제17회 교원컴퓨터프로그래밍 경진대회(예선)

2020. 9. 5.(토) 수험번호(　　) 소속(　　) 이름(　　) 경상남도교육청

1. 네트워크 기반 공격에 대한 설명으로 옳은 것은? (3.8점)

① 스푸핑 - 네트워크 상의 데이터를 감청하는 기법

② 스니핑 - A와 B 사이의 통신에 끼어들어 패킷을 위조하는 기법

③ 분산 반사 서비스 거부 - 정상적인 서비스 부하량을 늘리는 기법

④ 서비스 거부 - 외부에서 물리적인 장치명, 운영체제 등의 정보를 알아내는 행위

⑤ 네트워크 스캐닝 - 네트워크에 연결된 서버나 클라이언트의 개방 포트를 조사하는 행위

[1번 문항 정답] ⑤

[1번 문항 해설] 길라잡이 39~41페이지

① 스푸핑(Spoofing)과 세션 하이재킹(Session Hijacking)

스푸핑과 세션하이재킹은 A와 B 사이의 통신에 끼어들어 패킷을 위조하는 기법이다. A와 B 사이의 통신에 끼어들어 C가 B인 것처럼 A와 통신할 수 있다. Sniffing 기술에 근거하여 패킷을 도청하고 패킷의 Sequence를 추측하여 두 통신 사이에 엉뚱한 데이터를 집어넣을 수 있다. 패킷 암호화로 어느 정도 예방이 가능하다.

② 스니핑(Sniffing)과 암호화 프로토콜

스니핑은 네트워크 상의 데이터를 감청하는 것을 말한다. 스니핑은 Promiscuous Mode라는 TCP/IP 모드에서 작동한다. 최근 사용하는 스위치 장비는 Promiscuous Mode에서도 스니핑이 되지 않는다. 따라서 스위치 장비 자체를 Jamming하거나 Mirror 기능을 설정하여 스니핑을 하여야 한다.

③ 분산 반사 서비스 거부(DRDoS : Distribute Reflection Denial of Service) 공격

TCP 3중 연결을 이용하는 분산 서비스 공격으로 공격자는 출발지 IP를 공격 대상의 IP로 위조하여 syn 패킷을 다수의 반사 서버로 전송하여 공격 대상의 장비들이 응답하는 syn-ack 패킷을 받아 서비스가 거부 상태가 된다. 반사 서버(Reflection Server)는 DRDoS에 이용되는 라우터 또는 TCP 서버로 icmp 프로토콜의 echo request와 response를 이용하여 동일한 형태의 공격도 가능하다.

DRDoS 공격은 출발지 IP를 위조하는 공격이므로 IP 주소가 위조된 패킷이 인터넷망으로 인입되지 않도록 ISP(Internet Service Provider)가 직접 차단해야 한다. 라우터 및 스위치는 자신이 전달하는 패킷의 위변조 여부에 대해 판단할 수 없으므로 대응할 수 있는 유일한 방안은 발

생된 공격을 분석하여 공격 IP를 블랙리스트로 차단하는 방안이 유일하다.

④ 서비스 거부(DoS : Denial of Service) 공격

DoS 공격은 정상적인 서비스의 부하량을 늘리는 기법으로 공격이 매우 단순하며 공격자를 추적하기 어렵다. 효과가 즉시 나타나며 정상 접속과의 구분이 어렵다.

IPv4 프로토콜에서는 DoS 공격에 대해서는 완벽한 해결책이 없는 상태이다. 네트워크 트래픽 관리와 구성을 통하여 감소시킬 수는 있다. 네트워크는 가능한 VLAN과 서브넷 등으로 나눠서 공격자가 하나의 망으로 알 수 없도록 구성하는 것이 적절하다.

2. 메모리 주소 A, B, C, D에 1, 2, 3, 4가 순서대로 저장되어 있다. 아래의 명령어 실행 후 메모리 주소 E에 저장되는 결과는? (3.7점)

```
MOVE  R1, B
MOVE  R2, C
MUL   R1, R2
ADD   R1, A
ADD   R1, D
MOV   E, R1
```

① 8 ② 9 ③ 10 ④ 11 ⑤ 12

[2번 문항 정답] ④

[2번 문항 해설] 길라잡이 55페이지

문제에서 제시한 것은 2-번지 명령어 형식이다.

OP-Code	Operand 1(자료1의 주소)	Operand 2(자료2의 주소)

- 2-번지 명령어는 Operand부가 2개로 구성되며 컴퓨터에서 가장 많이 사용된다.
- Operand 1 의 원시 자료가 파괴된다.

MOVE R1, B	R1 ← B(2)
MOVE R2, C	R2 ← C(3)
MUL R1, R2	R1 ← R1(2) * R2(3)
ADD R1, A	R1 ← R1(6) + A(1)
ADD R1, D	R1 ← R1(7) + D(4)
MOV E, R1	E ← R1(11)

3. 캐시 접근시간이 100㎳이고 주기억장치 접근시간이 200㎳이다. 캐시 적중률이 40%일 때의 캐시 기반 평균 메모리 접근시간을 s1, 캐시 적중률이 80%일 때의 캐시 기반 평균 메모리 접근시간을 s2라고 할 때, s1과 s2의 값을 순서대로 나타낸 것은? (3.8점)

① 180㎳, 120㎳ ② 160㎳, 120㎳ ③ 120㎳, 160㎳
④ 140㎳, 180㎳ ⑤ 180㎳, 140㎳

[3번 문항 정답] ②
[3번 문항 해설] 길라잡이 65페이지

캐시 기반 평균 메모리 접근시간을 구하는 방법은 다음과 같다.
평균 메모리 접근시간 = 적중 시의 시간 + 실패 시의 시간
= 캐시기억장치 접근시간 × 적중률 + 주기억장치 접근시간 × (1-적중률)

- 캐시 적중률이 40%일 때
s1 = 100 × 0.4 + 200 × 0.6 = 40 + 120 = 160
- 캐시 적중률이 80%일 때
s2 = 100 × 0.8 + 200 × 0.2 = 80 + 40 = 120

4. 인터럽트에 대한 설명으로 옳은 것은? (4점)
① 오버플로우나 0으로 나누기가 발생한 경우 외부 인터럽트가 발생한다.
② 프로그램 처리 중 명령의 요청에 의해 내부 인터럽트가 발생한다.
③ 인터럽트의 종류는 외부 인터럽트와 내부 인터럽트로만 나뉜다.
④ 프로그램 실행 중 예상치 않은 상황으로 인해 실행 중인 작업을 중단하고 발생한 상황을 처리한다.
⑤ 하드웨어 방식의 인터럽트 우선순위 판별은 인터럽트 요청 플래그를 우선순위 별로 검사하여 서비스 루틴을 수행한다.

[4번 문항 정답] ④
[4번 문항 해설] 길라잡이 60~62페이지

- 인터럽트의 정의
인터럽트는 프로그램을 실행하는 도중에 예기치 않은 상황이 발생할 경우 현재 실행중인 작업을 즉시 중단하고 발생된 상황을 우선 처리한 후 실행 중이던 작업으로 복귀하여 계속 처리하는

것을 말한다.

- 인터럽트의 종류 및 발생원인

① 외부인터럽트

- 전원이상 인터럽트
- 기계착오 인터럽트 : CPU의 기능적인 오류 동작이 발생한 경우
- 외부 신호 인터럽트 : 타이머에 의해 규정한 시간을 알리는 경우, 키보드로 인터럽트키를 누른 경우, 외부장치로부터 인터럽트 요청이 있는 경우
- 입출력 인터럽트 : 입출력오류발생, 입출력장치가 데이터의 전송을 요구하거나 전송이 끝났음을 알릴 경우

② 내부인터럽트 : 프로그램 검사 인터럽트

- 0으로 나누기가 발생한 경우
- Overflow 또는 Underflow가 발생한 경우
- 프로그램에서 명령어를 잘못 사용한 경우
- 부당한 기억장소의 참조와 같은 프로그램 상의 오류

③ 소프트웨어 인터럽트 : 프로그램 처리 중 명령의 요청에 의해 발생하는 것으로 대표적인 형태는 감시프로그램을 호출하는 SVC(Supervisor Call)인터럽트가 있다.

