Chapter 5. Greedy Algorithms

한지수

Greedy Algorithm

- Minimum Spanning Tree
  - Kruskal
  - Prim
- Huffman Encoding
- Set cover

Minimum Spanning Tree

- Input: An undirected graph $G = (V, E)$; edge weights w_e
- Output: A tree $T = (V, E^{'})$, with $E^{'} \subseteq E$, that minimizes

$$weight(T) = \sum_{e \in E^{'}} w_e$$

- 사이클 없는 트리 중 weight 가 가장 작은 형태

Minimum Spanning Tree

- 특징
  - A tree on n nodes has (n-1) edges
  - Any Connected, undirected Graph $G = (V, E)$ with $|E| = |V| - 1$ is a tree.
- Cycle & the Cut Property
  - Removing a cycle edge cannot disconnect a graph

Minimum Spanning Tree

Greedy Approach (1)

- Kruskal's algorithm: 추가되지 않은(find) edge 중에 가장 작은 것을 고르고, 합치기(union)

Figure 5.4 Kruskal's minimum spanning tree algorithm.

`procedure kruskal`(G, w)
`Input:` A connected undirected graph $G = (V, E)$ with edge weights w_e
`Output:` A minimum spanning tree defined by the edges X

for all $u \in V$:
   makeset(u)

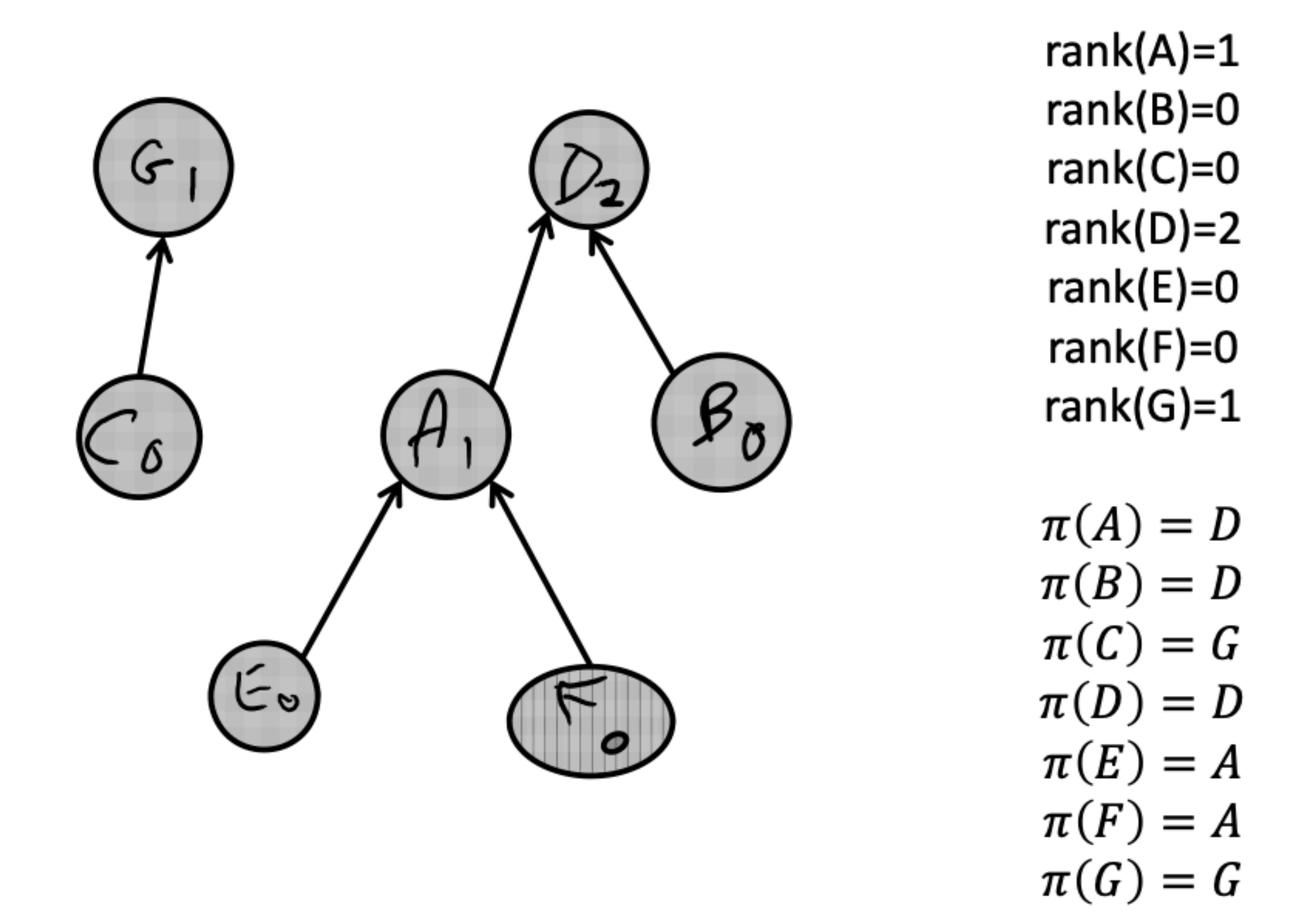
$X = \{\}$
Sort the edges E by weight
for all edges $\{u, v\} \in E$, in increasing order of weight:
   if find$(u) \neq$ find(v):
      add edge $\{u, v\}$ to X
      union(u, v)

