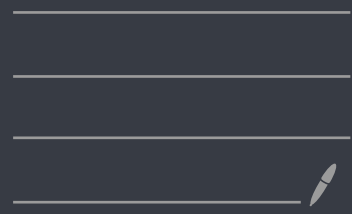
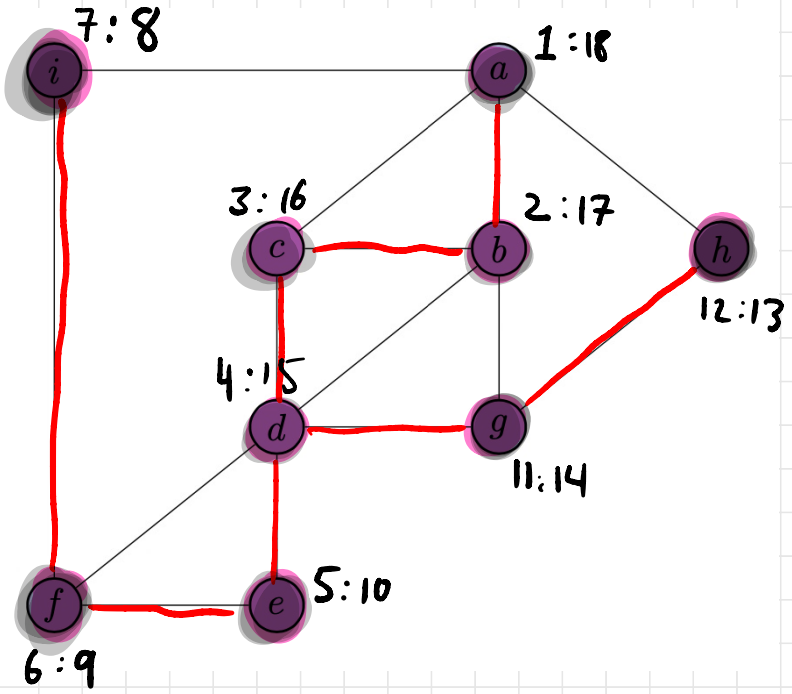
CSCE 310H - Fall 2021

Graph Algorithms





Start at (a)

- keep track of: discovery time
- finish time
- Colors:

  white = unvisited

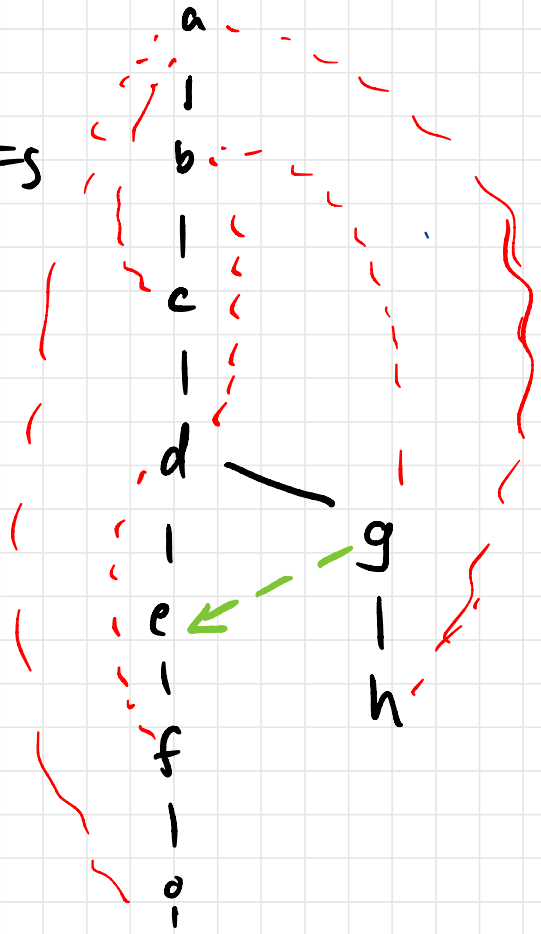
  gray = visited, but not processed

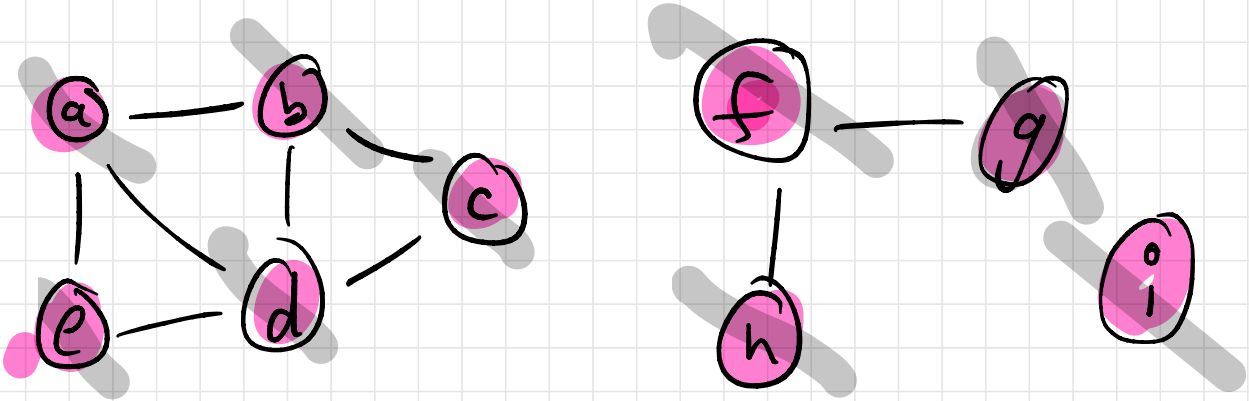
  black = processed.

DFS Tree:

- Tree edges: edges you traverse in DFS
- Forward: - ancestor to descented
- Back Edge: undir: same
- Cross Edges: all other edges.

DFS on undir: not possible




a
b
c
e
d
f
g
h
i

DFS(G, v)

Input: Undirected graph $G = (V, E)$, (a start vertex $\underline{\underline{v}}^{*}$)

color each vertex white (unvisited)
count ← 1
} initialization

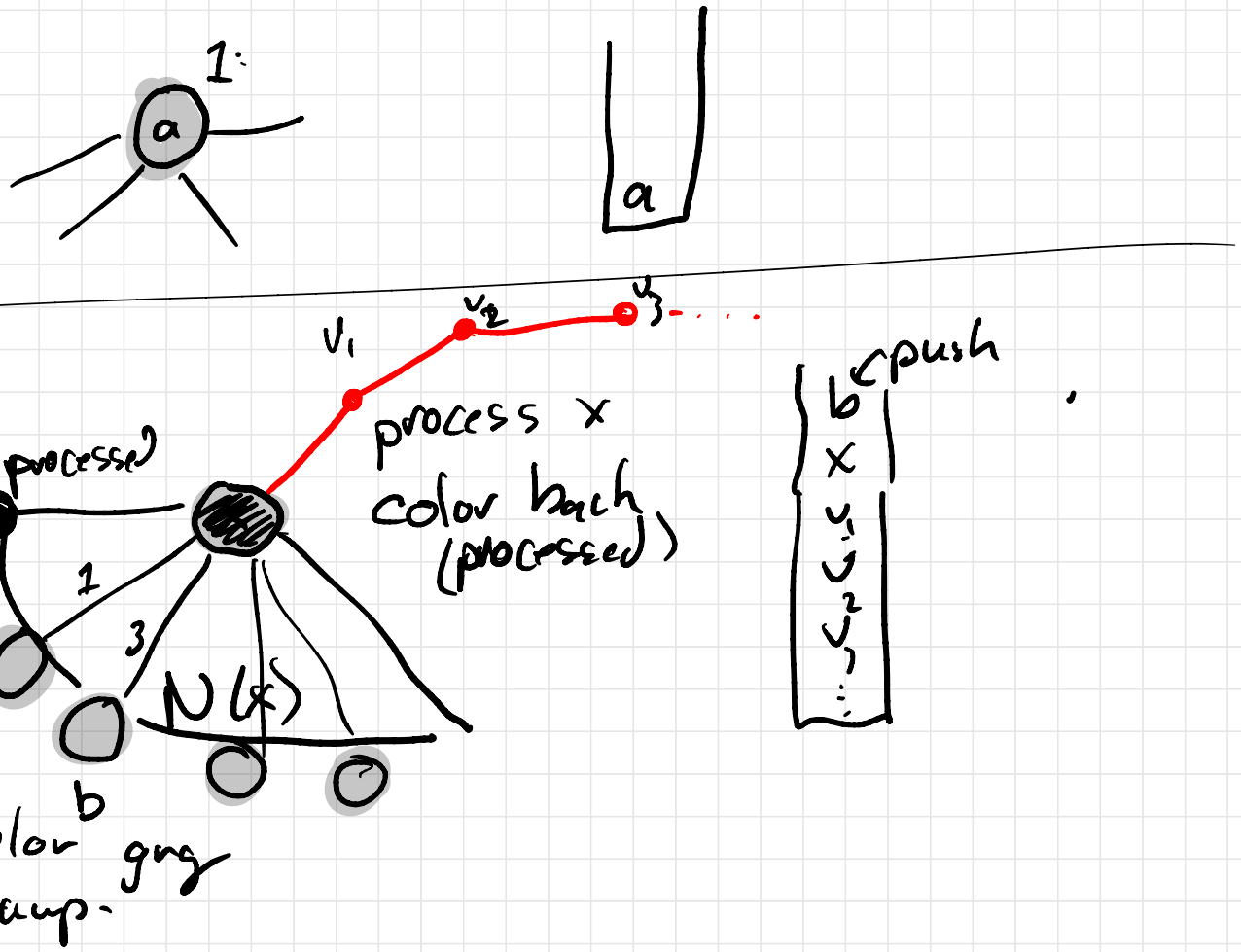
S ← empty stack
push v onto S
mark v with count (discovery time)
color v gray
} initial step in DFS

