# 1장

1. 계승, 피보나치수, 수열의 점화식, 하노이 타워, 병합정렬 등
2.
   ① $O$, $\Theta$
   ② $\Omega$, $\Theta$
3.
   ① $O$, $\Theta$, $o$
   ② $\Omega$, $\Theta$, $\omega$
4.
   ① a, b, c, d
   ② a, b, c, d
   ③ b, d, e, f
   ④ b, d
   ⑤ b, d, e, f
   ⑥ b, e
   ⑦ b, e

5. 병합정렬은 시작 초기에 자신과 똑같은 성격이지만 크기가 반인 두 개의 문제를 해결한다. 이후 이 두 문제를 병합함으로써 전체 문제가 해결된다.
6.

Claim 1: $2n^2-10n+3=O(n^2)$

<증명> 여러 가지 선택이 가능하나 $c=2$로 잡으면,

$$2n^2-10n+3 \le 2n^2$$
$$\frac{3}{10} \le n$$

$n_0=1$, $c=2$로 잡으면 $n_0 \le n$인 모든 $n$에 대하여 $2n^2-10n+3 \le cn^2$이다.

Claim 2: $2n^2-10n+3=\Omega(n^2)$

<증명> 여러 가지 선택이 가능하나 $c=9$로 잡으면,

$$2n^2-10n+3 \ge 9n$$
$$0 \ge 13n^2+10n-3$$
$$\frac{3}{13} \le n$$

$n_0=1$, $c=9$로 잡으면 $n_0 \le n$인 모든 $n$에 대하여 $2n^2-10n+3 \ge cn^2$이다.

위 Claim 1, 2로부터 $2n^2-10n+3=\Theta(n^2)$이다. ■

7. $n^2$에 비례한다. 점근적으로는 $\Theta(n^2)$이다.
8. $n^3$에 비례한다. 점근적으로는 $\Theta(n^3)$이다.

## 2장

1. 가정해도 된다. 어떠한 $n$이라도 $n$과 $5n$ 사이에 $5^k$이 되는 수가 하나 있다. 즉, $n \le 5^k < 5n$인 $5^k$이 하나 존재한다. 만일 $T(n)=O(n^r)$이라면 $T(5n)=O((5n)^r)=O(5^r n^r)=O(n^r)$이다. $T(n) \le T(5^r) \le T(5n)$이므로 $n=5^k$으로 잡아도 점근적 복잡도에는 영향을 미치지 않는다.
2.

```
search(A, p, r, x)
▷ 배열 A[p ... r]에서 원소 x가 있는지 체크한다.
{
    if (p ≤ r) then {
            q ← (p+r)/2;
            if (x < A[k]) then return search(A, p, q-1, x);   ▷ 왼쪽 그룹으로
            else if (x = A[k]) then return A[q]; ▷ 중간 원소가 바로 찾는 원소임
            else return search(A, q+1, r, x);                 ▷ 오른쪽 그룹으로
    } else
            return FALSE;
}
```

$T(n) = T(\frac{n}{2}) + a$, $a$는 상수

Claim: $T(n)=O(\log n)$, 즉, 충분히 큰 $n$에 대하여 $T(n) \le c\log n$인 상수 $c$가 존재한다.

<증명>

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + a \\ &\le c\log\frac{n}{2} + a \\ &= c\log n - c\log 2 + a \\ &\le c\log n \end{aligned}$$

$c \ge \frac{a}{\log 2}$인 $c$를 잡으면 된다. ■

3.

① 마스터 정리를 사용한다. $h(n) = n^{log_3 3} = n$, $f(n) = \sqrt{n}$, $h(n)$이 $f(n)$을 다항식의 비율로 압도하므로 $T(n) = \Theta(h(n)) = \Theta(n)$이다.

② 마스터 정리를 사용한다. $h(n) = n^{log_2 1} = 1$, $f(n) = n$, $h(n)$가 $f(n)$에 다항식의 비율로 압도되고, $af(\frac{n}{b}) = \frac{n}{2} \leq \frac{1}{2}f(n)$이므로 $(c = \frac{1}{2})$ $T(n) = \Theta(f(n)) = \Theta(n)$이다.

③

$$\begin{aligned} T(n-1) &= 2T(n-1) + 2 \\ &= 2(2T(n-2) + 2) + 2 = 2^2 T(n-2) + 2^2 + 2 \\ &= 2^2(2T(n-3) + 2) + 2^2 + 2 = 2^3 T(n-3) + 2^3 + 2^2 + 2 \\ &\cdots \\ &\cdots \\ &= 2^{n-1} T(1) + 2^{n-1} + 2^{n-2} + \cdots + 2 \\ &= \Theta(2^n) \end{aligned}$$

(위에서 $T(1)$은 무조건 상수가 된다.)

④

$$\begin{aligned} T(n) &= 3T(\frac{n}{3} + 15) + \Theta(n) \\ &\leq 3c(\frac{n}{3} + 15)\log(\frac{n}{3} + 15) + \Theta(n) \\ &= cn\log(\frac{n}{3} + 15) + 45c\log(\frac{n}{3} + 15) + \Theta(n) \\ &\leq cn\log\frac{2n}{3} + 45c\log\frac{2n}{3} + \Theta(n) \\ &= cn\log n + c(\log 2 - \log 3)n + 45c\log\frac{2n}{3} + \Theta(n) \quad -- (1) \\ &\leq cn\log n \end{aligned}$$

$c$를 적당히 크게 잡으면 위 (1)에서 $cn\log n$을 제외한 나머지 항이 0 이하가 되도록 할 수 있다. 또한 $n$은 $n > \frac{n}{3} + 15$와 $\frac{2n}{3} \geq \frac{n}{3} + 15$를 동시에 만족할 수 있도록 충분히 크면 되므로 문제 없다.

⑤ 교과서의 내용만으로는 풀기 힘든 것을 하나 포함시켜 놓았다. 교과서나 MIT Press의 Introduction to Algorithms에 있는 마스터 정리로는 커버하지 못하지만 원래의 마스터 정리는 이를 커버한다. ($\frac{f(n)}{h(n)} = \Theta(\log^k n)$이면 $T(n) = \Theta(h(n)\log^{k+1} n)$이다.) $h(n) = n^{log_2 2} = n$, $f(n) = n\log^2 n$, $\frac{f(n)}{h(n)} = \Theta(\log^2 n)$이므로 $T(n) = \Theta(h(n)log^{2+1} n) = \Theta(n\log^3 n)$이다. 이

것이 유형 2에 속하는 것이 아니라는 점을 상기하기에 좋은 예제이다.

4.

① $T(n) = 2T(n-1) + 1,\ T(1) = 1$

② 증명의 골격만 보이면, $T(n) \geq c2^n + 1$라고 클레임한 후

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &\geq 2(c2^{n-1} + 1) + 1 = c2^n + 3 \\ &\geq c2^n + 1 \end{aligned}$$

로 증명이 끝난다.

③ ②에서 추정후 증명법을 사용하였음.

5. $T(n) = T(n-2) + a,\ T(0) = T(1) = b$ ($a, b$는 상수)
대치법으로 전개하면 $T(n) = \Theta(n)$이 된다.

6. $T(n) = T(n-1) + \Theta(n)$
이를 대치법으로 전개하거나 추정 후 증명법으로 증명하면 $T(n) = \Theta(n^2)$이 된다.

7. $T(n) = T(\frac{n}{3}) + \Theta(n)$
마스터 정리에 의해 $T(n) = \Theta(n)$이 된다. 대치법이나 추정 후 증명법을 사용해도 잘 풀린다.

8. $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
마스터 정리에 의해 $T(n) = \Theta(n\log n)$이 된다. 대치법이나 추정 후 증명법을 사용해도 잘 풀린다.

9. $T(n) = 2T(\frac{n}{2}) + \Theta(1)$
마스터 정리에 의해 $T(n) = \Theta(n)$이 된다. 대치법이나 추정 후 증명법을 사용해도 잘 풀린다.

10. $T(n) = \frac{3}{2}T(\frac{n}{2}) + \Theta(1)$
마스터 정리에 의해 $T(n) = \Theta(n^{\log_2 \frac{3}{2}})$이 된다. 대치법이나 추정 후 증명법을 사용해도 잘 풀린다.

11. $T(n) = 3T(\frac{n}{3}) + \Theta(n)$
마스터 정리에 의해 $T(n) = \Theta(n\log n)$이 된다. 대치법이나 추정 후 증명법을 사용해도 잘 풀린다.

12. $T(n) = 2T(\frac{n}{3}) + \Theta(n)$
마스터 정리에 의해 $T(n) = \Theta(n)$이 된다. 대치법이나 추정 후 증명법을 사용해도 잘 풀린다.

