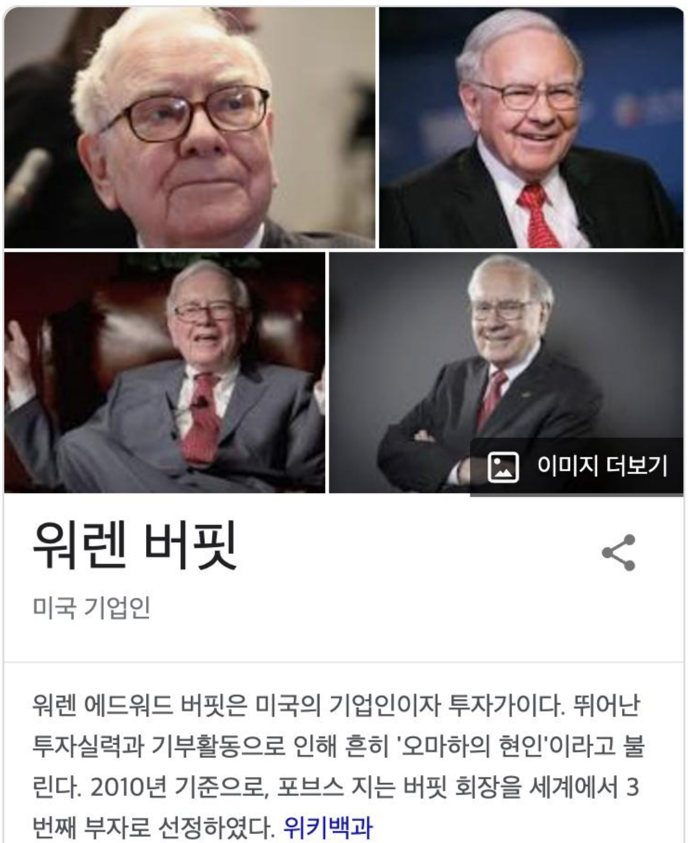
KOREATECH
한국기술교육대학교

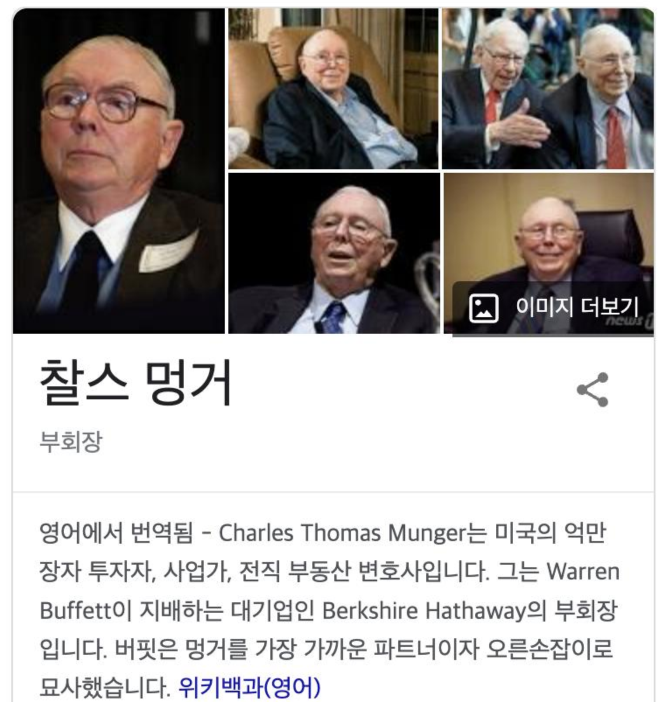
11장. 그리디 알고리즘

Youn-Hee Han
LINK@KOREATECH
http://link.koreatech.ac.kr

워런 버핏: 찰리 멍거는 유머 감각을 어디서 얻는지 들어봅시다.

찰리 멍거: 세상을 정확하게 바라보면 웃을 수 밖에 없습니다. 터무니 없습니까요.





학습 목표

◈ 그리디 알고리즘의 특징을 파악한다.

◈ 그리디 알고리즘으로 최적해가 보장되는 예와 그렇지 않은 예를 관찰한다.

◈ 매트로이드의 정의를 익힌다. (생략)

◈ 매트로이드가 만드는 문제 공간의 특성을 배운다. (생략)

01. 전형적인 그리디 알고리즘의 구조

그리디 알고리즘 소개

◈ 그리디 알고리즘 (Greedy Algorithm)

- 눈앞의 이익만 취하고 보는 알고리즘
- 현재 시점에 가장 이득이 되어 보이는 해를 선택하는 행위를 반복한다
- 대부분 최적해가 아닌 해가 산출된다.
- 최적해가 보장되는 경우가 있으며, 가능한 여러 예를 통하여 직관적으로 판단한다.

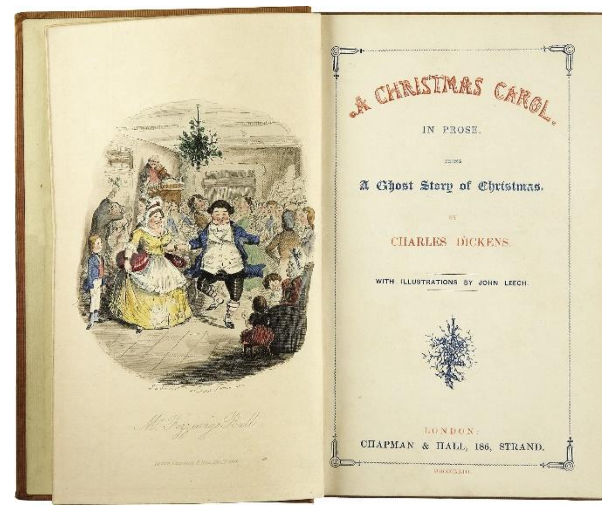
알고리즘 11-1	전형적인 그리디 알고리즘

```
do {
    가장 좋아 보이는 선택을 한다;
} until (해 구성 완료)
```

그리디 알고리즘 소개

◈ 그리디 알고리즘 (Greedy Algorithm)

- Charles Dickens' classic novel: A Christmas Carol
  - 주인공 'Scrooge'



- ➢ He is the most greedy person ever
- ➢ He never considered the past or future
- ➢ He greedily grab as much as gold as he could
- ➢ "Ghost of Christmas Past" reminded him of the past
- ➢ "Ghost of Christmas Future" warned him of the future
- ➢ Finally, he changed his greedy ways

그리디 알고리즘 소개

- ◈ 개념적으로 탐욕적 알고리즘은 동적 프로그래밍과 정반대의 전략
  - 동적 프로그래밍은 매우 엄격한 계획하에 "재귀적 속성"을 찾아서 그것을 기본으로 최종적인 (global) 해답을 Bottom-up 전략으로 찾아낸다.
- ◈ 하지만, 탐욕적 알고리즘은 매순간 선택의 과정을 거치며, 그 선택은 국부적(local)으로 최적
  - 최적이라고 생각했던 해답들을 모아서 최종적인(global) 해답을 만들었다고 해서, 그 해답이 궁극적으로 최적이라는 보장이 없다.
  - 따라서, 탐욕적인 알고리즘은 항상 최적의 해답을 산출한다는 보장이 없음