Minimum Spanning Tree

Kruskal’s algorithm - union

- Union 함수에 쓰이는 Rank
    - For any x, rank(x) < rank($\pi(x)$)
    - Any root node of rank k has at least 2^k nodes in its tree
    - If there are n elements overall, there can be at most $n/2^k$ nodes of rank k
    - Rank: 딸린 식구, $\pi(x)$: 부모
- 더 높은 rank 를 가진 set에 추가

```
procedure union(x, y)
r_x = find(x)
r_y = find(y)
if r_x = r_y:  return
if rank(r_x) > rank(r_y):
    π(r_y) = r_x
else:
    π(r_x) = r_y
    if rank(r_x) = rank(r_y) :  rank(r_y) = rank(r_y) + 1
```



Minimum Spanning Tree

Kruskal's algorithm

<u>function find</u>(x)
if $x \neq \pi(x)$: $\pi(x) = \texttt{find}(\pi(x))$
return $\pi(x)$

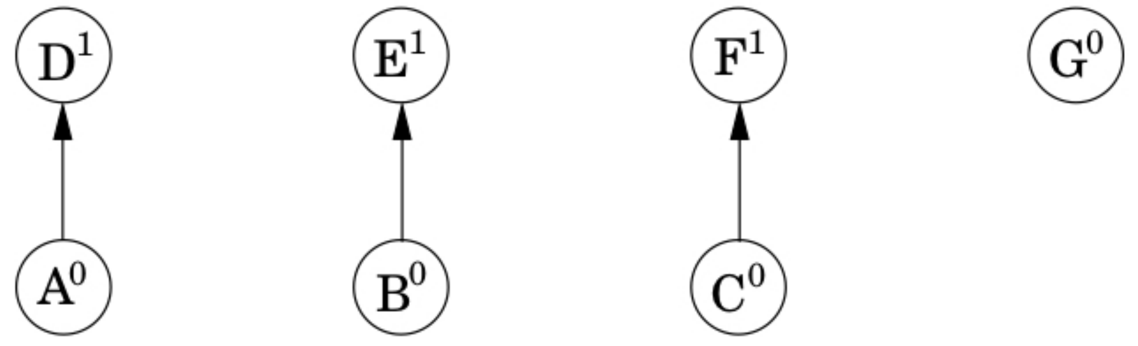
Find: 조상 일치하는 지/같은 set인지 확인
Union: rank 높은 set의 조상 달라붙기
(Rank 높은지 비교 -> 조상끼리 연결

<u>procedure union</u>(x, y)
$r_x = \texttt{find}(x)$
$r_y = \texttt{find}(y)$
if $r_x = r_y$: return
if $\texttt{rank}(r_x) > \texttt{rank}(r_y)$:
  $\pi(r_y) = r_x$
else:
  $\pi(r_x) = r_y$
  if $\texttt{rank}(r_x) = \texttt{rank}(r_y)$: $\texttt{rank}(r_y) = \texttt{rank}(r_y) + 1$