⋮

⟹

```
while S is not empty:
    count ++
    x ← S.peek()   // current vertex
    y ← next white (unvisited) vertex in N(x)
    if y = ∅   // no white vertices in N(x)
        //backtrack
        S.pop()
        process x
        color x black (processed)
        stamp x (finish Time
    else
        S.push(y)
        color y gray, stamp y (discov Time)
```

could be any criteria
- lexicographic
- random
- by weight.


1:
a
a
v_1
v_2
v_3
process x
color black
(processed)
processed
1
3
z
N(x)
b
color gray
stamp.
push
b
x
v_1
v_2
v_3

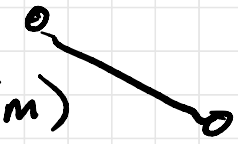
```
DFS-MAIN(G)
    Input: G = (V, E)
    color all vertices white
    for each v ∈ V:
        if v is white
            DFS(G, v)
```

Complexity of DFS?

- each vertex is pushed / popped exactly once
- while loop executes: $2n$ times $O(n)$

Elem. Operation?

- Examining an edge:
  each is examined twice: $2m = O(m)$



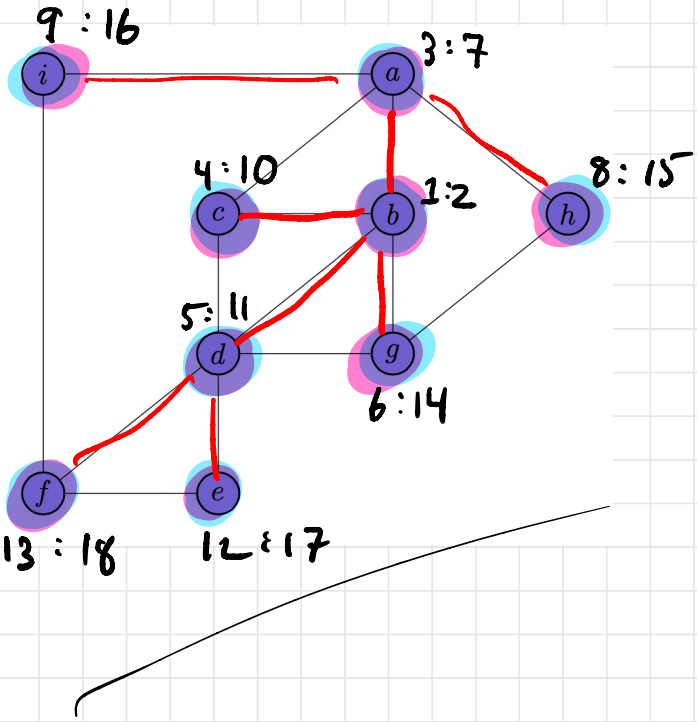
- Processing a node
  each is only processed once: $n = O(n)$

