COMP251: Mid-Term Review

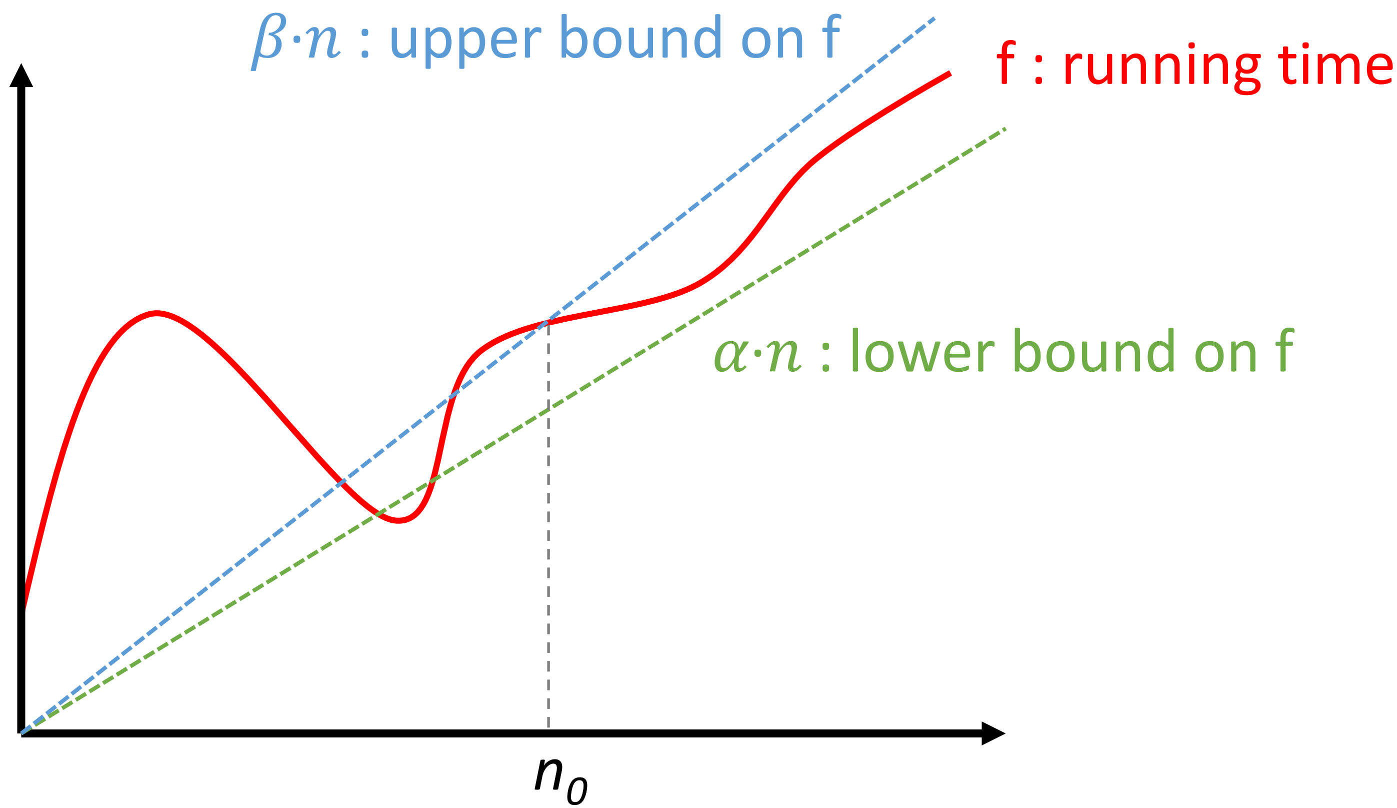
Jérôme Waldispühl
School of Computer Science
McGill University

Overview

Lecture 2	Hashing
Lecture 3	Heaps & Heapsort
Lecture 4	BST and AVL trees
Lecture 5	Red-black trees
Lecture 6	Disjoint sets
Lecture 7	Greedy algorithms (Scheduling, Huffman coding)
Lecture 8	Elementary graph algorithms
Lecture 9	Topological sort and strongly connected components
Lecture 10	Minimum Spanning Tree
Lecture 11	Single source shortest path
Lecture 12	Bipartite graphs
Lecture 13	Network flow 1
Lecture 14	Network flow 2

Techniques

Running time



- Running time is O (n)
- Running time is Ω(n)

$\Longrightarrow$Running time is Θ(n)

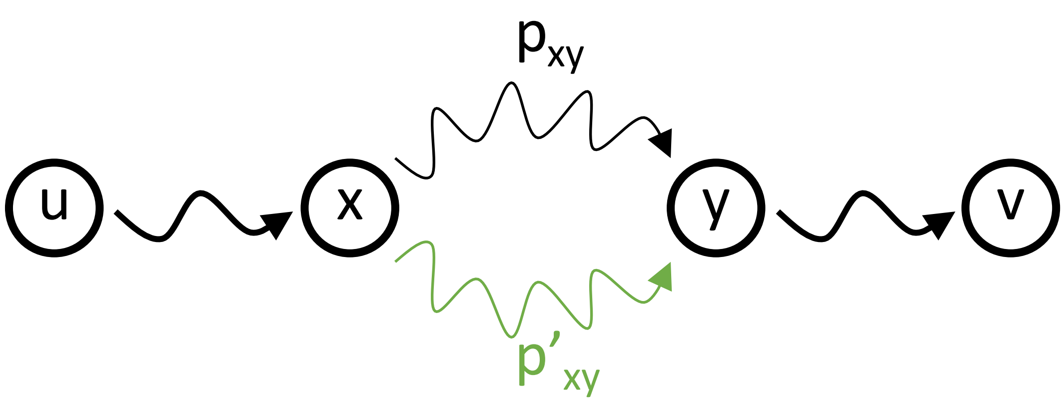
Proofs

- **Contradiction:** Given a proposition, assume opposite proposition is true, and then shows that it leads to a contradiction.
- **Cut and paste:** Used with graphs and greedy algorithms. Often used to prove an optimal solution of a problem is build from optimal solution of sub-problem (Optimal substructure). Assume a sub-problem is not optimal, and replace with optimal solution to show a contradiction.
- **Loop invariants:** Used to prove that a loop structure is doing what it is intended to do. You must specify:
  - Loop invariant property
  - Initialization
  - Maintenance
  - Termination

Optimal substructure

> ***Lemma***
> Any subpath of a shortest path is a shortest path.

Proof:



Suppose this path p is a shortest path from u to v.

Then $\delta(u,v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.

Now suppose there exists a shorter path $x \overset{p'_{xy}}{\rightsquigarrow} y$.

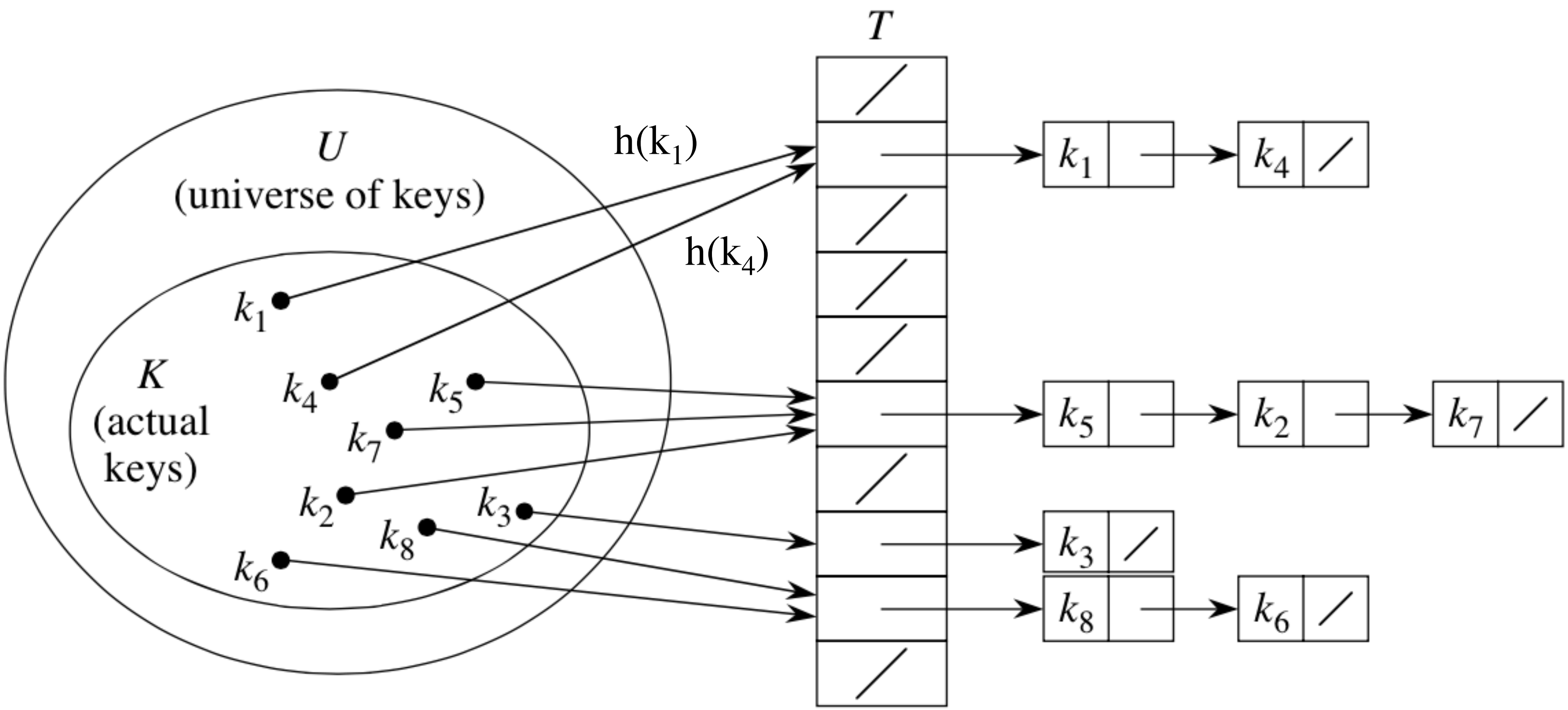
Then $w(p'_{xy}) < w(p_{xy})$. $w(p_{ux}) + w(p'_{xy}) + w(p_{yv}) < w(p_{ux}) + w(p_{xy}) + w(p_{yv})$

Contradiction of the hypothesis that p is the shortest path!

Hashing

Resolution by chaining

- Insertion time in $O(1)$ if we insert at the head of the list.
- Search time in $O(1)$ time in average, but not the *worst case*.



Hash function: $h : U \rightarrow \{0,1,\ldots,m-1\}$

Open addressing

Illustration: Where to store key 282?

Full!

h(282,0)=3

h(282,1)=1

h(282,2)=5

index	key
1	355
2	
3	567
4	233
5	282
6	799
7	

Note: Search must use the same probe sequence.

Linear & Quadratic probing

Linear probing:

$$h(k,i) = \left(h'(k) + i\right) \bmod m$$

Note: tendency to create clusters.

Quadratic probing:

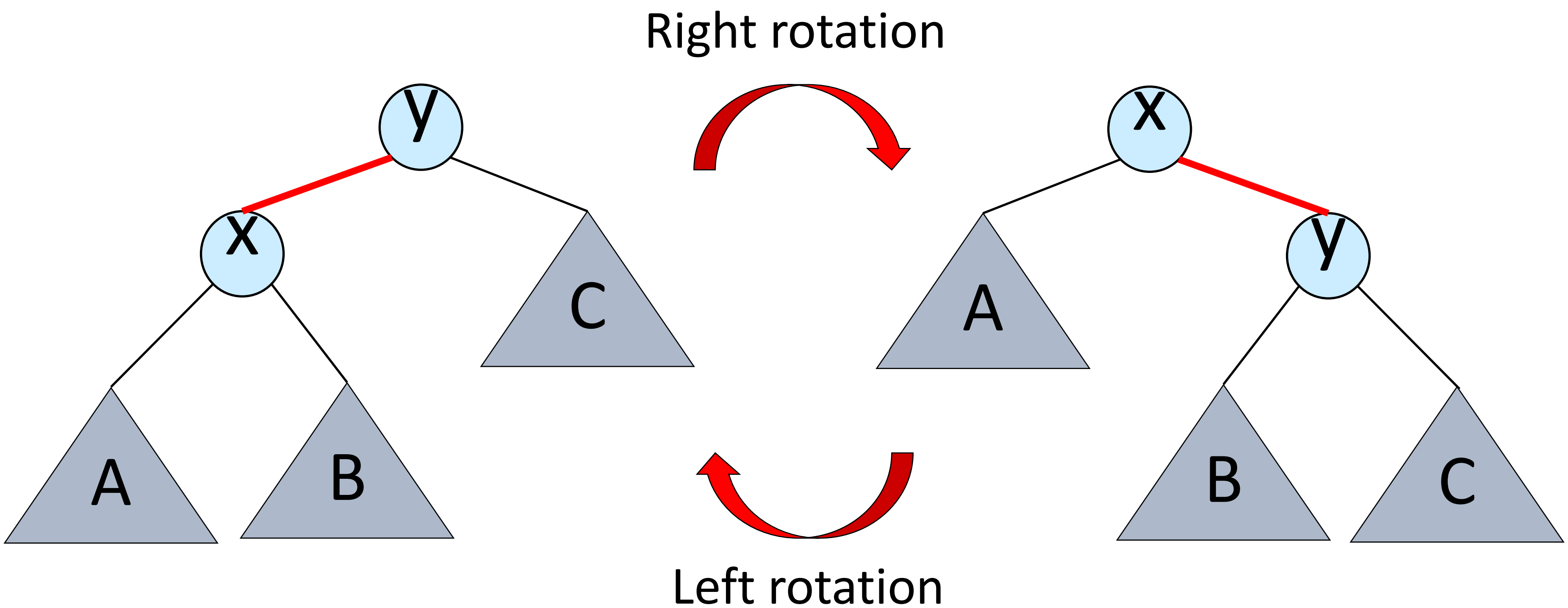
$$h(k,i) = \left(h'(k) + c_1 \cdot i + c_2 \cdot i^2\right) \bmod m$$

Remarks:

- We must ensure that we have a full permutation of ⟨ 0, … , m-1 ⟩.
- **Secondary clustering**: 2 distinct keys have the same h′ value, if they have the same probe sequence.

Trees

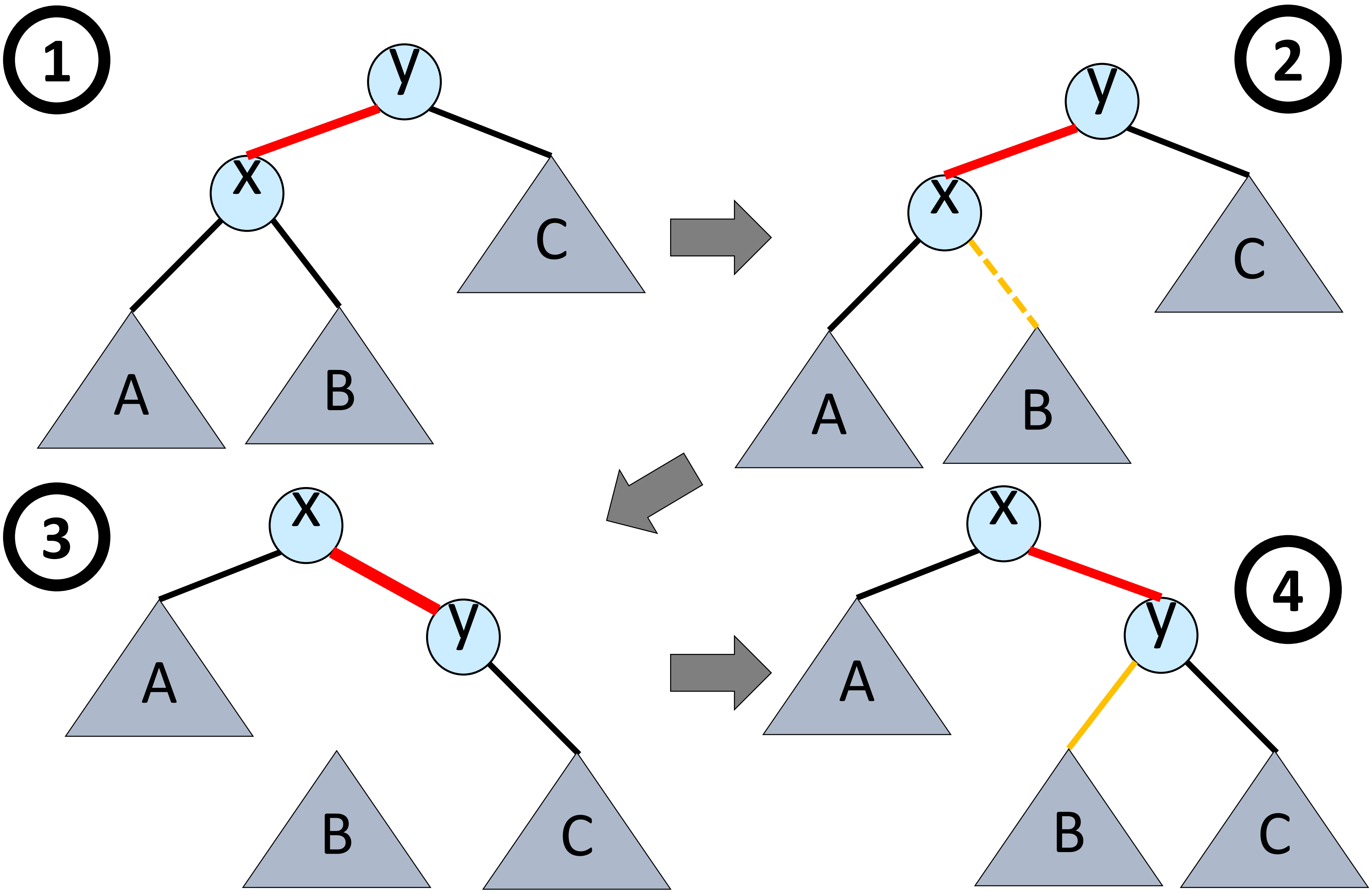
Rotations



Rotations:

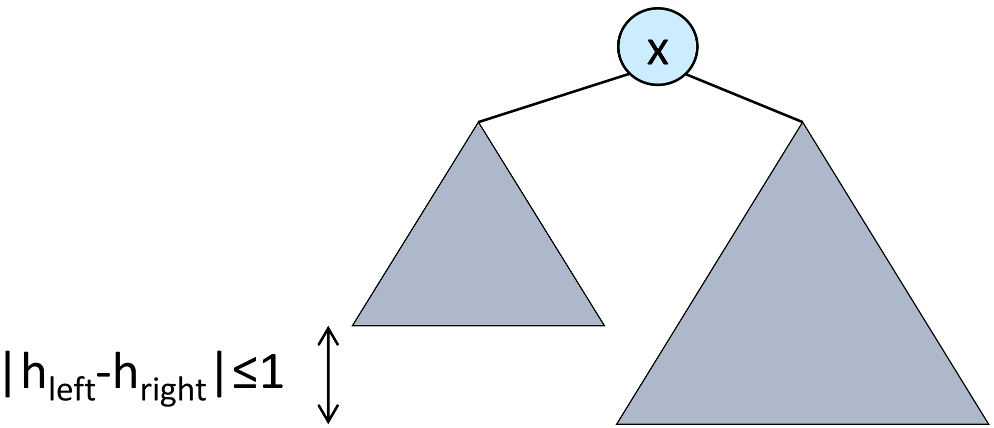
- Change tree structure
- Preserve the BST property.

Example: right rotation at y

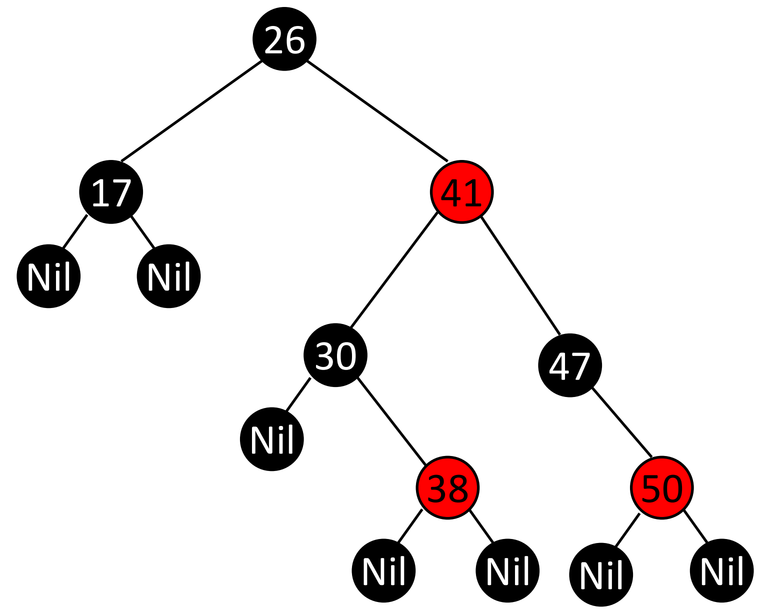


BST & Self-balanced trees

AVL trees

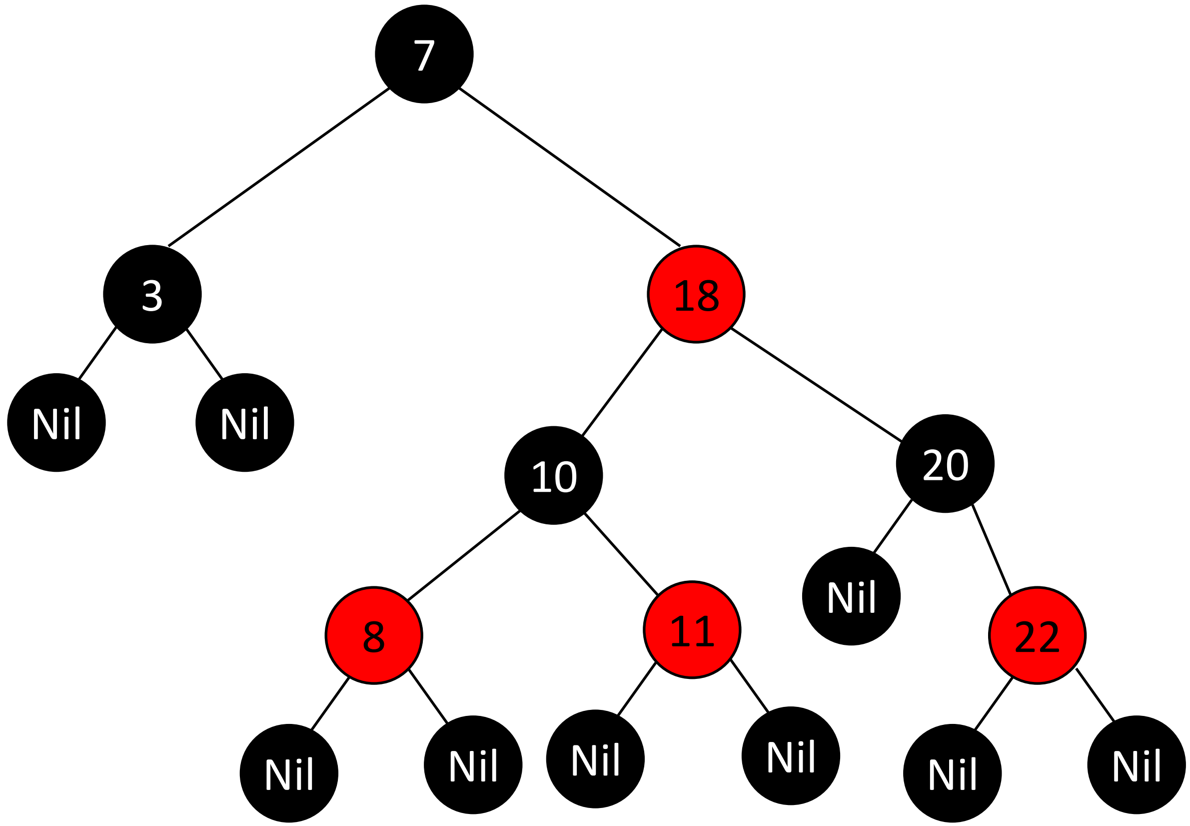


Red-black trees

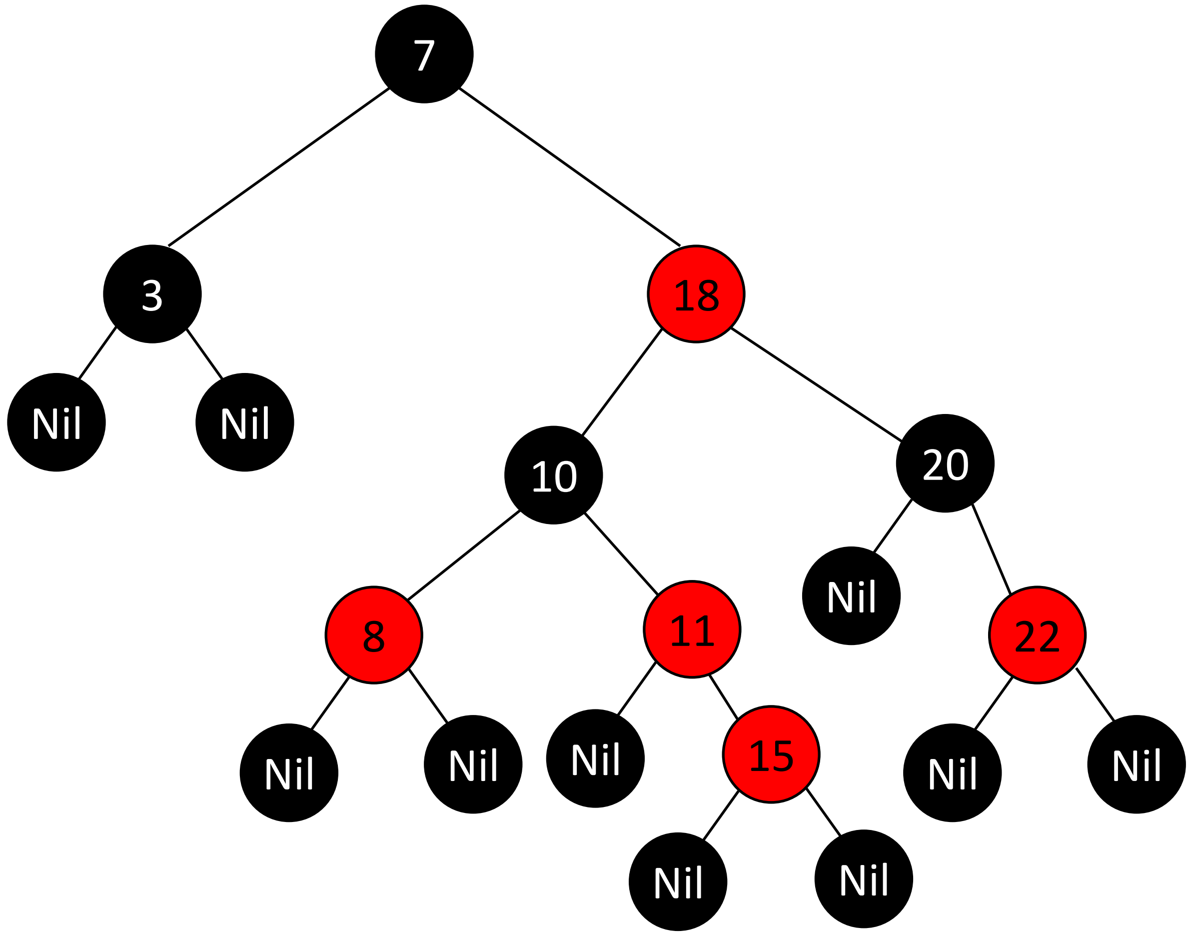


- BST (used to store keys)
- Running time dependent of the height ⟹ we try to keep the trees balanced.
- AVL & Red-Black trees are 2 types of self-balanced trees
- Challenge is to keep the AVL or Red-Black tree property valid after each operation.