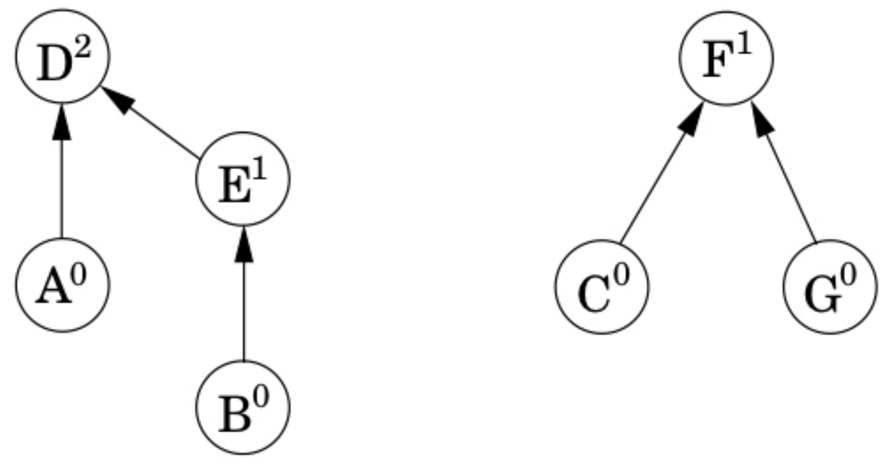
Figure 5.6 A sequence of disjoint-set operations. Superscripts denote rank.

After $\texttt{makeset}(A), \texttt{makeset}(B), \ldots, \texttt{makeset}(G)$:

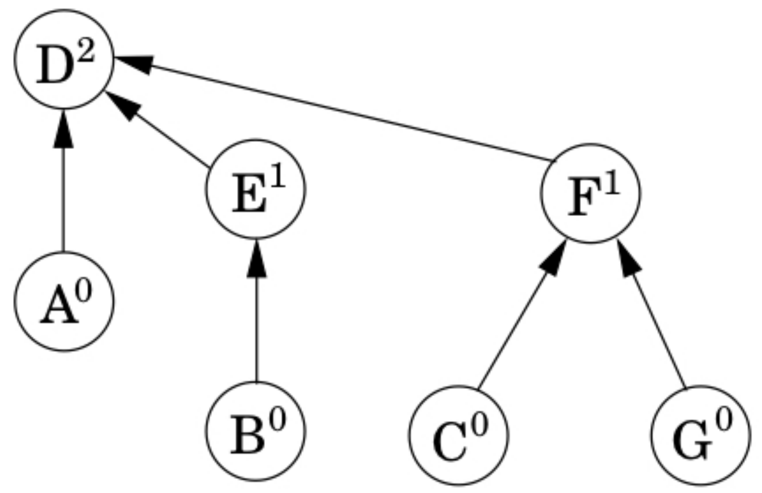
After $\texttt{union}(A, D), \texttt{union}(B, E), \texttt{union}(C, F)$:



After $\texttt{union}(C, G), \texttt{union}(E, A)$:



After $\texttt{union}(B, G)$:



Minimum Spanning Tree

Greedy Approach (1)

