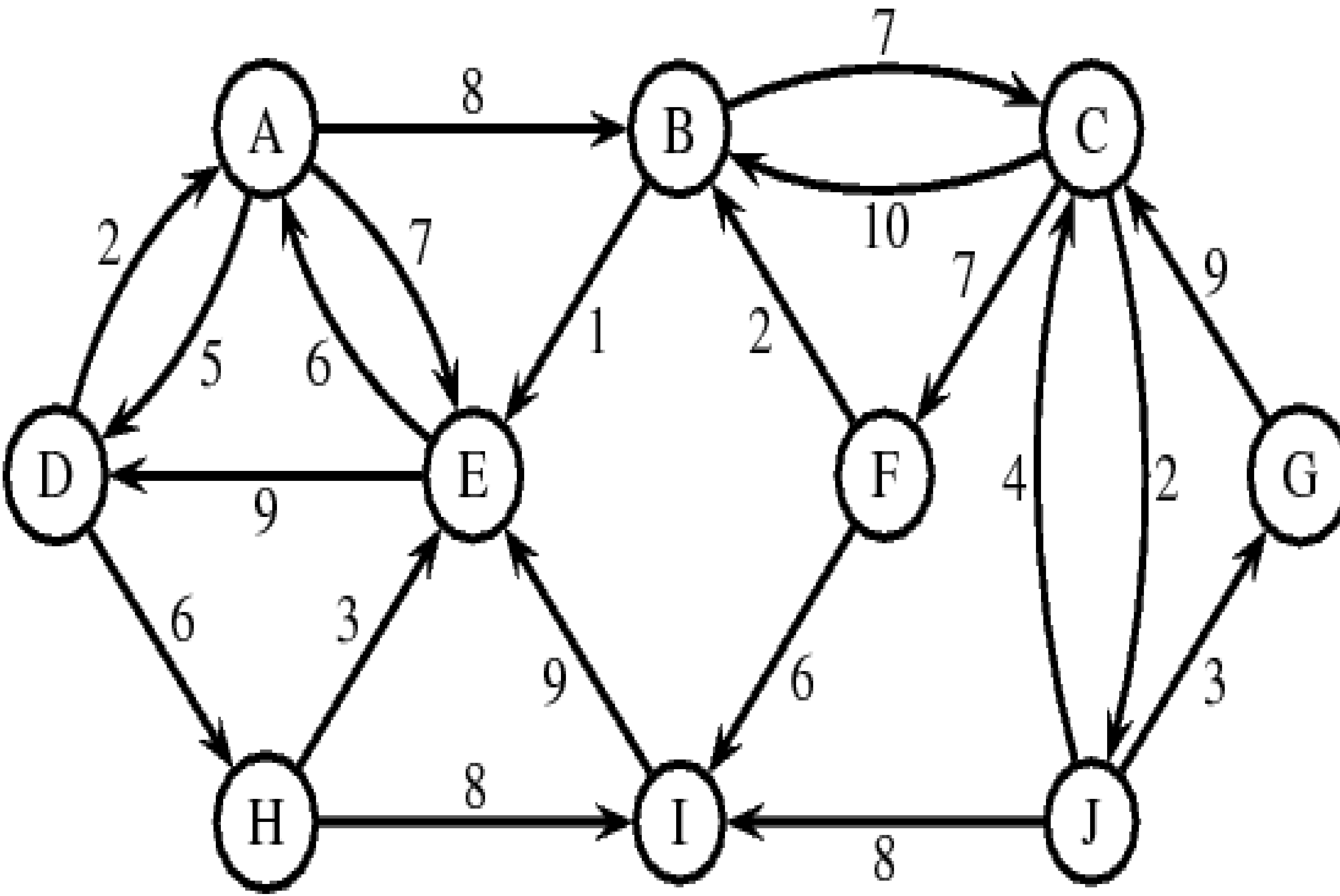
Max flow, min cut

Relationship between capacity and flows?

