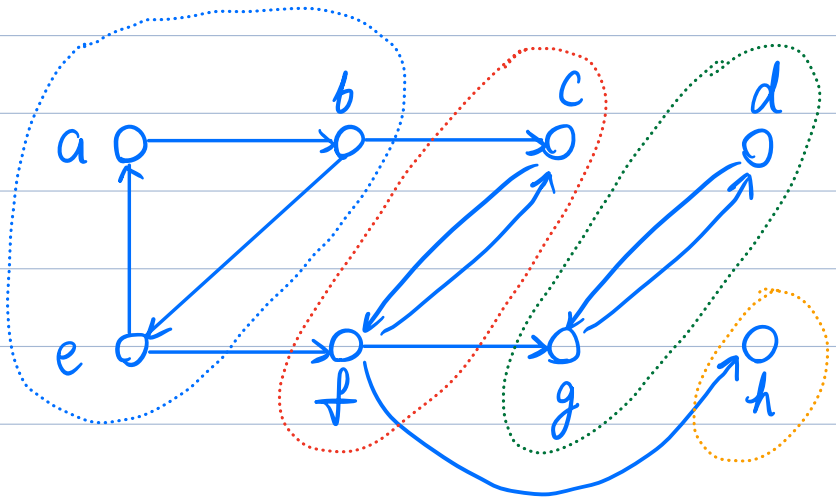
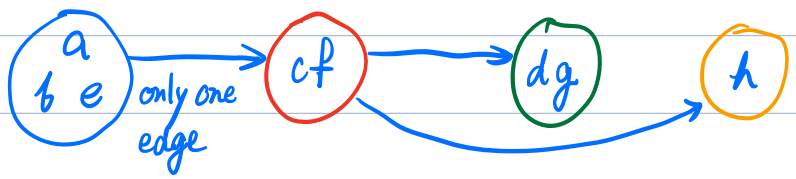
Recap: Strongly Connected Components (SCC):

Given a directed graph $G=(V,E)$, find its SCCs.





G^{scc}: Component graph

Key Lemma: the "virtual" walk above on G^{scc} induced by DFS on G is a valid DFS exploration of G^{scc}.

DFS Walk on G $\Rightarrow$ DFS Walk on G^{scc} $\Rightarrow$ topological sort of G^{scc}

↓ ↙

The last vertex to finish is in the left-most component (in the topo. sort of G^{scc}).

To get the entire left-most component from some vertex u:

Run DFS-Visit(G^T, u), where G^T is G w/ all edges reversed.

Overall algorithm (Kosaraju-Sharir):

SCC(G):

1. Run DFS(G) to compute finish(u) for all $u \in V$.
2. Run DFS(G^T), but in the main outer loop, consider vertice in order of decreasing finish time.
3. DFS in step 2 gives a DFS forest. Output the vertices of each tree in the forest as a separate SCC.

```
DFS(G)
1.  time=0
2.  For all u ∈ V:
3.      color(u)=WHITE
4.      parent(u)=⊥
5.  For all u ∈ V:        ← here! Loop by decreasing finish time.
6.    If color(u)=WHITE:
7.      DFS-Visit(G,u)
```

Runtime: $O(|V|+|E|)$

See directed graph, think SCC!

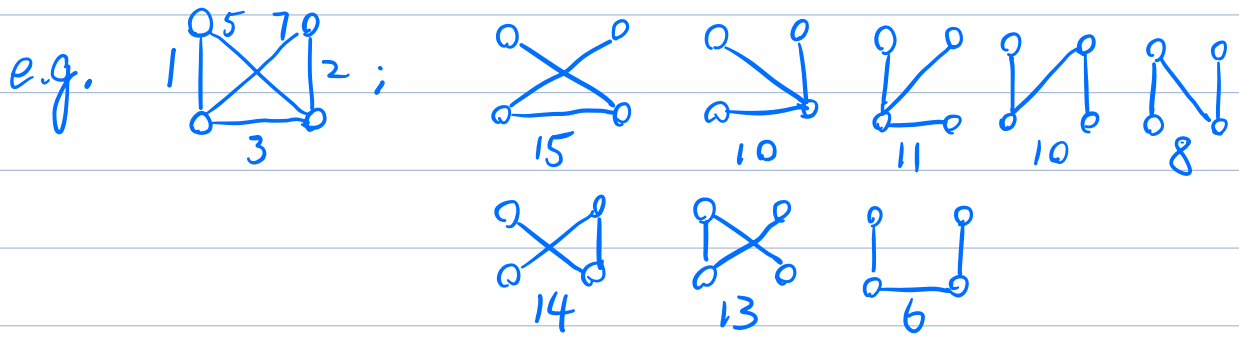
1. Minimum Spanning Tree (MST)

Input: G is an undirected, connected graph, with a weight $w(u,v)$ associated to each edge $\{u,v\} \in E$.