그리디 알고리즘 의사 코드

◈ 일반적인 그리디 알고리즘 의사 코드

알고리즘 11-3 | 그리디 알고리즘

```
Greedy(C)
▷ C : 원소들의 총 집합
{
    S ← ø;
    while (C≠ø and S는 아직 온전한 해가 아님) {
      ❶ x ← C에서 원소 하나 선택;
        집합 C에서 x를 제거한다;        ▷ C ← C−{x}
      ❷ if (S에 x를 더해도 됨) then S ← S ∪ {x};
    }
    if (S가 온전한 해임) then return S;
                    else return "해 없음!";
}
```

1. **선정과정**(selection procedure)
   - **현재 상태에서 가장 좋으리라고 생각되는**(greedy) **부분 해답을 선택**
2. **적정성 점검**(feasibility check)
   - **선택한 부분 해답을 최종 해답모음**(solution set) **에 포함시키는 것이 알고리즘이 풀고자 하는 목적에 비추어 적절한지를 결정한다.**
   - **적절하다면 최종 해답모음에 포함**(Union) **시킨다.**
3. **해답점검**(solution check)
   - **새로 얻은 해답모음이 풀고자 하는 문제의 최종 해답인지 결정한다.**

그리디 알고리즘 의사 코드

◈ 최소 신장 트리 산출 프림 알고리즘

– <u>대표적인 그리디 알고리즘</u>

알고리즘 11-3 | 그리디 알고리즘

```
Greedy(C)
▷ C : 원소들의 총 집합
{
    S ← ø;
    while (C ≠ ø and S는 아직 온전한 해가 아님) {
      ❶ x ← C에서 원소 하나 선택;
        집합 C에서 x를 제거한다;      ▷ C ← C-{x}
      ❷ if (S에 x를 더해도 됨) then S ← S ∪ {x};
    }
    if (S가 온전한 해임) then return S;
                else return "해 없음!";
}
```

선정과정(selection procedure)

적정성 점검(feasibility check)

해답점검(solution check)

알고리즘 11-2 | 프림 알고리즘

```
Prim(G, r)
▷ G=(V, E) : 그래프, r : 시작 정점
{
    S ← ø; T ← ø;      ▷ S : 정점 집합, T : 간선 집합
    정점 r을 집합 S에 더한다;
    while (S ≠ V) {
        S에서 V-S를 연결하는 간선들 중 최소 길이의 간선 (x, y)를 찾는다;
                                                  ▷ (x ∈ S, y ∈ V-S)
        T에 간선 (x, y)를 더한다;
        정점 y를 집합 S에 더한다;
    }
}
```

해답점검(solution check)

선정과정(selection procedure)
적정성 점검(feasibility check)

02. 그리디 알고리즘으로 최적해가 보장되지 않는 예

이진 트리의 최적합 경로 찾기

◈ 이진 트리의 최적합 경로 찾기 문제 정의

– 이진 트리 내 각 노드는 양의 가중치를 지님

– 트리 내 각 노드의 가중치들은 사전에 알지 못함
  - 임의의 노드에 도착하면 해당 노드의 가중치 뿐만 아니라 그 노드의 두 개 자식에 할당된 가중치가 공개

– 루트에서 시작하여 왼쪽으로 분기할 지 오른쪽으로 분기할 지 매 단계마다 결정
  - 리프 노드를 만나면 단계를 종료

– 이 과정에서 만난 노드에 있는 가중치 합이 이 경로의 점수가 되며, 이 합을 최대화 하는 문제

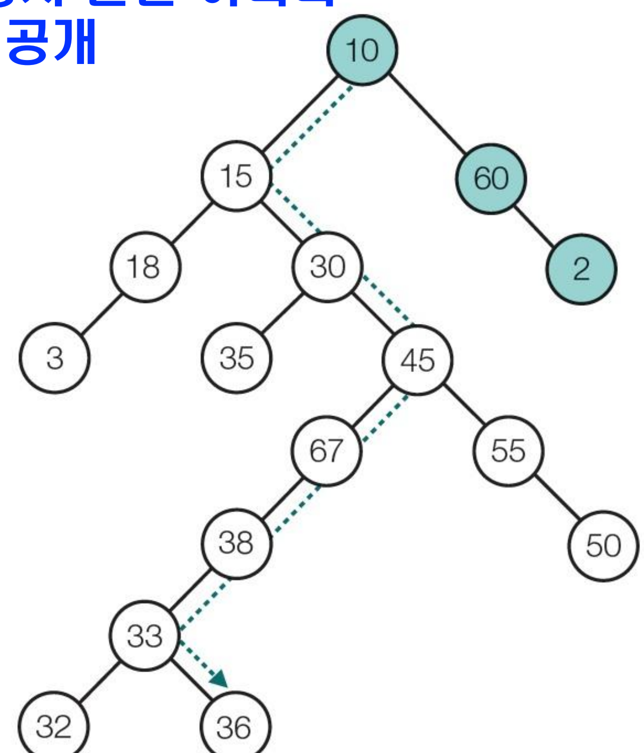


그림 11-1 그리디 알고리즘이 실패하는 예

이진 트리의 최적합 경로 찾기

◈ 이진 트리의 최적합 경로 찾기 문제의 그리디 알고리즘

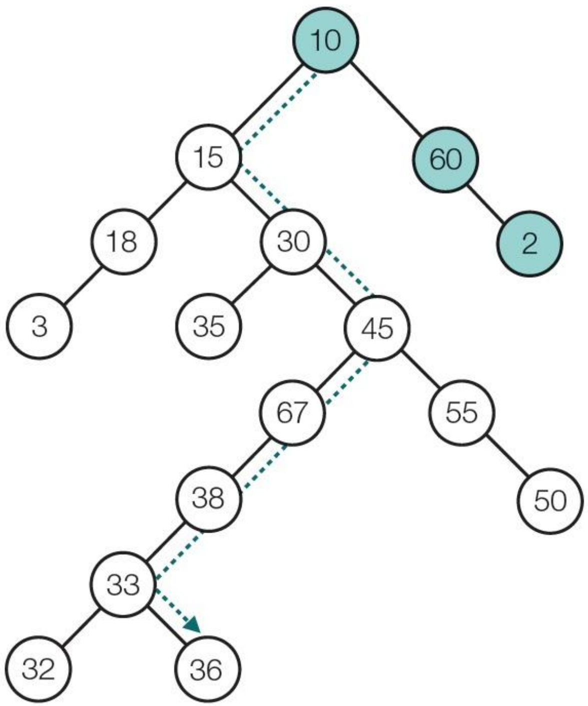


그림 11-1 그리디 알고리즘이 실패하는 예

알고리즘 11-4 | **이진 트리의 그리디 탐색**

```
Greedy_Sum(T, w[], r)
▷ T : 이진 트리, r : 루트 노드, w[] : 노드들에 할당된 수
{
    x ← r; sum ← 0;
    while (x가 리프 노드가 아님) {
        sum ← sum + w[x];
        x ← x의 자식 중 가중치가 큰 자식;
    }
    sum ← sum + w[x];
    return sum;
}
```

