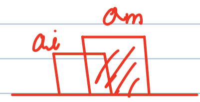
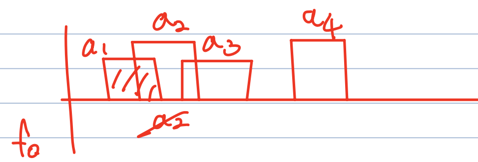
Recursive-AS(s, f, 0, 8)
Recursive-AS(f, f, m, 8)

Recursive-AS(s, f, i, n)

- i: 인덱스로 바꾸어 가면서
- n: # of activities

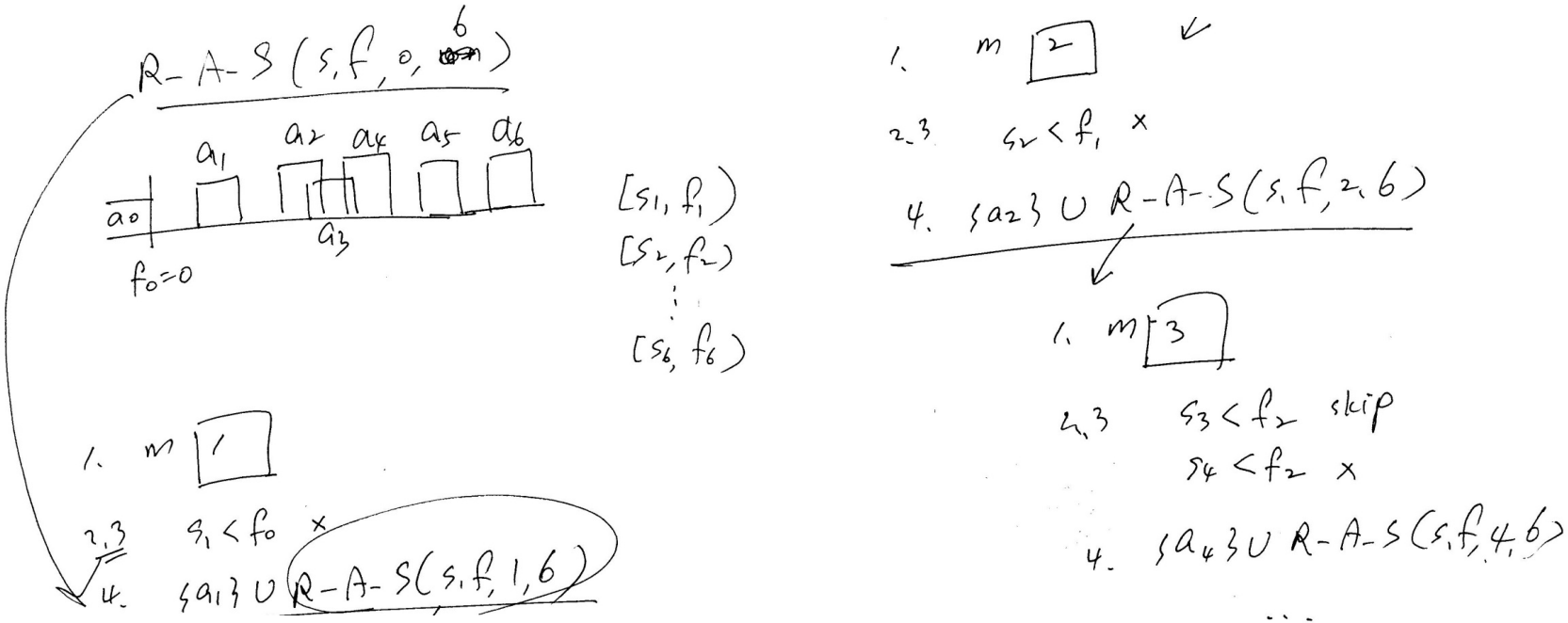
1. $m \leftarrow i+1$
2. while $m < n$ and $s_m < f_i$





RECURSIVE-ACTIVITY-SELECTOR(s, f, i, n)

```
1  m ← i + 1
2  while m ≤ n and s_m < f_i        ▷ Find the first activity in S_{i,n+1}.
3        do m ← m + 1
4  if m ≤ n
5     then return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6     else return ∅
```



Greedy versus dynamic programming

A: 0-1 knapsack problem

B: fractional knapsack problem

output

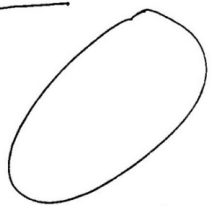
an optimal load

subject to the

weight constraint

$\leq W$

input



knapsack $\leq W$ pounds

W: an integer

items

(value, weight)

(v_1, w_1)

(v_2, w_2)

$\vdots$

(v_n, w_n)

A: binary choice (0/1)

B: fractional choice.

an instance

	$ value	pounds weight
item1	60	10
item2	100	20
item3	120	30

an optimal sol

A: item2, item3 total weight ≤ 50

total value ~~250~~ 220 $.

other possibilities

item1, item2 160 $ 30 pounds

item1, item3 180 $ 40 pounds

...

B: value/weight item1 - best
item2 - next
item3 - next.

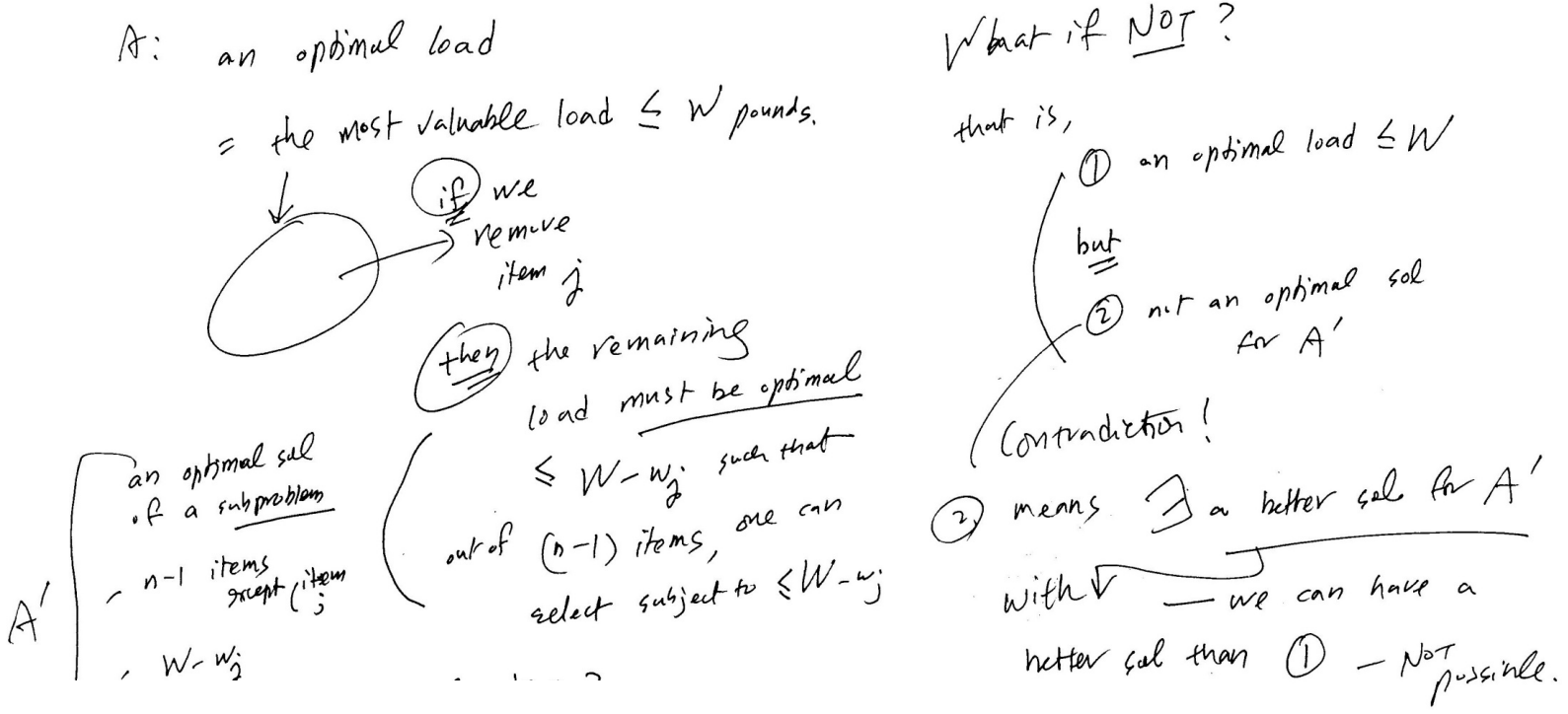
10 pounds - item1 60$

20 pounds - item2 100$ 30 pounds

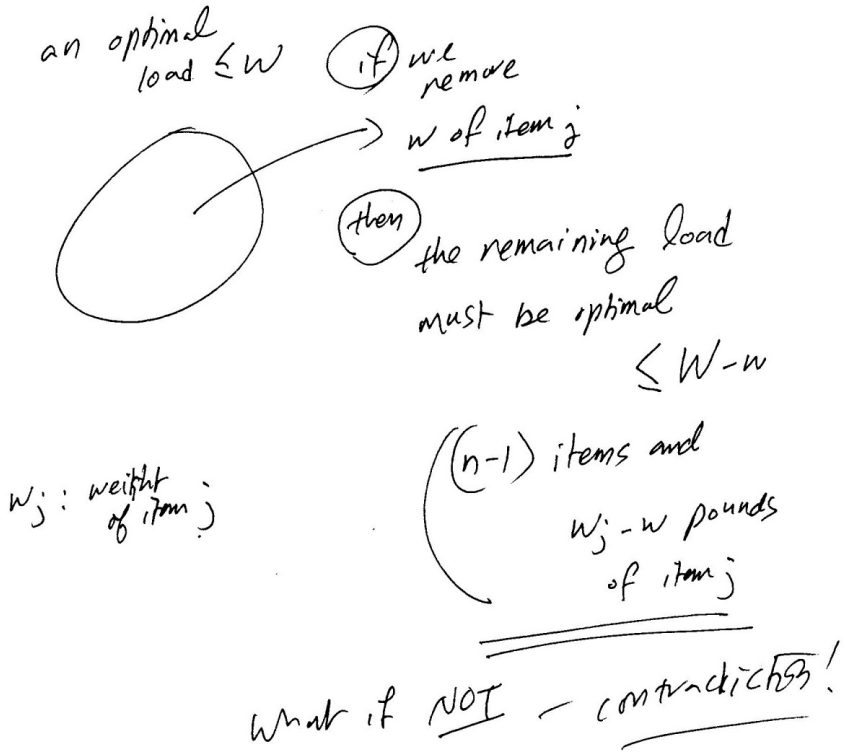
120 × 1/3 = 30 pounds item3 - 80$

240$

Optimal substructure property



Optimal substructure property



Greedy choice property
A: (no)
B: (yes) – criterion
value / weight

⇒A cannot be solvable by a greedy algorithm, but B can be solvable by a greedy algorithm

중간확인

16.4

a metroid

1930's whitney

↓ independence

- □

A metroid $M = (S, I)$

① $S \neq \emptyset$, finite set

② $I \subseteq 2^S$ such that

1) if _____ then _____

2) if _____ then _____

2^S power set

Ex $S = \{a, b\}$

$2^S = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

2.1) if $x \in I$ then all subsets of $x \in I$

Ex) $I = \{\ldots, \{a, b\}, \ldots\}$ (x)

$\{a\}$ $\emptyset$ $\{b\}$

2.2) if $x \in I$, $Y \in I$, $|x| > |Y|$

then $\exists a \in x - Y$ $\{a\} \cup Y \in I$

some element must exist in the set $x - Y$

Ex $x = \{1, 2, 3\}$

$Y = \{2, 4\}$

$x - Y = \{1, 3\}$

Ex) $I = [\ldots \underbrace{\{a, b\}}_{x}, \underbrace{\{c\}}_{Y}, \ldots]$

$x - Y = \{a\ \ b\}$

$Y + \{a\} = \{a, c\}$ also $\in I$

$Y + \{b\} = \{b, c\}$

Ex) $S = \{a, b, c, d, e\}$

$I_1 = \{\emptyset, \{a, b\}\}$ Not a subset

(C, S, I)

$I_2 = \{\phi, \{a,b\}, \{a\}, \{b\}\}$ (S, I_2)

$I_3 = \{\phi, \{a,b\}, \{a\}, \{b\}, \{c\}\}$ (S, I_3) (1) ✓ (2) ✗

$x = \{a,b\}$
$y = \{c\}$

$x - y = \{a, b\}$
↓
$\{a, c\}$
$\{b, c\}$

2.) the structure of a problem ⟷ a way to solve the problem

if $\mathcal{J}$ (problem) is a matroid then GA to solve $\mathcal{J}$

↑ the underlying structure ⇍ ASP GA로 풀 수 있다고 해서 metroid 문제인건 아니다.

