Graph

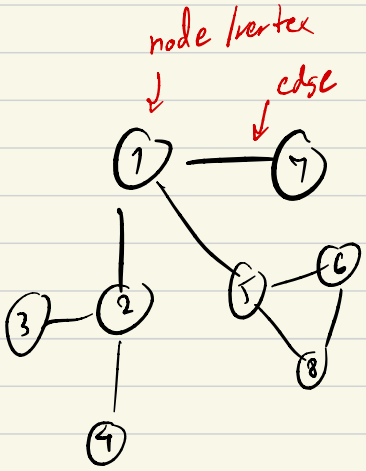
ต้องรู้ BFS, DFS → SSSP → MST

$G = (V, E)$

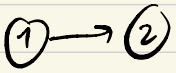
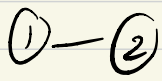
G คือ คู่อันดับของ เซต V และ เซต E

V คือ เซตของ node (จุดยอด) $V = \{1, 2, 3, 4, \dots, 8\}$

E คือ เซตของ Edge (เส้นเชื่อม) $E = \{(1,7), (1,2), (1,5), (5,6), (5,8), (6,8), (2,3), (2,4)\}$

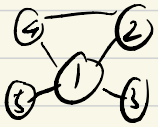


Edge

- มีทิศทาง (Directed)  1 ไป 2 ได้ 2 ไป 1 ไม่ได้
- ไม่มีทิศทาง (Undirected)  1 ไป 2 ได้ 2 กลับ 1 ได้

Degree (Undirected Graph)

Degree คือ จน. เส้นเชื่อมที่จุดยอดนั้น



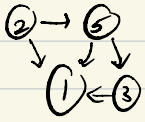
$deg(1) = 4$
$deg(4) = 2$
$deg(3) = 1$

* $\sum degree = 2|E|$

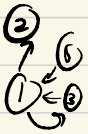
In-degree / Out-degree (Directed Graph)

In-degree คือ จน. ลูกศรที่ชี้เข้า

Out-degree คือ จน. ลูกศรชี้ออก



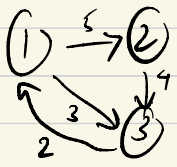
$indeg(1) = 3$
$indeg(3) = 1$



$outdeg(1) = 2$
$outdeg(2) = 0$

Weight

Weight คือ น้ำหนักของ edge



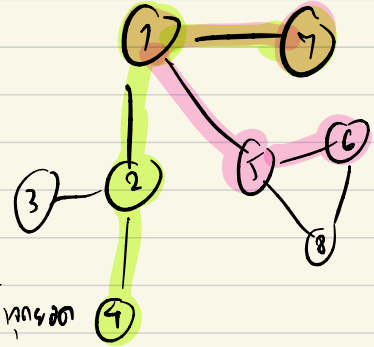
Path

Path คือ ทางเดิน

↳ 4 → 2 → 1 → 7 → 1 → 5 → 6

Path คือ ลำดับของจุดยอด

$P = (4, 2, 1, 7, 1, 5, 6)$

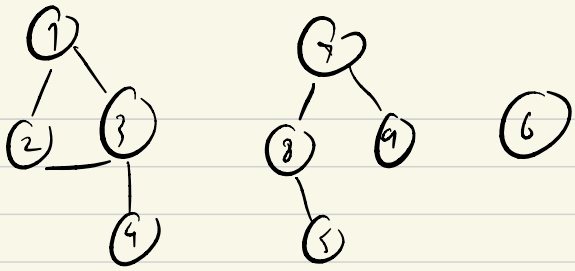


Component = 1

- Simple Path คือ ทางเดินที่ไม่ซ้ำจุดยอด

$P = (4, 2, 1, 7)$

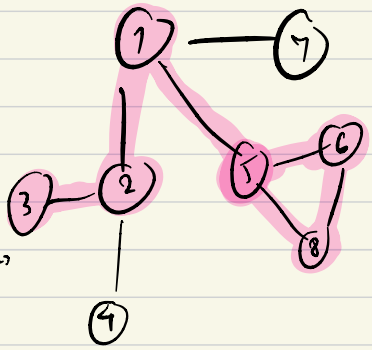
Component



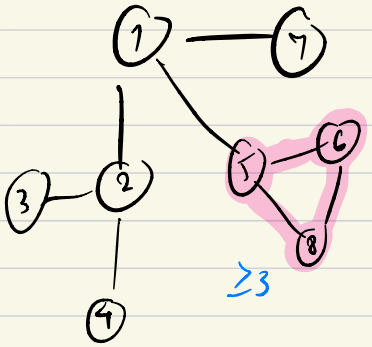
#Component = 3

(5, 8, 6, 5, 1, 2, 3, 2, 1, 5)

Cycle/Loop คือ วงรอบ
└คือ path ที่กลับมาที่เดิม

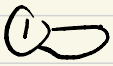


Simple loop
คือ cycle ที่มี node ซ้ำเพียง node เดียว (node ที่เริ่มต้น)