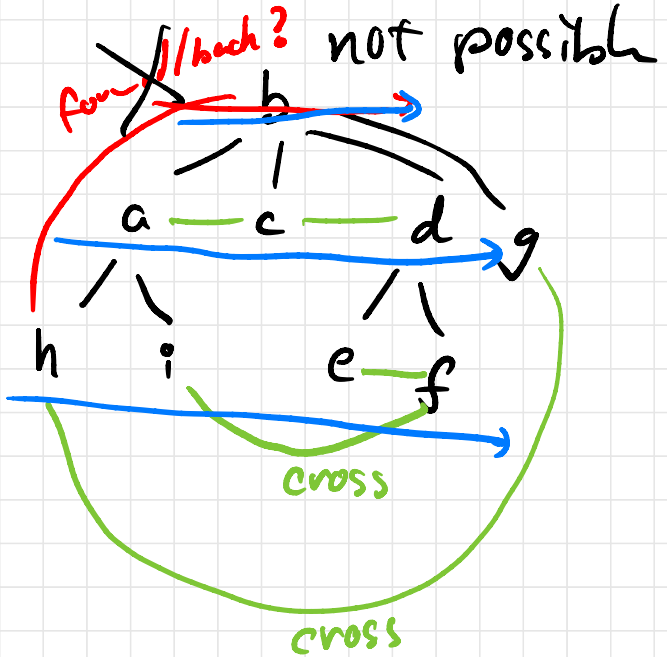
$$0 \leq |E| = m \leq \binom{n}{2}$$

$$m \in O(n^2)$$

$O(n)$ vs $O(m)$ vs $O(n^2)$

$$O(n+m)$$


9 : 16
3 : 7
4 : 10
1 : 2
8 : 15
5 : 11
6 : 14
13 : 18
12 : 17
i
a
c
b
h
d
g
f
e


fwd
b/back?
not possible
b
a
c
d
g
h
i
e
f
cross
cross

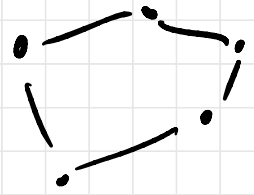
C_n a graph $|V| = n$

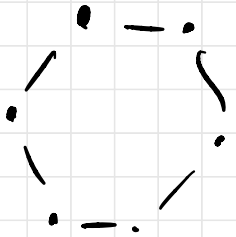
$|E| = n$

$G \cong C_n$

C_3 

C_4 

C_5 

$C_6 \cong$ 

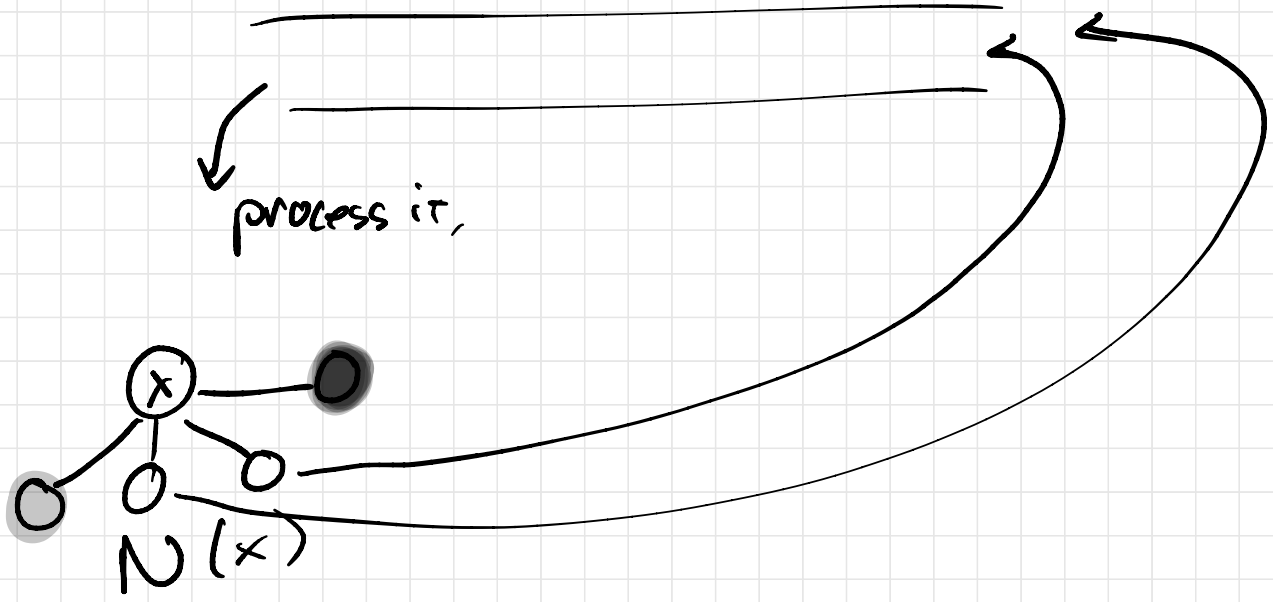
```
BFS-MAIN(G)
  Input: A graph G = (V, E)
    Foreach v ∈ V:
    └ color v white (unvisited)
    count ← 0
    Foreach v ∈ V:
    │   if v is white (unvisited)
    │   └ BFS(G, v)
```

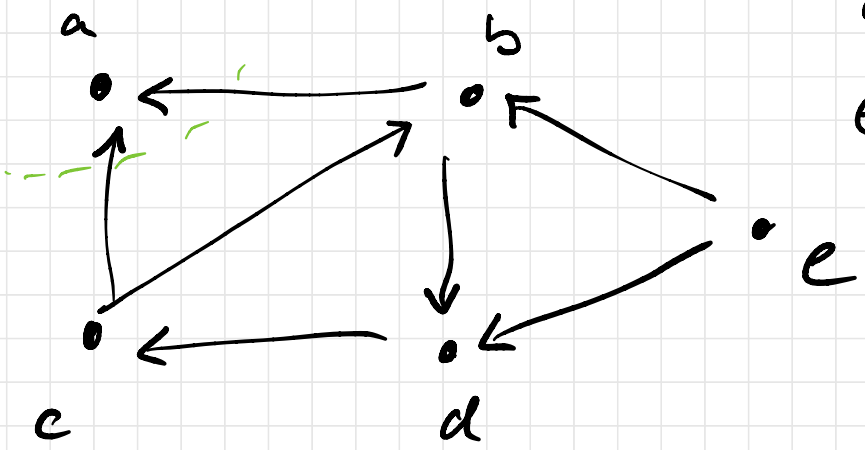
BFS(G, v)

Input: A graph $G = (V, E)$, start vertex $v \in V$

init:
- count ++
- Q ← empty queue
- Q.enqueue(v)
- color v gray (visited), stamp v with count (discovery)

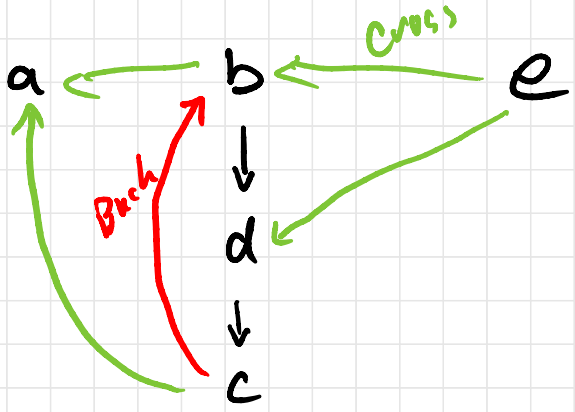
while Q is not empty
- x ← Q.dequeue
- for each white $y \in N(x)$ (could be lexicographic with ref to C)
  - count ++
  - color y gray, stamp y
  - Q.enqueue(y)
- count ++
- process x
  - color x black
  - stamp x w/ count (finish)

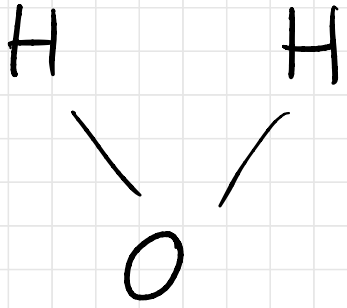

process it,
x
N(x)



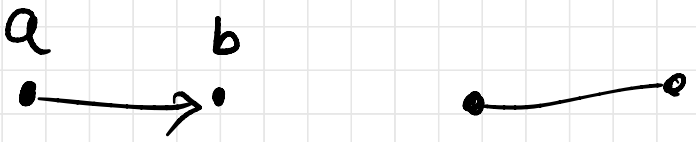
a is a sink

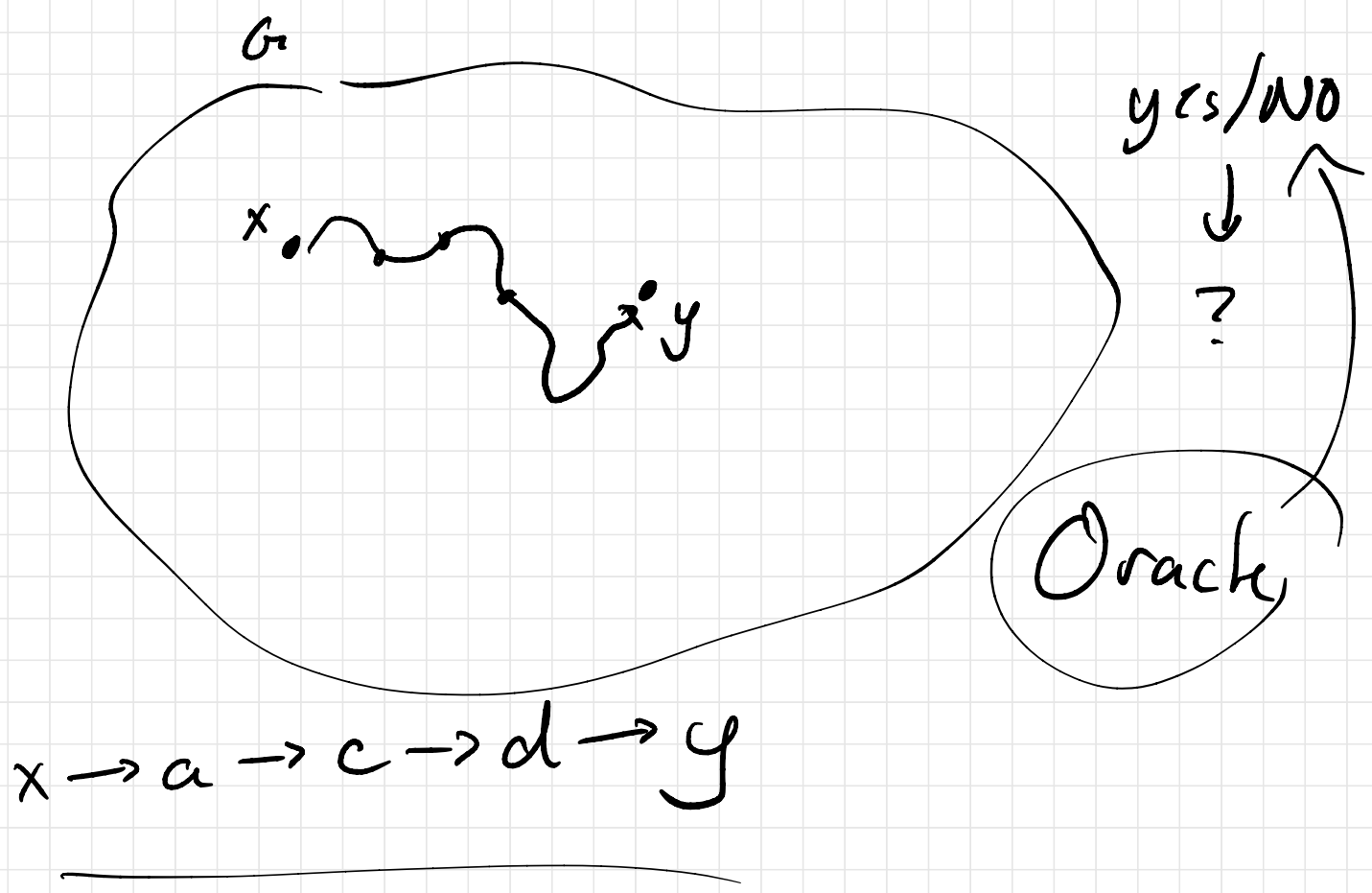
e is a source



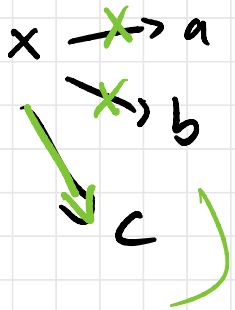


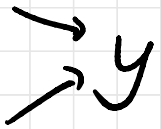
$$R \subseteq \{(a,b) \mid a,b \in A\}$$