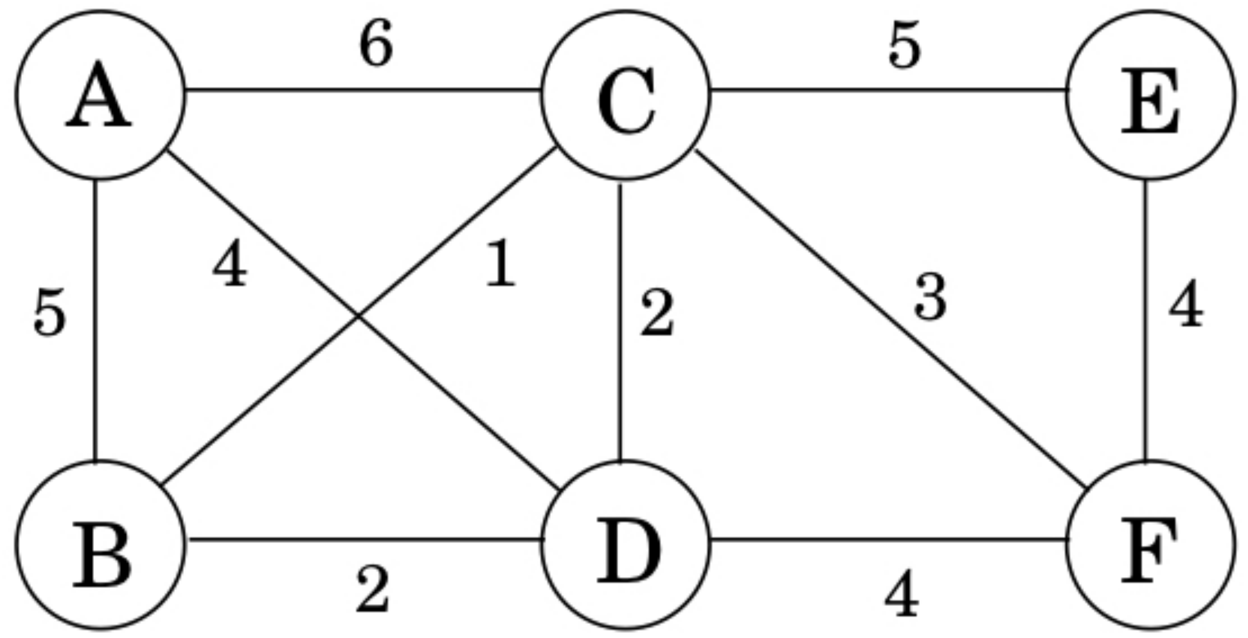
- Prim's Algorithm: 인접한 노드 들 중 최소 간선 고르고 (N-1)일때까지 진행

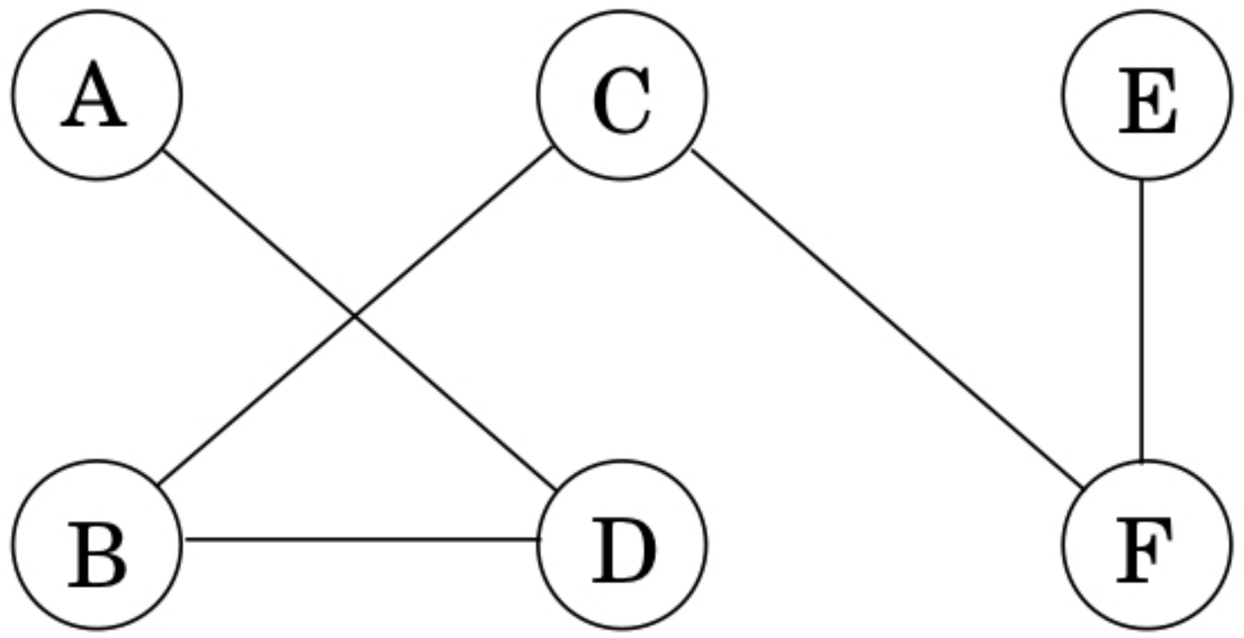
Figure 5.9 *Top:* Prim's minimum spanning tree algorithm. *Below:* An illustration of Prim's algorithm, starting at node A. Also shown are a table of cost/prev values, and the final MST.

```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
   cost(u) = ∞
   prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)    (priority queue, using cost-values as keys)
while H is not empty:
   v = deletemin(H)
   for each {v, z} ∈ E:
      if cost(z) > w(v, z):
         cost(z) = w(v, z)
         prev(z) = v
         decreasekey(H, z)
```

