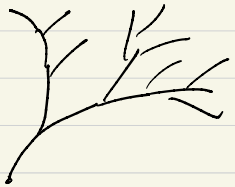
Ch 9. Trees.

§9.1. Introduction

A tree looks like

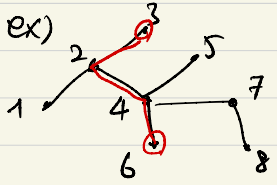


Def) A **tree** is a simple graph $T=(V,E)$ such that for any $u,v \in V$ there is a unique simple path from u to v.

no repeated

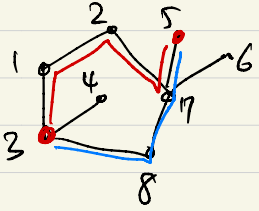
연결되었다!

ex)



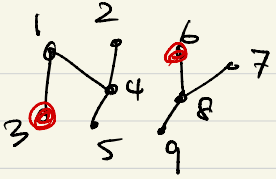
Tree

ex)



Not a tree

ex)

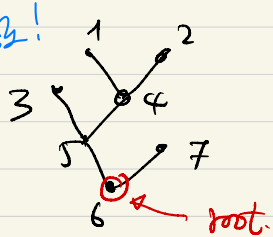


Not a tree

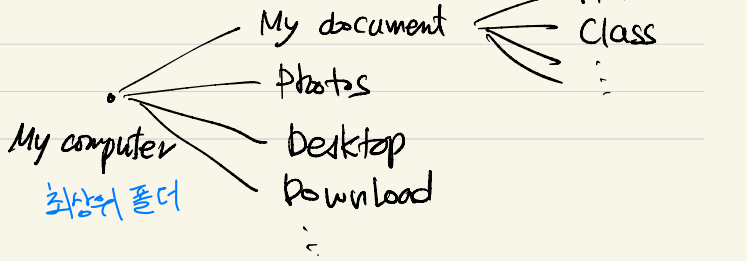
No path from 3 to 6.

Note : A tree is connected.

Def) A **rooted tree** is a tree with a special vertex called **root**.



ex) File directories



* Huffman Codes.

허프만 코드

In order to give inputs to computer we need to change data into bit strings.

letters		code
A	대응	000
B	→	001
C	→	010
D		011
E		100

BAD ⟷ 001/000/011

If one letter (E) appears more frequently than other letters (Z) then it will be more efficient to give a shorter bit string to (E). 자주 사용하는 것을 짧게!! (cost 아낌)

ex). AAABCAAAD

If we use the following coding system

letter	code
A	0
B	100
C	101
D	110
E	111

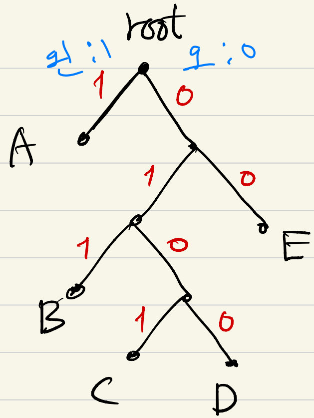
A가 자주나오므로 → shorter length

then the above word is written

0 0 0 100 101 0 0 0 110.

A A ...

ex).



	Huffman code
A	1
B	011
C	0101
D	0100
E	00

ex)

letter	frequency	code
A	2	111
B	3	110
C	7	10
D	8	01
E	12	00

encoding

Let's find an optimal Huffman code.