## 3장

1, 2, 3은 단순작업이므로 생략

4.

① $O(n)$

② $O(n)$

③ $O(n)$

5.

① $O(n^2)$

② $O(n^2)$

③ $O(n^2)$

④ $O(n\log n)$

⑤ $O(n^2)$

⑥ $O(n\log n)$

6.

① $O(n^2)$

② $O(n^2)$

③ $O(n^2)$

④ $O(n\log n)$

⑤ $O(n\log n)$

⑥ $O(n\log n)$

7. 원소 A[i]가 들어갈 자리를 찾기 위한 시간은 절약할 수 있지만, A[i]가 들어갈 자리부터 오른쪽에 있는 원소들을 모두 한 칸씩 이동해야 하므로 점근적 시간이 절약되지 않는다. 각 A[i]를 위해 평균 $\frac{i-1}{2}$ 번의 이동이 필요하다.

8. ③ 삽입정렬 - 이 경우에는 선형시간이 든다.

9. 이미 정렬된 배열이 입력으로 들어오는 특수한 상황에서는 다음의 세 그룹으로 나누어진다.

   $\Theta(n)$ - ③

   $\Theta(n\log n)$ - ④, ⑥

   $\Theta(n^2)$ - ①, ②, ⑤

   * 여기서 ⑤의 퀵정렬은 본문의 퀵정렬을 가정. 만일 기준 원소를 임의로 선택한다면 문제의 입력에 대해 퀵정렬은 평균적으로 $\Theta(n\log n)$이 소요된다.

10.

① $\Theta(n^2)$

② $\Theta(n^2)$

③ $\Theta(n^2)$

④ $\Theta(n\log n)$

⑤ $\Theta(n^2)$

⑥ $\Theta(n\log n)$

11. 정렬이 제대로 되지 않는다. 안정한 정렬이 아닌 경우 k번째 자리의 수가 같으면, 앞에서 k-1번째 자리까지 제대로 정렬되어 있던 것이 순서가 바뀔 수 있다.
12.
    ① 선택정렬 - X (사실 이것은 최대값을 정하는 방법에 따라 다르다. 최대값 중 가장 뒤에 있는 것을 최대값으로 삼으면 안정성을 유지할 수 있다.)
    ② 버블정렬 - O
    ③ 삽입정렬 - O
    ④ 병합정렬 - O (단, 왼쪽과 오른쪽의 원소가 같을 때는 왼쪽부터 꺼내야 한다.)
    ⑤ 퀵정렬 - X
    ⑥ 힙정렬 - X
    ⑦ 기수정렬 - O
    ⑧ 계수정렬 - O
13. 배열 A[1 ... n]의 모든 원소에 r+1을 더한 다음 [알고리즘 2-9] countingSort(A, B, n)을 호출하되 알고리즘 안의 상수 k 대신 2r+1을 사용한다. 이제 정렬된 결과에서 모든 원소에 r+1를 빼주면 된다.
14. A[i]가 A[2], A[3]보다 크지 않으면 힙성질이 깨지지 않는다. 이런 상황은 매우 드문데 A[i]가 A[2], A[3] 중 크지 않은 원소의 후손이고 해당 원소에서 A[i]에 이르는 경로에 있는 모든 원소는 값이 같아야 한다. (A[1]은 관심의 대상에서 제외하므로 A[1]과의 관계는 더 이상 생각할 필요 없다.)

## 4장

1. A[1 ... n]을 정렬한 다음 A[i]가 답이다. 점근적 소요시간은 정렬의 소요시간과 일치한다.
2.

```
select(A, 1, 7, 5)
    partition 결과: <3, 5, 2> <4> <8, 6, 9>
    select(A, 5, 7, 1) 호출
        partition 결과: <8, 6> <9>
        select(A, 5, 6, 1) 호출
            partition 결과: <6> <8>
            select(A, 5, 5, 1) 호출
                6을 return한다.
```

3. i가 원소수의 범위를 벗어나지 않는 한 그런 일은 없다. 즉, i가 0 이하이거나 원소수보다 클 경우에는 그런 경우가 생길 수 있는데 [알고리즘 4-1]은 i가 이 범위안에 있다고 가정하고 있으므로 이 경우에는 제대로 작동하지 않는다.
4. 상관없이 제대로 작동한다.
5. 역시 상관없이 작동한다.
6. $T(n)=T(9n/10)+\Theta(n)$로부터 $\Theta(n)$의 시간이 든다. (마스터 정리를 쓰면 간단하게 알 수

있다. 추정후 증명이나 반복대치로 증명해도 된다.)

7. T(n)=T(99n/100)+ Θ(n)로부터 Θ(n)의 시간이 든다.
8. 7개 짜리와 9개 짜리 모두 선형시간에 찾는다.

   7개 짜리의 경우, 점근적 수행시간은 $T(n)=T(\lceil \frac{n}{7} \rceil)+T(\frac{5n}{7}+3)+\Theta(n)$이 되어 $T(n)=\Theta(n)$이 된다.

   9개 짜리의 경우, 점근적 수행시간은 $T(n)=T(\lceil \frac{n}{9} \rceil)+T(\frac{13n}{18}+4)+\Theta(n)$이 되어 $T(n)=\Theta(n)$이 된다.
9. 3개 짜리는 최악의 경우 선형시간에 찾지 못한다.

   $T(n)=T(\lceil \frac{n}{3} \rceil)+T(\frac{2n}{3}+1)+\Theta(n)$이 되어 $T(n)=\Theta(n\log n)$이 된다.
10. 같은 점: 재귀적 호출을 사용하는 점, 분할 후 한쪽을 선택하는 점
    다른 점: select는 분할의 균형이 보장되지 않고, linearSelect는 분할이 최악의 경우 어떤 상수 비율 이상으로 기울어지지 않는다. 이것 때문에 최악의 경우에 대한 수행시간이 달라진다.
11. 알고리즘은 아래와 같다.

    수행시간은 $T(n)=T(\lceil \frac{n}{8} \rceil)+T(\frac{3n}{4}+3)+\Theta(n)$이 되어 $T(n)=\Theta(n)$이 된다.

```
linearSelect(A, p, r, i)
▷ A[p ... r]에서 i번째 작은 원소를 찾는다
{
    1. 원소의 총 수가 8개 이하이면 원하는 원소를 찾고 알고리즘을 끝낸다.
    2. 전체 원소들을 8개씩의 원소를 가진 ⌈n/8⌉ 개의 그룹으로 나눈다. (원소
       의 총수가 8의 배수가 아니면 이중 한 그룹은 8개 미만이 된다.)
    3. 각 그룹에서 중앙값을 (4번째 또는 5번째 원소중 임의로 정한다) 찾는다.
       이렇게 찾은 중앙값들을 m_1, m_2, ... , m_⌈n/8⌉ 이라 하자.
    4. m_1, m_2, ... , m_⌈n/8⌉ 들의 중앙값 M을 재귀적으로 구한다. 원소의 총수가
       홀수면 중앙값이 하나이므로 문제가 없고, 원소의 총수가 짝수일 경우는
       두 중앙값 중 아무거나 임의로 선택한다.        ▷ call linearSelect( )
    5. M을 기준원소로 삼아 전체 원소를 분할한다. (M보다 작거나 같은 것은 M
       의 왼쪽에, M보다 큰 것은 M의 오른쪽에 오도록)
    6. 분할된 두 그룹 중 적합한 쪽을 선택하여 단계 1~6을 재귀적으로 반복한
       다.                                          ▷ call linearSelect( )
}
```

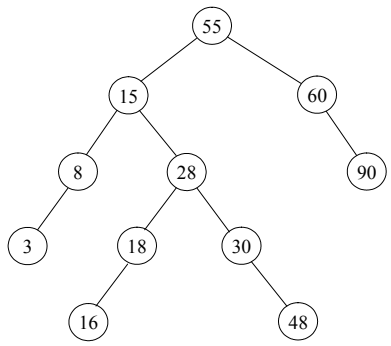
12. linearSelect 알고리즘을 이용하여 선형시간에 중앙값을 찾는다. 이렇게 찾은 중앙값을 기준원소로 하여 분할을 한 다음 재귀적으로 정렬한다. 중앙값으로 분할했으므로 배열

은 항상 이등분된다. 수행시간은 $T(n)=2T(\frac{n}{2})+\Theta(n)$이 된다. (여기서 중앙값을 찾는 작업과 분할 작업이 모두 선형시간이므로 $\Theta(n)$은 이들을 합친 시간이다.) 이로부터 최악의 경우에도 $T(n)=\Theta(n\log n)$이 된다.