해답점검(solution check)

선정과정(selection procedure)

No: 적정성 점검(feasibility check)

– 눈 앞의 이익만 쫒는 알고리즘
– DFS나 BFS와 같은 방식으로 모든 노드를 방문하면서, 모든 경로 각각에 대한 가중치 합을 확인해야 최적해를 얻어낼 수 있음

보따리 문제 (Knapsack Problem)

◈ 보따리 문제 정의

- 어떤 도둑이 보석상에서 보따리(Knapsack)을 매고 침입했다.
  - 각 보석은 $item_i$라고 지칭
  - 보석상에는 총 n개의 보석이 있음 ($1 \leq i \leq n$)
  - $S = \{item_1, item_2, \ldots, item_n\}$
- 훔칠 보석의 총 무게(부피)가 용량 M을 초과하면 보따리가 망가진다.
- 도둑이 똑똑하여 각 보석의 "무게(W_i)"와 "값어치(P_i)"를 알고 있다.
- 도둑은 총 무게가 M을 초과하지 않으면서 보석들의 총 값어치가 최대가 되도록 보석을 보따리에 담고자 한다.

보따리 문제 (Knapsack Problem)

◈ (형식적인) 보따리 문제 정의

- 각 보석은 $item_i$라고 지칭하며, 총 n개의 보석이 있음 ($1 \le i \le n$)
  - $S = \{item_1, item_2, \ldots, item_n\}$
- 훔칠 보석의 총 무게(부피)가 용량 M을 초과할 수 없음
- 각 $item_i$의 "무게(W_i)"와 "값어치(P_i)"가 주어짐
- $\sum_{item_i \in A} W_i \le M$를 만족하면서 $\sum_{item_i \in A} P_i$ 가 최대가 되는 $A(\subset S)$를 결정하는 문제

◈ 보따리 문제 종류

- 0/1 보따리 문제 (0/1 Knapsack Problem)
  - 보석을 자를 수 없음
- 자를 수 있는 보따리 문제 (Fractional Knapsack Problem)
  - 보석을 자를 수 있음

0/1 보따리 문제 (0/1 Knapsack Problem)

◈ Brute-Force 알고리즘

- n개의 $item_i$에 대해서 모든 부분집합을 다 고려한다.
- 모든 부분집합들 중에서 총 무게가 M을 초과하는 부분집합들을 버리고, 남은 것들 중에서 총 이익이 최대가 되는 것을 하나 선택한다.
- 그러나, 불행하게도 크기가 n인 집합의 부분집합의 수는 2^n개이다.
- 수행 시간 복잡도: $\Theta(2^n)$

$n=4$

$$2 \times 2 \times 2 \times 2 \equiv 2^4 \equiv 16$$

0/1 보따리 문제 (0/1 Knapsack Problem)

◈ 탐욕적 방법 1

- [선택전략] **가장 비싼 물건부터 우선적**으로 채운다.
- 애석하게도 이 알고리즘은 최적이 아니다!
- 왜 아닌가? 보기: $M = 30\text{kg}$ →

품목	무게	값
item$_1$	25kg	10 만원
item$_2$	10kg	9 만원
item$_3$	10kg	9 만원

- **탐욕적인 방법**: $item_1 \Rightarrow 25\text{kg} \Rightarrow 10$만원
- **최적인 해답**: $item_2 + item_3 \Rightarrow 20\text{kg} \Rightarrow 18$만원

0/1 보따리 문제 (0/1 Knapsack Problem)

◈ 탐욕적 방법 2

- [선택전략] **가장 가벼운 물건부터 우선적**으로 채운다.
- 마찬가지로 이 알고리즘도 최적이 아니다!
- 왜 아닌가? 보기: $M = 30\text{kg}$ →

품목	무게	값
$item_1$	25kg	20 만원
$item_2$	10kg	9 만원
$item_3$	10kg	5 만원

- **탐욕적인 방법**: $item_2 + item_3 \Rightarrow 20\text{kg} \Rightarrow 14$만원
- **최적인 해답**: $item_1 \Rightarrow 25\text{kg} \Rightarrow 20$만원

0/1 보따리 문제 (0/1 Knapsack Problem)

◈ 탐욕적 방법 3

- [선택전략] **무게 당 가치가 가장 높은 물건부터 우선적**으로 채운다.
- 가장 합리적인 방법
- 그래도 최적이 아니다!
- 왜 아닌가? 보기: $M = 30\text{kg}$ →

품목	무게	값	값어치
$item_1$	5kg	50 만원	10 만원/kg
$item_2$	10kg	60 만원	6 만원/kg
$item_3$	20kg	140 만원	7 만원/kg

- 탐욕적인 방법: $item_1 + item_3 \Rightarrow 25\text{kg} \Rightarrow 190$만원
- 최적인 해답: $item_2 + item_3 \Rightarrow 30\text{kg} \Rightarrow 200$만원

- <u>더 복잡한 탐욕적 방법을 쓰더라도, 0-1 배낭 채우기 문제는 풀리지 않는다.</u>

자를 수 있는 보따리 문제 (Fractional Knapsack Problem)

◈ 탐욕적 방법 알고리즘 의사 코드

알고리즘 11-5 보따리 문제를 위한 그리디 알고리즘

```
Greedy_Knapsack(p[], W[], M)
▷ P[] : 가치 배열, W[] : 부피 배열
▷ X : 보따리에 담는 물건 집합, M : 보따리 부피
{
    ▷ P와 W를 P[i]/W[i]에 따라 내림차순(단위 부피당 가치가 큰 순서)으로 정렬한다.
    beadRoom ← M; i ← 1;
    while (W[i] ≤ beadRoom and i≤n) {
        X ← X ∪ {i};
        beadRoom ← beadRoom − W[i];
        i++;
    }
    return X;
}
```

beadRoom: 보석을 담아 낼 수 있는 남은 무게

– 최적해는 아니지만 최적해와 근사한 해 산출

0/1 보따리 문제 (0/1 Knapsack Problem)

◈ Dynamic Programming 방법

– 최적의 답을 구하도록 알고리즘 설계 가능

– 수행 시간 복잡도: $\Theta(2^n)$

◈ [중요!!!]