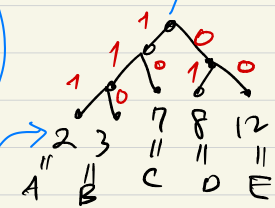
2, 3, 7, 8, 12 → 5, 7, 8, 12 —

↓

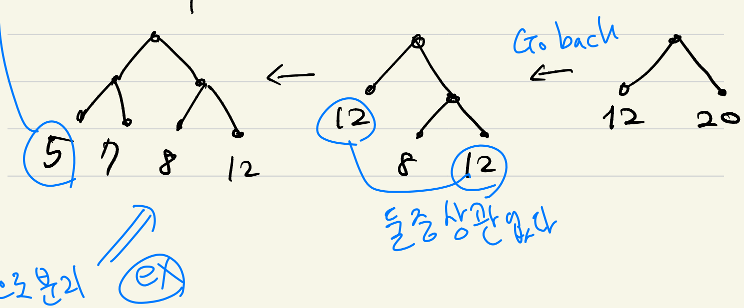
12, 8, 12 = 8, 12, 12

↓

20, 12.



↑



Algorithm to create an optimal Huffman code.

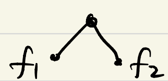
Input : A table of n letters and their frequencies (빈도수) $f_1 \le f_2 \le \cdots \le f_n$.

Output : An optimal Huffman code (as a tree)

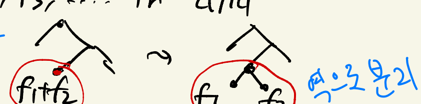
Alg : If $n=1$, $\cdot f_1$

재귀적으로

If $n=2$,



If $n \ge 3$, find Huff. code for $f_1+f_2, f_3, \ldots, f_n$ (합치고) and change



ex)

letter	frequency	code
A	2	111
B	3	110
C	17	10
D	8	01
E	12	00

CCADE

↓

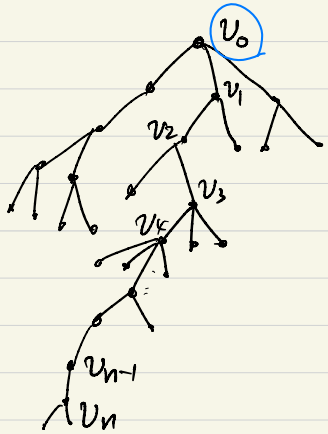
10101110100

C C A D E

recover과정: 시작부터 읽음 '1'은 매칭되는게 없어서 '10' 라인수를 늘려가며 b값을 대조함

§9.2. Terminology and Characterization of Trees.

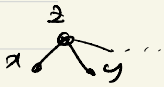
Def) T : a tree with root v_0.



$(v_0, v_1, \ldots, v_n)$: a simple path in T.

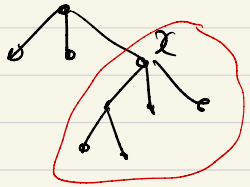
v_n is a child (자식) of v_{n-1}
v_{n-1} is the parent (부모) of v_n
$v_0, v_1, \ldots, v_{n-1}$ are ancestors (조상) of v_n

If x is an ancester of y,
y is a descendent (자손) of x.
If x, y are children of of z,
x and y are siblings (형제, 남매).
If x has no children,
x is a terminal vertex (or a leaf) (종점, 잎).
If x is not a terminal vertex,
x is an internal vertex (내부 정점).



The subtree of T rooted at x is the graph (V, E),
$V = \{x\} \cup \{\text{descendents of } x\}$
$E = \{e \mid e \text{ is an edge on a simple path from } x \text{ to some vertex in } v\}$

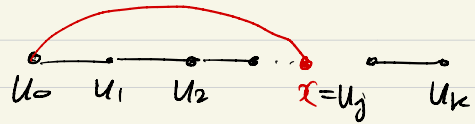
ex).



The subtree rooted at x.

Thm. T: tree with $n \geq 2$ vertices.
$\Rightarrow$ T has at least two vertices of deg 1.
(deg 1 vertex is also called a leaf.)

Pf) Find a longest simple path $(u_0, u_1, \dots, u_k)$.



Claim: $\deg u_0 = \deg u_k = 1$.

