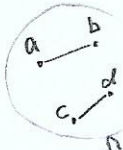
그래프?

$G = V \cdot E$ (V→노드의 집합, E→간선의 집합) · 공집합도 그래프이다.

노드와 간선의 집합



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

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

forest라는 구조

Connected graph

아무노드 2개를 선택해도 경로가 존재하는 그래프. 트리가 가장 간단한 Connected graph이다.

그래프G가 트리라면 $|E| = |V| - 1$

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

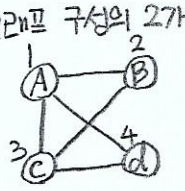
Connected and Strongly Connected.

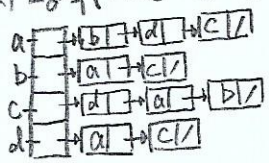
무방향 그래프가 주어졌을때 any two Vertex 사이에 경로가 존재 ← Connected

digraph 가 주어졌을때 any two Vertex 사이에 경로가 존재 ← Strongly Connected

direct graph : 방향이 있는 그래프

그래프 구성의 2가지 방식 (2차원배열 + 링크드리스트, 2차원 배열)





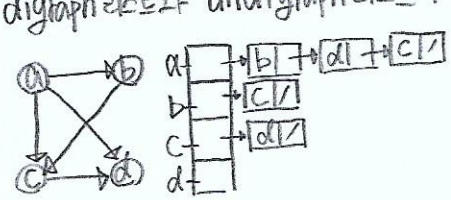
	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

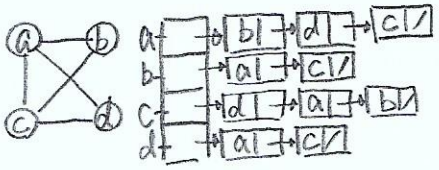
Adjacency (인접)

노드와 노드 사이에 간선이 있을때 ○—○ 두 노드는 인접해있다.

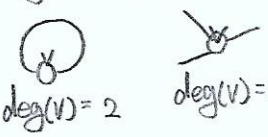
간선과 간선 사이에 같은 노드가 있을때 ⟩○⟨ 두 간선은 인접해있다.

digraph리스트와 undigraph리스트의 차이



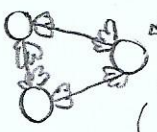


Degree. 노드에 연결된 엣지의 수



Hand shaking Lemma

여러사람이 악수를 할때 악수를하는 손의 총 갯수는 항상 짝수이다.



총6개의 손 ($2 \cdot |E|$ = 간선의 수의 2배)

같은말로 모든 Vertex Degree의 합은 even number이다.

↳ 홀수 Degree를 갖고있는 Vertex의 수는 짝수이다. (그래야워 말이됨)

증명해보자! (proof by Contradiction : 모순증명 $\sim P \vee Q$를 증명하기위해 $P \wedge \sim Q$가 False 인것을 보이는것)

($P \rightarrow Q$) 홀수 Degree를 갖고있는 Vertex의 수는 → 모두 짝수이다. (every)

($\sim P \vee Q$) 짝수 Degree를 갖고있는 Vertex의 수는 ∪ 모두 짝수이다 (every)

($P \wedge \sim Q$) 홀수 Degree를 갖고있는 Vertex의 수는 ∩ 홀수일수 있다 (some = ~every)

(모든 VertexDegree의 합은 even number = T임을이용 → False

↓

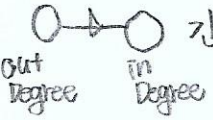
and gate에서 하나가 False이면 전체가 False

· 역이 False 이므로 원래명제는 True

악수가 n번일때 total number of shake 은 = $2 \times n$

악수하는 손

Digraph



간선 하나에 In, Out 존재

인접리스트 : 메모리 절약, 쉽게 수행가능. 단점 : a와 c사이의 간선을 확인할때 탐색해야함

2차원배열 : 간선확인 쉬움. 단점 : Modify 하기 힘들고, 메모리가 많이듦.

Graph Search algorithm → 그래프의 모든노드를 확인해보는것.

- Breath-First Search (BFS) 너비우선탐색
- Depth-First Search (DFS) 깊이우선탐색

1. Breath First Search.

d[v] = 거리를 나타낸다. 

못가는 노드는 거리를 ∞로 해놓는다.

π[v] = u 라면 v의 앞노드가 u임을 나타낸다. 

. 탐색이 끝나면 트리가 만들어진다.

White - 0 undiscovery
Gray - 1 Discovery
Black - 2 finish

방문 → 인접한 모든노드를 발견 → Finish

큐를 사용한다.