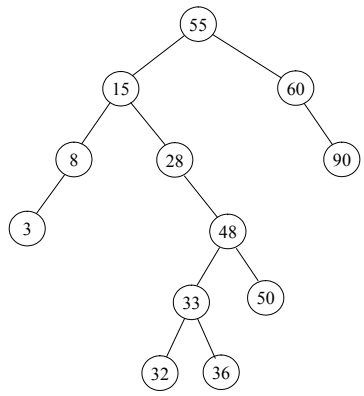
13. $\frac{n}{10}\le k<n$인 모든 $k$에 대해 성립하면 $n$에 대해서도 성립함을 보이면 이것은 $\frac{n}{10^2}$, $\frac{n}{10^3}$, ...와 같이 $n_0$에 이를 때까지 파급된다. 결국 이것은 $n_0 \le k < n$인 모든 $k$에 대해 성립하면 $n$에 대해서도 성립한다는 것과 같은 것이다.

## 5장

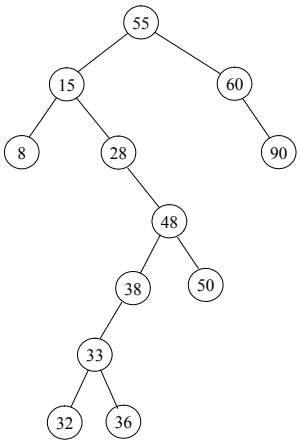
1. 55, 15, 28, 30
2.



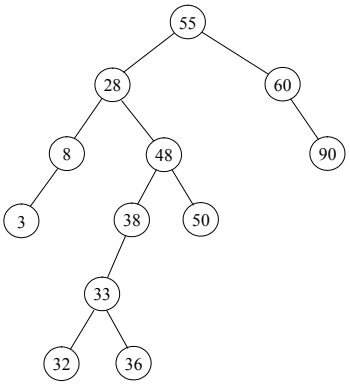

3.



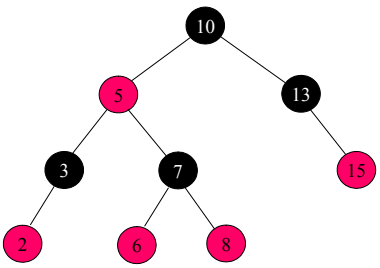

4.


55
15
60
8
28
90
48
38
50
33
32
36

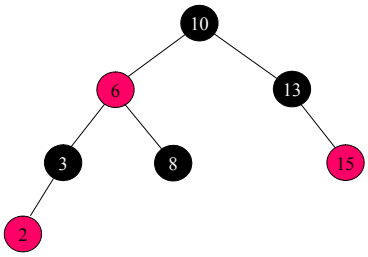
5.


55
28
60
8
48
90
3
38
50
33
32
36

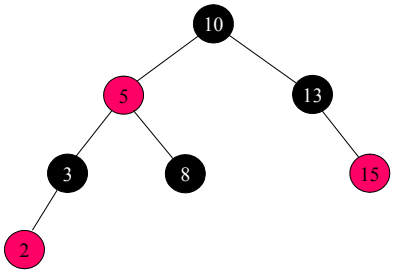
6.


10
5
13
3
7
15
2
6
8

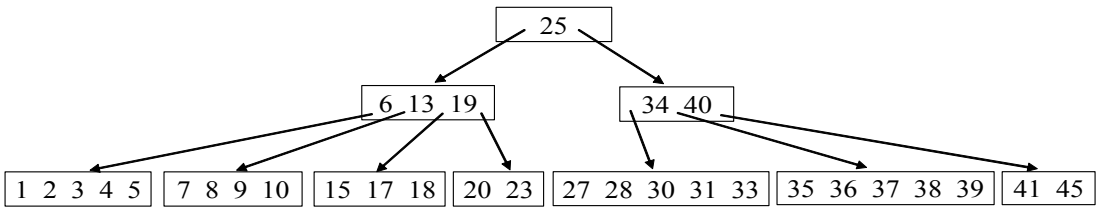
7.


10
6
13
3
8
15
2

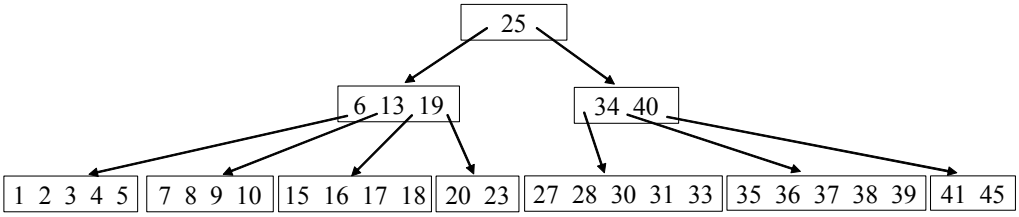
8.


10
5
13
3
8
15
2

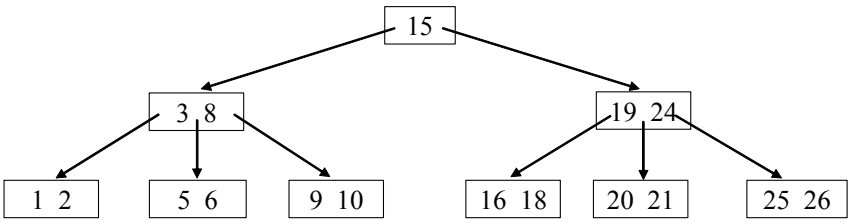
9.


25
6 13 19
34 40
1 2 3 4 5
7 8 9 10
15 17 18
20 23
27 28 30 31 33
35 36 37 38 39
41 45

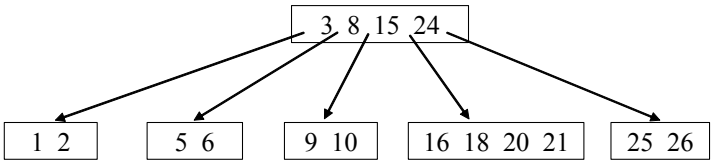
10.


25
6 13 19
34 40
1 2 3 4 5
7 8 9 10
15 16 17 18
20 23
27 28 30 31 33
35 36 37 38 39
41 45

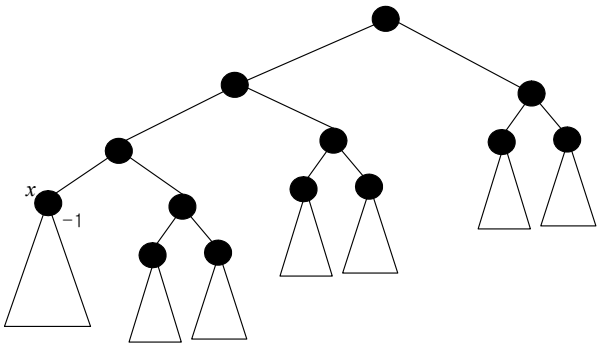
11.




12.




13. 아래 레드블랙 트리에서 노드 $x$에서 문제가 발생했다면 세 번 연속 같은 문제가 발생한다.




14. 레드블랙트리는 이진트리이고 B-트리는 다진트리이므로 임의의 노드에서 분기할 곳을 찾는데 있어 B-트리가 시간이 더 든다. 대신 B-트리의 높이가 낮으므로 트리에서 방문해야 할 노드의 수는 줄어든다. 외부검색트리라면 높이가 낮다는 것은 디스크 접근횟수를 줄이므로 절대적으로 중요하다.

15. $24k+4k+4+4 \le 8192$
$28k \le 8184$
$k \le 292.29$
최대 292개의 키를 갖도록 한다.

16. 트리의 높이를 $h$라면 $h=\log_2(n+1)$이다. 루트가 키 $x$에 의해 분기하므로 필요한 시간은 $T_x(h)=2+2T_x(h-2)$가 되어 전개하면 $\Theta(\sqrt{n})$이 된다. 여기서 $T_x(h)$는 높이가 $h$인 트리의 루트가 $x$에 의해 분기를 할 경우의 점근적 시간을 의미한다.

