Greedy Algorithms

Activity Selection

- Activity: an interval $[s,f)$ where $s \le f$
  - Starting time s and finish time f.
- Problem instance: A set of activities
  - $S=\{a_1,\ldots,a_n\}=\{[s_1,f_1),\ldots,[s_n,f_n)\}$
- Goal: Find a maximum-size subset $S^* \subseteq S$ such that any two activities in S^* do not overlap.
  - $[s,f),[s',f') \in S^*$ implies $[s,f) \cap [s',f') = \emptyset$.

Example

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Optimal solution: $S^*=\{a_1,a_4,a_8,a_{11}\}=\{[1,4),[5,7),[8,11),[12,16)\}$

Solvable by DP

- Termination: If $n=0$, return 0.
- Divide-and-Conquer: Let $S-a$ denote the set $\{a' \in S: a' \cap a = \emptyset\}$
  - Solve subproblems $S_i = S - a_i$ for $a_i \in S$.
- Combine: return $\max_{1 \le i \le n}(\mathrm{opt}(S_i)+1)$
- Time complexity: $O(n2^n)$

More Efficient DP

- Preprocessing: Sort the activity in the ascending order of finished time.
- Termination: If n=0, return 0.
- Divide-and-Conquer: Solve subproblems
  - $P_i = \{[s,f)\in S: f\le s_i\}$ and $Q_i=\{[s,f)\in S: s\ge f_i\}$.
- Combine: $\max_{1\le i\le n}(opt(P_i)+opt(Q_i)+1)$
- Time complexity: $O(n^3)$
  - The number of subproblems is $O(n^2)$.

Example

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

The subproblems are in the form $\{[s,f)\in S: s\geq f_i \text{ and } f\leq s_j\}$ for $i\leq j$.

More Efficient Method

- Strategy 1: reduce the number of total subproblems
- Strategy 2: reduce the number of dependent subproblems
- Greedy algorithms can achieve both of the above.
  - Many greedy algorithms are decrease-and-conquer algorithms. No overlapping subproblems.

Greedy Algorithm Values

- Preprocessing: Sort the activity in the ascending order of finished time.
- Termination: If n=0, return 0.
- Divide-and-Conquer: Solve the subproblem $S'=\{[s,f)\in S: s\geq f_1\}$.
- Combine: opt(S')+1
- Time complexity: O(nlogn)
  - The number of total subproblems: O(n)

Greedy Algorithm Solution

- Preprocessing: Sort the activity in the ascending order of finished time.
- Termination: If n=0, return $\varnothing=\{\}$.
- Divide-and-Conquer: Solve the subproblem $S'=\{[s,f)\in S: s\geq f_1\}$.
- Combine: $opt(S')\cup\{f_1\}$
- Time complexity: O(nlogn)
  - The number of total subproblems: O(n)

Example

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Example

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Example

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Example

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Optimal solution: $S^*=\{a_1,a_4,a_8,a_{11}\}=\{[1,4),[5,7),[8,11),[12,16)\}$

Greedy algorithm

- Obviously, it is much more efficient!
  - Decrease-and-conquer
  - If the input is given in the ascending order of finished time, it is in O(n).
- But, is it correct?
  - Yes. You have to show this!
  - Optimal substructure
  - Greedy choice property

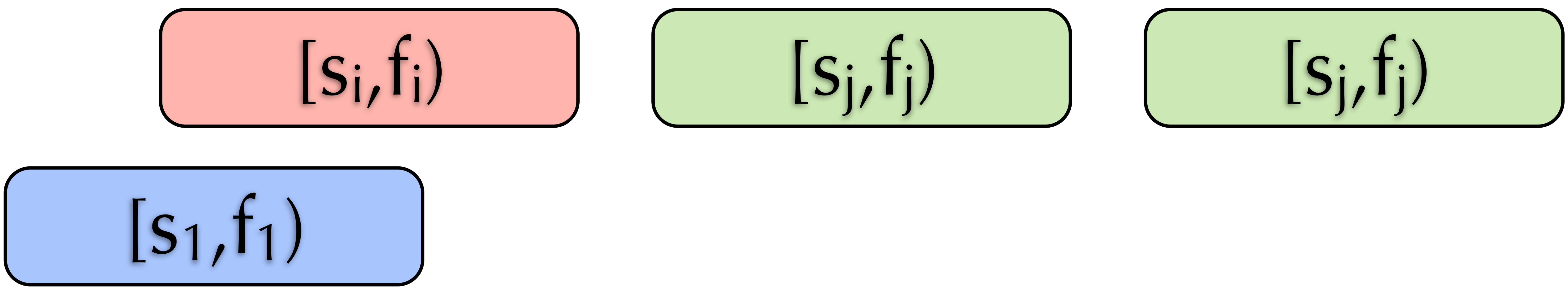
Optimal Substructure

- Similar to dynamic programming
- An optimal solution contains optimal solutions to the subproblems
  - Ex: if $a_1 \in opt(S)$, $opt(S)$ contains the optimal solution to $S-a_1$. $opt(S) \setminus \{a_1\}$ is an optimal solution to $S-a_1$.
  - BWOC: $|opt(S-a_1)|>|opt(S)|-1$. It is clear that $opt(S-a_1) \cup \{a_1\}$ is a solution to S, thus $|opt(S-a_1) \cup \{a_1\}|>|opt(S)|$. A contradiction.

Greedy Choice

- We can assemble a global optimal solution by making locally greedy choices.
- Ex: $[s_1,f_1)$ must be in opt(S) if $f_1 \leq f_2 \leq \ldots \leq f_n$.
- BWOC: Assume $[s_1,f_1) \notin$ opt(S) and let $i>1$ be the minimum index s.t. $[s_i,f_i) \in$ opt(S). opt(S)$\cup\{[s_1,f_1)\}\setminus\{[s_i,f_i)\}$ is still an optimal solution to S since $f_1 \leq s_j$ for each activity $[s_j,f_j) \in$ opt(S) other than $[s_i,f_i)$. A contradiction.

Greedy Choice



Fractional Knapsack

- You can take fractions of items
  - 0-1 knapsack: to take or not to take, that's the question.
- Greedy algorithm works: take the most valuable (per kg) first.
- Homework: Give an instance of the 0-1 knapsack problem which fails the greedy algorithm.

Fractional Knapsack

- Homework: Show that the greedy algorithm works for the fractional knapsack problem.
- Homework: Give an efficient implementation of the greedy algorithm solving the fractional knapsack problem.

Task Scheduling

- There are n tasks $t_1,\ldots,t_n$.
  - To finish a task takes a day.
  - Deadline of t_i: d_i
  - Reward of t_i: r_i if finished no later than d_i.
- How can we achieve the maximum total rewards?

Example

i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
r_i	50	60	20	40	30	70	10

1234567: we can get 130

2134765: we can get 140

4216735: we can get 230

1st Attempt: FAILED

- Greedy choice: task with earliest deadline
- Maintain a set S which is empty initially.
- Sort the tasks into $t_1, \ldots, t_n$ by their deadline decreasingly.
  - Tie-breaking: by reward decreasingly
- For i = 1 to n do
  if $d_i \leq i$ then $S := S \cup \{i\}$
  Output $\Sigma_{i \in S} r_i$

Example

i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
r_i	50	60	20	40	30	70	10

i	5	2	4	6	1	3	7
d_i	1	2	3	4	4	4	6
r_i	30	60	40	70	50	20	10

5246137: we can get 200

2nd Attempt: FAILED

- Observation: We don't need to do the task after its deadline!
- $j=1$
  For $i = 1$ to n do
  if $d_i \leq j$ then $S := S \cup \{i\}$, $j = j+1$
  Output $\Sigma_{i \in S} r_i$

Example

i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
r_i	50	60	20	40	30	70	10

i	5	2	4	6	1	3	7
d_i	1	2	3	4	4	4	6
r_i	30	60	40	70	50	20	10

5246137: we can get 210

3rd Attempt: Success?

- An index set T is good iff we can finish every task t_j such that $j \in T$ on time.
- Maintain a set S which is empty initially.
- Sort the tasks into $t_1, \ldots, t_n$ by their rewards decreasingly.
- For i = 1 to n do
  if $S \cup \{i\}$ is good then $S := S \cup \{i\}$

  Output $\Sigma_{i \in S} r_i$

Example

i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
r_i	50	60	20	40	30	70	10

i	6	2	1	4	5	3	7
d_i	4	2	4	3	1	4	6
r_i	70	60	50	40	30	20	10

2416753: we can get 230

Homework

- Show that the 3rd attempt is correct.
- Give an implementation which runs in $O(n^2)$.
- Give an implementation which runs in $O(n \log n)$.
- Note: You can prove these directly. But, 16.4 and 16.5 contain the answers to the first two problems.

Minimum Spanning Tree

- Suppose there is a power plant on a small island and some houses need to be connected to the plant properly.
  - A house is connected to the plant directed.
  - A house is connected to another house which is properly connected.
- What is the minimum total length of the power cables?

Minimum Spanning Tree

- Input: a graph G=(V,E) and a weight function w: E→R.
- Output: a spanning tree with minimum total edge weight.
- Method: Greedy
  - Kruskal's algorithm
  - Prim's algorithm

Kruskal's Algorithm

- Sort the edges into $e_1,\ldots,e_m$ in ascending order of weight: $w(e_i) \leq w(e_j)$ iff $i \leq j$.
- Maintain a partition P of V, initially $P=(\{v_1\},\{v_2\},\ldots,\{v_n\})$.
- For i=1 to m do

  Let $\{s,t\}=e_i$

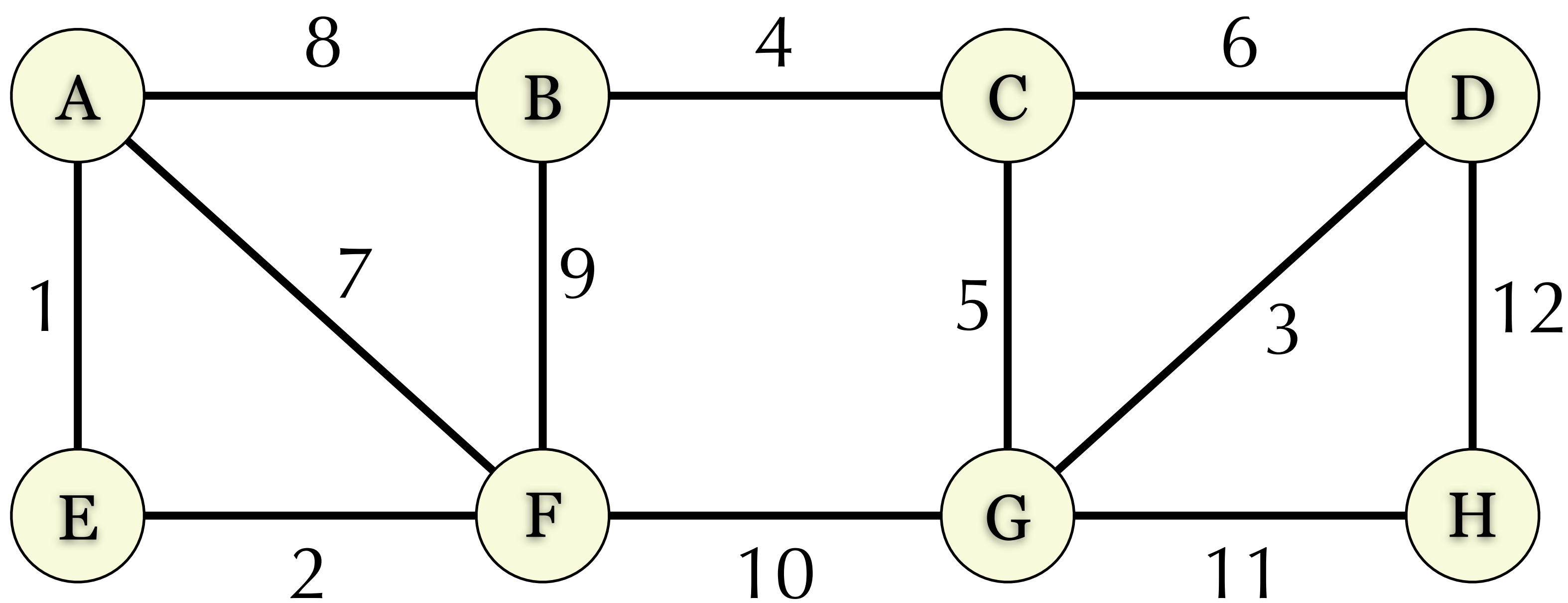
  If s and t are in diff partitions S and T

  put e_i into the output and Union(S,T)