– 아직 아무도 이 문제의 최악의 경우 수행시간 측면에서 지수(exponential) 시간 복잡도보다 좋은 알고리즘을 만들지 못했다.

– 또한, 아직 아무도 그러한 알고리즘은 없다라고 증명한 사람도 없다.

– 대표적인 NP-Hard 문제!

자를 수 있는 보따리 문제 (Fractional Knapsack Problem)

◈ 물건의 일부분을 잘라서 담을 수 있다.

◈ 탐욕적 방법 3 사용하면 최적해 산출 가능

- [선택전략] **무게 당 가치가 가장 높은 물건부터 우선적**으로 채운다.
- 보기: $M = 30\text{kg}$ →

품목	무게	값	값어치
$item_1$	5kg	50 만원	10 만원/kg
$item_2$	10kg	60 만원	6 만원/kg
$item_3$	20kg	140 만원	7 만원/kg

- $item_1 + item_3 + item_2 \text{ x } 1/2 \Rightarrow 30\text{kg} \Rightarrow 220$만원
- 최적이다!

동전 바꾸기

◈[쉬어가기] 미국과 한국의 동전 (Coins) 시스템


One dollar ($1) – Andrew Johnson


Half dollar ($0.50) – John F. Kennedy


Quarter ($0.25) – George Washington


Dime ($0.10) – Franklin D. Roosevelt


Nickel ($0.05) – Thomas Jefferson


Penny or Cent ($0.01) – Abraham Lincoln


오백원 (₩500)– 두루미


백원 (₩100) – 이순신


오십원 (₩50) – 쌀


십원 (₩10) – 다보탑


오원 (₩5) – 거북선

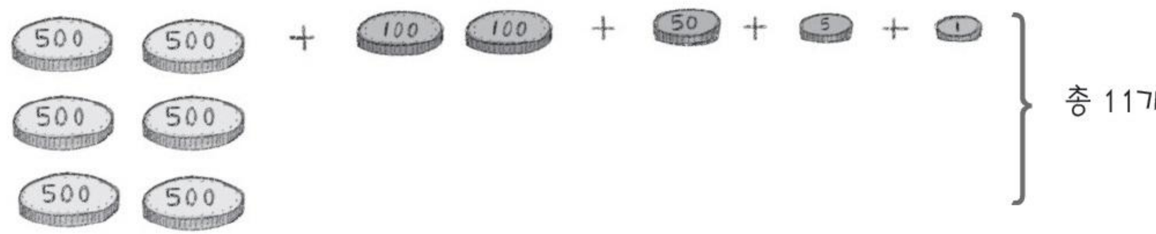

일원 (₩1) – 무궁화

동전 바꾸기

◈ 동전 바꾸기 문제

– 가지고 있는 동전 중에서 거스름 돈을 줄 때에 동전의 개수가 최소가 되도록 거스름 돈을 주는 문제

$x = 3{,}256$원



◈ 그리디 알고리즘

– 거슬러 주어야 하는 총액수를 x라 하자.

– 먼저, 가치가 가장 높은 (액면가가 높은) 동전부터 x가 초과되지 않도록 계속 내준다.

: 각 스텝마다, 가장 높은 액면가를 지닌 동전을 꺼낸다.

: 만약 해당 동전을 꺼냄으로써 전체 거스름돈이 x를 초과하면 도로 집어넣는다.

– 이 과정을 총액이 정확히 x가 될 때까지 계속한다.

동전 바꾸기

◈ 동전 시스템과 그리디 알고리즘의 최적성

- 미국이나 한국에서 유통되고 있는 동전 시스템하에서 그리디 알고리즘을 적용하여 거스름돈을 주면, 동전의 개수는 항상 최소가 됨
  - 동전의 액면이 커지면서 바로 아래 동전이 지닌 액면의 배수가 되는 동전 시스템 → 최적성 보장됨!!!

◈ 동전 시스템이 변경 → 그리디 알고리즘은 최적해 보장 못함

- 예를 들어, 500원, 400원, 100원, 75원, 50원 동전 시스템
- $x = 1{,}300$원
  - 그리디 알고리즘 해: 500원 2개, 100원 3개 → 총 5개
  - 최적해: 500원 1개, 400원 2개 → 총 3개
- 동적 프로그래밍 전략으로 최적해 산출 가능

03. 그리디 알고리즘으로 최적해가 보장되는 예

최소 신장 트리

◈ 프림 알고리즘 & 크루스칼 알고리즘

알고리즘 10-3 | 프림 알고리즘(버전 1)

```
Prim(G, r)
▷ G=(V, E): 주어진 그래프
▷ r: 시작 정점
{
      S←ø;        ▷ S: 정점 집합
   ❶ for each u∈V
            d[u]←∞;
      d[r]←0;
   ❷ while (S≠V) {                  ▷ n회 순환
        ❸ u←extractMin(V−S, d);
          S←S∪{u};
        ❹ for each v∈L(u)       ▷ L(u): 정점 u의 인접 정점 집합
             ❺ if (v∈V−S and w(u, v) < d[v]) then {
                   ❻ d[v]←w(u, v);
                   ❼ tree[v]←u;
                }
      }
}

extractMin(Q, d[])
{
      집합 Q에서 d값이 가장 작은 정점 u를 리턴한다;
}
```

알고리즘 10-5 | 크루스칼 알고리즘

```
Kruskal(G)
{
      T←ø;        ▷ T: 신장 트리
   ❶ 단 하나의 정점만으로 구성된 n개의 집합을 초기화한다;
   ❷ 모든 간선을 가중치의 크기 순으로 정렬하여 배열 A[1...E]에 저장한다;
   ❸ while (T의 간선 수 < n−1) {
        ❹ A에서 최소 비용의 간선 (u, v)를 제거한다;
        ❺ if (정점 u와 v가 다른 집합에 속함) then {
             ❻ T←T∪{(u, v)};
             ❼ 정점 u와 v가 속한 두 집합을 하나로 합친다;
          }
      }
}
```

뒤도 보지 않고 앞도 보지 않는 탐욕적인 선택

→ 최적해 산출!