# 6장

1.

| | |
|---|---|
| 0 | |
| 1 | 40 |
| 2 | |
| 3 | |
| 4 | 30 |
| 5 | |
| 6 | |
| 7 | 20 |
| 8 | 33 |
| 9 | 46 |
| 10 | 10 |
| 11 | 50 |
| 12 | 60 |

2.

| | |
|---|---|
| 0 | |
| 1 | 40 |
| 2 | |
| 3 | |
| 4 | 30 |
| 5 | |
| 6 | |
| 7 | 20 |
| 8 | 33 |
| 9 | 60 |
| 10 | 10 |
| 11 | 46 |
| 12 | 50 |

3. 문제가 틀렸다. f(x)= x mod 11을 사용하면 33의 경우 f(x)=0이 되어 충돌이 생기면 더 이상 진행이 되지 않는다. 아래 그림은 f(x)= 1 + (x mod 11)을 사용했을 경우의 해시테이블 모양이다.
(본문의 [그림 6-8]의 예도 같은 에러를 포함하고 있다. f(x)의 값이 0이 될 수 없도록 바꾸어야 한다.)

| | |
|---|---|
| 0 | 46 |
| 1 | 40 |
| 2 | |
| 3 | |
| 4 | 30 |
| 5 | |
| 6 | 60 |
| 7 | 20 |
| 8 | 33 |
| 9 | |
| 10 | 10 |
| 11 | 50 |
| 12 | |

4. 삽입 비용만 더 들고, 평균적인 검색시간은 동일하다.
5. 삽입시간은 연결리스트의 길이에 비례하므로 원래 버전보다 비용이 더 든다.
   성공적인 검색의 경우 비용이 원래 버전과 동일하다. 실패하는 검색은 연결리스트를 다 보지 않고도 끌낼 수 있으므로 비용이 덜 든다. 삭제는 성공적인 검색과 연동되므로 비용이 동일하다.
6. $f(x)=dk$, $m=dl$이라 하자. 여기서 $k,l$은 양의 정수이다. 여기서 $h_i(x)=(h(x)+if(x))\bmod m$ 이라면 $i$번째 조사에서는 $h(x)$에서 $if(x)$ 만큼 점프를 한다. $l$번째 조사에서는 $lf(x)$ 만큼 점프를 하는데 $lf(x)=ldk=km$이 되어 $m$의 배수이므로 $h(x)$ 자리와 일치하게 된다. 그러므로 기껏해야 $l$번, 즉, $\frac{m}{d}$번밖에 조사하지 못한다.
7. 100, ..., 1000은 각각 1, 2, ..., 10번의 조사가 필요하다.
   101, ,,., 1001은 각각 10, 11, ..., 19번의 조사가 필요하다.
   102, ..., 1002는 각각 19, 20, ..., 28번의 조사가 필요하다.
   103, ..., 1003은 각각 28, 29, ..., 37번의 조사가 필요하다.
   104, ..., 1004는 각각 37, 38, ..., 46번의 조사가 필요하다.
   이들의 합은 1271이므로 50으로 나누면 각 원소당 평균 25.4회의 조사가 필요하다.
8. ① $h_i(x)=(h(x)+\frac{i(i+1)}{2})\bmod m$
   $=(h(x)+\frac{1}{2}i^2+\frac{1}{2}i)\bmod m$
   이므로 이차원 조사이다.
   ② 모순을 유도하기 위해 $i\neq j$인 $0\le i,j\le m-1$에 대하여 $h_i(x)=h_j(x)$인 $i,j$가 존재한다고 가정하자.
   이것은 $(h_i(x)-h_j(x))\bmod m=0$란 뜻이다. 전개하면

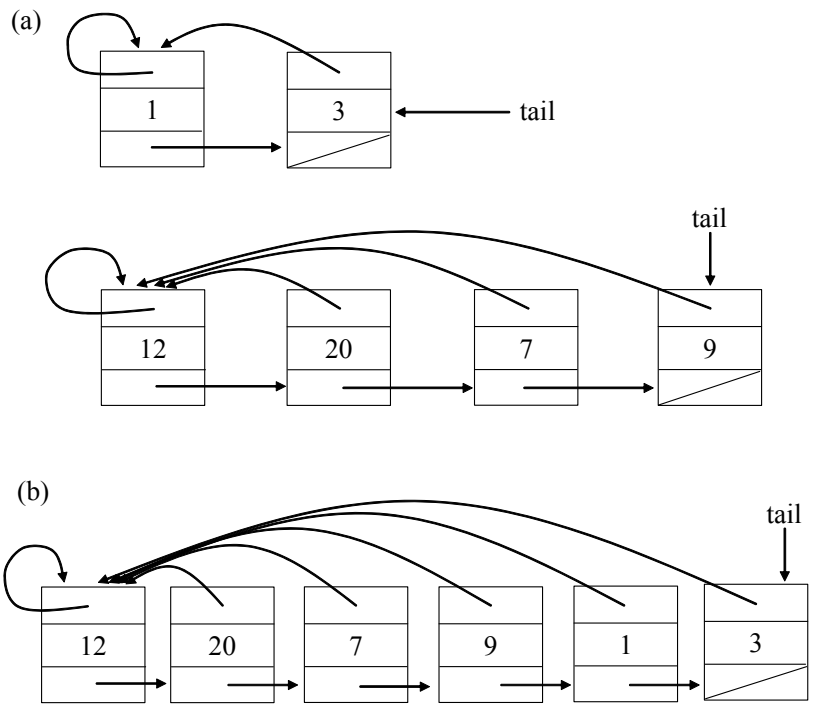
$$\frac{1}{2}(i^2+i-j^2-j)\bmod m=0$$
$$\frac{(i-j)(i+j+1)}{2}\bmod m=0$$
$$(i-j)(i+j+1)=2m \text{ or } 4m \text{ or } 6m \ldots$$
($i \neq j$이므로 0은 될 수 없다)

$i-j, i+j+1$은 하나가 짝수이면 다른 하나는 반드시 홀수이다. 홀수는 소인수분해해도 2의 인자를 갖고 있지 않으므로 짝수 부분이 최소한 $2m(m=2^k)$의 배수가 되어야 한다. 그러나 $i-j<m, i+j+1<2m$이므로 $i-j, i+j+1$ 중의 하나가 $2m$의 배수가 되는 것은 불가능하다.

그러므로 서로 다른 두 $i, j$가 $h_i(x)=h_j(x)$가 되는 경우는 존재하지 않는다. 따라서 $h_0(x), h_1(x), \ldots, h_{m-1}(x)$는 {0, 1, 2, ..., m-1}로 이루어지는 순열을 이룬다.

# 7장

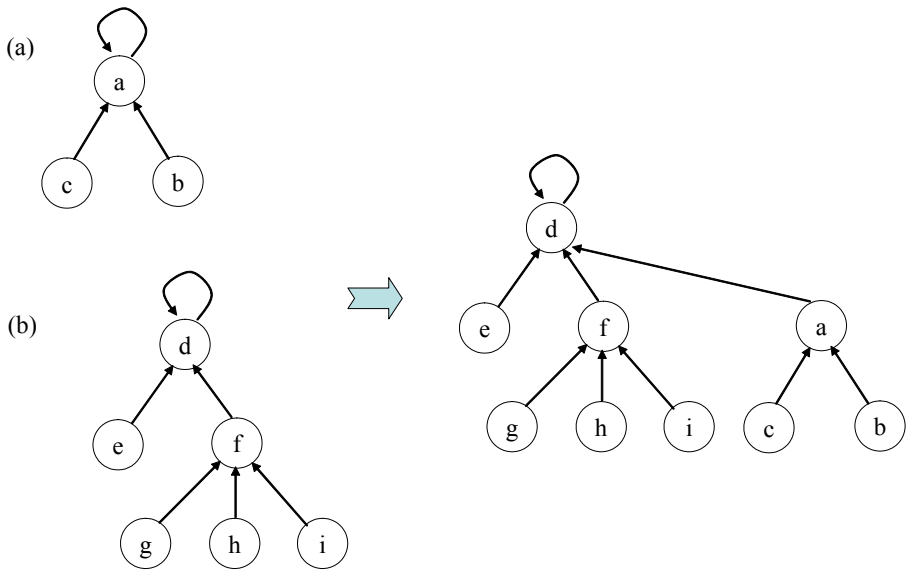
1.




2. Make-Set은 상수시간이 소요되며 $n$번 수행되므로 $\Theta(n)$ 만큼 소요된다. Find-Set도 상수시간이 소요되면 최대 $m-n$번 수행되므로 $\Theta(m-n)$ 만큼 소요된다. 시간을 결정하는 것은 Union이다. 최악의 경우는 모든 원소가 하나의 집합에 속하는 상태에서 Make-Set에 의해 하나의 원소로 구성된 집합이 만들어지고, 이미 만들어지는 집합과 이 새로 만든 집합이 Union되면서 큰 집합의 원소들이 작은 집합 뒤에 연결되는 경우이다. Make-Set은 총 $n$번만 수행되므로 Union은 $n-1$번을 초과할 수 없다. 그러므로 Union

이 미칠 수 있는 최악의 수행시간은 $\Theta(n^2)$이다. 그러므로 최악의 경우 수행시간은 $\Theta(m-n+n^2)$이다.

3.




4.

```
Find-Set(x)
{
        t ← x;
        while (t!=p[t])
                t ← p[t];
        root ← t;
        t ← x;
        repeat{
                prev ← t;
                t ← p[t];
                p[prev] ← root;
        }until (t!=p[t])
        return t;
}
```

5. [정리 2]로부터 집합의 원소수가 n이면 랭크는 기껏해야 $\lfloor \log_2 n \rfloor$ 이다.