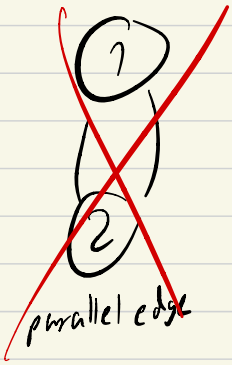


≥3

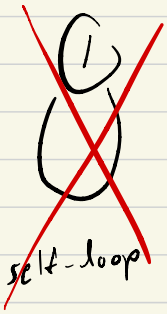
Self loop



Simple Graph - graph ที่ไม่มี parallel edge กับ self-loop



parallel edge



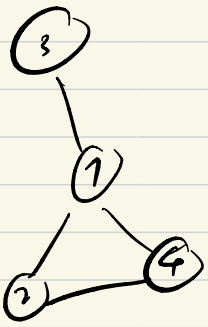
self-loop

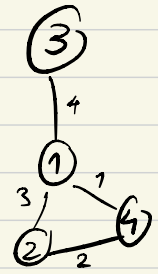
การเก็บ Graph

- Adjacency matrix

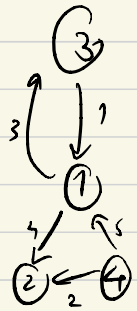
	1	2	3	4
1	0	3	4	1
2	3	0	0	2
3	4	0	0	0
4	1	2	0	0

G[1][1] = 0
G[1][2] = 1





	1	2	3	4
1	0	4	3	0
2	0	0	0	0
3	1	0	0	0
4	5	2	0	0



|V| |E| → 4 5

1 3 3
3 1 1
1 2 4
⋮

- Adjacency list (Vector)

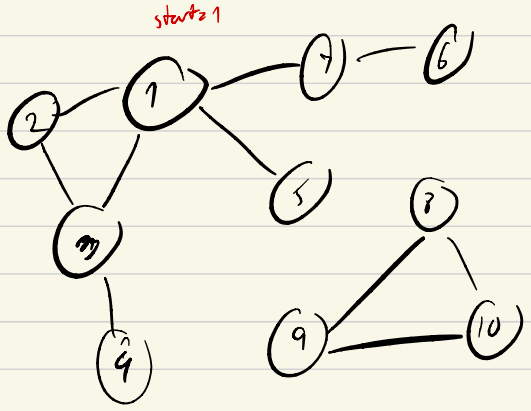
1 → 2, 3 vector[1] = {2, 3}
2 → 1, 4 vector[2] = {1, 4}
3 → 1 vector[3] = {1}
4 → 2, 1 vector[4] = {2, 1}

Graph Traversal → BFS / O(V+E)
→ DFS / O(V+E)

BFS (Breadth-First Search) - queue

visited qq

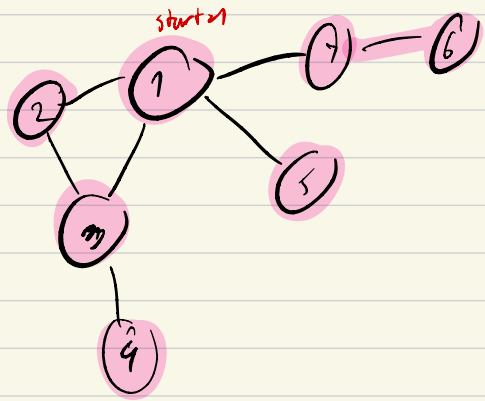
start = 1

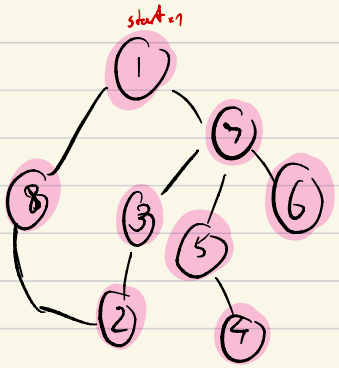



```
bool visited[N] = {}
void bfs(int start){
    queue<int> qq
    qq.push(start)

    while(!qq.empty()){
        int u = qq.front()
        qq.pop()
        visited[u] = true
        for(int v: G[u]){
            if(!visited[v]){
                qq.push(v)
                visited[v] = true
            }
        }
    }

}

int main(){
    for(u=1; u<=V; u++){
        if(!visited[u]){
            bfs(u)
        }
    }
}
```

O(V+E)

DFS (Depth-First Search) → recursion






```
void dfs(int v){
    if(vis[v]) return
    vis[v] = true
    for(v : G[v]){
        dfs(v)
    }
}

int main(){
    for(v=1; v<=V; v++){
        dfs(v)
    }
}
```

x3/2

a ↗ x3 b
↘ //2

a=11 b=15

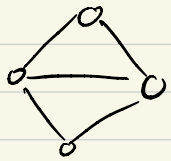
11 →(/2) 5 →(x3) 15 = 2

11 →(x3) 33 →(/2) 16 →(x3) ... →15

11 ↗(x3) 33 (cnt=1)
↘(/2) 5 (cnt=1)

*O(k)
a,b≤k

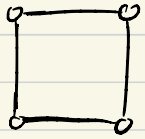
Euler path



เดินครบทุก edge โดยไม่ซ้ำ edge

deg ดีไม่เกิน 2 เท่านั้น ที่เหลือเป็นคู่

Euler circuit



เดินครบทุก edge โดยไม่ซ้ำ edge และกลับมาที่เดิม

deg ทุก node เป็นเลขคู่

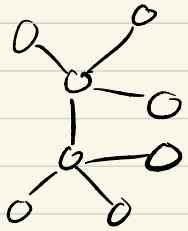
Tree

special case ของ graph

3 ข้อ 1. มี n-1 edge

2. ไม่มี cycle

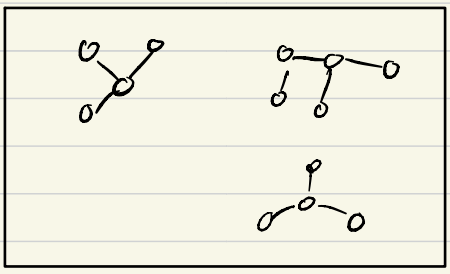
3. ต้องเชื่อมกัน (Connected component)