④ 인터럽트 우선 순위

전원 > 기계착오 > 외부신호 > 입출력 > 명령어잘못 > 프로그램 검사 > SVC

5. 다음 식을 불대수의 기본 정리를 이용하여 **최대로 간소화**한 것은? (3.9점)

$$\overline{(A+B+C)}+B\cdot C+A\cdot \overline{B}\cdot \overline{C}+\overline{(\overline{B}+C)}$$

① $B\cdot C+\overline{B}$

② $\overline{A}\cdot \overline{B}\cdot \overline{C}+B\cdot C$

③ $\overline{B}\cdot \overline{C}+B$

④ $\overline{B}\cdot \overline{C}+\overline{A}\cdot B+A\cdot B$

⑤ $A\cdot \overline{C}+B$

[5번 문항 정답] ③

[5번 문항 해설] 길라잡이 71~73페이지

불대수의 기본 정리

- 교환법칙 : $A+B=B+A,\ A \cdot B=B \cdot A$
- 결합법칙 : $A+(B+C)=(A+B)+C,\ A \cdot (B \cdot C)=(A \cdot B) \cdot C$
- 분배법칙 : $A \cdot (B+C)=A \cdot B+A \cdot C,\ A+(B \cdot C)=(A+B) \cdot (A+C)$
- 동일법칙 : $A+A=A,\ A \cdot A=A$
- 보수법칙 : $A+\overline{A}=1,\ A \cdot \overline{A}=0$
- 항등법칙 : $A+0=A,\ A+1=1,\ A \cdot 0=0,\ A \cdot 1=A$
- 드모르간의 법칙 : $\overline{A+B}=\overline{A} \cdot \overline{B},\ \overline{A \cdot B}=\overline{A}+\overline{B}$

$\overline{(A+B+C)}+B \cdot C+A \cdot \overline{B} \cdot \overline{C}+\overline{(\overline{B}+C)}$

$= \overline{A} \cdot \overline{B} \cdot \overline{C}+B \cdot C+A \cdot \overline{B} \cdot \overline{C}+B \cdot \overline{C}$

$= (\overline{A}+A) \cdot \overline{B} \cdot \overline{C}+B \cdot (C+\overline{C})$

$= \overline{B} \cdot \overline{C}+B$

6. 가상기억장치 관리기법 중 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체하는 것은? (3.6점)

① FIFO(First In First Out)　② LFU(Least Frequently Used)

③ LRU(Least Recently Used)　④ NUR(Not Used Recently)

⑤ 최적교체

[6번 문항 정답] ⑤

[6번 문항 해설] 길라잡이 91페이지

① FIFO(First In First Out)

페이지가 교체될 때마다 주기억장치에 가장 오래 있었던 페이지부터 교체하는 방법

참조열	2	3	2	1	5	2	3	5	4	3	2	1
페이지 프레임	2	2	2	2	5	5	5	5	4	4	4	4
		3	3	3	3	2	2	2	2	2	2	1
				1	1	1	3	3	3	3	3	3
페이지부재	F	F		F	F	F	F		F			F

② LFU(Least Frequently Used)

사용빈도가 가장 적은 페이지를 교체한다. 활발하게 사용되는 페이지는 교체되지 않고 사용된

다. 실행 초기에 많이 사용된 페이지가 그 후로 사용되지 않을 경우 프레임을 계속 차지할 수 있다.

참조열	2	3	2	1	5	2	3	5	4	3	2	1
페이지 프레임	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	5	5	5	5	5	5	5	1
				1	1	1	3	3	4	3	3	3
페이지부재	F	F		F	F		F		F	F		F

[

③ LRU(Least Recently Used)

현 시점에서 가장 오랫동안 사용되지 않은 페이지를 제거하는 방법으로 참조된 시간을 기록해야 하므로 시간 오버헤더가 발생하고 실제 구현은 복잡하다.

참조열	2	3	2	1	5	2	3	5	4	3	2	1
페이지 프레임	2	2	2	2	2	2	2	2	4	4	4	1
		3	3	3	5	5	5	5	5	5	2	2
				1	1	1	3	3	3	3	3	3
페이지부재	F	F		F	F		F		F		F	F

④ NUR(Not Used Recently)

LRU와 비슷한 알고리즘으로 최근에 사용하지 않은 페이지를 교체하는 기법이다. 최근의 사용 여부를 확인하기 위해 참조비트와 변형비트를 두어 두 비트의 값에 따라 교체 순서가 결정된다.

7. 패리티 검사 방식 중 수직 중복 검사 방식을 적용했을 때, 빈칸 ㉠에 들어갈 패리티 비트로 옳은 것은? (3.8점)

㉠							
0	1	0	1	1	1	0	1
0	0	0	0	1	1	1	0
1	0	1	0	1	0	1	0
1	1	1	0	0	1	0	0
0	0	0	0	1	0	1	0
0	0	0	1	0	0	1	0
1	0	1	0	1	0	1	1

① 10101110 ② 01011110 ③ 10101010
④ 01011011 ⑤ 01010000

[7번 문항 정답] ①

[7번 문항 해설] 길라잡이 100페이지

패리티 검사 방식에는 수직 중복 검사(Vertical Redundancy Check), 세로 중복 검사(Longitudinal Redundancy Check), 순환 중복 검사(Cyclic Redundancy Check), 검사 합(Checksum)으로 구분할 수 있으며 검사 합을 제외한 나머지는 데이터 링크층에서 사용하기 위해 물리 계층에서 구현된다. 이 중 순환 중복 검사 방식은 다항식을 사용하는 방식으로 동기식 전송에서 주로 사용하는 방식이다.

· 전송 오류의 검출을 위해 사용한다.
· 잉여 비트-패리티 비트를 사용한다.
· '1'의 개수를 센다.
· 전송 비트 내의 '1'의 개수가 짝수 또는 홀수 개가 되도록 결정한다.
· **수직 중복 검사 방식은 짝수 개가 되도록 한다.**
· 전송 효율은 매우 좋다.
· 짝수 개의 비트가 오류가 발생할 경우 오류 검출률이 낮아지는 단점이 있다.

1	0	1	0	1	1	1	0
0	1	0	1	1	1	0	1
0	0	0	0	1	1	1	0
1	0	1	0	1	0	1	0
1	1	1	0	0	1	0	0
0	0	0	0	1	0	1	0
0	0	0	1	0	0	1	0
1	0	1	0	1	0	1	1

8. 명제인 것만을 <보기>에서 있는 대로 고른 것은? (3.7점)

<보기>

ㄱ. 모든 3의 배수는 홀수이다.
ㄴ. 배구 선수들은 키가 크다.
ㄷ. $x^3+3x^2+3x+1=0$

① ㄱ ② ㄴ ③ ㄱ, ㄷ ④ ㄴ, ㄷ ⑤ ㄱ, ㄴ, ㄷ

[8번 문항 정답] ①

[8번 문항 해설] 길라잡이 111~114페이지

명제란 참(true) 또는 거짓(false)을 분명하게 판별할 수 있는 문장을 말한다. 보통 명제는 p, q, r,… 등과 같이 소문자로 나타낸다. 명제가 참일 때는 T(혹은 1), 거짓일 때는 F(혹은 0)를 사용한다.

다음 문장들이 명제인지 판별해보면,

· 2 + 3 = 5 ➡ 참이므로 명제이다

· 1 + 2 = 4 ➡ 거짓이므로 명제이다.

· $2x-4=0$ ➡ x에 대한 조건이 없어서 참, 거짓을 판별할 수 없으므로 명제가 아니다.

ㄱ. 모든 3의 배수는 홀수이다. : 거짓인 명제이다. (반례: 3의 배수 중 6은 짝수)

ㄴ. 배구 선수들은 키가 크다. : 명제가 아니다. (‘키가 크다’의 기준이 없으므로 판별할 수 없음)

ㄷ. $x^3+3x^2+3x+1=0$: 명제가 아니다. (x에 대한 조건이 없어서 참, 거짓을 판별할 수 없음)

9. 다음 수식의 계산 결과는? (3.6점)

$1110_{(2)}$ + $15_{(16)}$ + $13_{(8)}$

① 40 ② 42 ③ 44 ④ 46 ⑤ 48

[9번 문항 정답] ④

[9번 문항 해설] 길라잡이 122~128페이지

<풀이1>

$1110_{(2)}=1\times2^3+1\times2^2+1\times2^1+0\times1=8+4+2+0=14$

$15_{(16)}=1\times16^1+5\times1=16+5=21$

$13_{(8)}=1\times8^1+3\times1=8+3=11$

$\therefore 1110_{(2)}+15_{(16)}+13_{(8)}=14+21+11=46$

<풀이2>

$1110_{(2)}+10101_{(2)}+1011_{(2)}=101110_{(2)}=46$

10. 2의 보수 표현에 대한 설명으로 옳은 것을 <보기>에서 있는 대로 고른 것은? (3.9점)

<보기>

ㄱ. 음수의 최상위비트는 1이다.
ㄴ. 1의 보수에 1을 더한 수이다.
ㄷ. 어떤 수 n과의 합이 1이 되는 수이다.
ㄹ. 30의 2의 보수 표현은 $11100001_{(2)}$이다.

① ㄱ, ㄴ ② ㄱ, ㄷ ③ ㄴ, ㄷ ④ ㄴ, ㄹ ⑤ ㄷ, ㄹ

[10번 문항 정답] ①
[10번 문항 해설] 길라잡이 128~134페이지

ㄱ. 옳다: 양수인 경우는 최상위 비트가 0, 음수인 경우는 최상위 비트가 1이다.
ㄴ. 옳다: 2진수에서 0에 대한 2의 보수는 2라 할 수 있는데 자리올림하면 결국 0에 대한 2의 보수는 0이 된다. 또한 1에 대한 2의 보수는 1이 된다. 따라서 2의 보수는 주어진 양의 정수의 2진 표현에 대해 1의 보수를 구한 뒤, 1을 더하면 된다.
ㄷ. 옳지 않다: 어떤 수 n과의 합이 2가 되는 수이다.
ㄹ. 옳지 않다: 30의 2의 보수 표현은 $11100010_{(2)}$이다.
(30의 2진 표현은 $00011110_{(2)}$이므로, 30의 1의 보수 표현은 $11100001_{(2)}$이고, 여기에 1을 더한 30의 2의 보수 표현은 $11100010_{(2)}$이다.)