Goal: Find a spanning tree $T \subseteq E$ (acyclic and connects all vertices) with smallest total weight

$$w(T) = \sum_{\{u,v\} \in T} w(u,v)$$

e.g.



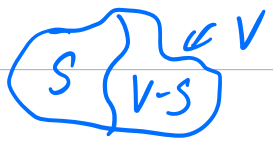
How to find MST? Grow the tree greedily!

```
GenericMST(G, w):
    A = ∅
    While A is not a spanning tree
        find {u,v} ∈ E safe for A
        A = A ∪ {u,v}
    Return A
```

Loop Inv.: A is subset of some MST

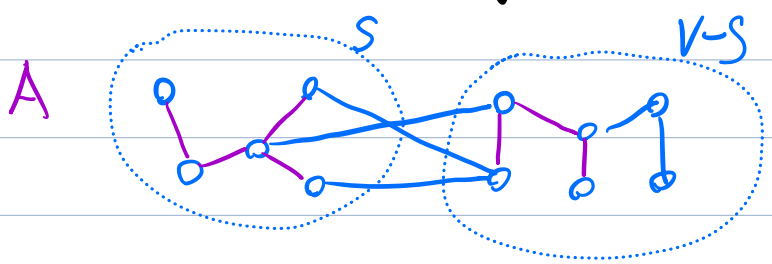
{u,v} safe for A if A ∪ {u,v} also a subset of some MST

By loop inv., ∃ some safe edge. but how do we find it?

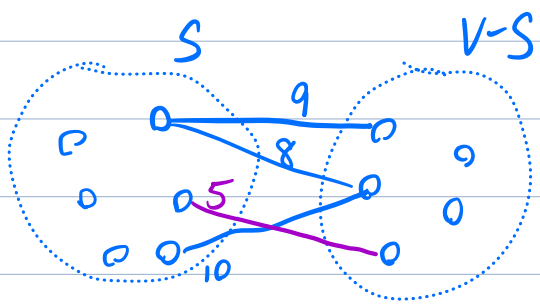


Some Definitions First:

- A cut of $G=(V,E)$ is a partition of vertices $(S, V-S)$
- Edge $\{u,v\}$ (↑ same as $\{v,u\}$) crosses cut $(S, V-S)$ if $u \in S$ and $v \in V-S$
- Cut respects A if no edge in A crosses the cut



- $\{u,v\}$ a light edge crossing a cut if it has minimum weight of any edge crossing the cut



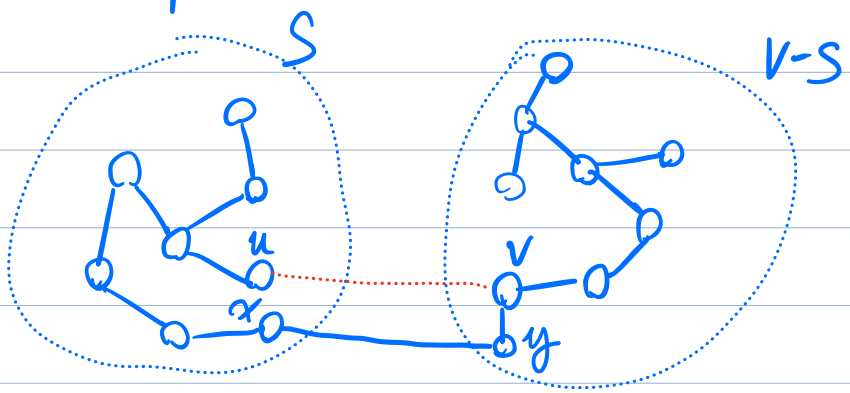
light edge

Thm: Let $A \subseteq T \subseteq E$ for some MST T, $(S, V-S)$ be a cut respecting A, and $\{u,v\}$ be a light edge crossing $(S, V-S)$. Then $\{u,v\}$ is safe for A.

Proof: Let T be a MST containing A. If $\{u,v\} \in T$, we're done.

So assume $\{u,v\} \notin T$.

Since T is a spanning tree, $\exists$ path in T from u to v. Further, since $u \in S$ and $v \in V-S$, some edge (x,y) in that path crosses the cut $(S, V-S)$.