Insert RB Tree – Example

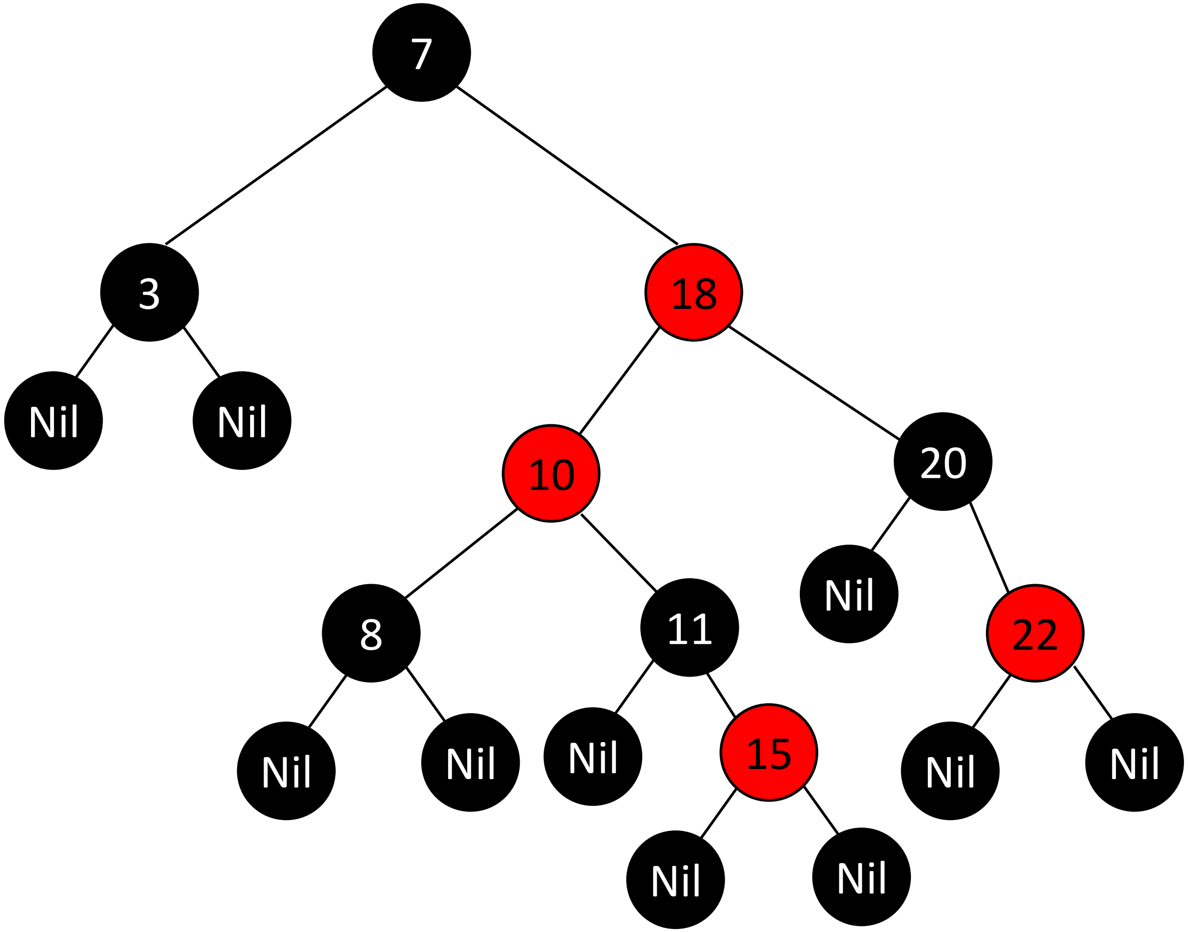


Insert RB Tree – Example



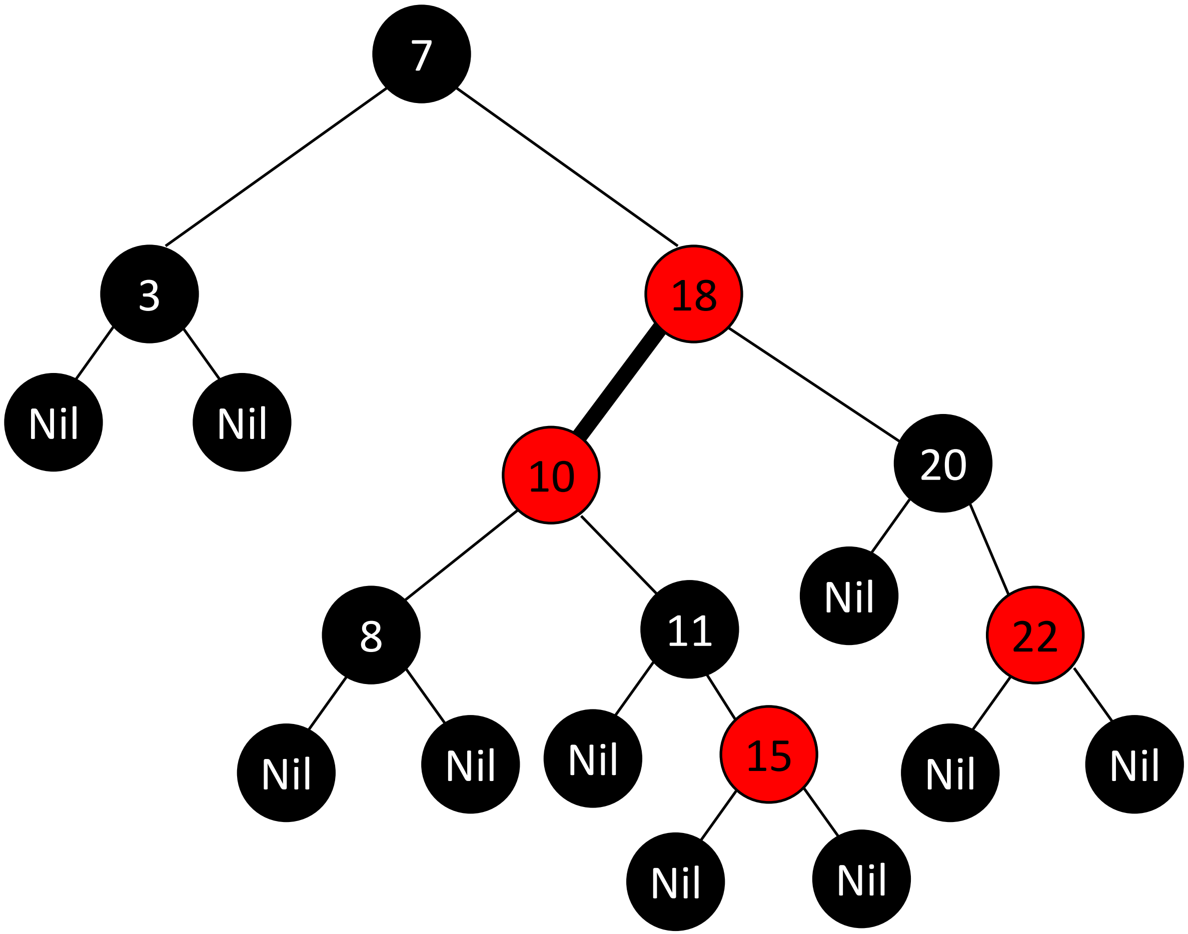
Insert(T,15)

Insert RB Tree – Example



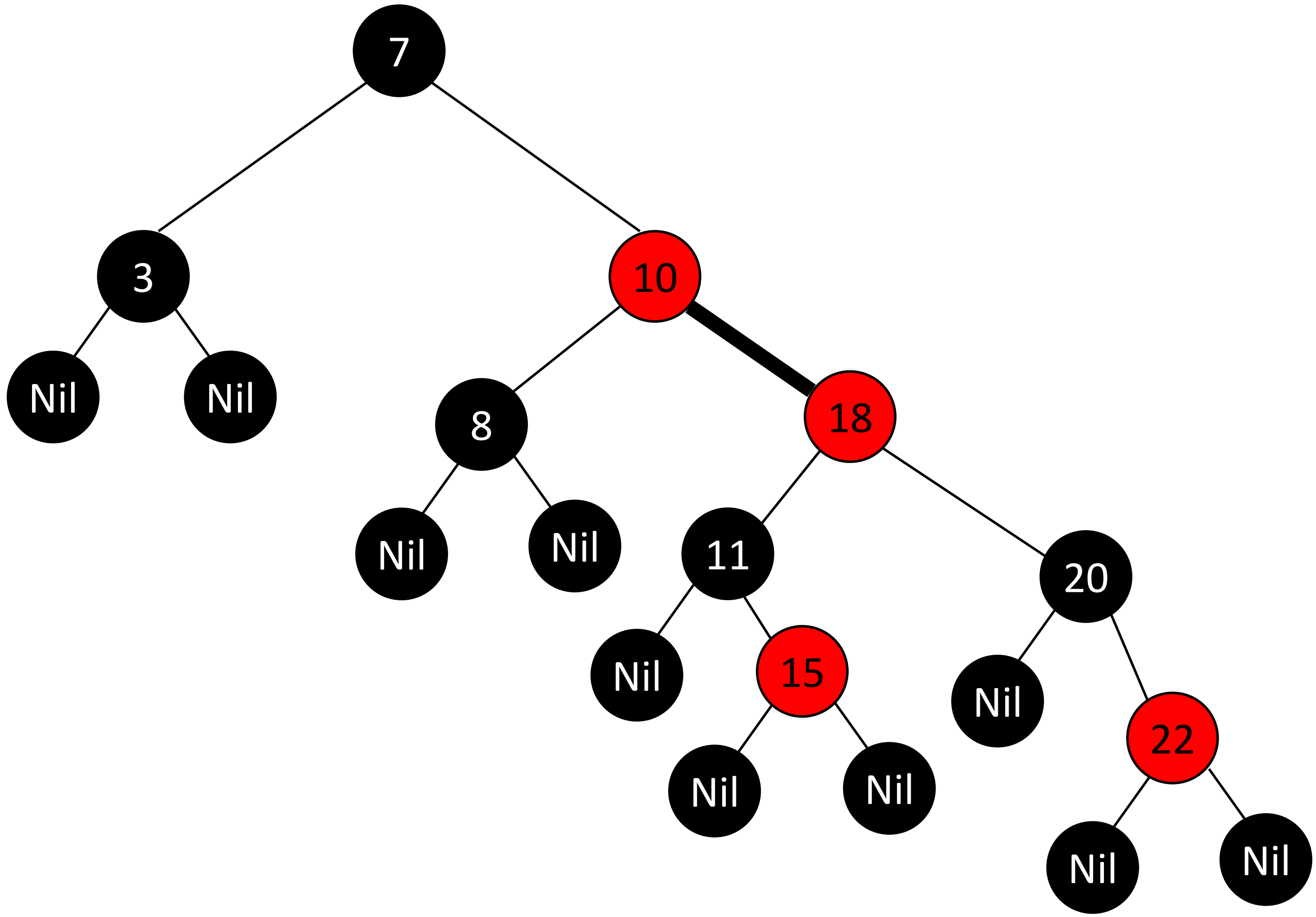
Recolor 10, 8 &11

Insert RB Tree – Example



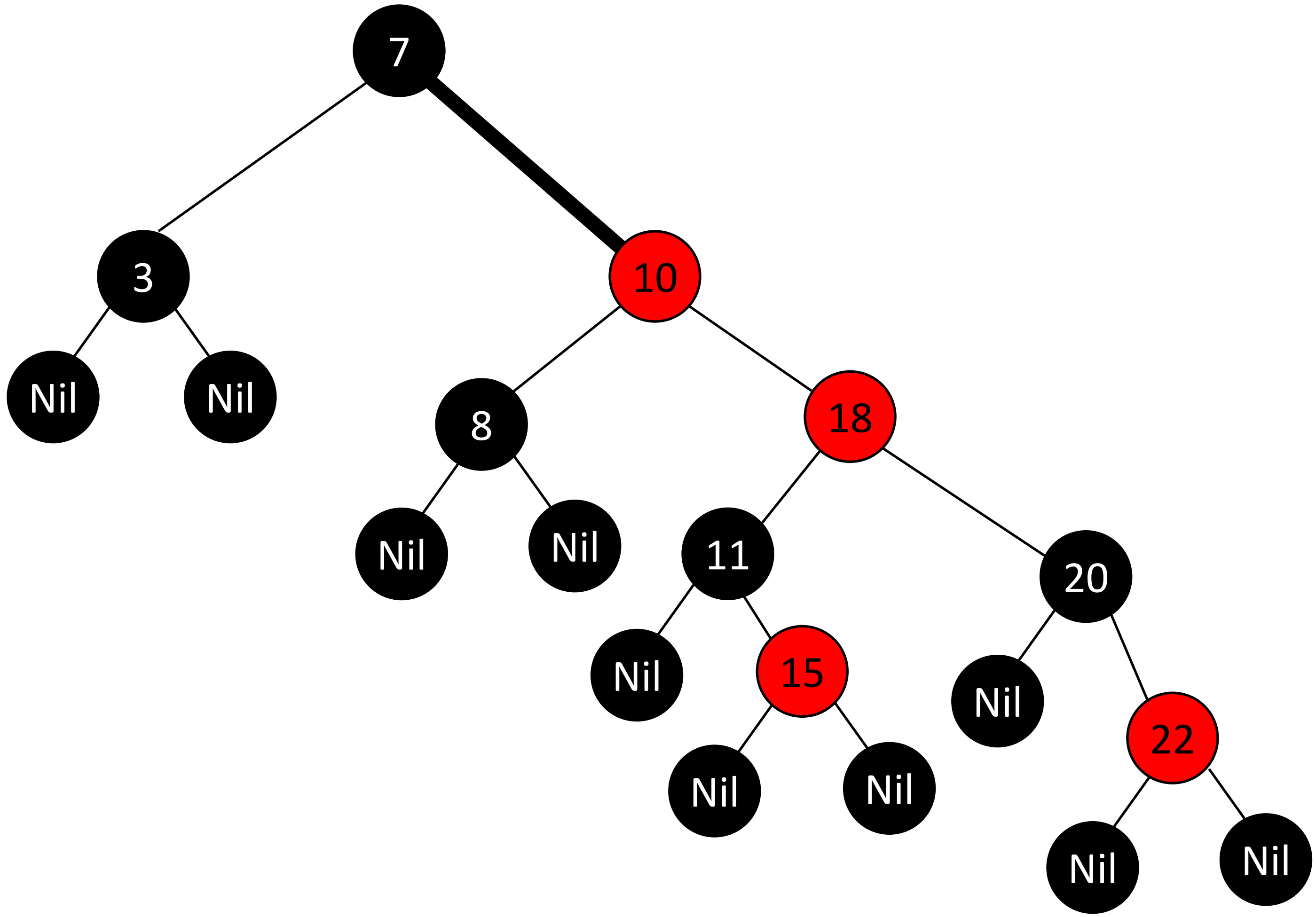
Right rotate at 18

Insert RB Tree – Example



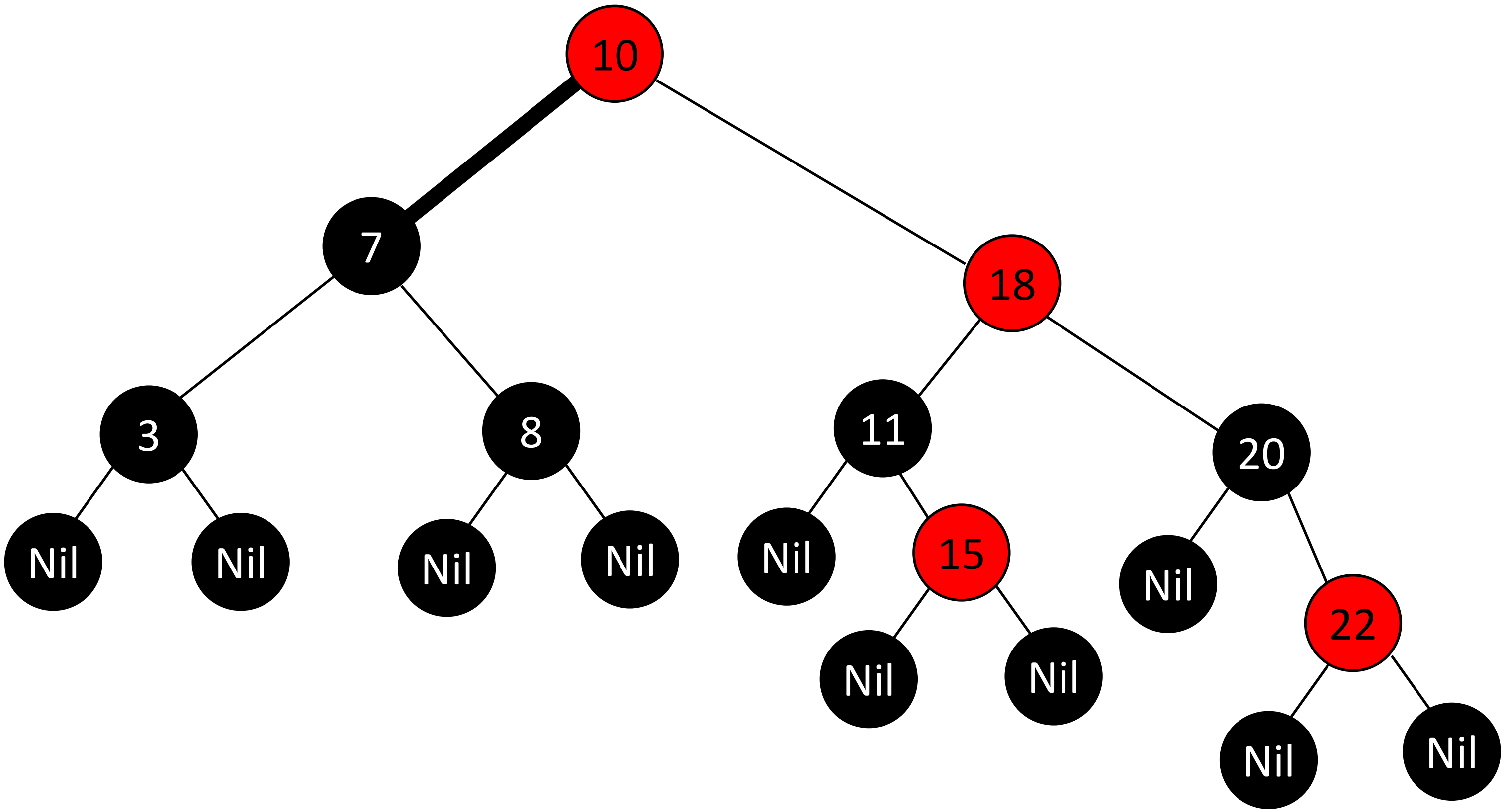
Right rotate at 18 (parent & child with conflict are aligned)

Insert RB Tree – Example



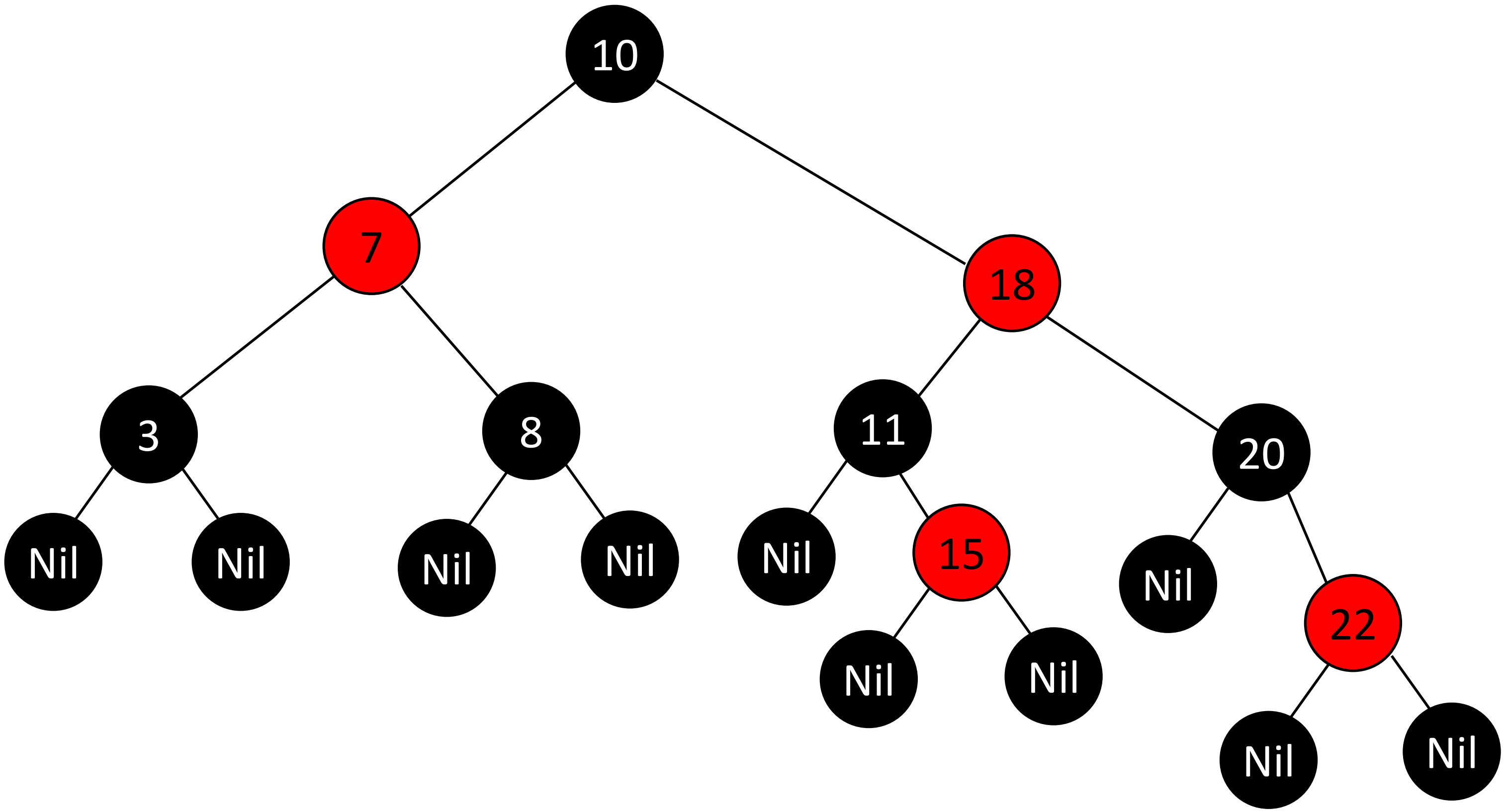
Left rotate at 7

Insert RB Tree – Example



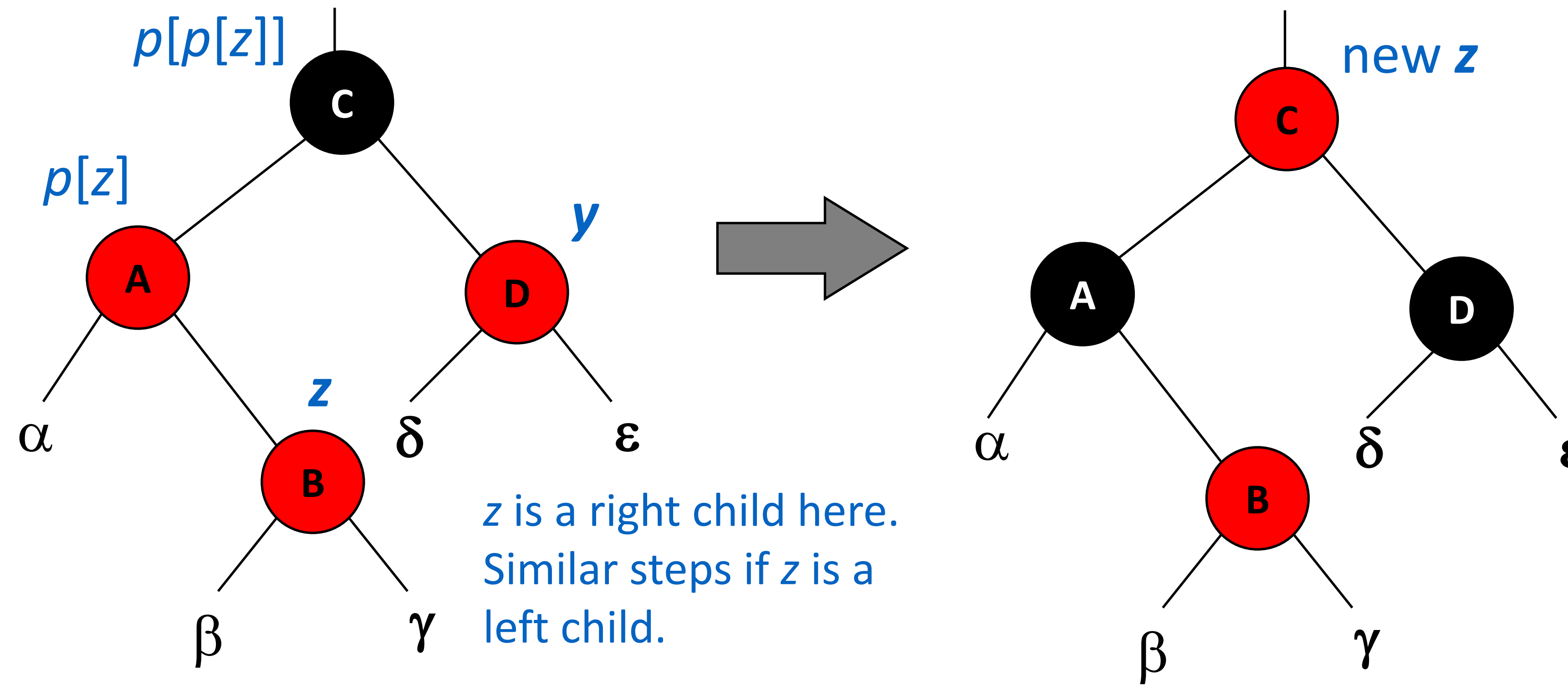
Left rotate at 7

Insert RB Tree – Example



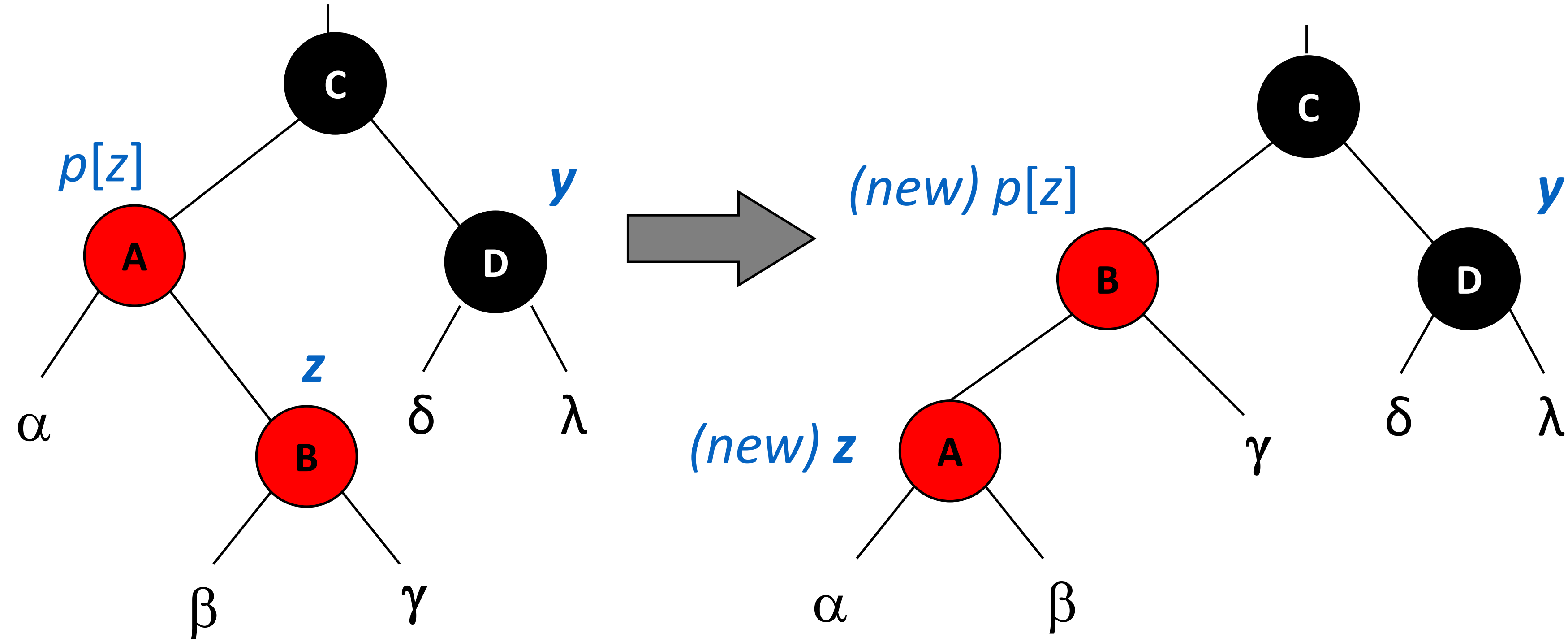
Recolor 10 & 7 (root must be black!)