11. 집합 A={1,2,3,4,6,12}에 관한 관계 R={(a,b) | (a mod b)∈A, a,b∈A}로 정의되어 있다. 다음에서 관계 R의 성질에 대해서 옳은 것만을 <보기>에서 있는 대로 고른 것은? (4.3점)

<보기>

ㄱ. 추이관계 ㄴ. 비반사관계 ㄷ. 반대칭관계

① ㄱ ② ㄴ ③ ㄱ, ㄷ ④ ㄴ, ㄷ ⑤ ㄱ, ㄴ, ㄷ

[11번 문항 정답] ②

[11번 문항 해설] 길라잡이 137~139페이지

R={(a,b) | (a mod b)∈A, a,b∈A}에서 b ∈ A={1,2,3,4,6,12}이므로,

i) b=1일 때, 모든 n∈A에 대하여 (n mod 1) = 0 ∉ A 이므로, (n,1) ∉ R

ii) b=2일 때,

a: 홀수이면 (a mod 2) = 1 ∈ A 이므로, (a,2) ∈ R

a: 짝수이면 (a mod 2) = 0 ∉ A 이므로, (a,2) ∉ R

따라서 a 의 값이 홀수 일 때만 집합 R에 속하므로, (1,2), (3,2) ∈ R

iii) b=3일 때,

(1 mod 3) = 1 ∈ A 이므로, (1,3) ∈ R

(2 mod 3) = 2 ∈ A 이므로, (2,3) ∈ R

(3 mod 3) = 0 ∉ A 이므로, (3,3) ∉ R

마찬가지로 3의 배수가 아니면 집합 R에 속하므로, (1,3), (2,3), (4,3) ∈ R

iv) b=4일 때,

(1 mod 4) = 1 ∈ A 이므로, (1,4) ∈ R

(2 mod 4) = 2 ∈ A 이므로, (2,4) ∈ R

(3 mod 4) = 3 ∈ A 이므로, (3,4) ∈ R

(4 mod 4) = 0 ∉ A 이므로, (4,4) ∉ R

마찬가지로 4의 배수가 아니면 집합 R에 속하므로, (1,4), (2,4), (3,4), (6,4) ∈ R

v) b=6일 때,

(1 mod 6) = 1 ∈ A 이므로, (1,6) ∈ R

(2 mod 6) = 2 ∈ A 이므로, (2,6) ∈ R

(3 mod 6) = 3 ∈ A 이므로, (3,6) ∈ R

(4 mod 6) = 4 ∈ A 이므로, (4,6) ∈ R

(6 mod 6) = 0 ∉ A 이므로, (6,6) ∉ R

(12 mod 6) = 0 ∉ A 이므로, (12,6) ∉ R

따라서 (1,6), (2,6), (3,6), (4,6) ∈ R

vi) b=12일 때,

(1 mod 12) = 1 ∈ A 이므로, (1,12) ∈ R

(2 mod 12) = 2 ∈ A 이므로, (2,12) ∈ R

(3 mod 12) = 3 ∈ A 이므로, (3,12) ∈ R

(4 mod 12) = 4 ∈ A 이므로, (4,12) ∈ R

(6 mod 12) = 6 ∈ A 이므로, (6,12) ∈ R

(12 mod 12) = 0 ∉ A 이므로, (12,12) ∉ R

따라서 (1,12), (2,12), (3,12), (4,12), (6,12) ∈ R

i), ii), iii), iv), v), vi) 에 의해

R={ (1,2), (3,2), (1,3), (2,3), (4,3), (1,4), (2,4), (3,4), (6,4), (1,6), (2,6), (3,6), (4,6), (1,12), (2,12), (3,12), (4,12), (6,12) }

={ (1,2), (1,3), (1,4), (1,6), (1,12), **(2,3)**, (2,4), (2,6), (2,12), **(3,2)**, (3,4), (3,6), (3,12), (4,3), (4,6), (4,12), (6,4), (6,12) } 이다.

ㄱ. 추이관계가 아니다.

(2,3), (3,2) ∈ R 이지만, (2,2) ∉ R 이므로, 관계 R은 추이관계가 아니다.

[추이관계]
A의 모든 원소에 대하여 aRb이고 bRc일 때 aRc이면, R은 추이관계(전이 관계 혹은 이행 관계)라고 한다.

ㄴ. 비반사관계이다.

모든 원소 a ∈ A={1,2,3,4,6,12} 에 대하여 (a,a) ∉ R 이므로, 관계 R은 비반사관계이다.

[비반사관계]
A의 모든 원소 a에 대하여 $a\not Ra$이면, R은 비반사관계라고 한다.

ㄷ. 반대칭관계가 아니다.

(2,3), (3,2) ∈ R 이지만, 2≠3 이므로, 관계 R은 반대칭관계가 아니다.

[반대칭관계]
A의 모든 원소에 대하여 aRb이고 bRa일 때 $a=b$이면 R은 반대칭관계라고 한다.

12. $n!$ 은 1부터 n까지의 정수를 연속적으로 곱한 값이다. 20! 의 오른쪽 끝에 연속되는 0의 개수는? (4점)

$n!$ 예시	오른쪽 끝에 연속되는 0의 개수
3! = 3×2×1 = 6	0
6! = 6×5×4×3×2×1 = 720	1
12! = 12×11×10×9×8×7×6×5×4×3×2×1 = 479,001,600	2

① 2 ② 3 ③ 4 ④ 6 ⑤ 12

[12번 문항 정답] ③
[12번 문항 해설] 길라잡이 146페이지

오른쪽 끝에 연속되는 0의 개수를 찾는 문제에 대해서 힌트를 찾아보자. 10을 소인수 표현으로 나타내어보면 10=5×2인데, 10이 되기 위해서는 소인수가 2와 5가 동시에 있어야 한다. 그중에서도 소인수 5의 개수를 확인하면 오른쪽 끝에 연속되는 0의 개수를 쉽게 찾을 수 있다.
(예) 6! = 6×**5**×4×3×2×1 에서 소인수 5가 1개 있으므로, 오른쪽 끝에 연속되는 0의 개수도 1개 (6! = 72**0**)
(예) 12! = 12×11×**10**×9×8×7×6×**5**×4×3×2×1 에서 소인수 5가 2개 있으므로, 오른쪽 끝에 연속되는 0의 개수도 2개 (12! = 479,001,6**00**)

이제 20! = 20×19×18×17×16×…×7×6×5×4×3×2×1 에서 소인수 5의 개수를 확인하자. 1과 20 사이의 수 중에서 소인수 5가 포함된 수를 찾아보면, 20, 15, 10, 5 임을 알 수 있다. 따라서 소인수 5가 4개 있으므로 오른쪽 끝에 연속되는 0의 개수도 4개이다.
(20! = 2,432,902,008,176,64**0,000**)

13. 다음은 스택의 push와 pop 연산을 구현한 것이다. 밑줄 친 ㉠, ㉡에 들어갈 내용으로 옳은 것은? (4점)

```
typedef struct{
 int stack[10];
 int top;
} Stack;

void init(Stack *s) {
     s->top = -1;
}

void push(Stack *s, int item) {
     if( s->top < 10 ) {
          s->stack[_____㉠_____] = item;
}

int pop(Stack *s) {
     if(_____㉡_____) {
          return s->stack[(s->top)--];
}
```

	㉠	㉡
①	s->top	s->top != 0
②	++(s->top)	s->top > 0
③	++(s->top)	s->top > -1
④	(s->top)++	s->top > -1
⑤	(s->top)++	s->top != 0

[13번 문항 정답] ③
[13번 문항 해설] 길라잡이 158~163페이지

스택(stack)은 쌓아 올린 더미를 의미하는 것으로, 대표적인 선형 자료구조 중 하나이다. 선형 자료구조란 자료를 접근하는 방식이 순차적으로 이루어지는 자료구조를 의미한다. 스택은 한쪽은 막혀있고 다른 한쪽은 뚫려있는 구조로 실제로 이런 구조가 존재하는 것이 아니라 이렇게 정의하여 사용을 한다. 한쪽이 막혀있는 선형 구조이므로 자료의 삽입과 삭제 작업은 한쪽 끝으로만 즉 막히지 않은 한쪽으로만 이루어지게 된다.

이런 입출력 형태를 후입 선출(LIFO: Last-In First-Out)이라고 한다. 스택은 이러한 후입 선출의 형식으로 입출력이 일어나는 자료구조이다. 스택은 제일 먼저 입력된 데이터가 맨 아래에

쌓이고 가장 최근에 입력된 데이터가 가장 위에 쌓이는 구조를 가지고 있으며, 스택의 중간에서는 데이터를 입출력할 수 없다.

스택을 구현하는 방법에는 배열을 이용하는 방법과 연결 리스트를 이용하는 방법이 있다. 배열로 구현하는 방법은 간단한 반면 스택의 크기가 고정되는 단점이 있다. 연결 리스트를 이용하는 방법은 구현이 약간 복잡한 반면, 스택의 크기를 필요에 따라 가변적으로 할 수 있다.

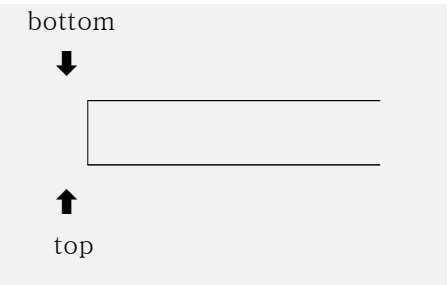
스택에는 두 가지 기본 연산이 있다. 하나는 삽입 연산이며, 또 하나는 삭제 연산으로 pop 연산이다. 스택에서는 자료가 어느 위치까지 저장되어 있는지 입구를 관리하기 위한 스택 포인터가 필요한데, 스택의 삽입과 삭제가 이루어지는 부분을 스택의 상단 top이라 하며, 반대쪽인 바닥부분을 스택의 하단 bottom이라고 한다.