$x \to a \to c \to d \to y$





$x \rightsquigarrow y$

yes

$a \rightsquigarrow y$

no

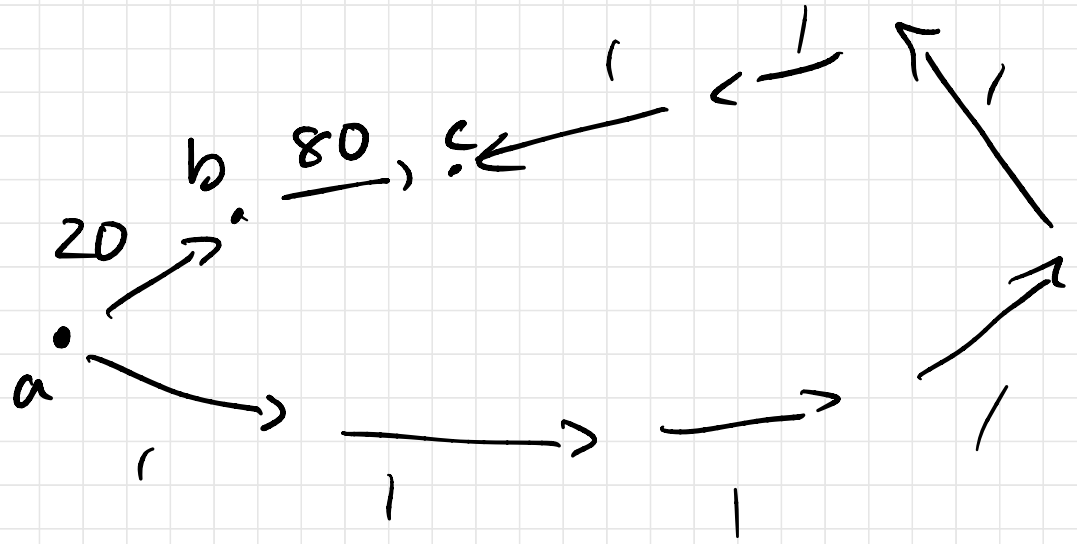
$b \rightsquigarrow y$

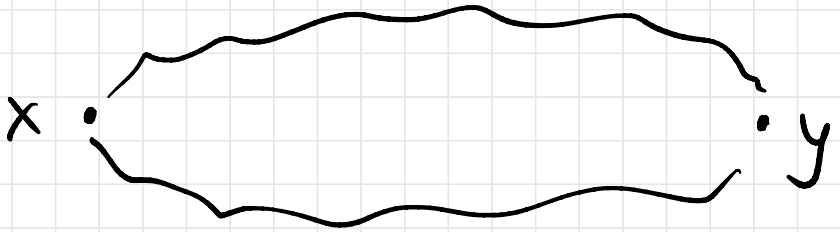
no

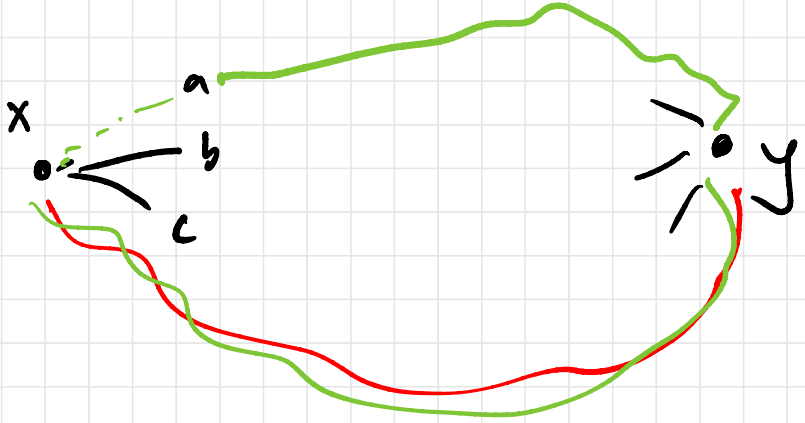
$x \to c \to ? \to ? \cdots y$

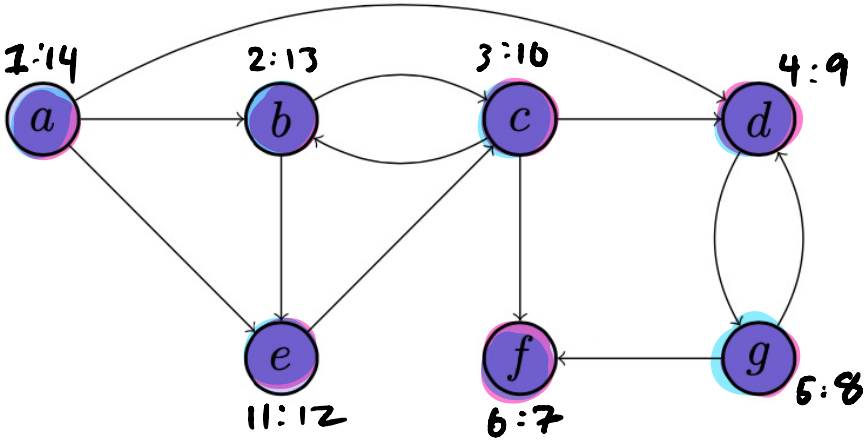
$c \rightsquigarrow y$

yes


a
b
c
20
80
1
1
1
1
1
1
1


x
y

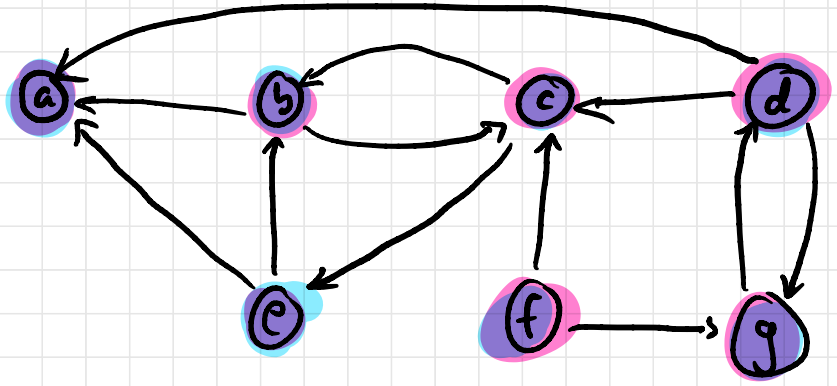

x
a
b
c
y

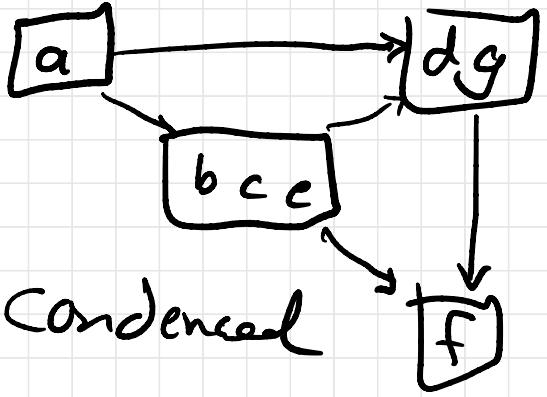


finish time

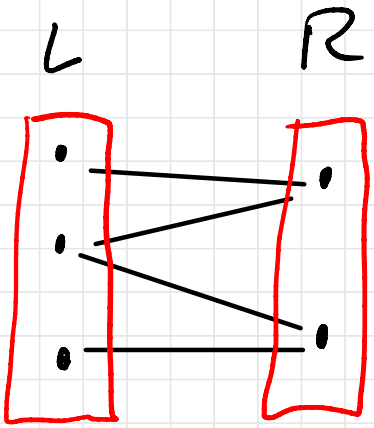
each time we start over, new node

DFS



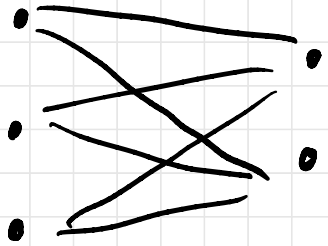


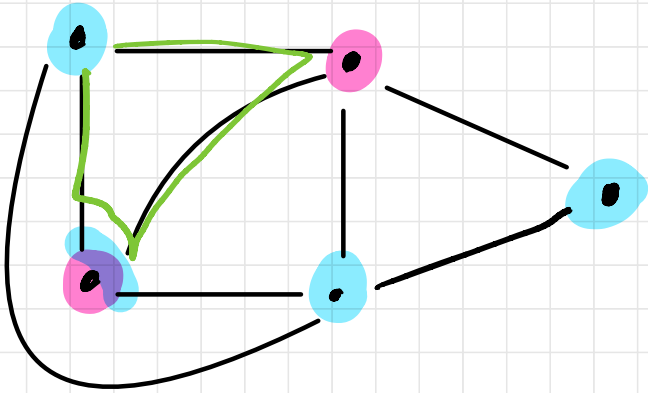
Condensed