Case 1 – uncle *y* is red



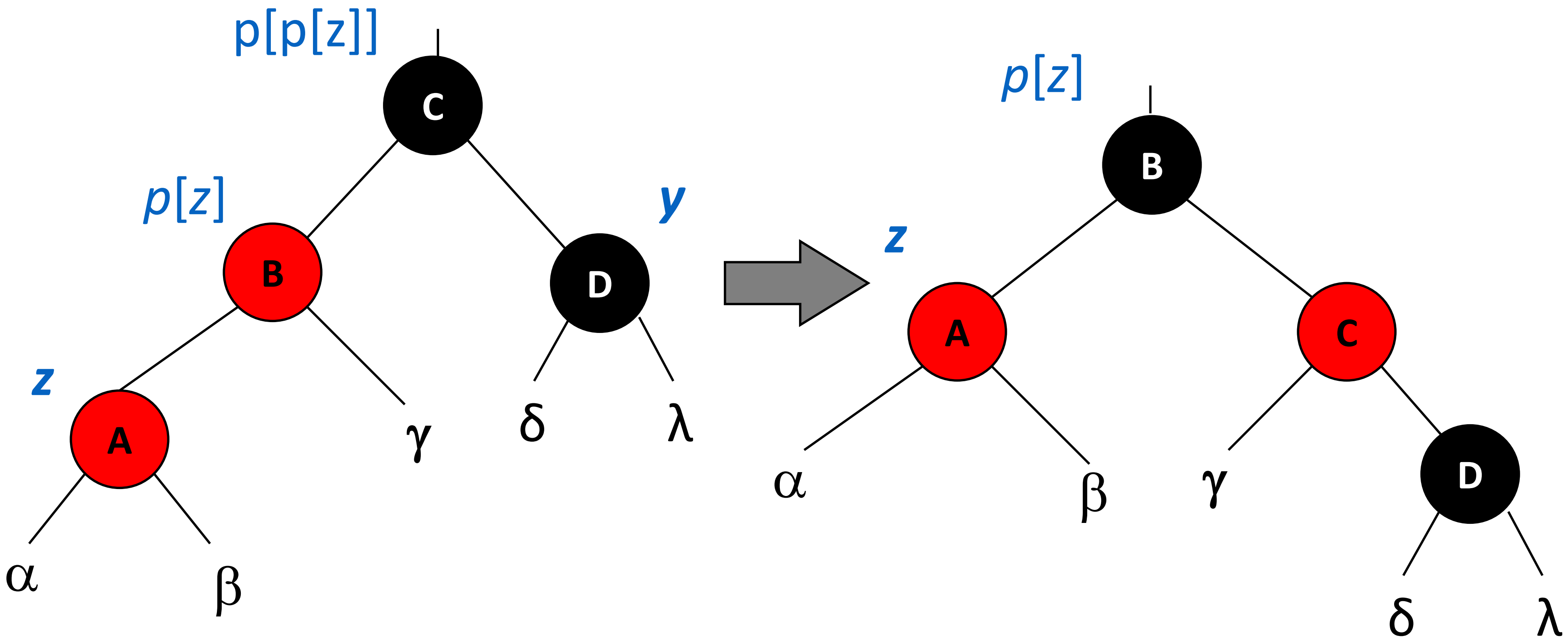
- $p[p[z]]$ (z's grandparent) must be black, since z and $p[z]$ are both red and there are no other violations of property 4.
- Make $p[z]$ and y black $\Rightarrow$ now z and $p[z]$ are not both red. But property 5 might now be violated.
- Make $p[p[z]]$ red $\Rightarrow$ restores property 5.
- The next iteration has $p[p[z]]$ as the new z (i.e., z moves up 2 levels).

Case 2 – *y* is black, *z* is a right child



- Left rotate around *p*[*z*], *p*[*z*] and *z* switch roles $\Rightarrow$ now *z* is a left child, and both *z* and *p*[*z*] are red.
- Takes us immediately to case 3.

Case 3 – *y* is black, *z* is a left child



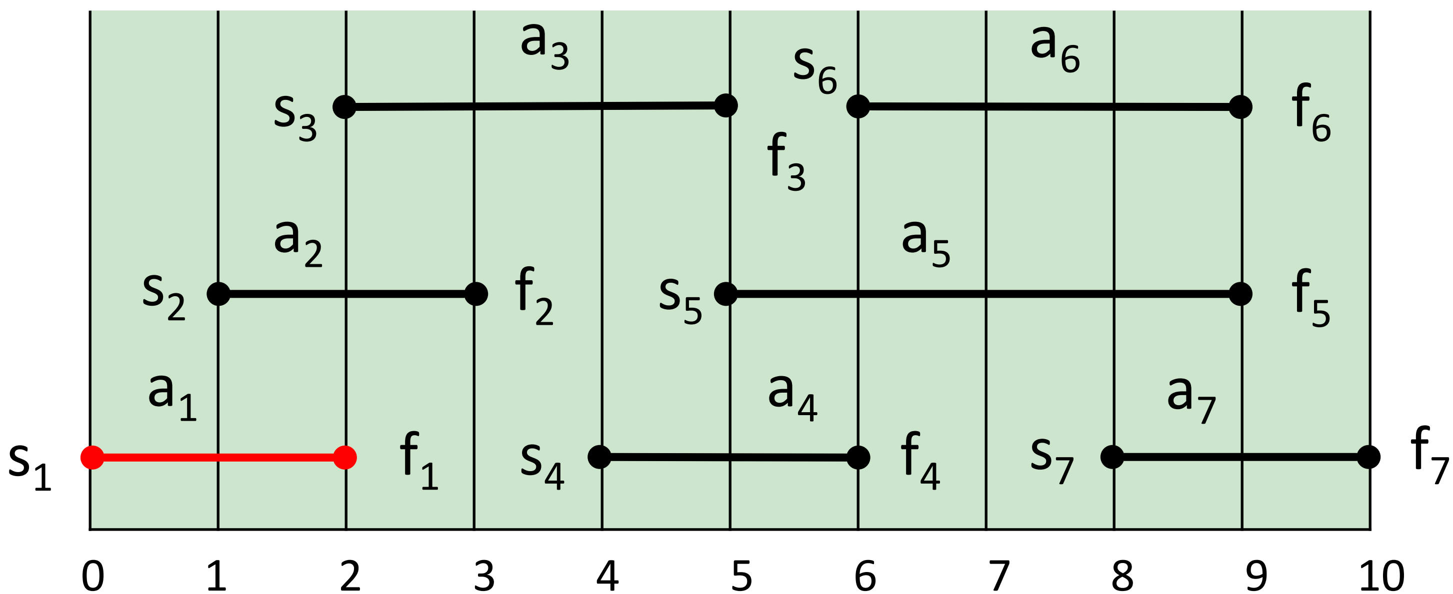
- Make $p[z]$ black and $p[p[z]]$ red.
- Then right rotate right on $p[p[z]]$ (in order to maintain property 4).
- No longer have 2 reds in a row.
- $p[z]$ is now black $\Rightarrow$ no more iterations.

Greedy algorithms

Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

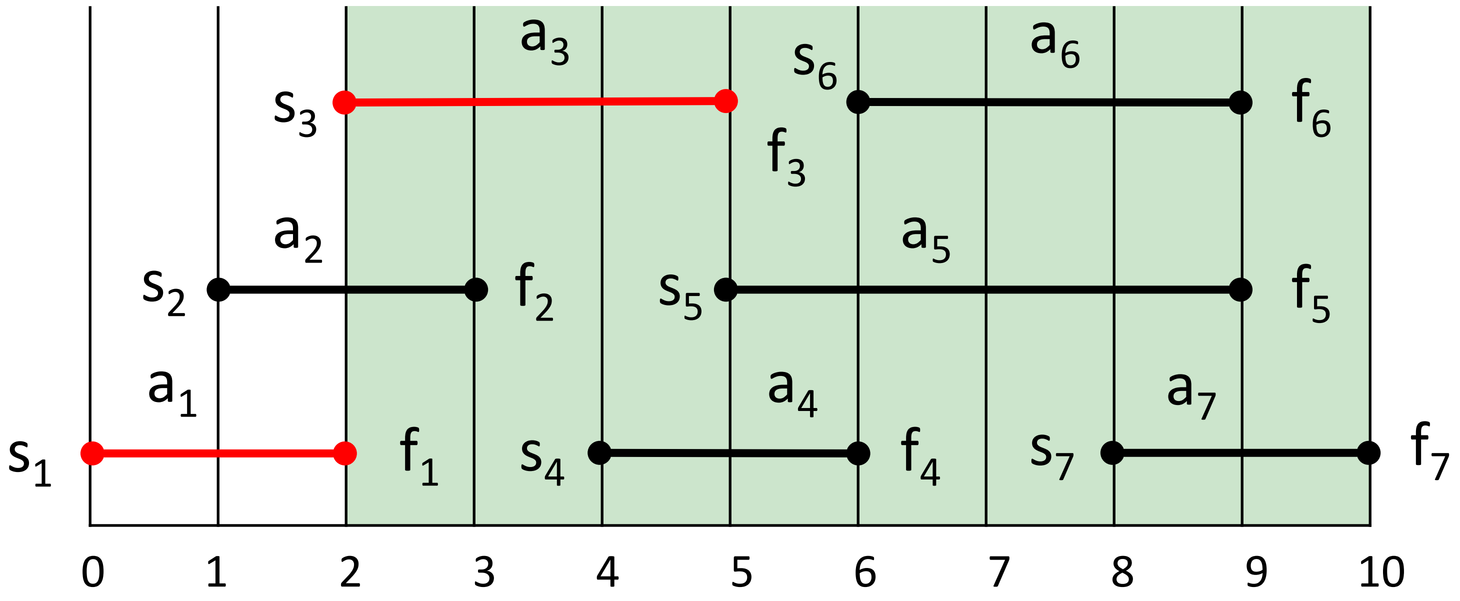
Activities sorted by finishing time.



Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

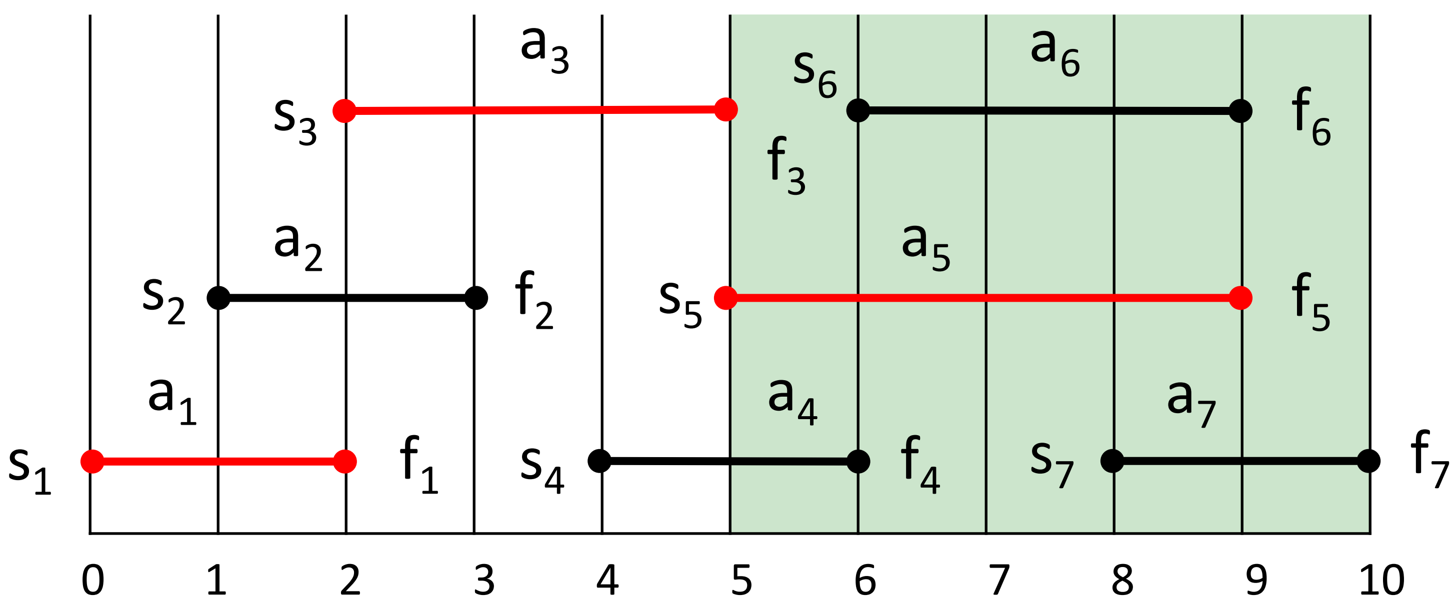
Activities sorted by finishing time.



Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

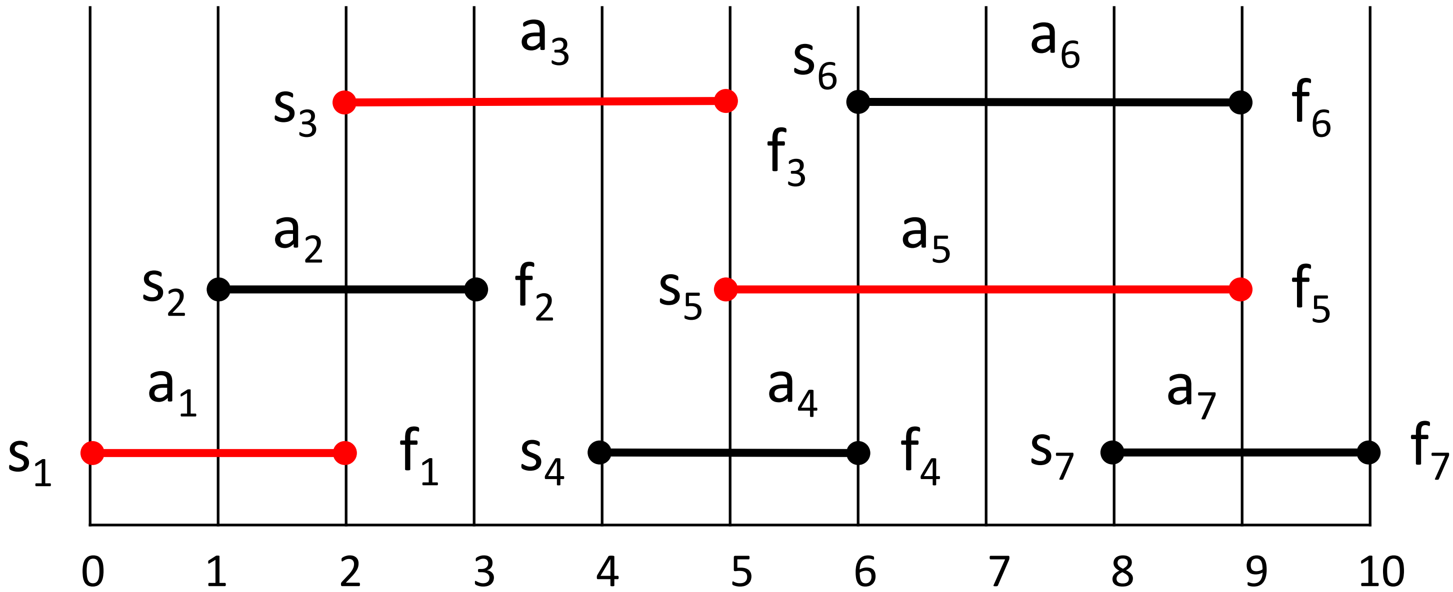
Activities sorted by finishing time.



Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

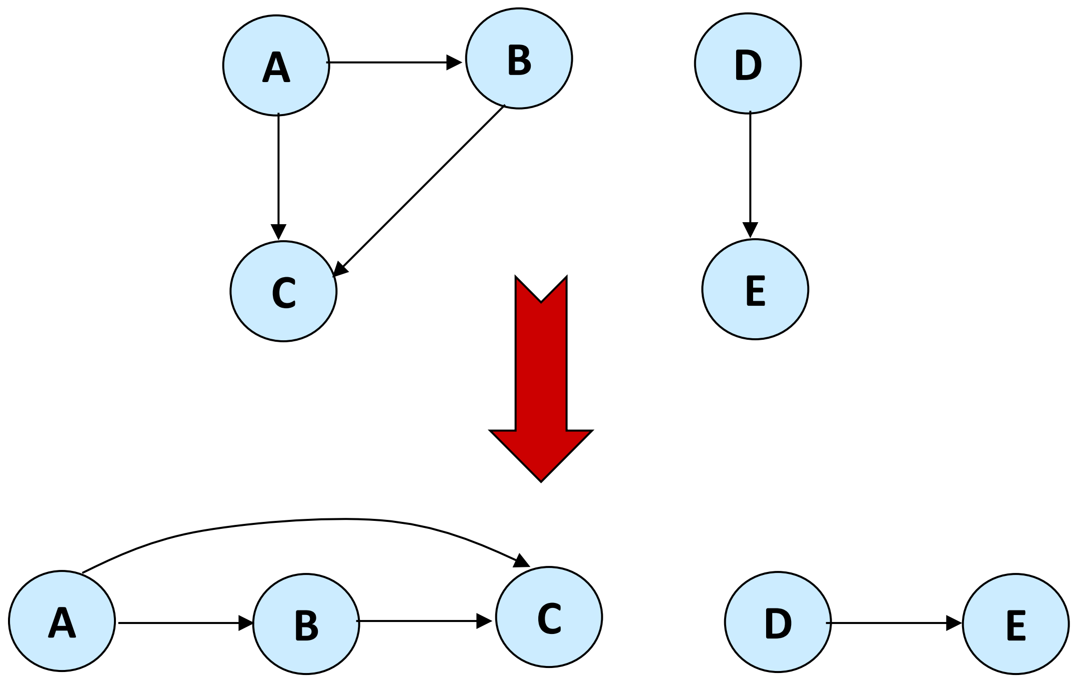
Activities sorted by finishing time.



Graph Algorithms

Topological Sort

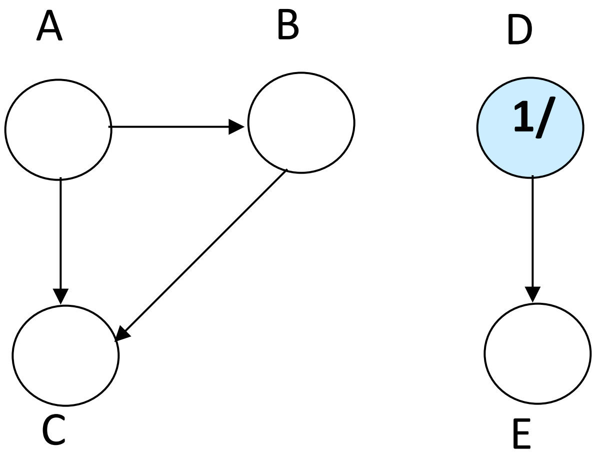
Want to "sort" a directed acyclic graph (DAG).



Think of original DAG as a **partial order**.

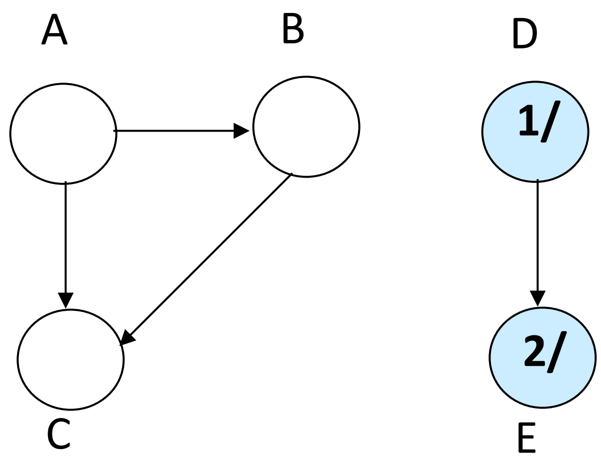
Want a **total order** that extends this partial order.

Example 1



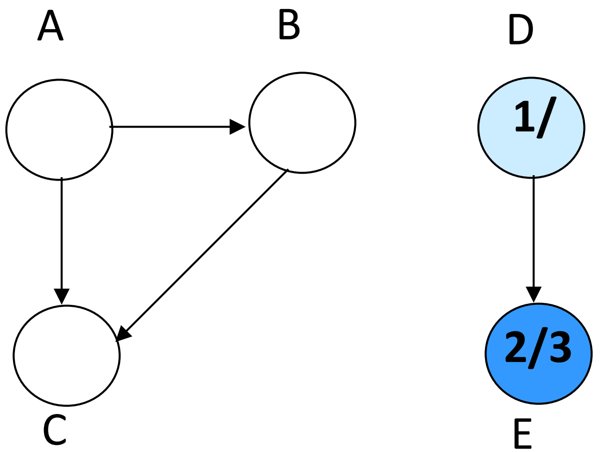
Linked List:

Example 1

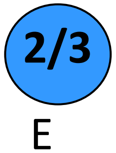


Linked List:

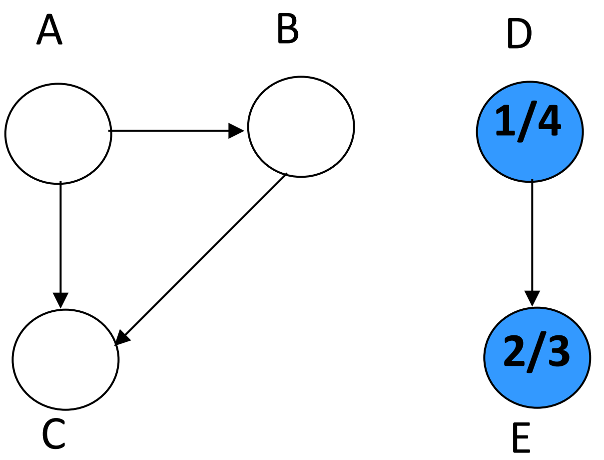
Example 1



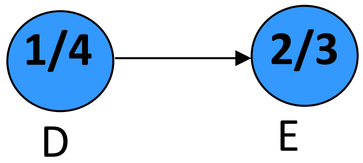
Linked List:



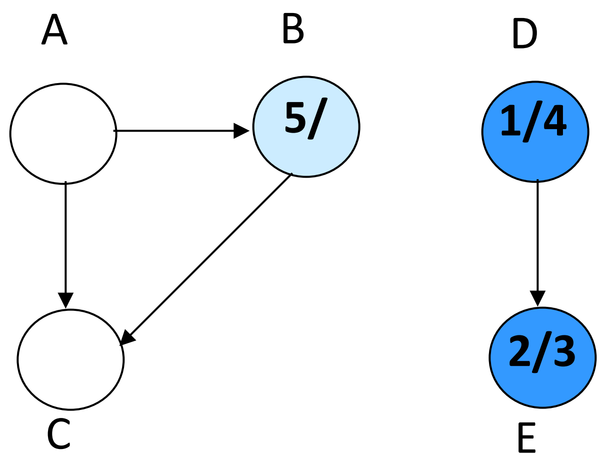
Example 1



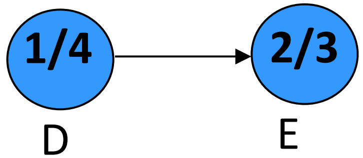
Linked List:



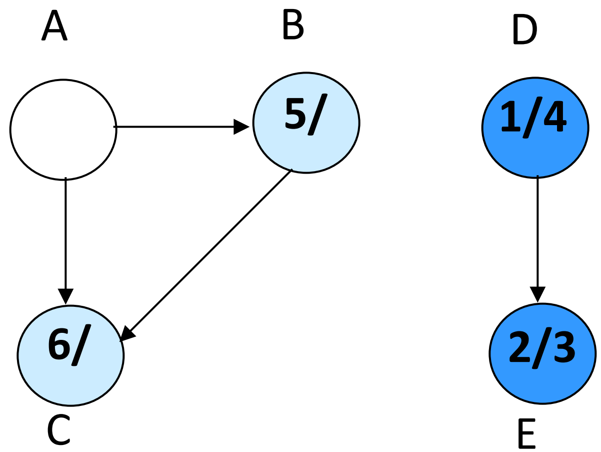
Example 1



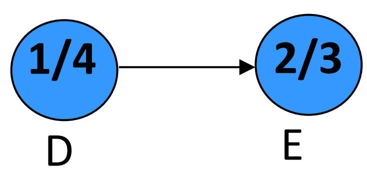
Linked List:



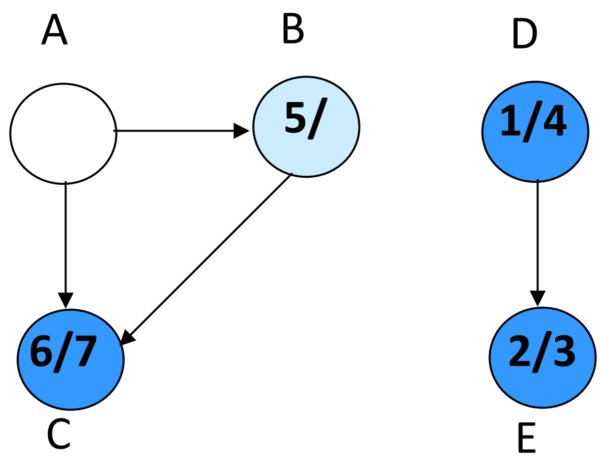
Example 1



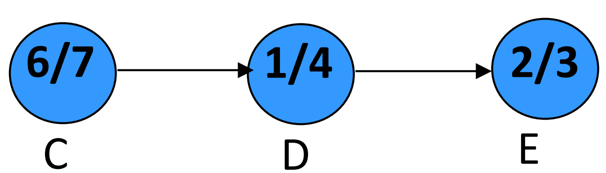
Linked List:



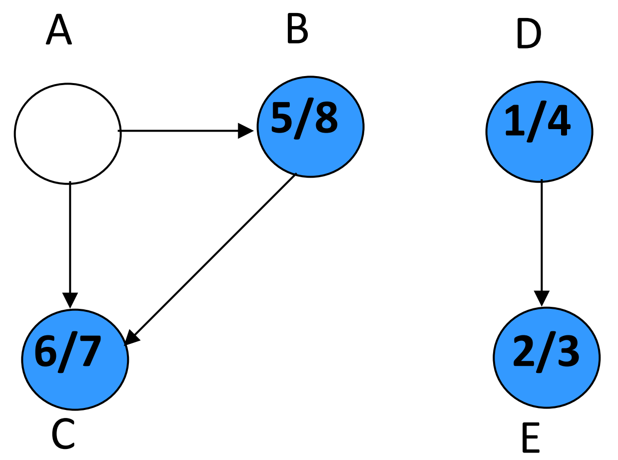
Example 1



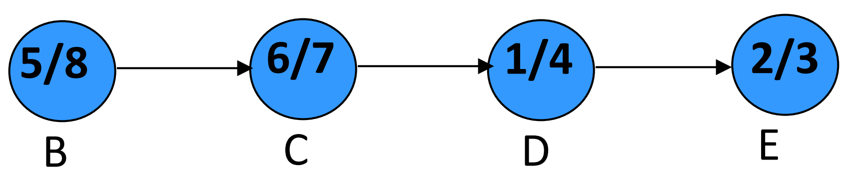
Linked List:



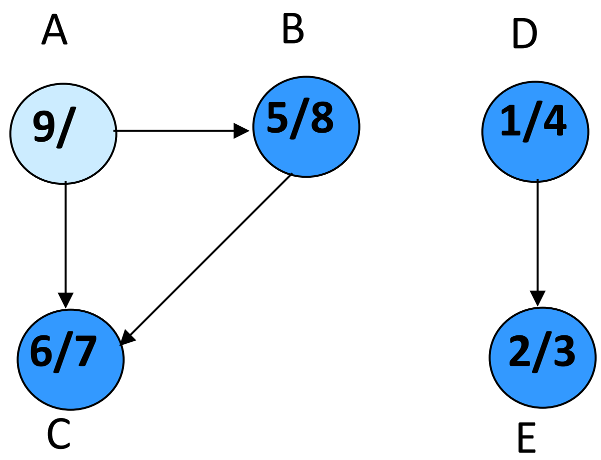
Example 1



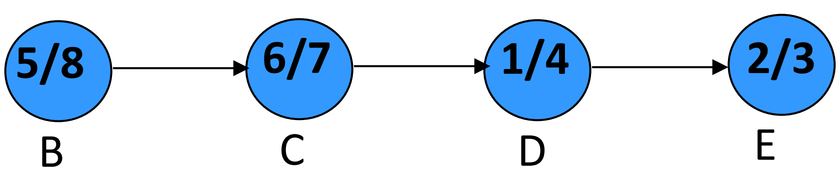
Linked List:



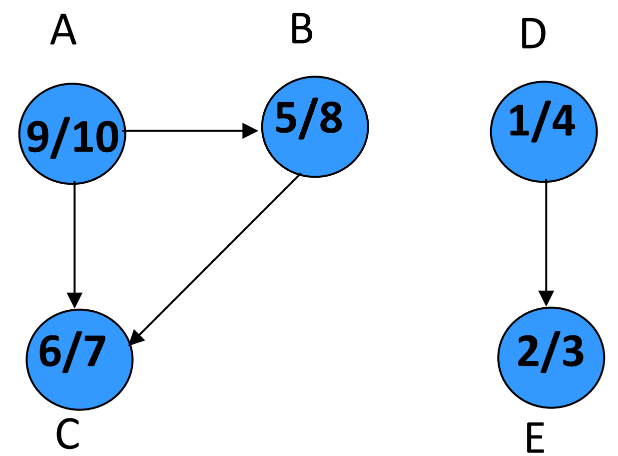
Example 1



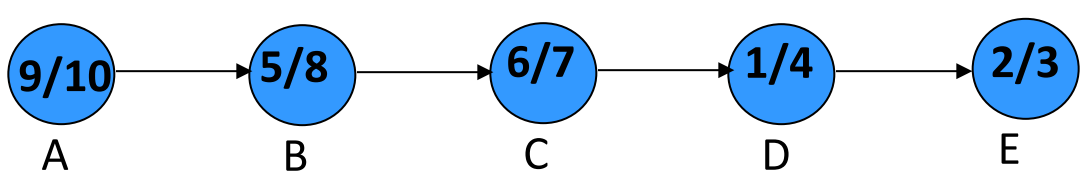
Linked List:



Example 1

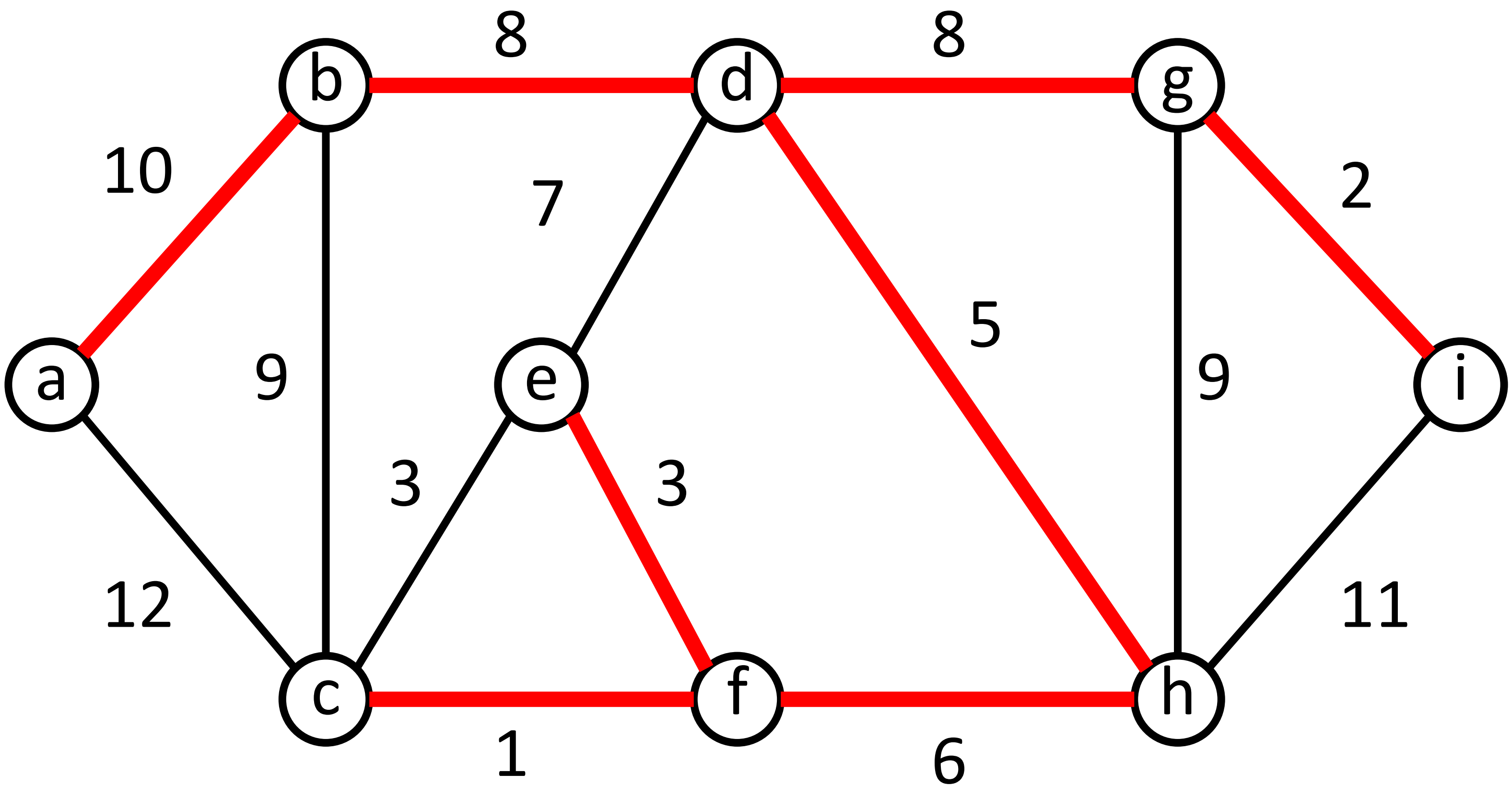


Linked List:



Minimum Spanning Trees

Minimum Spanning Tree (MST)

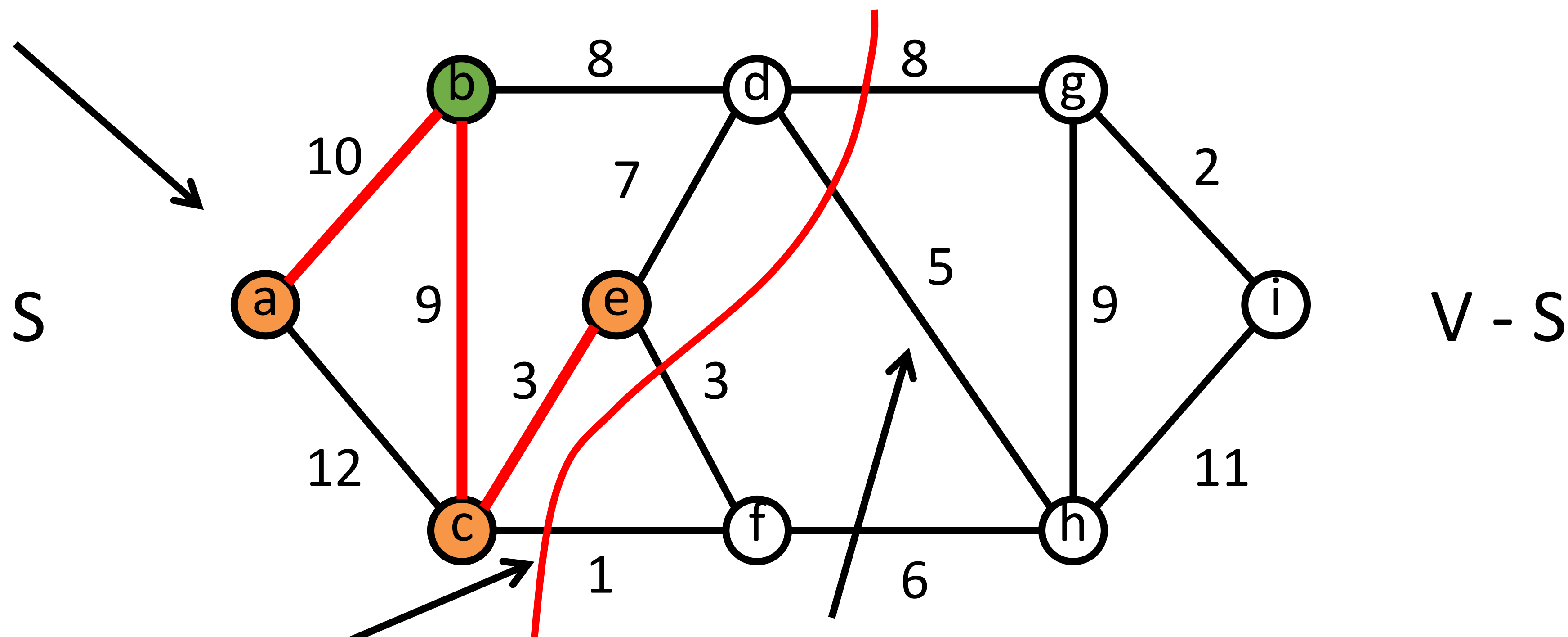


- It has $|V| - 1$ edges.
- It has no cycles.
- It might not be unique.

Definitions

A cut **respects** *A* if and only if no edge in *A* crosses the cut.

cut partitions vertices into disjoint sets, *S* and *V* – *S*.



A **light** edge crossing cut (may not be unique)

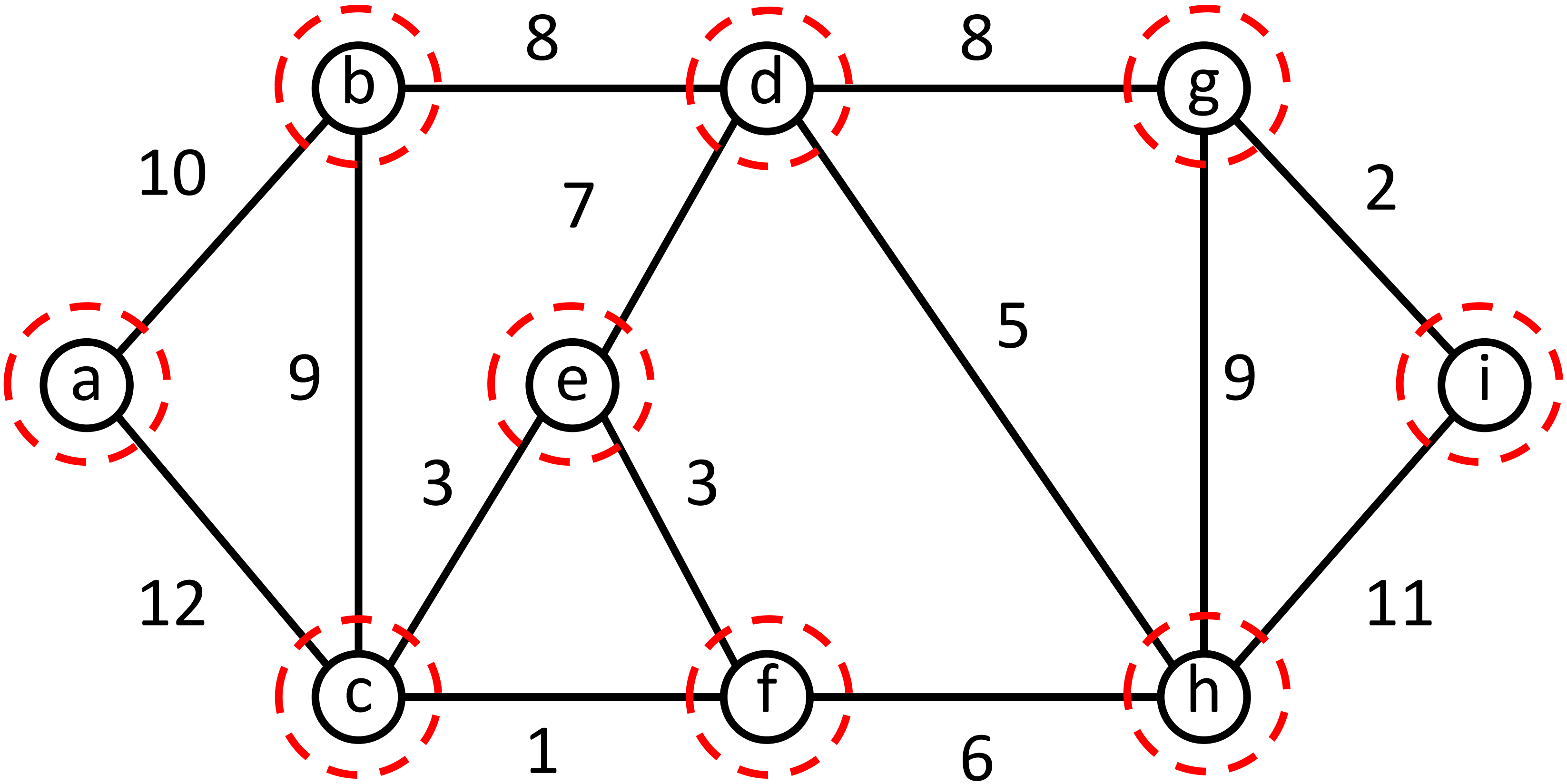
This edge **crosses** the cut. (one endpoint is in *S* and the other is in *V* – *S*.)

Kruskal's Algorithm

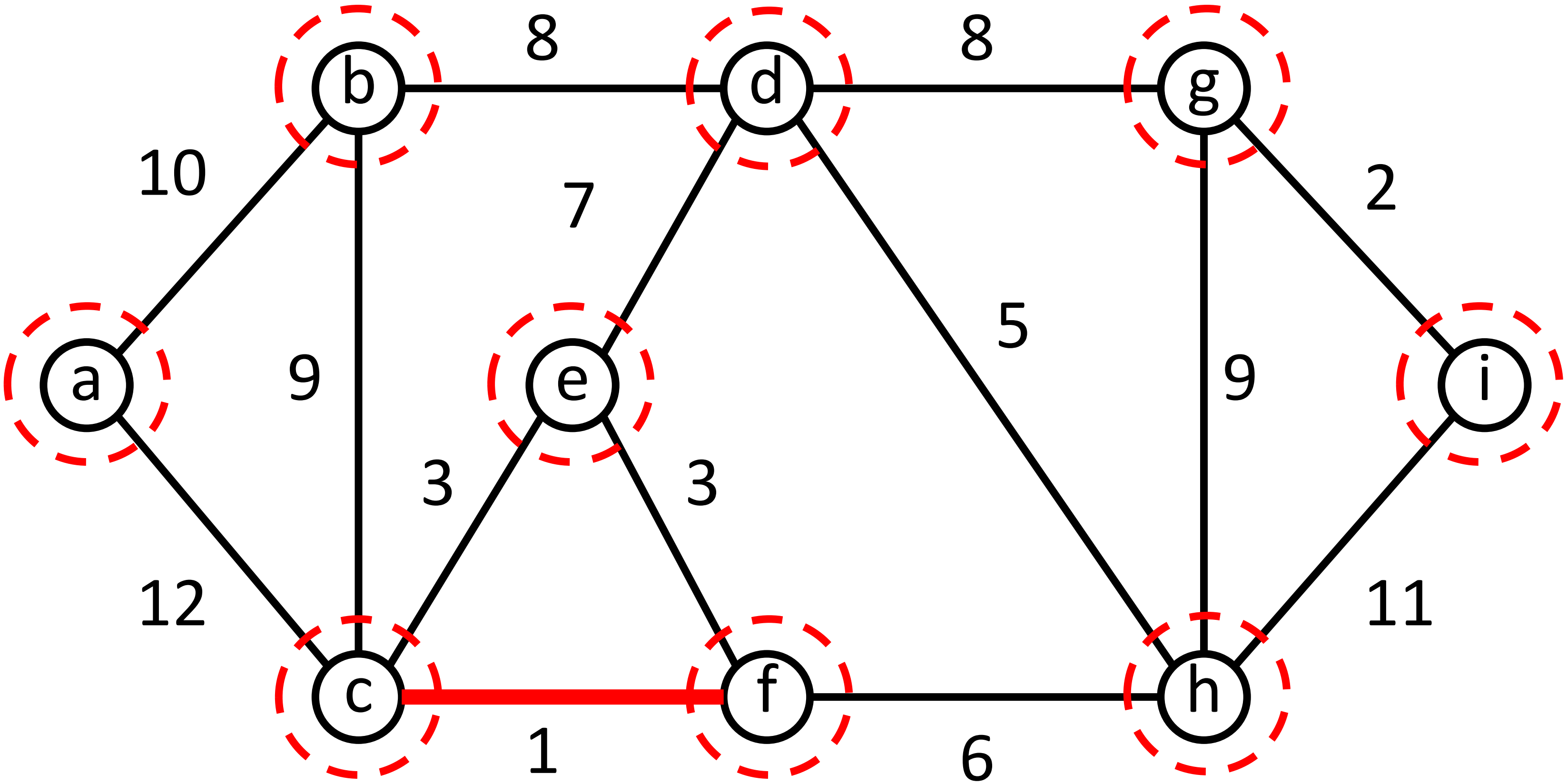
1. Starts with each vertex in its own component.
2. Repeatedly merges two components into one by choosing a light edge that connects them (i.e., a light edge crossing the cut between them).
3. Scans the set of edges in monotonically increasing order by weight.
4. Uses a **disjoint-set data structure** to determine whether an edge connects vertices in different components.

Note: We also covered the Prim's algorithm to calculate a MST.