1차원 배열로 구현한 스택을 가정하고 연산을 설명하면 다음과 같다. 자료가 스택에 삽입되어 push 될 때마다 top의 값은 1씩 증가하고 top이 가리키는 위치에 자료를 저장한다. 삭제에 의해 pop 될 때는 먼저 top이 가리키는 위치의 자료를 추출하고 난 뒤 top이 1만큼 감소된다.

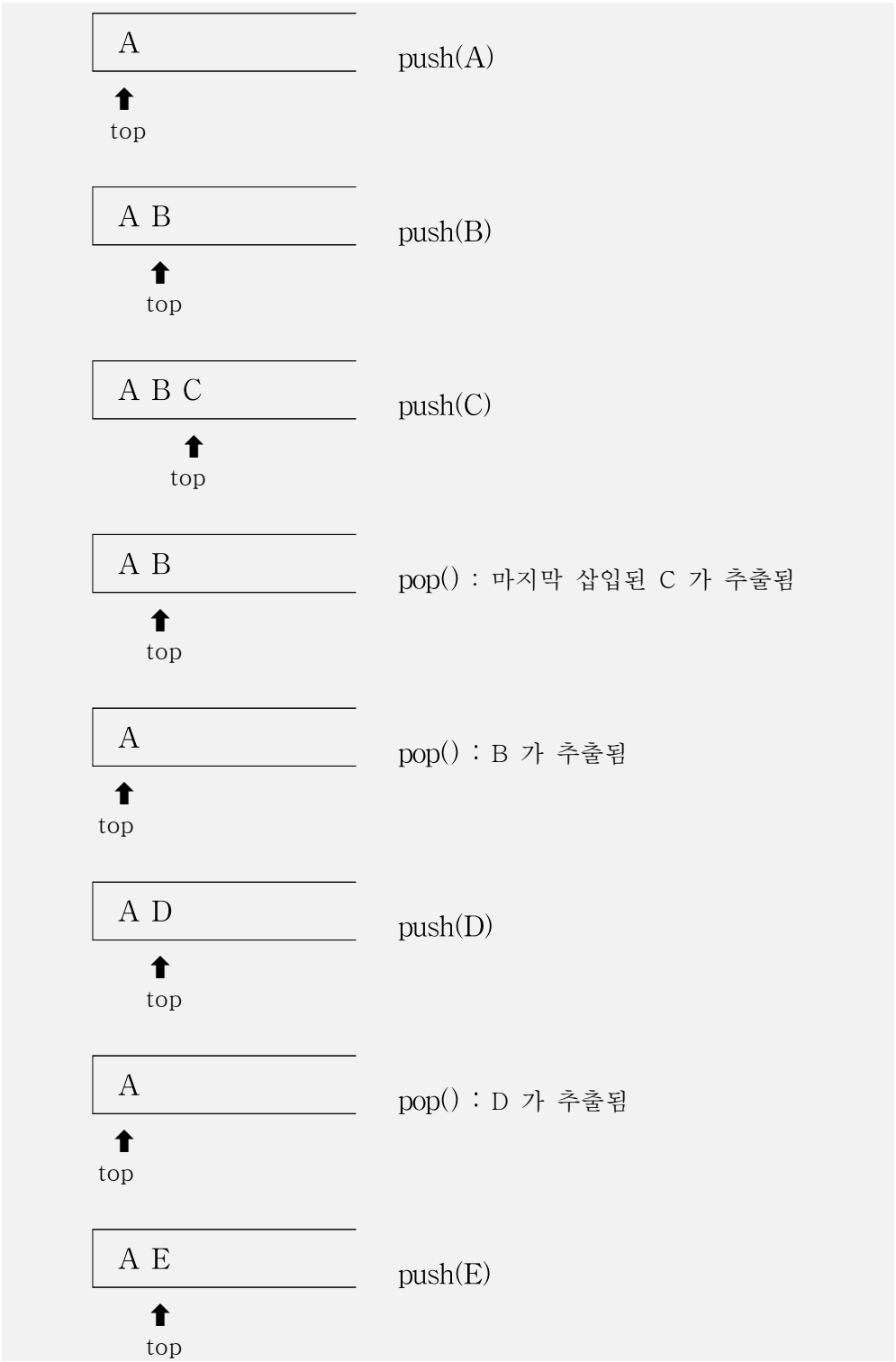
스택에 저장되는 것을 요소(element)라 부르고, top 이 초기값인 경우 즉 스택에 요소가 하나도 없는 비어있는 상태를 공백 스택(empty stack)이라 하고, top 이 스택의 최대 크기인 경우는 가득 찬 상태이다.

push(A), push(B), push(C), pop(), pop(), push(D), pop(), push(E)의 일련의 삽입과 삭제 연산은 아래 그림과 같이 이루어진다.

· 초기 상태 스택 : top과 bottom은 초기값 상태로 empty stack
(bottom의 위치는 일련의 연산 중 고정임)



· push(A), push(B), push(C), pop(), pop(), push(D), pop(), push(E)



14. 트리에 대해서 사용하는 몇 가지 표기법을 아래와 같이 정의한다.

- height(n): 노드 n을 루트로 하는 트리의 높이
- count(n): 노드 n을 루트로 하는 트리에 있는 모든 노드의 수
- n.left: 노드 n의 왼쪽 자식노드
- n.right: 노드 n의 오른쪽 자식노드

다음에서 완전이진트리에 관한 식으로 옳은 것은? (이 트리의 루트 노드는 r이다.) (4.1점)

① height(r.left) ≤ height(r.right)

② count(r) ≤ $2^{\text{height(r)}} - 1$

③ count(r.left) ≤ count(r.right)

④ count(r) ≤ 2 × count(r.left)

⑤ height(r) = height(r.right) + 1

[14번 문항 정답] ②

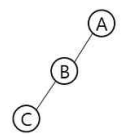
[14번 문항 해설] 길라잡이 169~170페이지

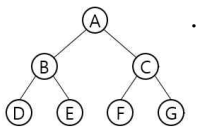
이진 트리의 성질

· n개의 노드를 가진 이진 트리는 n-1개의 간선을 가진다.

· 루트를 제외한 모든 노드는 오직 1개의 부모노드만 가지고, 부모와 자식 간에는 1개의 간선만 존재한다.

· 높이가 h인 이진 트리의 경우, 최소 h개의 노드를 가지며, 최대 2^h-1개의 노드를 가진다.

높이=3 [



. ..... $2^{1-1} = 1$
..... $2^{2-1} = 2$
..... $2^{3-1} = 4$

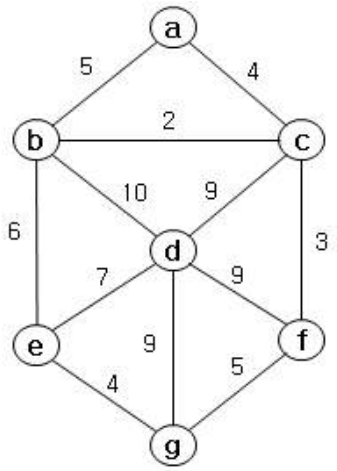
최소 노드 개수 = 3 최대 노드 개수 = 1 + 2 + 4 = 7

①, ③, ⑤의 경우 완전이진트리의 왼쪽 노드가 모두 채워지기 전에는 오른쪽 노드가 추가되지 않는다. 따라서 왼쪽 서브트리의 노드 개수가 오른쪽 서브트리의 노드 개수보다 적을 수 없다.

④ count(r) ≤ 2 × count(r.left) 는 항상 성립하지는 않는다.

- 포화이진트리일 때 count(r) = 2 × count(r.left) + 1 > 2 × count(r.left)

15. 최소비용신장트리를 구하는 Prim 알고리즘은 시작 정점에서부터 출발하여 신장트리 집합을 단계적으로 확장해나가는 방법이다. 시작 단계에서는 시작 정점만이 신장트리 집합에 포함되고, 앞 단계에서 만들어진 신장트리 집합에 인접한 정점들 중에서 최저 간선으로 연결된 정점을 선택하여 트리를 확장한다. 이 과정은 n개의 정점을 가진 트리가 n-1개의 간선을 가질 때까지 계속된다.