회의실 배정 문제

◈ 회의실 배정 문제 정의

- 회사에 회의실이 1개 있음
- 회의실을 사용하고자 하는 부서는 원하는 회의 시작 시간과 종료 시간을 명시해서 신청서를 제출
  - n: 신청 회의 수
  - $S = \{(s_i, t_i) | i = 1, 2, \ldots, n\}$
    - s_i: 회의 i의 시작 시간
    - t_i: 회의 i의 종료 시간
- 이렇게 받은 n개의 회의 신청에 대해 겹치는 회의가 없게 하면서 가장 많은 수의 회의를 소화할 수 있도록 회의실 사용 스케줄을 정하는 문제

회의실 배정 문제

◈ 회의실 배정 문제의 그리디 알고리즘

알고리즘 11-7 회의실 배정을 위한 그리디 알고리즘

```
Greedy_Schedule(S, n)
▷ S = {(s_i, t_i) | i = 1, 2, ⋯, n}, n : 신청 회의 수
▷ s_i : 시작 시간, t_i : 종료 시간
{
    t_i에 대한 오름차순으로 정렬하고, 이 순서대로 S = {(s_i, t_i) | i = 1, 2, ⋯, n}의
        번호를 다시 매긴다;      ▷ 즉, 종료 시간이 가장 이른 회의가 (s_1, t_1)이 된다.
    T ← {1};
    last ← 1;
    for (i ← 2, i ≤ n, i++)
        if (t_last ≤ s_i) {
            T ← T ∪ {i};
            last ← i;
        }
    return T;
}
```

1, 2, 3, 4, 5, 6, 7, 8

[3, 5], [1, 6], [6, 7], [5, 9], [8, 13], [7, 14], [12, 18], [16, 20]

last (1), last (3), last (5), last (8)

1, 2, 3, 4, 5, 6, 7, 8

T={ [3, 5], ~~[1, 6]~~, [6, 7], ~~[5, 9]~~, [8, 13], ~~[7, 14]~~, ~~[12, 18]~~, [16, 20] }

04. 메트로이드: 그리디 알고리즘으로 최적해가 보장되는 공간 구조 (Skip)

→ 설계한 그리디 전략이 최적해를 산출하는지 증명하는 방법

허프만 코딩 알고리즘

허프만 코드

[NOTE] 교재에서 다루지 않지만 반드시 알아야 할 내용

◈ Representation of characters in computer
- 7 bits/char (ASCII)
- 2 bytes/char (KSC Hangul)
- 2 bytes/char (UNICODE)
- Isn't there more efficient way to store text?

◈ Huffman Code: 문자들로 이루어진 데이터 파일 크기를 작게 만들기 위해 문자 각각을 코드화 하는 방법 중 하나
- Data compression based on frequency → Huffman Code
  - Binary coding (0과 1만 사용, 즉 이진 코딩)
  - Variable length coding (가변 길이 이진 코딩)
  - Prefix coding (전치 코딩)
  - 더 자주 출현하는 문자에 대하여 더 짧은 코드 할당

허프만 코드

◈ 고정 길이 이진 코딩 vs. 가변 길이 이진 코딩

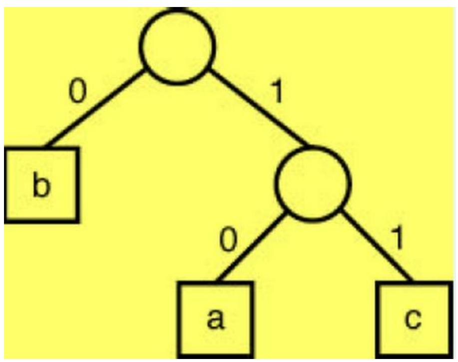
- 텍스트 파일의 문자 집합: {a, b, c}
- 고정길이 이진 코딩
  - a: 00, b: 01, c: 11
  - ababcbbbc ➔ 000100011101010111 (18 비트)
- 가변길이 이진 코딩
  - b가 가장 빈번하게 나온다는 것을 안다고 가정 ➔ b를 0으로 코딩
  - a: 10, b:0, c:11
  - ababcbbbc ➔ 1001001100011 (13 비트)

◈ 최적 이진 코딩 문제(Optimal Binary Code)

- 주어진 텍스트 파일에 있는 문자들을 이진 코드로 표현하기 위해 필요한 비트의 개수가 최소가 되는 이진 문자 코드를 찾는 문제

허프만 코드

- ◈ Huffman code는 <u>prefix code (전치 코드)</u>
  - 어떤 문자 코드도 다른 코드의 prefix가 되지 않는다.
    - 즉, 한 문자의 코드가 다른 문자의 코드의 앞부분이 될 수 없다.
    - {b, a, c}={0, 10, 11} : prefix code
    - {a, b}={01, 011} : non-prefix code



  - 가변 길이 코딩
  - Prefix-free code 라고도 불리운다.
  - 전치 코드의 장점
    - 인코딩되어 있는 것을 디코딩할 때 1-pass 에 디코딩 가능
    - 디코딩시(복원시)에 모호성 제거
    - 전치 코드는 <u>Leaf가 문자로 구성된 이진 트리 (Binary Tree)</u>로 표현 가능

허프만 코드

◈ Huffman code의 예

– 문자 집합 {a, b, c, d, e, f}에 대한 3가지 코드법

문자	빈도수	C1(고정 길이)	C2	C3(허프만)
a	16	000	10	00
b	5	001	11110	1110
c	12	010	1110	110
d	17	011	110	01
e	10	100	11111	1111
f	25	101	0	10

– 각 코드법을 사용한 결과 인코딩 파일의 총 비트 수

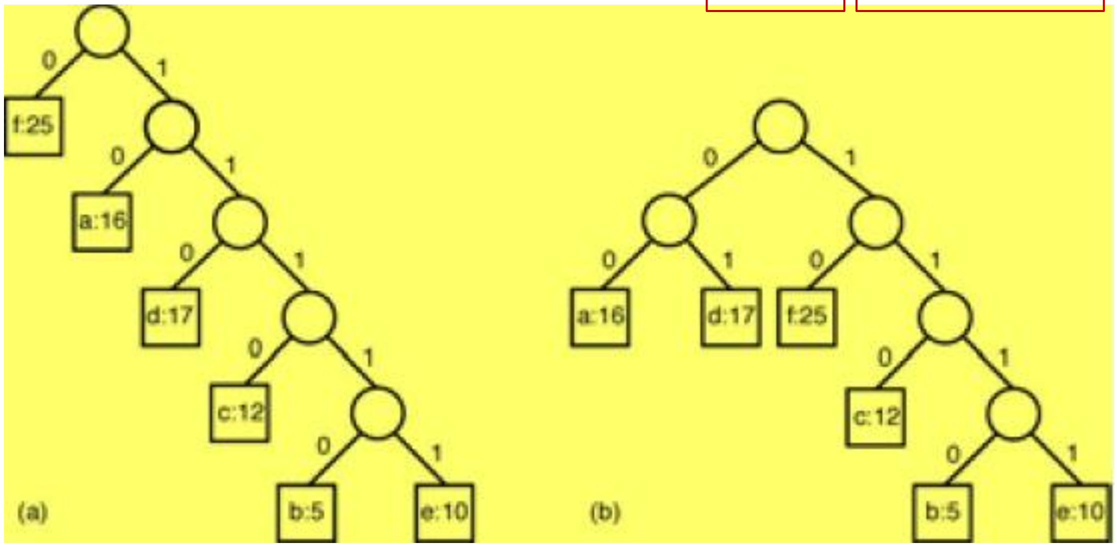
- C1: 16 * 3 + 5 * 3 + 12 * 3 + 17 * 3 + 10 * 3 + 25 * 3 = 255
- C2: 16 * 2 + 5 * 5 + 12 * 4 + 17 * 3 + 10 * 5 + 25 * 1 = 231
- C3: 16 * 2 + 5 * 4 + 12 * 3 + 17 * 2 + 10 * 4 + 25 * 2 = 212
- C3 방법이 가장 효율이 좋다.