$c(S,T) = \sum_{u \text{ in } S}\sum_{v \text{ in } T} c(u,v)$

$f(S,T) = \sum_{u \text{ in } S}\sum_{v \text{ in } T} f(u,v) - \sum_{u}\sum_{v} f(v,u)$



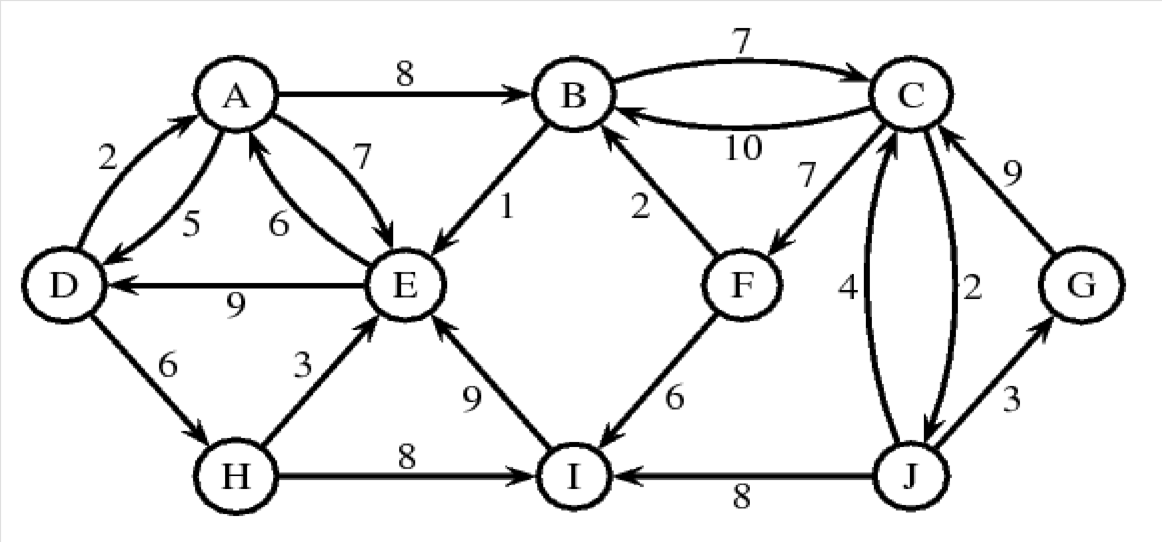

A
B
C
D
E
F
G
H
I
J
8
7
10
2
5
6
7
1
2
7
9
9
4
2
6
3
9
6
3
8
8

Max flow, min cut

Relationship between capacity and flows?

$c(S,T) = \sum_{u \text{ in } S}\sum_{v \text{ in } T} c(u,v)$

$f(S,T) = \sum_{u \text{ in } S}\sum_{v \text{ in } T} f(u,v)-\sum_{u}\sum_{v} f(v,u)$



cut capacity $\geq$ flows across cut

Max flow, min cut

Lemma 26.4

Let (S,T) be any cut, then f(S,T) = |f|

Proof:

Page 722

(Again, kinda long)

Max flow, min cut

Corollary 26.5

Flow is not larger than cut capacity

Proof:

$$
\begin{aligned}
|f| &= \sum_{u \text{ in } S}\sum_{v \text{ in } T} f(u,v)-\sum_{u}\sum_{v} f(v,u) \\
&\leq \sum_{u \text{ in } S}\sum_{v \text{ in } T} f(u,v) \\
&\leq \sum_{u \text{ in } S}\sum_{v \text{ in } T} c(u,v) \\
&= c(S,T)
\end{aligned}
$$

Max flow, min cut

Theorem 26.5

All 3 are equivalent:

1. f is a max flow
2. Residual network has no aug. path
3. $|f| = c(S,T)$ for some cut (S,T)

Proof:

Will show: 1 => 2, 2=>3, 3=>1

Max flow, min cut

f is a max flow => Residual network has no augmenting path

Proof:

Assume there is a path p

$|f \uparrow f_p| = |f| + |f_p| > |f|$, which is a contradiction to $|f|$ being a max flow

Max flow, min cut

Residual network has no aug. path =>
|f| = c(S,T) for some cut (S,T)

Proof:

Let S = all vertices reachable from s in G_f

u in S, v in T => f(u,v) = c(u,v) else there would be path in G_f

Max flow, min cut

Also, $f(v,u) = 0$ else $c_f(u,v) > 0$ and again v would be reachable from s

$$
\begin{aligned}
f(S,T) &= \sum_{u \text{ in } S}\sum_{v \text{ in } T} f(u,v) - \sum_u\sum_v f(v,u) \\
&= \sum_{u \text{ in } S}\sum_{v \text{ in } T} c(u,v) - \sum_u\sum_v 0 \\
&= c(S,T)
\end{aligned}
$$