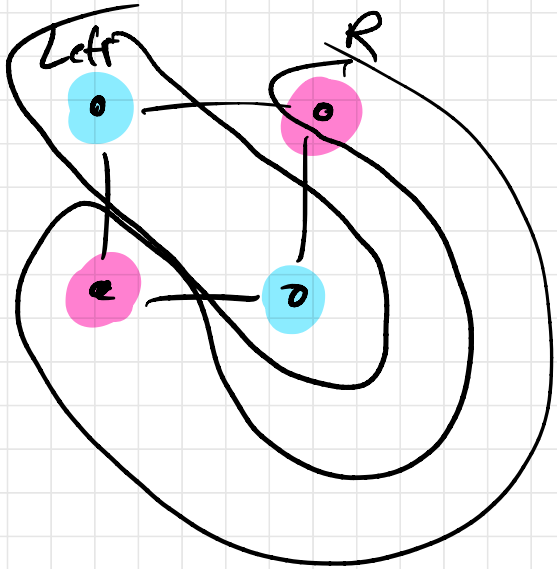


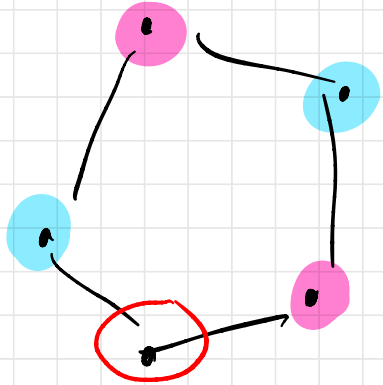
$K_{n,m}$

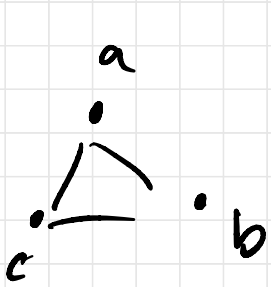
$K_{3,2}$:

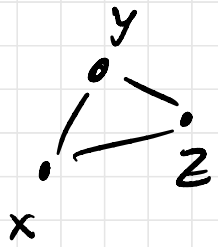


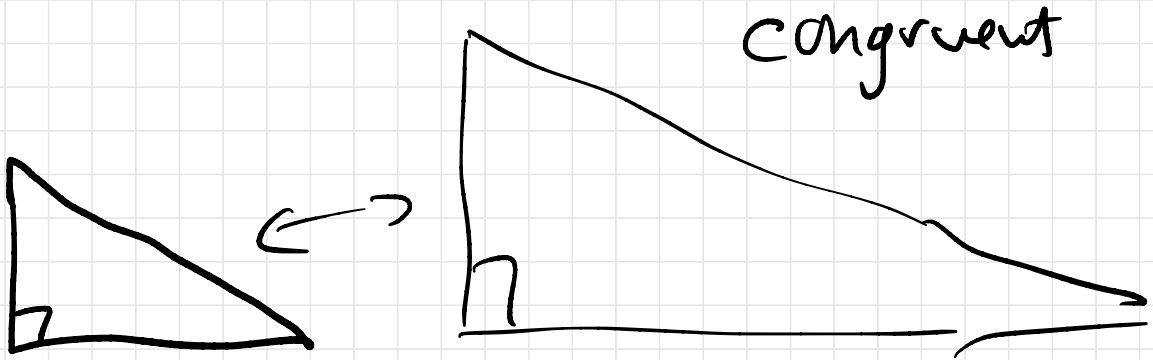


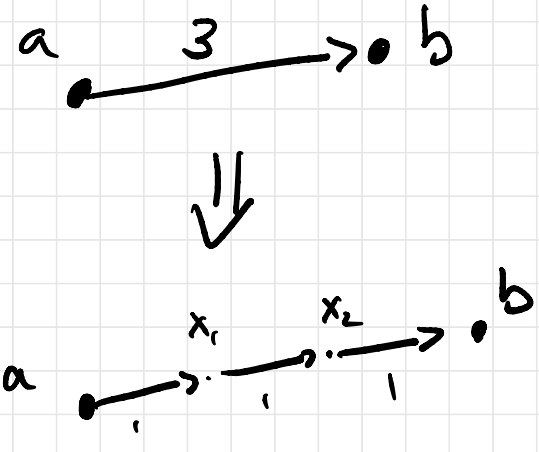

Left
R

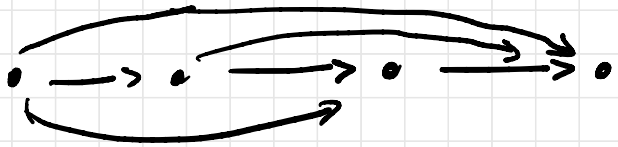


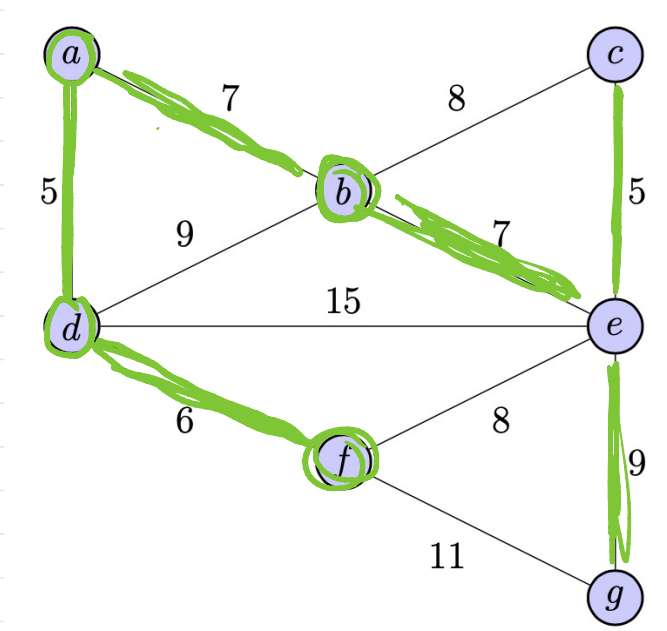

a
b
c


y
z
x


congruent


a
3
b
x1
x2
b
a
1
1
1





order

a d, 5 → added

c e, 5 → added

d f 6 → added

a, b 7 → added

b, e 7 → added

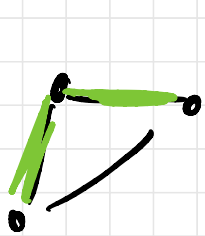
b, c 8 → ignore

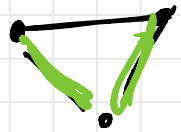
e f 8 → ignore

b d 9 → ignore

e g 9 → added

f g 11 → ?