Since $\{u,v\}$ is a light edge,

$$w(u,v) \leq w(x,y)$$

Let $T' = T \setminus \{\{x,y\}\} \cup \{\{u,v\}\}$. It is a spanning tree, and

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

But T was MST, so T' is also MST $\Rightarrow \{u,v\}$ is safe

$\blacksquare$

2. Kruskal's Algorithm

Idea: Put each vertex in its own connected component. Then repeatedly add the edge with lowest weight that connects two connected components to A.

Correctness: Let one connected component be C, then the edge we added is a light edge for the cut $(C, V-C)$, and is therefore safe.

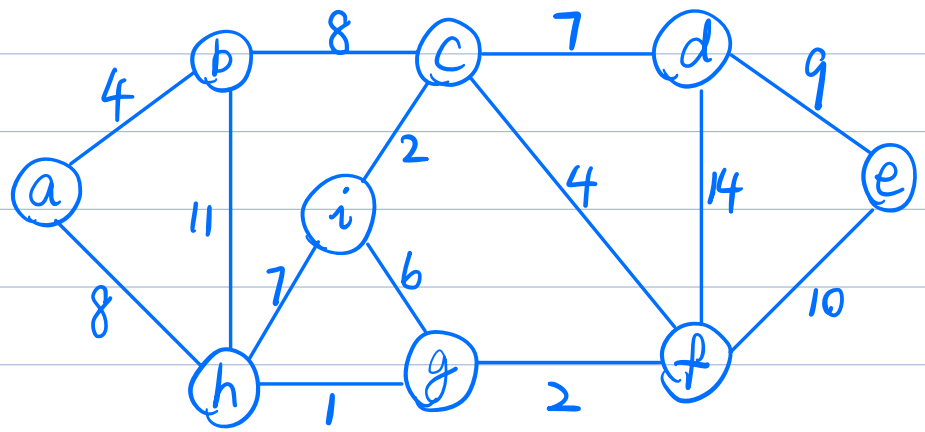
```
MST-Kruskal(G=(V,E), w):
1.    A=∅
2.    For all v∈V:                    } |V| · T(MakeSet)
3.        MakeSet(v)                  } // each vertex in its own connected comp.
4.    Sort E in non-decreasing order  |E| log|E| ≤ 2|E|log|V|
5.    For {u,v}∈E:
6.        If FindSet(u)≠FindSet(v): // u and v in diff. comp.
7.            A=A∪{u,v}
8.            Union(u,v) // merge u's and v's components
9.    Return A
```

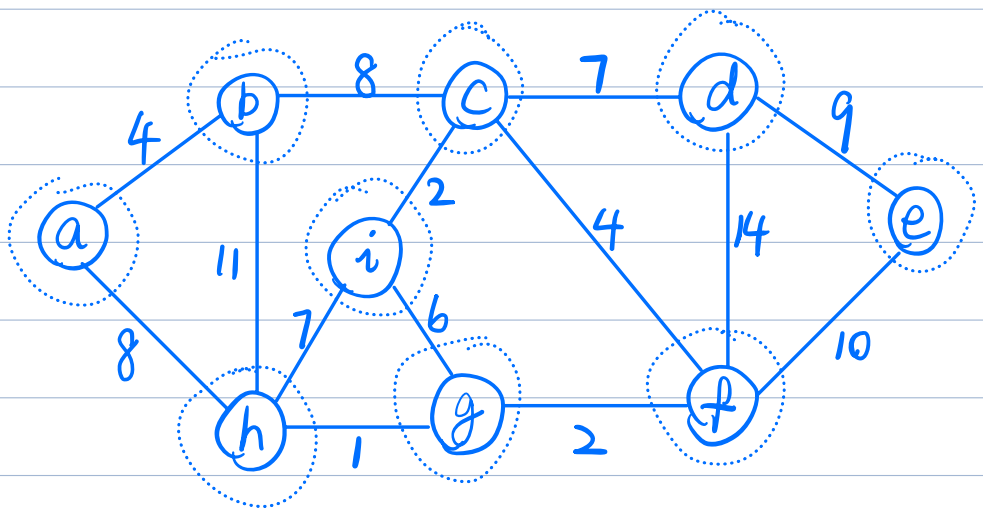
Runtime: $|V| \cdot T(\text{MakeSet}) + T(\text{Sort } E) + |E| \cdot T(\text{FindSet}) + |V| \cdot T(\text{Union})$