위의 그래프에 대하여 최소비용신장트리를 Prim의 방법으로 구할 때, 6번째로 선택되는 정점은? (단, 시작 정점 a가 첫 번째로 선택되는 정점이다.) (3.9점)

① b ② c ③ d ④ e ⑤ f

[15번 문항 정답] ④

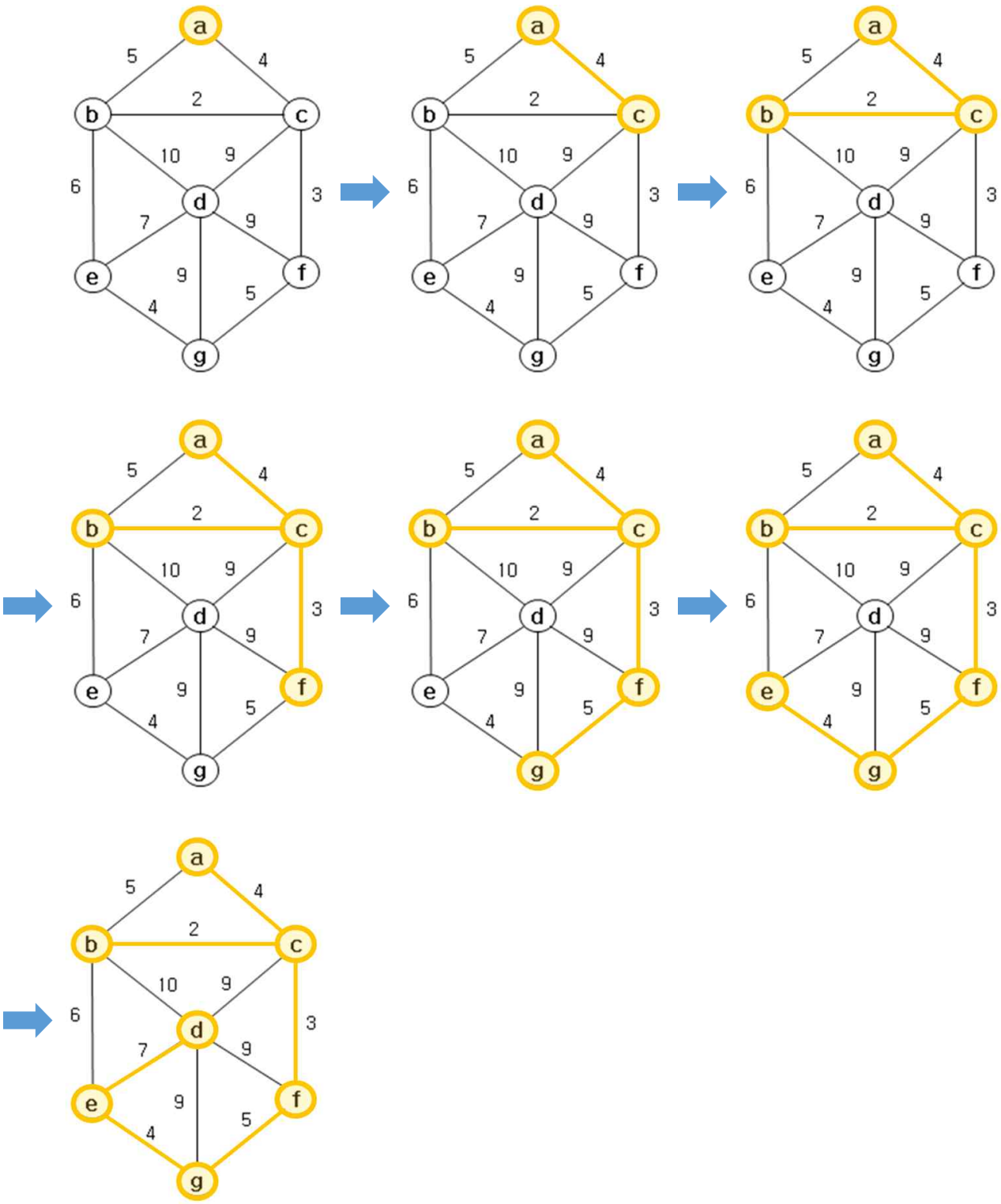
[15번 문항 해설] 길라잡이 177페이지

Prim 알고리즘은 시작 정점에서부터 출발하여 신장트리 집합을 단계적으로 확장해나가는 방법이다. 시작 단계에서는 시작 정점만이 신장트리 집합에 포함되고, 앞 단계에서 만들어진 신장트리 집합에 인접한 정점들 중에서 최저 간선으로 연결된 정점을 선택하여 트리를 확장한다. 이 과정은 트리가 n-1개의 간선을 가질 때까지 계속된다. n개의 정점을 가지는 신장 트리의 간선은 n-1개이기 때문이다.

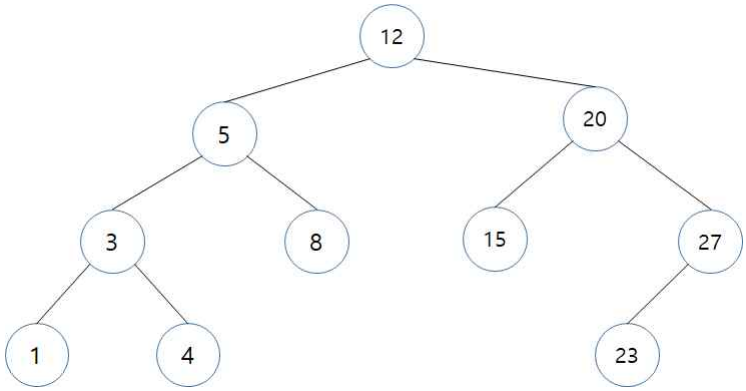
Kruskal 알고리즘과 비교를 해보면 첫 번째, Kruskal 알고리즘은 간선을 기반으로 하는 알고리즘인 반면 Prim 알고리즘은 정점을 기반으로 하는 알고리즘이라는 점이다. 또한 Kruskal 알고리즘에서는 전 단계에서 만들어진 신장트리와는 상관없이 무조건 최저 간선만을 선택하는 방법이었던데 반하여 Prim 알고리즘은 전 단계에서 만들어진 신장트리를 확장하는 방식이다. Prim 알고리즘은 이전 단계에서 만들어진 신장 트리 정보를 활용하므로 그 정보를 저장할 필요가 있다.

위 그래프에서 정점 a를 기점으로 하여 최소비용 신장트리를 Prim 알고리즘에 따라 확장해가

는 과정은 다음과 같다.



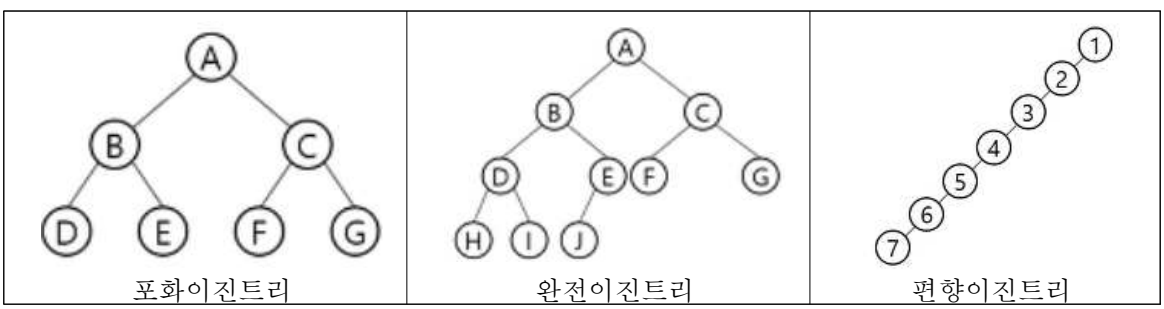
16. 다음 이진탐색트리에 11, 13, 14가 순서대로 삽입된 후 15가 삭제되었다. 이때 만들어진 트리에 대한 설명으로 옳은 것은? (4.1점)



① 높이는 5이다.
② 포화이진트리(full binary tree)이다.
③ 단말 노드들은 1, 4, 11, 14, 23이다.
④ 편향이진트리(skewed binary tree)이다.
⑤ 노드 삽입은 완전이진트리(complete binary tree)를 유지하도록 삽입된다.

[16번 문항 정답] ③
[16번 문항 해설] 길라잡이 167-168, 178-183페이지

높이가 h인 이진트리의 경우, 최소 h개의 노드를 가지며, 최대 2^h-1개의 노드를 가진다.
대표적인 이진트리의 종류는 다음과 같다.
- 포화이진트리(full binary tree) : 각 레벨에 노드가 꽉 차있는 이진트리
- 완전이진트리(complete binary tree) : 높이가 K일 때 레벨 1부터 K-1까지는 노드가 모두 채워져 있고, 마지막 레벨 K에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진트리
- 편향이진트리(skewed binary tree) : 최소 개수의 노드를 가지면서 왼쪽이나 오른쪽의 서브 트리만 가지고 있는 이진트리