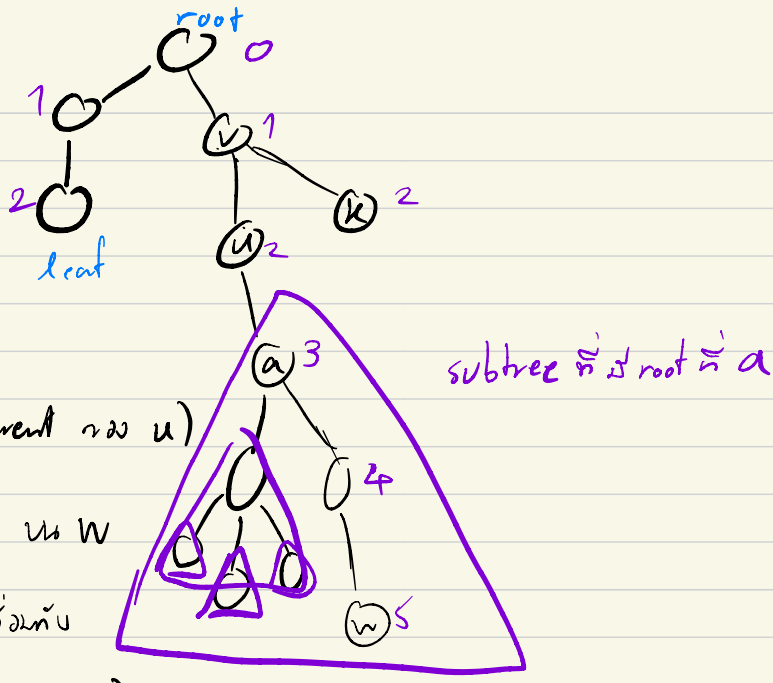
- ทุกคู่ (u,v) ของ graph จะมี 1 path เท่านั้น

Forest

graph ที่มี tree หลายต้น



Terminology



root - จุดกำเนิดของต้นไม้

leaf - node ที่ไม่มีลูก

parent - พ่อ (v เป็น parent ของ u)

ancestor - node ทุก node บน w

siblings - node ที่มี parent ร่วมกัน

child - ลูก (a เป็น child ของ v)

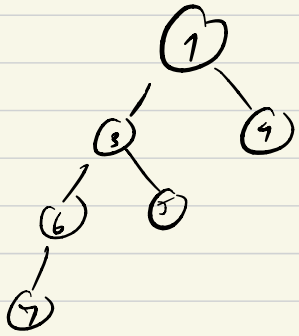
descendant - node ทุก node ใต้ v

depth - ความลึก

subtree - ต้นไม้ย่อย

Binary Tree

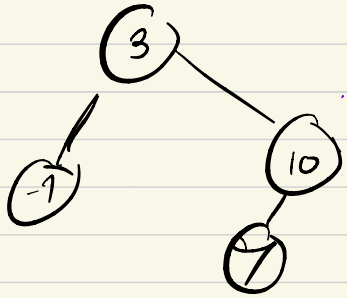
node ทุก node มีลูก ≤ 2 nodes



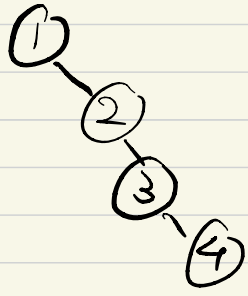
Binary Search Tree

node ลูกทางซ้าย < node พ่อ

node พ่อ < node ลูกทางขวา



Find (O(h))



Find (O(h))

Binary Search Tree

Self-Balanced BST (AVL Tree)

→ set, map ใน STL

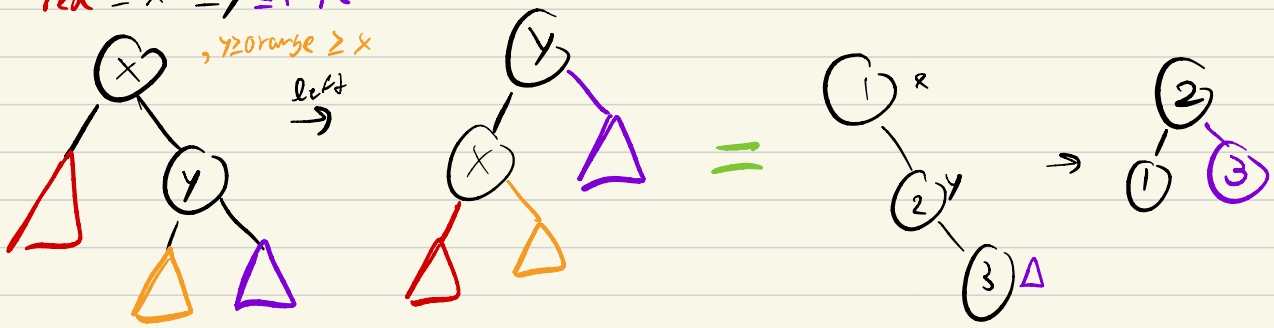
- การหา Find, Add, Delete ใช้ O(log n)

Balance : abs(dep(LC) - dep(RC)) ≤ 1

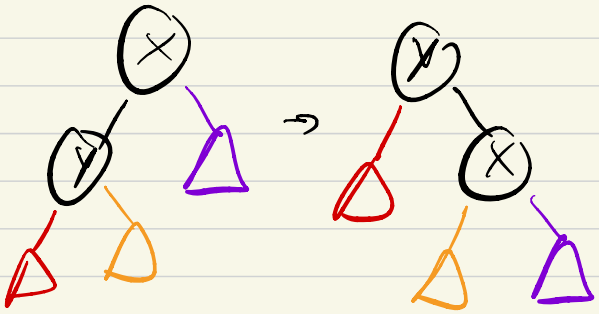
4 Operations : Left Rotation, Right-left Rotation *
Right Rotation, Left-right Rotation *