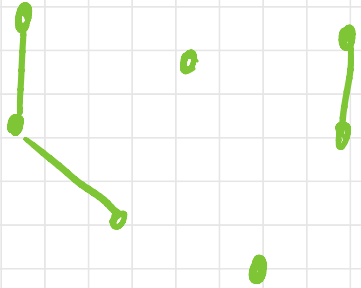




$n = 6$

spanning forest

$m = 4$



Fact: each graph is sparse

$m = O(n)$

KRUSKAL-MST(G)

Input: A weighted, undirected graph $G = (V, E)$

Output: A MST T

1 Sort the edges E in non-decreasing order

2 $E_T \leftarrow \emptyset$ // edge set of the tree

3 $V_T \leftarrow V$

4 for each $e_k \in E$ (sorted)

5 | if $(V, E_T \cup \{e_k\})$ is acyclic

6 | | add e_k to E_T

7 output (V, E_T)

if $|E_T| = n-1$

| break

↑ could be a forest.

Analysis:

Sorting: $|E| \log(|E|) = O(m \log(m))$

Acyclicity Testing: DFS $O(n+m) \Rightarrow O(n)$

- executed at most m times
- $O(m \cdot (n+m)) = O(m^2) = O(n^4)$
- Fact: each DFS is on a sparse graph $m = O(n)$

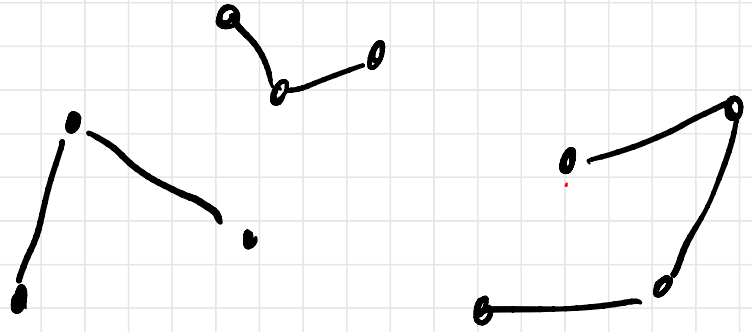
$$O(n^4) \Rightarrow O(n^2)$$

$$|E| > |V|$$

$$|V| = n$$

$$0 \leq |E| = m \leq \binom{n}{2}$$

$$n^2$$

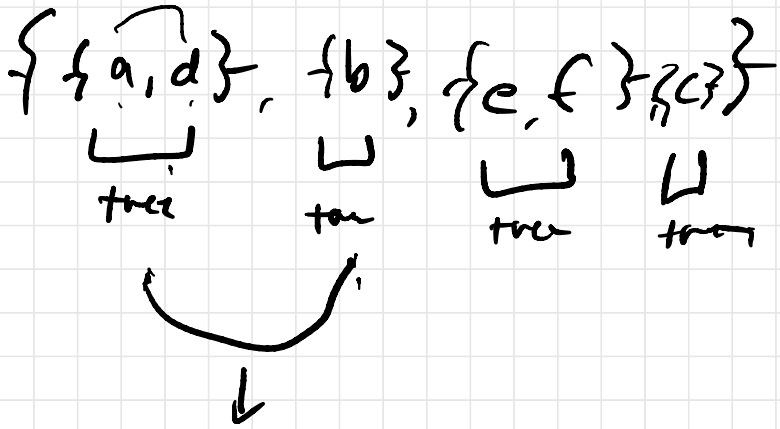


(x,y) => does adding (x,y) (induce) create a cycle

is x, y in the same connected subtree?

{a b c d e f}

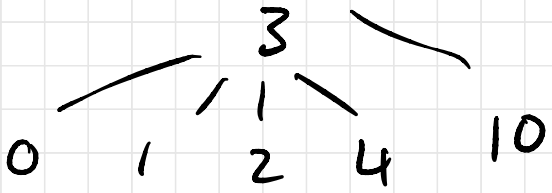
{a d} {a b} ...