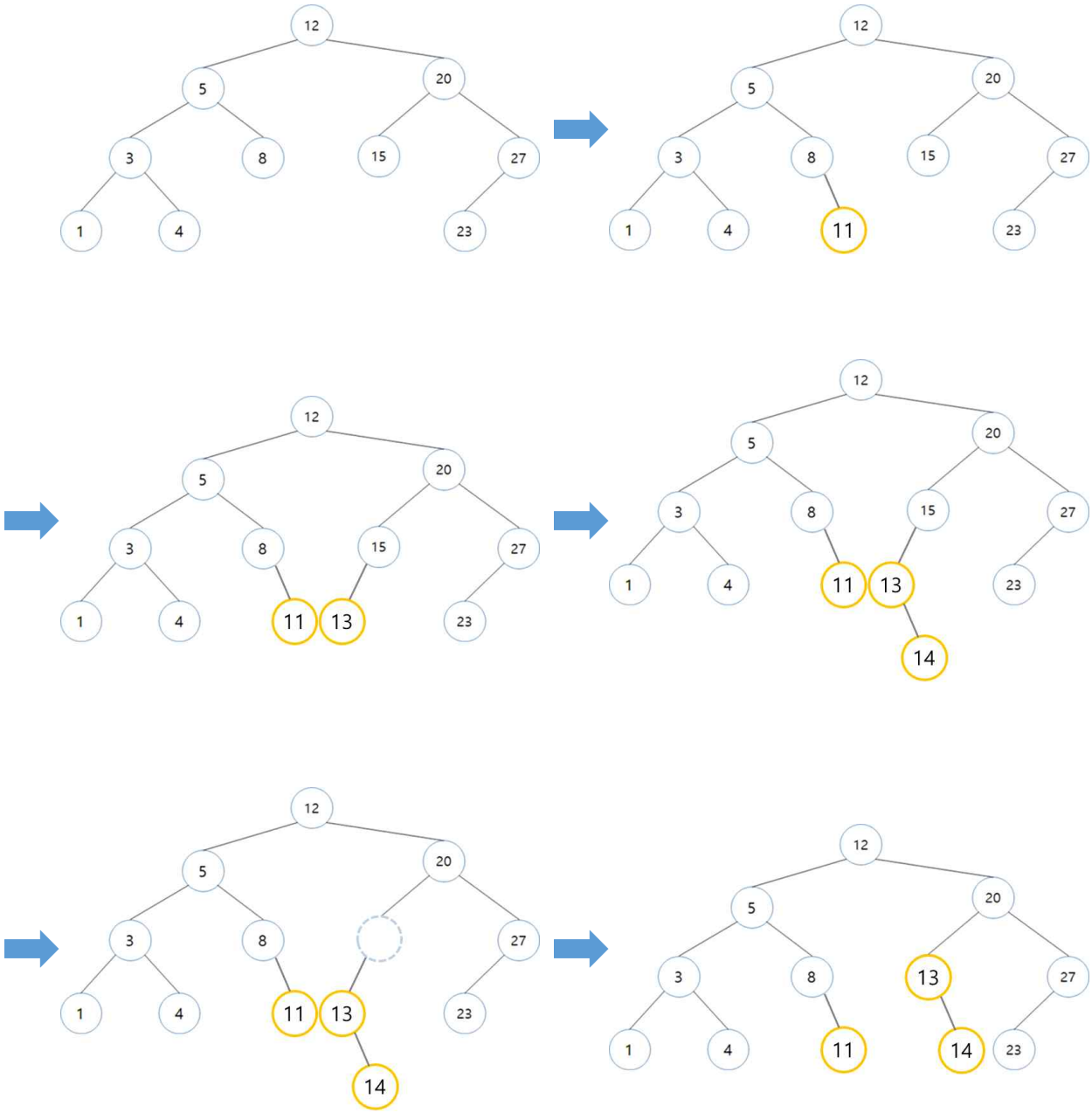


이진트리 중 이진탐색트리(binary search tree)는 이진트리 기반의 탐색을 위한 자료구조이다. 탐색은 우리의 일상생활에서 많이 사용되는데 전화번호부에서 전화번호를 찾거나, 사전에서 단어를 찾거나, 어떤 특정한 날에 선약이 없는가를 검사할 때 등 다양하게 사용된다. 탐색은 컴퓨터 응용 프로그램에서도 많이 사용되며, 가장 시간이 많이 걸리는 작업 중의 하나이므로 탐색을 가능한 효율적으로 수행하는 것이 무척 중요하다.
이진탐색트리는 루트의 값이 왼쪽 서브트리에 있는 임의의 노드의 값보다 크고 오른쪽 서브트리에 있는 임의의 노드의 값보다 작은 이진트리의 특수한 형태이다. 이진탐색트리가 만족해야할 성질에 대한 정의는 다음과 같다.

· 이진탐색트리의 노드에 저장된 키(key)는 유일하다.
· 왼쪽 서브트리의 키들은 그 서브트리의 루트의 키보다 작다.
· 오른쪽 서브트리의 키들은 그 서브트리의 루트의 키보다 크다.
· 왼쪽과 오른쪽 서브트리도 이진탐색트리이다.

이진탐색트리에 자료를 삽입하기 위해서는 먼저 탐색을 수행하는 것이 필요하다. 왜냐하면 이진탐색트리에서는 같은 키 값을 갖는 노드가 없어야 하기 때문이고 또한 탐색에 실패한 위치가 바로 새로운 노드를 삽입하는 위치가 되기 때문이다. 똑같은 자료라 하더라도 삽입되는 순서에 따라 이진탐색트리의 모양은 달라지며 새로운 노드가 삽입되는 위치는 단말노드가 된다.
노드를 삭제하는 것은 이진탐색트리에서 가장 복잡한 연산이다. 노드를 삭제하기 위해서는 먼저 노드를 탐색해야 한다는 것은 삽입과 마찬가지이다. 하지만 삭제하려고 하는 킷값이 트리 내의 어디에 있느냐에 따라 세 가지 경우를 고려해야 한다.
1. 단말노드를 삭제하는 것은 아래에 더 이상의 노드가 없으므로 해당 단말노드를 삭제하고 부모노드를 찾아서 부모노드 안의 포인터를 NULL로 만들어 연결을 끊으면 된다.
2. 삭제되는 노드가 왼쪽이나 오른쪽 서브트리 중 하나만 가지고 있는 경우에는 해당 노드를 삭제하고, 서브트리를 삭제된 노드의 부모노드에 붙여주면 된다.
3. 삭제되는 노드가 왼쪽과 오른쪽 서브트리 모두 가지고 있는 경우에는 해당 노드를 삭제하고, 대신 왼쪽 서브트리에서 가장 큰 노드나 오른쪽 서브트리에서 가장 작은 노드를 삭제된 노드의 위치로 대체하고 이진탐색트리가 형성되도록 순환적으로 과정을 진행하면 된다. 왼쪽 서브트리에서 가장 큰 값은 왼쪽 서브트리의 가장 오른쪽에 있는 노드이며, 오른쪽 서브트리에서 가장 작은 값은 오른쪽 서브트리의 가장 왼쪽에 있는 노드가 된다.

문제에 제시된 이진탐색트리에 노드가 삽입, 삭제되는 과정은 다음과 같다.



이진탐색트리의 최종 형태를 살펴보면 높이가 4이며, 자식노드가 하나밖에 없는 부모노드가 있으므로 포화이진트리(full binary tree, 정이진트리)의 형태로 볼 수 없다.
단말노드는 1, 4, 11, 14, 23이다.

17. 다음과 같은 빈도로 문자가 사용되었을 때, 각 문자의 허프만코드로 옳지 <u>않은</u> 것은? (4.1점)

문자	b	e	g	j	l	o	s	u	z
빈도수	18	8	27	13	36	2	20	6	3

① b: 011 ② g: 00 ③ l: 10 ④ s: 111 ⑤ z: 11010

[17번 문항 정답] ⑤
[17번 문항 해설] 길라잡이 188~191페이지

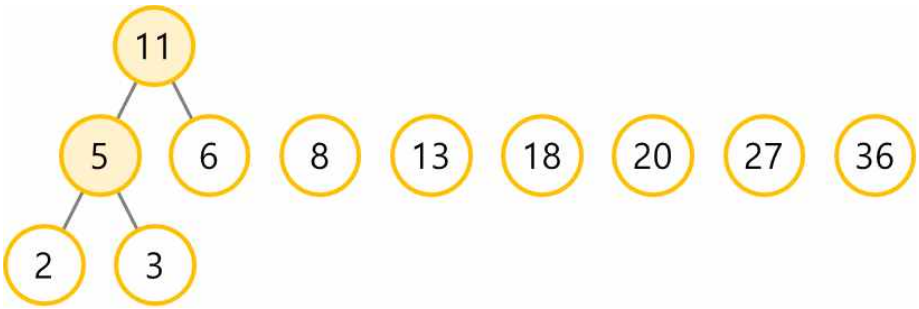
컴퓨터는 영숫자, 특수문자 등과 같은 문자 자료를 인코딩하여 0과 1의 2진 코드 형태로 저장한다. 인코딩하는 방법 중 가장 많이 사용되고 있는 아스키코드의 경우 고정길이 코드로서, 사용 빈도에 관계없이 모든 문자에 대해 같은 용량의 저장 공간을 필요로 한다. 각 문자의 빈도가 알려져 있다면, 각 문자에 고정된 길이의 비트를 할당하는 것보다 빈도에 따라 서로 다른 비트 수를 할당한 가변 길이 코드 방식을 사용한다면, 저장 공간을 줄이고 효율적으로 사용하는 것이 가능해진다. 이진트리는 각 글자의 빈도가 알려져 있는 메시지의 내용을 압축하는데 사용될 수 있다. 이런 특별한 종류의 이진트리를 허프만코딩트리라고 부르고, 루트노드로부터 단말노드까지의 경로에 있는 간선의 라벨값으로 부여된 코드를 읽으면 허프만코드가 된다. 즉 각 글자의 빈도수에 따라 서로 다른 비트 수가 할당된 코드가 만들어지게 된다.
허프만코드를 구성하는 알고리즘은 다음과 같다.

① 빈도수가 적은 문자에서 많은 문자 순으로 빈도수에 따라 문자를 나열한다.
② 가장 적은 빈도수를 가지는 문자 두 개를 추출하여 이들을 단말노드로 두고, 두 문자의 빈도수의 합을 루트노드로 하는 이진트리를 구성한다.
③ ②의 과정을 모든 문자의 빈도수의 합을 루트로 하는 이진트리가 완성될 때까지 반복한다.
④ 트리가 완성되면 루트노드에서 단말노드까지, 왼쪽 간선은 비트 0을, 오른쪽 간선은 비트 1을 부여하여 각 문자의 코드를 완성한다.