Left Rotation

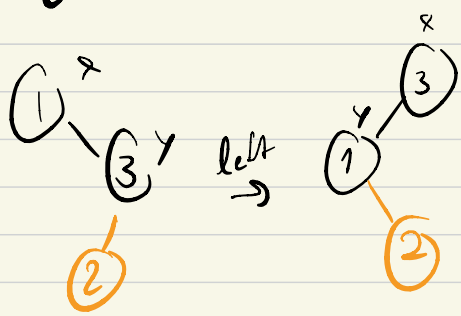
red ≤ x ≤ y ≤ purple
, y ≥ orange ≥ x



Right Rotation

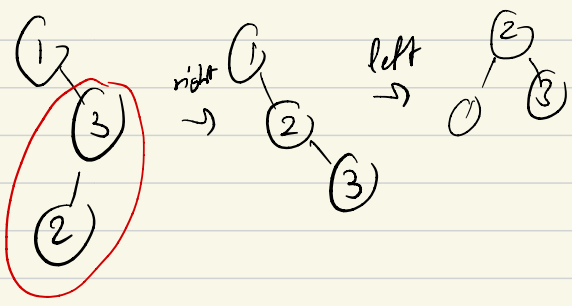


Right-Left Rotation



↓ only left rotation is not enough

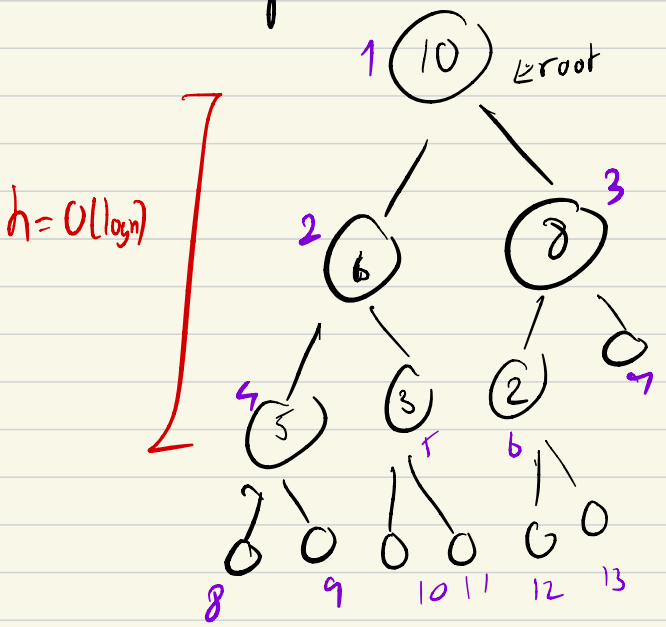
right-left (on child)



Heap (Priority Queue)

- min, max
- BT
- parent มีค่า มากกว่า/น้อยกว่า child

Max-heap



7, 2, 3, 5, 4, 6

Insert O(h) = O(logn)

heapify up

Top O(1)

Pop O(h) = O(logn)

replace root with last element

and heapify down

7, 5, 2, 3, 4

Bubble → $O(n^2)$ ×

Merge → O(nlogn)

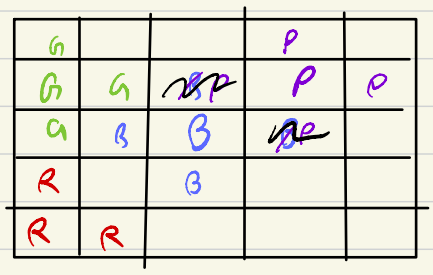
PQ (Heap) → O(nlogn)

Multiset (AVL) → O(nlogn)

เร็วสุด → ช้าสุด

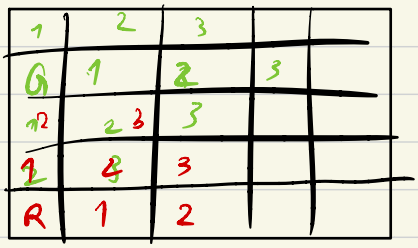
Mergesort, PQ → Multiset Bubble

Flood Fill (BFS/DFS עם מטריצה)



↓

$O(R \cdot C)$



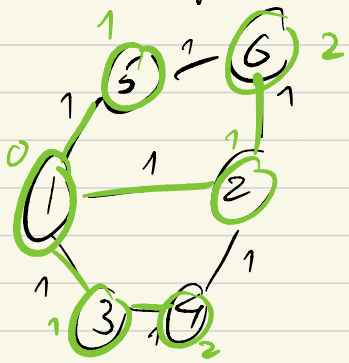
$O(k \cdot R \cdot C)$

Shortest Path

Dijkstra

Floyd Warshall

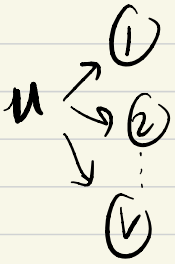
Shortest path where every edge's weight is the same



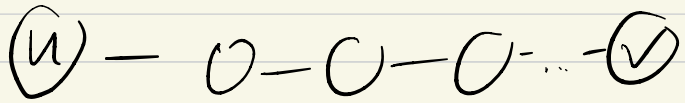
BFS

Dijkstra $O(E \log E)$

- מוצא shortest path מנקודה u לכל node בגרף

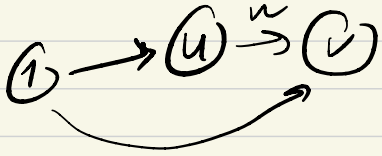


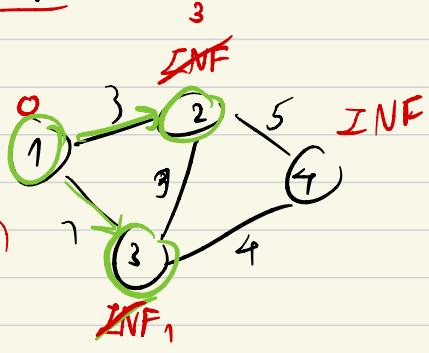
- shortest path contains at most $V-1$ edges