↓ selection 16.6에 자세한 설명

1 unit time

16.5 Unit task scheduling problem

- n unit tasks
- deadlines $\leq n$ (positive integers)
- penalties : positive real #s

Assumption: $p_1 \geq p_2 \geq p_3 \cdots \geq p_n$

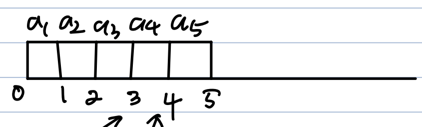
$n!$

minimize total penalties incurs by missing deadlines.

Ex

	d	p
a_1	2	100
a_2	3	60
a_3	1	50
a_4	2	50
a_5	4	20

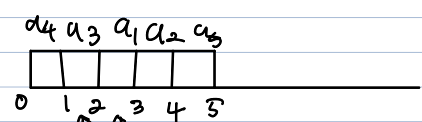
a_1, a_2, a_3, a_4, a_5



$50 + 50 + 20 = 120$

A brute force

$O(n!)$ → $O(n^2)$



$50 + 100 + 60 + 20 = 230$

총 5! (120) 가지 경우

$S = \{a_1, a_2, \ , \ , a_5\}$

A matroid

$M = (S, I)$

$S = \{\ \}$

$I = \{\ \}$

$S = \{a_1, a_2, \cdots a_n\}$

$I = \{\ \}$

$S = \{a_1, a_2, a_3 \ldots a_n\}$

(I) $x = \{\ \}$

a subset x of $S \in I$

iff

All tasks in x cause finished w/o penalties

$\{a_1, a_2, a_3\} \in I$

2

$\{a_3, a_1, a_2\} \in I$

$\{a_1, a_3, a_2\} \notin I$

deadline = 2 1 2

so it cannot be member of I

11/3

16.4 a matroid ⟶ 23.8 graphic matroid

16.5 unit test scheduling prob

solvable by greedy Algorithms

Ch 23

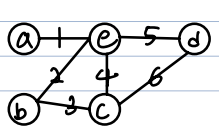
Graphic matroid

- an undirected matroid

$G = (V, E)$ $M_G = (S_G, I_G)$ ⟶ $S_G = E$

a subset of E called $x \in I_G$ iff x is acyclic

Ex $G = (V, E)$ $M_G = (S_G, I_G)$



$S_G = \{1, 2, 3, 4, 5, 6\}$ $2^{S_G} = \{\ \}$ (64)

$I_G = \{\{1,2\}, \{1\}, \emptyset, \{1,2,3\}, \{1,2,3,6\}\}$ {2,3} → 사이클

{1,3} {2} {3}
{1,4} :
:
{4}

{1,2,3,4}
2, 3, 4가 사람이여서

Unit task scheduling problem

Ex

	a_1	a_2	a_3	a_4	a_5
d	2	1	3	2	4
Ⓟ	50	40	30	30	10

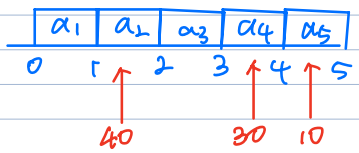
a brute way

$O(n!)$ → 모든경우 다체크

↓

$O(n^2)$

min penalty



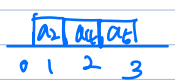
$M=(S,I)$

$S=\{a_1, a_2, \ldots a_n\}$

Ⓘ → a subset of $x \in I$ iff all tasks in x can be finished by deadline (possibility)

	a_1	a_2	a_3	a_4	a_5
d	2	1	3	2	4
Ⓟ	50	40	30	30	10

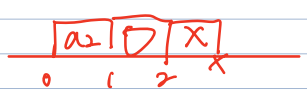
$\{a_2, a_4, a_5\} \in I$ (2, 4, 5) $x \in I$ 000 a_2, a_4, a_5 ↑ I



a_5 a_4 a_2

$\{a_1, a_2, a_4\} \notin I$

deadline 2 1 2



16.4

$M = (S, I)$

$S = \{\quad\}$ $\quad \exists = \emptyset$

$I = \{\quad\}$ member of I

(1) if $x \in I$, then $\in I$

(2) if $x \in I$, $y \in I$ $|x| < |y|$

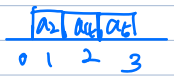
then $\exists a \in y - x$, $\{a\} \cup x \in I$

prove

(1) $\{a_2, a_4, a_5\} \in I$ (1, 2, 4)

$x \in I$

$a_2, a_4, a_5 \in I$



prove

(2) $x = \{a_2, a_3, a_4, a_5\}$ (1, 3, 2, 4)

$y = \{a_2, a_3\}$ (1, 3)

a_2, a_3 - 패드자리O

~~a_5, a_4~~ - 패드자리X

2 4

$x - y = \{a_4, a_5\}$

$\exists a \in x - y$, $\{a\} \cup y \in I$

$\{a_2, a_3, a_4\}$ (1, 3, 2)

↗ 패딩자리 없다!

$\{a_2, a_3, a_5\}$ (1, 3, 4)

패딩자리 없다!

A matroid and greedy algorithms – section 16.4

S: a finite set ($\neq \phi$)

I: a set of subsets of S

such that

① if $x \in I$ then all of x's subsets are members of I.

② if $x \in I$, $y \in I$, $|x| > |y|$ then $\exists a \in x - y$ such that $\{a\} \cup y \in I$.

(S, I) is a matroid.

$S = \{1, 2, 3\}$

$I_1 = \{\phi, \{1\}\}$

$I_2 = \{\{1\}, \{2\}\}$

$I_3 = \{\phi, \{1\}, \{2\}, \{3\}, \{1, 3\}\}$

(S, I_1) matroid

$(S, I_2) \times$ $(S, I_3) \times$

$\{1, 3\} - \{2\} = \{1, 3\}$

at least $\{2\} \cup \{1\}$

or

$\{2\} \cup \{2\}$

must be a membr of I_3.

If the underlying structure of an optimization problem is a matroid,
the problem is solvable by a greedy algorithm.

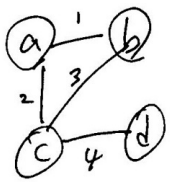
What about the other direction? That is, can we say that

if an optimization problem is solvable by a greedy algorithm thenthe underlying structure of the problem is a matroid

NOT always!

Def. Let $G = (V, E)$ be an undirected graph.

The graphic matroid $M_G = (S_G, \mathcal{I}_G)$ is defined as follows.



G

$S_G = E$

$\mathcal{I}_G$: a subset of E x is a member of $\mathcal{I}_G$ iff x is acyclic.

$M_G = (S_G, \mathcal{I}_G)$

$S_G = \{1, 2, 3, 4\}$

$\mathcal{I}_G = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,3,4\}, \{2,3,4\}\}$

$\{1,2,3\}$ a cycle.

unit task scheduling problem

(16.5 textbook)

input

$a_1, a_2, \cdots, a_n$ — unit tasks

deadlines $1 \le d_i \le n$

penalties a positive integer

assumption (sorted in a non-increasing order)

output

a schedule (= a permutation) of tasks so as to minimize the total penalties incurred by missing deadlines

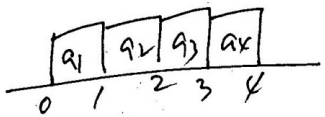
	a_1	a_2	a_3	a_4
d	2	1	3	2
p	7	6	3	3

4! schedules

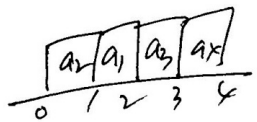
a_1, a_2, a_3, a_4

$(a_2, a_1, a_3, a_4$

...



$6 + 3 = 9$



3

The underlying structure of the unit task scheduling problem is a matroid

a matroid $M = (S, I)$

2 properties.

① Let S be $\{a_1, a_2, \cdots, a_n\}$

then define I as

a set of subsets of S

such that

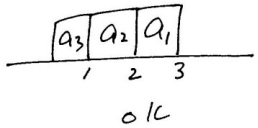
x consists of activities that can be schedulable without penalties.

	a_1	a_2	a_3	$\cdots$	a_n	
d	d_1	d_2	d_3	$\cdots$	d_n	$1 \le d_i \le$
p	p_1	p_2	p_3	$\cdots$	p_n	$p_1 \ge p_2 \ge \cdot$

1st property is satisfied

	a_1	a_2	a_3	a_4	a_5	a_6
d	4	3	1	2	4	3
p	70	60	50	50	40	30

$\{a_1, a_2, a_3\} \in I$



o/c

$\{a_2, a_3, a_4, a_6\} \notin I$

it is not possible to schedule all of a_2, a_3, a_4, a_6 without penalties.

$\therefore$ As long as x consists of activities schedulable without penalties, $x \in I$

What about the 2nd property (to be a matroid)?

let $N_t(A)$ be the # of tasks in A whose deadlines $\leq t$ ($N_0(A) = 0$)

Ex.

	a_1	a_2	a_3	a_4	a_5
d	2	1	3	4	2
p	50	40	30	30	10

$A = \{a_1, a_2, a_3\}$ $N_2(A) = 2$ a_1, a_2

$N_3(A) = 3$ a_1, a_2, a_3

$A = \{a_2, a_4, a_5\}$

$N_1(A) = 1$ a_2

$N_3(A) = 2$ a_2, a_5

$N_4(A) = 3$ a_2, a_4, a_5

$t = 0, 1, \ldots n$

$N_t(A)$ — A: a set of tasks

↳ # of tasks in A whose deadlines $\le t$

	a_1	a_2	a_3	a_4	a_5
d	2	1	3	2	4
(P)	50	40	30	30	10

$A = \{a_2, a_3, a_5\}$ (1, 3, 4)

$N_0(A) = 0$

$N_1(A) = 1$

$N_2(A) = 1$

$N_3(A) = 2$

$N_4(A) = 3$

$N_5(A) = 3$

Consider any two members $x, y \in I$, $|x| > |y|$

Create a table

t	$N_t(x)$	$N_t(y)$
0	0	0
1	1	1
2	2	2
→ 3	3	2
→ 4	4	3
→ 5	4	3

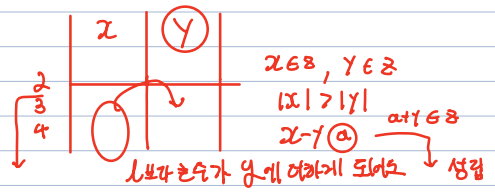
$x = \{a_1, a_2, a_3, a_5\}$ (2, 1, 3, 4)

$y = \{a_2, a_4, a_5\}$ (1, 2, 4)

select the largest value t such that $N_t(x) \le N_t(y)$

t

$t = 2$



2nd property

Unit task scheduling problem

obs

1) # of tasks max

2) without penalty

input :			
	$a_1, a_2, \ldots a_n$	→ selection of tasks	
d	$d_1, d_2, \ldots d_n$		$1 \le d_i \le n$
P	$P_1, P_2, \cdots P_n$		$P_1 \ge P_2 \ge P_n$

(1) Every task can be finished without penalty

$I = \{$ Entire set member $\}$

Ex	a_1	a_2	a_3	a_4	a_5
d	1	2	3	2	2
p	50	40	40	10	1

(2)

1) # of tasks max = 3 tie
2) without penalty
select the largest number of penalty

$\{a_1, a_2, a_3\}$ 130 ✓
$\{a_1, a_4, a_3\}$ 100
$\{a_1, a_5, a_3\}$ 91
$\{a_4, a_5, a_3\}$ 51
$\{a_2, a_3, a_4\}$ 90

5 ways

Sudo code

$x \leftarrow \emptyset$ a greedy selection

```
for i ← 1 to n do
  | if x ∪ {a_i} ∈ I
  |   then X ← X ∪ {a_i}
return X
```

rec AS(S, i, f, n)

Sm < fi

I φ ⇉ 0 0 0 0

Example

a_1	a_2	a_3	a_4	a_5	a_6	a_7
4	2	4	3	1	4	6
70	60	50	40	30	30	10

$x = \{ \}$
$X = \{a_1, a_2, a_3, a_4, a_7\}$

a_5: $a_1 \sim a_4$까지 4가지중 가장 큰 deadline이 4인데 a_5가 들어오므로 가장 큰 deadline이 5보다 작아서 X
a_6: 같은이유

$I = \{a_1, a_2, a_3, a_4, a_7\}$

$\emptyset, \{a_1\}$

Runtime
$O(n^2)$

I = { O }
I = { O, O }
I = O
I = O O O O

1 + 2 + 3 + 4 + … + n } $O(n^2)$

Consider 2 members of I x, y such that $|x| > |y|$

→ We show that there MUST exist an element a in $x - y$ such that $\{a\} \cup y \in I$.