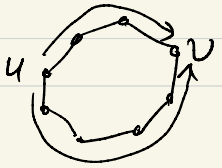
Suppose $\deg u_0 \geq 2$. Then u_0 has a neighbor $x \neq u_1$.
If $x \notin \{u_1, \dots, u_k\}$ $\Rightarrow$ then we can extend this path by adding x. Then we get a longer path, a contradiction.
If $x \in \{u_2, \dots, u_k\}$, say $x = u_j$
Then we get a cycle $(u_0, \dots, u_j, u_0)$.
There are two simple paths from u_0 to u_j: $(u_0, \dots, u_j)$ and (u_0, u_j), contradiction.

So $\deg u_0 = 1$. Similarly, $\deg u_k = 1$. □

Def) A graph is acyclic if it has no cycles.

Thm (Characterization of Tree).
T: a simple graph with n vertices.
TFAE. (The Following Are Equivalent)
① T is a tree (For any two vertices, there is a unique simple path between them)
② T is connected and acyclic.
③ T is connected and has $n-1$ edges.
④ T is acyclic and has $n-1$ edges.

Pf) We will prove ① → ② → ③ → ④ → ①
① → ②: T is connected. Suppose T has a cycle.
Consider a simple cycle. Take two vertices u, v in this cycle. Then there are two simple paths from u to v.



Since this is a contradiction, T must be acyclic.

connected는 증명되서 증명X

② ⇒ ③ : We show that T has n-1 edges by induction on n.

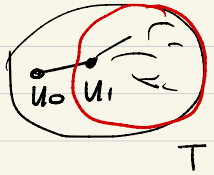
If n=1, it's clear. T = • 하나의 점

Suppose that the theorem holds for $n \geq 1$ and consider a connected and acyclic graph ← 가정 T with n+1 vertices.

Consider a longest simple path $(u_0, \dots, u_k)$. Then by the same argument in the proof of the previous theorem

$\deg u_0 = \deg u_k = 1$.

Let $T' = T$ with u_0 and edge (u_0, u_1) removed.



Then T' is connected and acycle with n vertices.

By ind hyp, T' has n-1 edges.

Then T has n edges. ← n일 때 성립함을 보임

n=1일 때 check

By induction ③ holds for all n. □

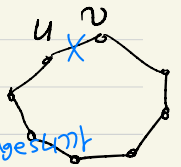
귀류법사용 (거짓으로 두고 모순발견)

③ ⇒ ④ : Suppose T is not acyclic.

Then T has a simple cycle.

Take one edge (u,v) in this cycle.

let $T = T' - (u,v)$.



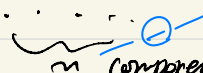
T=n-1 edges이니까

Then T' is still connected and has n-2 edges.

However, if there are n vertices, we need at least n-1 edges for a connected graph.

(∵ Adding 1 edge can reduce # components by 1.

연결되어있는 성치들



if 1 edge → n-1 components

연결되어있게 되려면

∴ 1개의 트리 (component)가 되려면 (n-1) edge여야함

Contradiction. So T must be acyclic.

하지만 n-2이므로 모순!!

④ ⇒ ① : If T is not connected, let $T_1, \dots, T_k$ be the components of T. $(k \geq 2)$ let n_i be # vertices in T_i.

Each T_i is connected and acyclic. By ② ⇒ ③

T_i has $n_i - 1$ edges. Then

edges in $T = \sum_{i=1}^{k} (n_i - 1) = \left(\sum_{i=1}^{n} n_i\right) - k$

$= n - k < n-1$.

n=1이어야함 모순

contradiction.

So T is connected.

증명 ① connected 하다보였고 ② unique

증명 ① connected 하다보였고

② 증명

Suppose that there are two distinct simple paths from u to v in T.

Then we can find a simple cycle.

This is a contradiction. ← acyclic이 전제였기에

So there is a unique simple path from u to v.

∴ 가 Tree이다 □

§9.3. Spanning Trees.

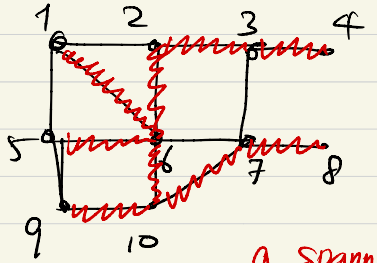
스패닝 트리

Def) G: a graph

A spanning tree of G is a subgraph of G that is a tree containing all vertices of G.