  If there are $n-1$ edges in the output

  break

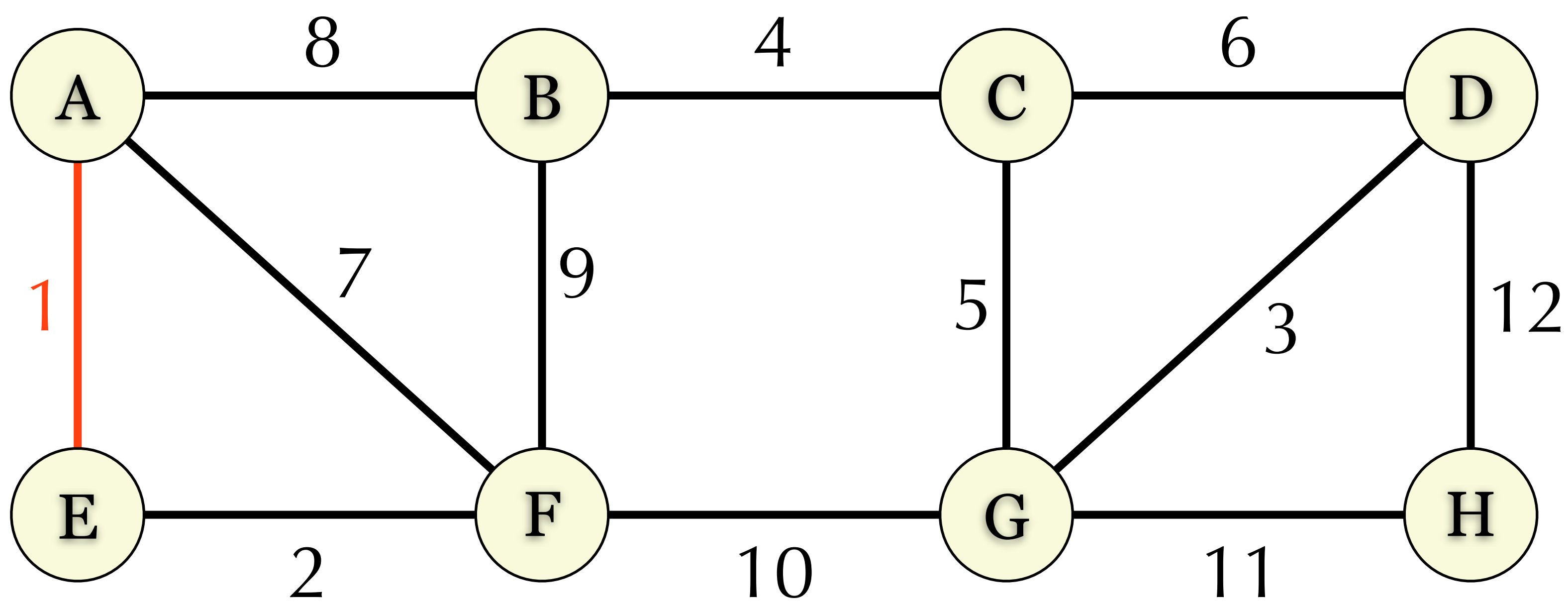
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	B	C	D	E	F	G	H

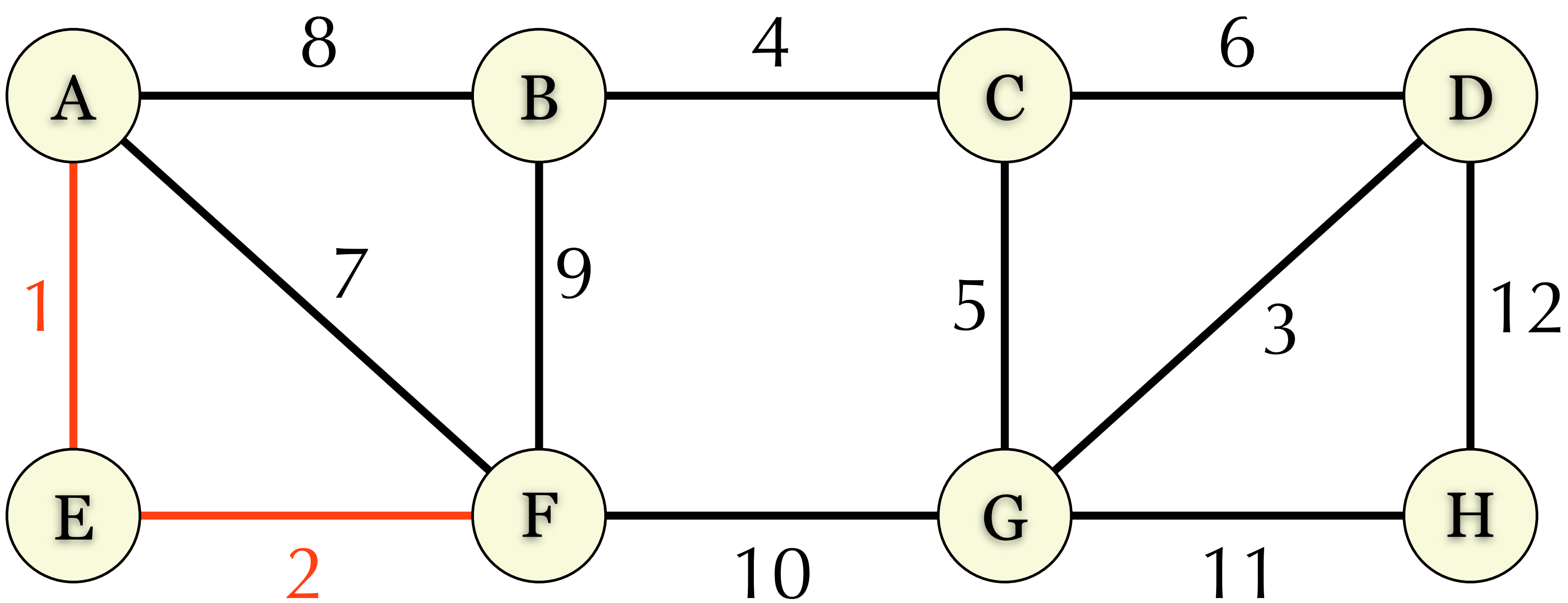
Example



Disjoint sets

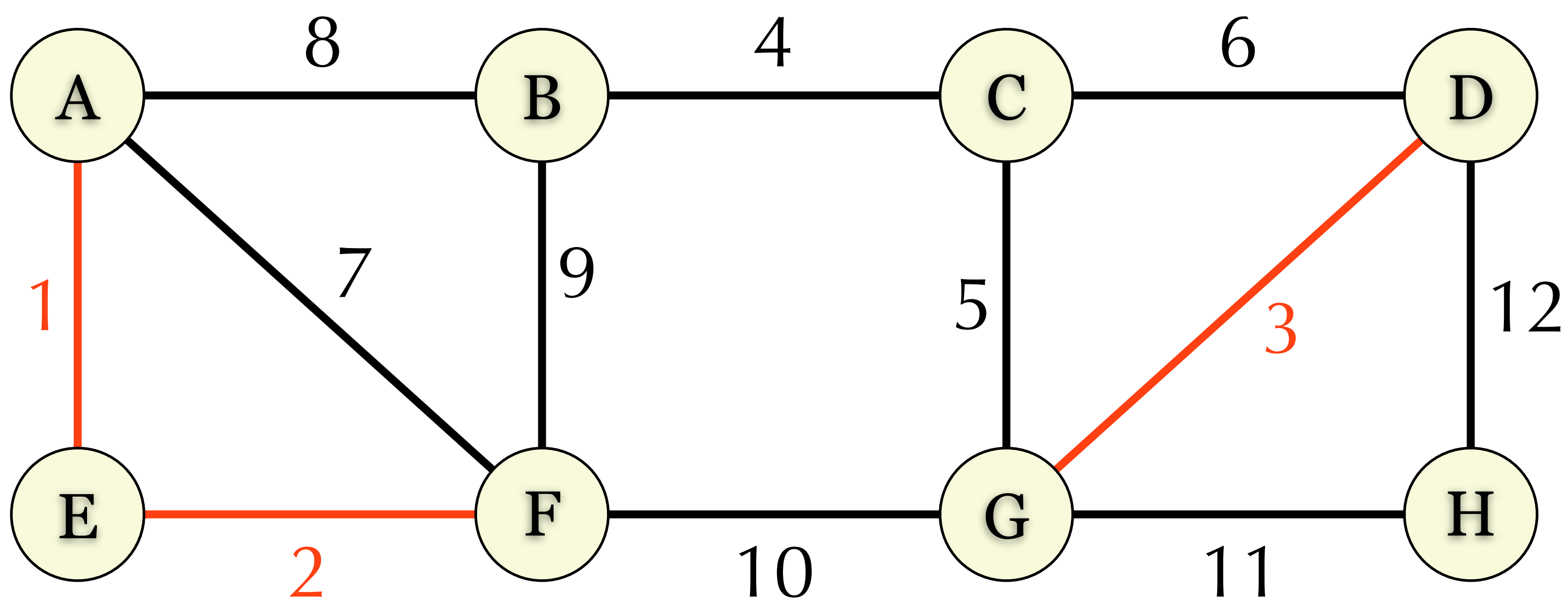
i	A	B	C	D	E	F	G	H
r_i	A	B	C	D	A	F	G	H

Example



Disjoint sets	i	A	B	C	D	E	F	G	H
	r_i	A	B	C	D	A	A	G	H

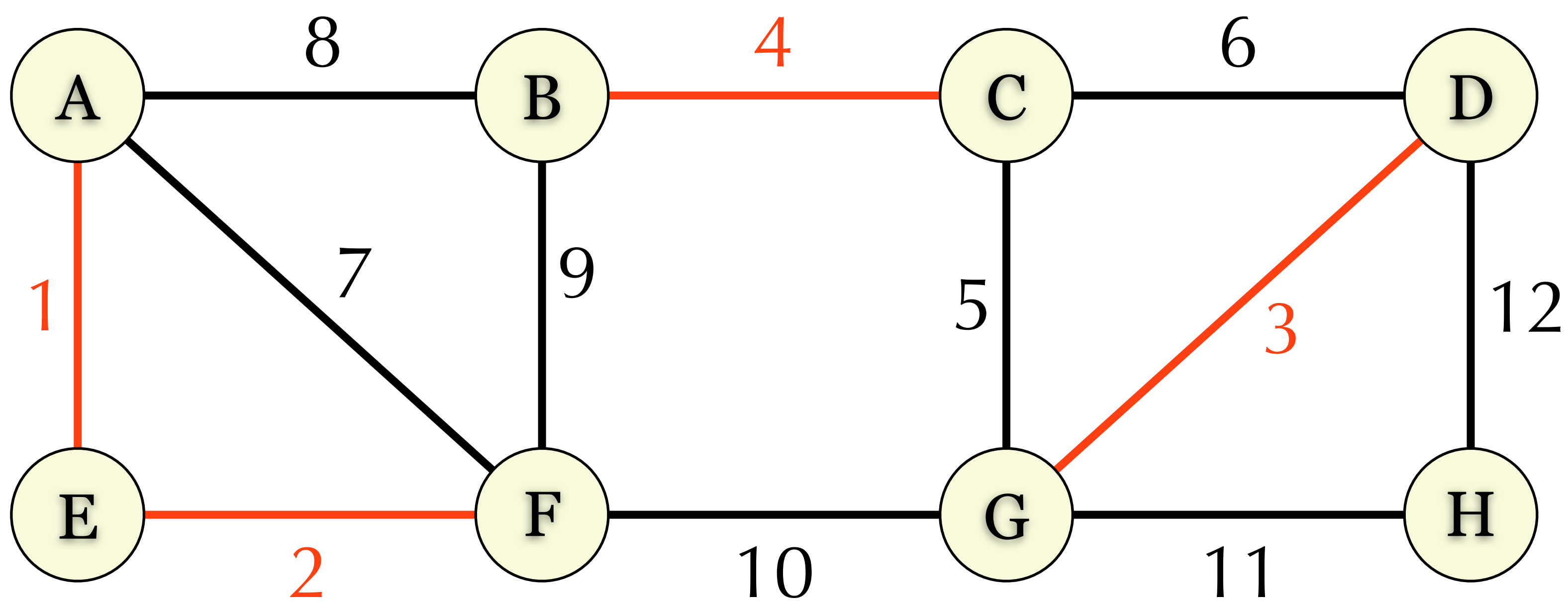
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	B	C	D	A	A	D	H

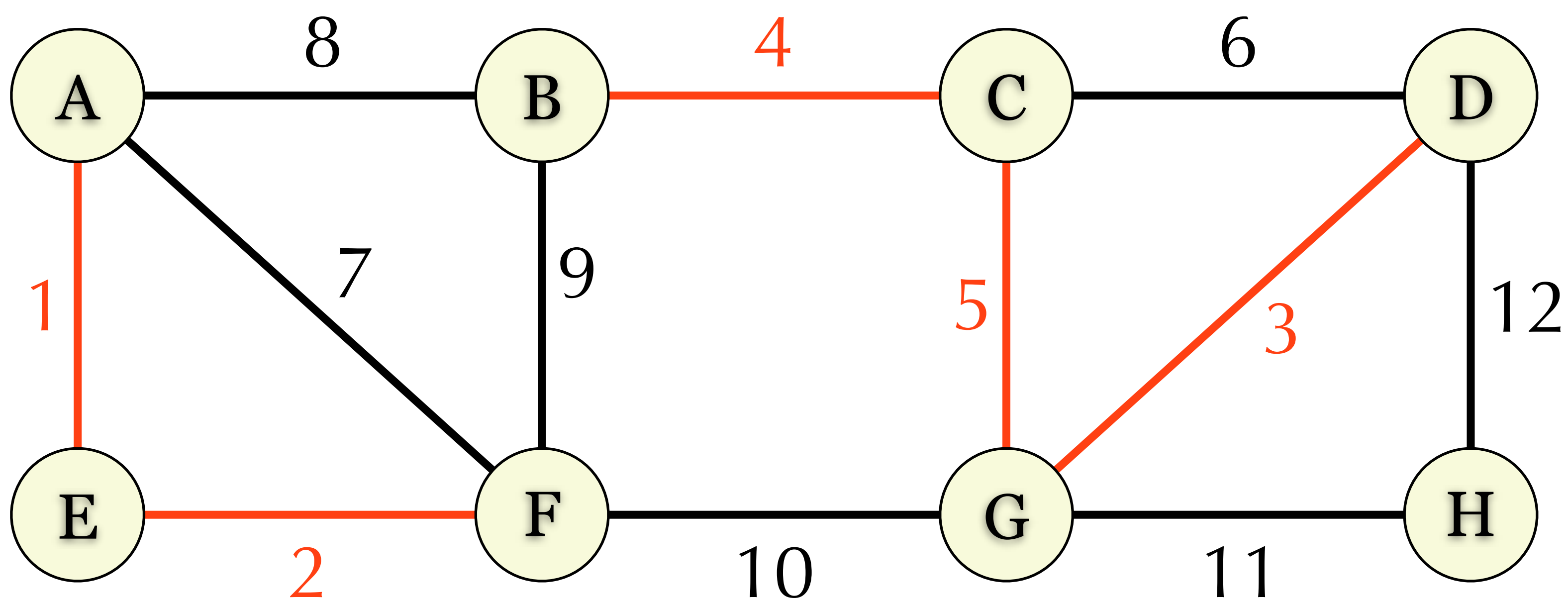
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	B	B	D	A	A	D	H