6.

| $\log^* n$ | $n$ |
|---|---|
| 1 | 1 |
| 2 | $2^2 = 4$ |
| 3 | $2^4 = 16$ |
| 4 | $2^{16} = 65536$ |
| 5 | $2^{65536}$ |

# 8장

1.

① 14가지

② 계산과정을 아래의 테이블에 보인다. 흑색, 적색, 청색, 녹색의 순서로 계산된다. 최적순서는 $A_1(A_2(A_3(A_4A_5))$이고 최소비용은 1185이다.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | | 1000 | 1050 | 1975 | 1185 |
| 2 | | | 700 | 1225 | 1035 |
| 3 | | | | 2100 | 735 |
| 4 | | | | | 315 |
| 5 | | | | | |

2. n 개의 행렬을 곱하는 순서의 총 가지수를 T(n)이라 하면

$$T(n) = T(1)T(n-1) + T(2)T(n-2) + T(3)T(n-3) + \ldots + T(n-1)T(1)$$

이 된다. $c = \frac{1}{2}$, $n_0 = 1$로 잡으면 모든 $n \geq n_0$에 대해 $T(n) \geq c2^n$을 만족하므로 $T(n) = \Omega(2^n)$이다. 이것의 증명은 다음과 같다.

우선 T(1)=1이다. $T(1) \geq c2^1$을 만족한다. $n$보다 작은 모든 $k$에 대해 $T(k) \geq c2^k$이라고 가정하면,

$$\begin{aligned} T(n) &= T(1)T(n-1) + T(2)T(n-2) + T(3)T(n-3) + \ldots + T(n-1)T(1) \\ &\geq c2^1 c2^{n-1} + c2^2 c2^{n-2} + \ldots + c2^{n-1} c2^1 \\ &= nc^2 2^n \\ &\geq c2^n \end{aligned}$$

이 식은 모든 $n \geq 2$에 대해 만족된다. 따라서 모든 $n \geq 1$에 대해 $T(n) \geq c2^n$이다.

3. 아래는 다이내믹 프로그래밍으로 최장공통부분순서를 구하는 과정에서 채우는 테이블의 내용이다. LCS의 길이는 9이고, 최장공통부분순서는 011101010이다.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 |
| 4 | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 |
| 5 | 0 | 1 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 5 | 5 |
| 6 | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
| 7 | 0 | 1 | 2 | 3 | 3 | 4 | 5 | 6 | 6 | 6 | 6 | 6 |
| 8 | 0 | 1 | 2 | 3 | 3 | 4 | 5 | 6 | 7 | 7 | 7 | 7 |
| 9 | 0 | 1 | 2 | 3 | 4 | 4 | 5 | 6 | 7 | 8 | 8 | 8 |
| 10 | 0 | 1 | 2 | 3 | 4 | 4 | 5 | 6 | 7 | 8 | 8 | 9 |
| 11 | 0 | 1 | 2 | 3 | 4 | 4 | 5 | 6 | 7 | 8 | 8 | 9 |

4. rMatrixChain을 호출하기 전에 배열 $m[0...n, 0...n]$을 준비하고 $m[0,*]$와 $m[*,0]$은 모두 -1, 나머지는 모두 -1로 초기화 해둔다. 재귀호출을 사용한 동적 프로그래밍 알고리즘은 다음과 같다.

```
matrixChain(i, j)
{
    if (m[i,j] ≠ −1) then return m[i,j];   ▷ 한번 호출된 적이 있으면 진행중지
    min ← ∞;
    for k ← i to j-1 {
                q ← rMatrixChain(i, k) + rMatrixChain(k+ 1, j) + p_{i-1} p_k p_j;
                if (q<min) then min ← q;
    }
    return min;
}
```

5. 최장공통 부분순서의 복원을 위해 세 상수 I, J, IJ를 준비한다. B[i, j] = IJ이면 A[1...i]와 A[1...j]의 LCS의 마지막 문자는 A[i]=A[j]임을 뜻하므로 이를 이용해서 LCS를 복원할 수 있다.

```
LCS(m, n)
▷ 두 문자열 X_m과 Y_n의 LCS의 길이 구하기
{
    ▷ 세 상수 I, J, IJ를 준비한다. 값은 아무렇게나 주어도 상관없다.
    for i ← 0 to m
            C[i, 0] ← 0;
    for j ← 0 to n
            C[0, j] ← 0;
    for i ← 1 to m
            for j ← 1 to n
```

```
                    if ( x_i = y_j) then {
                            C[i, j] ← C[i-1, j-1] + 1;
                            B[i, j] ← IJ;
                    } else {
                            if (C[i-1, j] > C[i, j-1])
                                    then {
                                            C[i, j] ← C[i-1, j];
                                            B[i, j] ← J;
                                    } else {
                                            C[i, j] ← C[i, j-1];
                                            B[i, j] ← I;
                                    }

                    }
        return C[m, n];
}
```

6.

```
matrixPath(i, j)
▷ (i, j)에 이르는 최고점수
{
        c[1,1] ← m_11;
        for i ← 2 to n
                c[i,1] ← m_i1 + c[i-1,1];
        for j ← 2 to n
                c[1,j] ← m_1j + c[1,j-1];
        for i ← 2 to n
                for j ← 2 to n
                        c[i,j] ← m_ij + max(c[i-1,j], c[i,j-1], c[i-1,j-1]);
        return c[n,n];
}
```

7. 주어진 n 개의 수를 정렬하면 길이 n인 다른 수열을 만들어진다. 이 수열과 주어진 수열의 LCS를 구하면 최장 단조증가 부분수열의 길이가 된다. 수행시간은 $\Theta(n^2)$이다.

8. $w_i$가 마지막 문자로 임의의 행이 끝나는 경우의 최소여백제곱합을 $s_i$라 하자.

$$s_i = \min_{k \quad s.t. \sum_{j=k+1}^{i} w_j + (i-k) \le 80} \left\{ s_k + (80 - (i-k) - \sum_{j=k+1}^{i} w_j)^2 \right\}$$

$s_i$를 구하기 위해 살펴보아야 할 $k$의 가지수는 최대 40을 넘지 못한다. 이것은 단 한글자로 이루어진 40개의 단어가 연속될 경우이므로 실제로는 이것보다 훨씬 작은 수의 $k$만 살펴보면 된다. 이것은 40 이하의 상수이므로 하나의 $s_i$를 구하는 데는 상수 시간이 든다. 그러므로 $s_1$부터 $s_n$까지 차례로 구하는 데는 $\Theta(n)$ 시간이 든다. 최종적인 답은 $\min_{k \ \ s.t. \sum_{j=k+1}^{n} w_j + (i-k) \le 80} \{s_k\}$이다. 이것은 마지막 행의 여백을 고려하지 않는다는 점을 반영한 것이다.

9. ①$s_i$를 직원 $i$를 루트로 하는 트리에서 직원 $i$를 포함하는 최대 날라리 분위기라 하고, $t_i$를 직원 $i$를 포함하지 않는 최대 날라리 분위기라 하자. $s_i$와 $t_i$는 각각 다음과 같은 관계식을 가진다.

$$s_i = \sum_{k\text{는 } i\text{의 직속부하직원}} t_k$$
$$t_i = \sum_{k\text{는 } i\text{의 직속부하직원}} \max\{s_k, t_k\}$$

최종적인 답은 $\max\{s_{\text{사장}},\ t_{\text{사장}}\}$이다.

② $s_{\text{사장}}$이 최대가 되도록 한다.

10.

$a_{ij}$: 문자열 $x_i...x_j$에 대하여 결과가 $a$가 될 수 있는가? (Yes:1, No: 0)
$b_{ij}$: 문자열 $x_i...x_j$에 대하여 결과가 $b$가 될 수 있는가?
$c_{ij}$: 문자열 $x_i...x_j$에 대하여 결과가 $c$가 될 수 있는가?

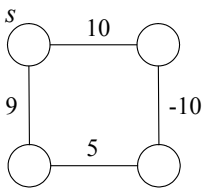
$$a_{ii} = \begin{cases} 1 & \text{if } x_i = a \\ 0 & \text{if } x_i \neq a \end{cases}$$
$$a_{ij} = \cup_{i \le k \le j-1} (a_{ik} \wedge c_{k+1,j}) \vee (b_{ik} \wedge c_{k+1,j}) \vee (c_{ik} \wedge a_{k+1,j})$$

답은 $a_{1n}$

## 9장

1. 인접행렬을 사용한다. 이 경우에는 인접행렬의 모든 원소가 유용한 정보를 갖게 된다. 인접리스트로 할 경우 링크를 위한 오버헤드가 더 들 뿐이다.
2. 인접리스트를 사용한다. 이 경우에는 인접행렬의 대부분 원소가 무용한 정보를 갖게 된다. 인접리스트는 간선의 수 만큼의(또는 간선수의 두 배 만큼의) 노드를 사용하므로 공간이 훨씬 절약된다.