Construct a table

t	$N_t(x)$	$N_t(y)$
0	0	0

Observation

let k be the largest t such that

$$N_t(x) \leq N_t(y)$$

- x must contain more tasks with deadline $k+1$ than y

Construct a table

t	$N_t(x)$	$N_t(y)$
0	0	0

Let a be the task $\in x - y$ with deadline $k+1$

→ $\{a\} \cup y \in I$ why?

① y is a member of I
→ schedulable without penalties

② a deadline $k+1$

③ maximum value of deadlines in $y = k$

∴ $\{a\} \cup y$ schedulable without penalties

How can we exploit the structural property to solve the problem?
- a “greedy” algorithm

$X \leftarrow \emptyset$

for $i \leftarrow 1$ to n do

  if $X \cup \{a_i\} \in I$

    then $X \leftarrow X \cup \{a_i\}$

return X

$O(n^2)$ algorithm

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
d	4	2	4	3	1	4	6
p	70	60	50	40	30	20	10

Fig 16.7 textbook

$X = \emptyset$

$X \cup \{a_1\} \in I$ $\therefore X = \{a_1\}$

$X \cup \{a_2\} \in I$ $\therefore X = \{a_1, a_2\}$

$X \cup \{a_3\} \in I$ $\therefore X = \{a_1, a_2, a_3\}$

$X \cup \{a_4\} \in I$ $\therefore X = \{a_1, a_2, a_3, a_4\}$

$X \cup \{a_5\} \notin I$

$X \cup \{a_6\} \notin I$

$X \cup \{a_7\} \in I$ $\therefore X = \{a_1, a_2, a_3, a_4, a_7\}$

total penalty = 50

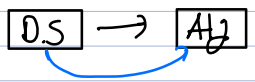
Ch21 data structure disjoint sets

Ch6 Max-heap (HS) A={a,b,c} B={1,2,3} C={ }) n

11/7/22

special data structure for disjoint set
↳ union find data structure.

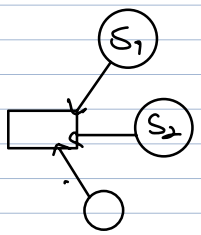
QS → data structure X
InS → X
MS → X
HS → O use data structure. → depending on Algorithm we use may or may not need data structure.

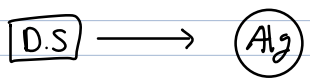


$S_1 = \{a, b, c\}$

$S_2 = \{1, 2, 3\}$

$S_3 = \{x, y\}$





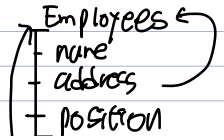
① operation

② operation

stack FILO
- push
- pop

queue FIFO
- Enque
- dequeue

Data type but Abstract



int x, y
x+y
x-y
:

ops - operation을 할 수 있다. (similar to ds)

special data structure for disjoint set

- find-set(b) = a
- union(a, 1) = {_____ 3
- create(x) = {x}

make-set

$S_1 = \{a, b, c\}$

$S_2 = \{1, 2, 3\}$

$S_3 = \{x, Y\}$

1. connected component
   ch23
2. Kruskal's Al
   MST

3 operations.

time-complexity of this data structure is not determined by single operation.

Use two parameter m, n.

of 100

m : total number of these 3 operations together

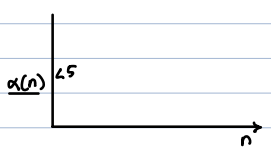
n : the number of make-set operations

of 20

① good example of linked list

② $O(m + n\lg n)$

$O(m\alpha(n))$ $\alpha(n)$: a function of n (Even this is a function it is smaller than 5, so this function works almost as the same as a constant.



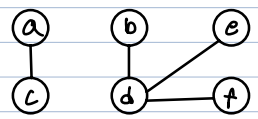
- find-set(b) = a
- union(a, l) = {_____}
- create(x) = {x}
  make-set

these 3 operations are interdependent
ex) union( ) 를 사용하려면 create( ) 를 사용해야함.

A connect component = { }

- an undirected graph
  (of a directed G_1, SCC)

Ex
input:



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

(input: Undirected graph $G = (V, E)$

Output: connected components

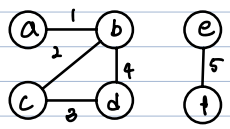
we should use a DS for disjoint sets
- make-set()
- find-set()
- union( )

G ↺
OOOO

Stack | Queue

Ch 23 MSTP — Prim's Alg, Kruskal's Alg.] concurrently using the same data structure

Ex)



{a} {b} {c} {d} {e} {f}
{a, b}
5 edges {1, 2, 3, 4, 5}
edge 1이 존재하면 (a, b) 가 존재해야함
find-set(a)] → union(a, b)
find-set(b)]
after these two executed combine (a, b)

at the end { } , { }

11/8/24

Ch 21 a data structure for disjoint sets

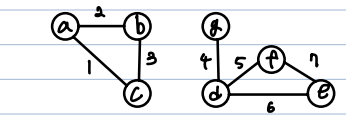
m, n

① make set
② union
③ find-set

n
m

- $O(m + n \lg n)$
- $O(m\,\alpha(n))$

Find connected components (CT)



① make set
② union
③ find-set

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

1. {a}, {b}, {c}, {d}, {e}, {f}, {g} → Similar to kruskal's Alg. (ch23)

2. for each edge (in an arbitrary order)

(x, y) ③ find-set

find-set(x) = find-set(y)

returns same?

if find-set(x) ≠ find-set(y) → use union(x, y)

let's check

for edge 1. {a, c}, {b}, {d}, {e}, {f}, {g}

edge 2. {a, b, c}, {d}, {e}, {f}, {g}

for edge 3. find-set(b) ≠ find-set(c), so keep the same set

edge 4 {a, b, c}, {d, g}, {e}, {f}

edge 5 {a, b, c}, {d, g, f}, {e}

edge 6 {a, b, c}, {d, g, f, e}

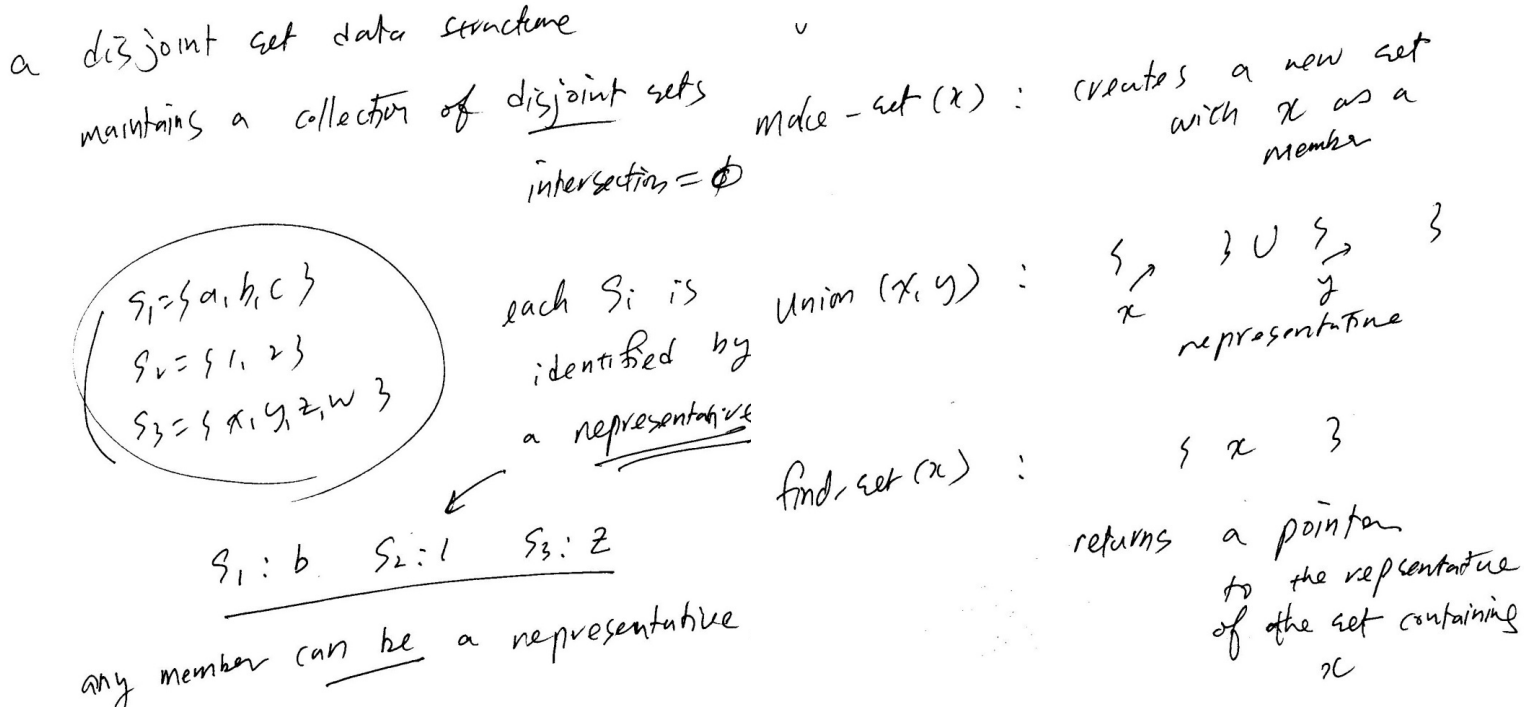
edge 7 "

$O(m \cdot \alpha(n))$

$\uparrow\downarrow$

$O(V+E)$

Ch 21. data structures for disjoint sets



Time complexity
make-set(x), union(x,y), find-set(x)

time complexity
- always analyzed in terms of
2 parameters (n = make-set operations
m = make-set, union, find-set } operations together

$m \geq n$

theorem 21.1
$O(m + n \lg n)$

theorem 21.13
$O(m \alpha(n))$

$\alpha(n)$ - a function that grows very slowly.

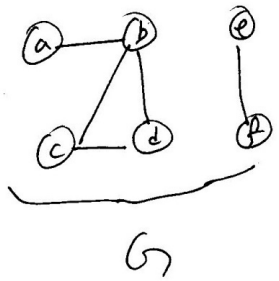
for $n \leq 10^{80}$, $\alpha(n) \leq 5$

connected component

– a subgraph G' of an undirected graph G in which for each pair of nodes in G' $\exists$ a path

In general, nodes of G'

Ex.



G

2 connected components:

$\{a, b, c, d\}$

$\{e, f\}$

Can we determine connected components of an undirected graph using disjoint set data structure?

YES!

```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ V[G]
2       do MAKE-SET(v)
3  for each edge (u, v) ∈ E[G]
4       do if FIND-SET(u) ≠ FIND-SET(v)
5            then UNION(u, v)

SAME-COMPONENT(u, v)
1  if FIND-SET(u) = FIND-SET(v)
2     then return TRUE
3     else return FALSE
```

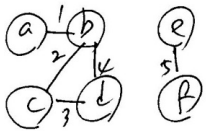
Time complexity for CC(G)

number of MAKE-SET operations
= number of nodes =V

number of FIND-SET operations and
UNION operations
≤ number of edges = E

Using the 2nd implementation
O((V+E) α (V))

```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ V[G]
2       do MAKE-SET(v)
3  for each edge (u, v) ∈ E[G]
4       do if FIND-SET(u) ≠ FIND-SET(v)
5            then UNION(u, v)
```



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

edges 1, 2, 3, 4, 5

a — b

b — c

c — d

b — d

e — f

find-set(a) ≠ find-set(b)

{a} {b}

union(a, b)

{a, b}

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

find-set(b) ≠ find-set(c)

{a, b, c}

v

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

find-set(c) find-set(d)

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

find-set(b) = find-set(d)

find-set(e) find-set(f)

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

Ch 22

Understand Graph

① representation of graphs < linked list (Adjacency lists) / matrix (adj)

take graph as an input

Alg (G) prog (G)

"Graph search" Alg.