Minimum Spanning Tree

Greedy Approach (1)





Set S	A	B	C	D	E	F
$\{\}$	$0/\text{nil}$	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		$5/A$	$6/A$	$4/A$	∞/nil	∞/nil
A, D		$2/D$	$2/D$		∞/nil	$4/D$
A, D, B			$1/B$		∞/nil	$4/D$
A, D, B, C					$5/C$	$3/C$
A, D, B, C, F					$4/F$	

MST 만들기: Kruskal, Prim

Kruskal

- 간선 선택 (<u>사이클 형성</u>하지 않는)
- 이전 tree와 연관성 없음
- 간선 개수 == (총 정점-1)이면 끝

```
edge_set kruskal_MST(edge_set E, int n){
sort(E); // 모든 간선을 오름차순으로 정렬
edge_set MST_E = { };
for(i = 0; i<n; i++) init_set(i); // n개의 집합(트리)를 생성
while(MST_E의 간선 수 < n-1) // 종료조건 명시
(u, v) = E의 최소 가중치 간선;
E = E - {(u, v)};
if(find(n) != find(v)){ // u와 v가 다른 집합(트리)의 원소인지 확인
MST_E = MST_E U {(u, v)};
union(u, v); // 두 집합을 합병하는 연산
}
}
return MST_E;
}
```