3. 생략
4. v로부터 DFS를 수행하면 S에 있는 정점들 중 중앙에 있는 정점이 항상 맨먼저 방문된다. DFS span이 최소일 때는 S의 중앙에 있는 정점을 방문한 다음 S의 다른 정점들을 모두 방문하고 S 오른쪽으로 빠져나가는 경우로 DFS span은 7이다. DFS span이 최대일 때는 S의 중앙에 있는 정점을 방문한 다음 S에 속하는 정점을 한 개 이상 남긴 상태로 S 오른쪽으로 빠져나간 다음 돌아오면서 남은 정점을 방문하는 경우로 DFS span은 15이다.
5. 생략
6. 크루스칼 알고리즘이 유리하다. 크루스칼 알고리즘은 ②에서 모든 간선을 정렬하는 데 $O(E\log E)$, ③의 while 루프에서 ⑤와 ⑦의 집합 처리와 관련하여 $O(V+E)\log^* V)$의 시간이 든다. 만일 모든 간선의 가중치가 {1, 2, ..., n}에 속하면 계수정렬의 사용하여 선형시간 정렬이 가능하다. 그러므로 크루스칼 알고리즘의 수행시간이 $O(V+E)\log^* V)$이 되어 프림 알고리즘보다 유리해진다.
7. 생략
8. 




9. 최소힙에서 임의의 원소값이 감소하면 해당 원소와 자식들간의 힙성질은 전혀 손상되지 않는다. 해당 원소와 부모 원소간의 힙성질은 깨질 수 있다. 만일 부모 원소보다 더 작으면 부모 노드와 값을 교환한다. 이런 식으로 계속 부모 노드와 비교를 해나가면서 필요하면 값을 교환한다. 값을 교환할 필요가 없는 상태를 만나면 그것으로 힙성질을 만족시키기 위한 수선은 끝이다. $n$개의 원소로 이루어진 힙의 높이는 $O(\log n)$이므로 최악의 경우라 해도 $O(\log n)$ 시간에 수선할 수 있다.
10. 모든 간선의 가중치가 $1, 2, ..., W$ 중의 하나이면 다익스트라 알고리즘에서 집합 S의 바깥에 있는 정점들의 d[] 값은 $\infty$를 제외하고는 최대 $W+1$ 가지밖에 있을 수 없다. 왜냐하면 집합 S에 속한 정점들의 d 값 중 최대값을 x라 하면, S 바깥에 있는 정점들 중 d 값이 $\infty$인 것을 제외하고는 d 값이 x+W를 넘을 수 없다. 그러므로 d 값은 모두 x와 x+W 사이이고, 그렇지 않은 것은 모두 $\infty$이다. 다익스트라 알고리즘의 ③의 extractMin()과 ⑥의 힙 수선은 각각 $O(\log W)$의 시간이면 된다. 그러므로 다익스트라 알고리즘은 $O((V+E)\log W)$이면 충분하다.

    최소힙을 extractMin()과 수선을 $O(\log W)$ 시간에 하는 것은 자명하지는 않다. 간단히 설명하면,
    크기가 $W+1$인 배열 H[0 ... W]를 만든다. 배열의 각 원소에는 S 바깥의 정점들이 d 값에 따라 연결리스트로 관리된다. ⑥에서 d 값이 변하면 해당 정점은 배열의 다른 연결리스트로 옮긴다. 배열을 관리하기 위해 현재 S에 속하는 원소들의 d 값 중 가장 큰 것과 같은 d 값을 가진 정점이 들어갈 배열의 인덱스를 나타내는 변수를 하나 둔다(이 변

수값을 b라 하자). S에 새 정점이 들어가 d 값이 커지면 b값이 바뀐다. ⑥에서 d 값이 변하면 b값을 참조하여 배열의 어느 연결리스트에 해당 정점을 저장할지를 알 수 있다. 예를 들어, S에 속하는 정점들 중 최대 d 값이 x이고, S 바깥의 임의의 정점의 d 값이 x+ i라면($0 \le i \le W$) 리스트 H[b+ i]에 연결된다. 이 배열의 원소들(연결리스트의 헤더들) 중 비어있지 않은 것들로 힙을 구성하면 위와 같이 extractMin()과 수선을 $O(\log W)$ 시간에 할 수 있다.

11. 아래와 같이 행렬 $\pi$에 각 $(i, j)$ 쌍에 대해 정점 $i$에서 정점 $j$에 이르는 최단경로에서 $j$에 이르기 직전에 방문하는 정점을 기록해 둔다.

```
FloydWarshall(G)
{
    for i ← 1 to n
        for j ← 1 to n
            d_ij^0 ← w_ij;
    for k ← 1 to n                          ▷ 중간정점 집합 {1, 2, ..., k}
        for i ← 1 to n                      ▷ i : 시작 정점
            for j ← 1 to n {                ▷ j : 마지막 정점
                if (d_ij^(k-1) ≥ d_ik^(k-1) + d_kj^(k-1)) then {
                    d_ij^k ← d_ik^(k-1) + d_kj^(k-1);
                    π_ij^k ← π_kj^(k-1);
                } else {
                    d_ij^k ← d_ij^(k-1);
                    π_ij^k ← π_ij^(k-1);
                }
            }
}
```

이 행렬 $\pi^n$에서 $\pi^n_{ij}$은 정점 $j$ 직전에 방문되는 정점이다. 이를 $p$라면 $\pi^n_{ip}$는 $p$ 직전에 방문되는 정점이다. 이런 식으로 정점 $i$에서 정점 $j$에 이르는 최단경로를 알아낼 수 있다.

12.

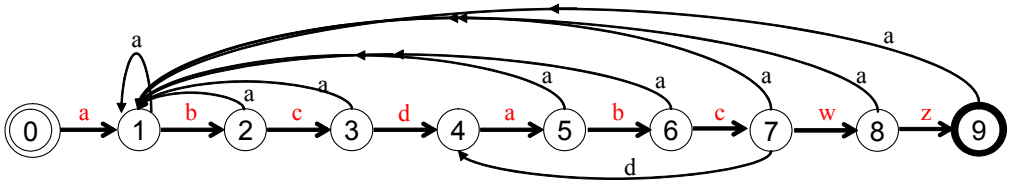
```
Warshall(G)
{
    for i ← 1 to n
        for j ← 1 to n
            if (i = j or w_ij ∈ E)
                then d_ij^0 ← 1;
                else d_ij^0 ← 0;
```

```
    for k ← 1 to n                          ▷ 중간정점 집합 {1, 2, ..., k}
        for i ← 1 to n                      ▷ i : 시작 정점
            for j ← 1 to n                  ▷ j : 마지막 정점
                d_ij^k ← d_ij^{k-1} ∨ (d_ik^{k-1} ∧ d_kj^{k-1});
}
```

13. 해당 그래프에서 최소신장트리를 구하여 이를 T라 하자. 모순을 유도하기 위해 다른 최소신장트리 T'가 존재한다 가정하자. T와 T'가 서로 다른 신장트리이므로 서로 상대방에 없는 간선이 적어도 하나씩 존재한다. T'에는 있지만 T에는 없는 간선 중 하나를 e라 하자. e를 T에 더하면 싸이클이 만들어진다. 이 싸이클에는 T'에 없는 간선이 적어도 하나 있다. 이 중 하나를 f라 하자. 만일 모든 간선들의 가중치가 다르다고 했으므로 e와 f의 가중치는 다르다. 만일 e가 f보다 작다면 T'에서 e를 제거하고 f를 더하면 T'보다 가중치의 합이 작은 신장트리가 만들어진다. T'는 최소신장트리라는 가정에 모순된다. 만이 e가 f보다 크다면 T에서 f를 제거하고 e를 더하면 T보다 가중치의 합이 작은 신장트리가 만들어진다. 역시 T가 최소신장트리라는 가정에 모순된다. 따라서 서로 다른 최소신장트리가 두 개 이상 존재할 수 없다.
14.
    a. e가 T에 속해 있는 경우
    e의 가중치가 작아진 경우에는 여전히 T는 최소신장트리이다. e의 가중치가 커진 경우에만 생각하면 된다. e를 T에서 제거하면 T는 두 개의 트리 $T_1$, $T_2$로 나누어진다. T에 속하지 않는 간선들 중 e보다 작은 모든 간선에 대해 두 끝점이 $T_1$과 $T_2$에 걸치는 것이 있는지 본다. 만일 이런 간선이 존재하면 T는 더 이상 최소신장트리가 아니다.
    b. e가 T에 속하지 않는 경우
    e의 가중치가 커진 경우에는 여전히 T는 최소신장트리이다. e의 가중치가 작아진 경우에만 생각하면 된다. e를 T에 포함시키면 T는 싸이클을 하나 만든다. 이 싸이클 상에서 e보다 가중치가 큰 간선이 존재하면 T는 더 이상 최소신장트리가 아니다.