허프만 코딩 문제

◈ 허프만 코딩 문제

– 즉, 주어진 문자 집합에 대해 최적 코드에 해당하는 이진 트리를 구축하여 최적 이진 문자 코드(Huffman code)를 만드는 문제

문자	빈도수	C1(고정 길이)	C2	C3(허프만)
a	16	000	10	00
b	5	001	11110	1110
c	12	010	1110	110
d	17	011	110	01
e	10	100	11111	1111
f	25	101	0	10



허프만 코딩 문제

◈ 허프만 코딩 문제 알고리즘

- 자료구조 Priority Queue 이용
  - Top Priority: 빈도수가 가장 작은 문자 → 내부적으로는 Heap 으로 구현
- 노드 타입 선언

```
class nodetype
{
    char symbol;          // 문자값
    int frequency;        // 파일에 있는 문자의 빈도수
    nodetype left;
    nodetype right;
}
```

- 초기화 1: nodetype 객체들을 가리키는 n개의 노드 포인터 p에 대해 오른쪽과 같이 초기화

```
p.symbol = 파일에서 별개의 문자;
p.frequency = 파일에서 그 문자의 빈도수;
p.left = p.right = NULL;
```

- 초기화 2: n개의 nodetype 객체들을 낮은 빈도수부터 오름차순으로 우선순위 큐 (Priority Queue) 에 삽입

허프만 코딩 문제

◈ 허프만 코딩 문제 알고리즘

- 알고리즘 의사 코드
  - 1) Priority Queue 에서 차례로 두 개를 선택하여
  - 2) 선택된 각각을 left와 right child에 배치하는 새로운 (부모) 노드를 생성하여 binary subtree 생성
  - 3) 새로운 노드의 빈도수로서 선택된 두 빈도수의 합을 지정
  - 4) 그 새로운 노드를 Priority Queue 에 삽입
  - 더 이상 진행할 수 없을 때까지 위 1), 2), 3), 4)과정 반복
    - ➢ 더 이상 진행할 수 없음 == 선택할 노드가 한 개 밖에 없음
    - ➢ 1), 2), 3) 과정을 n-1번 수행하면 더 이상 진행 안됨
- Priority Queue에 남아 있는 하나의 노드 (트리) 자체가 허프만 코드를 나타냄

허프만 코딩 문제

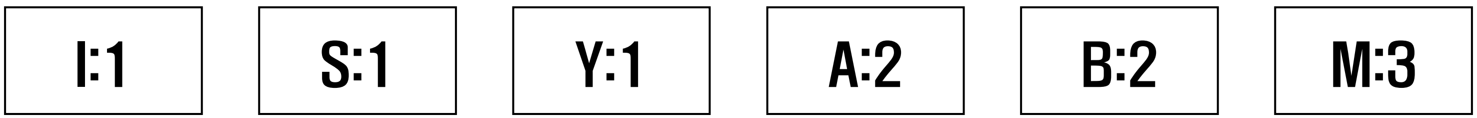
◈ 허프만 코딩 문제 알고리즘

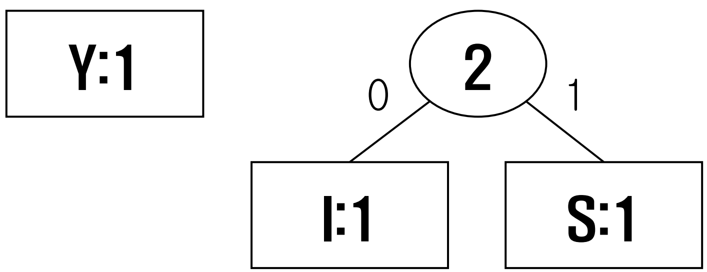
– [의사 코드]

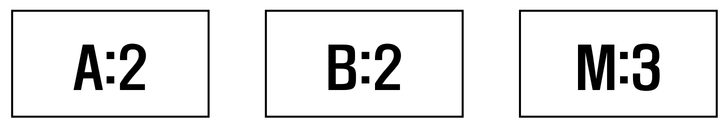
```
for ( i=1; i < = n-1; i++) {      // 해답 여부의 점검이 없다.
    remove(PQ, p);                // 대신 i = n - 1일 때 해답을 얻는다.
    remove(PQ, q);                // 선택절차
    r = new nodetype();           // 적정성 점검은 없다.
    r.left = p;
    r.right = q;
    r.frequency = p.frequency + q.frequency;
    insert(PQ, r);
}
remove(PQ, r);
return r;
```

허프만 코딩 문제

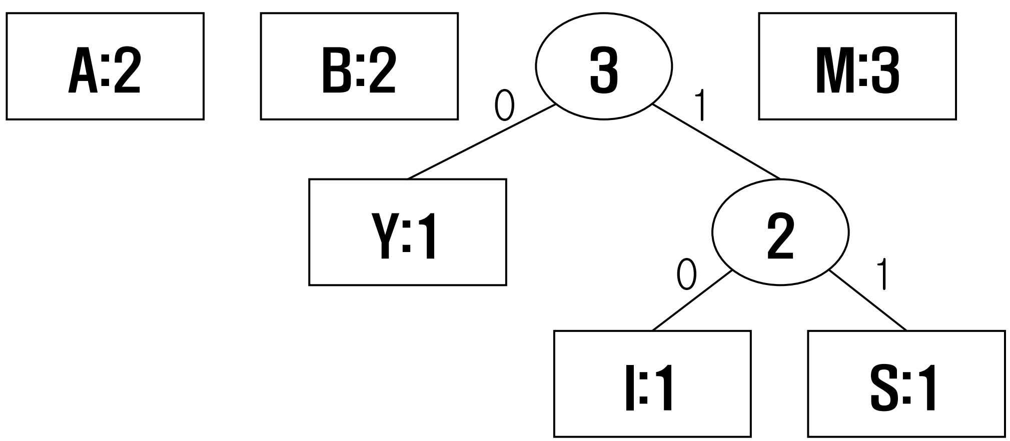
◈ 허프만 코딩 문제 알고리즘 수행 예 – I (1/3)