모든 정점 포함하는

ex) $G =$



a spanning tree.

(트리의 한 종류)

Thm G has a spanning tree $\Leftrightarrow$ G is connected.

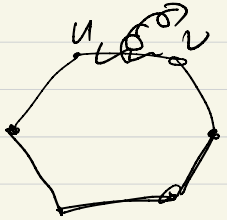
Pf) ($\Rightarrow$) Clear.

($\Leftarrow$) If G is a tree, then G itself is a spanning tree.

↑ 트리자체가 모든정점을 포함해야함

그렇지않은 모든 acyclic하지 않다면?

Otherwise, G has a cycle.



acyclic, connect $\Rightarrow$ tree

Remove one edge in this cycle.

Then the resulting graph

$G' = G - (u,v)$

is connected and has one less edges than G.

줄어 스패닝트리가짐

If G' is a tree then it is a spanning tree of G.

Otherwise we can find a cycle in G' and remove one edge there.

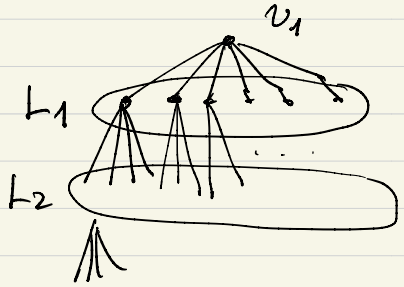
Repeating in this way we get a spanning tree of G. □

너비 우선
Breadth-First Search BFS (ST를 구성하는 알고리즘)

Input: A connected graph G with vertices ordered $v_1, v_2, \ldots, v_n$.

Output: A spanning tree T.

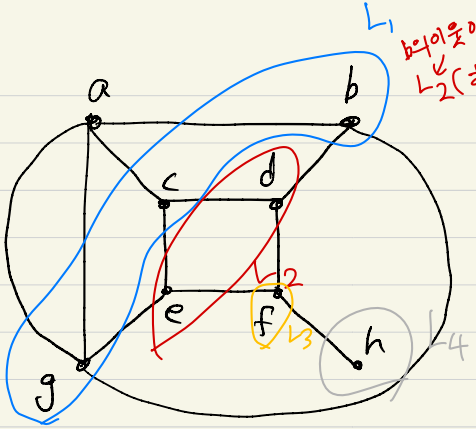
Algorithm.

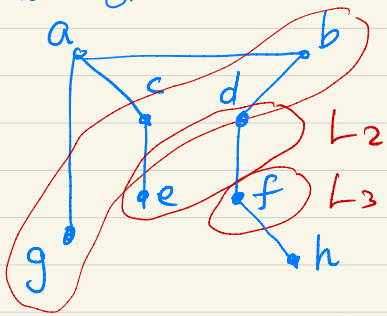


Find all neighbors of v_1

Find all new neighbors (나에게 나타나지않은 이웃) of the vertices in L_1 in the given order.

Continue this process then we obtain a spanning tree.





Spanning tree T.

Depth-First Search.

Input : A connected graph G with vertex ordering $v_1, \ldots, v_n$.

Output : A spanning tree T.

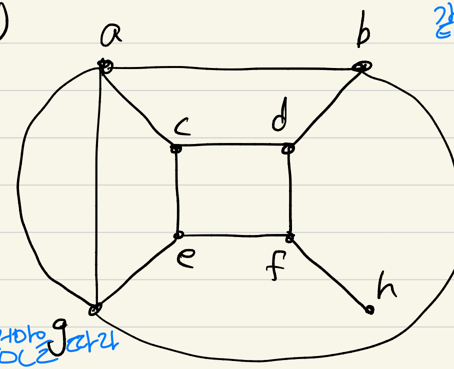
Algorithm

① Start with v_1 and keep going until no longer possible. (더이상 / 가능하지않은)
(when we visit a new vertex select the 1st vertex in the ordering). → 1번째 만난 정점만을 따라 쭉 직진!!