② Breadth first search Alg (BFS) → Queue

③ Depth first search Alg (DFS) → Stack

without stack (textbook) ↳ a compiler

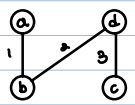
A systematic way to visit each and every node in a graph

Representation

"adjacency" — 2 nodes

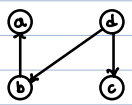
"Incidence" — undirected (a node, an-edge)

Ex



a, b - adjacent

d, b - "



a is adjacent to b

b is NOT adjacent to a (because of direction)

b is adjacent to d

d is Not adjacent to b (")

"Incidence"

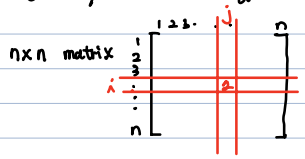
edge 1 is incident on a b

"touches"

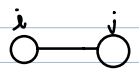
① adjacency matrix < both / d / a

n nodes

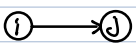
0's, 1's

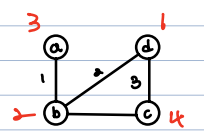


it when undirected graph G

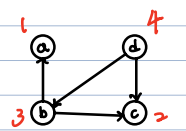


when directed graph G





_ d b a c _

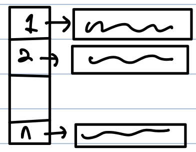


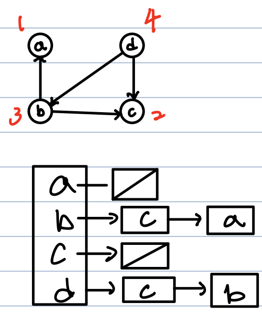
a c b d _

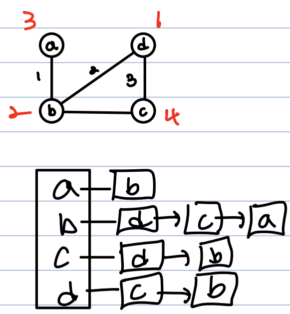
d	0	1	0	1
b	1	0	1	1
a	0	1	0	0
c	1	1	0	0

a	0	0	0	0
c	0	0	0	0
b	1	1	0	0
d	0	1	1	0

② adjacency lists





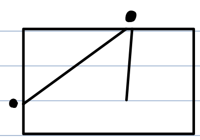


① adjacency matrix → 0을 많이 낭비하기 때문에

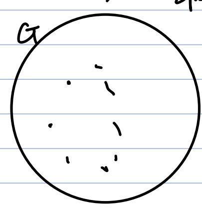
② adjacency lists ← better memory usage.

"sparse"

Program



G



a sparse graph

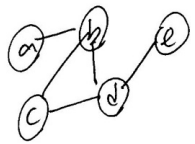
Ch 22 Elementary graph algorithms

22.1 representations of graphs

① adjacency matrix
② adjacency lists

both directed & undirected graphs

Ex. undirected graph



5×5 matrix (5 nodes)

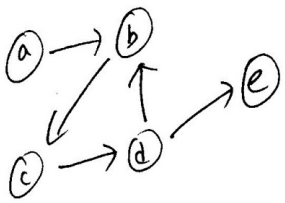
	a	b	c	d	e
a		1			
b	1		1	1	
c		1			1
d		1			1
e			1	1	

binary 0/1 value.

symmetric

all others : 0

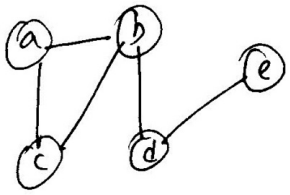
directed graph

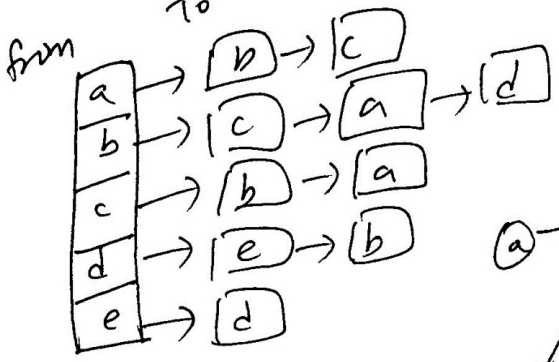


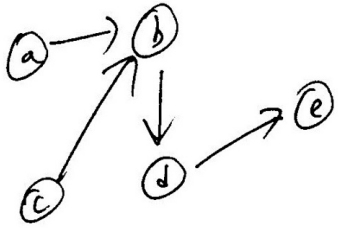
from \ to	a	b	c	d	e
a		1			
b			1		
c				1	
d		1			1
e					

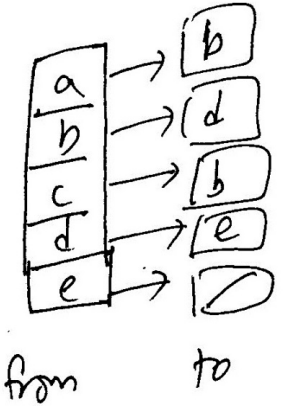
all others 0.

adjacency lists



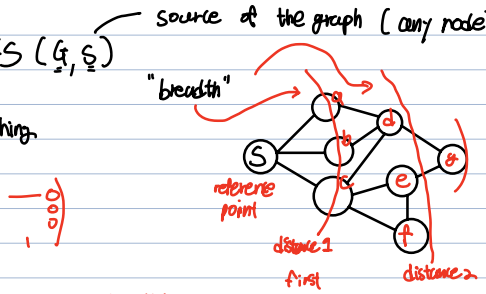






22. BFS (G, s) ← source of the graph (any node)

Graph searching

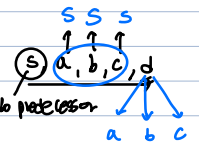


같은 distance 이면 | b, a, c
같은 breadth 안에서 | c, a, b
있는 순으의 순서는 | a, b, c
상관없다

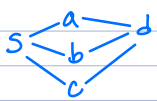
for each node x

① color[x] : white, grey, black

② π(x)
the predecessor
"from drawing above"



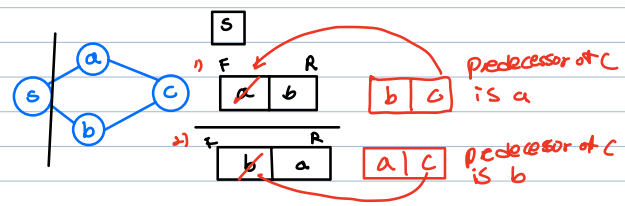
In order to identify predecessor from all possible 3, we need to recognize Algorithm



1) F R
s | a b c
a dequeued
b c d ← d was enqueued because a is dequeued.
a

2) s
b c a
c a d
b
d
"
b

Pg 24



2 cases

V[G] - S_W → G_T → B

For each node x ① c[x]
② π(x)
③ d(x)

1
9 pop aneda

10
18

11/10 OS

a queue

BFS (breadth first) : graph searching

visited / Not

1
9 "prop"

10 main logic

18 lines

for each node x

- $C[x]$ c: color : ① while → grey → black
- $\pi[x]$:
- $d(x)$: distance

Adj M Adj lists

BFS (G, S)

1
9 "prop" ← (a) ← (s)

10 while ⓠ ≠ φ

18 lines

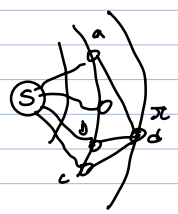
Prim's Alg Ch29

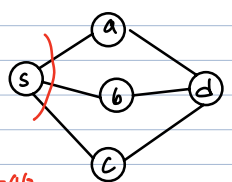
both use queue OS

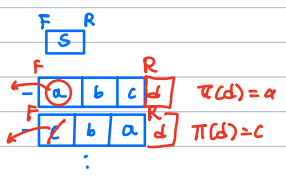
Difference

BFS
- Standard que

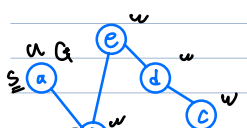
Prim's Alg
- R 6.5







Adj (x)

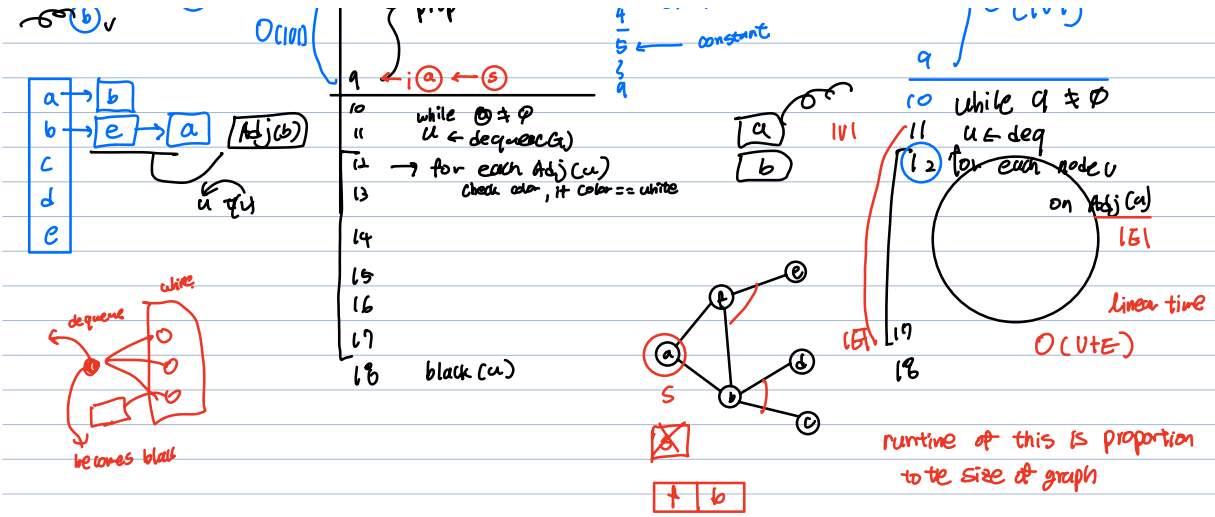


BFS (G, S)

1 2 "prop"

: O(|V|)

1) O(|V|)



- $\pi(x)$
- $c(x)$ ← Essential
- $d(x)$

why do we need ?
- helpful for other Alg

O(V+E)

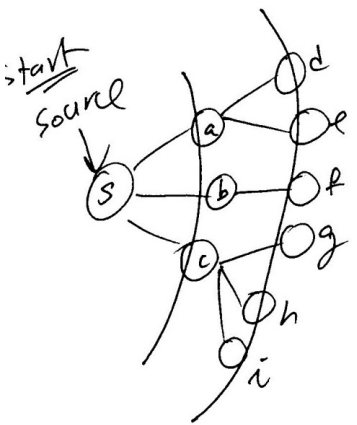
Input : G ← an undirected graph — adj lists

Output :
if G is connected, yes otherwise NO

for each node d(e)

22.2 breadth first search

both directed / undirected graphs.



a, b, c equidistant from s

at the same breadth

d, e, f, g, h, i are at the same breadth

all of {a, b, c} must be visited BEFORE any one of {d, e, f, s, h, i} is visited

"Breadth-first"

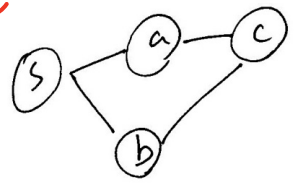
```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2       do color[u] ← WHITE
 3          d[u] ← ∞
 4          π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       do u ← DEQUEUE(Q)
12          for each v ∈ Adj[u]
13               do if color[v] = WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u] + 1
16                         π[v] ← u
17                         ENQUEUE(Q, v)
18          color[u] ← BLACK
```

main logic

first location of queue

(s) source node

for each node, x,
color(x) : white, gray, black
d(x) : from the source node
π(x) : the predecessor