## 10장

1. P[1...m]과 A[i...i+m-1]을 비교할 때 while 루프를 사용하여 문자를 하나하나 비교할 수 있다. 서로 다른 문자가 나타나는 순간 while 루프를 끝내는 것이 상식적이다. 텍스트와 문자열의 유사도에 따라 다르겠지만 일반적인 경우처럼 매치가 일어나는 것이 그리 흔하지 않은 환경이라면 P[1...m]과 A[i...i+m-1]를 비교할 때 평균적으로 $O(1)$ 시간에 끝날 것이다. 따라서 대부분의 경우 원시적 매칭 알고리즘은 $\Theta(n)$에 가까운 시간이 소요될 것이다.
2.




여기서 명시하지 않은 모든 문자에 대해서는 상태 0으로 감

3. a=0, b=1, c=2로 대응시키기로 하자. “ababaca"는 0101021과 대응된다. 3진수이므로 $1+2*3^1+1*3^3+1*3^5=277$이 되어 277 mod q가 될 것이다. q는 대체로 상당히 큰 수를 잡으므로 277 mod q는 277이 될 것이다.
4. 30 mod 13 = 4이다. A[ ]에서 mod 13으로 4가 되는 부분 문자열은 17, 56, 43으로 세 번 발생한다. (문자열이 문자 집합 {0, 1, 2, ..., 9}로 제한된다는 것을 명시하는 것이 문제가 더 명확했을 것이다.)
5. $m\times m$ 패턴을 선형 문자열로 취급한다. 즉, A[i, j]로부터 시작되는 $m\times m$ 패턴을 문자열 A[i, j]A[i, j+1],...,A[i, j+m-1]A[i+1, j],A[i, j+1]...A[i+1, j+m-1]...A[i+m-1, j]A[i+m-1, j+1]...A[i+m-1, j+m-1]로 간주한다. A[i, j]로부터 시작되는 길이가 m인 문자열의 수치화 값을 $b_{ij}$라 하자. 모든 $b_{ij}$는 본문의 방법으로 쉽게 미리 계산해 놓을 수 있다. A[i, j]로부터 시작되는 $m\times m$ 패턴의 수치화 값을 $k_{ij}$라면

$$k_{ij}=((d^m \bmod q)(k_{i-1,j}-b_{i-1,j}*(d^{m(m-1)} \bmod q))+b_{i+m-1,j}) \bmod q$$

에 의해 간단히 계산할 수 있다. $d^m \bmod q$와 $d^{m(m-1)} \bmod q$는 초기에 한번만 계산해 놓으면 되므로 $k_{ij}$ 계산은 상수 시간에 가능하다.
6. $\pi[1...8]=0,1,1,2,3,4,1,2$
7. TT가 패턴 T'을 갖는지 KMP 알고리즘을 이용해서 체크하면 된다. TT는 T를 반복해서 나열한 것.
8. 텍스트의 일부 문자를 보지 않고도 끝낼 수 있다는 점.
9. jump['a']=1, jump['b']=3, jump['c']=2, jump[기타문자]=10
10.

```
naiveMatching(A, P, i)
{
    if (i ≤ n-m+1) then {
        if (P[1...m]=A[i...i+m-1])
        then A[i] 자리에서 매칭이 발견되었음을 알린다;
        maiveMatching(A, P, i+1);
    }
}
```

11.

① A[i-j+1...i-1]과 P[1...j-1]까지는 매칭된 상태
② KMP(A, P, 1, 1, $\pi$)
③ 수행시간은 변함없다.

12.

```
BoyerMooreHorspool(A, P, k, jump)
{
    if (k≤n) {
        j=m;
        while (j>0 and P[j]=A[k]) {
            j--; k--;
        }
        if (j=0) then A[k-m+1] 자리에서 매칭이 발견되었음을 알린다;
        BoyerMooreHorspool(A, P, k+jump[k], jump);
    }
}
```

최초의 호출은 BoyerMooreHorspool(A, P, m, jump)

## 11장

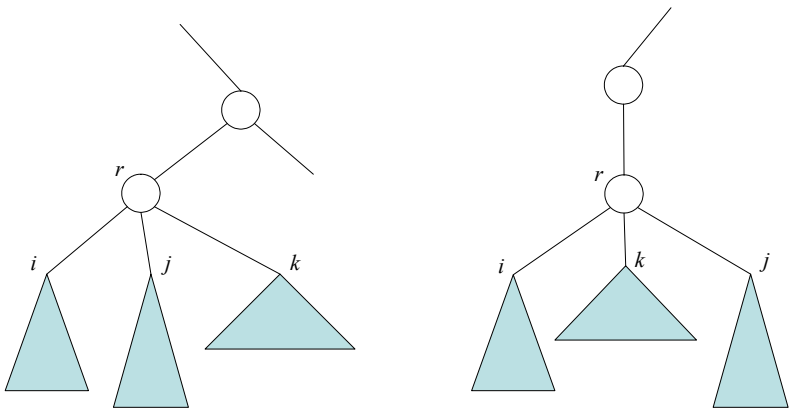
1. ① P에 속한다
   ② NP에 속한다 (P에 속하므로 당연히 NP에도 속한다)
2. 완전 부분 그래프를 구성하는 K 개의 정점 집합으로 주어졌다고 하자. 그래프가 인접행렬로 표현되어 있다면 K 개의 정점 각각이 다른 K-1 개 정점에 모두 연결되어 있는지 보면 되므로 $\Theta(K^2)$의 시간이 든다. 그래프가 인접 리스트로 표현되어 있다면 K 개의 정점 각각의 연결 리스트가 K-1 개 정점 모두를 포함하고 있는지 보면 된다. 연결 리스트에는 K 개 정점이 아닌 다른 정점들도 중간에 나타날 수 있으므로 각 리스트를 확인하는 데 $O(V)$의 시간이 든다. 그러므로 총 $O(KV)$의 시간이 든다.
3. 결정문제(Yes/No 문제)가 아닌 것은 NP가 될 수 없다. 그렇지만 임의의 NP-완비 문제로부터 이 문제로 다항식 시간 변환하는 것은 가능한 경우가 있다. 즉, NP가 아니라도 NP-하드가 될 수 있다.
4. 그래프를 만드는데, 정점은 부울 변수나 그 역과 대응되도록 하고, 절 $x \vee y$가 $\Phi$에 포함되면 $\neg x$ 정점에서 $y$ 정점으로 연결되는 간선과 $\neg y$ 정점에서 $x$ 정점으로 연결되는 간선을 만들어준다. 이 그래프에서 임의의 부울 변수 $x$에 대해 정점 $x$와 $\neg x$가 같은 싸이클에 포함되면 만족가능하지 않고, 그렇지 않으면 만족가능하다.
5. 먼저 위상정렬을 한 다음 이것이 해밀토니안 경로인지 확인하면 된다. 이것이 해밀토니안 경로가 아니면 더 이상 해밀토니안 경로를 만들 수 있는 방법이 없다.
6. A의 사례를 B의 사례로 다항식 시간 변환 가능하므로 B의 사례의 답으로 A의 사례의 답을 알 수 있다. B의 사례를 C의 사례로 다항식 시간 변환 가능하므로 C의 사례의 답

으로 B의 사례의 답을 알 수 있다. 그러므로 C의 사례의 답으로 A의 사례의 답을 알 수 있다. 다항식 시간 변환을 두 번 했으므로 여전히 다항식 시간에 처리된다.
그러므로 문제 A에서 문제 C로 다항식 시간 변환 가능하다.

7. 3SAT 문제의 사례가 주어지면 이 사례에 포함된 리터럴들을 $l_1, l_2, \ldots, l_n$이라 하자. 먼저 $l_1$에 1을 할당한 다음 이 사례가 만족가능한지 알고리즘에게 묻는다. No라고 대답하면 $l_1$에 0을 할당한 다음 만족 가능한지 묻는다. 다시 No라고 하면 이 사례는 만족가능하지 않다. 둘 중 하나에 대해 Yes라고 대답하면 $l_1$은 해당 값으로 고정한다. 이후에 $l_2, l_3, \ldots, l_n$ 순으로 계속 같은 방식으로 묻는데 만족가능하다고 했으므로 각 리터럴의 적어도 한 값에 대해서는 만족가능하다는 답이 나온다. 리터럴들을 Yes 대답이 나온 값으로 고정해 나가면 된다. 다항식 시간 알고리즘을 최대 $2n$ 번 호출하면 되므로 다항식 시간에 완료할 수 있다.