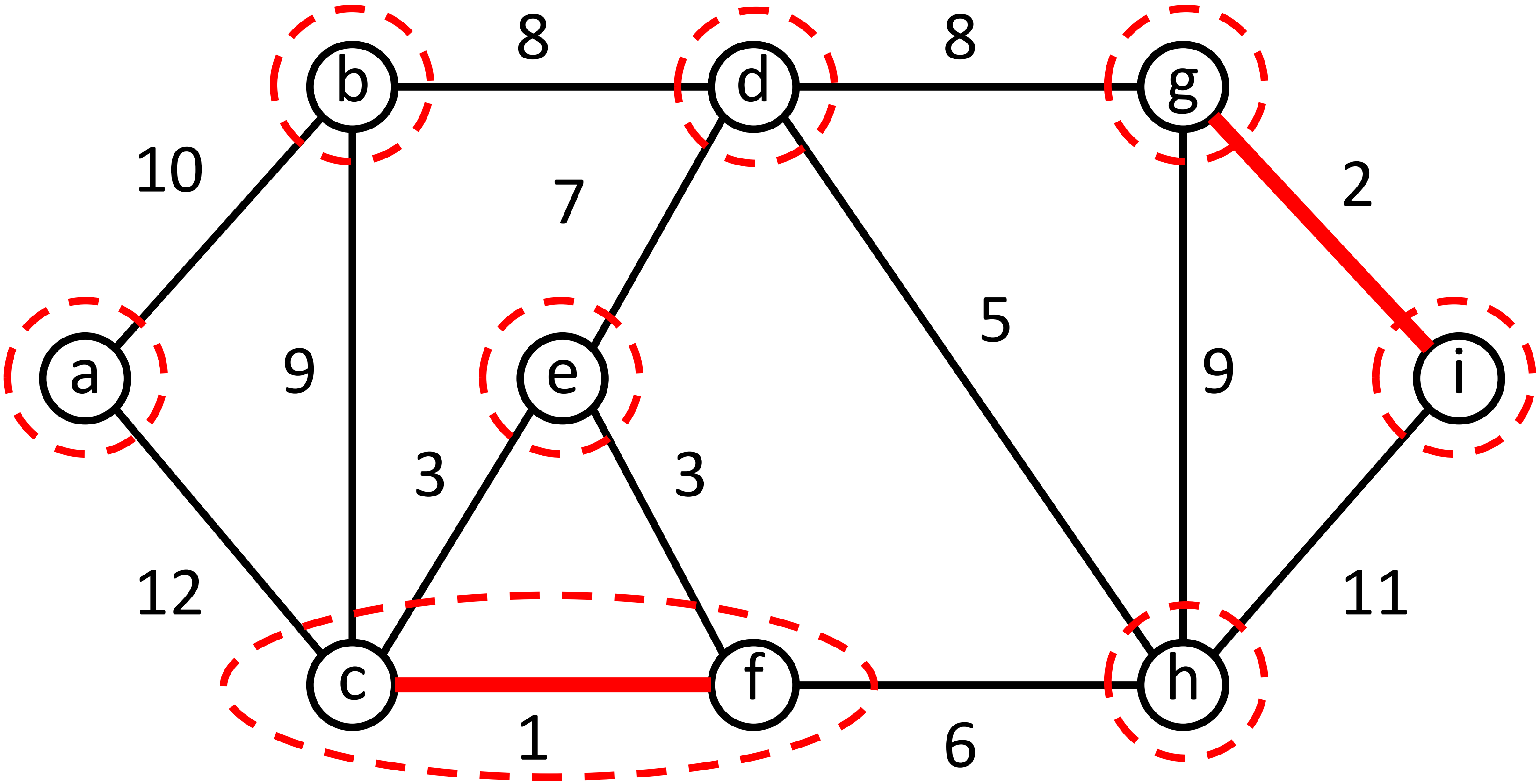
Example



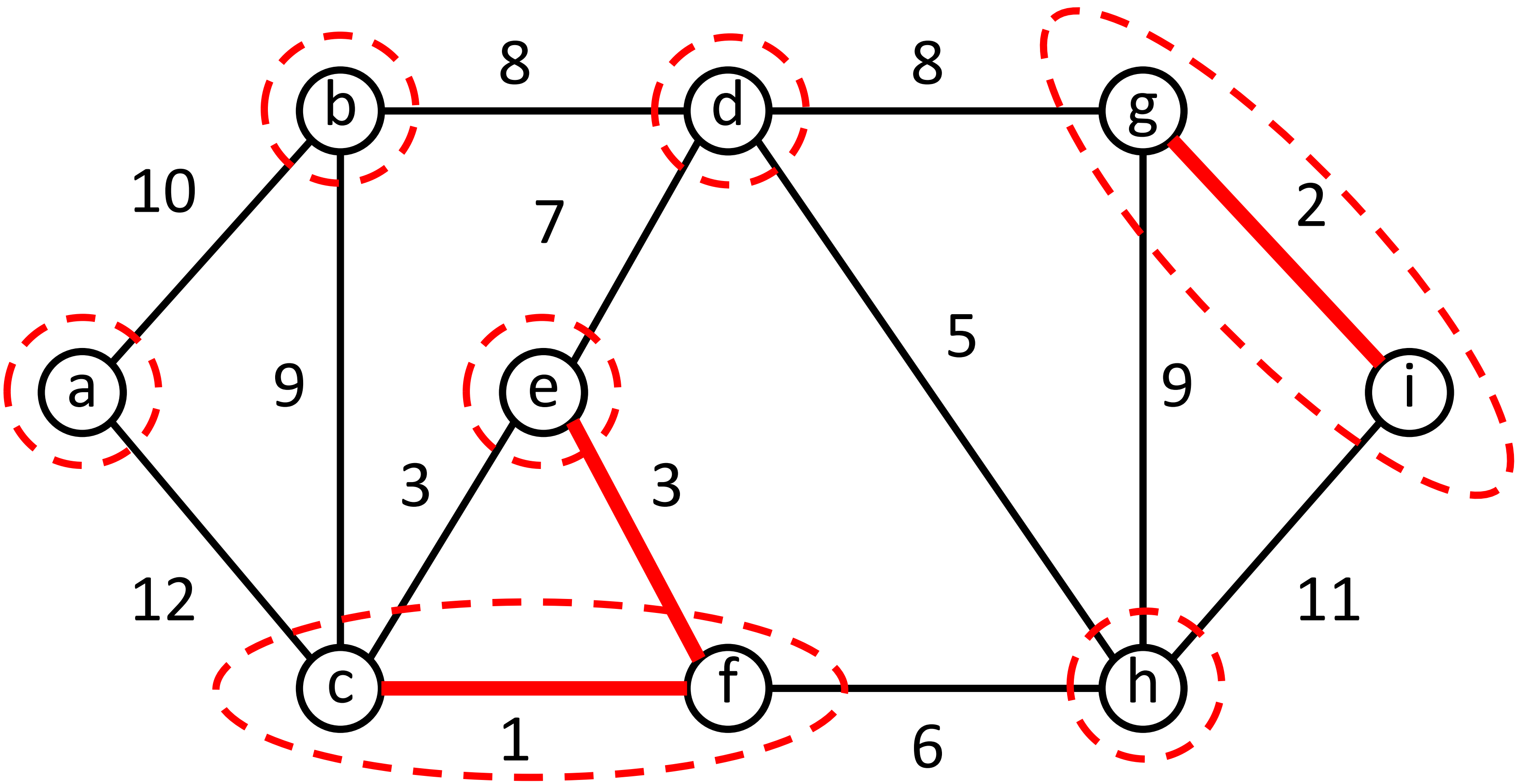
Example



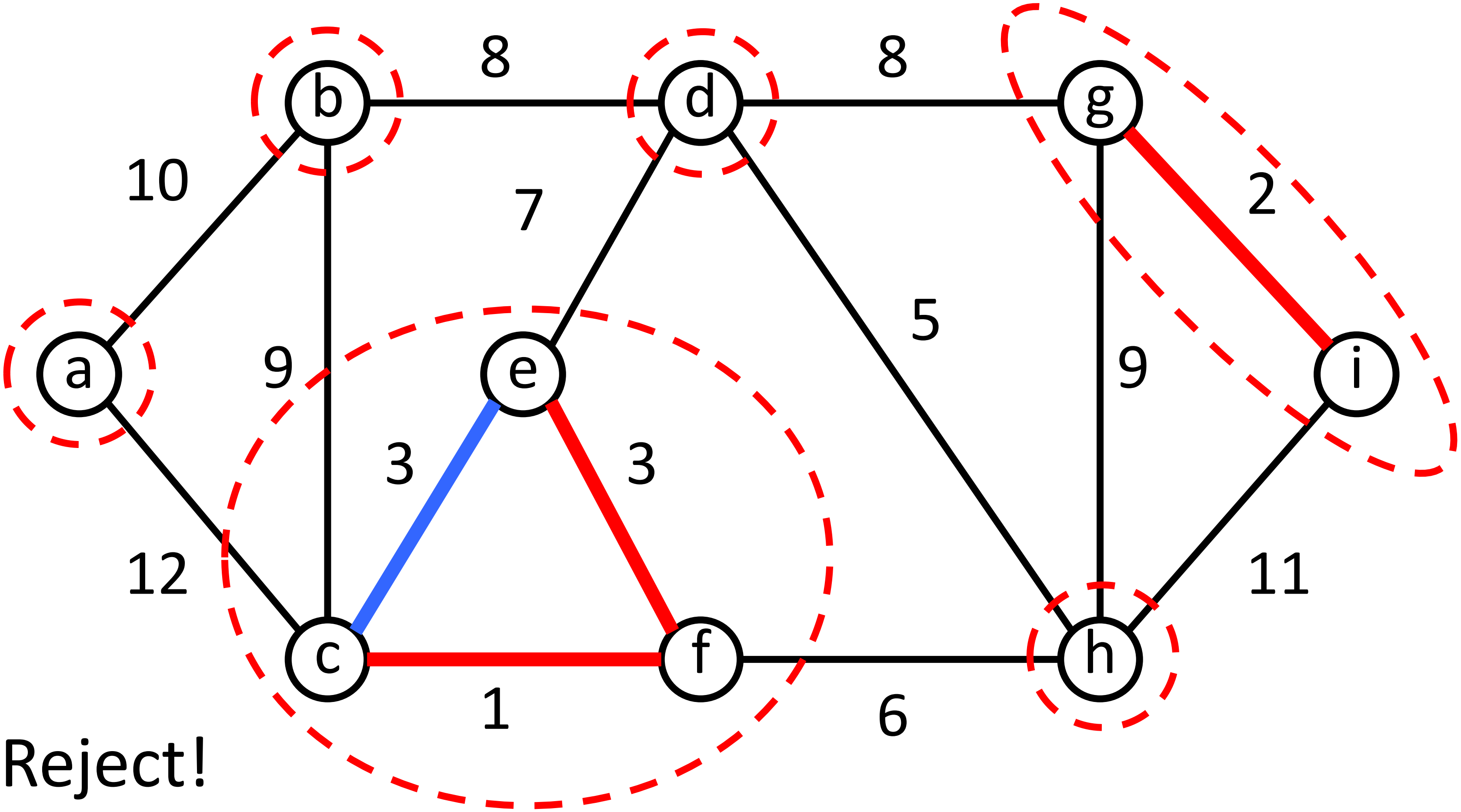
Example



Example

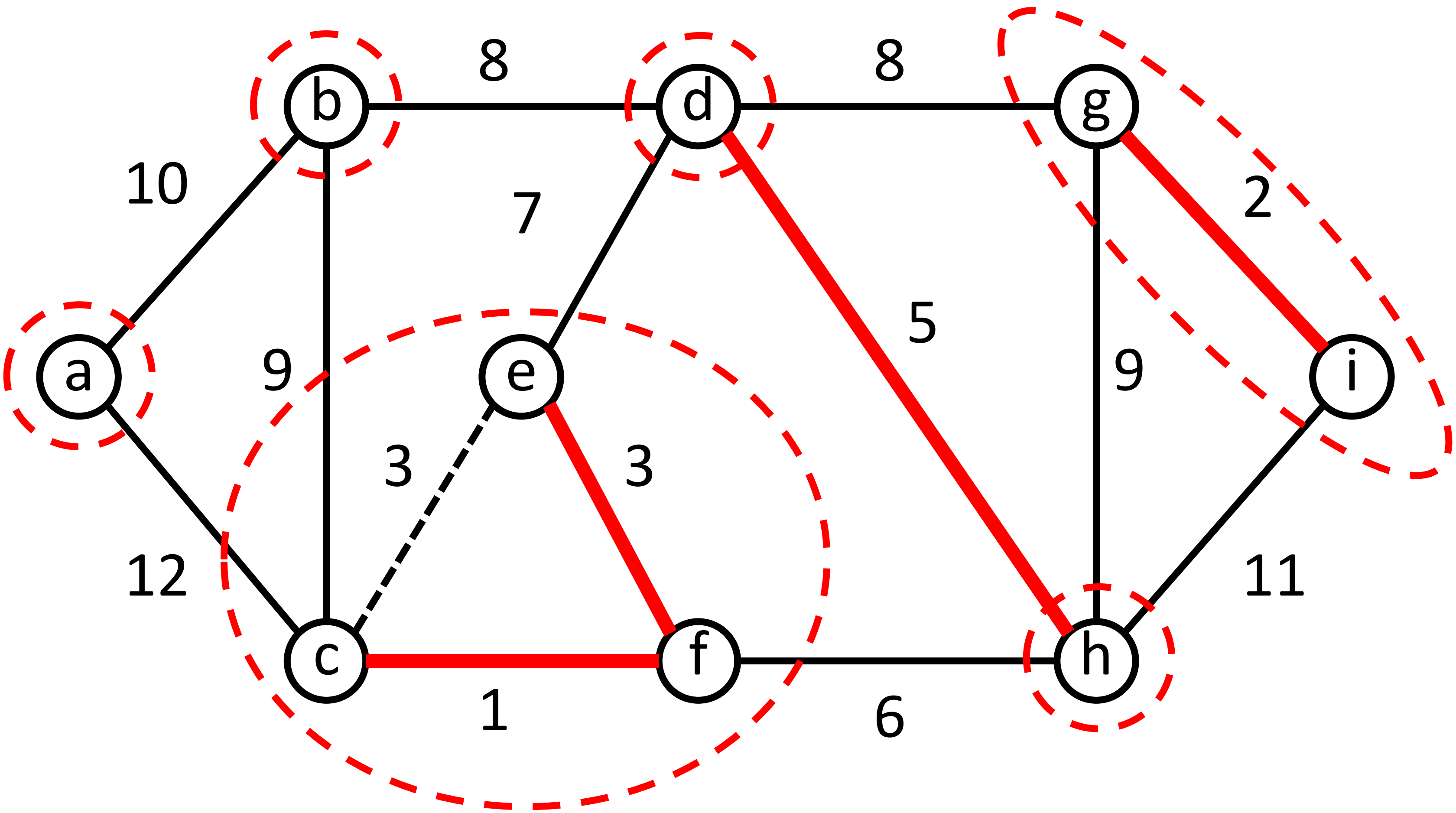


Example

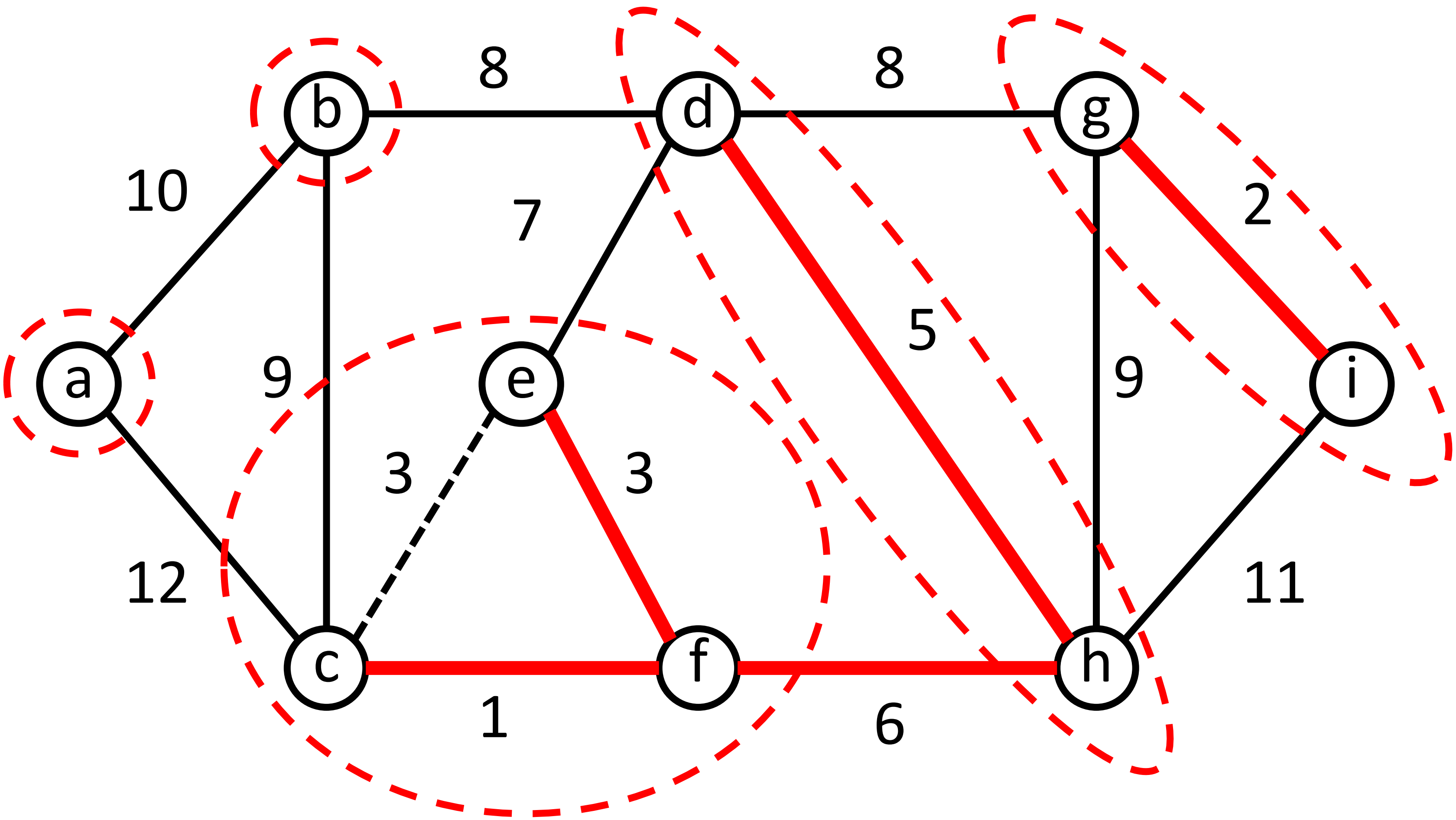


Reject!