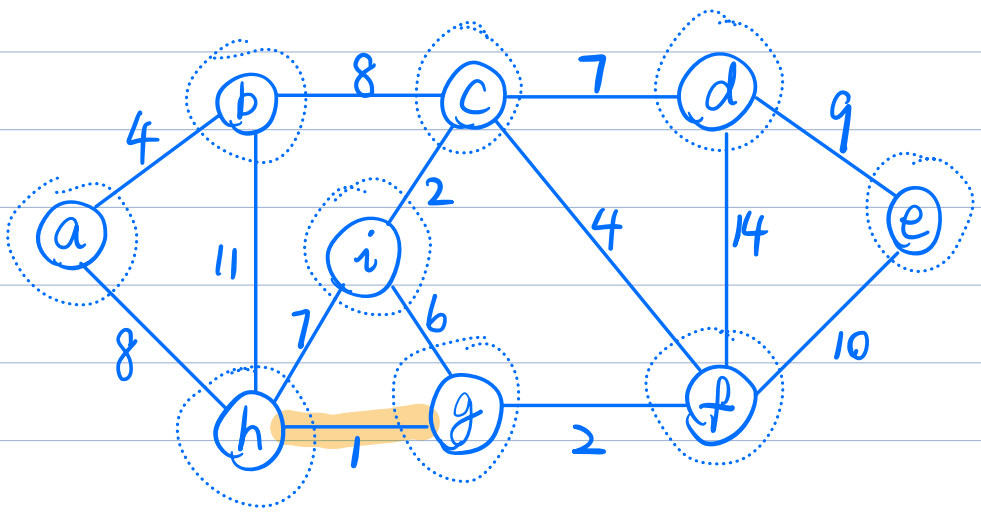
Disjoint Set (CLRS §19.3):
MakeSet $\Theta(1)$
FindSet/Union $\Theta(\log |V|)$

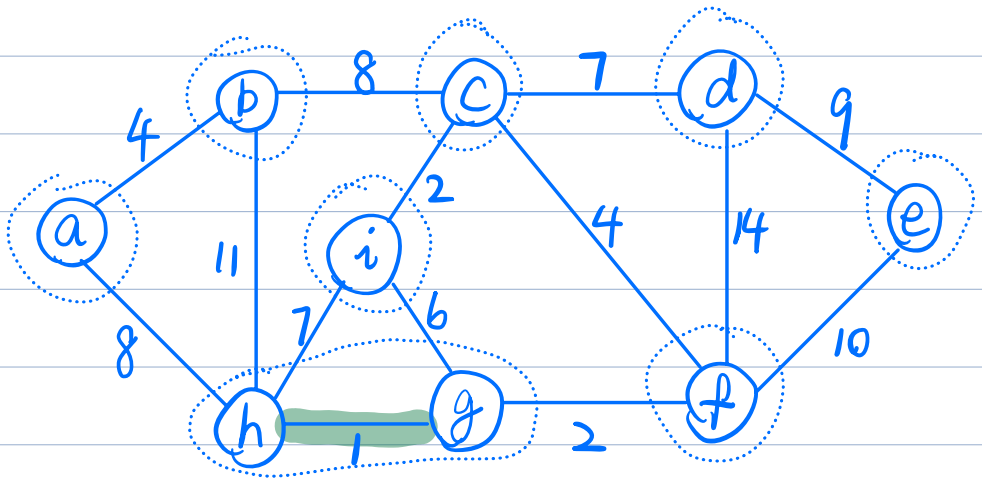
$\Rightarrow \Theta(|V| + |E| \log |V| + |E| \log |V| + |V| \log |V|) = \Theta(|E| \log |V|)$

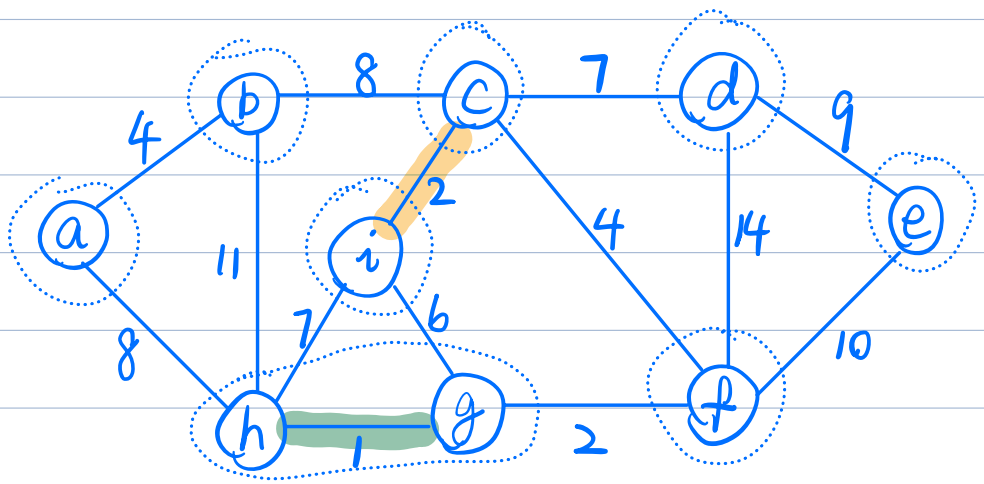
e.g.