Bellman Ford / SPFA

- negative cycle can crash the Algo





min heap

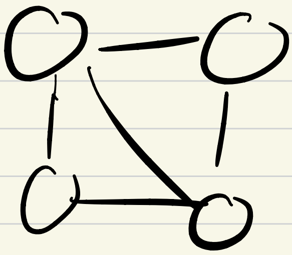
$(0, 1)$; (dis, node)

$(3, 2)$

```
pq.push ({0, 1})
while (!pq.empty()){
    auto [total_dis, u] = pq.top(); pq.pop();
    if(dis[u] != total_dis) continue
    for( auto [v, w] : G[u]){
        if ( dis[u] + w < dis[v]){
            dis[v] = dis[u] + w
            pq.push ({dis[v], v})
```

2 3 4

Dense graph



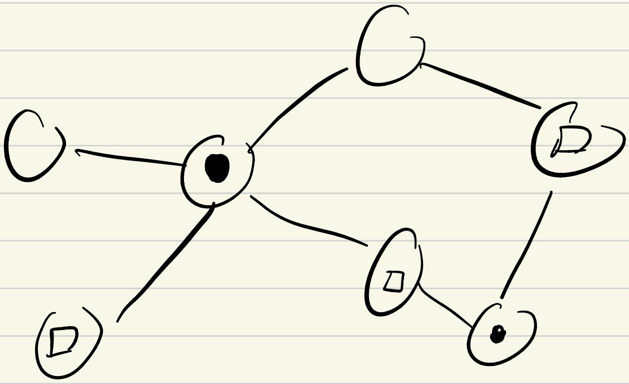
$$E = \frac{V(V-1)}{2} \approx V^2$$

$O(V^2 \log V)$

```
for (V: sou){
    for(1→V){
        ดูว่าตัวไหนยังไม่ได้เพิ่ม + dis น้อยสุด
    }
    for (v,w : G[u]){
    }
}
```

$\bar{O}(V^2)$

Multisource

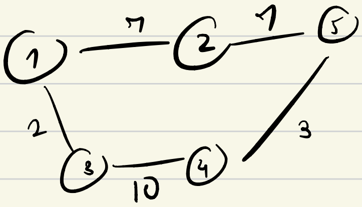


Multi state

หา Shortest path จาก 1 ไป V โดยมีตั๋วฟรีค่าผ่านทาง 1ใบ

* คิดต้องเก็บอะไรเพิ่ม / ข้อมูลไม่พอ → เพิ่ม state

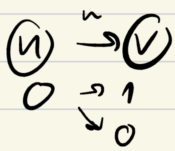
* นิยามให้ชัดเจน



dis[u][0] := SP จาก 1 ไป u โดยไม่ใช้ตั๋ว

dis[u][1] := SP จาก 1 ไป u โดยใช้ตั๋วแล้ว

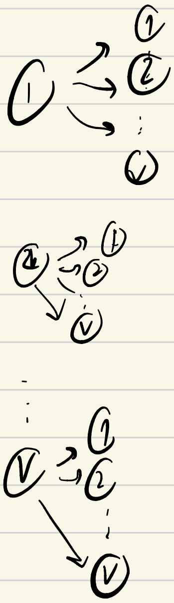
dis[u][0] ↗ dis[v][0] + w
dis[u][0] ↘ dis[v][1]



dis[u][1] → dis[v][1] + w

```
auto it = G[u].begin →

for(auto [v, w] : G[u]){
    if( dis[u][0] + w < dis[v][0])
        relax
    if( dis[u][1] + w < dis[v][1])
        relax
    if( dis[u][0] < dis[v][1])
        relax
}
```

Floyd Warshall (Allpair shortest path)



- หา shortest path สำหรับทุกคู่ของ node
- $O(V^3)$; $V \geq 1000$ แตก
- Adjacency Matrix

```
cin V, E
 for(v : 1→V) G[v][v] = 0
for( 1→E){
  int u, v, w
  cin u, v, w
  G[u][v] = min (G[u][v], w)
  G[v][u] = min (G[u][v], w)
}
```

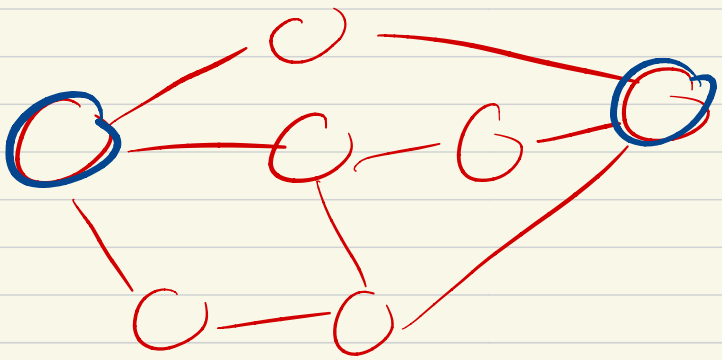
$$G[u][v] = \begin{cases} 0 ; & u = v \\ INF ; & u \neq v \end{cases}$$

```
for(k : 1→V){
  for( i : 1→V){
    for (j : 1→V){
      G[i][j] = min (G[i][j], G[i][k] + G[k][j])
    }
  }
}
```