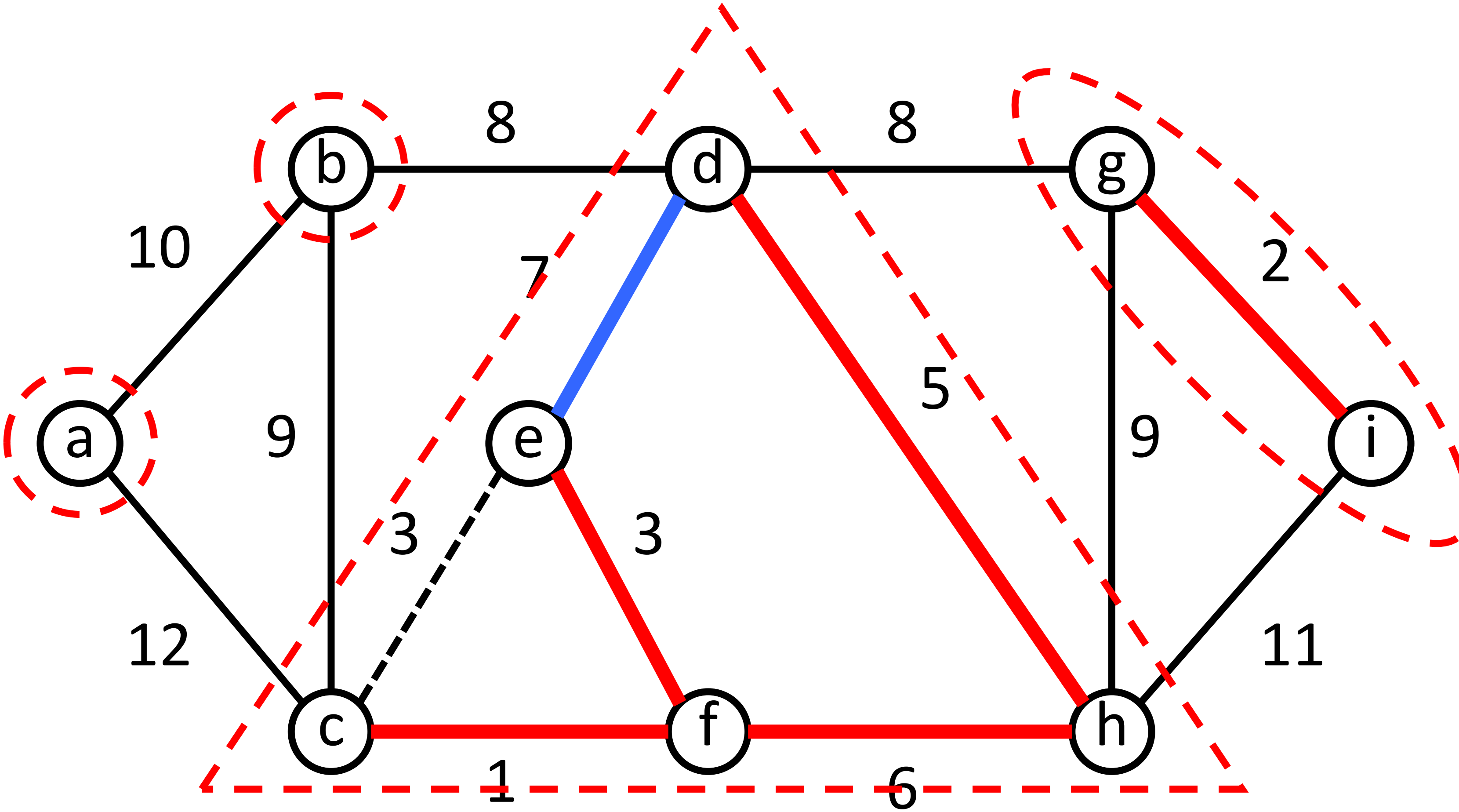
Example



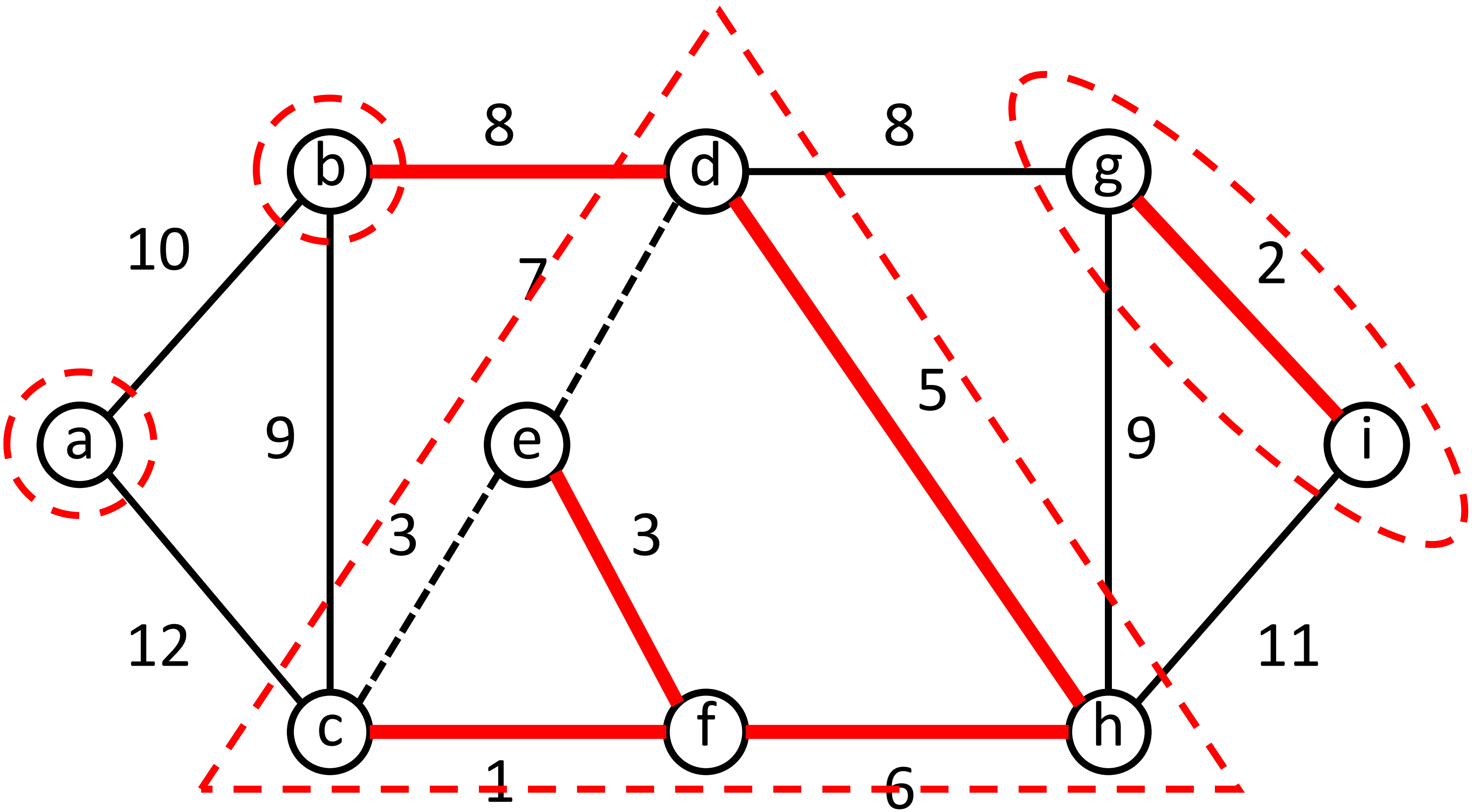
Example



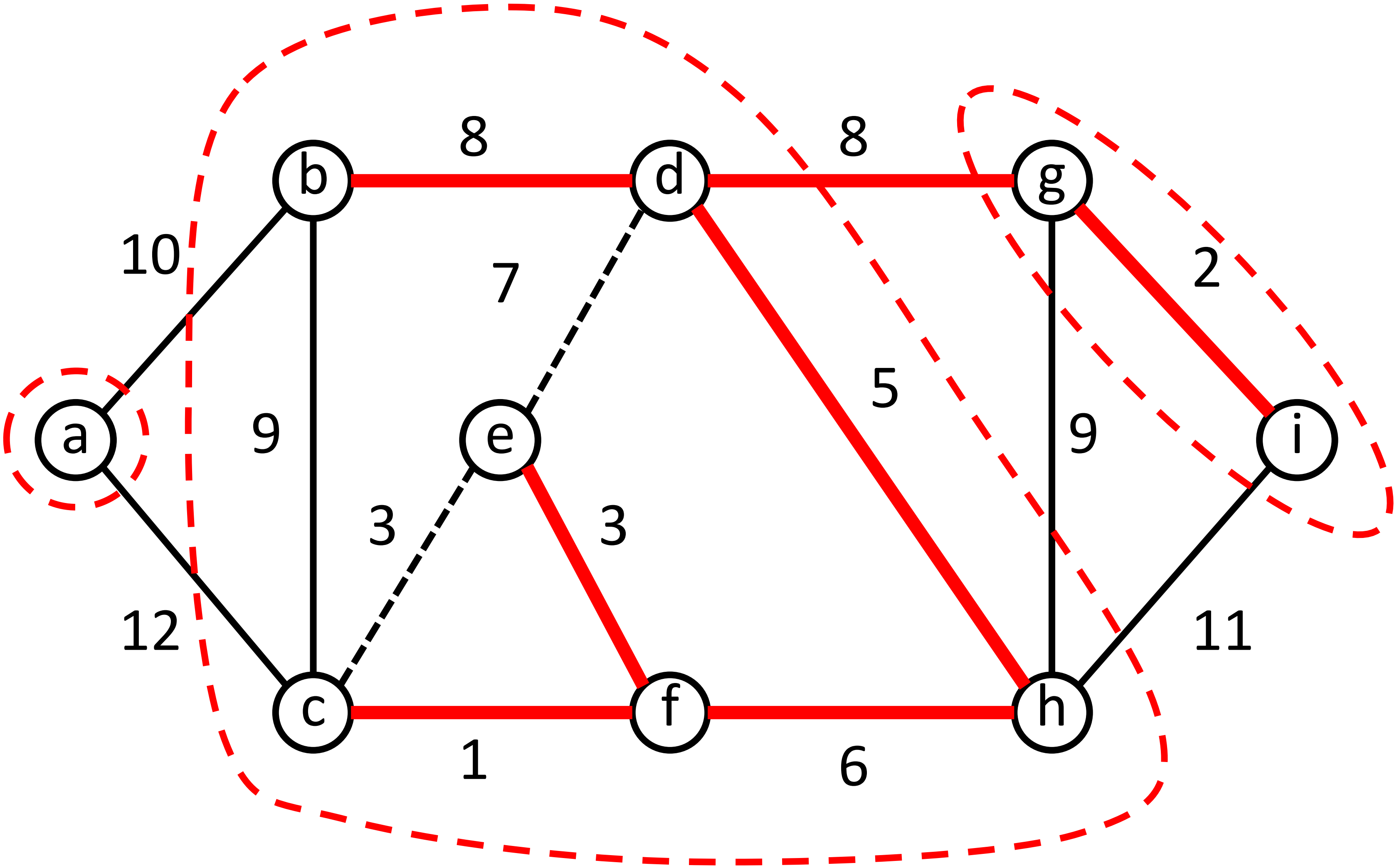
Example



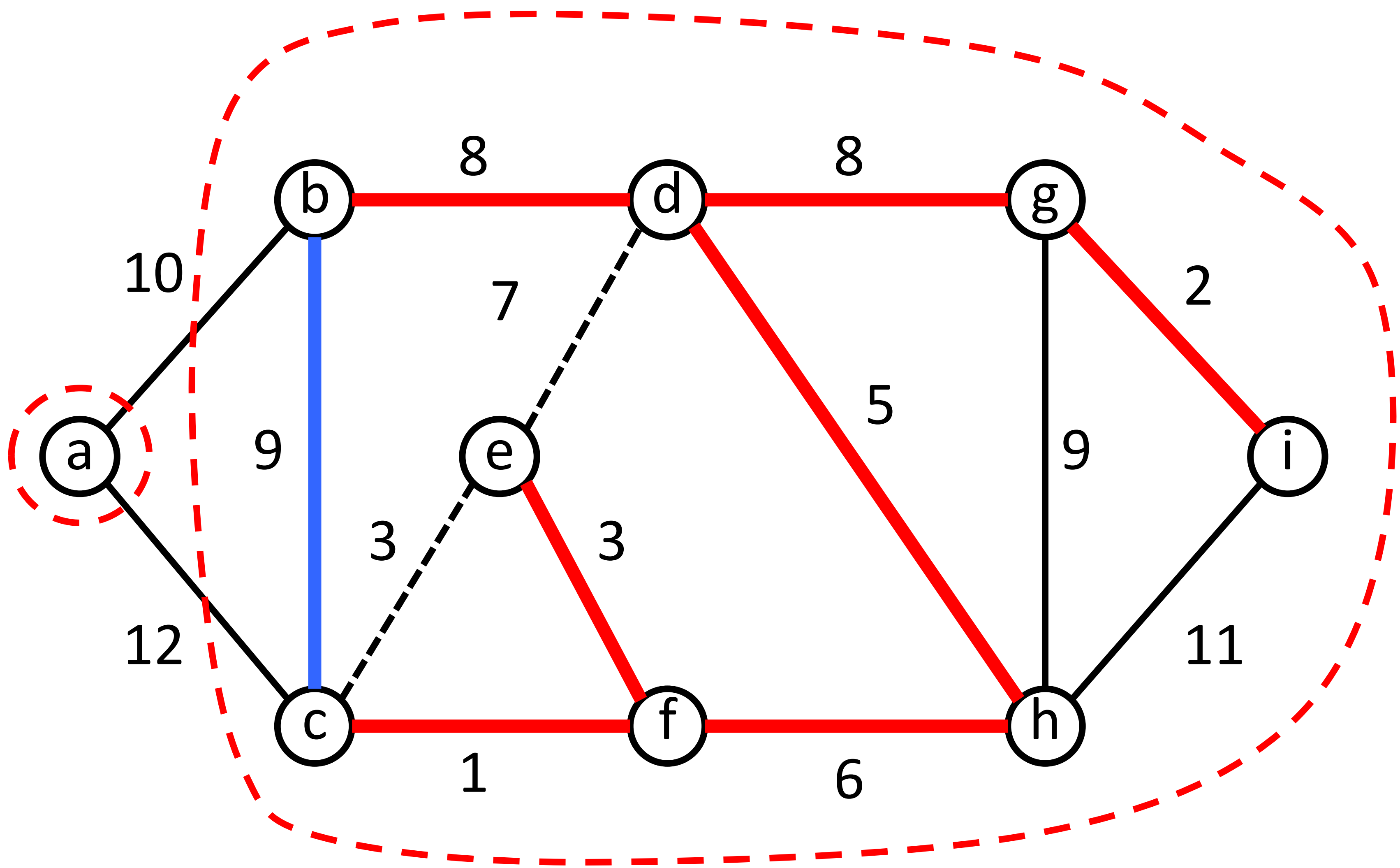
Example



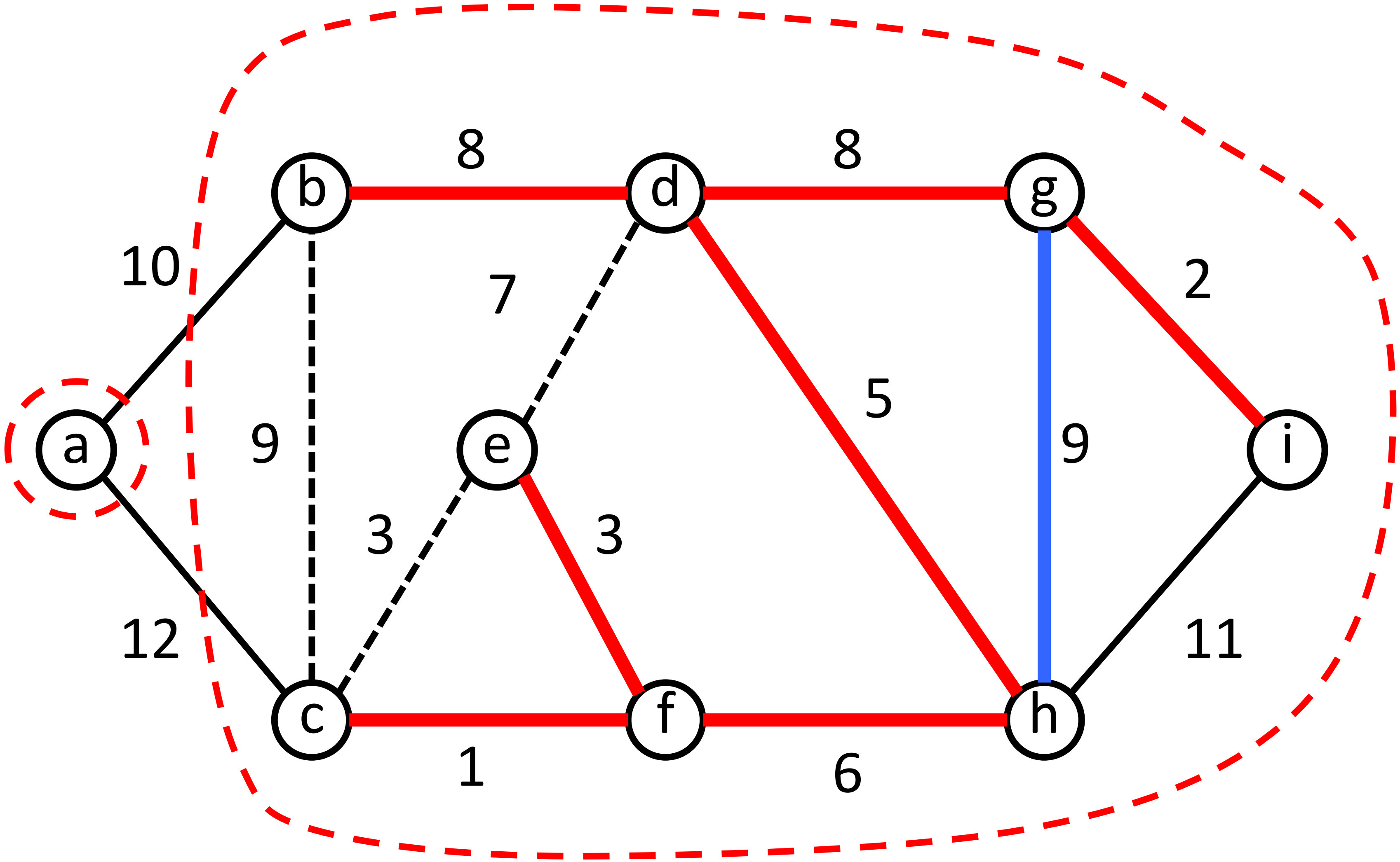
Example



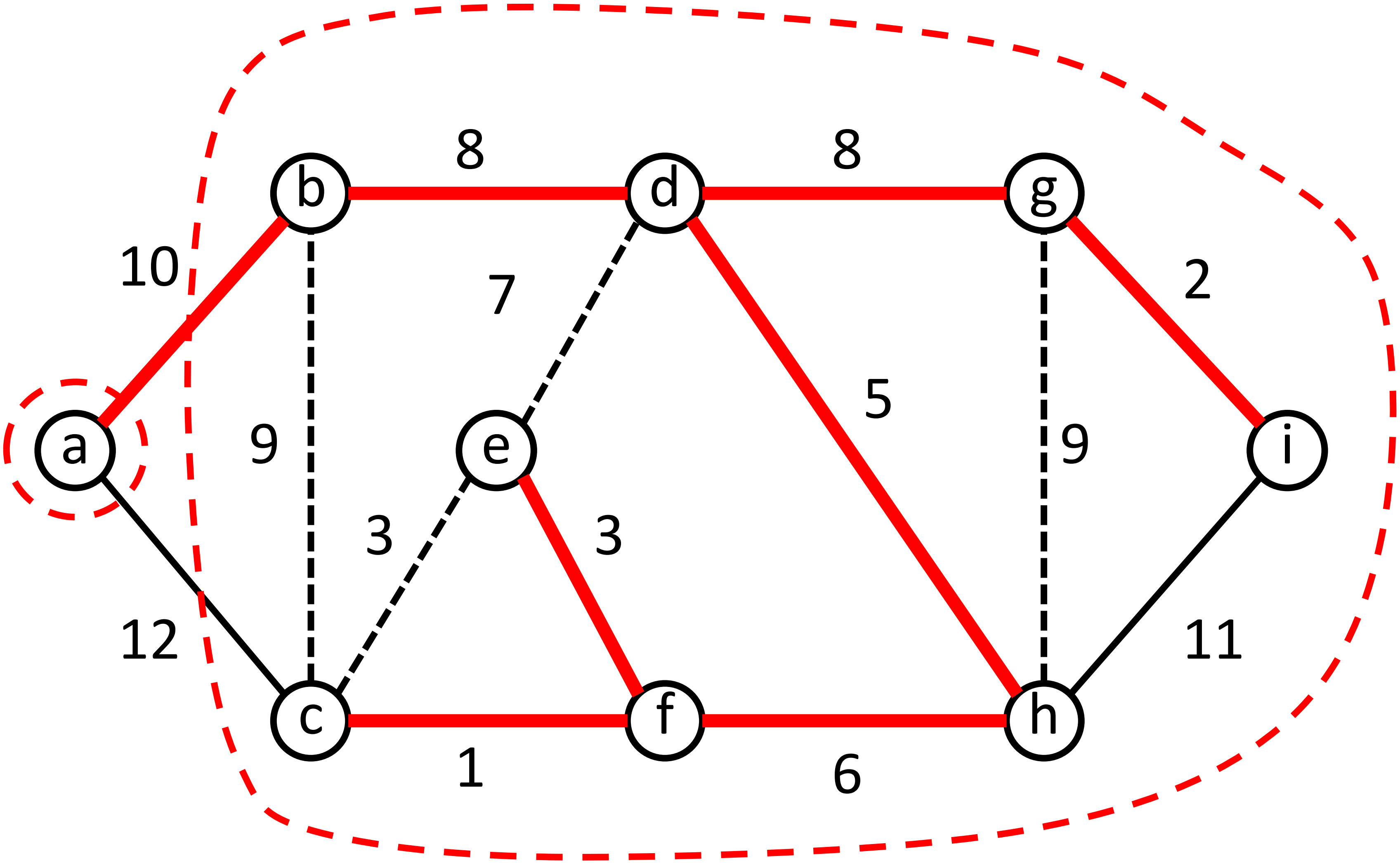
Example



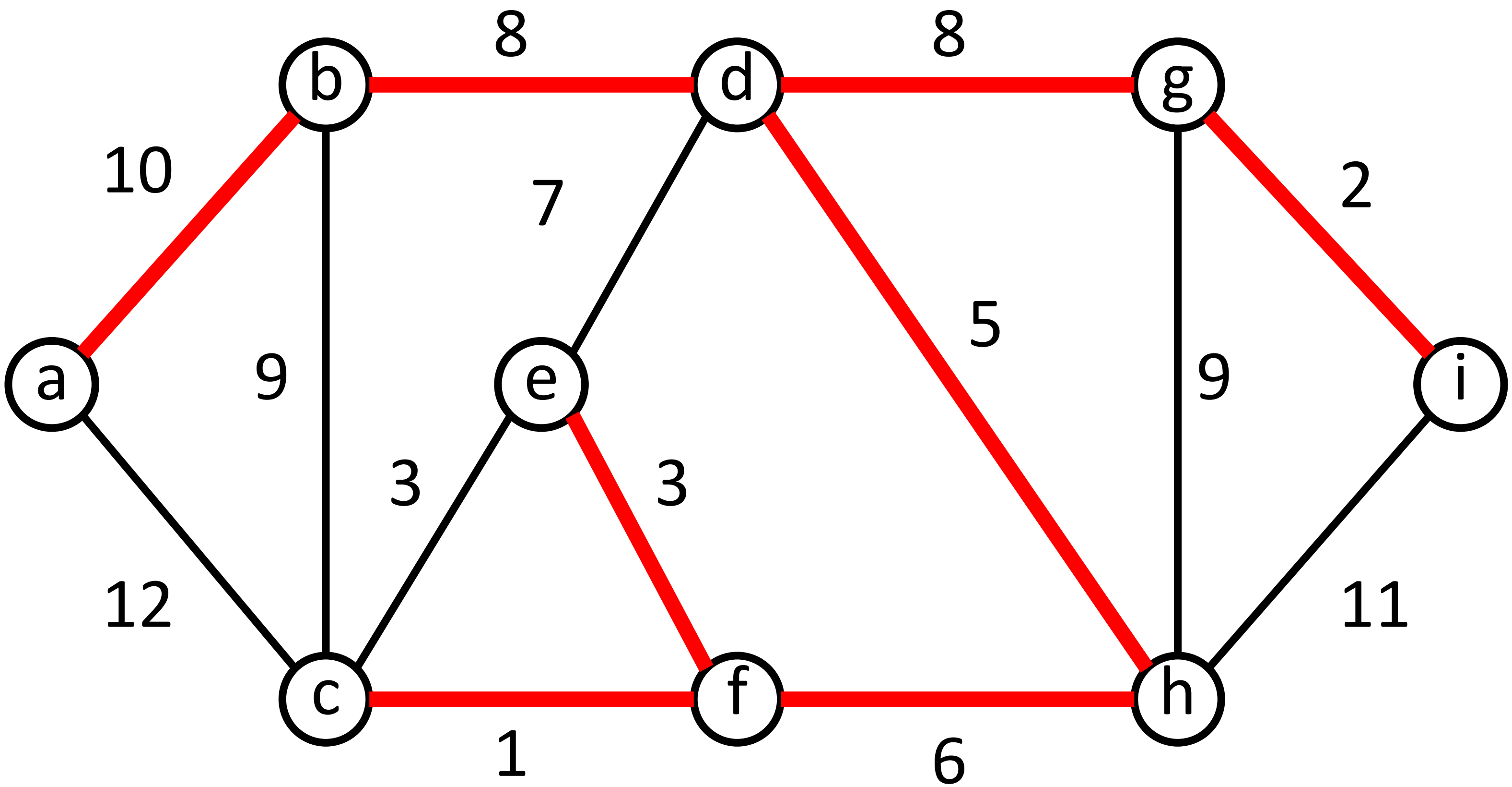
Example



Example



Example



Safe edge

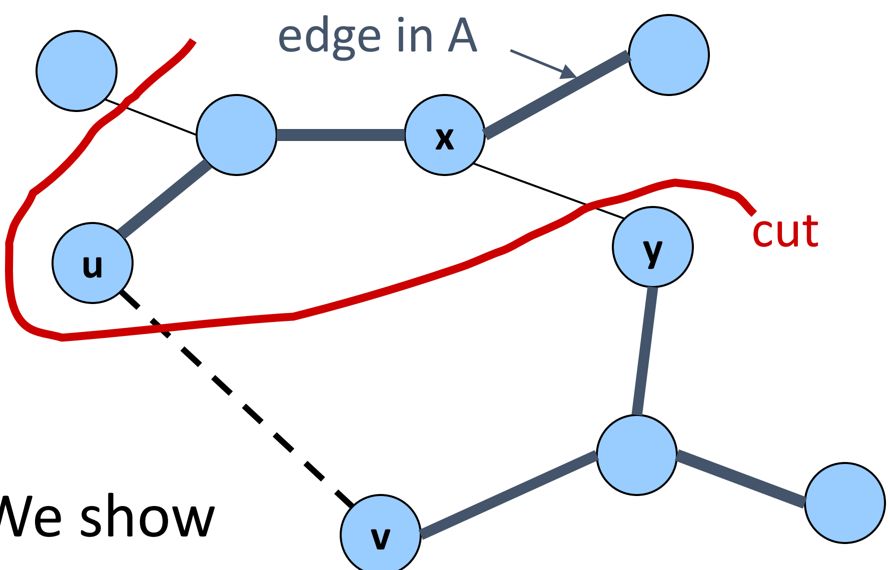
Theorem 1: Let (S, V-S) be any cut that respects A, and let (u, v) be a light edge crossing (S, V-S). Then, (u, v) is safe for A.

Proof:

Let T be a MST that includes A.

Case 1: (u, v) in T. We're done.

Case 2: (u, v) not in T. We have the following:



We show edges in T

(x, y) crosses cut.

Let T´ = T - {(x, y)} ∪ {(u, v)}.

Because (u, v) is light for cut, w(u, v) $\leq$ w(x, y). Thus,

w(T´) = w(T)-w(x, y)+w(u, v)$\leq$w(T).

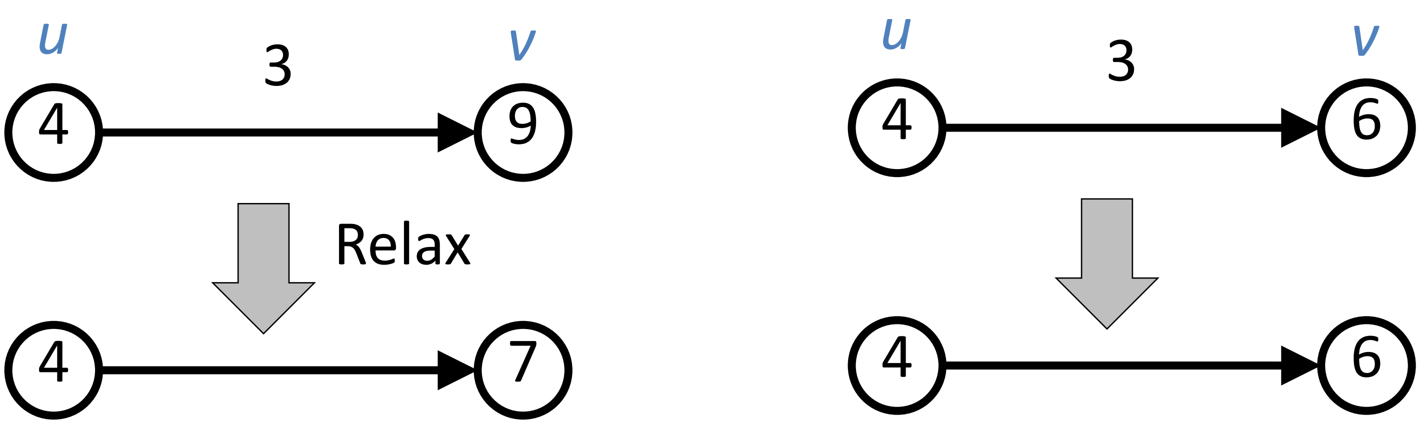
Hence, T´ is also a MST.

So, **(u, v) is safe for A**.

Single source shortest paths

Relaxing an edge

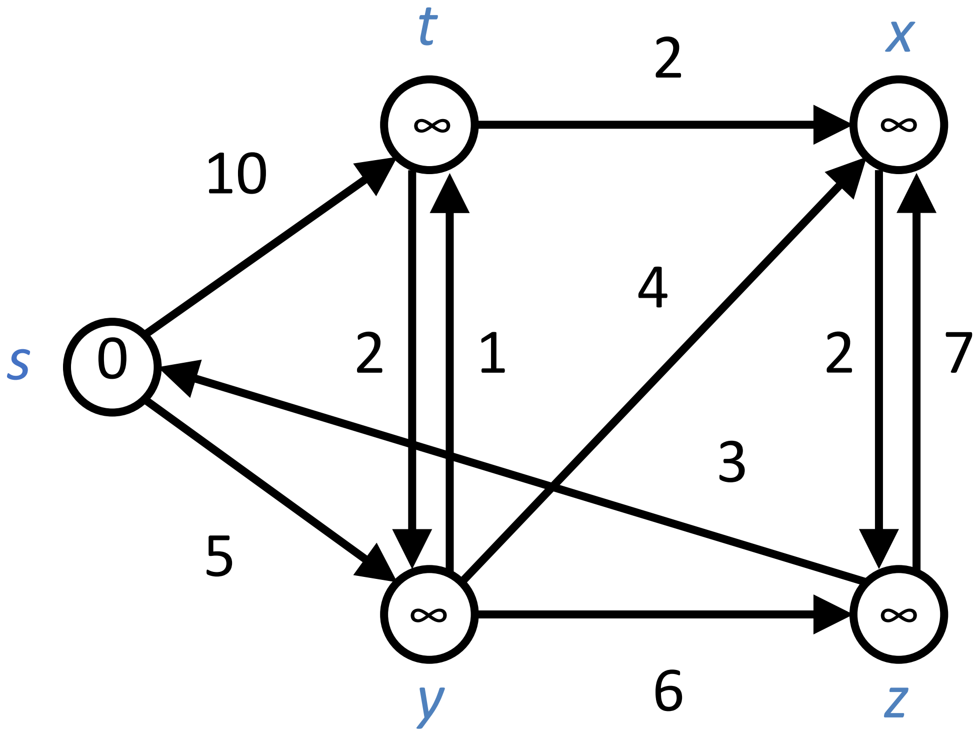
```
RELAX(u,v,w)
if d[v]>d[u]+w(u,v) then
   d[v] ← d[u]+w(u,v)
   π[v]←u
```



Dijkstra's algorithm

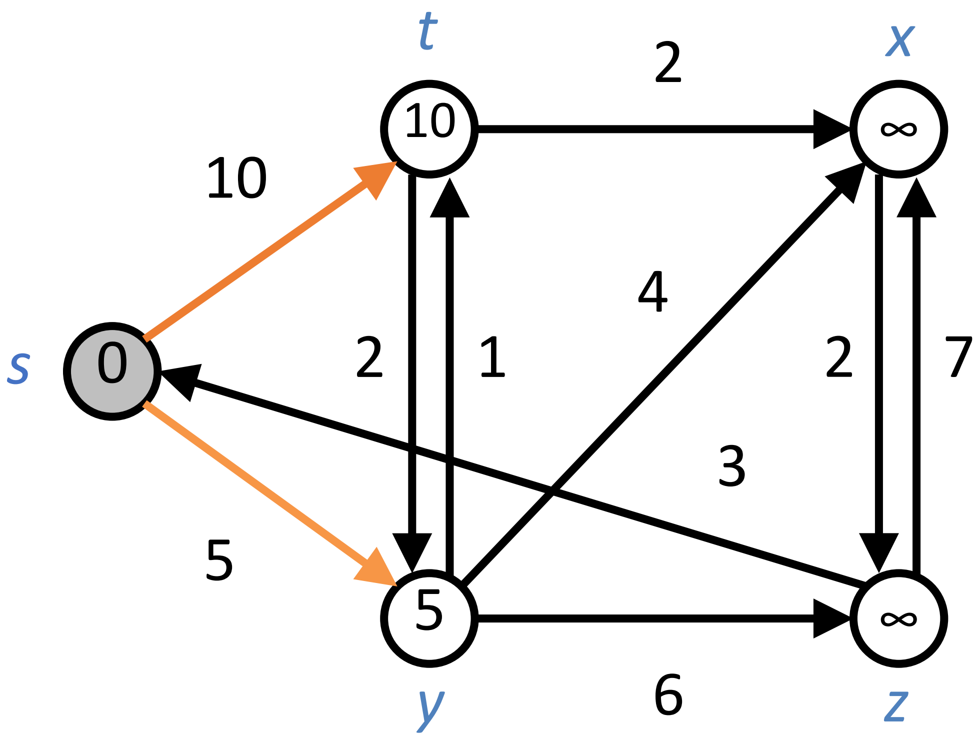
```
DIJKSTRA(V, E,w,s)
INIT-SINGLE-SOURCE(V,s)
S ← ∅
Q ← V
while Q ≠ ∅ do
   u ← EXTRACT-MIN(Q)
   S ← S ∪ {u}
   for each vertex v ∈ Adj[u] do
      RELAX(u,v,w)
```

Example



Q	s	t	y	x	z

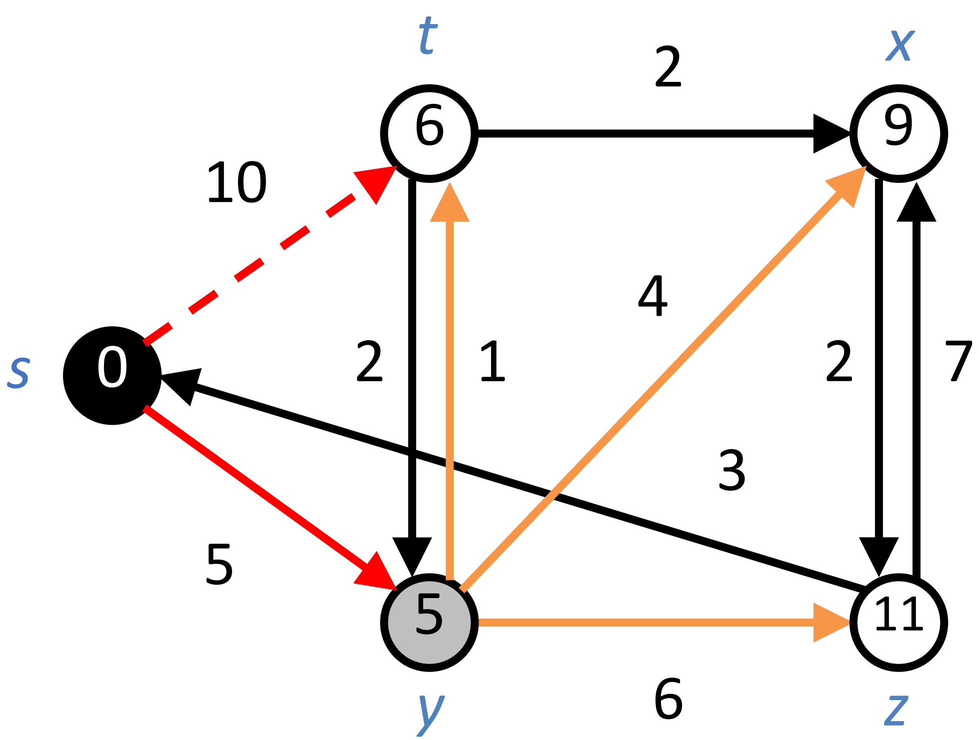
Example



Q

y	t	x	z	

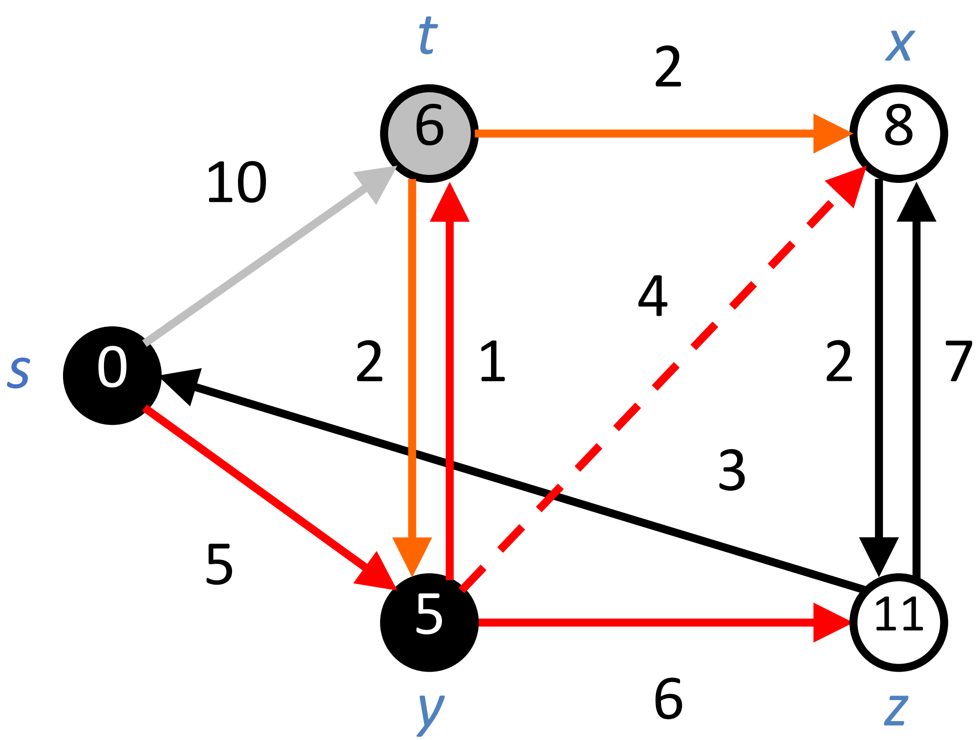
Example



Q

t	x	z		

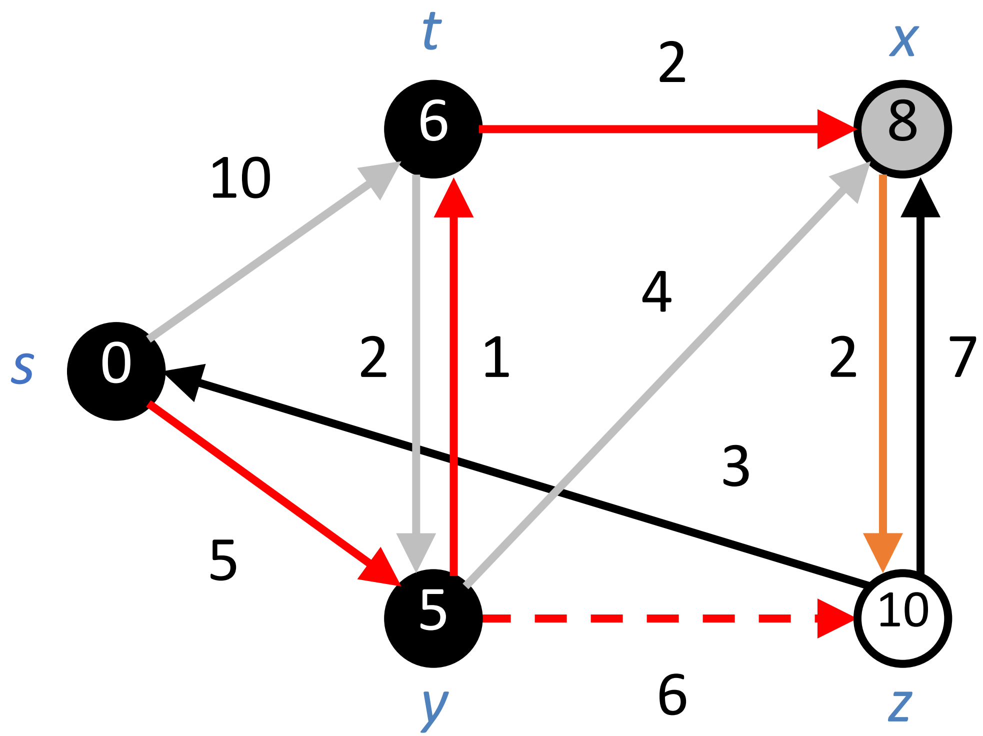
Example



Q

x	z			

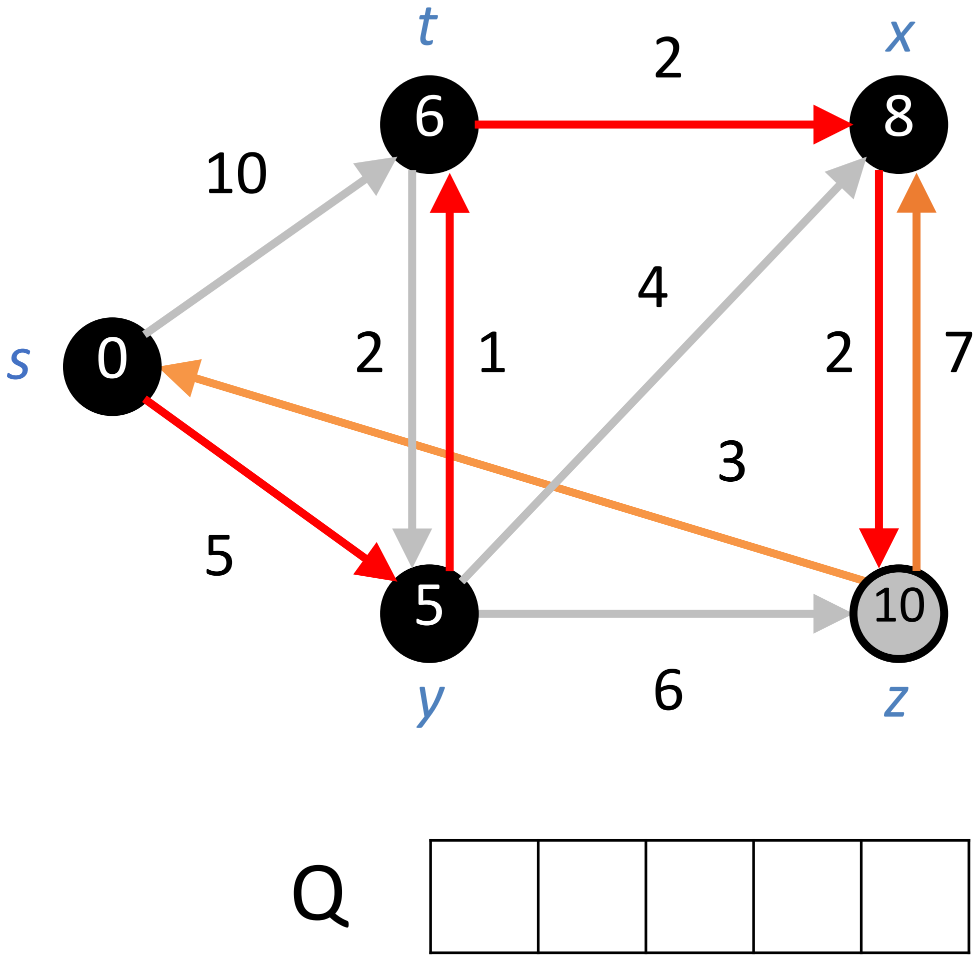
Example



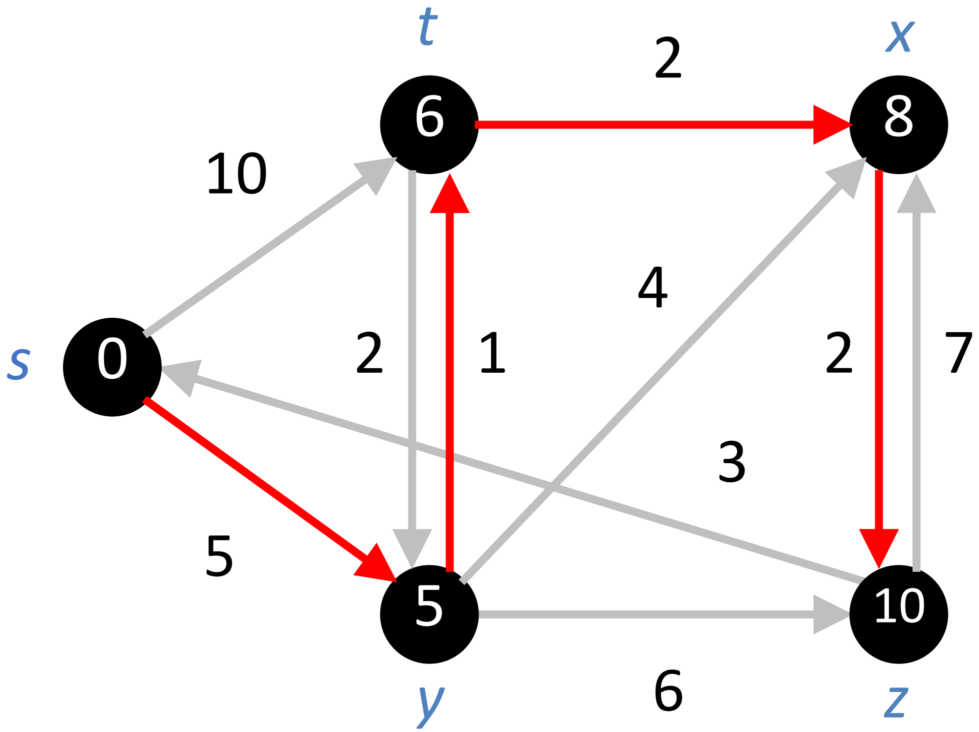
Q

z				

Example



Example



Dijkstra's algorithm

- Variable used to calculate shortest path: *d*
- Property used to calculate shortest path: $d[v] = \delta(s,v)$

```
DIJKSTRA(V, E,w,s):
INIT-SINGLE-SOURCE(V,s)
S ← ∅
Q ← V
while Q ≠ ∅ do
   u ← EXTRACT-MIN(Q)
   S ← S ∪ {u}
   for each vertex v∈Adj[u] do
      RELAX(u,v,w)
```

Initialization
Initially, $S = \varnothing \Longrightarrow$ True

Loop Invariant:
At the start of each iteration:
$d[v] = \delta(s,v)\ \forall v \in S$.

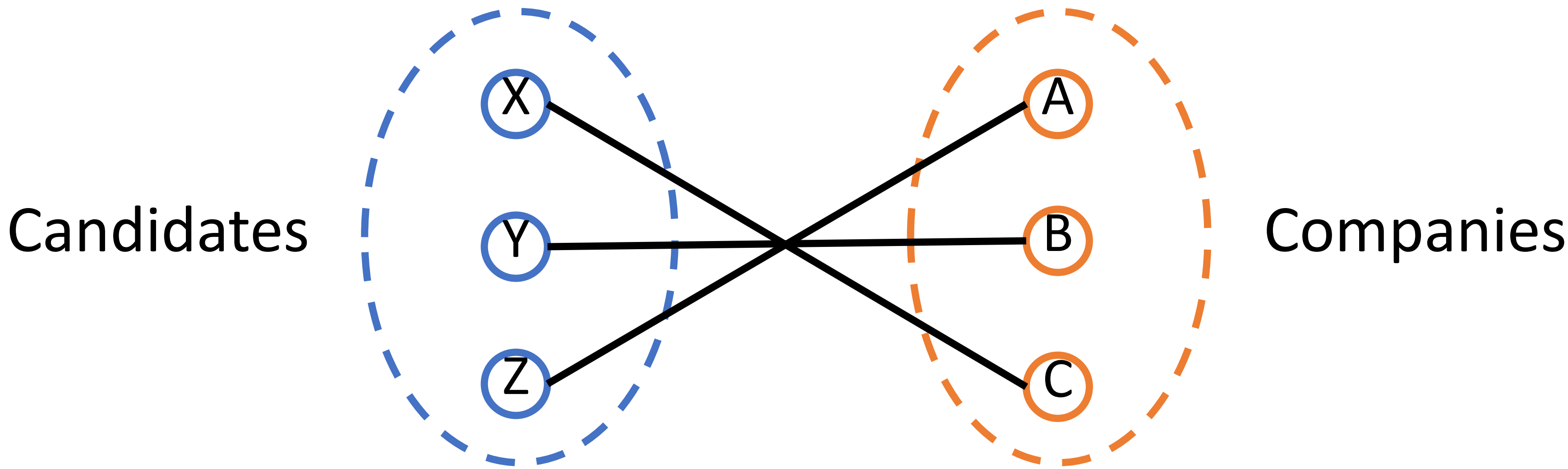
Termination:
Stops when $Q=\varnothing \Rightarrow$
$d[v] = \delta(s,v)\ \forall\ v \in V$
(by Loop Invariant Property)

Maintenance:
Show that $d[u] = \delta(s,u)$ when
u is added to S in each
Iteration.