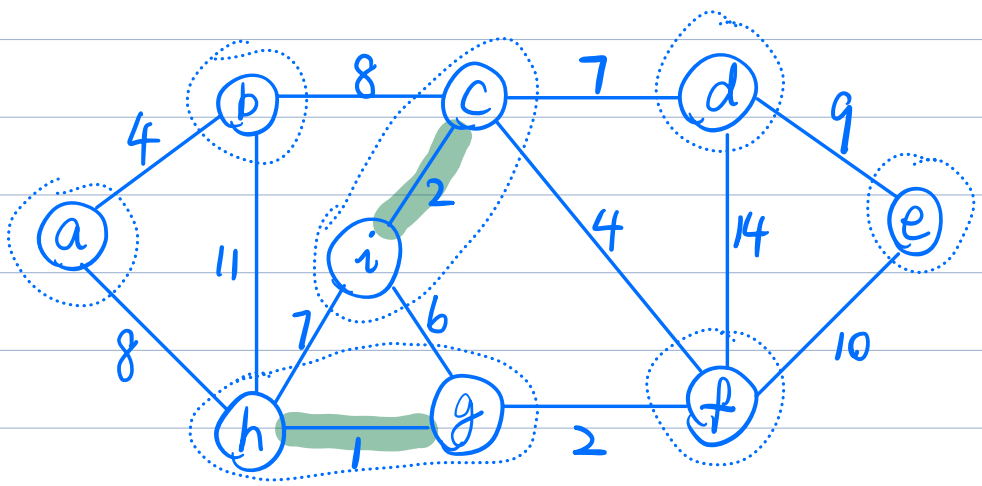

a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
14
9
10
1
2

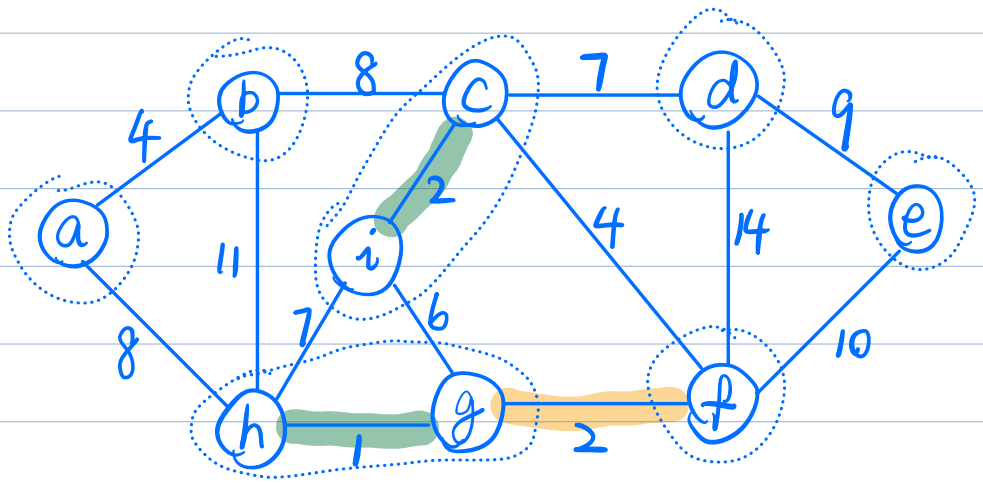

a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
14
9
10
1
2

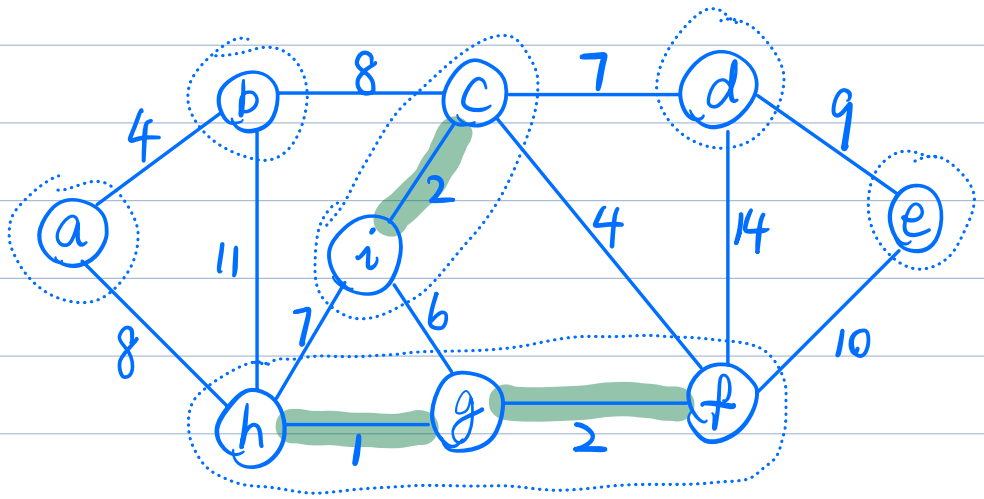

a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
14
9
10
1
2

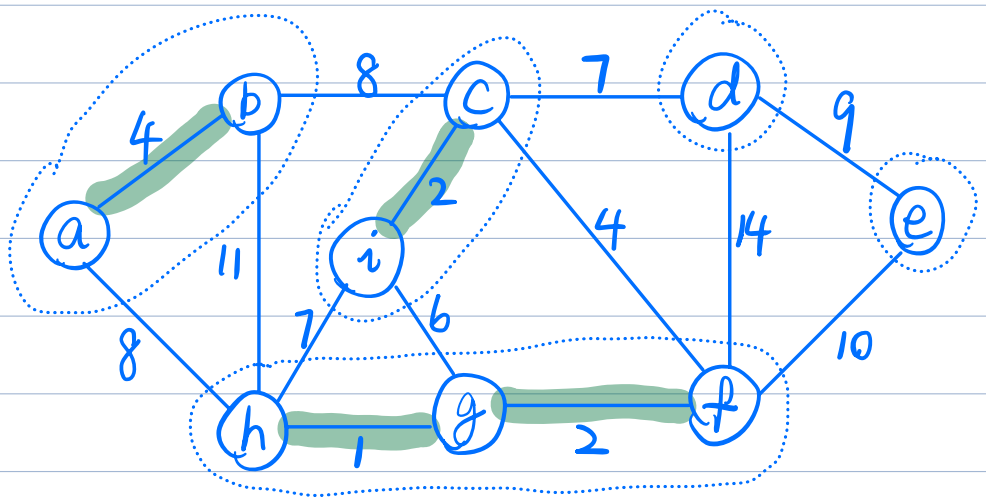