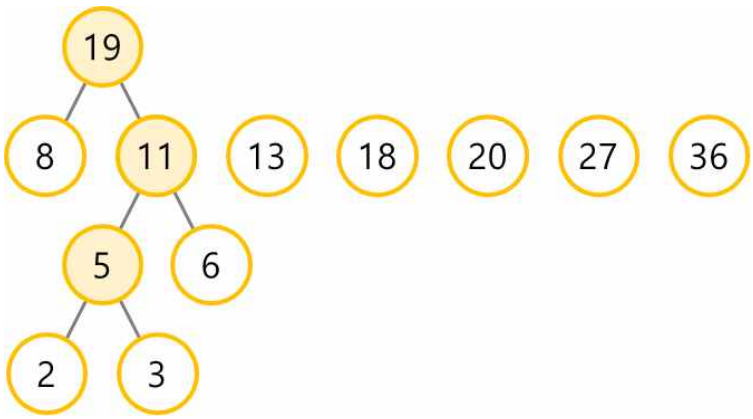
문제에 주어진 문자의 빈도수로 허프만코딩트리를 구성하는 과정은 다음과 같다.

단계 #1 : 빈도수에 따른 나열

단계 #2-1 : 적은 빈도의 두 노드를 선택해서 이진트리 구성

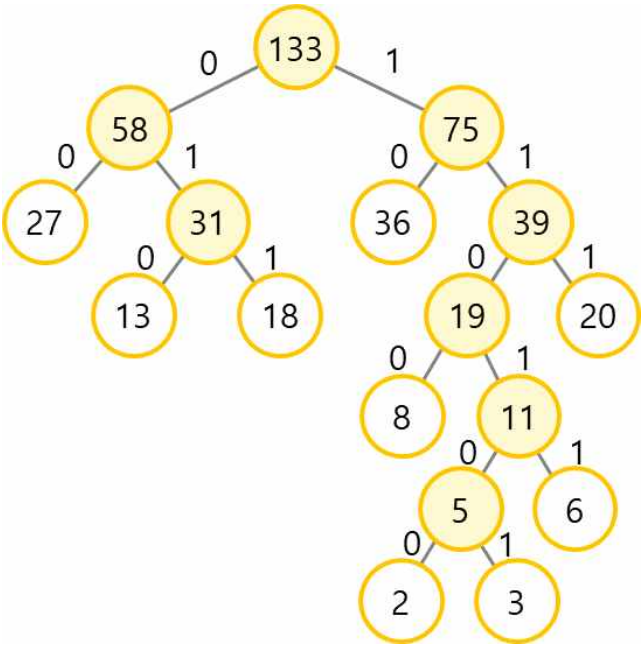


단계 #2-2 : 적은 빈도의 두 노드를 선택해서 이진트리 구성



단계 #2-3 : 적은 빈도의 두 노드를 선택해서 이진트리 구성

(중략)



단계 #3 : 비트 코드 부여

구성된 허프만 코딩트리에 나타난 문자별 허프만 코드는 아래와 같다.

문자	b	e	g	j	l	o	s	u	z
빈도수	18	8	27	13	36	2	20	6	3
허프만코드	**011**	1100	**00**	010	**10**	110100	**111**	11011	**110101**

18. 다음 프로그램 실행 결과로 출력되는 *의 개수는? (3.9점)

```
int main(void) {
 int i = 0, j = 0;
 for (i = 0; i < 10; i++) {
    if (i % 3 == 0)
       continue;
    if (i > 8)
       break;
    for (j = 0; j < i; j++)
       printf("*");
 }
}
```

① 26 ② 27 ③ 28 ④ 29 ⑤ 30

[18번 문항 정답] ②

[18번 문항 해설] C언어

for(i=0;i<10;i++) 수행과정	for문 안에 실행되는 식	printf("*")실행횟수
i=0	if (i % 3 == 0) continue;	0
i=1	for (j = 0; j < i; j++) printf("*");	1
i=2	for (j = 0; j < i; j++) printf("*");	2
i=3	if (i % 3 == 0) continue;	0
i=4	for (j = 0; j < i; j++) printf("*");	4
i=5	for (j = 0; j < i; j++) printf("*");	5
i=6	if (i % 3 == 0) continue;	0
i=7	for (j = 0; j < i; j++) printf("*");	7
i=8	for (j = 0; j < i; j++) printf("*");	8
i=9	if (i % 3 == 0) continue;	0

19. 다음 프로그램을 실행시켰을 때 출력값은? (4점)

```
#include <stdio.h>
#include <string.h>
void main() {
    int array[256];
    char *str = "Boyer-Moore algorithm";
    int len = strlen(str);
    int i, j = 0;
    for (i = 0; i < 256; i++) array[i] = 0;
    while (j < len) {
        array[str[j]] = j;
        j = j + 1;
    }
    printf("%d", array['o']);
}
```

① 1 ② 2 ③ 4 ④ 15 ⑤ 21

[19번 문항 정답] ④

[19번 문항 해설] C언어

str 안에 들어가 있는 문장에서 알파벳 'o'의 위치 중 마지막 위치를 찾는 프로그램이다.

프로그램의 초기값은 아래와 같다.

array	0	1	2	…	…	…	…	…	…	…	…	…	…	253	254	255								
	0	0	0	…	…	…	…	…	…	…	…	…	…	0	0	0								
str	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20			
	B	o	y	e	r	-	M	o	o	r	e		a	l	g	o	r	i	t	h	m			
len	21																							

while (j < len)	array[str[j]] = j;	반복문 해설
j = 0	array[str[0]] = 0;	array[str[0]] = array[‘B’]가 되며 대문자 ‘B’는 66이므로, array[66]에 str 안의 ‘B’위치인 0이 들어가게 된다. array[66] = 0
j = 1	array[str[1]] = 1;	array[str[1]] = array[‘o’]가 되며 소문자 ‘o’는 111이므로, array[111]에 str 안의 ‘o’위치인 1이 들어가게 된다. array[111] = 1
j = 2	array[str[2]] = 2;	array[str[2]] = array[‘y’]가 되며 소문자 ‘y’는 121이므로, array[121]에 str 안의 ‘y’위치인 2가 들어가게 된다. array[121] = 2
⋮	⋮	⋮
j = 15	array[str[15]] = 15;	array[str[15]] = array[‘o’]가 되며 소문자 ‘o’는 111이므로, array[111]에 str 안의 ‘o’위치인 15로 덮어쓰기 된다. array[111] = 15
⋮	⋮	⋮

20. 다음 프로그램을 식으로 표현한 것은? (4점)

```
#include <stdio.h>
void main() {
    int i, j, s, n;
    s = 0;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        for (j = 1; j <= 2; j++)
            s = s + i;
    printf("%d", s);
}
```

① $\sum_{k=1}^{n} k$ ② $\sum_{k=1}^{n} 2k$ ③ $\sum_{k=1}^{2n} k$ ④ $\sum_{k=1}^{n}\sum_{m=1}^{2} m$ ⑤ $\sum_{k=1}^{n}\sum_{m=1}^{n} m$

[20번 문항 정답] ②

[20번 문항 해설] 길라잡이 121페이지

위의 프로그램은 일정한 규칙을 가지고 연속적으로 나열되는 값들의 합을 나타내는 프로그램이며, 이렇게 일정 규칙으로 나열된 값의 합을 특정 기호인 $\sum$(시그마)로 표현가능하다.

합의 표시 $\sum_{i=a}^{b} i$

일정 규칙으로 나열된 값의 합(i : 합의 색인)

$$\sum_{k=1}^{n} k = 1+2+3+\ldots+n$$

프로그램 실행 후 입력이 4일 때,

s	0
n	4

for (i = 1; i <= n; i++)을 변수 i의 변화에 따라 살펴보면 다음과 같다.

for (i = 1; i <= n; i++)	for문 수행과정	s의 값

i=1	for (j = 1; j <= 2; j++)
	s = s + i;
	의 실행에 의해서
	s = (s + i) + i
	= s + 1 + 1 = 6
		2

i=2	for (j = 1; j <= 2; j++)
	s = s + i;
	의 실행에 의해서
	s = (s + i) + i
	= s + 2 + 2 = 6
		6

i=3	for (j = 1; j <= 2; j++)
	s = s + i;
	의 실행에 의해서
	s = (s + i) + i
	= s + 3 + 3 = 12
		12

i=4	for (j = 1; j <= 2; j++)
	s = s + i;
	의 실행에 의해서
	s = (s + i) + i
	= s + 4 + 4 = 20
		20

위와 같이 s=(1+1) + (2+2) + (3+3) + … + (n+n)이므로 이를 $\sum_{k=1}^{n} 2k$로 표현할 수 있다.