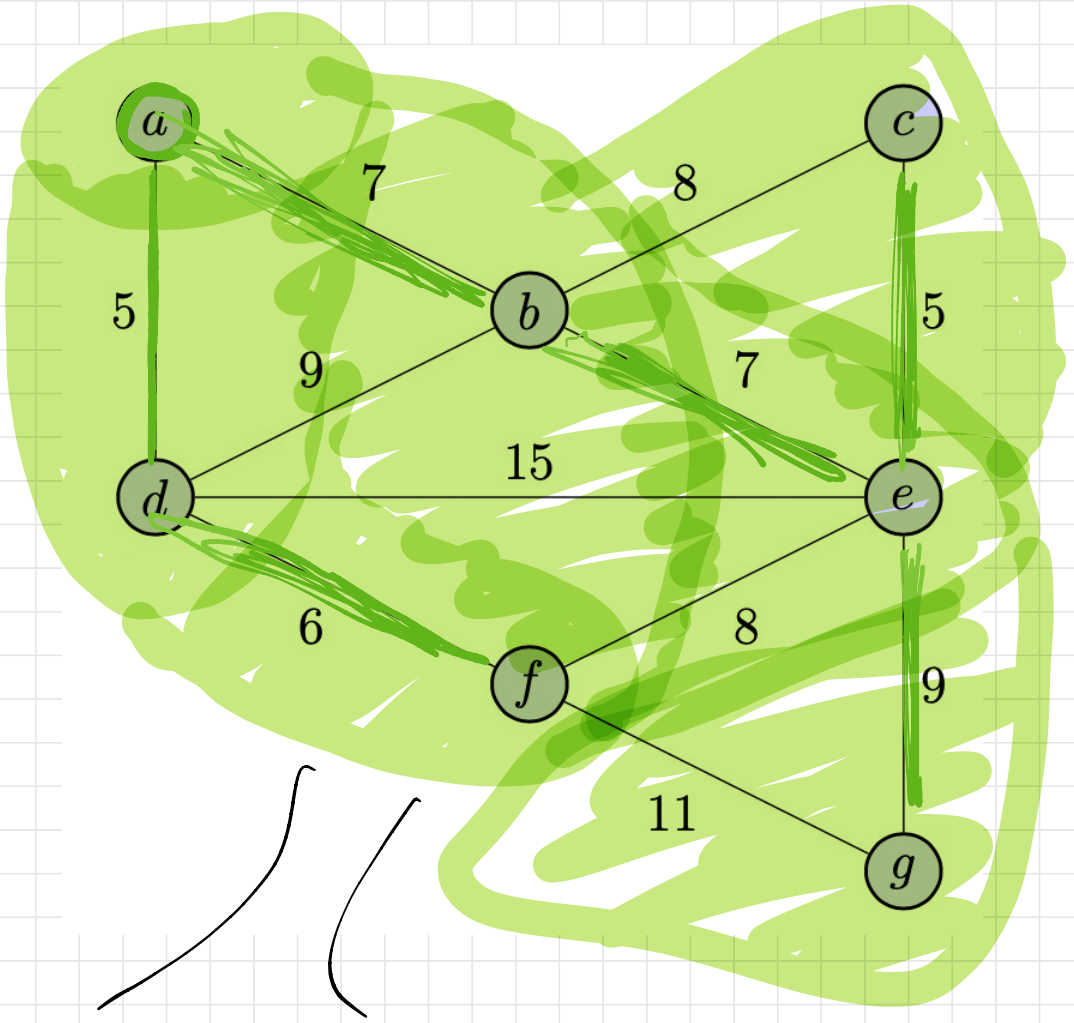


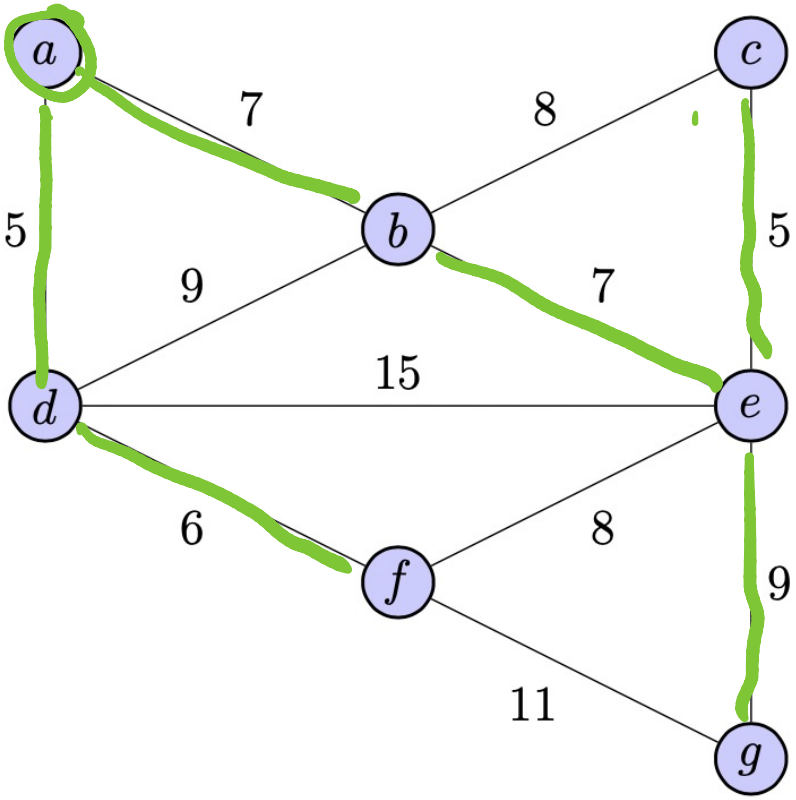
{a b d} {e f}, {c}

{a b d e f}, {c}

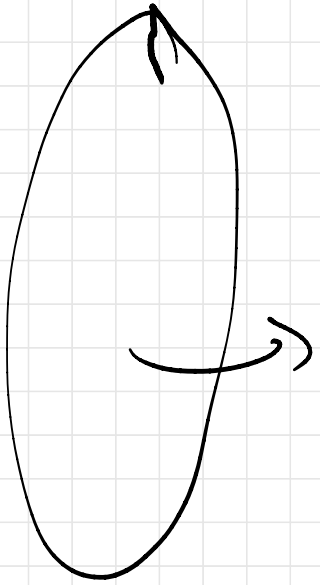
done


3
1
0
1
2
4
10


a
b
c
d
e
f
g
7
8
5
5
9
7
15
6
8
9
11



heap



Pose

(a, 0, -)

(d 5 a)

(f 6 d)

(b, 7, a)

(e, 7 b)

(c 5, e)

(g, 9 e)

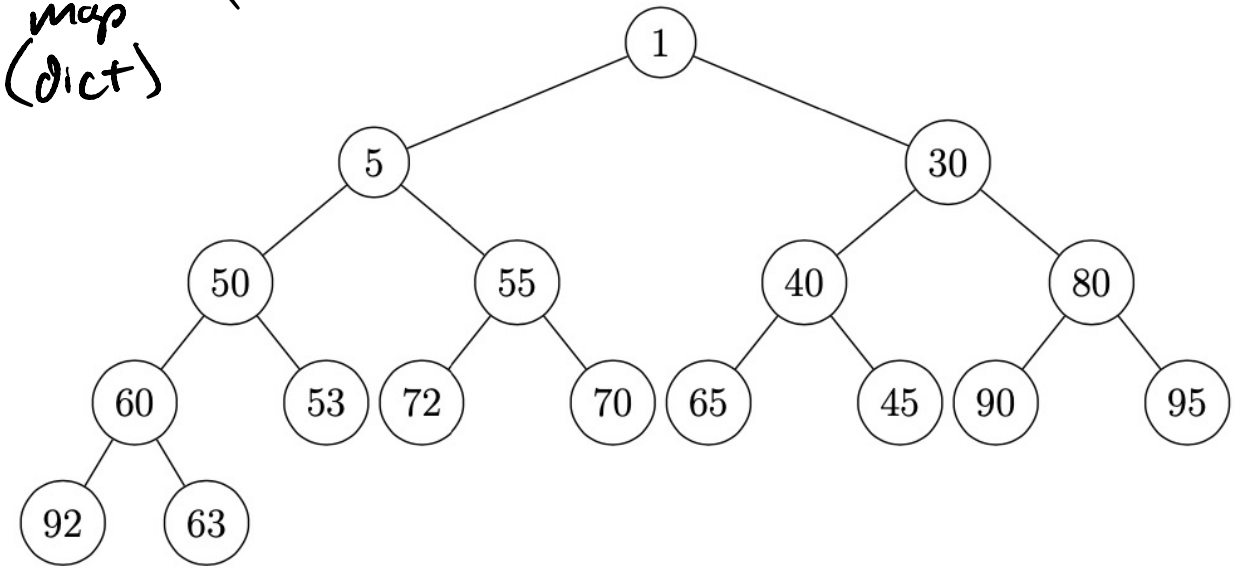
triples: vertex, ~~wright of~~ weight of the "frontier" edge, "predessesor"

key

50 → value Node of 50 (or index)

50 → 3

map (dict)



3 → (node of 3)

Prim's

Input: A weighted graph $G = (V, E)$

Output: A MST of G

$H \leftarrow$ min Heap

// stores triples: (v, p, w)

v -> vertex
p -> predecessor
w -> weight (priority)

$T = (V_T, E_T) \leftarrow (\emptyset, \emptyset)$

H.enqueue $(v_1, \emptyset, 0)$

for all other vertices $v_2 \dots v_n$:

  H.enque $(v_k, \emptyset, \infty)$

while H is not empty:

  $(x, p, w) \leftarrow$ H.getMin

  $V_T \leftarrow V_T \cup \{x\}$

  $E_T \leftarrow E_T \cup \{(x, p)\}$

  ⋮

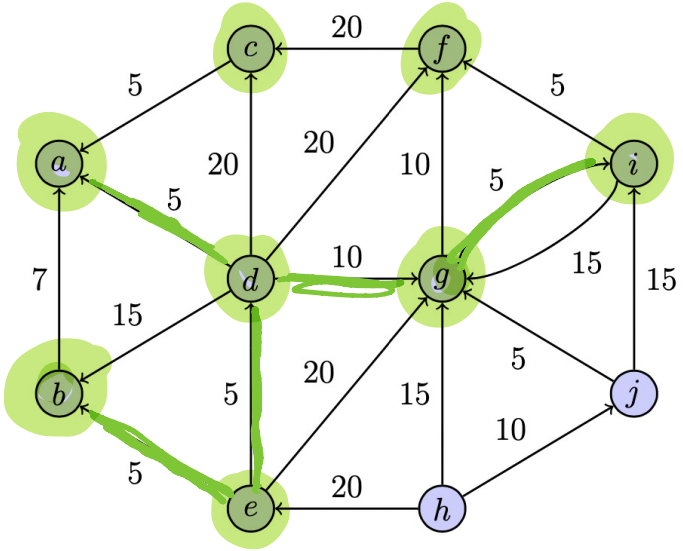
  for each $y \in N(x)$:

    H.update $(y, x, wt(x,y))$

output T

decrease priority if $wt(x,y)$ is less

you can get to y from x



Heap

Done

(e, ∅, 0)

(b, e, 5)

(d e 5)

(a, d, 10)

(g d 15)

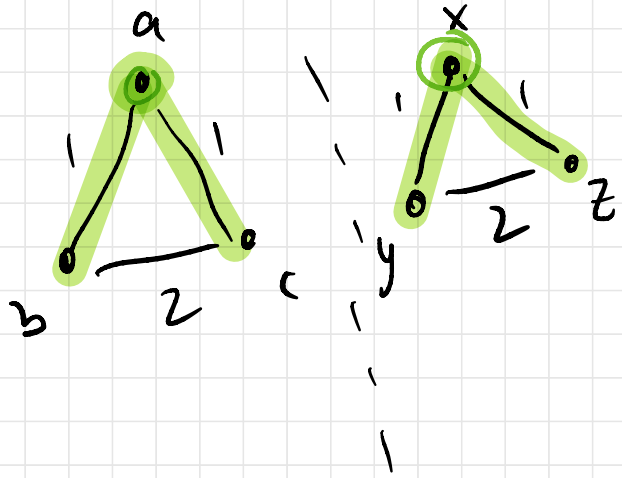
(i g 20)

(c d 25)

(h d 25)

(f d 25)