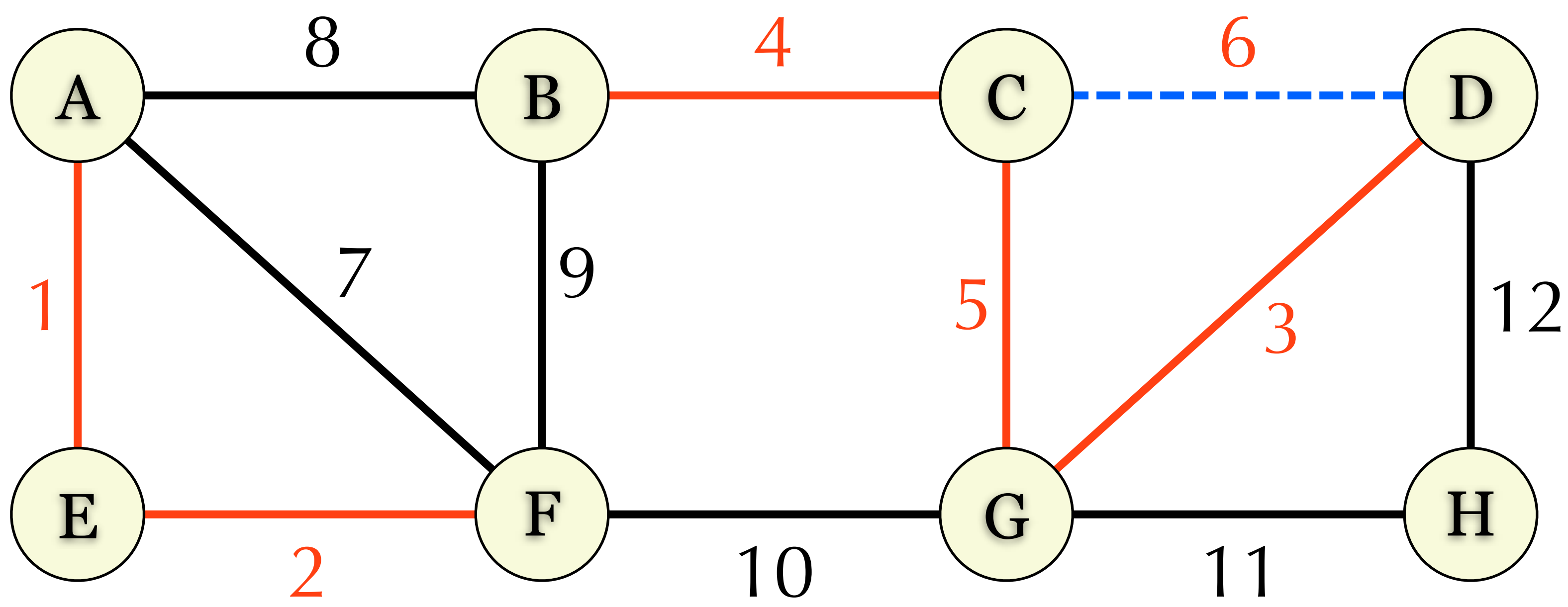
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	B	B	B	A	A	B	H

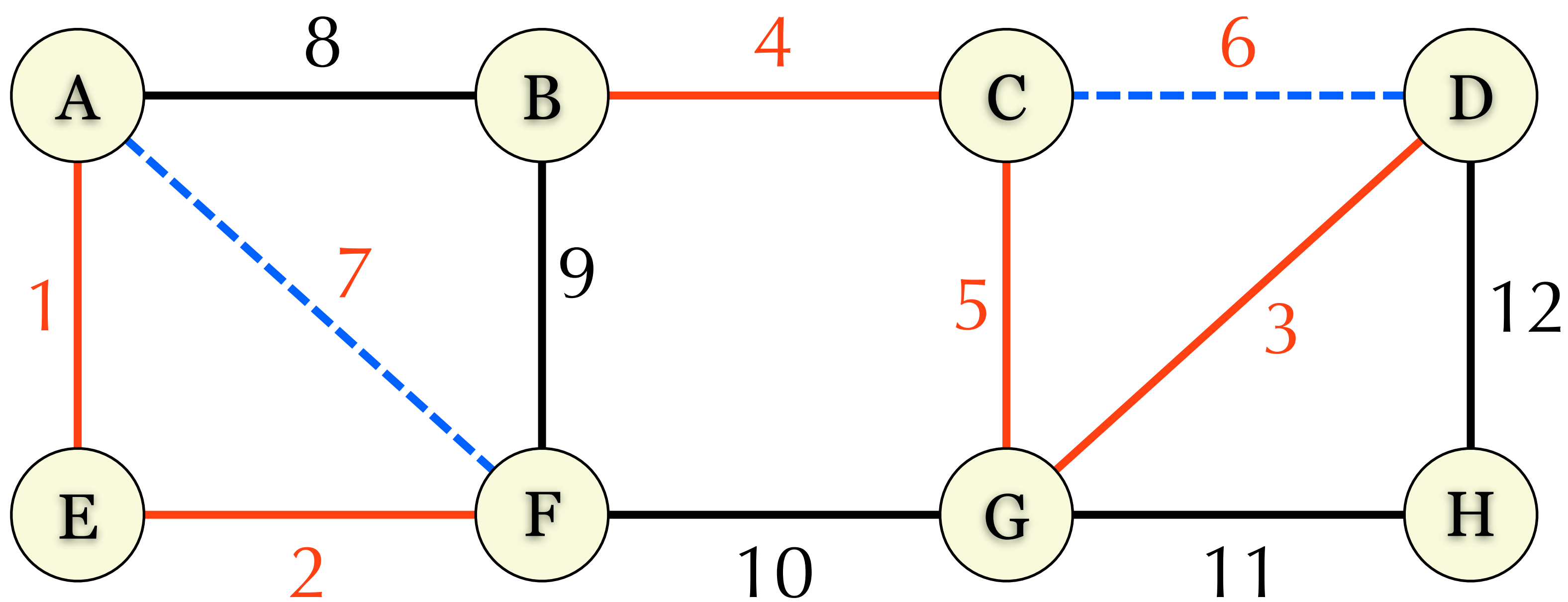
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	B	B	B	A	A	B	H

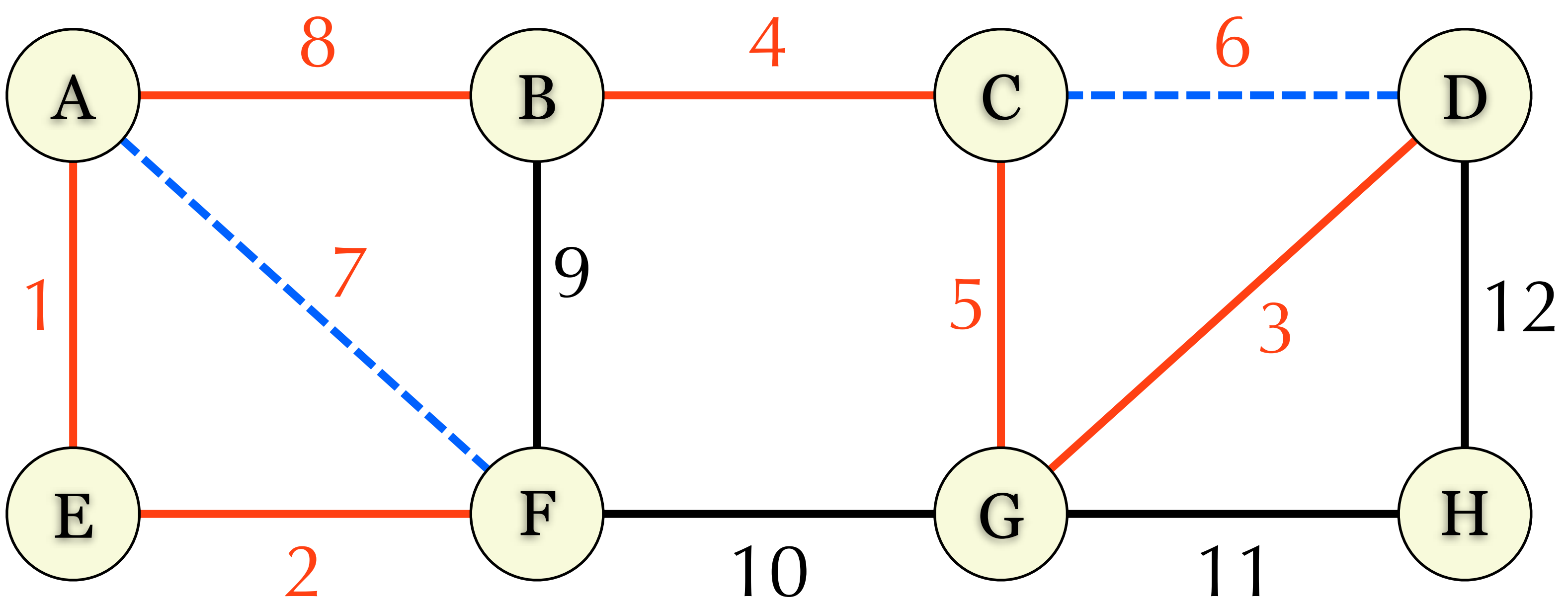
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	B	B	B	A	A	B	H

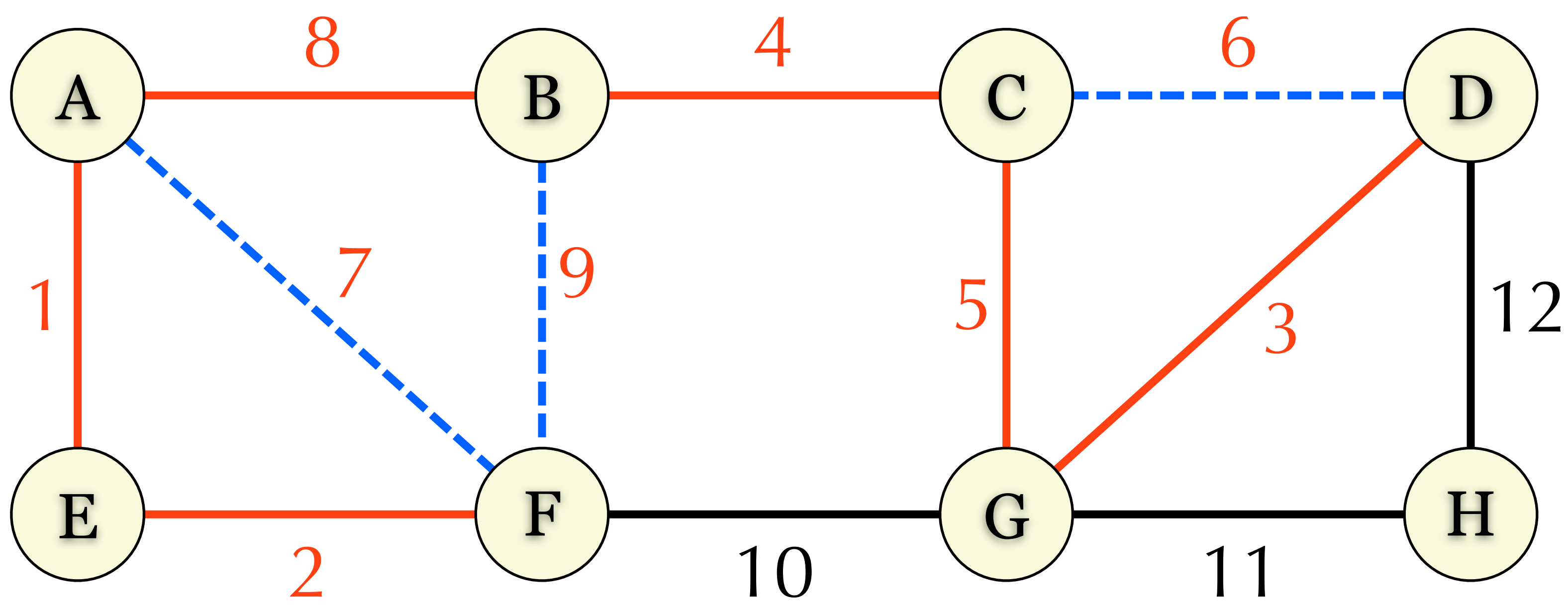
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	A	B	B	A	A	B	H