Bipartite graphs

Example

Q: Is X-C, Y-B, Z-A a good assignment?



Candidates' preferences

	1st	2nd	3rd
Xavier	Alphabet	Baidu	Campbell
Yulia	Baidu	Alphabet	Campbell
Zoran	Alphabet	Baidu	Campbell

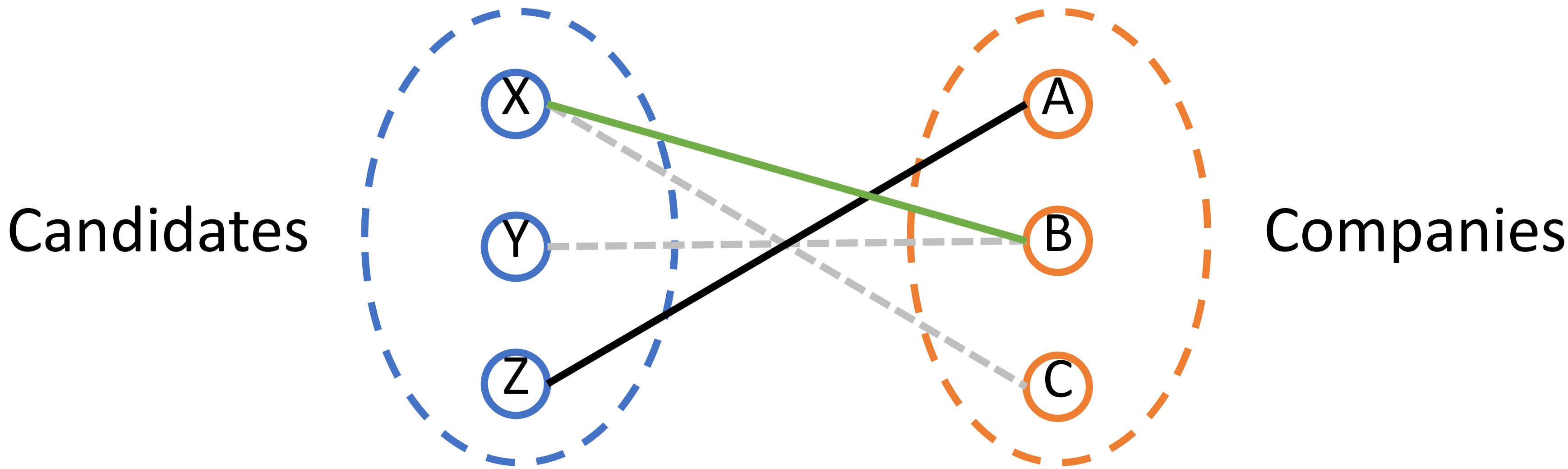
Companies' preferences

	1st	2nd	3rd
Alphabet	Yulia	Xavier	Zoran
Baidu	Xavier	Yulia	Zoran
Campbell	Xavier	Yulia	Zoran

Example

Q: Is X-C, Y-B, Z-A a good assignment?

A: No! Xavier and Baidu both prefer to be matched together…



Candidates' preferences

	1st	2nd	3rd
Xavier	Alphabet	Baidu	Campbell
Yulia	Baidu	Alphabet	Campbell
Zoran	Alphabet	Baidu	Campbell

Companies' preferences

	1st	2nd	3rd
Alphabet	Yulia	Xavier	Zoran
Baidu	Xavier	Yulia	Zoran
Campbell	Xavier	Yulia	Zoran

Gale-Shapley algorithm

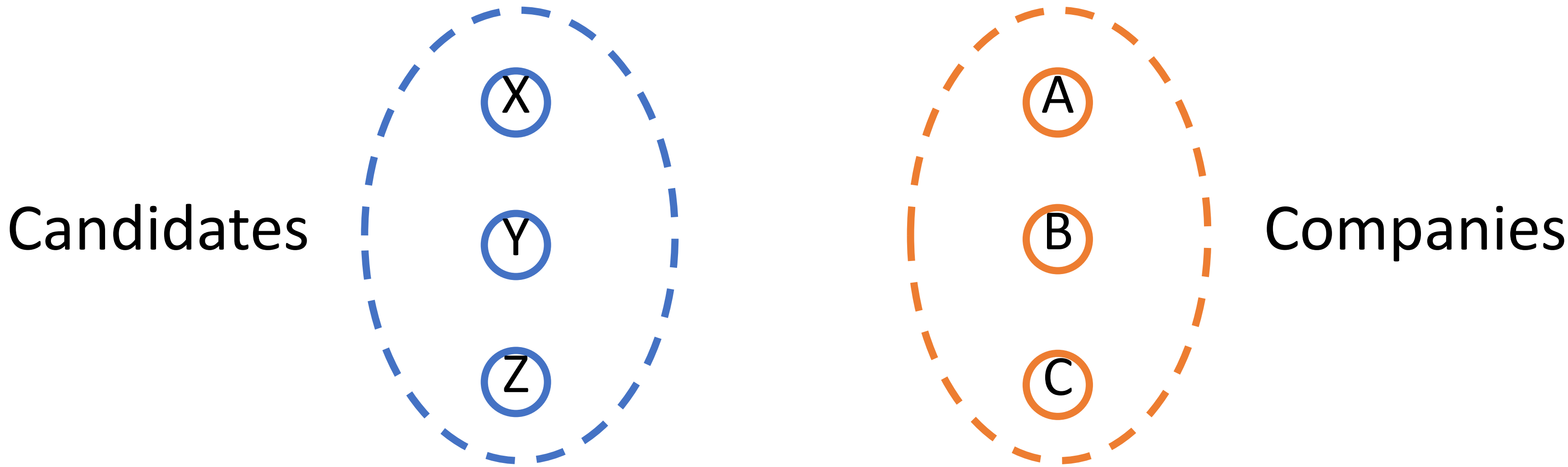
For each α∈A, let pref[α] be the ordering of its preferences in B

For each β∈B, let pref[β] be the ordering of its preferences in A

Let matching be a set of crossing edges between A and B

`matching`←∅
while there is α∈A not yet matched **do**
  β←pref[α].removeFirst()
  if β not yet matched **then**
    `matching`←`matching`∪{(α,β)}
  else
    γ←β's current match
    if β prefers α over γ **then**
      `matching`←`matching`-{(γ,β)}∪{(α,β)}
return `matching`

Example



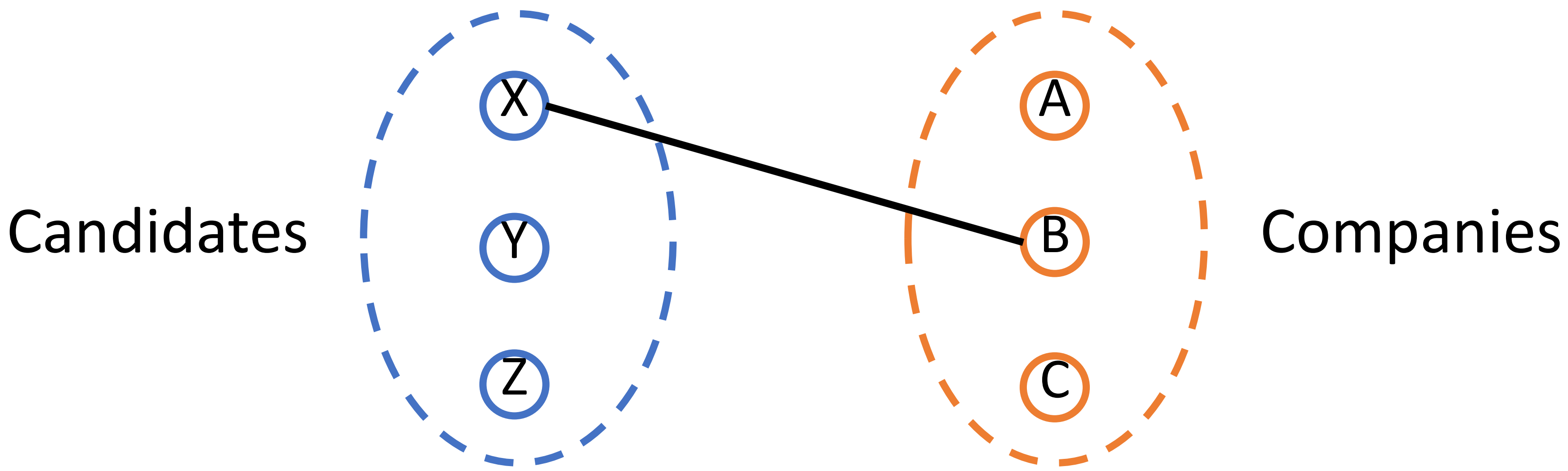
Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



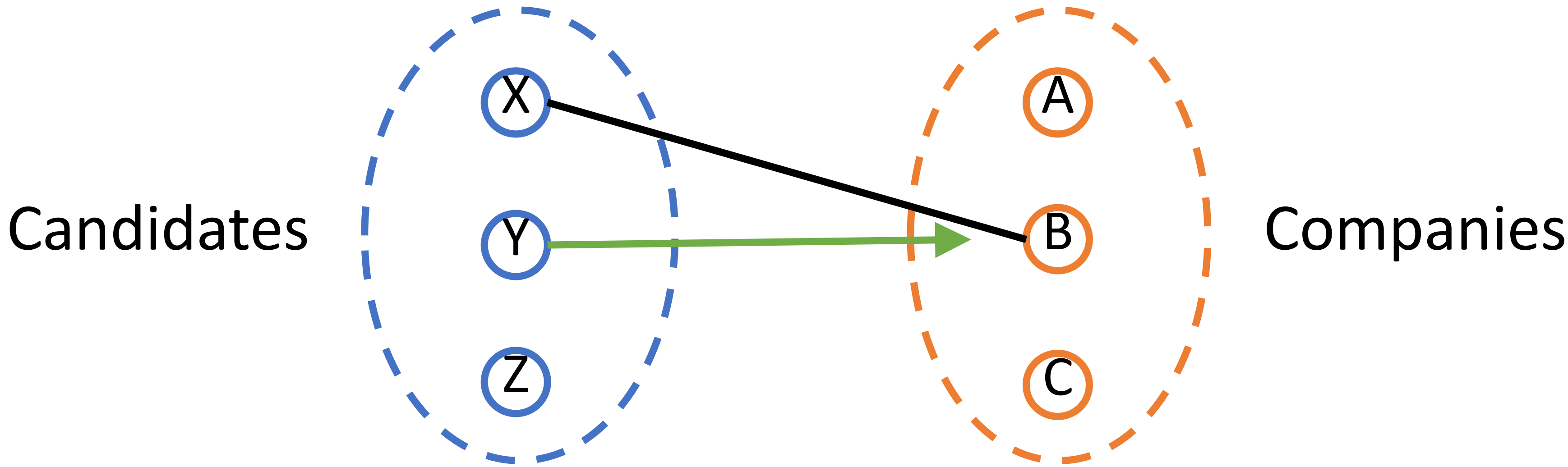
Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



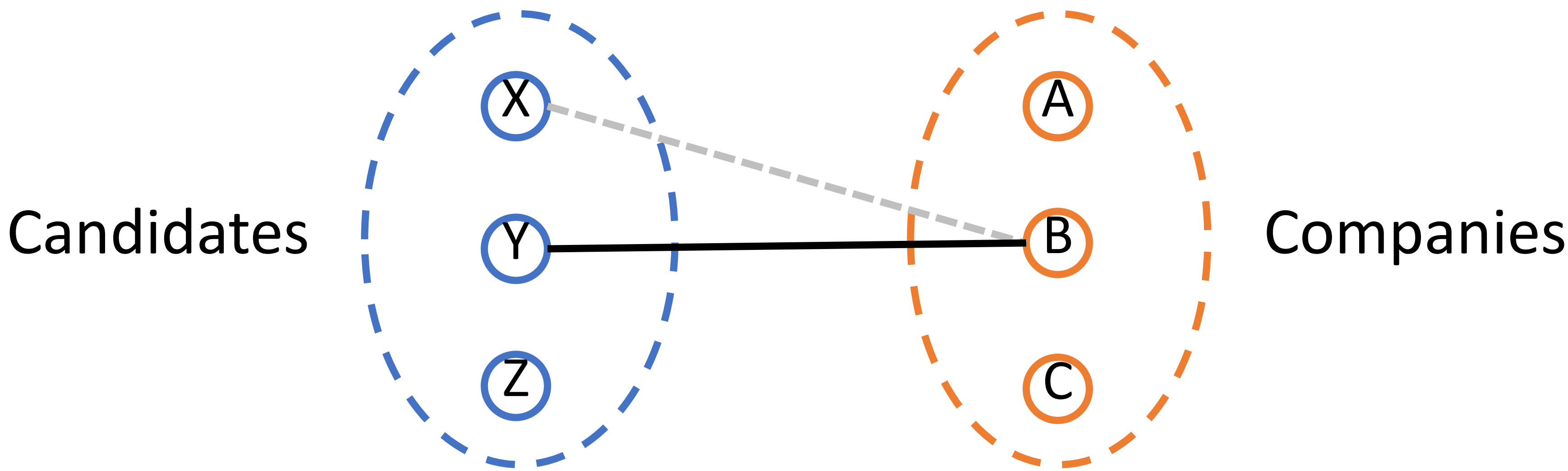
Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



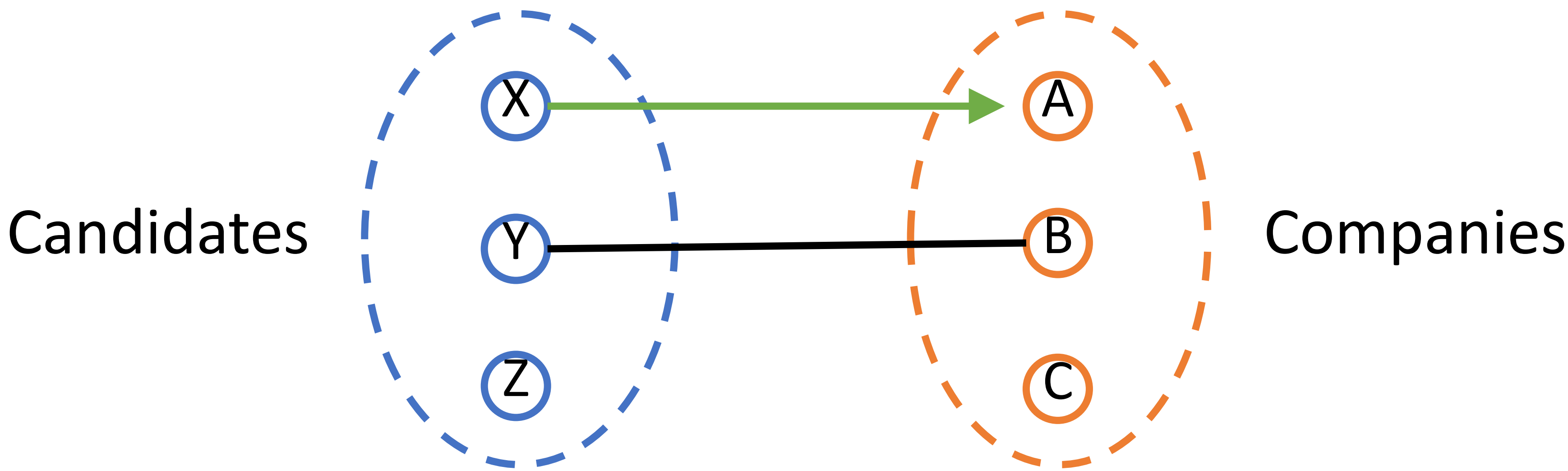
Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



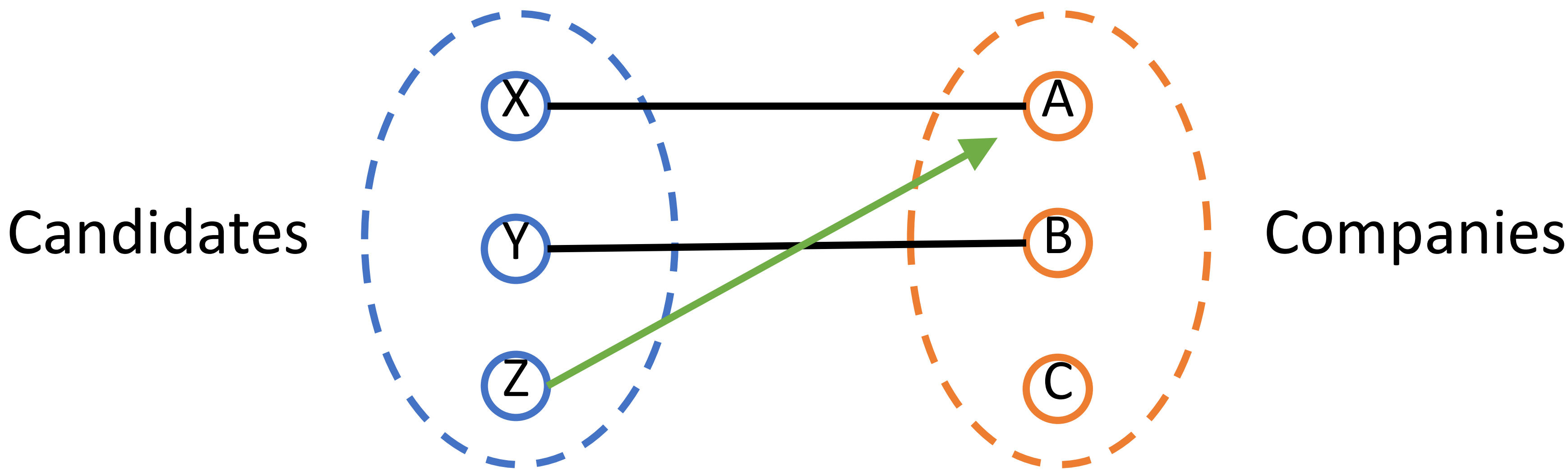
Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



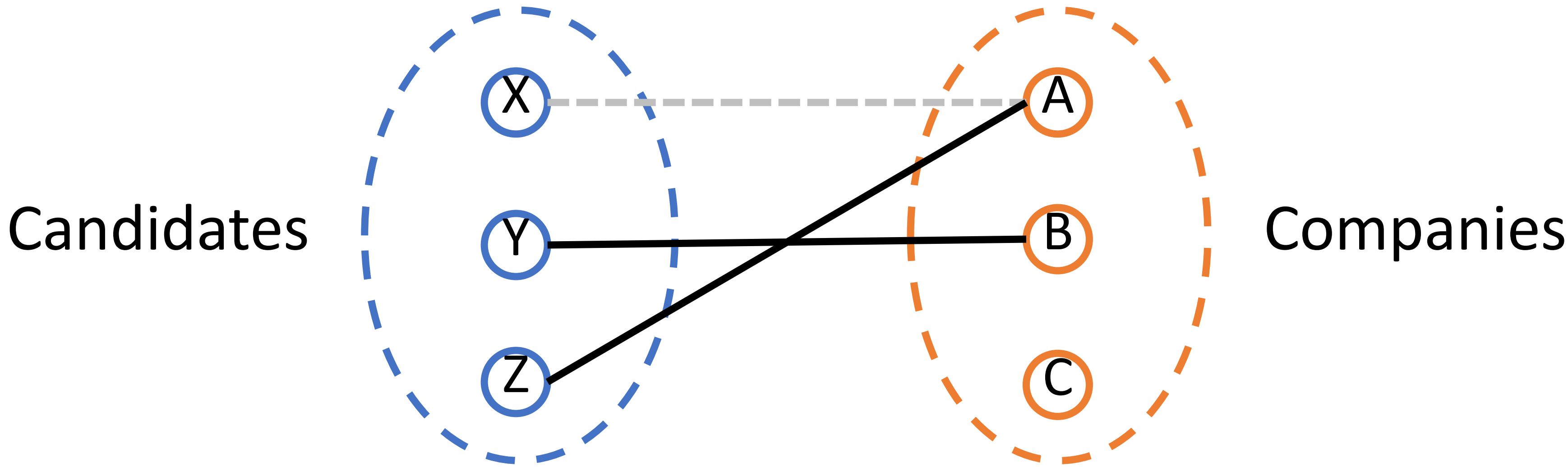
Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



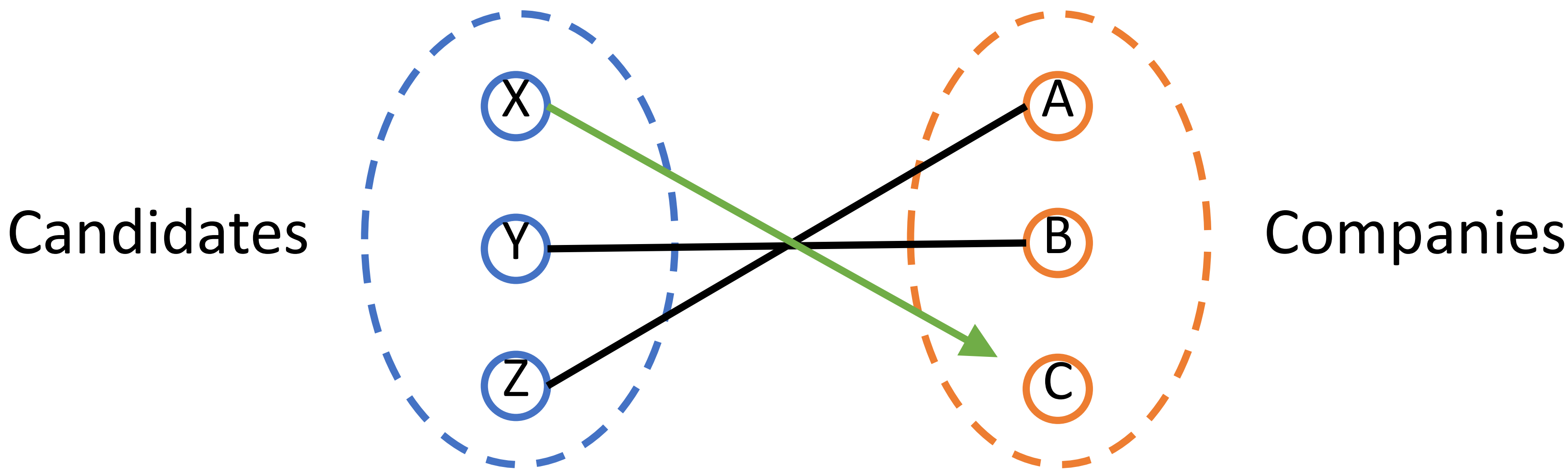
Men's preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



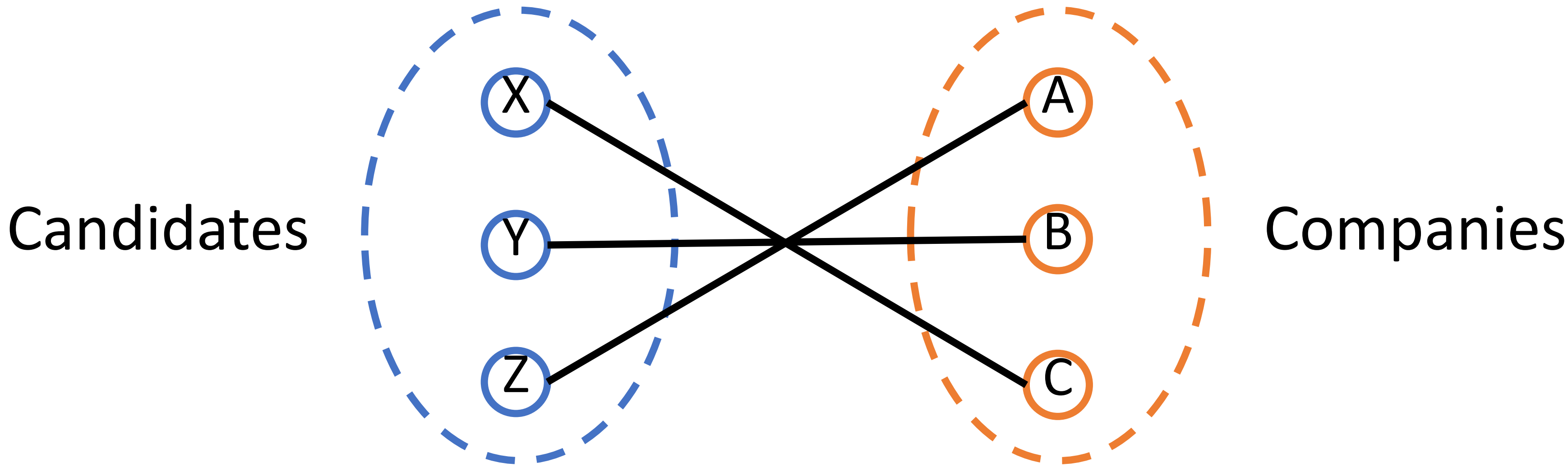
Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Example



Candidates' preferences

	1st	2nd	3rd
Xavier	Baidu	Alphabet	Campbell
Yulia	Baidu	Campbell	Alphabet
Zoran	Alphabet	Campbell	Baidu

Companies' preferences

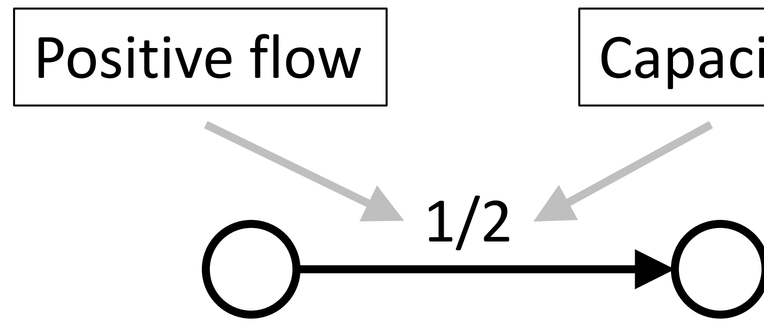
	1st	2nd	3rd
Alphabet	Zoran	Xavier	Yulia
Baidu	Yulia	Zoran	Xavier
Campbell	Xavier	Yulia	Zoran

Flow networks

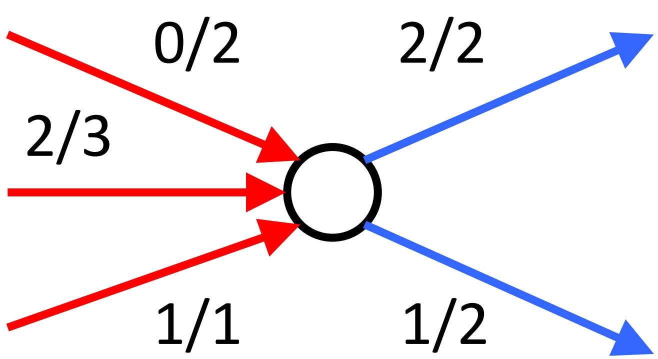
Definitions

Positive flow: A function p : V × V → **R** satisfying.