시작점

```
BFS (G,S)
  for each vertex u in V[G]-S      · 시작점을 제외한 모든 노드를 초기화
      do color[u] ← white
         d[u] ← ∞
         π[u] ← nil
  color[S] ← gray                  · 시작점을 큐에 넣기위한 준비
  d[S] ← 0
  π[S] ← nil
  Q ← ∅                            · 큐를 깨끗하게 비우기
  enqueue (Q,S)
  while Q ≠ ∅                      · 큐가 빌때까지 반복 → 큐에는 모든노드가 한번씩은 들어가므로 O(V)
      do u ← dequeue (Q)
         for each v in Adj[u]      · u노드와 인접한 모든 노드에 대해서 → 모든노드의 인접리스트를 한번씩은 돌게되는데, 모든 인접리스트의 합은 간선의 수 O(E)
                                                                        (중복x)
            do if color[v] = white
                  then color[v] ← gray
                       d[v] ← d[u]+1
                       π[v] ← u
                       enqueue (Q,v)
         color[u] ← black          · 큐에서 나왔으면 끝.
```

white - 큐에 아직 안들어감
gray - 큐에 들어감
black - 큐에서 나옴

total running time of BFS is O(V+E)

- weight 값이 여기서는 없어용
- 다 끝나면 트리가 만들어짐.
  이 트리의 predecessor subgraph 도 BFT이다.
  트리니까 |E| = |V|-1 이다.

Depth first search (시작점이 딱히 없다.)

일단 깊게가고, 리턴될때 다른데 갈수있나 알아본다.  ← 해당안됨

d[v] = discovery time (white→gray 될때)

f[v] = finishing time (gray→black 될때)

π[v] = u. u의 adjacency list 탐색할때 v가 나오면 predecessor of v = u.

white - undiscovered

gray - Discovered but not finished

black - Finished

스택을 이용.

```
DFS(G)
for each vertex u∈V[G]                    ) V번
    do color[u] ← white
       π[u] ← NIL
time ← 0
for each vertex u∈V[G]                    ) V번
    do if color[u] == white
          then DFS-VISIT(u)
```

```
DFS-VISIT(u)
color[u] ← Gray
time ← time+1
d[u] ← time
for each v∈adj[u]                         ) E번
    do if color[v] == white
          then π[v] ← u
               DFS-VISIT(v)
color ← BLACK
f(u) ← time ← time+1
```

total running time of DFS is O(V+E)

Minimum Spaning tree

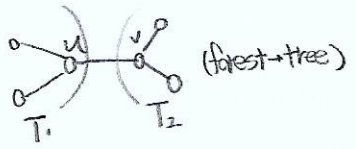
그래프에있는 엣지와 노드만 이용해서 모든노드를 포함하는 트리 (Spaning tree)

엣지의 weight 가있는(weight graph), 코스트가 가장 작은트리

$T \subseteq E$ (T=spaning tree)

MST의 부분트리도 MST

$T = T_1 + (u,v) + T_2$ (T_1 = 그래프 G_1의 MST, T_2=그래프 G_2의 MST)



(forest→tree)

증명

$W(T) = w(T_1) + w(T_2) + w(u,v)$

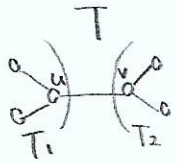
if T is a MST → than any SubTree Is a MST

P Q

P→Q가 참임을 증명하기위해 $\sim(P \to Q)$ 가 True라 가정하고 증명과정중 False 임을보임.

$P \to Q = \sim P \vee Q$, $\sim(P \to Q) = P \wedge \sim Q$

T is a MST → Some SubTree Is Not MST



T_1이 NOT MST 라면 그래프 G_1에서 MST인 T_1'이 존재할것이다.

$T = T_1 + (u,v) + T_2$

V

$T' = T_1' + (u,v) + T_2$

T is a MST 가 거짓이 된다.

$P \wedge \sim Q$ = False → $\sim P \vee Q$ (유효래밍제가 참이된다) = True

P
False

GENERIC-MST (G, w)

A ← ∅

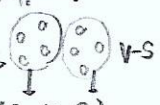
while A 가 spanning tree가 아니면

    do find edge (u,v) that is safe for A (safe = Cycle이 발생하지 않는)

    A ← A ∪ {(u,v)}

return A

판건은 어떻게 safe를 찾을것인지. V

용어 

Cut. 노드집합의 파티션 A cut (S, V-S) → 아직 선택되지않은 노드들

Cross. Edge (u,v) ∈ E crosses cut (S, V-S)

    S와 (V-S)를 연결하는 Edge (u,v)

light edge. Cross Edge중 가장 Cost가 낮은것.

A ⊆ E

G의 MST 가 T일때 A ⊆ T (A는 T의 subTree)

(u,v) 가 S와 V-S를 연결하는 light-edge 일때 → (u,v) ∈ T

P & Q

증명

P → Q = ~P ∨ Q ≡ True 를증명하기 위해 P ∧ ~Q 를 증명하다가 False임을 보임.

(u,v) 가 S와 V-S를 연결하는 light edge 일때 → (u,v) is not in T

(u,v) 가 T에 속하지 않으면 T에속하는 (u', v')가 존재할것이다.

T는 MST이므로 (u', v') < (u, v) (나중엔 부가설명 더필요)

(u,v)가 light edge라는것이 거짓.

(P) ∧ ~Q ≡ False . ~P ∨ Q ≡ True

↑

False

집합의 용어

MakeSet(x) → x하나만가지고 집합을 만든다.

Union (S_i, S_j) → S_i와 S_j를 합집합으로 만든다.

FindSet(x) → x가 어디 Set에 있는지 찾는것.