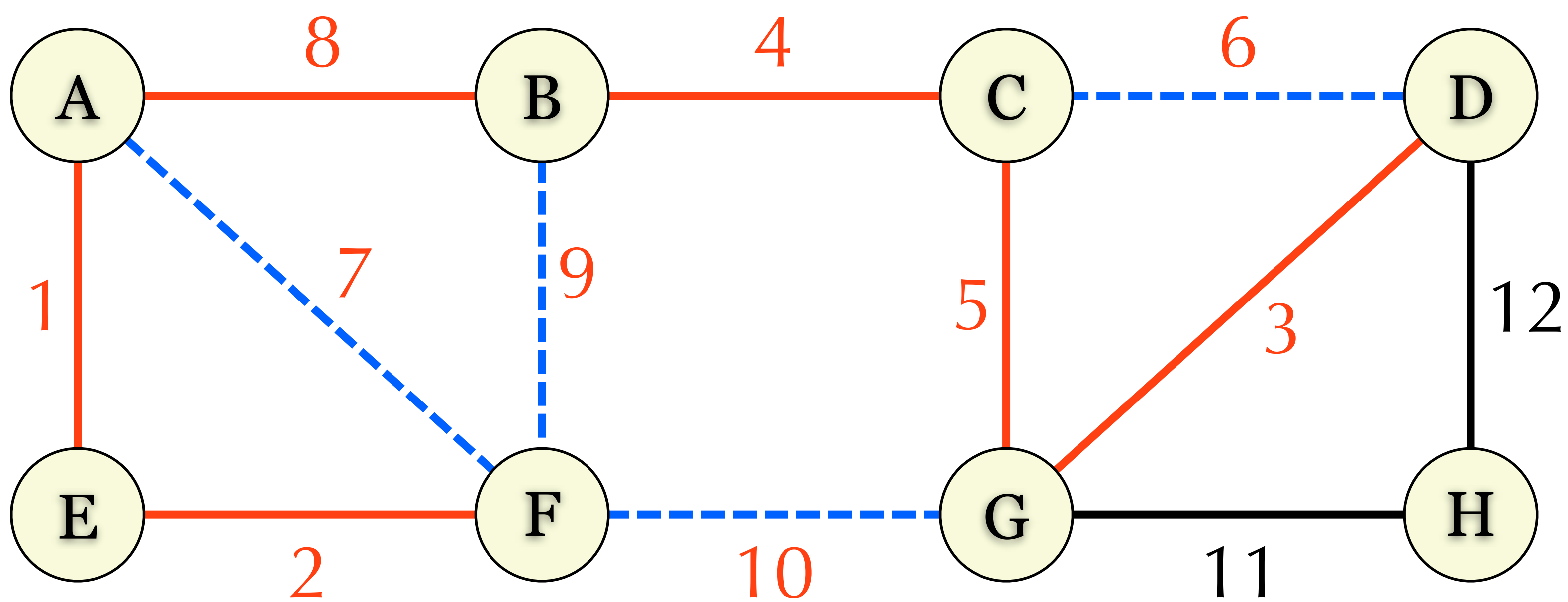
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	A	B	B	A	A	B	H

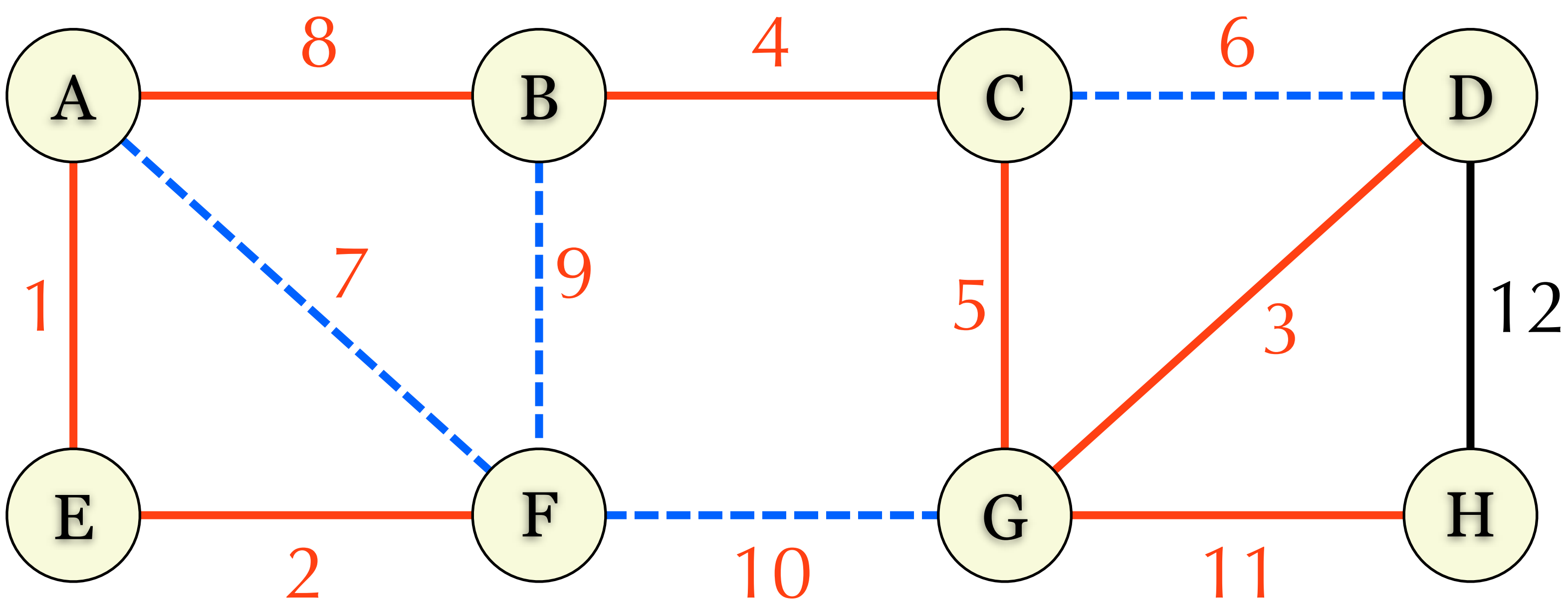
Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	A	B	B	A	A	B	H

Example



Disjoint sets

i	A	B	C	D	E	F	G	H
r_i	A	A	B	A	A	A	A	A

Prim's algorithm

- Maintain a connected subgraph G'=(V',E')
- For each vertex $u \notin V'$, maintain the minimum cost C[u] to connect u to G'.
- Initially, V'={v}, E'=$\emptyset$, C[u]=w(u,v).
- For i=2 to n do

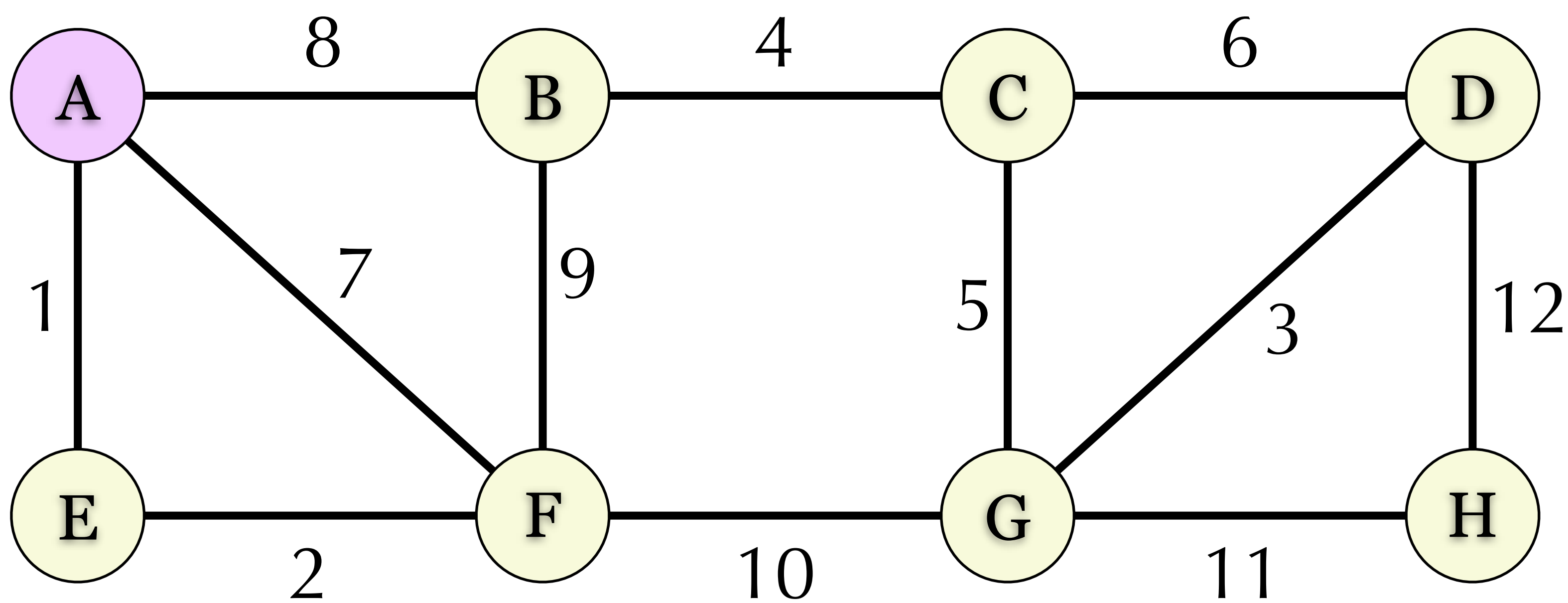
  Let $u^* = \mathrm{argmin}_{u \notin V'} C[u]$

  For each $v^* \in V \setminus V'$ s.t. $\{u^*, v^*\} \in E$ do

    update $C[v^*]$

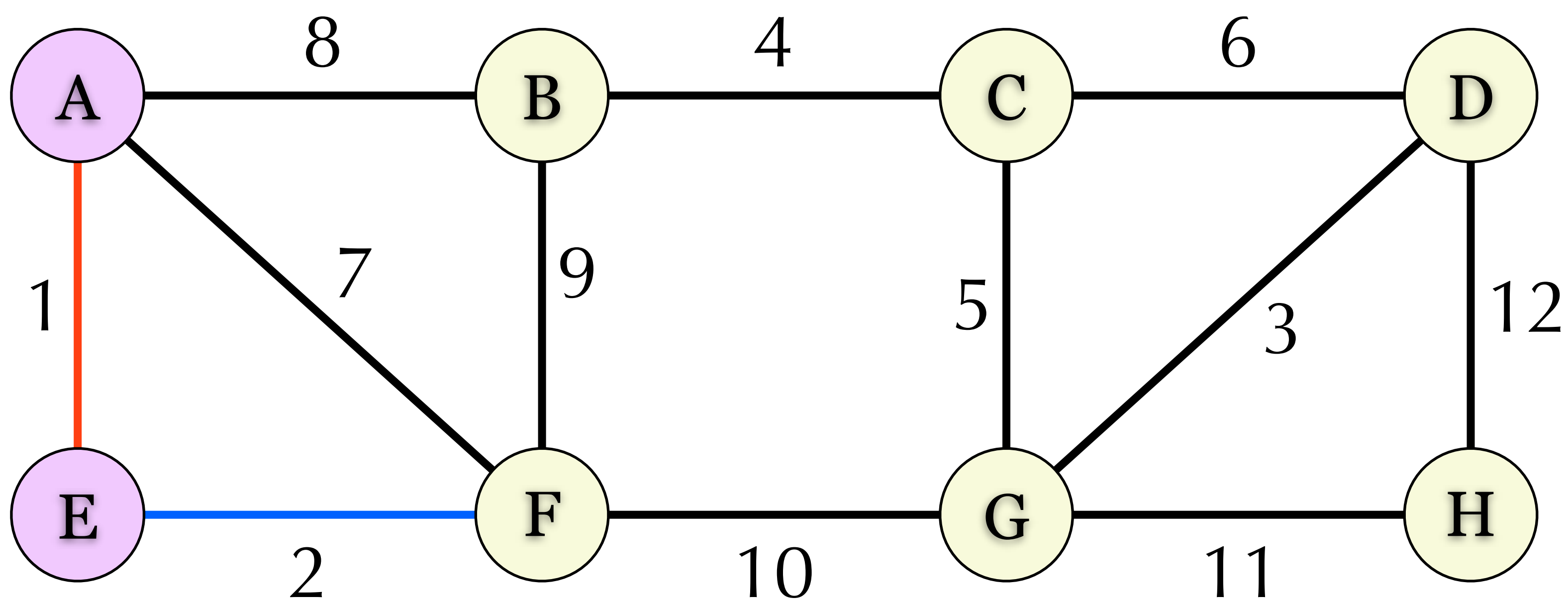
  update G' by adding u^* into V'