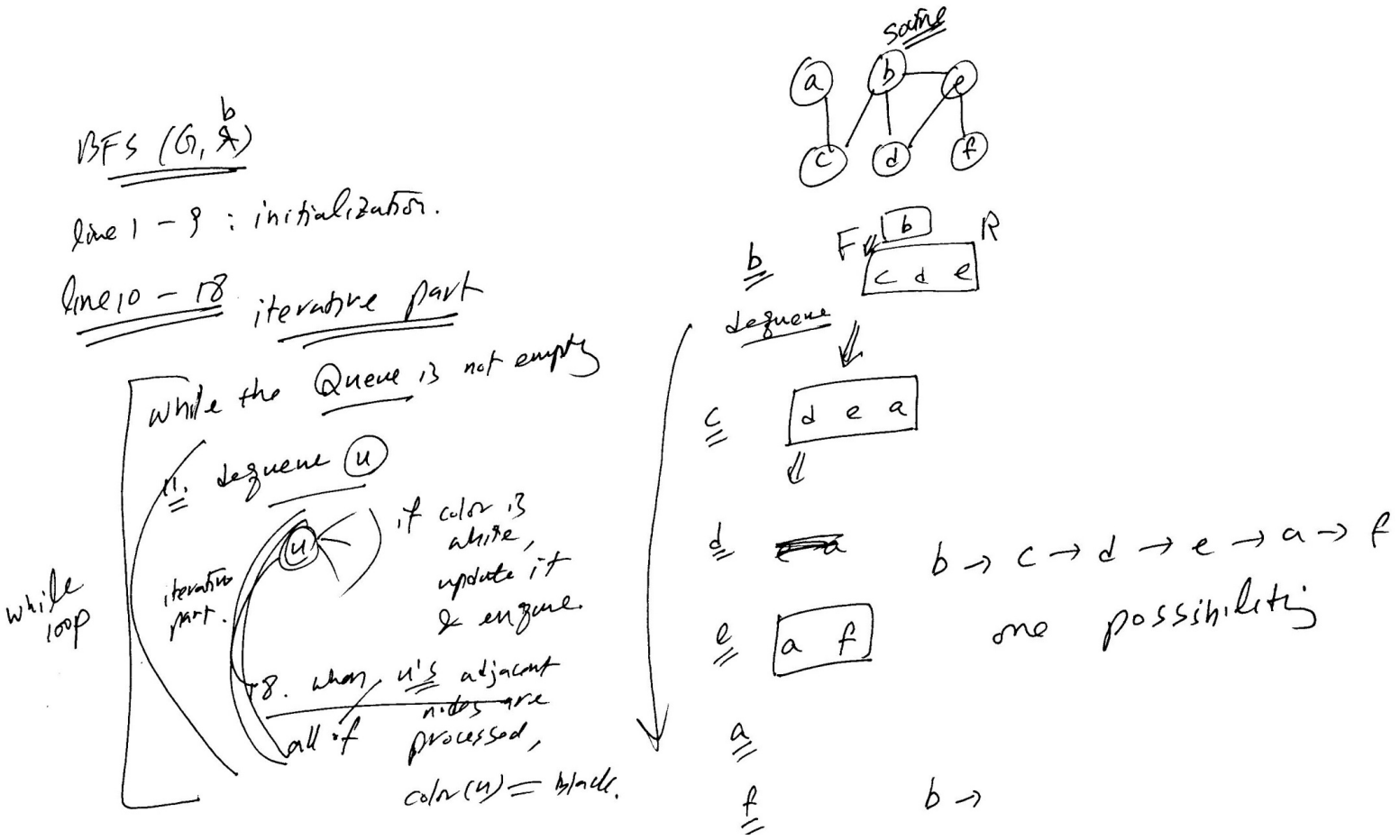


s, a, b, c
a possibility

s, b, a, c
a different possibility

π(c) = b

π(c) = a


BFS (G, s)
line 1 – 9 : initialization.
line 10 – 18 iterative part
while the Queue is not empty
while loop
1. dequeue (u)
iterative part.
if color is white, update it & enqueue.
8. when u's adjacent nodes are all of processed, color(u) = black.
same
a b c d e f
b
F b R
c d e
dequeue
c
d e a
d
e
a f
a
f
b → c → d → e → a → f
one possibility
b →

```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2       do color[u] ← WHITE
 3          d[u] ← ∞
 4          π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       do u ← DEQUEUE(Q)
12          for each v ∈ Adj[u]
13               do if color[v] = WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u] + 1
16                         π[v] ← u
17                         ENQUEUE(Q, v)
18          color[u] ← BLACK
```

Time complexity

line 1 ~ line 4 : |V|
line5,6,7,8,9 : constant time

line 10 ~ 18 : at most |V|

line 12 ~ 17 : at most |E|

O(V + E)

linear in the size of the input graph!

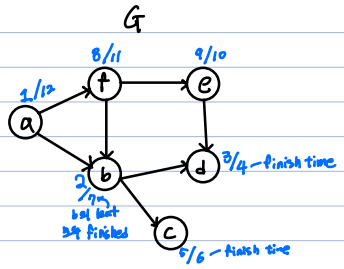
22.3 Depth first search (G) < Adj matrix / Adj lists

- Both undirected, directed graph, we can use DFS

Ch 21

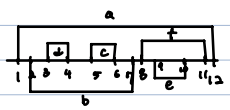
$\in$ Connected components

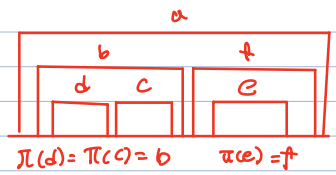
Ex ϕ



For each node x
- $c[x]$
- $\pi(x)$
- $d(x)$ (discovery time) ~~discover~~ discovered for the first time
- $f(x)$ (finish time)

We can "linearize"





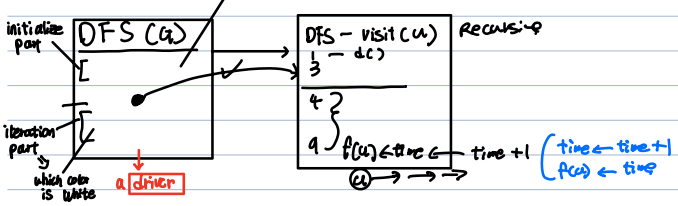
$\pi(x)$

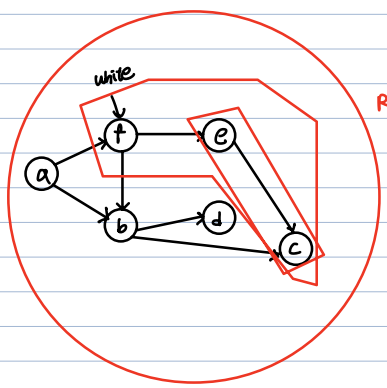
$\pi(b) = \pi(t) = a$

$\pi(d) = \pi(c) = b$ $\pi(e) = t$

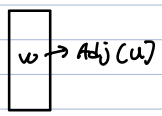
$\pi(x)$ — x가 갈 수 있게 되는 이웃 노드

maintain variable time ← declare global to be used in DFS-visit function

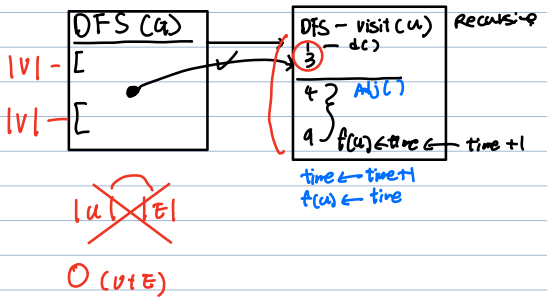




Recursively

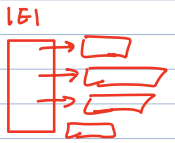


Runtime


DFS(G)
|V| -
|V| -
DFS-visit(u)
Recursive
d(u)
3
4
Adj(u)
9
f(u) ← time
time + 1
time ← time+1
d(u) ← time

|u|
|E|

O(V+E)

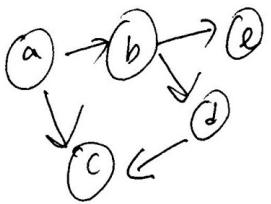

|E|

22.3 depth first search

discover
finish
time stamp

depth - first

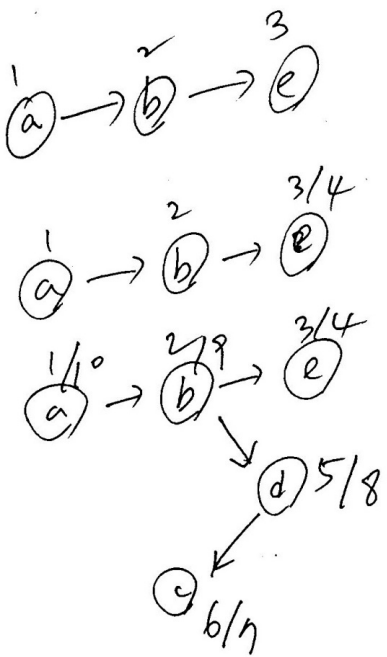
both directed / undirect



a, b, e, d

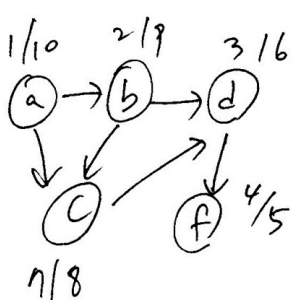
a, b, d,

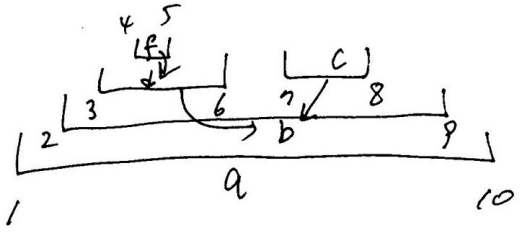
a, c, b,



```
DFS(G)
1  for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3          π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V[G]
6       do if color[u] = WHITE
7             then DFS-VISIT(u)
```

```
DFS-VISIT(u)
1  color[u] ← GRAY        ▷ White vertex u has just been discovered.
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]    ▷ Explore edge (u, v).
5       do if color[v] = WHITE
6             then π[v] ← u
7                  DFS-VISIT(v)
8  color[u] ← BLACK       ▷ Blacken u; it is finished.
9  f[u] ← time ← time + 1
```





time is incremented by 1, AND f[u] is updated with “time”

$\pi(c) = b$ $\pi(b) = a$

$\pi(f) = d$ $\pi(d) = b$

```
DFS(G)
1  for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3          π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V[G]
6       do if color[u] = WHITE
7             then DFS-VISIT(u)
```

```
DFS-VISIT(u)
1  color[u] ← GRAY        ▷ White vertex u has just been discovered.
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]    ▷ Explore edge (u, v).
5       do if color[v] = WHITE
6             then π[v] ← u
7                  DFS-VISIT(v)
8  color[u] ← BLACK       ▷ Blacken u; it is finished.
9  f[u] ← time ← time + 1
```

time complexity

O(|V| + |E|)

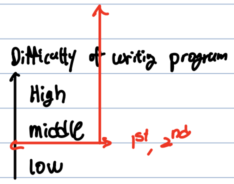
DFS(G)
line1 ~ line3 O(|V|)
line 5 iterations at most O(|V|)

linear in the size of the input graph

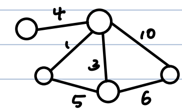
DFS-VISIT(u)
line4 at most O(|E|)

Ch23 min spanning tree Problem

- Prim's ⟵ P.Q 6.5
- kruskal's disjoint set Ch21



$|V| = n$

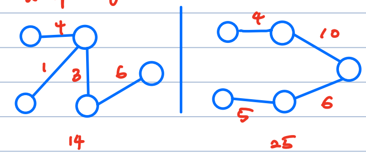


Input: a connected weighted ($\in R^+$) undirected graph $G = (V, E)$

Output: a tree that consists of all nodes (n) in V and edges in E (n-1)

subjected to
sum of weights min

a spanning tree



11/14 목

Ch23 min spanning tree prob ⟶ ① an optimization problem ⟵ (ch 15, 16 ASLP (minimize), MCM (max), ASP (minimize), OTSP (max), 0/1 fract)

min → undirected graphs only

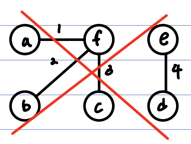
— ② decision prob
Y/N
T/F

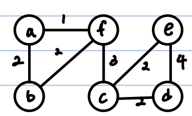
— ③ counting problem
input: $\boxed{G}$, a, b (two nodes)
output: # of paths from a to b

Input: an 1) undirected, 2) weighted, 3) connected graph $G = (V, G)$

1) undirected: no direction
2) weighted: each node is assigned a pos real #
3) connected: for each different nodes a, b, ∃ a path between a, b

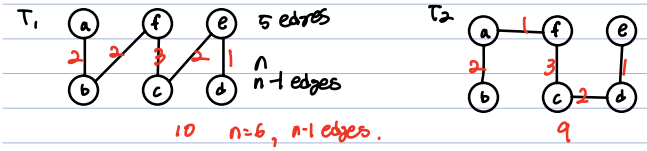
Ex





Output: a spanning Tree T ⟵ every node, edges from G

such that the sum of all edge weights is min



This problem is solvable by Greedy Algorithm

1. Kruskal's Alg → disjoint set DS
2. Prim's Alg → Priority Queue

common dinominator

Generic - MST (G, w)

graph, weights

Greedy Algorithm & DS

Ch 23 minimum spanning tree

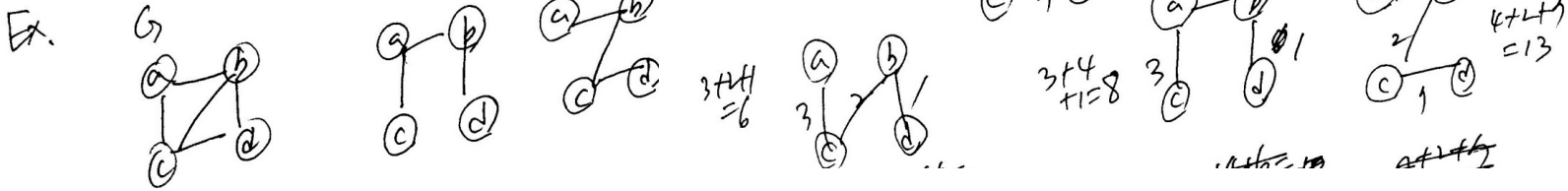
a spanning tree (G) — connected, an undirected graph

- a subgraph of G such that all nodes of G are included and it forms a tree

a minimum spanning tree of G

weights: positive real #s.