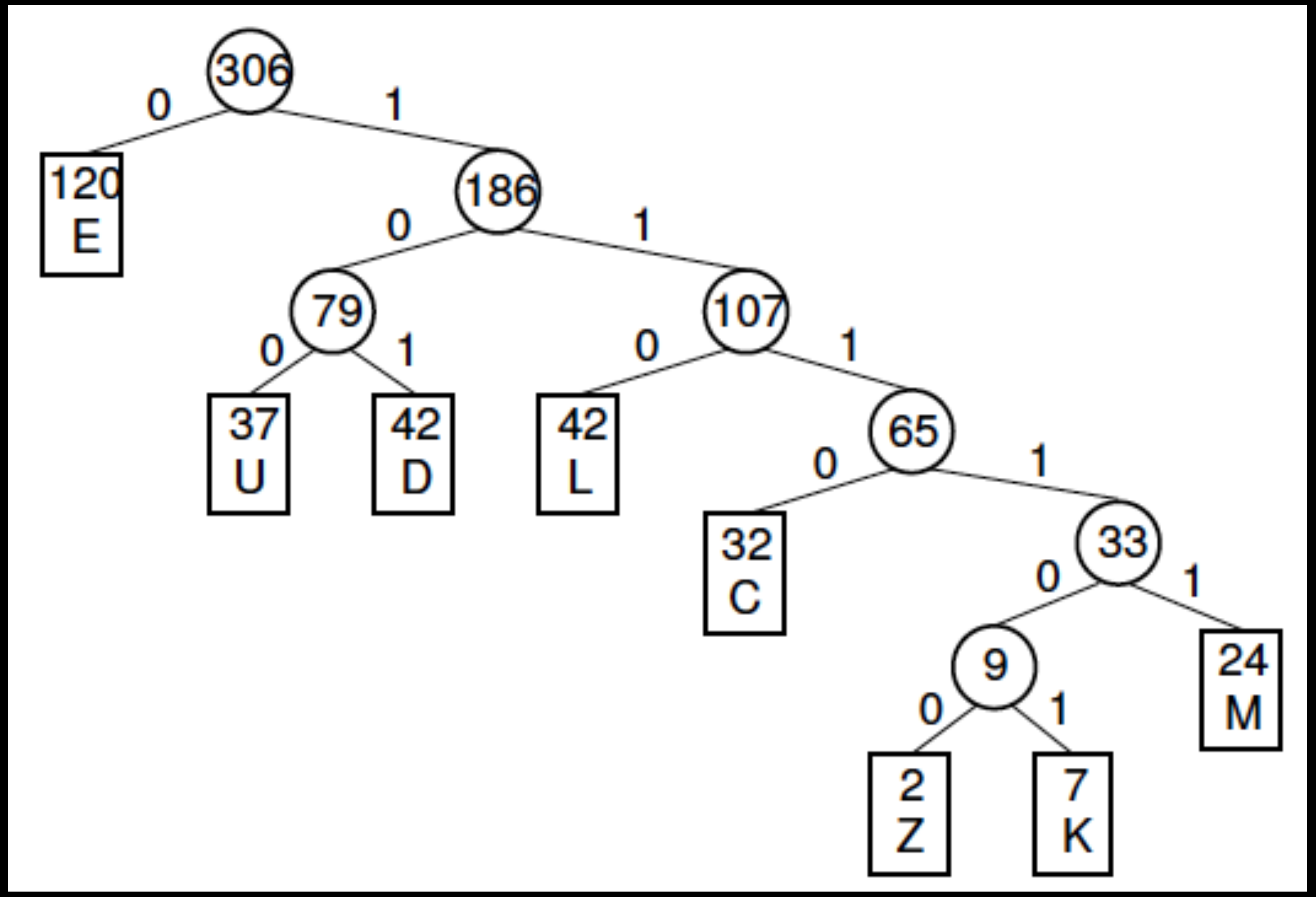
Prim

- 정점 선택(인접한 애들만)
- 이전 tree와 연관성 ->tree 계속 유지
- 정점 선택 다하면 끝

```
edge_set prim_MST_1(edge_set E, vertex s){
edge_set MST_E = { }; // 초기화
vertex_set MST_V = { s }; // 시작점 설정
loop(n-1) // n-1번 방복
(u, v) = E의 최소 가중치 간선, 단 u ∊ MST_V, v !∊ MST_V; // 같은 vertex면 안됨
MST_E = MST_E U (u, v) // 검증된 간선을 추가
MST_V = MST_V U v; // 검증된 정점을 추가
}
return MST_E;
}
```

Huffman Encoding

- 데이터 문자의 등장 빈도에 따라 다른 길이의 부호를 사용하는 알고리즘: variable-length encoding
- Full binary tree로 구현
  - 많이 사용시 짧은 코드, root쪽
  - 적게 사용시 긴 코드, bottom of the optimal