Example



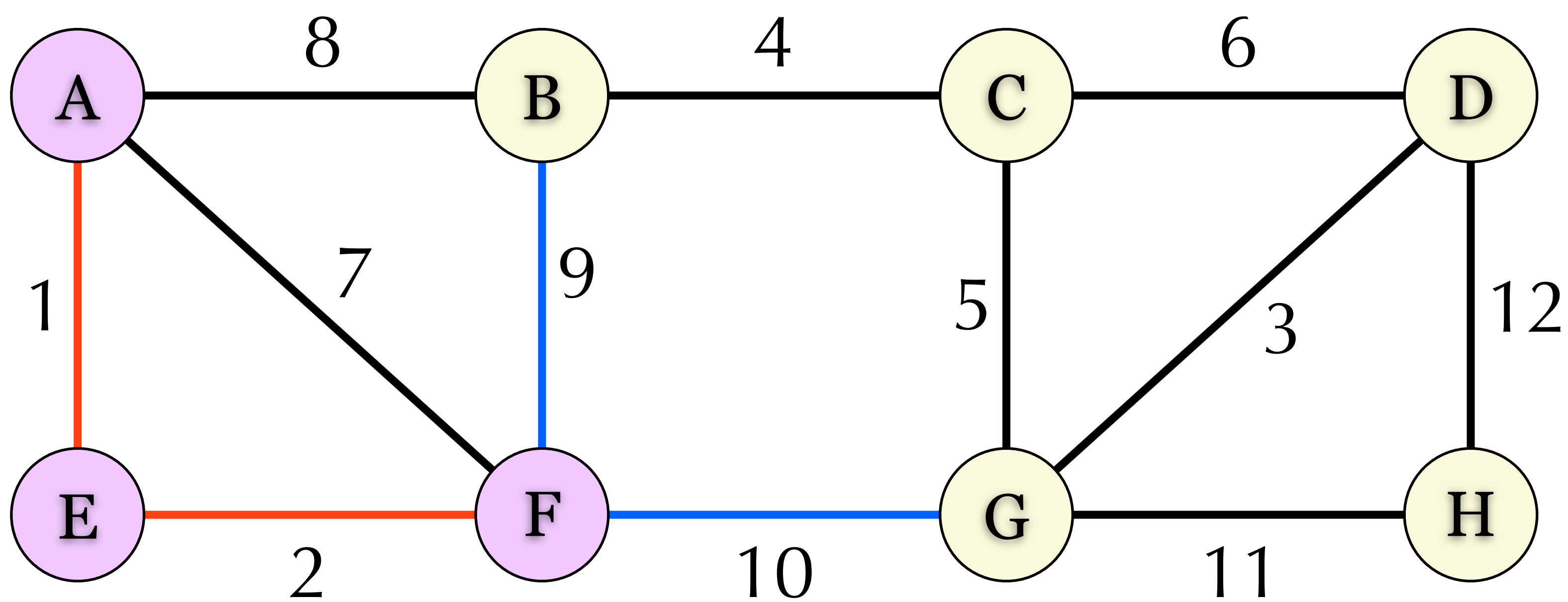
v	A	B	C	D	E	F	G	H
C[v]	0	8	∞	∞	1	7	∞	∞

Example



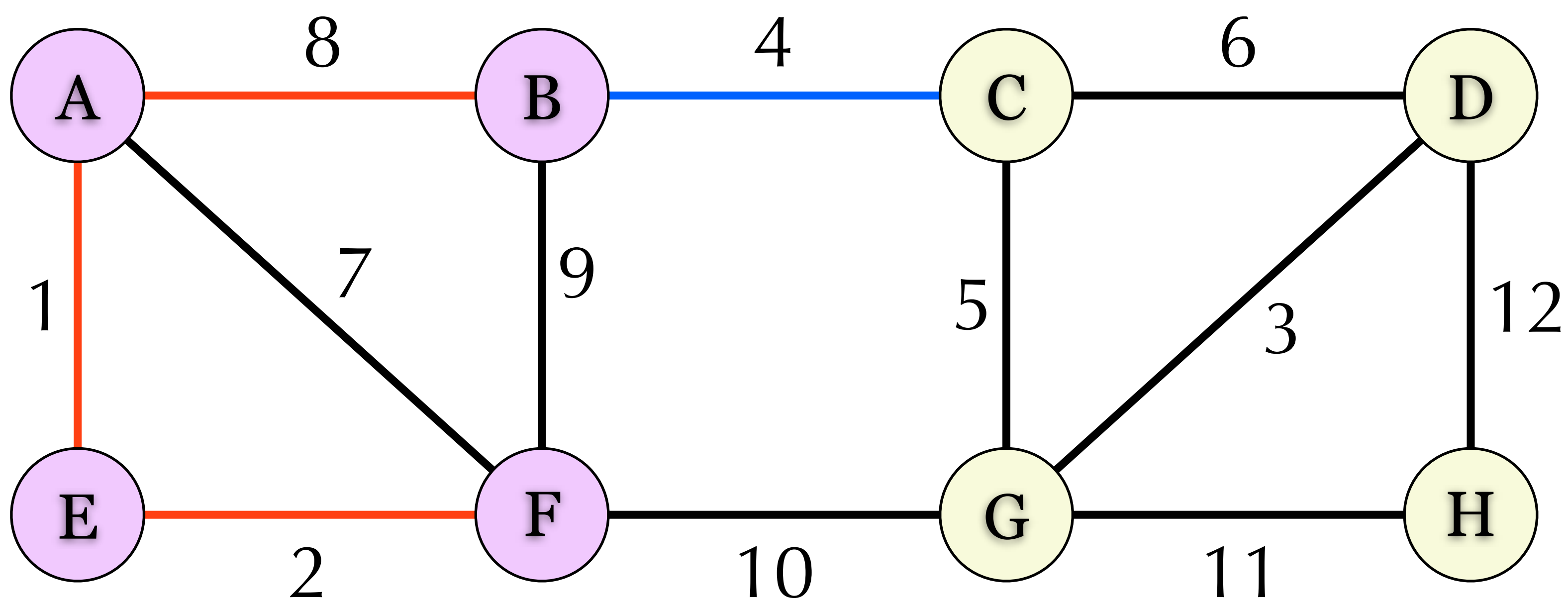
v	A	B	C	D	E	F	G	H
C[v]	0	8	∞	∞	1	2	∞	∞

Example



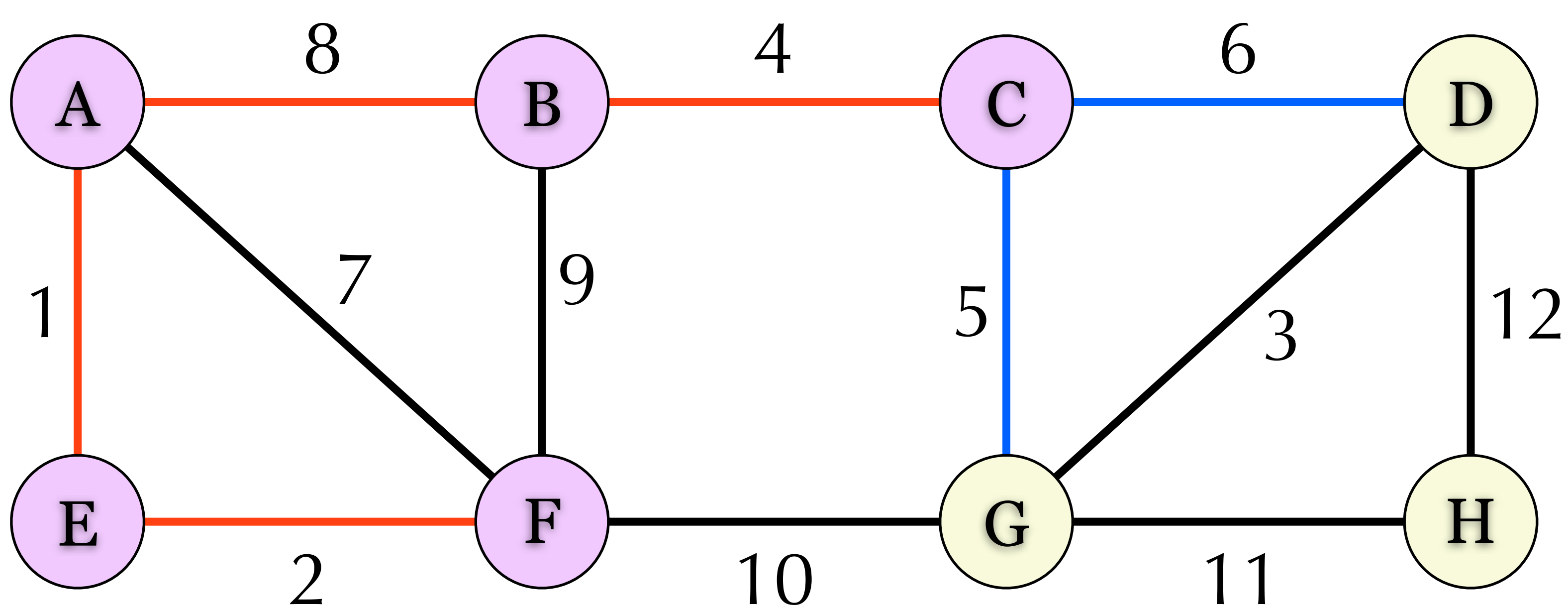
v	A	B	C	D	E	F	G	H
C[v]	0	8	∞	∞	1	2	10	∞

Example



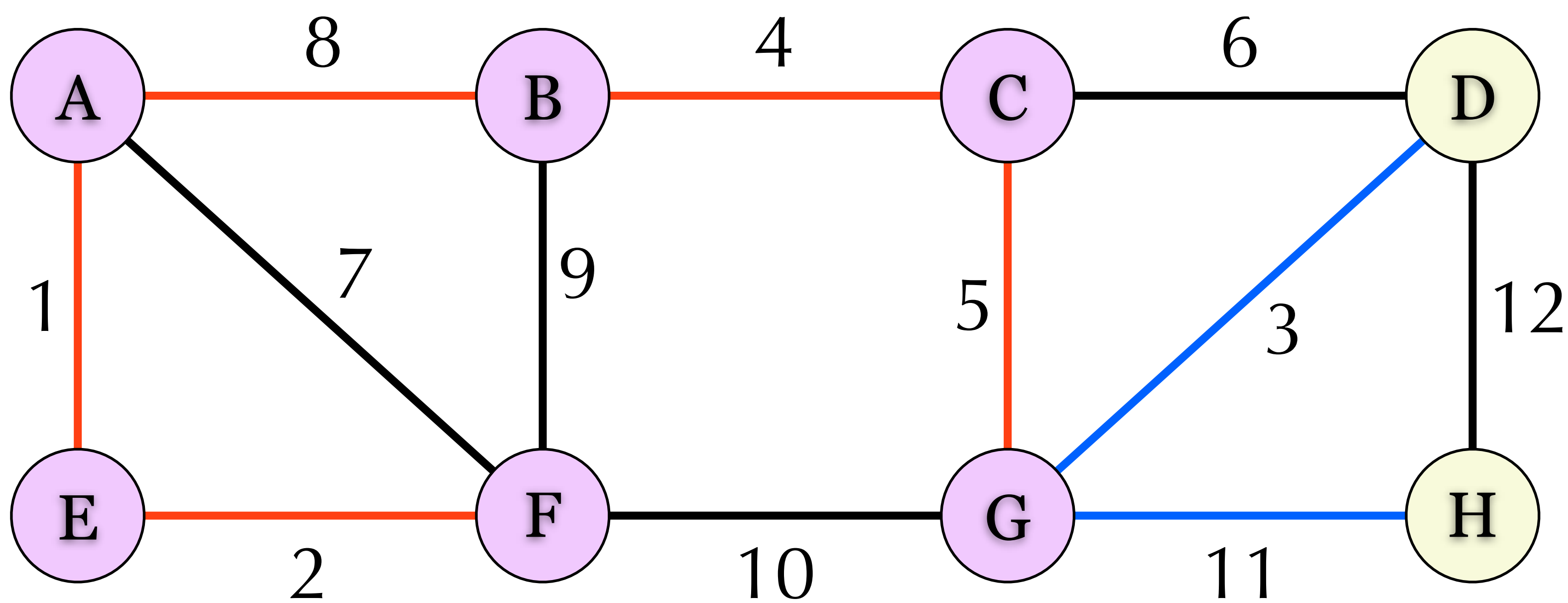
v	A	B	C	D	E	F	G	H
C[v]	0	8	4	∞	1	2	10	∞

Example



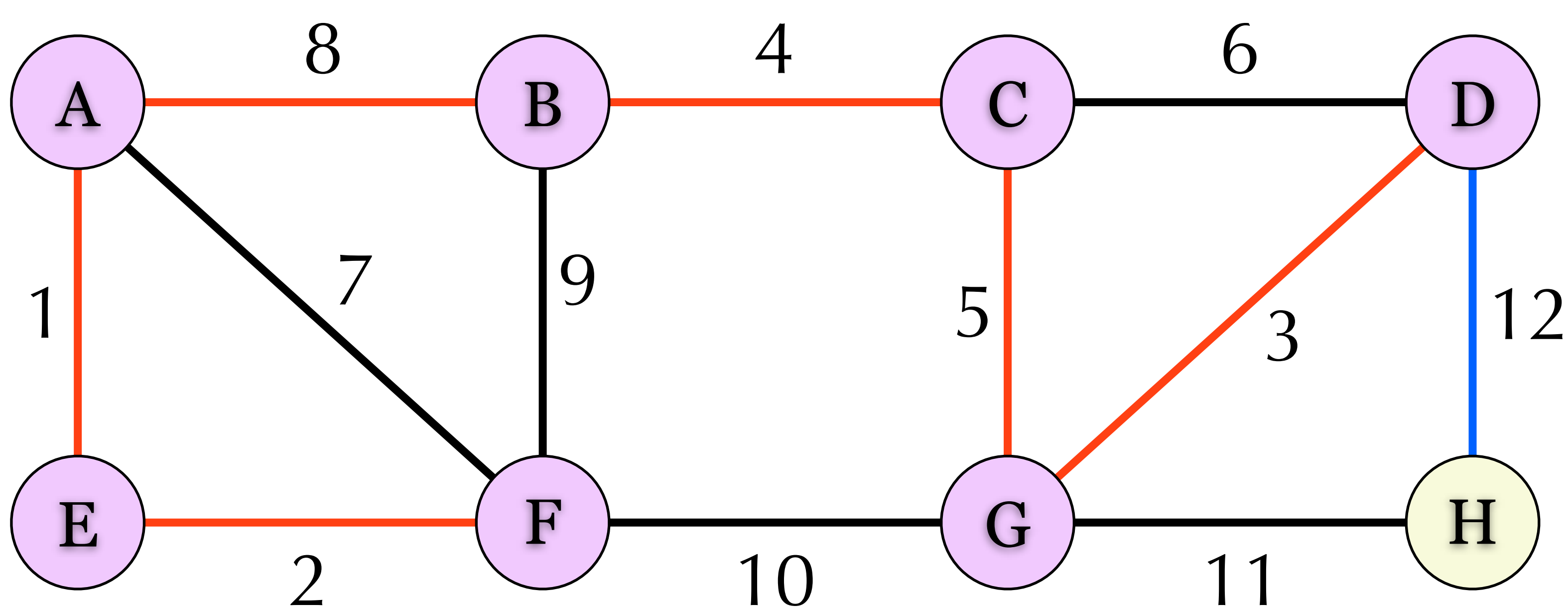
v	A	B	C	D	E	F	G	H
C[v]	0	8	4	6	1	2	5	∞

Example



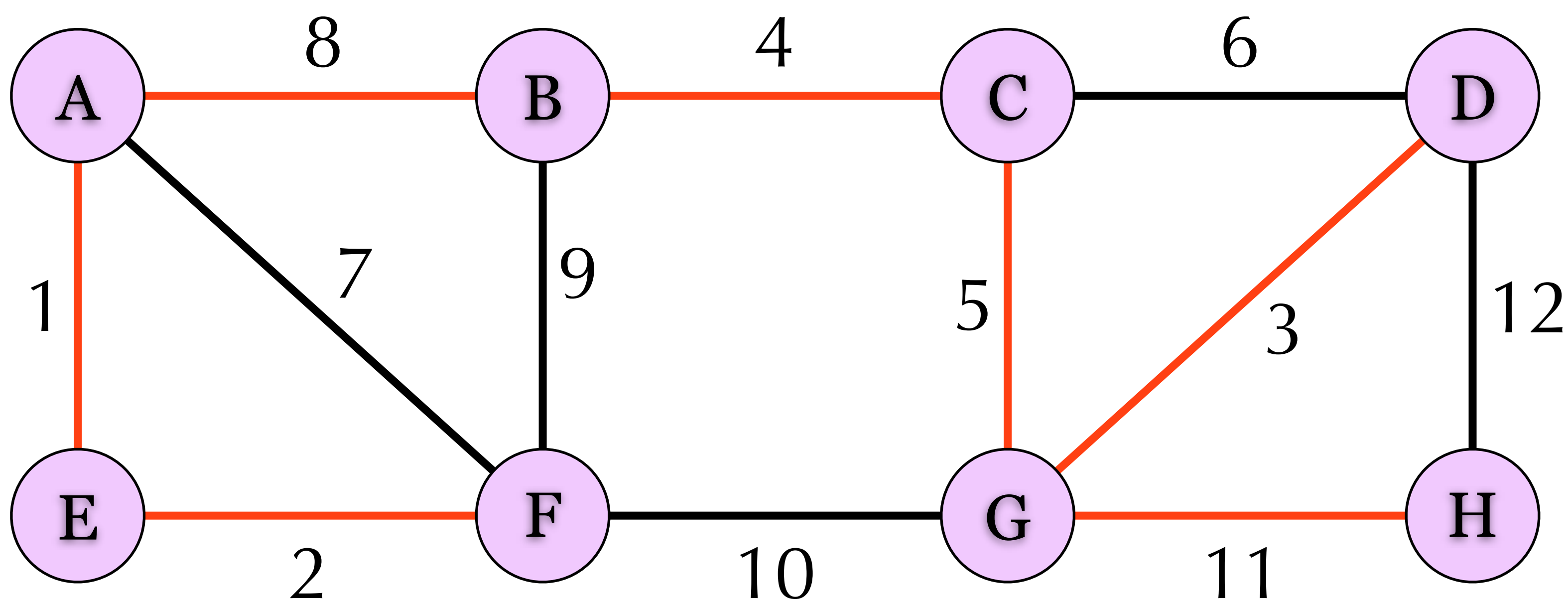
v	A	B	C	D	E	F	G	H
C[v]	0	8	4	3	1	2	5	11

Example



v	A	B	C	D	E	F	G	H
C[v]	0	8	4	3	1	2	5	11

Example



v	A	B	C	D	E	F	G	H
C[v]	0	8	4	3	1	2	5	11

Kruskal vs Prim

- Time
  - Kruskal: $O(E\log V)$