② When get stuck, go back one step and continue.

③ Repeat this until all vertices are visited.

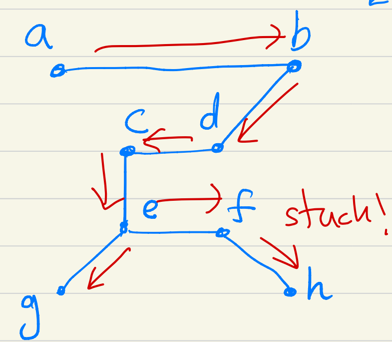
ex)



같은 조건의 정점중에서 순서정하는 기준

Vertex ordering

a b c d e f g h.

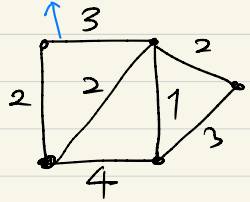


spanning tree

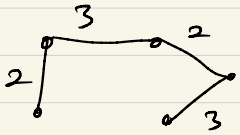
§9.4. Minimal Spanning Trees

edge weight ↑ ↳ weighted graph/net

ex)

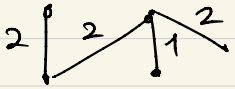


Spaning trees :



has weight 10.

∨



has weight 7.

Def) G: a weighted graph

A minimal spanning tree of G is a spanning tree of G with smallest weight. (총합)

Spanning tree → 엣지 수 최소화

minimum spanning tree → 엣지 가중치 총합 최소화

프림 알고리즘

Prim's Algorithm (Finding a minimal spanning tree)

Input : A weighted graph G (connected).

Output : A minimal spanning tree T.

Idea : Add a new edge with smallest weight to increase the size of T.
정점수

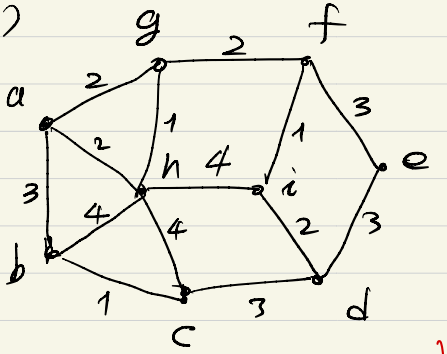
Algorithm

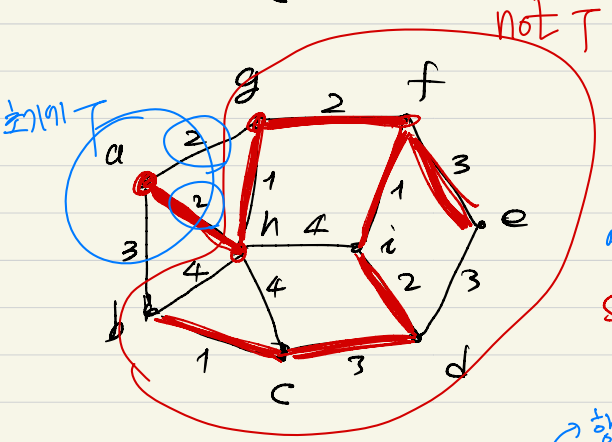
① Let T = ∘ (Take any vertex).

② Find an edge with smallest weight between a vertex in T and a vertex not in T.
Add this edge and the new vertex in this edge to T.

③ Keep doing ② until we get a spanning tree.

ex)





T와 not T 사이를 연결하는 최소가중치의 엣지 찾기 (a-g, a-h 둘 다 2인데 h 선택!)

Spanning tree.

Prim's algorithm is a greedy algorithm.
↗ 항상 그리디를 사용할 수 있는 것이다.

(매 순간마다 최소 가중치의 엣지를 누적해가는 방법)
⇒ 궁극적인 최적의 솔루션을 항상 보장하는 것은 아니다)