Capacity constraint: For all u, v ∈ V, 0 ≤ p(u, v) ≤ c(u, v), ⍰⍰



Flow conservation: For all u ∈ V – {s, t}, $\underbrace{\sum_{v \in V} p(v,u)}_{\text{Flow into u}} = \underbrace{\sum_{v \in V} p(u,v)}_{\text{Flow out of u}}$

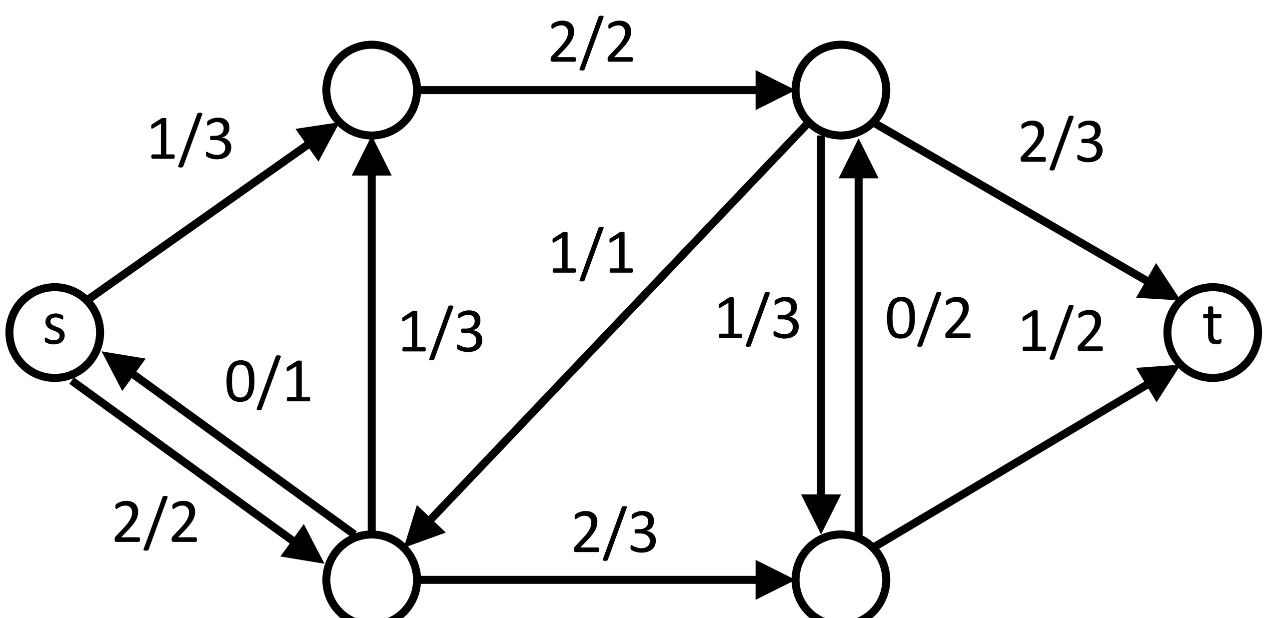


Flow in: 0 + 2 + 1 = 3

Flow out: 2 + 1 = 3

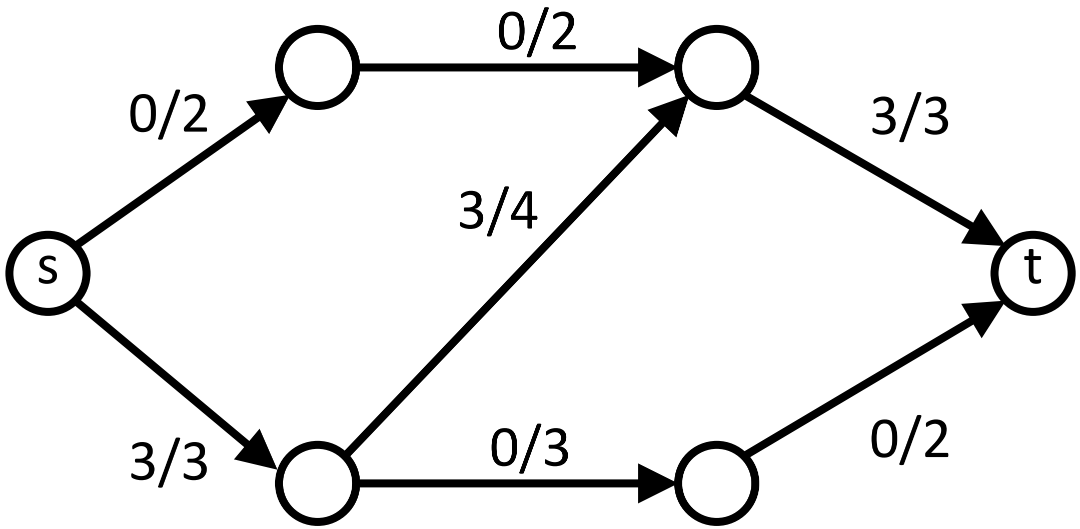
Max flow

- Flow out of source *s* == Flow in the sink *t*
- Flow = $\sum_{v \in V} f(s,v)$
- **Objective:** find maximum flow

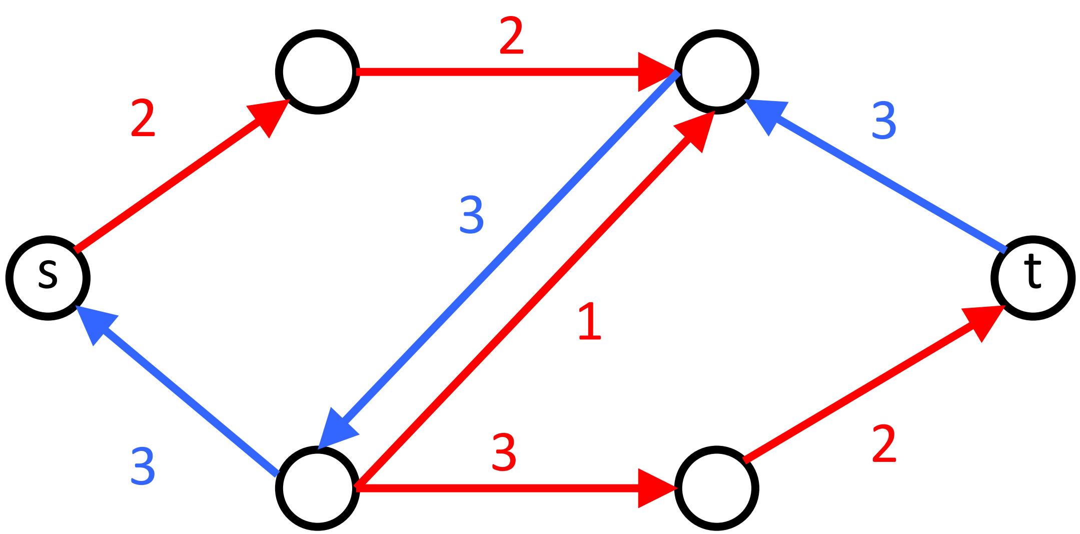


Example

Flow in G

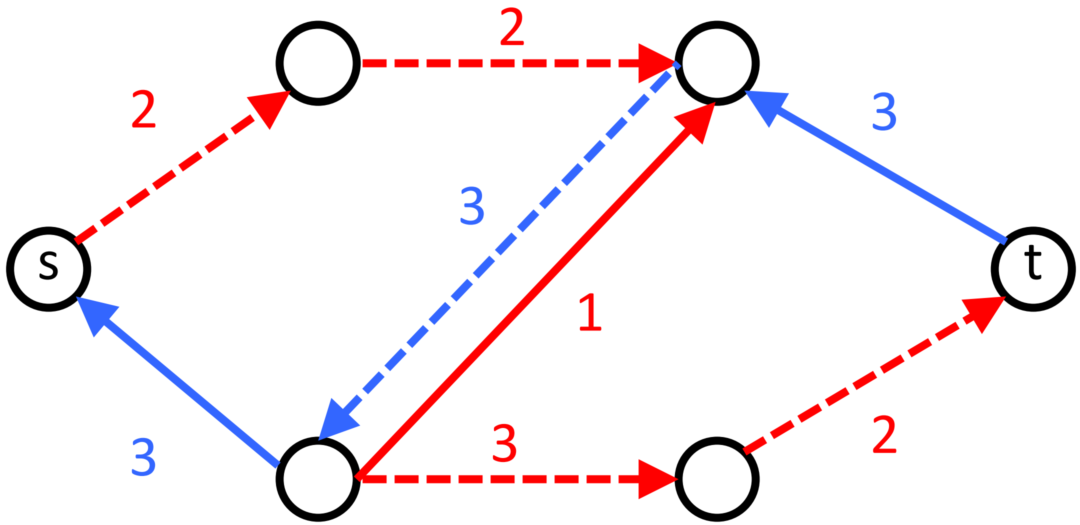


Residual graph G_f

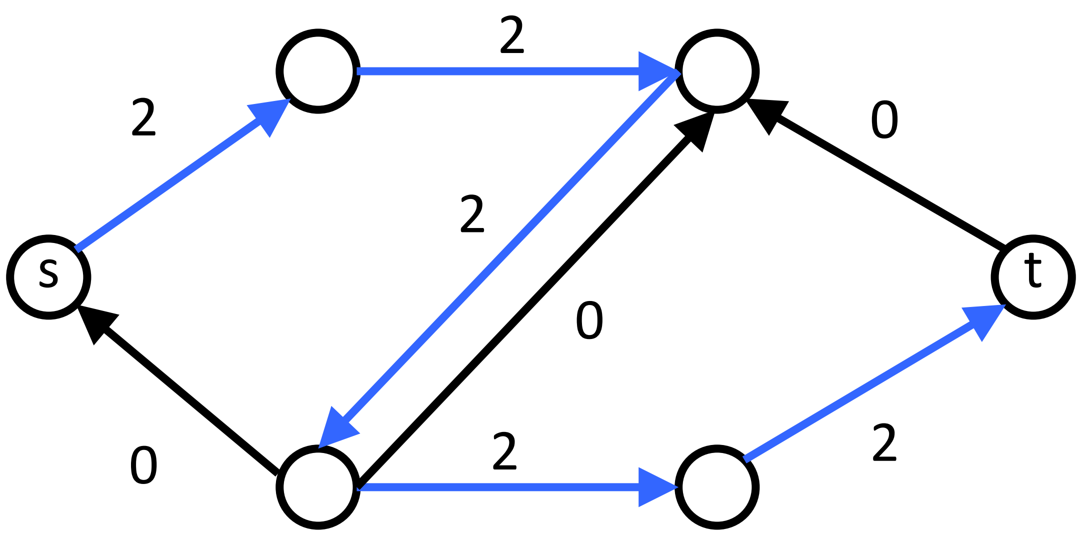


Example

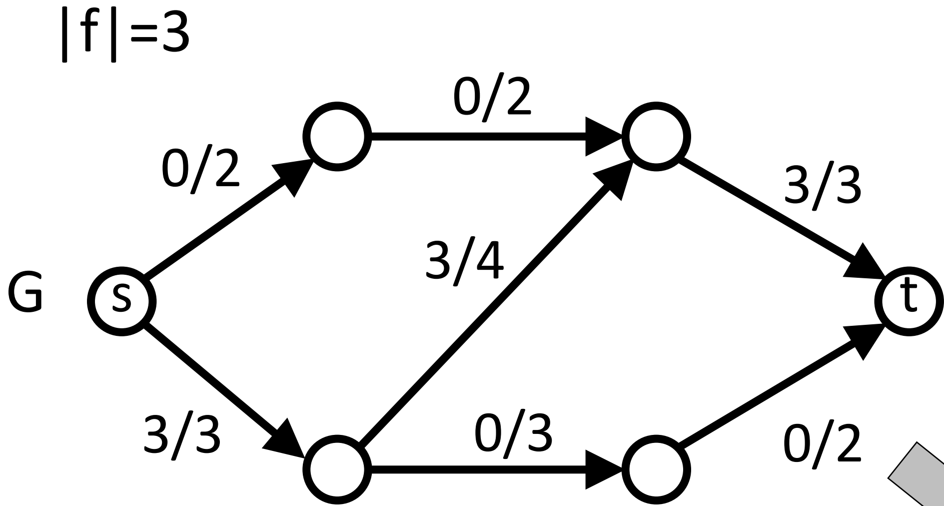
Residual graph G_f

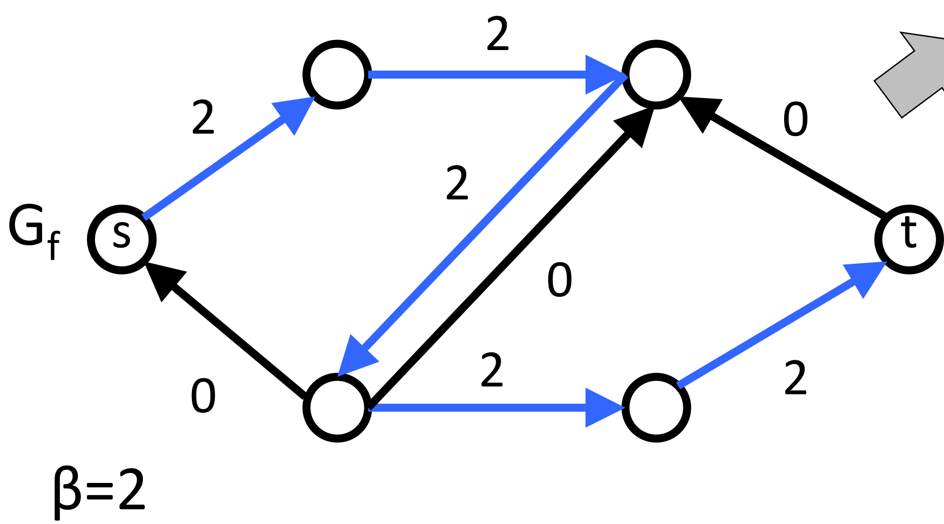


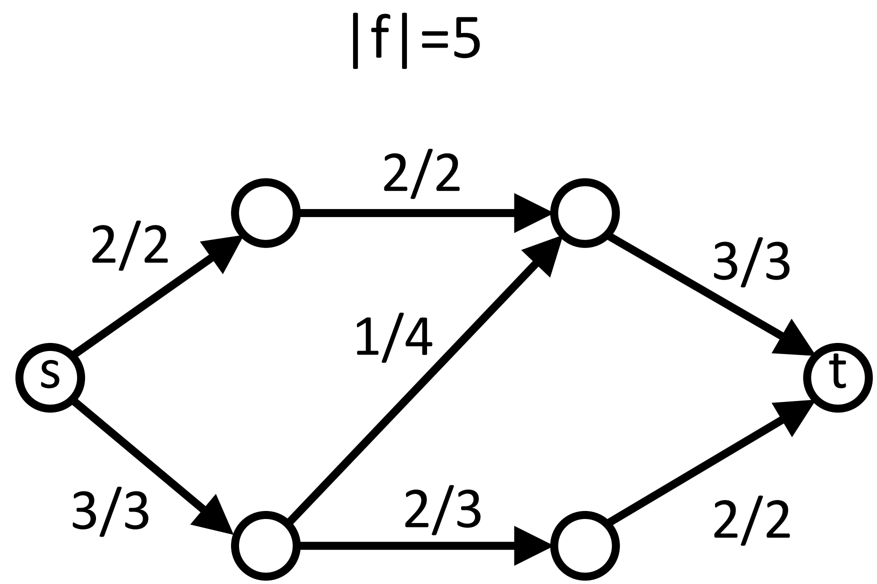
Flow in G_f



Example







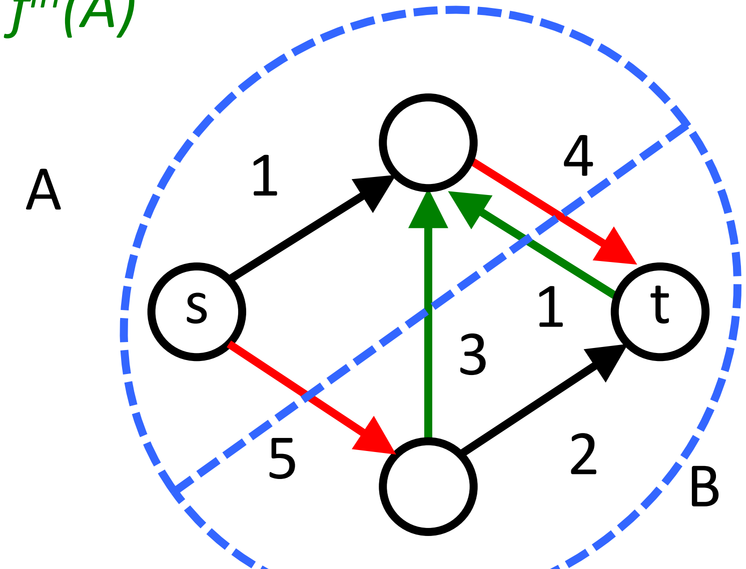
Question: How do we know if a flow is maximum?

Flow through a cut

Claim: Given a flow network. Let f be a flow and A, B be a s-t cut. Then,

$$|f| = \sum_{e \in cut(A,B)} f(e) - \sum_{e \in cut(B,A)} f(e)$$

Notation: $|f| = f^{out}(A) - f^{in}(A)$



$|f| = 9 - 4 = 5$

Max flow = Min cut

- Ford-Fulkerson terminates when there is no path s-t in the residual graph G_f
- The set of nodes accessible from *s* in G_f defines a cut in G

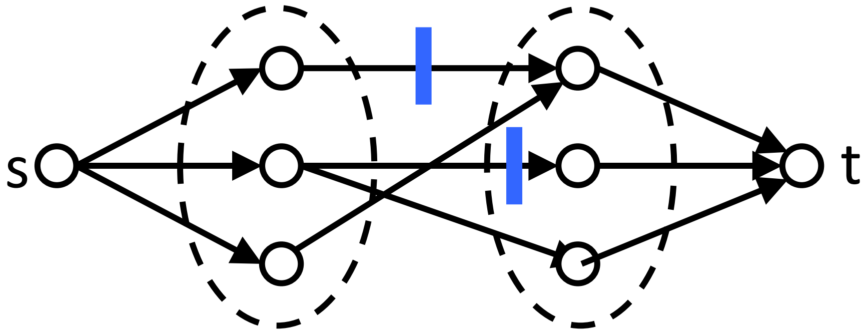
$$\begin{aligned} |f| &= f^{out}(A) - f^{in}(A) \\ &= \sum_{e \in cut(A,B)} f(e) - \sum_{e \in cut(B,A)} f(e) \end{aligned}$$

- Ford-Fulkerson flow = $\sum_{e \in cut(A,B)} c(e) - 0$

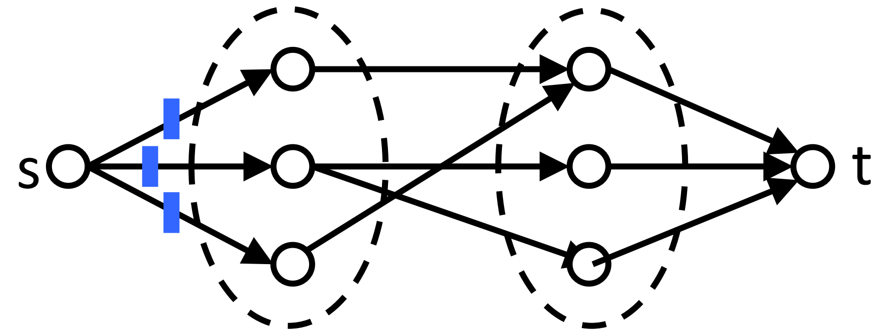
  = capacity of cut(A,B)

Example: Min Cut

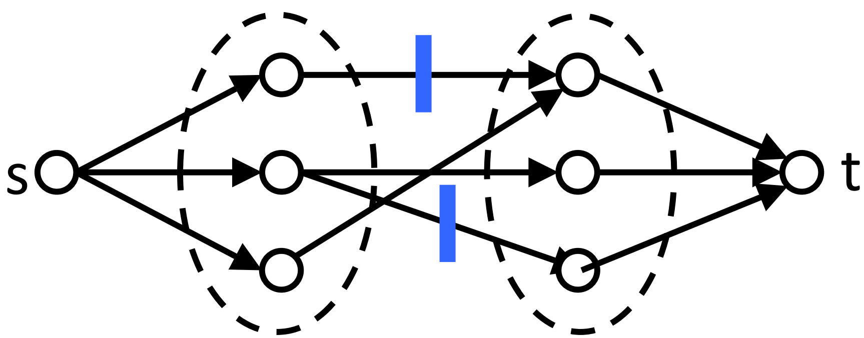
Note: All edges have a capacity of 1.



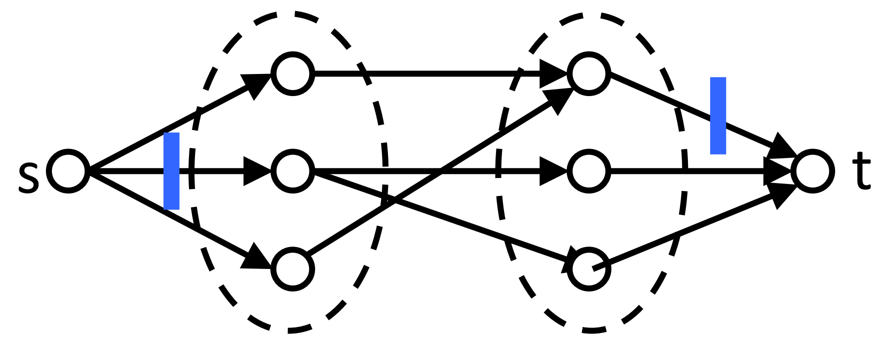
Not a cut!



Not a min cut!



Not a cut!



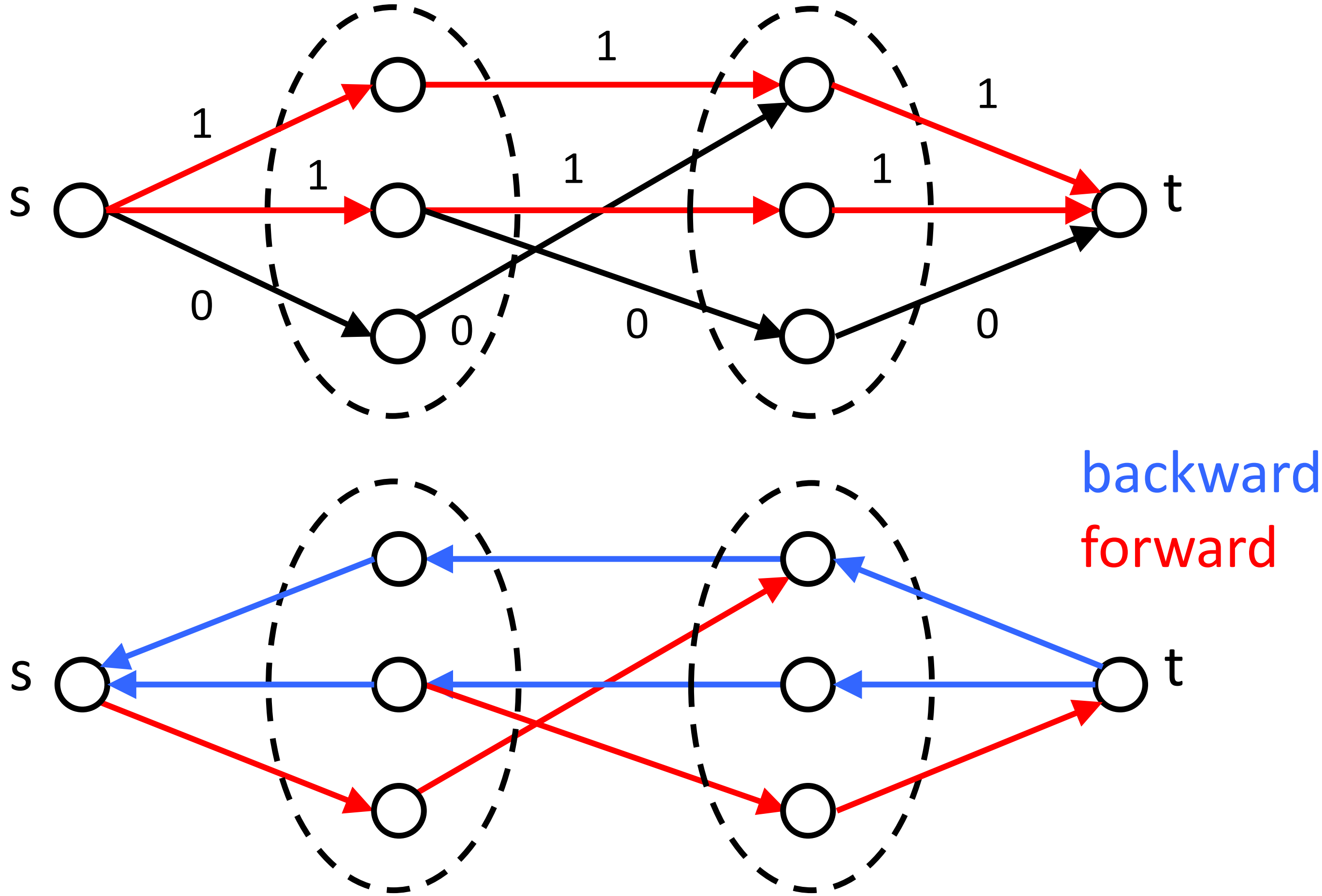
min cut!

Example: Calculate Min Cut

To find a min cut compute a max flow.

Flow

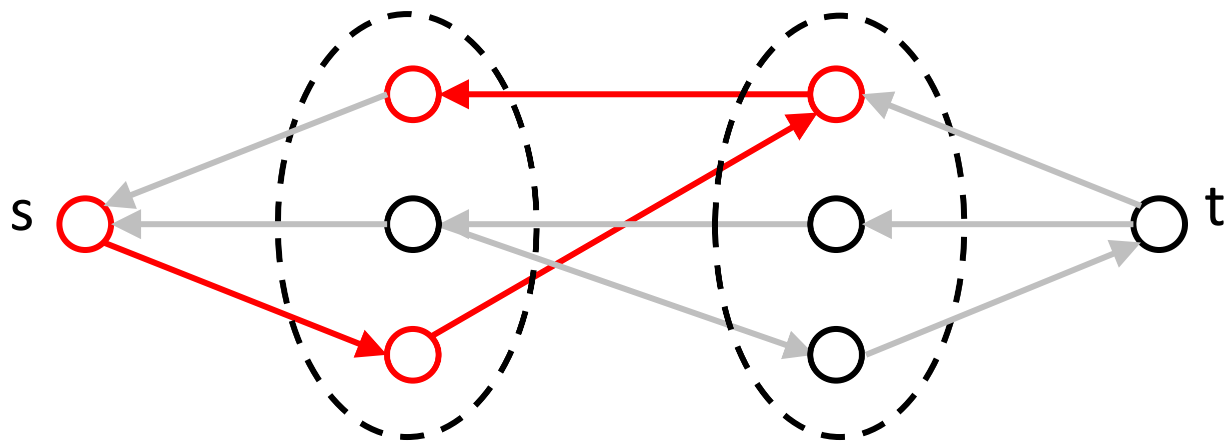
Residual graph



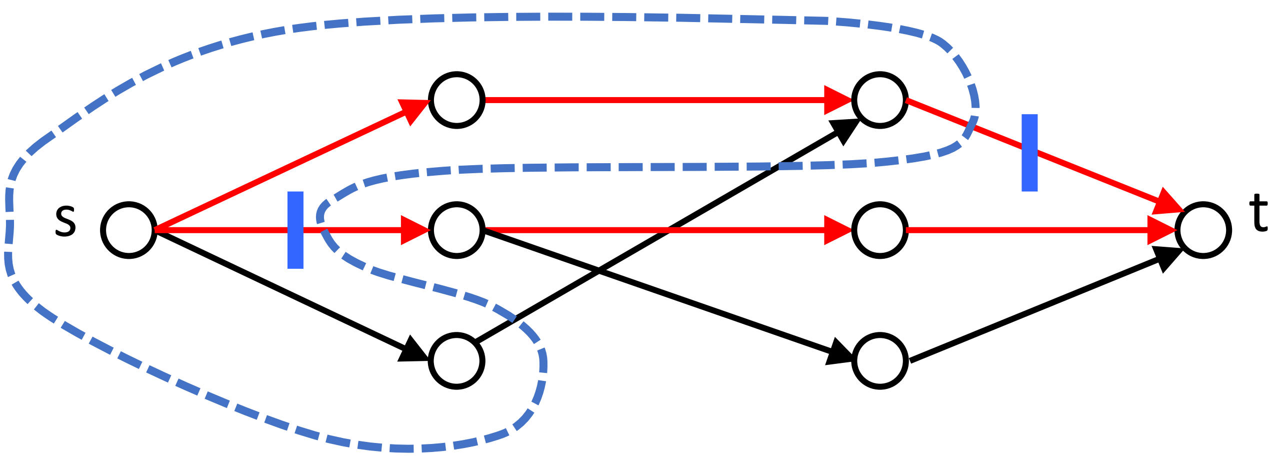
Example: Calculate Min Cut

To find the cut run BFS (or DFS) from s on the residual graph.
The reachable vertices define the (min) cut.

Residual graph with DFS



Min cut (in G!)



Announces

- Office hours extended until 3pm today.
- Office hours (Carlos & Roman) on wednesday from 2pm to 4pm in TR3110.
- Mid-term exam scheduled at 11h30 (regular class hours) in ADAMS Auditorium.
- One crib sheet (2 pages) allowed.