Ex.



```
GENERIC-MST(G, w)
1  A ← ∅
2  while A does not form a spanning tree
3      do find an edge (u, v) that is safe for A
4      A ← A ∪ {(u, v)}
5  return A
```

A starts with an empty set, each iteration Add nodes and edges.

Greedy Way

n-1 edges

"safe" – addition of an edge (u,v) does not cause a problem and "minimum" is still maintained

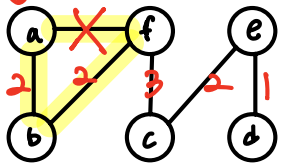
$X \leftarrow \emptyset$

whether there exists a cycle or not.

- not a cycle
- min weight

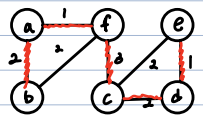
```
for i ← 1 to n do
    if X ∪ {ai} ∈ I
        then X ← X ∪ {ai}
return X
```

$O(n^2)$ algorithm



unit task scheduling problem

Ex



$A = \{$ $\}$

n nodes

0) $A = \{(e,d)\}$

+) $A = \{(a,f), (e,d)\}$

다음으로 가장 작은 변들부터 이기 때문에

*) $A = \{(a,f), (e,d), (c,d)\}$

**) $A = \{(a,b),$ $\}$ (c,e) 는 이제 cycle 을 만드므로 선택 불가

*) $A = \{(f,c),$ $\}$

implemented

M=(S,T)

I={X, ··· }

X={ __ }

16.5

utsp

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v ∈ V[G]
3       do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6       do if FIND-SET(u) ≠ FIND-SET(v)
7            then A ← A ∪ {(u, v)}
8                 UNION(u, v)
9  return A
```

disjoint set data structure

priority queue

```
MST-PRIM(G, w, r)
1  for each u ∈ V[G]
2       do key[u] ← ∞
3          π[u] ← NIL
4  key[r] ← 0
5  Q ← V[G]
6  while Q ≠ ∅
7       do u ← EXTRACT-MIN(Q)
8          for each v ∈ Adj[u]
9               do if v ∈ Q and w(u, v) < key[v]
10                   then π[v] ← u
11                        key[v] ← w(u, v)
```

priority queue (special queue)

Kruskal's Alg

- make - set
- find - set
- union

Prim's

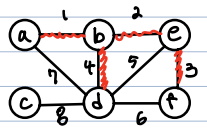
① a sorted seq

② A ← A ∪ { }

Connect components 하기 위해 for each edge

o o o o o — Any arbitrary order

MST - Kruskal (G, W)



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

order

1. A = { }
v
3.
5
8

2. (a,b) — A = {(a,b)}
(a,b) ← (b,e) — A = {(a,b), (b,e)}
(e,f) — A = {(a,b), (b,e), (e,f)}
(b,d) — A = {(a,b), (b,e), (e,f), (b,d)}
X (e,d) — e, d 모두 ↑ 기준 A set에 있다 → Skip
X (d,f)
X (a,d)
(c,d) — A = {(c,d), }

11/15/화

Ch23 (min) spanning tree problem

|V| = n

G = (V, E)

① Kruskal's Alg → O(E lg V)
② Prim's Alg → O(E lg V)

Generic MST (G, w)

A ← ∅

select a safe edge for A

Return A

① an edge with min weight at that moment
② A = { } acyclic (does not form a cycle)

n-1 edges
↓
tree 를 만들어야 하기 때문에

G = (V, E)

Kruskal (G, w)

① Kruskal's Alg → make-set, find-set, union (m; m, n)

$O(V)$

sort edges in non-decreasing order → $O(E \lg E)$

[①
② $O((V+E)\alpha(V))$

ch21 CCS

① an edge with min weight at that moment

② $A = \{\ \}$ acyclic (does not form a cycle)

for each edge in order (u,v)

if find-set(u) ≠ find-set(v) $|E|$

then $A \leftarrow A \cup \{(u,v)\}$

return A

time complexity

① $O(m + n \lg n)$

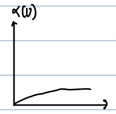
② $O(m\alpha(n))$ $O(V+E)$

① $O(E \lg E)$

② $O((V+E)\alpha(V))$ → $O(E \lg V)$

$\alpha(V) = O(\lg V)$

≤ 5



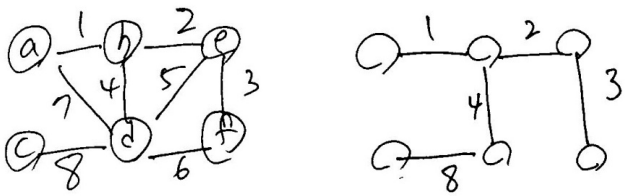
Kruskal's Alg (G, w)

① undirected

② connected → $|E| \geq |V| - 1$

③ weighted (R^+)

$|V| \times |V| \geq |E| \geq |V| - 1$



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

(a,b)
(b,e)
(e,f)
(b,d)
(d,e)
(d,f)
(a,d)
(c,d)

Line 5.

if find-set(a) ≠ find-set(b)

{a,b}

{a,b,e}

{a,b,e,f}

{a,b,e,f,d}

find-set(d) = find-set(f)
cycle !!

find-set(a) = find-set(d)
cycle!!

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

MST-KRUSKAL(G, w)

1 $A \leftarrow \emptyset$
2 **for** each vertex $v \in V[G]$
3 **do** MAKE-SET(v)
4 sort the edges of E into nondecreasing ord
5 **for** each edge $(u, v) \in E$, taken in nondecr
6 **do if** FIND-SET(u) $\neq$ FIND-SET(v)
7 **then** $A \leftarrow A \cup \{(u, v)\}$
8 UNION(u, v)
9 **return** A

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v ∈ V[G]
3       do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6       do if FIND-SET(u) ≠ FIND-SET(v)
7            then A ← A ∪ {(u, v)}
8                 UNION(u, v)
9  return A
```

time complexity

line 1,2,3 – initialization (line 3 : |V| times MAKE-SET operation)

line 4 – O(|E| * log |E|)

line 5,6,7,8 O(|E|) FINDE-SET, UNION operations

O((V + E) * α(V))

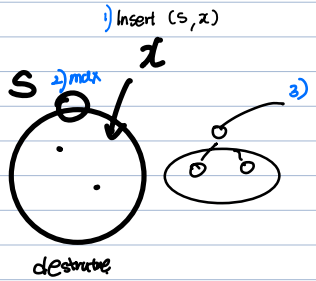
|E| ≥ |V| - 1 because G is a connected graph $\alpha(V) = O(\log V) = O(\log E)$

Therefore O(E log E), but |E| < |V| * |V| and <u>O(E log V)</u>

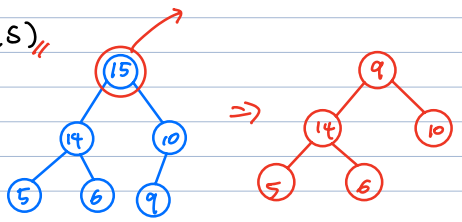
6.5 Priority Queue

↳ A data structure Ch 6

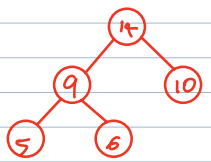
O(lgn)
① Insert (S, x) min PQ min heap
② Maximum (S) – min(S)
③ Extract-max (S) – E-min(S)
④ Increase-key (S, x, k) – Decrease key



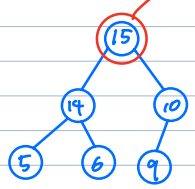
① H-Extract-max (S)



O(h) O(lgn)
max-heapify (S, 1)



Max(S) Return 15



A max

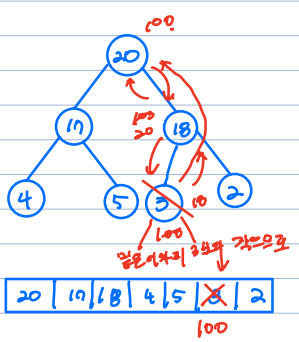
15	14	10	5	6	9

destructive

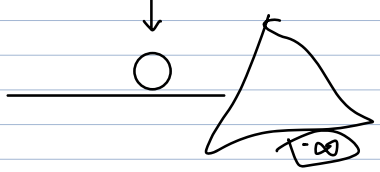
② H-increase-key (S, x, k)

index number

6 100



③ Max-heap-insert (S, x)



A ***priority queue*** is a data structure for maintaining a set S of elements, each with an associated value called a ***key***. A ***max-priority queue*** supports the following operations.

INSERT(S, x) inserts the element x into the set S. This operation could be written as $S \leftarrow S \cup \{x\}$.

MAXIMUM(S) returns the element of S with the largest key.

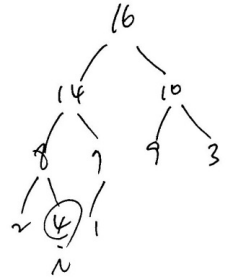
EXTRACT-MAX(S) removes and returns the element of S with the largest key.

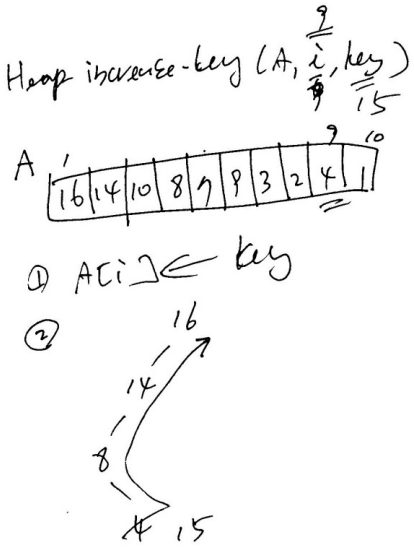
INCREASE-KEY(S, x, k) increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value.

Data structure vs abstract data type

Data type – int, float, char, etc

```
HEAP-EXTRACT-MAX(A)
1  if heap-size[A] < 1
2     then error "heap underflow"
3  max ← A[1]
4  A[1] ← A[heap-size[A]]
5  heap-size[A] ← heap-size[A] − 1
6  MAX-HEAPIFY(A, 1)
7  return max
```






```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2     then error "new key is smaller than current key"
3  A[i] ← key
4  while i > 1 and A[PARENT(i)] < A[i]
5        do exchange A[i] ↔ A[PARENT(i)]
6           i ← PARENT(i)
```

```
MAX-HEAP-INSERT(A, key)
1  heap-size[A] ← heap-size[A] + 1
2  A[heap-size[A]] ← −∞
3  HEAP-INCREASE-KEY(A, heap-size[A], key)
```

O(lg n)

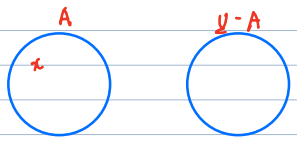
Prim's Alg (G, w, r) cf BFS - ch22.