a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
2
4
14
7
6
10
8
1
2

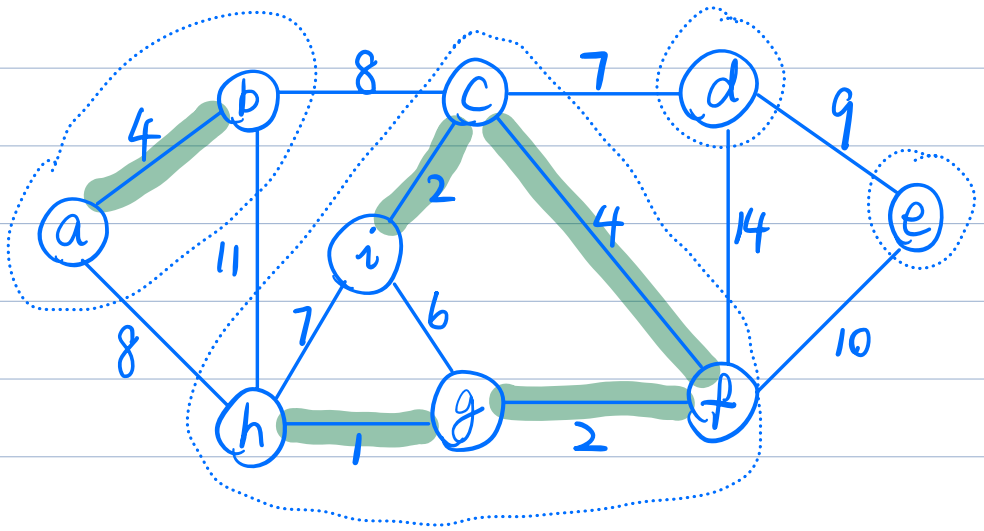

a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
2
4
14
7
6
10
8
1
2

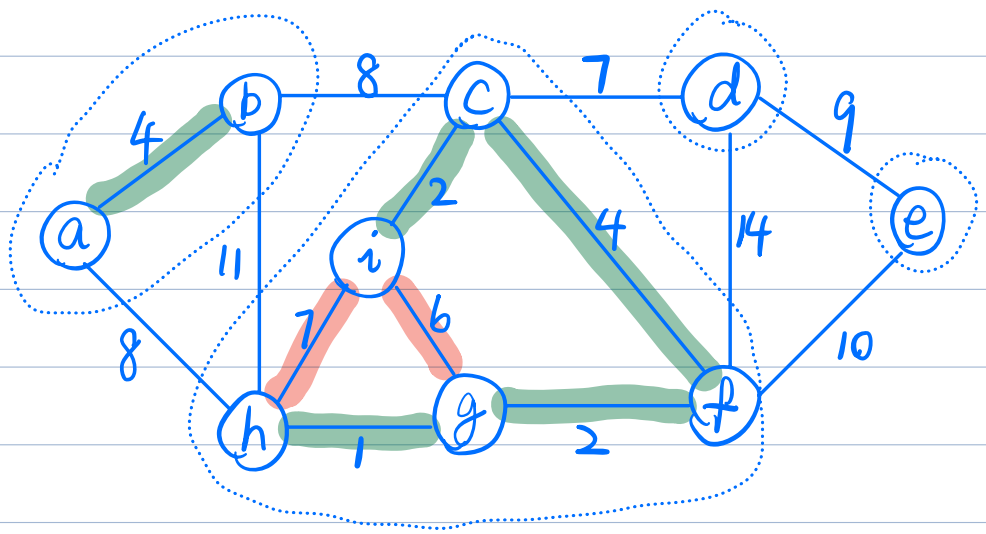

a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
2
4
14
7
6
10
8
1
2

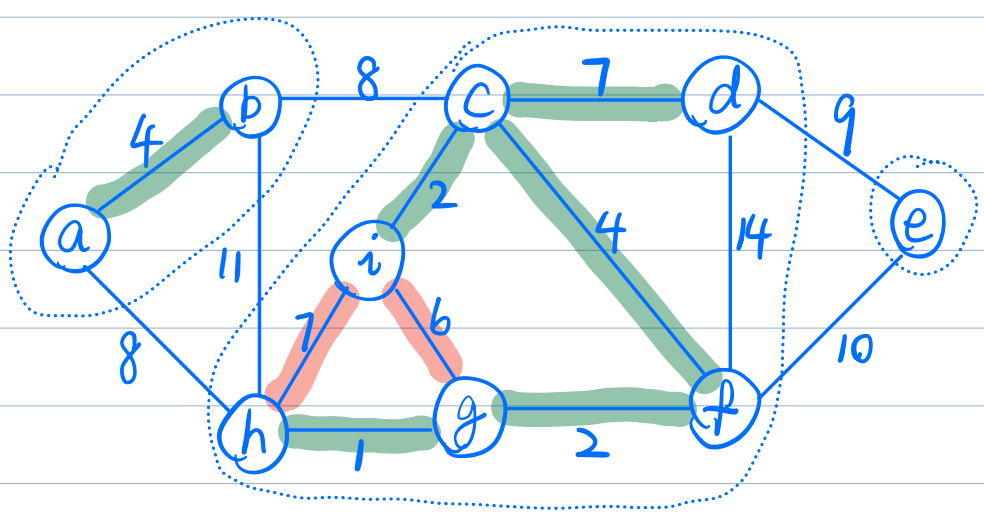