    - We'll discuss disjoint sets later
  - Prim: $O(V^2)$ or $O(E+V\log V)$
- Implementation
  - Kruskal: Disjoint sets & Sort
  - Prim: $O(V^2)$ by Arrays
  - Prim: $O(E+V\log V)$ by Fibonacci heaps

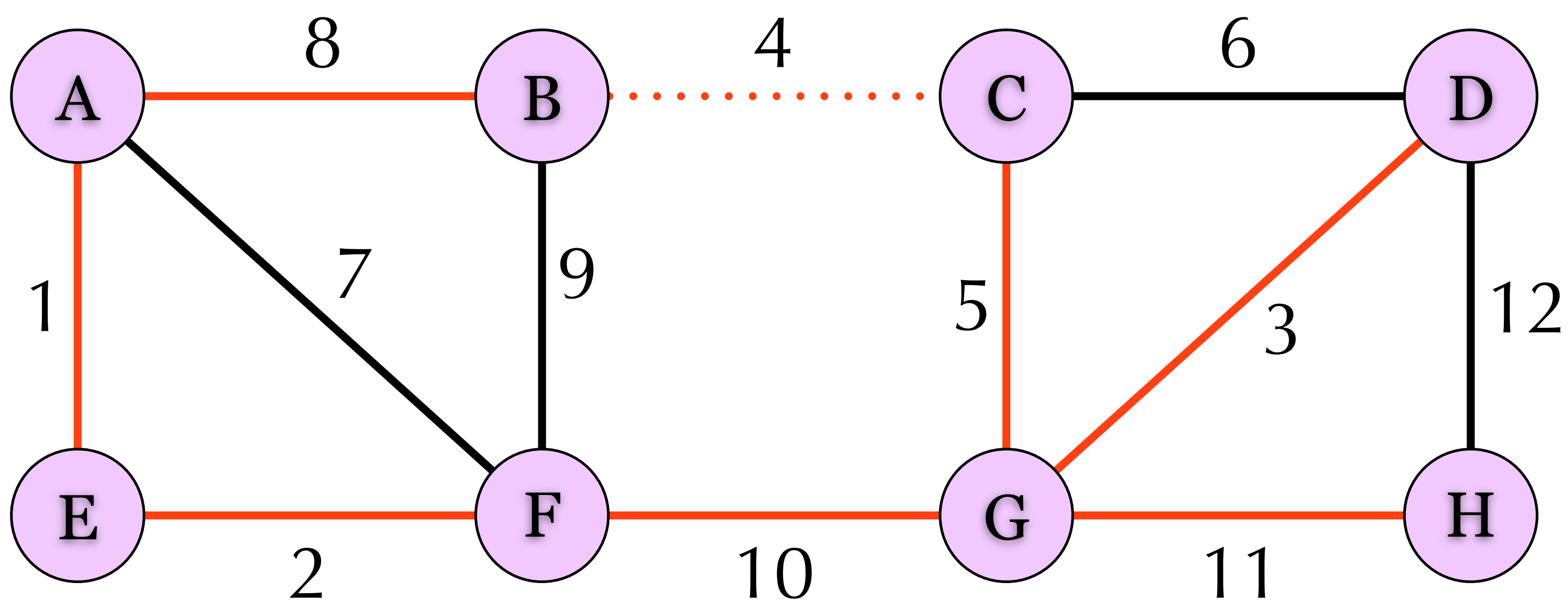
Correctness

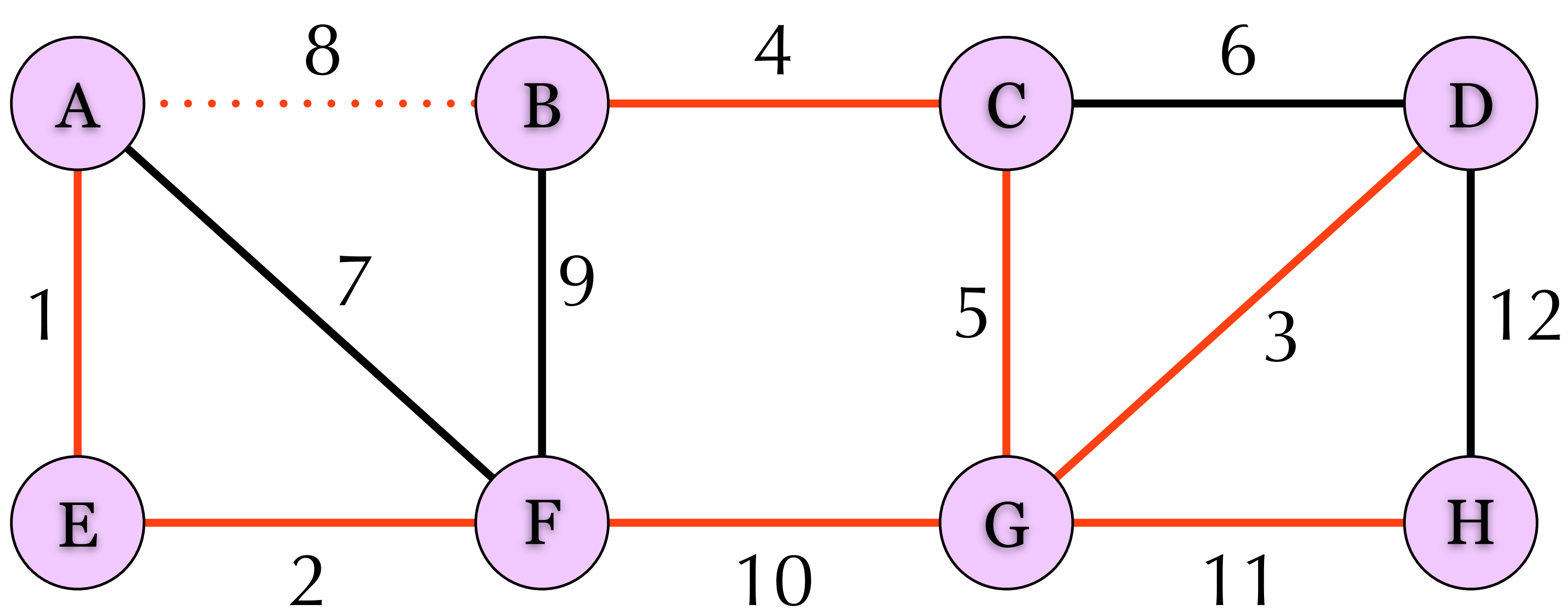
- Greedy choice property:
  - If $e_v = \text{minarg}_{v \in e} w(e)$, then e_v is in the MST.
- Optimal substructure:
  - Let (u,v) be an edge in the MST T_G of $G=(V,E)$.
  - $V' = V \setminus \{u,v\} \cup \{v'\}$
  - $f(u)=f(v)=v'$, $f(p)=p$ for $p \notin \{u,v\}$.
  - $f(p,q)=\{f(p),f(q)\}$, $E'=\{f(p,q):\{p,q\} \in E\}$
  - $G'=(V',E')$ and $T_{G'}=(V',\{f(p,q):\{p,q\} \in T_G\})$.
  - $T_{G'}$ is the MST of G'.