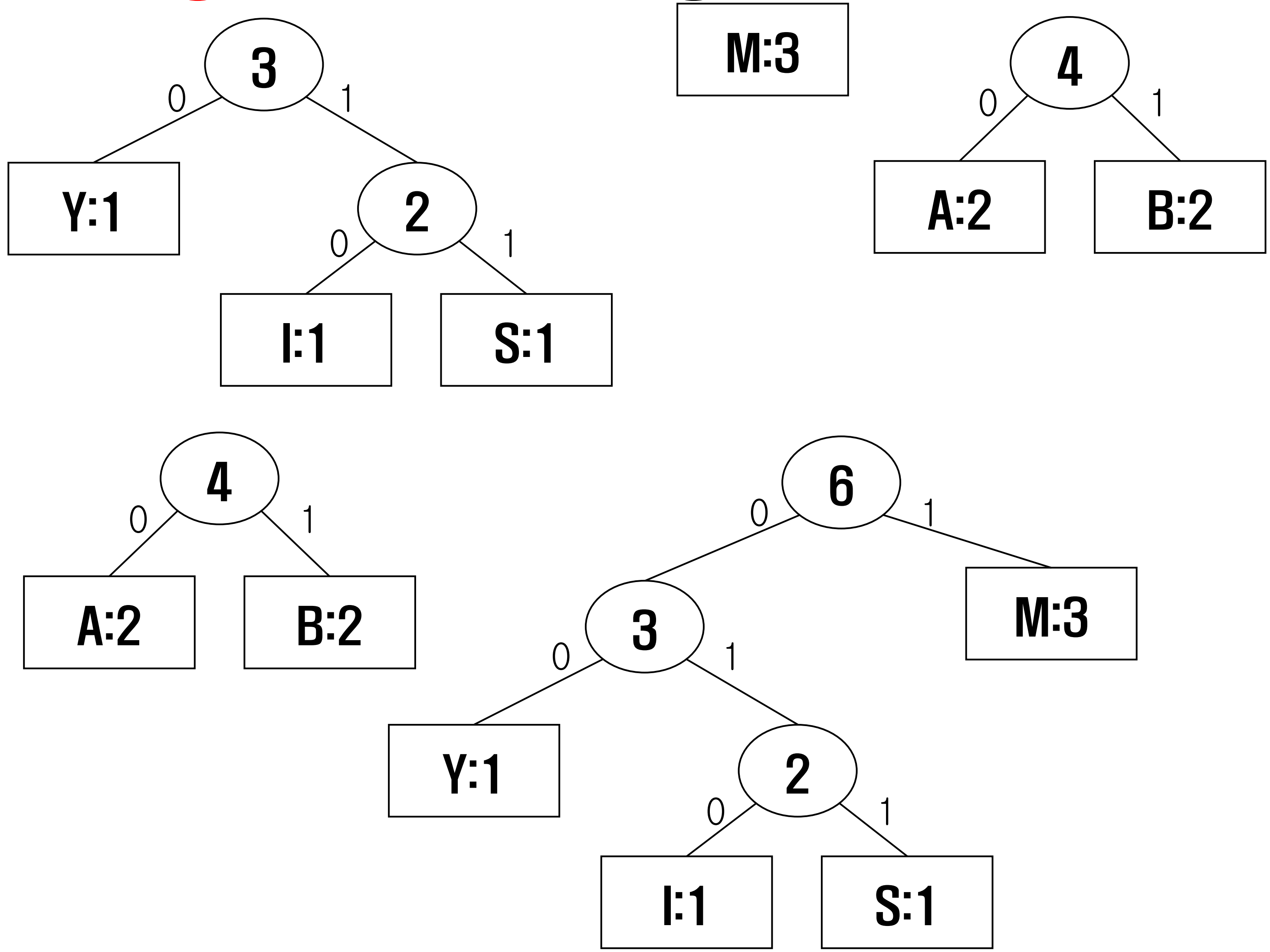


- <u>1st 우선순위: 더 낮은 빈도수의 노드</u>
- 2nd 우선순위: 새롭게 구성된 노드
- 3rd 우선순위: 더 낮은 알파벳의 노드



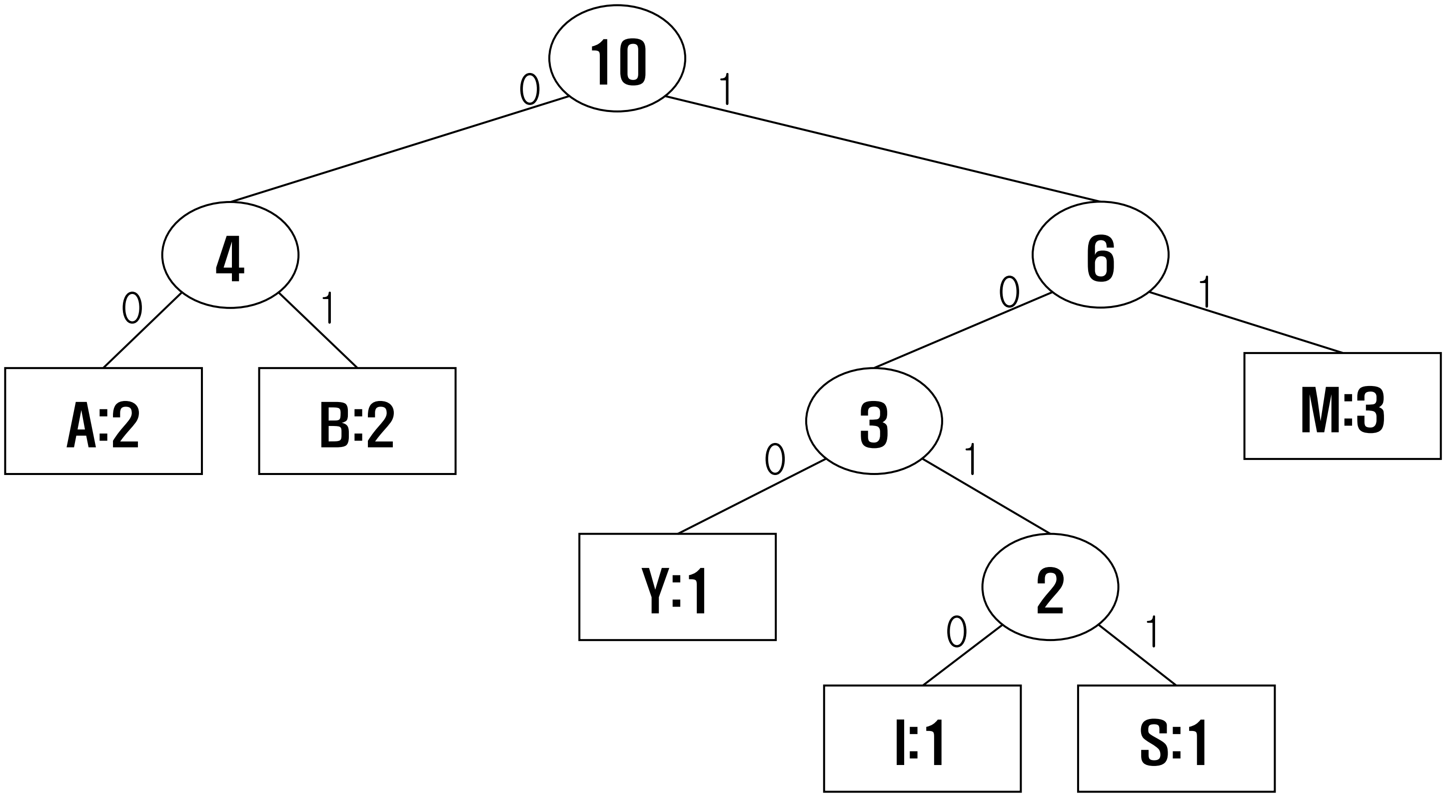
허프만 코딩 문제

◈ 허프만 코딩 문제 알고리즘 수행 예 – I (2/3)