a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
10
2
4
8
7
6
1
2

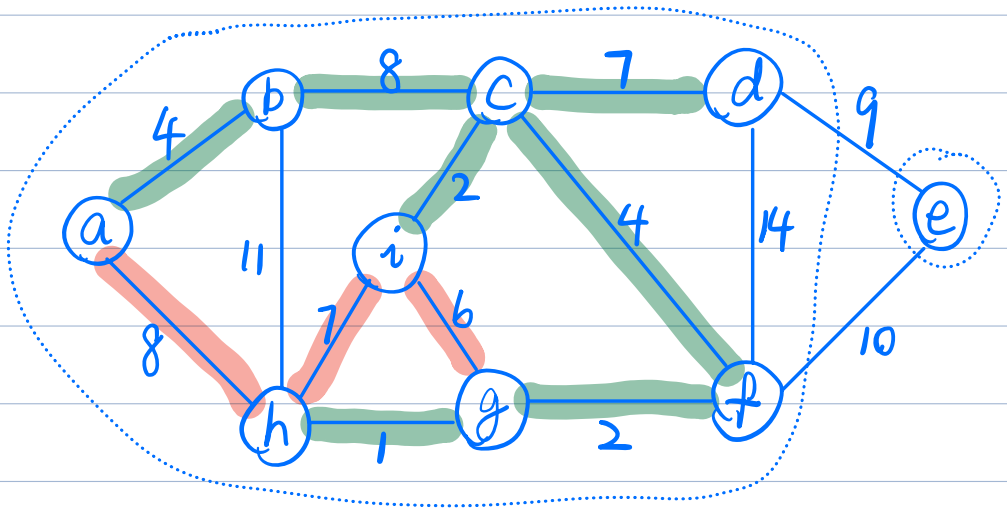

a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
10
2
4
8
7
6
1
2

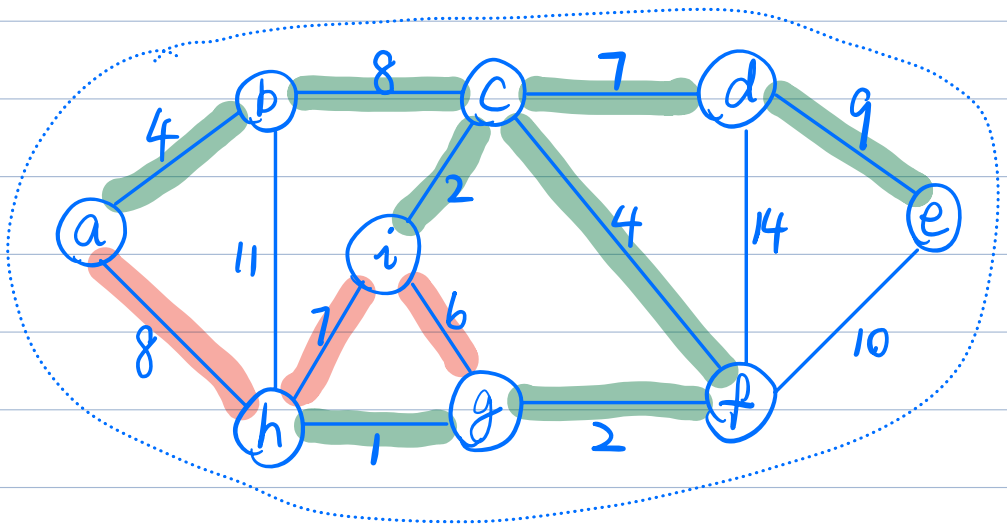

a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
10
2
4
8
7
6
1
2


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