โจทย์ SSSP

บังคับผ่าน 1 node

ต้องหาเส้นทางสั้นสุด จาก S ไป E โดยต้องผ่าน จุด k



S, E เหมือนกันทุกตัวอย่าง

$Q \le 10^5$ ตัวอย่าง

$1 \le k \le V$

$V, E \le 10^5$

S → ทุก node

E → ทุก node

dij 2 รอบ → 1. เริ่ม S dis1
2. เริ่ม E dis2

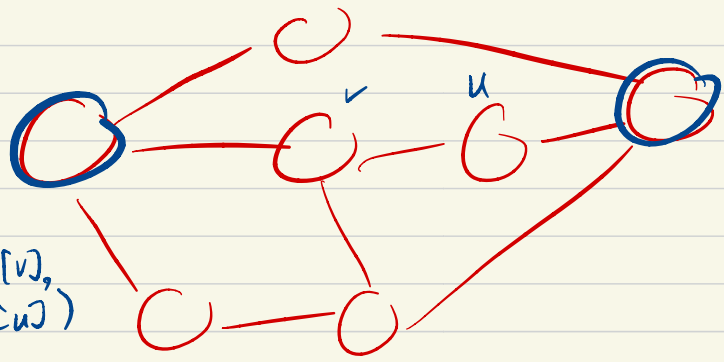
```
while(Q--){
    int k;
    cout dis1[k] + dis2[k]
}
```

dis1[k] : S → k

dis2[k] : k ← E

S → k → E

บังคับผ่าน edge 1 เส้น

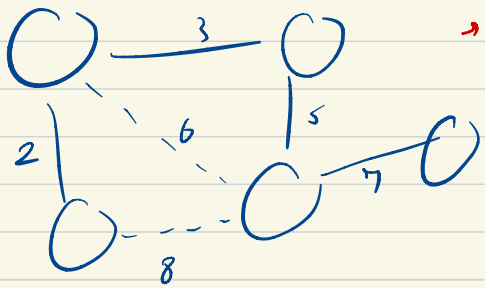


run dijkstra 2 รอบ → 1. จาก S
2. จาก E

```
while(Q--){
    int u,v;
    cout w + min(dis1[u]+dis2[v],
                 dis1[v]+dis2[u])
}
```

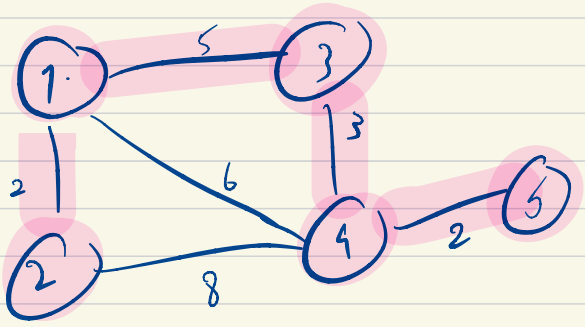
MST (Minimum / Maximum Spanning Tree) → (Kruskal) ใช้ DSU
→ (Prim)

Spanning Tree - Tree ภายในกราฟ Graph



→ Spanning Tree

Σw minimum



sort edge ตาม weight → E

เชื่อม u,v ถ้าอยู่คนละ component → DFS/BFS O(V+E)

$O(E(V+E))$ → E^2

1. เปิดมา มี V component
2. ทุกการเชื่อมจะลด #component ไป 1
3. เชื่อมทั้งหมด V-1 ครั้ง

2+5+3+2 = 12

CODE

```
for(i:1→V) par[i]=i    //important
sort edge
for(auto [w,u,v] : edges){
    if(merge(u,v)){
        ans+=w
    }
}
cout ans
```

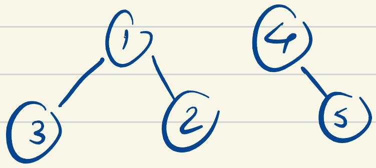
DSU (Distjoint Set Union)

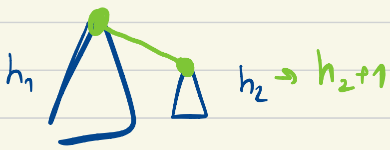
2. → หา หัว $O(h) \to O(\log n)$
   → merge (Union) → $O(\log n)$

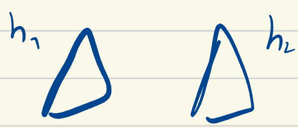
n ชั้น → 5 ชั้น

1. $O(\log n)$ Union by Rank

Merge (u,v)
1. หาหัว u,v เก็บใน hu, hv → $O(\log n)$ (hu: ตัว u, hv: ของ v)
2. ให้ตัวสูงเป็น parent ของตัวเตี้ยกว่า (Merge ที่หัว) → $O(1)$