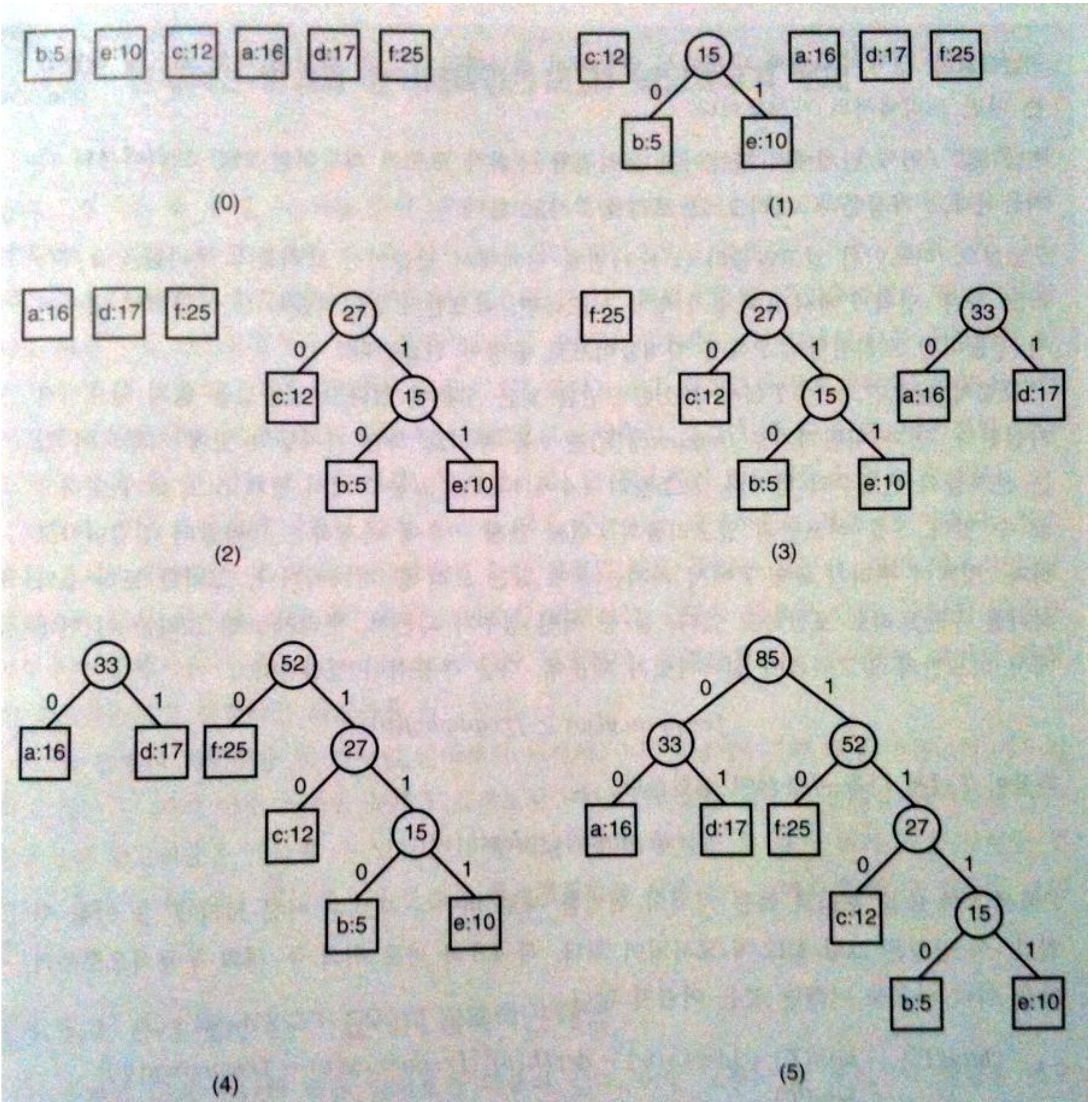
허프만 코딩 문제

◈ 허프만 코딩 문제 알고리즘 수행 예 – I (3/3)



허프만 코딩 문제

◈ 허프만 코딩 문제 알고리즘 수행 예 – II

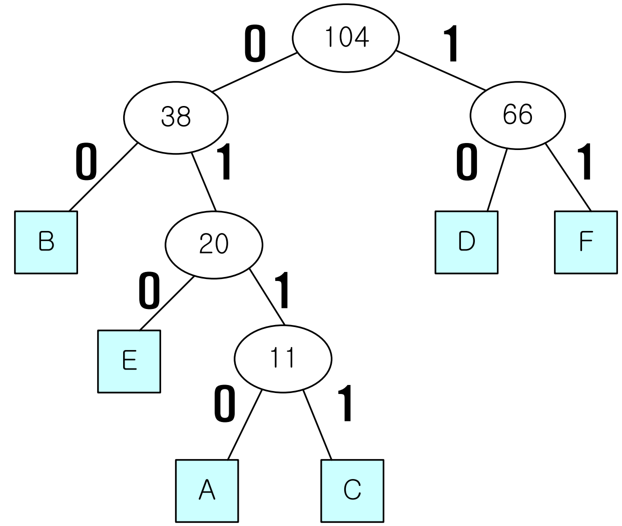


허프만 코딩 문제

- 1st 우선순위: 더 낮은 빈도수의 노드
- 2nd 우선순위: 새롭게 구성된 노드
- 3rd 우선순위: 더 낮은 알파벳의 노드

◈ 허프만 코딩 문제 알고리즘 수행 예 – III

문 자	빈 도
A	2
B	18
C	9
D	30
E	9
F	36



A = 0110
B = 00
C = 0111
D = 10
E = 010
F = 11

문 자	빈 도	원래(ASCII) 크기	압축된 크기	차이
A	2	7*2=14	4*2=8	6
B	18	7*18=126	2*18=36	90
C	9	7*9=63	4*9=36	27
D	30	7*30=210	2*30=60	150
E	9	7*9=63	3*9=27	36
F	36	7*36=252	2*36=72	180
계	104	728	240	488

Questions & Answers