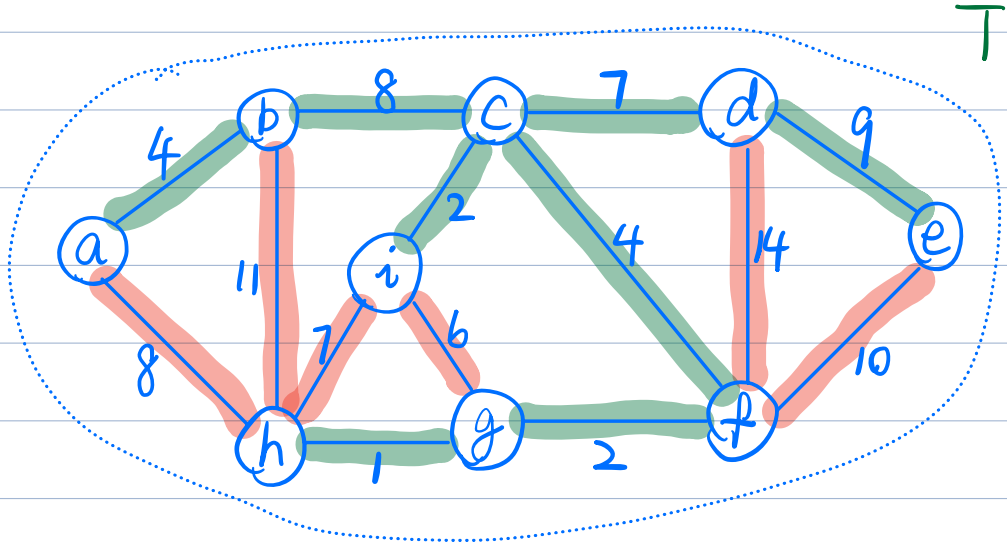

a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
10
6
1
2


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


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


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

T:


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