Selection of a safe edge

$G = (V, E)$

Dijkstra's - ch25

- PQ

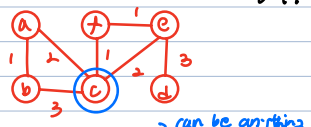


two sets of nodes

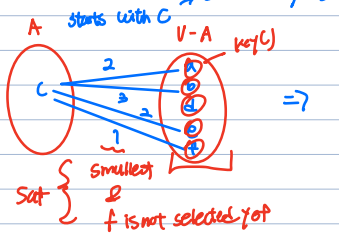
DFS (G)

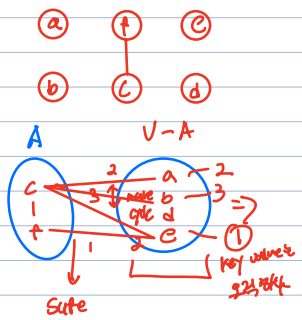
BFS (G, s)

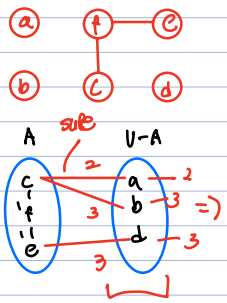
Ex

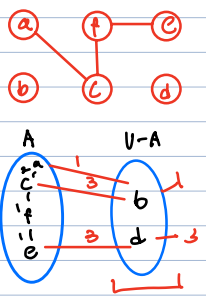


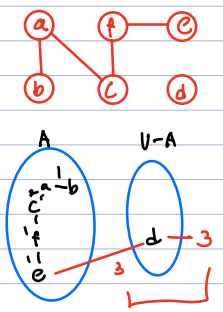
→ can be anything

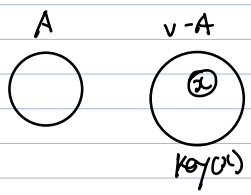




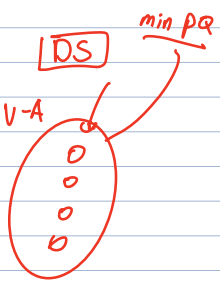




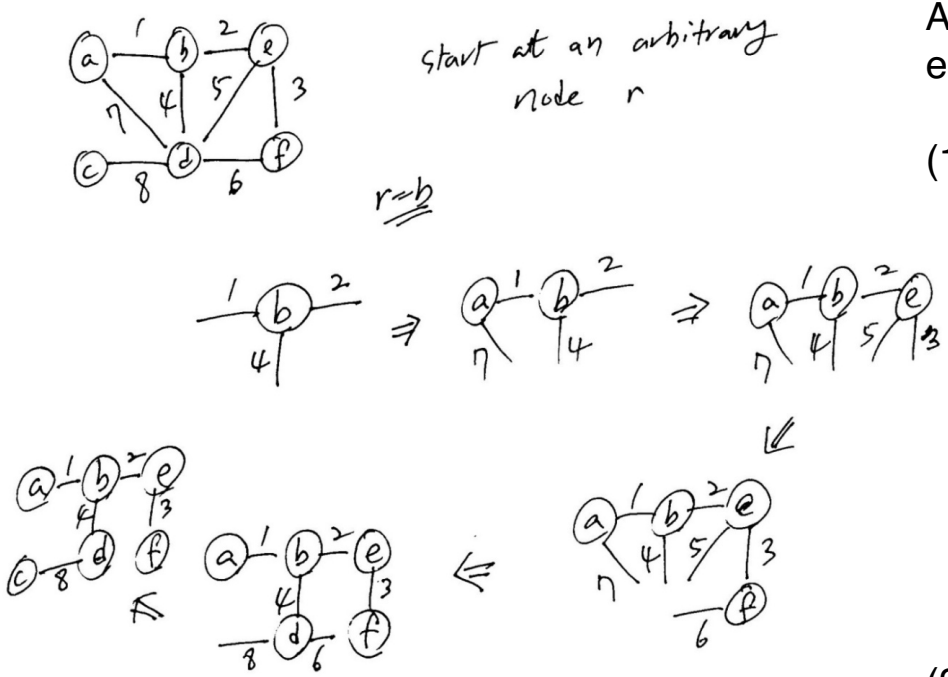




min of all edges that connect v & some node in A



Prim's algorithm

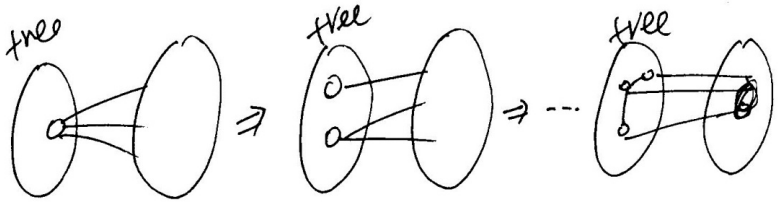


A = {r}
each time,

(1) a selection is made

a "safe" edge "e" that connects 2 sets of nodes A and B with the property that
- addition of "e" to A does not cause a problem ("cycle")
- weight(e) is minimum among the candidate nodes

(2) updates are done



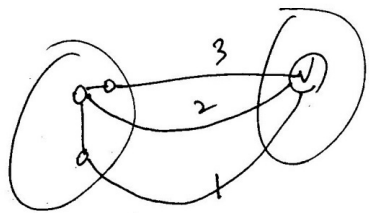
always 2 sets whose union = V (disjoint)

key(v): the min weight of any edge connecting v to a node in the tree.

in each step a safe edge that connects the 2 sets is selected.

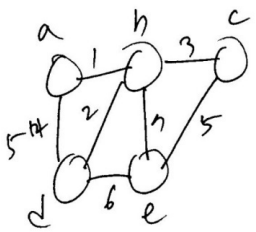
a data structure that maintains all nodes that are NOT in the tree based on key values.

Ex.



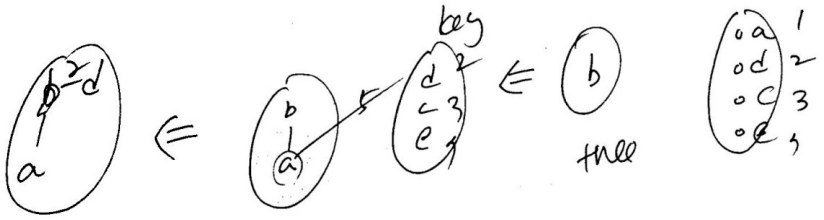
key(v)=1

```
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      key[v] ← w(u, v)
```



G.

MST-Prim(G, w, r)



$u \in$ Extract_Min Q

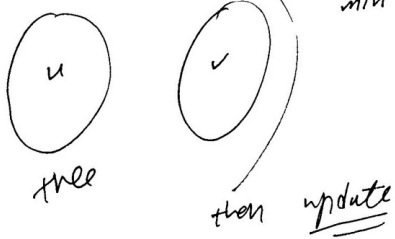
(r) is the 1st node extracted.

for each $v \in Adj(u)$

if $v \in Q$ and $w(u,v) <$ key (v)

not in the tree

current min



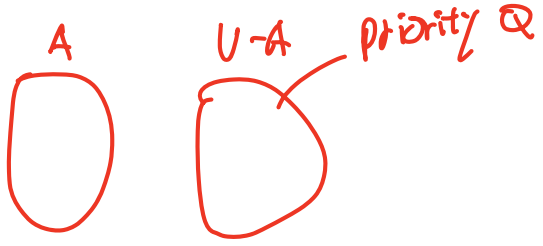
```
MST-PRIM(G, w, r)
 1  for each u ∈ V[G]
 2       do key[u] ← ∞
 3          π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  while Q ≠ ∅
 7       do u ← EXTRACT-MIN(Q)
 8          for each v ∈ Adj[u]
 9               do if v ∈ Q and w(u, v) < key[v]
10                     then π[v] ← u
11                          key[v] ← w(u, v)
```

Line 11 – DeacreaseKey(A,i,k)

Time complexity

Line1~5 O(|V|)
Line 6 O(|V|) iterations
Line 7 O(log V)
Line 8~11 O(|E|)

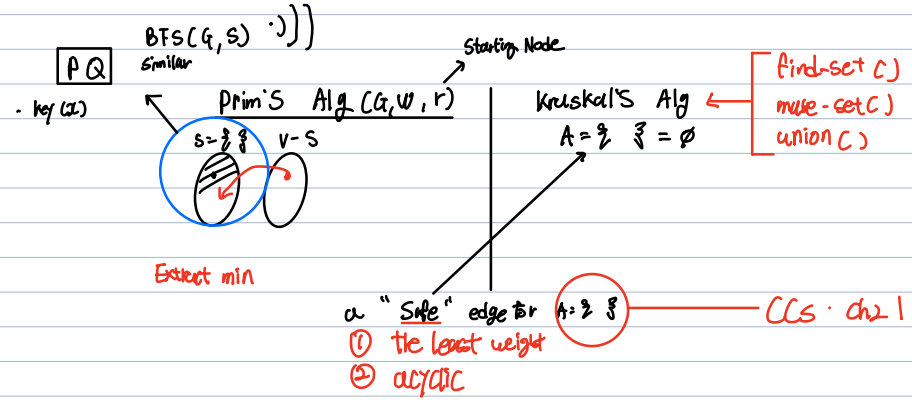
Therefore O(V log V + E log V) = O(E log V)

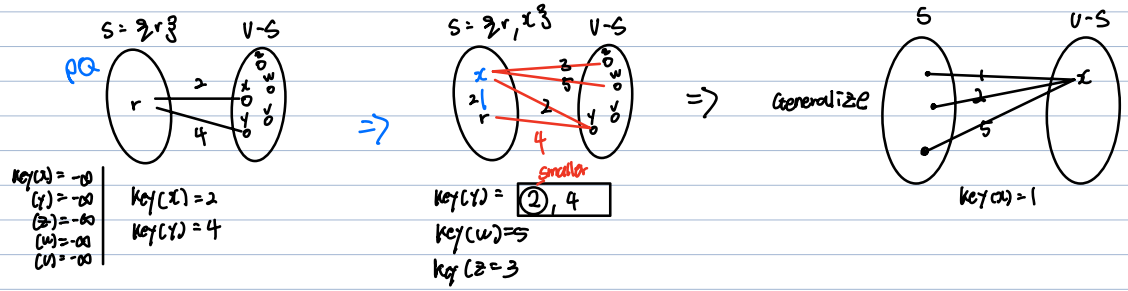


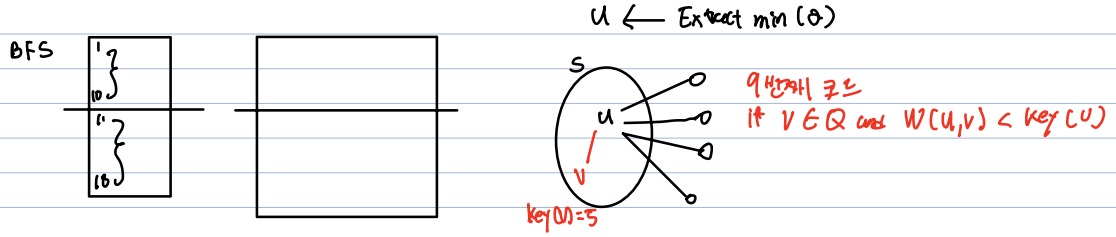
11 (21 월

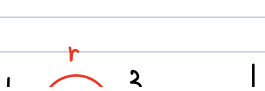
Ch23 MSTP

weighted connected undirected
↓
a spanning tree (G)
$G=(V,E)$
selection of n-1 edges $|V|=n$

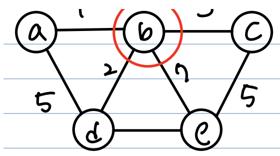




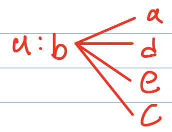


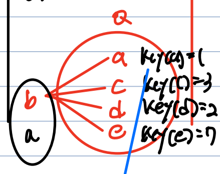


line 4
key(b)=0
while Q≠∅
for each v ∈ Adj[u]



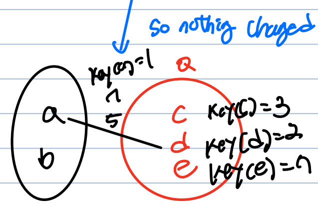
Key (a) ← ∞
b ∞
c ∞
d ∞
e ∞

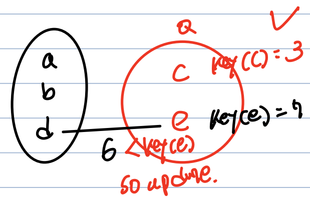


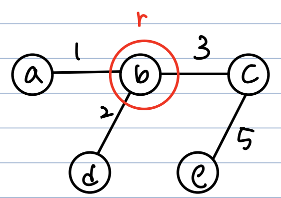


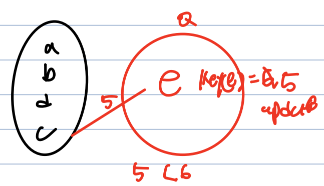
if ① v ∈ Q
② w(u,v) < key(v)

then we now update key value
then key(v) ← w(u,v)









11/22 화

Ch 23 Prim's Alg
p.40

O(v) [ ] key( )
π( )

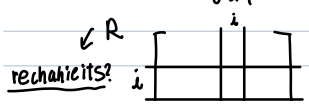
while Q ≠ ∅
u ← extract min
② v ∈ Adj(u)
− v ∈ Q
−

O(lg v) O(E lg v)

O(E lg v)

Pg 41

The transitive closure (G) — a directed graph
↓
n×n a boolean matrix (# of nodes; 0 or 1)



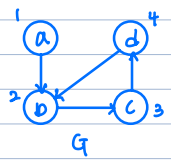
1 ≤ i, j ≤ n

when can we have 1?