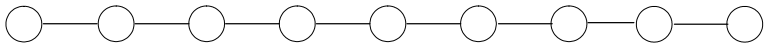
8.

1) ISP에서 질문에 대한 대답이 Yes라는 근거가 주어지면(그래프 G에서 T와 동형인 부분 그래프가 주어지면), 이것이 T와 동형임을 확인하는 것은 다항식 시간에 가능하므로 ISP는 NP이다. 두 트리가 동형임을 확인하는 것은 trivial 하지는 않은데 골격만 간단히 기술하면 다음과 같다. 우선 각 그래프에서 모든 리프 노드에 0의 label을 붙인다. 이 리프 노드들을 level 0라 하자. 이 정점들을 level 1이라 하자. 이런 식으로 level을 부여하면 임의의 리프 노드로부터의 최단거리가 해당 정점이 속한 level이 된다. Level k에 속하는 각 정점에 대하여 이에 인접한 level k-1의 정점들을 label 순으로 나열한다. 이 순열이 같은 정점들은 모두 같은 label을 부여한다(앞에서 한번 사용한 label과는 다른 값을 사용해야 한다). 아래 그림은 labeling의 한 예를 보인다. 두 정점에 인접한 정점들의 label의 조합이 같으면({i, j, k}와 {i, k, j})) 같은 label(r)이 부여된 예를 보이고 있다.




이런 식으로 모든 level에 대해 label 작업을 한다. 최종적으로 모든 level에서 두 트리의 정점 수가 같고 각 정점들의 label을 크기순으로 나열했을 때 같은 순열이 되면 두 트리는 동형이다. 위 작업은 다항식 시간에 이루어진다. 그러므로 ISP는 NP이다.
(ISP의 대답이 Yes라는 근거로 부분 그래프에 더하여 일대일 대응함수 $f$까지 제공될 수도 있다. 이 경우에는 확인 방법이 trivial하다. 다항식 시간 변환에 초점을 맞추기 위해서는 이 방법이 더 나을 것이다.)

2) HAM-PATH 사례의 그래프 G=(V, E)와 |V| 개의 정점을 가진 체인형 트리 T를 ISP 사례로 사용한다. 이 사례를 만드는 것은 다항식 시간에 가능하다. 아래 그림은 |V|가 9인 체인형 트리를 예로 보인다.



그래프 G가 그래프가 T와 동형인 부분그래프를 갖는지 질문한다. 이 질문에 대한 대답이 Yes이면 그래프 G는 해밀토니안 경로를 갖는다. 대답이 No이면 해밀토니안 경로를 갖지 않는다. 그러므로 HAM-PATH는 ISP로 다항식 시간 변환가능하다.

위 1), 2)로부터 ISP는 NP-완비다.

9.

1) G에서 H와 동형인 부분 그래프와 일대일 대응 함수 $f$가 주어질 때 이 두 그래프가 동형임을 확인하는 것은 다항식 시간에 가능하다. 그러므로 SUBGRAPH-ISOMOPHISM은 NP이다.

2) CLIQUE의 사례 그래프 G=(V, E)와 정수 K가 주어지면 그래프 G는 그대로 사용하고 이에 더하여 크기가 K인 완전 그래프 C를 하나 만든다. 이것을 SUBGRAPH-ISOMOPHISM의 사례로 사용한다. 즉, 변환은 $\Theta(K^2)$의 시간이면 충분하다. SUBGRAPH-ISOMOPHISM에서 아래와 같이 질문한다.

"G가 C와 동형인 부분 그래프를 포함하는가?"

이 질문에 대한 대답과 CLIQUE의 질문 "G가 크기 K인 완전 부분 그래프를 포함하는가?"에 대한 대답은 일치한다. 그러므로 CLIQUE은 SUBGRAPH-ISOMOPHISM으로 다항식 시간 변환 가능하다.

위 1), 2)로부터 SUBGRAPH-ISOMOPHISM은 NP-완비다.

10.

1) BIN-PACKING의 질문에 Yes라고 대답할 수 있는 근거로 부분 집합 $U_1, U_2, \ldots, U_k$가 주어지면 각 집합에 속한 항목들의 합이 B를 초과하지 않음을 확인하는 것은 선형시간에 가능하다. 그러므로 BIN-PACKING은 NP이다.

2) PARTITION의 사례가 주어지면 다음과 같이 BIN-PACKING의 사례를 만든다.
U=A, PARTITION의 각 원소 a는 BIN-PACKING의 한 항목이 된다. K=2로 잡고, B는 모든 항목의 합을 2로 나눈 수로 잡는다. 이것이 BIN-PACKING의 사례가 된다.

이 변환은 항목의 수에 대한 선형시간이면 된다. BIN-PACKING에서 아래와 같이 질문한다.

"U를 2개의 상호배타적인 집합 $U_1, U_2$로 분할하되 각 집합이 B를 초과하지 않는 방법이 존재하는가?"

이 질문에 대한 대답과 원래의 PARTITION 사례의 질문에 대한 대답은 일치한다. 그러므로 PARTITION은 BIN-PACKING으로 다항식 시간 변환가능하다.

위 1), 2)로부터 BIN-PACKING은 NP-완비다.

11. 8절에서 보인 바와 같이 삼각부등식을 만족하면 최적해의 3/2배를 초과하지 않는 해를 구할 수 있다. 도시들간의 거리가 1부터 100 사이이면서 삼각부등식을 만족하지 않는 사례에서 모든 간선에 98을 더하면 삼각부등식을 만족하게 된다. 이렇게 변환하면 모든 해는 원래 사례의 해보다 $98n$ 만큼 크다(n은 정점의 총 수). 따라서 최적해 C에 대응되는 해는 $C+98n$이 된다. 근사 비율 3/2를 보장하는 알고리즘이 있으므로 이를 이용해서 $\frac{3}{2}(C+98n)$을 초과하지 않은 해를 구할 수 있다. 여기서 $98n$을 빼면 $C+(\frac{1}{2}C+49n)$이 된다. 주어진 TSP 문제에서 어떤 경우든 최적해 C보다 $\frac{1}{2}C+49n$ 이상 크지 않은 해를 구할 수 있다.

## 12장

1~6. 생략

7. 잃는 점: 최적해를 보장할 수 없다.
   얻는 점: 잔여거리보다 작은 것을 보장하기 위해 추정치와 실제잔여치의 차이가 필요 이상으로 커질 가능성이 많다. 잔여거리보다 작은 것이 보장되지는 않지만 잔여거리를 더 정확히 추정할 수 있는 수단이 있으면 최적해는 보장하지 못하지만 탐색은 더 빠르고 효율성은 높아질 것이다.
   실제 잔여거리보다 작은 추정값을 구할 수 없을 때, 잔여거리보다 작은 것이 보장되지 않는 추정값이라도 쓰면 근사해를 구할 수 있다. (잔여거리의 추정치를 무조건 0으로 놓으면 실제 잔여거리보다 작은 추정값으로 삼을 수는 있으나 아무 의미없는 추정치라 탐색의 품질에 도움을 주지 못한다. 이럴 때 잔여거리보다 작은 것이 보장되지는 않지만 잔여거리를 추정할 수 있는 수단이 있으면 그것을 사용하는 것이 더 나을 수 있다. 물론 추정의 합리성에 따라 품질은 영향을 받겠지만...)
8. 본문에서 사용한 방식은 설명의 간명함을 위해 추정치를 정적으로 구하는 방식이다. 즉, 각 정점에서 다른 정점들에 이르는 간선들 중 가장 짧은 것을 초기에 한 번 구해 놓은 다음 이후에는 어떤 상황이든 이것을 해당 정점과 관계된 추정치로 삼는다. 이것은 지금까지 탐색해온 상황을 고려하지 않으므로 다소 낭비가 있다. 예를 들어 [그림 12-8]에

서 11번 노드의 1-3-4를 보면 추정 잔여거리가 16이다. 이것은 3(정점 4)+10(정점 2)+3(정점 5)에 의해 계산된 것이다. 그런데 정점 5와 관계된 값 3은 정점 4로 연결될 때 가능한 값이다. 그렇지만 정점 4는 이미 정점 3으로부터 연결되어 있으므로 정점 5 다음에 4를 방문할 수는 없다. 정점 5 다음에 방문할 가능성이 남아있는 정점은 1, 2 뿐이다. 그러므로 정점 5와 관계된 추정치로는 10이 최선이다. 이를 이용해서 1-3-4의 추정치를 추정하면 17+23=40이 된다. 33보다 더 실제치에 가까운 추정치이다.