Max flow, min cut

$|f| = c(S,T)$ for some cut (S,T)
$\Rightarrow$ f is a max flow

Proof:
$|f| \leq c(S,T)$ for all cuts (S,T)

Thus trivially true, as $|f|$ cannot get larger than $C(S,T)$

Edmonds-Karp

exists shortest path (BFS)

~~Ford-Fulkerson~~(G, s, t)

for: each edge (u,v) in G.E: (u,v).f=0

while: ~~exists path from s to t in G_f~~

find $c_f(p)$ // minimum edge cap.

for: each edge (u,v) in p

if(u,v) in E: $(u,v).f=(u,v).f + c_f(p)$

else: $(u,v).f=(u,v).f - c_f(p)$

Edmonds-Karp

Lemma 26.7

Shortest path in G_f is non-decreasing

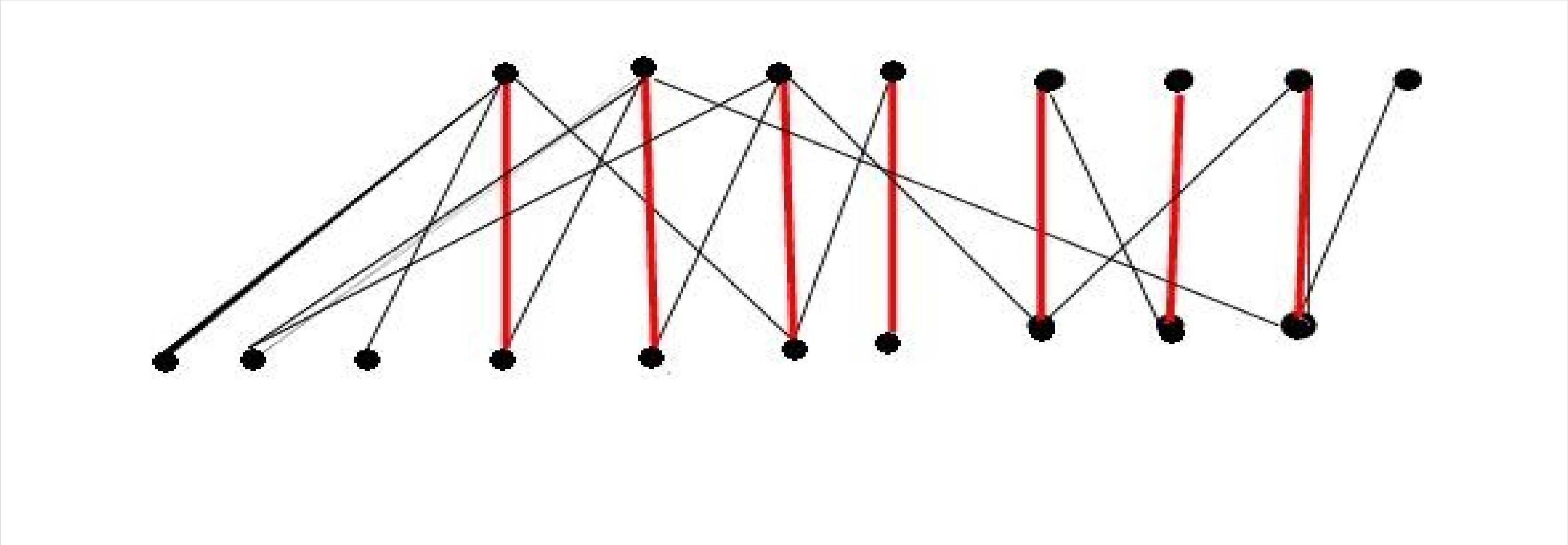
Theorem 26.8

Number of flow augmentations by Edmonds-Karp is $O(|V||E|)$

So, total running time: $O(|V||E|^2)$

Matching

Another application of network flow is maximizing (number of)matchings in a bipartite graph



Each node cannot be “used” twice

Matching

Add “super sink” and “super source”
(and direct edges source -> sink)
capacity = 1 on all edges