R[i,j] is 1 if ∃ a pos length path from node i to node j

Warshall's Algorithm ≈ Floyd's

Ex



Adj mat

	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	0	0	0	1
d	1	0	0	0

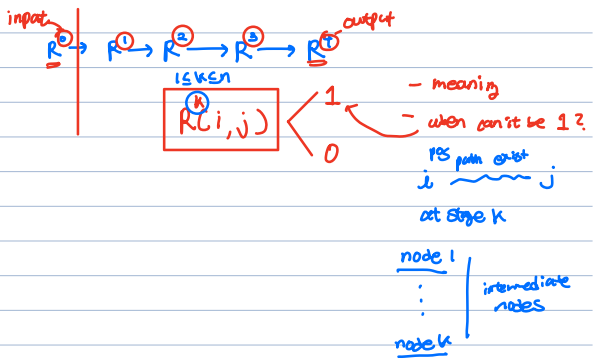
R^0

R(G)

	a	b	c	d
a	0	1	1	1
b	0	1	1	1
c	0	1	1	1
d	0	1	1	1

R^4 → 4 nodes

R^n

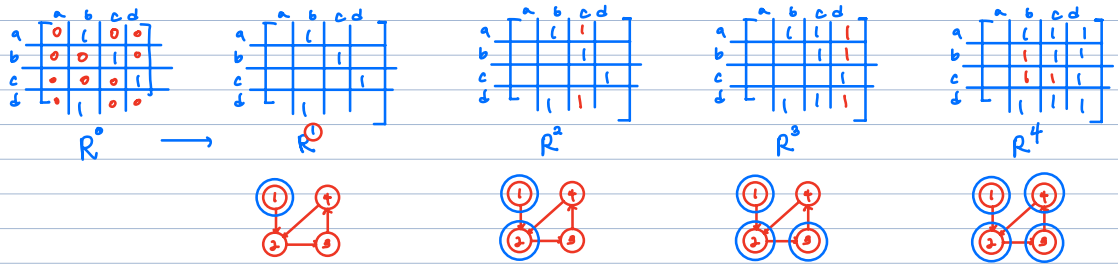


$R^k(i,j) = 1$

Base case

① $R^{k-1}(i,j) = 1$

② $R^{k-1}(i,k) = 1$ and $R^{k-1}(k,j) = 1$



n × n

① k n

② ~ n

③ 0 n

Reachability

the transitive closure of a directed graph with n vertices is the n by n boolean matrixm, where the element in ith row and jth column is 1 if there exists a directed path of a positive length from the ith vertex to the jth vertex. otherwise, the element is 0.

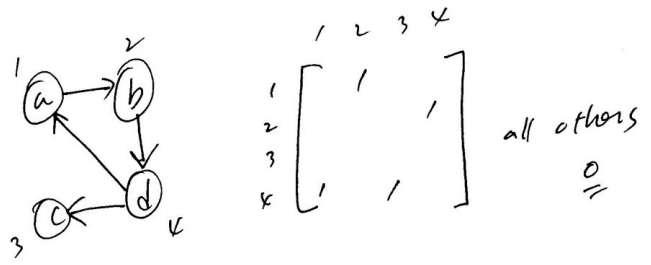
example: consider a directed graph that consists of 4 nodes, a, b, c, d and 4 edges, (a,b), (b,d), (d,a), (d,c)

the adjaceny matrix and the transitive closure as follows:

	a	b	c	d
adjacency matrix a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

row → column

transitive closure	1	1	1	1
	1	1	1	1
	0	0	0	0
	1	1	1	1



$A = \{a, b, c, d\}$

$R = \{(a,b), (b,d), (d,a), (d,c)\}$

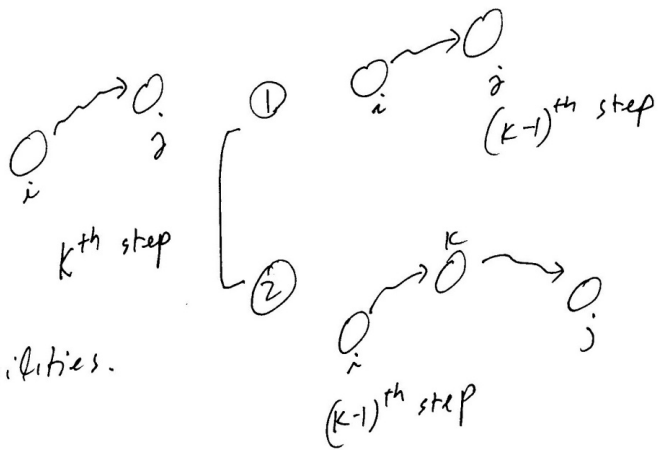
$\Downarrow$

$R^t = \{(a,a), (a,b), (a,c), (a,d), (b,a), (b,b), (b,c), (b,d), (d,a), (d,b), (d,c), (d,d)\}$

```
R^{0} <- adjacency matrix of the input directed graph
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      R^{k}[i,j] = R^{k-1}[i,j] or (R^{k-1}[i,k] and R^{k-1}[k,j])
return R^{n}
```

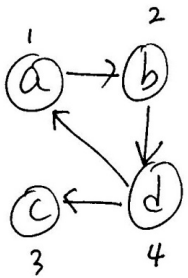
$$R^k(i,j) = or\left(R^{k-1}(i,j),\ \Uparrow\right)$$

$$and\left(R^{k-1}(i,k),\ R^{k-1}(k,j)\right)$$



2 possibilities.

Ex



adjacency matrix $= R^0$

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	0	0	0
4	1	0	1	0

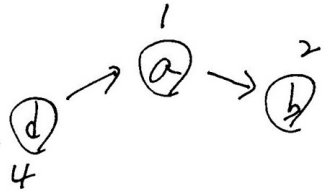
$R^1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ $R^2 = ?$

node 2 is an intermediate vertex

	1	2	3	4
1		1		1
2				
3				
4	1	(1)	1	

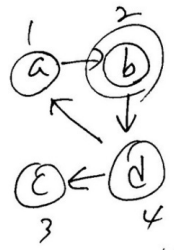
new

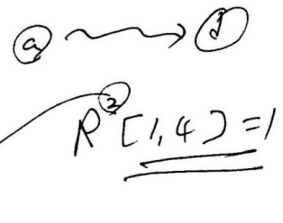
node 1 is an intermediate vertex



$R^1[4,2] = 1$

$R^2 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$





$R^{(2)}[1,4] = 1$

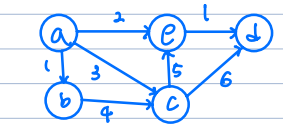
$R^3 = R^2$ $R^4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Floyd's Alg

All pairs shortest distances problem
paths

Dijkstra's Alg — distances
single source shortest paths

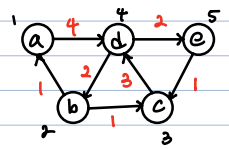
Ex



a, d
c, e

Distance Matrix — a distance matrix

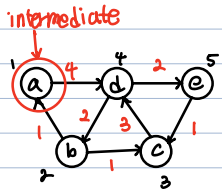
Ex



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

D^0

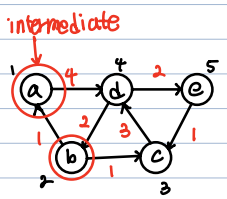
⇒ intermediate



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

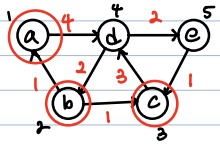
D^1

intermediate



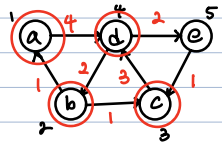
	a	b	c	d	e
a	0			4	
b	1	0	1	5	
c			0	3	
d	3	2	3	0	2
e			1		0

D^2



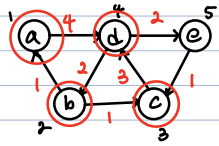
	a	b	c	d	e
a	0			4	
b	1	0	1	4	
c			0	3	
d	3	2	3	0	2
e			1	4	0

D^3



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

D^4



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

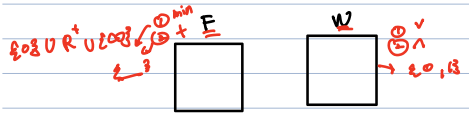
D^5

floyd = 최단경로 찾기

$$D(i,j) = \min(D^{k-1}(i,j),\ D^{k-1}(i,k) + D^{k-1}(k,j))$$

$$R^k(i,j) = \text{or}(R^{k-1}(i,j),\ \text{and}(R^{k-1}(i,k),\ R^{k-1}(k,j)))$$

Warshall = 1 or 0



Floyd's algorithm solves all-pairs shortest paths problem

$D = D^0 \Rightarrow D^1 \Rightarrow D^2 \cdots \Rightarrow D^n$

example: consider a directed graph that consists of 4 vertices, a,b,c,d and 5 edges, (a,c), (b,a), (d,a), (c,d), (c,b). assume that each has has the following weights:

- (a,c) - 3
- (b,a) - 2
- (d,a) - 6
- (c,d) - 1
- (c,b) - 7

the distance matrix for this graph is as follows:

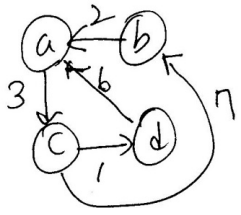
0	infinity	3	infinity
2	0	infinity	inginity
infinity	7	0	1
6	infinity	infinity	0

```
D <- distance matrix of the input directed graph
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      D[i,j] = min{D[i,j], D[i,k] + D[k,j]}
return D
```

min distance from i to j at stage $\leq k$

at stage $\leq k$

$$D^k[i,j] = \min\left(D^{k-1}(i,j),\ D^{k-1}(i,k) + D^{k-1}(k,j)\right)$$



D

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

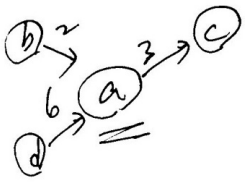
$D_0 = D$

node a

intermediate node

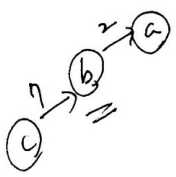
$D_1 =$

	c
b	5
d	9



$D_1 = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$

node b



$D_2 =$

	a			
a	0			
		0		
c	9		0	
				0

a, b, c

$D_3 =$

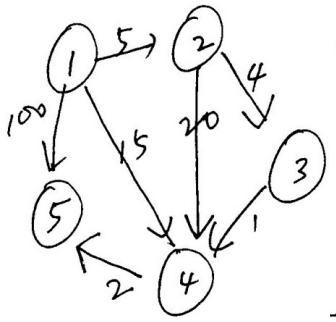
	b	d
a	10	4
b		6
d	16	

D_4 c → d → a

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

Single source shortest distances (paths) problem – Dijkstra's algorithm

```
1  S = {1}
2  for i = 2 to n do
3     D[i] = C[1,i]      // initialization
4  for i= 1 to n-1
5     choose a vertex w in V - S such that
6          D[w] is a minimum
7     add w to S
8     for each vertex v in V - S
9          D[v] = min(D[v], D[w]+C[w,v])
```



$D(2) = c(1,2) = 5$

$D(3) = c(1,3) = \infty$

$D(4) = c(1,4) = 15$

$D(5) = c(1,5) = 100$

initialization.