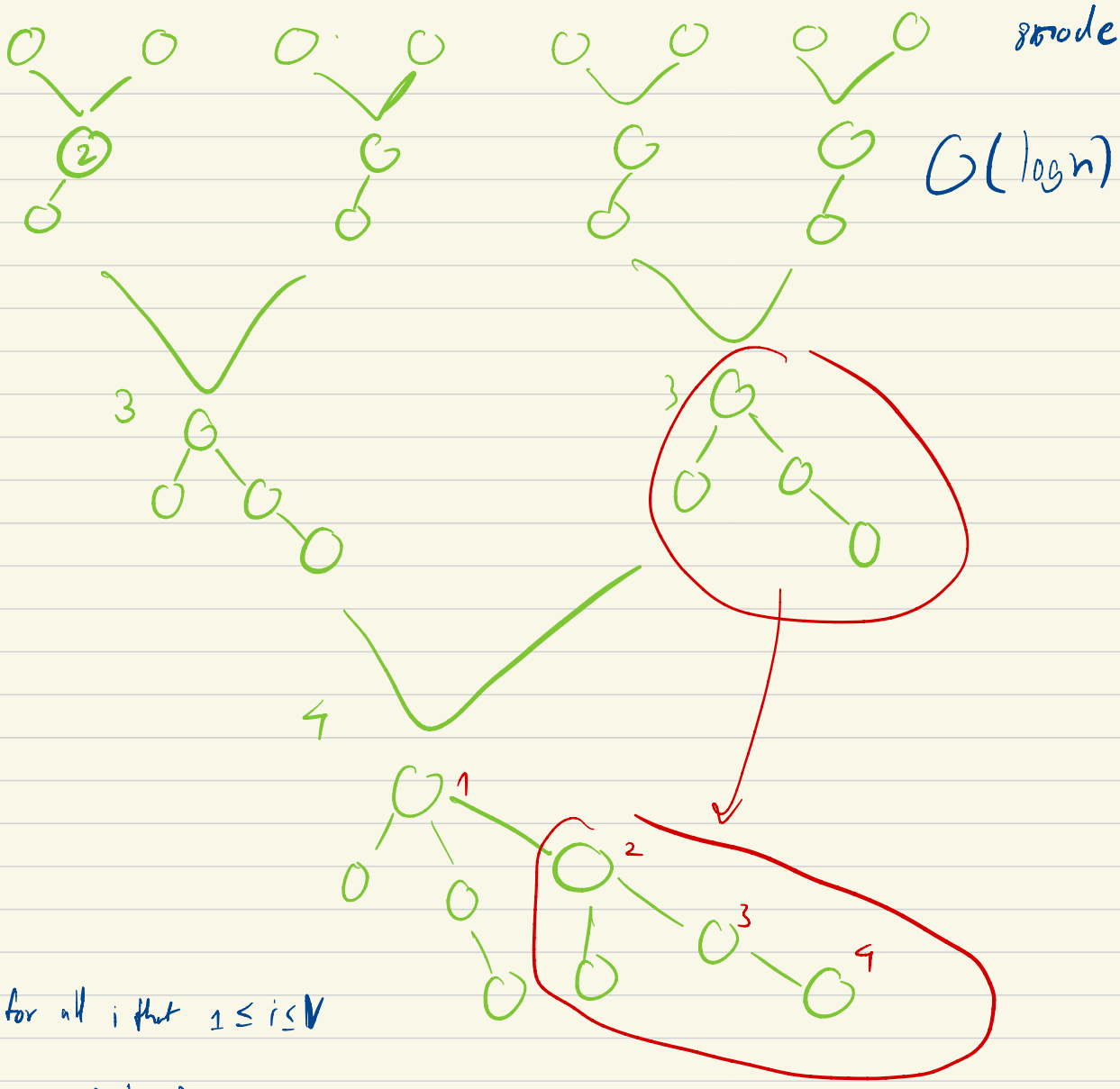






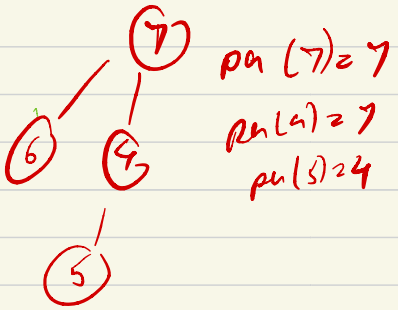
$h_1 > h_2$

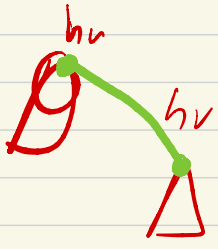
$h_1 > h_2 \to h_1$ $\quad h_1 \geq h_2+1$ $\quad h_1 = h_2 \to h_1+1$



pa(i)=i for all i that $1 \le i \le N$

```
int find_parent(int u){
    if(pa(u)!=u) return find_parent(pa(u))
    return u
}
bool unite(int u, int v){
    int hu = find_parent(u)
    int hv = find_parent(v)

    if(hu!=hv){
        if(lv(hu)>lv(hv)){
            pa(hv)=hu
        }else if(lv(hu)<lv(hv)){
            pa(hu)=hv
        }else{
            pa(hv)=hu
            lv(hu)++
        }
        return true
    }else{
        return false
    }
}
```



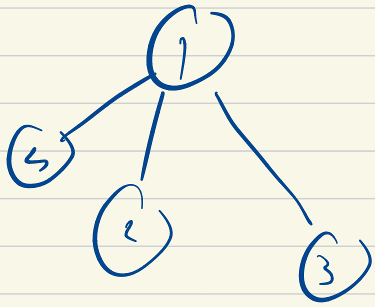


2. Path Compression $O(\alpha(n)) \approx O(1)$

(α = inverse ackermann of n)

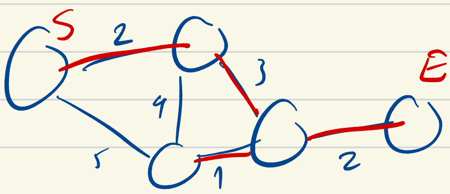
Amortized (ถัวเฉลี่ย)

```
int find_parent(int u){
    if(pa(u)!=u) return pa(u)=find_parent(pa(u))
    return u
}

bool merge(int u,int v){
    int hu = find_parent(u)
    int hv = find_parent(v)

    if(hv!=hv){
        pa(hv)=hu
        return true
    }
    return false
}
```