Greedy Choice

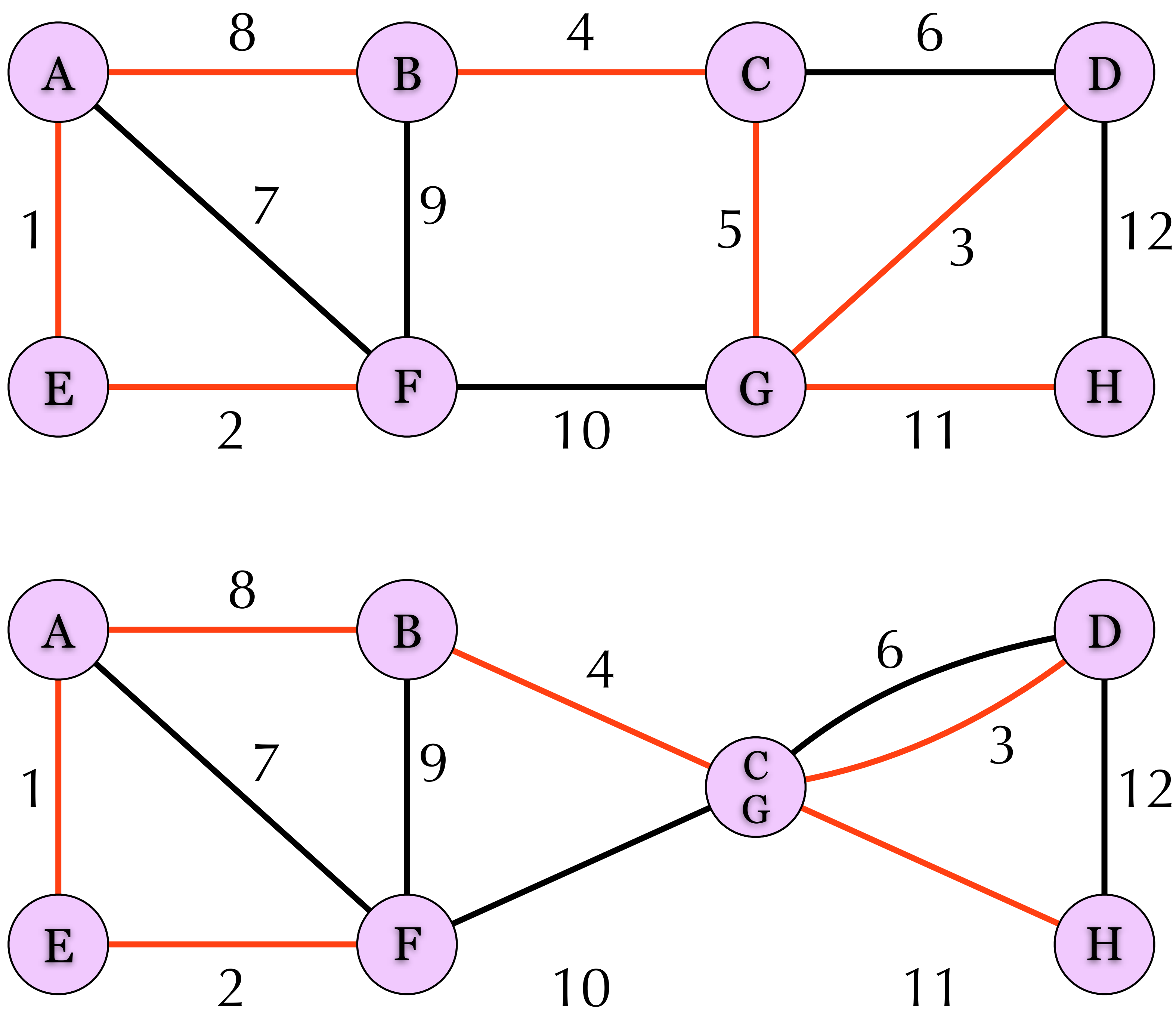
- BWOC, assume $e_v=\{v,u\}$ is not in $T=(V,E_T)$ and e_v has strictly less weight than all the other edges incident to v.
- There is a path $p=(v,v_1,\ldots,v_k,u)$ from v to u.
- Let $T'=(V,E_T\backslash\{\{v,v_1\}\}\cup\{\{v,u\}\})$.
- Since $w(v,v_1)>w(v,u)$, we have $cost(T)=cost(T')-w(v,v_1)+w(v,u)<cost(T')$. A contradiction.

Greedy Choice





Optimal Substructure



Proof

- BWOC, let $T'=(V',E_{T'})$ be the MST of G' s.t. $cost(T')<cost(T_{G'})$
- $f^{-1}(p',q')=\text{minarg}_{f(p,q)=\{p',q'\}}w(p,q)$
  - There might be multiple edges (p,q) such that $f(p,q)=\{p',q'\}$.
- Let $T''=(V,\{u,v\}\cup\{f^{-1}(p',q'):\{p',q'\}\in E_{T'}\})$
- $cost(T'')=cost(T')+w(u,v)$
  $<cost(T_{G'})+w(u,v)=cost(T_G)$.
  A contradiction.

Prim vs Kruskal

- Prim:
  - Apply the greedy choice on a vertex.
  - Solve the subproblem.
- Kruskal:
  - Check whether the remaining edge of the minimum weight is still a greedy choice.
  - If yes, reduce the problem.

Homework

- MST related
  - What if there are two power plants on the island?
  - How to find the maximum spanning tree?
  - How to find the second minimum spanning tree in $O(V^2)$-time?
  - How to count how many distinct MSTs?

Prefix Code

- Character code: each character c is represented by a unique (binary) string s_c.
- Variable length code: $|s_c|$ and $|s_{c'}|$ are not necessarily equal for $c \neq c'$.
- Prefix code: s_c is not a prefix of $s_{c'}$ for $c \neq c'$.
  - Prefix: abcd is a prefix of abcdefgh.
  - To avoid ambiguity

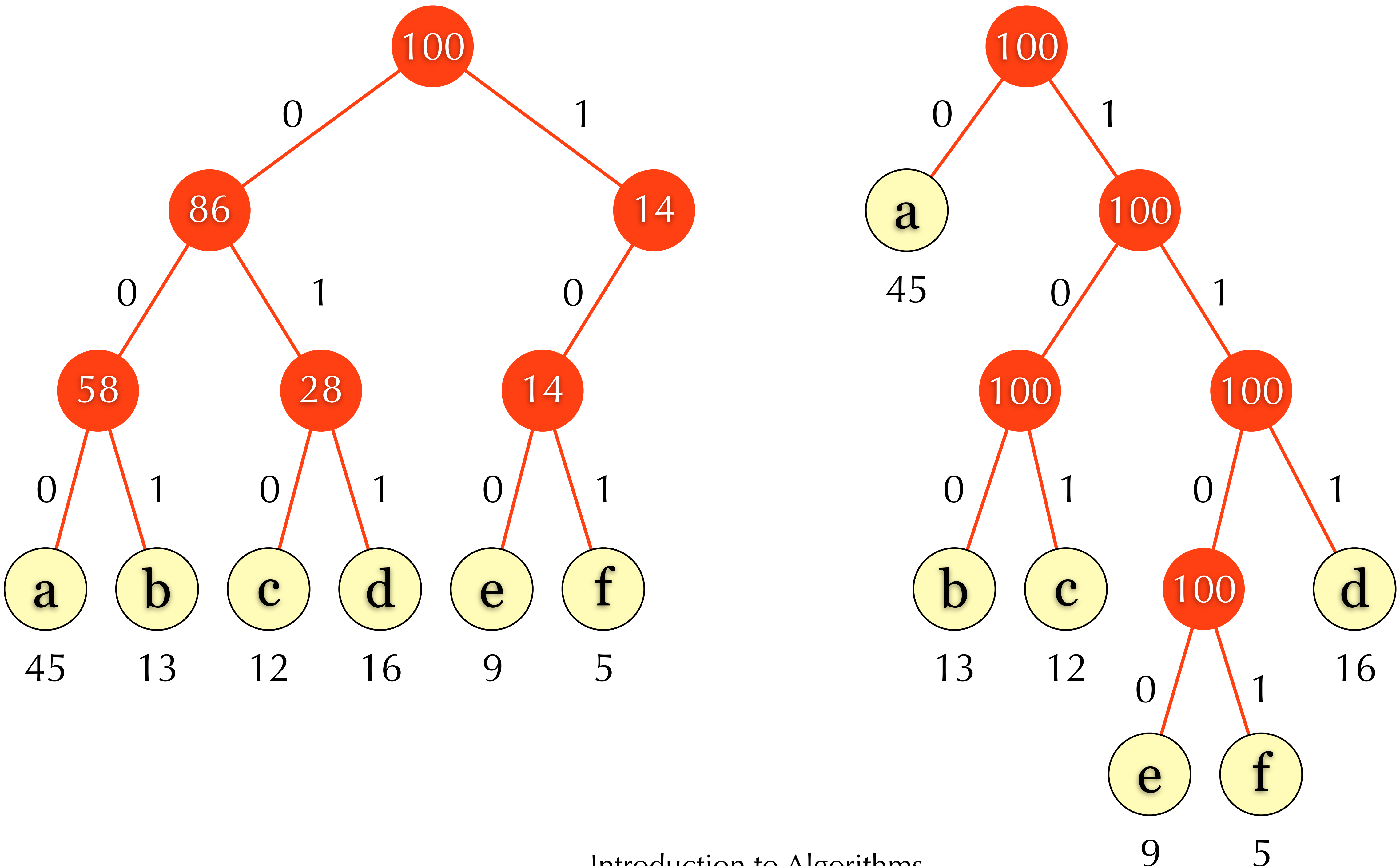
Example

Character	a	b	c	d	e	f
Freq. in file	45	13	12	16	9	5
Fixed length code	000	001	010	011	100	101
Variable length code	0	101	100	111	1101	1100

FLC: $(45+13+12+16+9+5)\times3=300$

VLC: $45\times1+(13+12+16)\times3+(9+5)\times4=224$

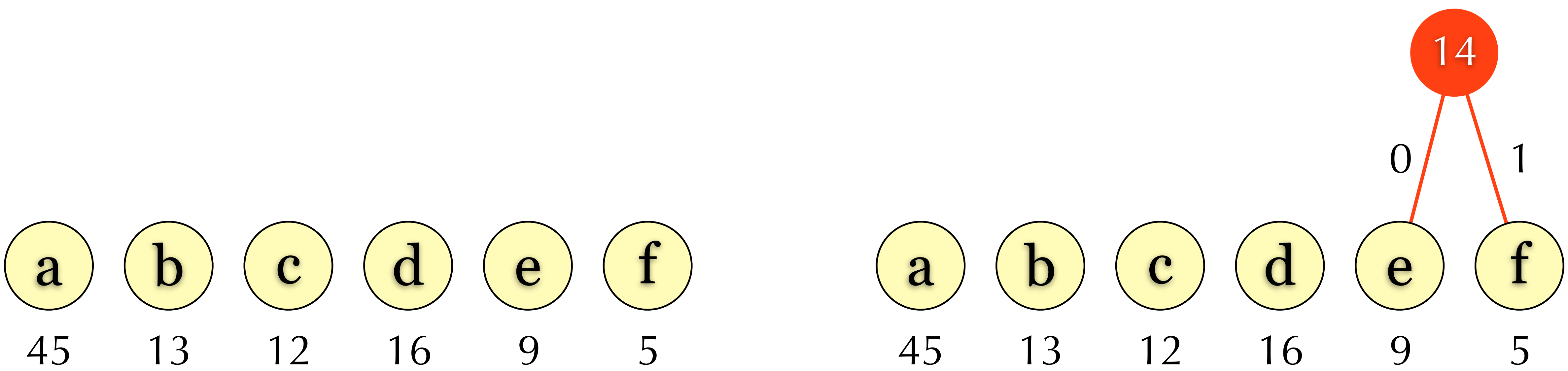
Prefix Encoding Tree



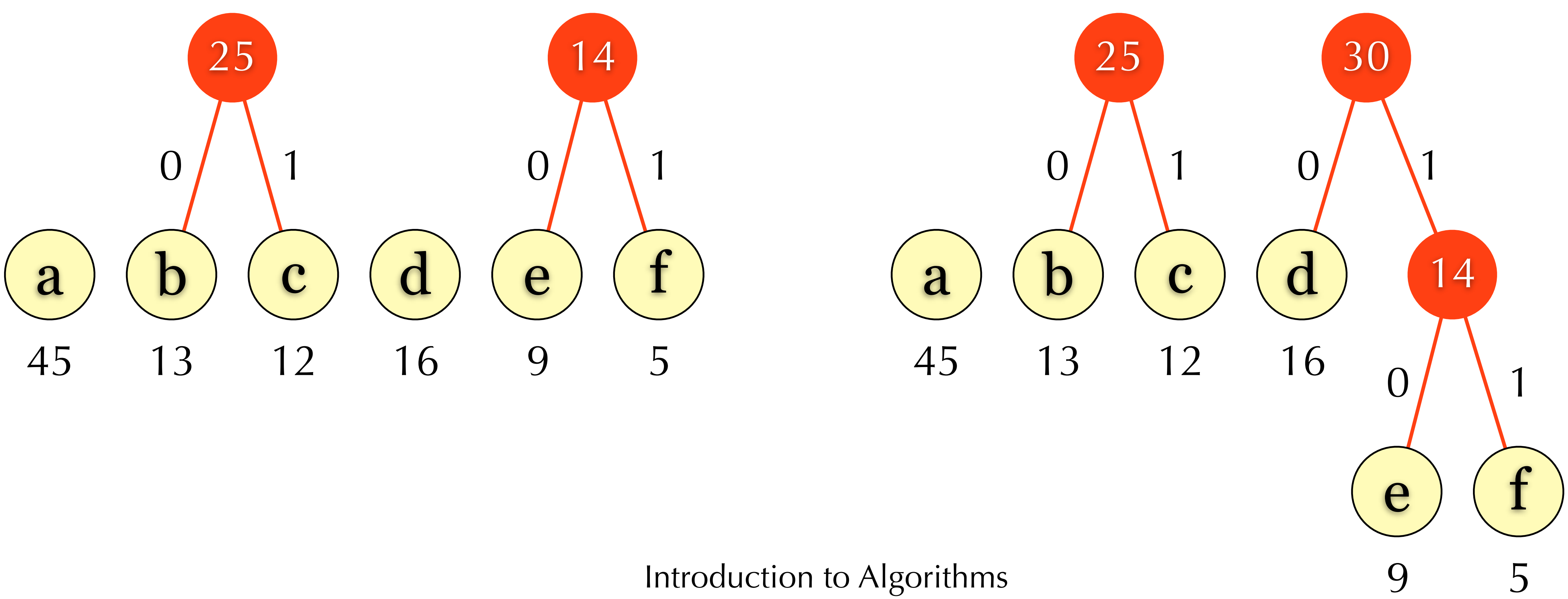
Huffman Code

- An optimal variable length prefix code
- Constructed by a greedy algorithm:
  - Merge two least frequent nodes n_a & n_b
    - Generate a new node n_{ab} whose frequency is f_a+f_b.
  - Repeat the merge process until exact one node remains.