	A	B	C	D	E	F
고정 길이 코드	000	001	010	011	100	101
가변 길이 코드	1000	1001	101	00	01	11

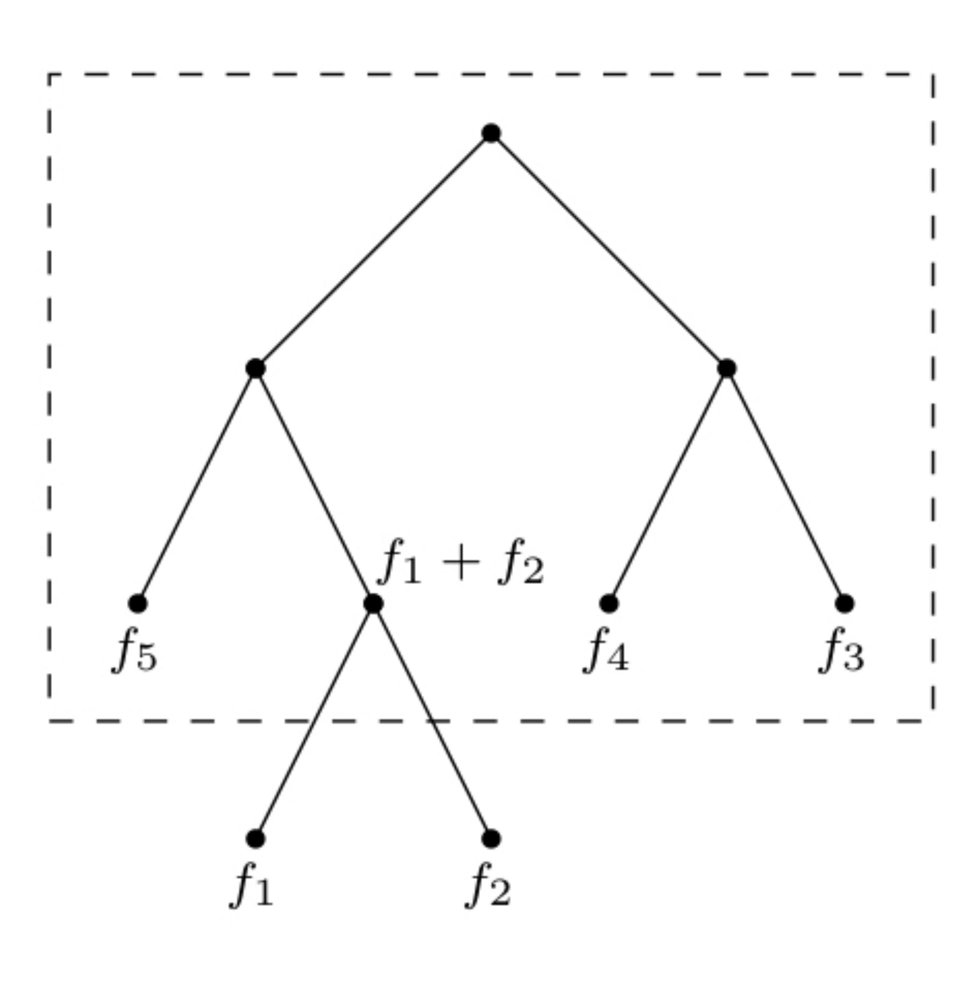
Huffman Encoding

Algorithm

cost of tree = $\sum_{i=1}^{n} f_i$. (depth of i th symbol in tree = bit 개수) => (빈도 * 길이)합이 적어야

```
procedure Huffman(f)
Input:  An array f[1···n] of frequencies
Output:  An encoding tree with n leaves

let H be a priority queue of integers, ordered by f
for i = 1 to n:  insert(H, i)
for k = n + 1 to 2n − 1:
  i = deletemin(H),  j = deletemin(H)
  create a node numbered k with children i, j
  f[k] = f[i] + f[j]
  insert(H, k)
```

1. 알파벳 별 빈도수 저장 -> Priority Queue 생성
2. Priority Queue에 symbol들 저장
3. Greedy 적용
   가장 작은 frequency(빈도) 가지는 아이들 제거(deletemin)
   둘을 합치면 (frequency 높아지니까)
   다시 priority Queue에 저장



Horn Formulas 명제 논리

- x
- $\bar{x}$: not x
- Implication, Negative clauses 가지고 variables의 관계를 찾는 방식
- Implication
  - $(z \wedge w) \Rightarrow u$: z,w hold면 u도 true
- Negative Clauses
  - $(\bar{u} \vee \bar{v} \vee \bar{y})$

For instance, suppose the formula is

$$(w \wedge y \wedge z) \Rightarrow x, \quad (x \wedge z) \Rightarrow w, \quad x \Rightarrow y, \quad \Rightarrow x, \quad (x \wedge y) \Rightarrow w, \quad (\overline{w} \vee \overline{x} \vee \overline{y}), \quad (\overline{z}).$$

Set cover Problem 집합 덮개 문제

SET COVER

Input: A set of elements B; sets $S_1, \ldots, S_m \subseteq B$

Output: A selection of the S_i whose union is B.

Cost: Number of sets picked.

- 전체집합, 그 집합의 부분집합 -> 부분집합 중 가능한 적은 집합 고르고, 그 집합들의 합집합이 원래 전체집합이 되도록