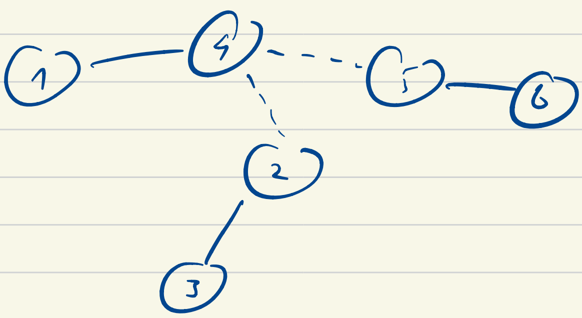
โจทย์ DSU /MST

1. ระยะทางเบาสุด (max w จาก S ไป E น้อยที่สุด)



2. ตัด edge ถามเชื่อมไหม / ปิด node

2 → 1. u กับ v เชื่อมกันไหม
→ 2. ตัด edge ระหว่าง u,v (การันตีว่ามี edge นี้)

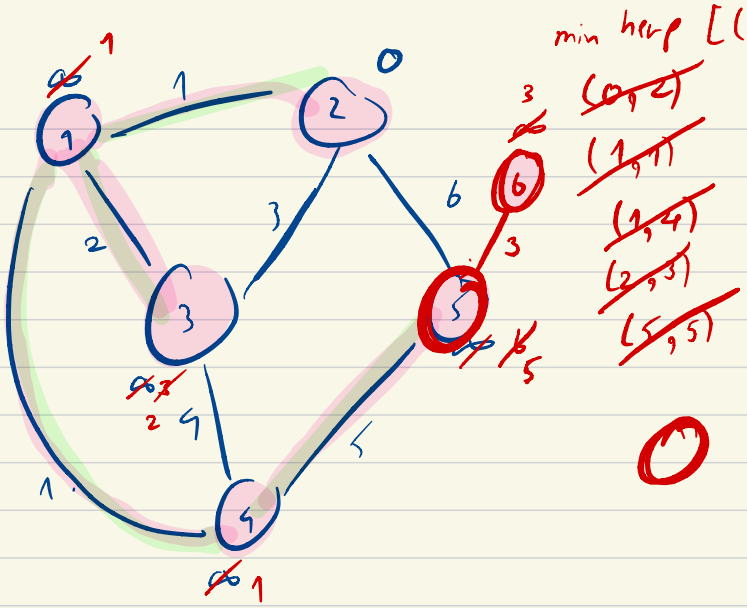


1 1 6 → Y
2 2 4
1 1 6 → Y
2 4 5
1 1 6 → N

* คิดจากหลังไปหน้า

ใช้ DSU เชื่อม edge จาก หลังไปหน้า

Prim



min heap [(wei, node)]

(0,1)
(1,1)
(1,4)
(2,3)
(5,5)

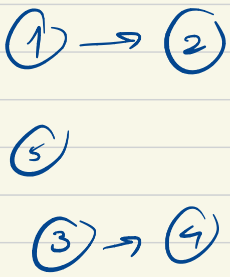
bfs / dfs
list / mat
mst
sssp

wei[u] := น้ำหนักน้อยสุดของเส้นเชื่อมที่ถึงจุดนี้

```
vector<int> wei(V+5, INF)
wei[1]=0
pq -> min heap
while(!pq.empty()){
  auto [wa, u] = pq.top()
  pq.pop
  if(vis[u]) continue
  vis[u]=true
  for(auto [v,w] : G[u]){
    if(w < wei[v] and !vis[v]){
      wei[v]=w
      pq.push(wei[v], v)
    }
  }
}
mst = Σ wei[i]   (i=1 ถึง V)
```

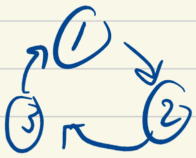
Topological Order → DP

กางเกงใน ① กางเกง ② ถุงเท้า ③ รองเท้าผ้าใบ ④ เสื้อยืด ⑤



1, 2, 5, 3, 4

1, 5, 3, 2, 4



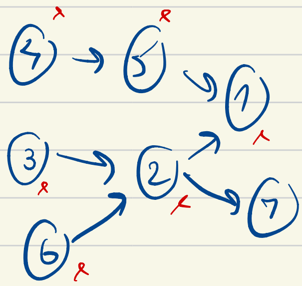
→ topo Do Not Exist

DFS หาได้ / BFS หาไม่ได้

2. → Iterative (Queue)
   ↳ Recursive (DFS)

DFS

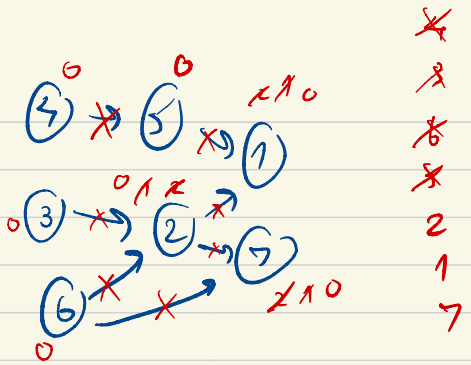
```
stack<int> topo_order
void topo(int u){
    if(vi[u]) return
    vi[u] = true
    for(v : G[u]){
        topo(v)
    }
    topo_order.push(u)
}
```



4, 3, 5, 6, 2, 7, 1

Queue

```
for(u:1~V){
  if(indeg[u]==0)
    qq.push(u)
}
while(!qq.empty()){
  int u=qq.front(); qq.pop()
  topo_order.push_back(u)

  for(v: G[u]){
    indeg[v]--
    if(indeg[v]==0){
      qq.push(v)
    }
  }
}
```



topo order : 4, 3, 6, 5, 2, 1, 7