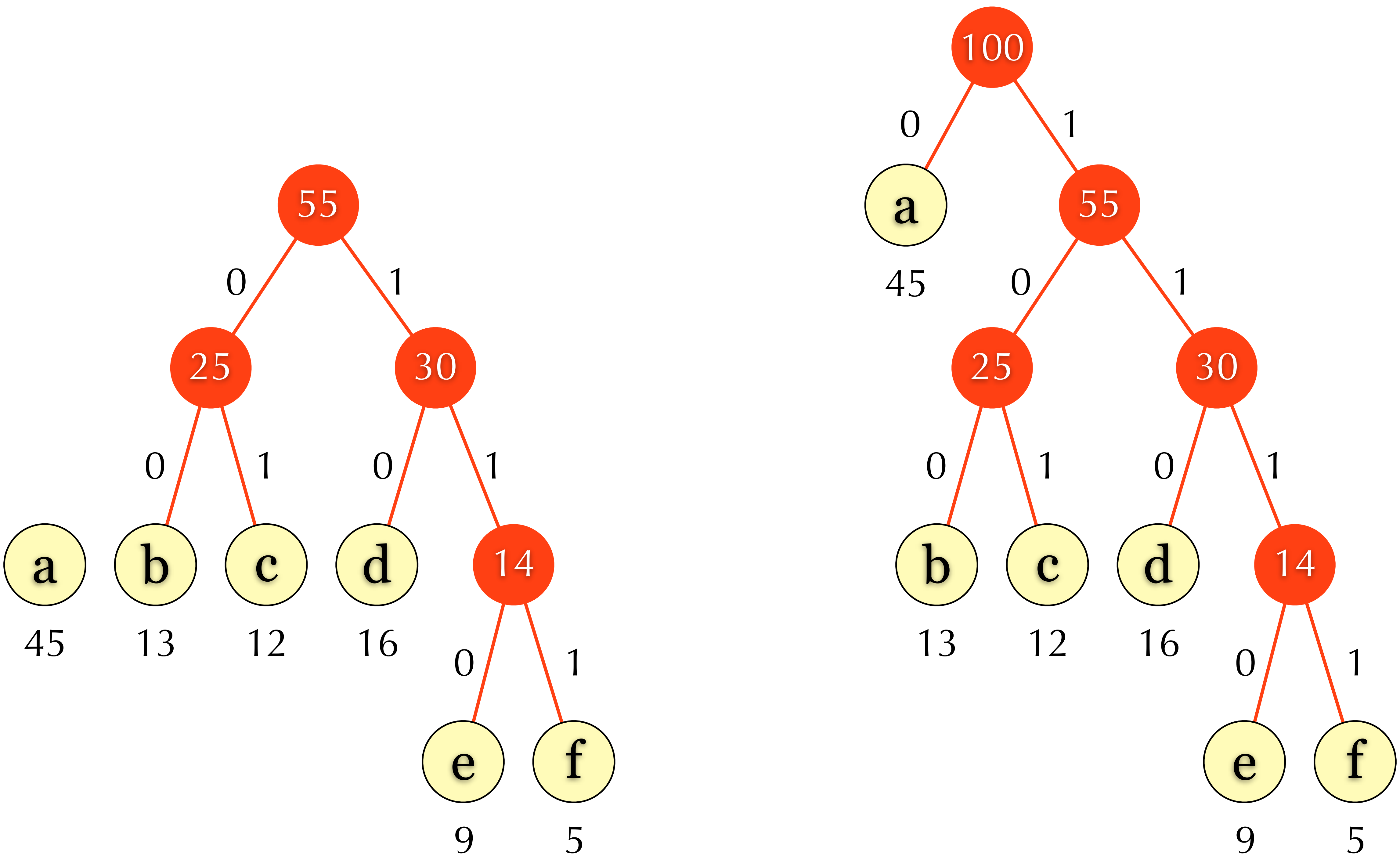
Huffman Encoding Tree



Huffman Encoding Tree



Huffman Encoding Tree



Correctness

- We need to show two things
  - Greedy choice property (Lemma 16.2)
    - The least frequent two characters have the same length in some optimal prefix code.
  - Optimal substructure (Lemma 16.3)
    - If we merge the least frequent two nodes, then the tree is still optimal to the subproblem.

Lemma 16.2

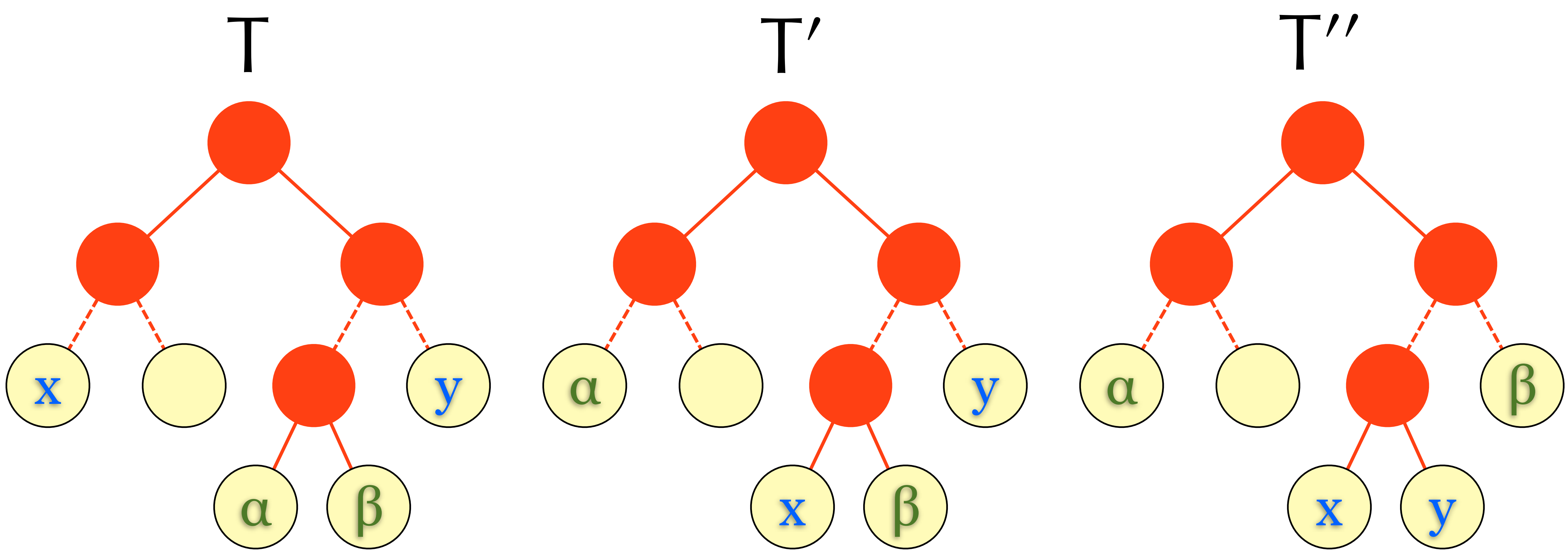
- Let C be an alphabet in which each character $c \in C$ has frequency f_c. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

Proof

- Let T be the encoding tree of some optimal prefix code of C.
- If the codewords for x and y in T have the same length and differ only in the last bit, then we are done!
- Let α be the character of the longest codeword, and β be α's sibling in T.
  - Their codewords have the same length and differ only in the last bit!

Proof

- We exchange x and α to obtain T' from T.
- We exchange y and β to obtain T'' from T'.



Proof

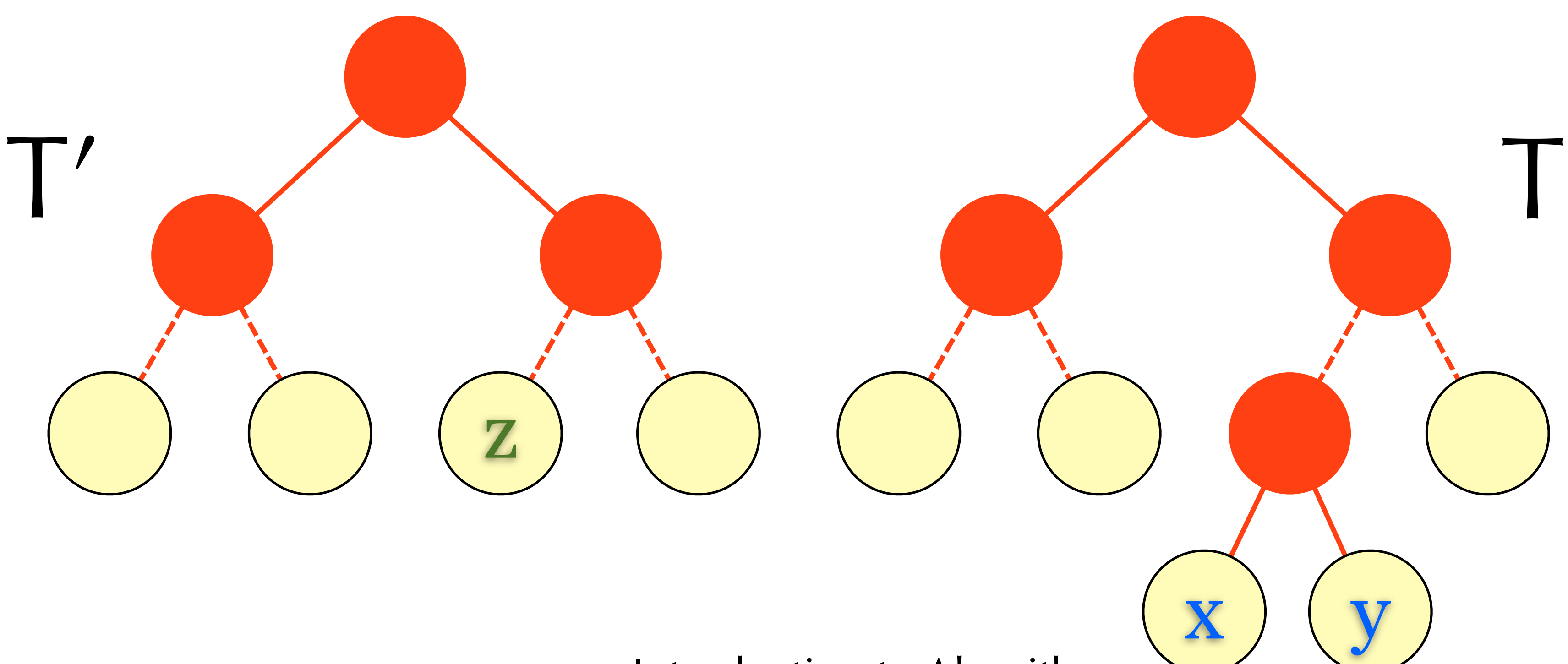
- Let d_c be the length of codeword of c.
- $\text{cost}(T') = \text{cost}(T) + f_\alpha d_x + f_x d_\alpha - f_x d_x - f_\alpha d_\alpha$
  $= \text{cost}(T) + (f_\alpha - f_x)(d_x - d_\alpha) \leq \text{cost}(T)$
  $\text{cost}(T) + (\quad \geq 0 \quad)(\quad \leq 0 \quad)$
  - T' is optimal, too.
- $\text{cost}(T'') = \text{cost}(T') + f_\beta d_y + f_y d_\beta - f_y d_y - f_\beta d_\beta$
  $= \text{cost}(T') + (f_\beta - f_y)(d_y - d_\beta) \leq \text{cost}(T')$
  $\text{cost}(T') + (\quad \geq 0 \quad)(\quad \leq 0 \quad)$
  - T'' is optimal, too. We are done.

Lemma 16.3

- Let C be a given alphabet with frequency f_c defined for each character $c \in C$.
- Let x and y be two characters in C with minimum frequency.
- Let C' be the alphabet C with the characters x and y removed and a new character z added, so that $C' = C \setminus \{x, y\} \cup \{z\}$.
- Define frequency of characters in C':
  - $f_z = f_x + f_y$
  - The frequency of other characters remain unchanged.
- Let T' be any tree representing an optimal prefix code for the alphabet C'. Then the tree T , obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C .

Proof

- cost(T)
  $=\text{cost}(T')-d_z f_z+d_x f_x+d_y f_y$
  $=\text{cost}(T')-d_z f_z+(1+d_z)f_x+(1+d_z)f_y$
  $=\text{cost}(T')+f_x+f_y-d_z(f_z-f_x-f_y)$
  $=\text{cost}(T')+f_x+f_y$



Proof

- BWOC, T is not optimal for C, but T^* is.
- WLOG (by lemma 16.2), x and y are siblings in T^*.
- By removing the leaves representing x and y in T^*, we obtain T^{**}.
- Similar to the previous argument: $cost(T^{**})=cost(T^*)-f_x-f_y$ $<cost(T)-f_x-f_y=cost(T')$, a contradiction.

Alternative: Lemma 16.3a

- The original version is somewhat tedious.
- We can focus on the optimal solutions exhibiting the greedy choice property.
- Let T be an optimal tree for alphabet C satisfying lemma 16.2 and T' be the tree obtained from removing the leaves representing x and y in T.
- Then T' is an optimal tree for alphabet $C' = C \setminus \{x,y\} \cup \{z\}$ where z is represented by the parent x and y of in T and $f_z = f_x + f_y$.

Homework

- How much space is needed to store a Huffman encoding tree?
- Design an algorithm to construct an optimal ternary (3-ary) prefix code.