(j ∞)



Heap

(x - ∞) → 0

(y - ∞)

(z - ∞)

Done

(a - 0)

(b | a)

(c | a)

(x - 0)

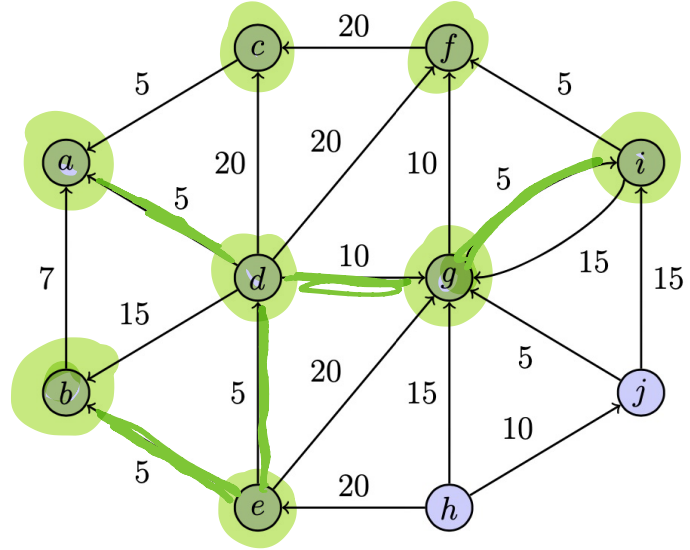
(y | x)

(z | x)

restart

(wt, node, node)

node: 0 - (n-1)



heap

Done

(e, ∅, 0)

(b, e, 5)

→ (d e 5)

(a, d, 10)

→ (g d 15)

→ (i g 20)

(c d 25)

(h d 20)

(f d 25)

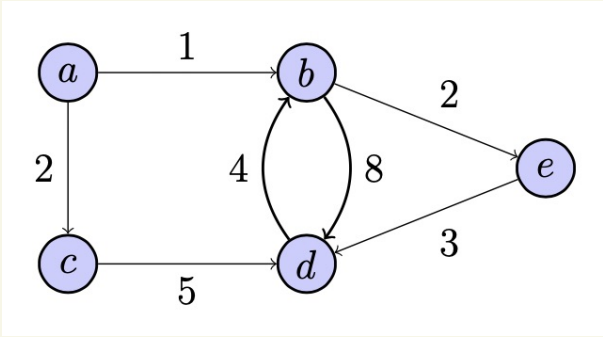
(j ∞)

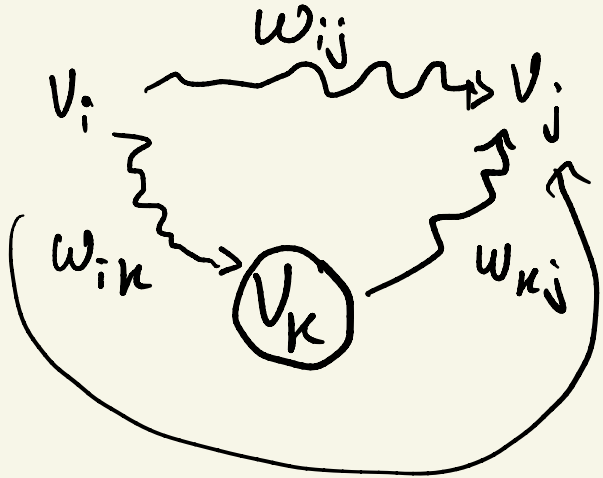
e → d → g → i 20

Input: triples from Dijkstras, destination vertex y, source s

Output: The path $s \rightsquigarrow y$

```
p = y
curr ← y
while curr ≠ s
    (a, b, w) ← triple with (curr, p, wt)
    p = b + p        prepend b (previous vertex) to the start of your path)
    curr ← b
p = s + p
output p
```



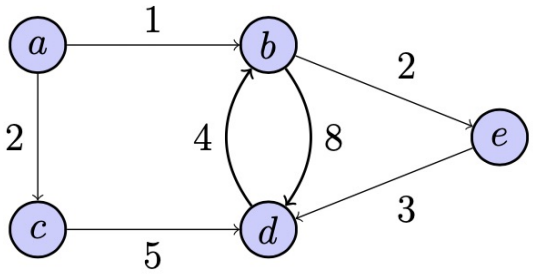


$w_{ik} + w_{kj}$

	a	b	c	d	e
a	0	1	2	∞	∞
b	∞	0	∞	8	2
c	∞	∞	0	5	∞
d	∞	4	∞	0	∞
e	∞	∞	∞	3	0

Choice:

$\min \{ w_{ij}, w_{ik} + w_{kj} \}$



	a	b	c	d	e
a	0	1	2	7	3
	∞	0	∞	8	2
	∞	∞	0	5	∞
	∞	4	∞	0	6
	∞	∞	∞	3	0

(d) Distance matrix after iteration $k = 2$

$k = 3 \quad (v = c)$

$d_{ad} = d_{1,4} = 9$

vs

$d_{1,3} + d_{3,4} = 2 + 5 = 7$

$d_{a,c} + d_{c,d} = 2 + 5$

$S_{14} \leftarrow$ update to $S_{ik} = S_{13}$

$= S_{ac}$

Floyd's

Input: A weighted Graph $G = (V, E)$

Output: A shortest distance matrix D, successor matrix S

$D \leftarrow (n \times n)$ matrix

for $i, j = 1 \dots n$

  $d_{ij} \leftarrow wt(v_i, v_j)$

  $s_{ij} \leftarrow j$ for all edges $(v_i, v_j) \in E$

  $s_{ij} \leftarrow -$ otherwise

$O(n^3)$

$= O(n \cdot n^2)$

Θ

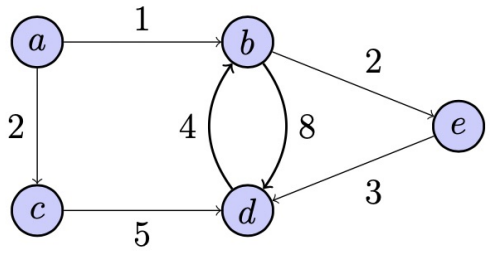
for $i = 1 \dots n$ ← for each pair of vertices

  for $j = 1 \dots n$

    for $k = 1 \dots n$ ← intermediate vertex

      $d_{ij} \leftarrow \min \{ d_{ij}, d_{ik} + d_{kj} \}$

      if d_{ij} changes, update $s_{ij} \leftarrow s_{ik}$



$$S = \begin{bmatrix} - & b & c & b & b \\ - & - & - & e & e \\ - & d & - & d & d \\ - & b & - & - & b \\ - & d & - & d & - \end{bmatrix}$$

(h) Successor matrix produced by Floyd's Algorithm

$a \to d$?

$S_{ad} = b$

$S_{bd} = e$

$S_{ed} =$

$a \to b \to c \to d$

Construct Shortest Path From Floyd's

Input: A successor matrix S from Floyd-Warshall, vertices v_i v_j

Output: shortest path $p: v_i \rightsquigarrow v_j$

$p \leftarrow v_i$

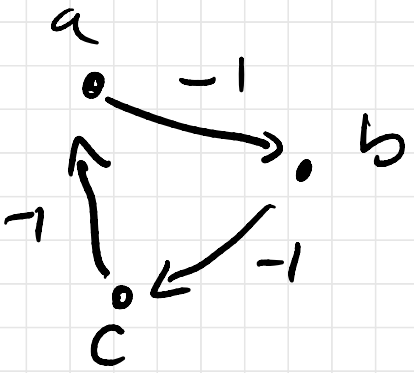
$x \leftarrow v_i$ "x" current vertex

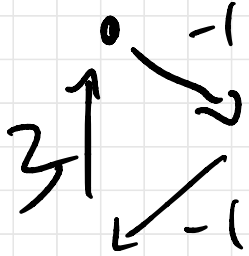
while $x \neq v_j$

  $x \leftarrow s_{x, v_j}$ → index of $v_j = j$

  $p = p + x$

output p





$$A^{n} = A^{n+1}$$

Dijkstras: $O(|E| \log |V|)$

$= O(m \log(n))$

$= O(n^2 \log(n))$

n times (for each vertex)

$O(n^3 \log(n))$