21. 다음은 삽입 정렬 프로그램이다. 밑줄 친 ㉠, ㉡에 들어갈 내용으로 옳은 것은?
(단, 오름차순으로 정렬된다.) (4.1점)

```
void insertion_sort(int list[], int size) {
  int i, j, temp, a;
  for (i = 1; i < size; i++) {
      temp = list[i];
      j = i;
      while (_____㉠_____)
          list[j + 1] = list[j];
      _____㉡_____;
      for (a = 0; a < size; a++)
          printf("%d ", list[a]);
      printf("\n");
  }
}
int main(void) {
  int list[5] = { 5, 3, 1, 8, 7 };
  insertion_sort(list, 5);
}
```

	㉠	㉡
①	--j >= 0 && temp < list[j]	list[j] = temp
②	j-- >= 0 && temp > list[j]	list[j - 1] = temp
③	--j >= 0 && temp > list[j]	list[j] = temp
④	j-- >= 0 && temp < list[j]	list[j + 1] = temp
⑤	--j >= 0 && temp < list[j]	list[j + 1] = temp

[21번 문항 정답] ⑤
[21번 문항 해설] 길라잡이 215~217페이지

삽입 정렬(insertion sort)은 현재의 값을 이미 정렬되어 있는 값들과 차례대로 비교하여 적당한 위치에 삽입하는 것으로 주어진 값 중 처음 하나의 값을 시작으로 차례대로 삽입하면서 자신의 자리를 찾아가는 방식이다.

① 기준 값의 앞은 이미 정렬된 상태이며 기준 값을 이 정렬된 리스트에 삽입한다.
② 정렬된 리스트의 값과 기준 값을 하나씩 비교하여 기준 값보다 크면 한 자리씩 뒤로 밀어준다.
③ 만약 기준 값보다 작은 값이 나타나면 작은 값 뒤에 기준 값을 삽입한다.
④ 두 번째 값부터 기준으로 마지막 값까지 ① ~ ③ 과정을 반복한다.

위 프로그램은 삽입 정렬을 C언어로 구현한 예제이다. 값을 하나씩 앞에 있는 값과 비교하여 알맞은 위치에 삽입한다. 각 단계별 처리 과정은 다음 그림과 같다.

```
void insertion_sort(int list[], int size) {
 int i, j, temp, a;
 for (i = 1; i < size; i++) {      // i를 1부터 4까지 반복한다.
    temp = list[i];                // temp에 list[i]의 값을 넣는다.
    j = i;                         //  j에 i의 값을 넣는다.

    while (--j >= 0 && temp < list[j]) // j의 값을 감소시키면서
       list[j + 1] = list[j];   //list[i] 값보다 작은 값이 있을 때까지 반복한다.

    list[j + 1] = temp;          // 작은 값의 위치 다음에 list[i] 값을 삽입한다.

    for (a = 0; a < size; a++)
       printf("%d ", list[a]);    // 정렬된 과정을 출력한다.
    printf("\n");
 }
}

int main(void) {
 int list[5] = { 5, 3, 1, 8, 7 }; // list 배열을 선언하고 정렬할 자료를 입력한다.
 insertion_sort(list, 5);          // 삽입정렬함수를 호출한다.
}
```

list	5	3	1	8	7
i=1	3	5	1	8	7
i=2	1	3	5	8	7
i=3	1	3	5	8	7
i=4	1	3	5	7	8

22. 다음 프로그램을 실행시켰을 때 출력값은? (4.3점)

```
#include <stdio.h>
int A(int m, int n) {
    if (m == 0) return n+1;
    if (n == 0) return A(m-1, 1);
    else return A(m-1, A(m, n-1));
}
void main() {
    printf("%d", A(2, 3));
}
```

① 5 ② 6 ③ 7 ④ 8 ⑤ 9

[22번 문항 정답] ⑤

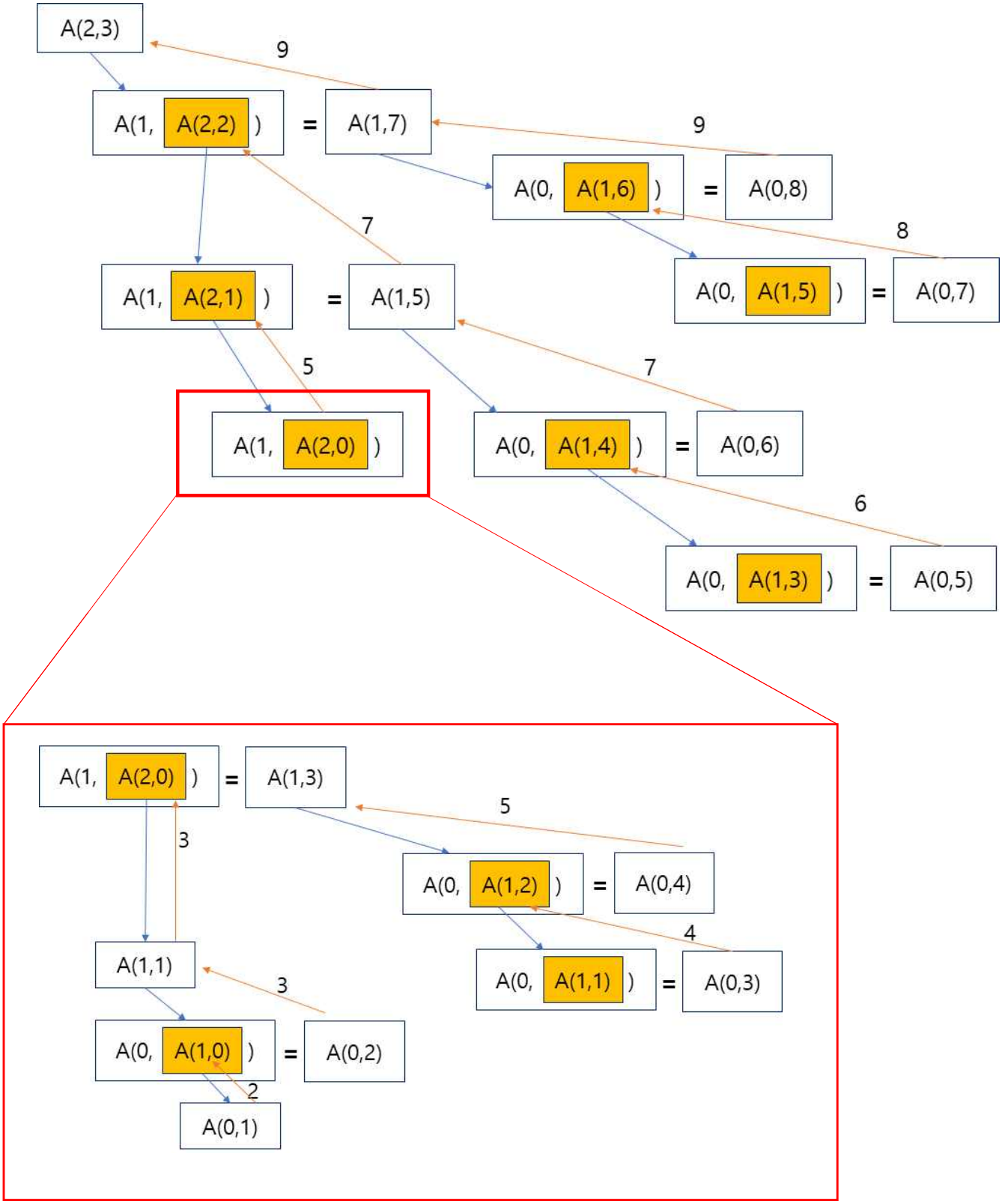
[22번 문항 해설] 길라잡이 246~248페이지

위의 프로그램은 재귀함수를 이용한 프로그램이다.

이 경우 실제로 어떻게 프로그램이 실행되는지 재귀호출 과정을 살펴보아야 한다.

main() 안의 printf()가 실행되면서 A(2, 3)을 호출하게 된다.

함수 호출을 구조화하면 다음과 같다.



23. 다음 프로그램을 실행시켰을 때 출력값은? (4.4점)

```
#include <stdio.h>
int rsa(int a, int b, int c) {
    int r;
    if (b == 1)
        return a % c;
    if (b%2 == 0) {
        r = rsa(a, b/2, c);
        return r*r % c;
    }
    else {
        r = rsa(a, b-1, c);
        return a*r % c;
    }
}
void main() {
    int ret;
    ret = rsa(120, 33, 37);
    printf("%d", ret);
}
```

① 10 ② 12 ③ 16 ④ 33 ⑤ 34

[23번 문항 정답] ①

[23번 문항 해설] 길라잡이 246~248페이지

$$120 * rsa(120,32,37)\%37 = 10$$

$$rsa(120,16,37) * rsa(120,16,37)\%37 = 34$$

$$rsa(120,8,37) * rsa(120,8,37)\%37 = 16$$

$$rsa(120,4,37) * rsa(120,4,37)\%37 = 33$$

$$rsa(120,2,37) * rsa(120,2,37)\%37 = 12$$

$$rsa(120,1,37) * rsa(120,1,37)\%37 = 7$$

$$120\%37 = 9$$

24. n×n 체스판에 n개의 퀸(queen)을 배치하고자 한다. 퀸들은 같은 행이나, 같은 열이나, 같은 대각선상에 위치하지 않아야 한다. 이를 백트래킹으로 해결하고자 한다. 다음은 프로그램의 의사코드(pseudo code)일 때, promising 함수의 밑줄 친 ㉠에 들어갈 내용으로 옳은 것은? (4.5점)

```
// col[i] = i번째 행에서 퀸이 놓여있는 열
void queens(index i) {
     index j;
     if(promising(i))
          if(i == n)
               print col[1] through col[n];
          else
          for(j=1;j <= n;j++) {
              col[i+1] = j;
              queens(i+1);
          }
}
bool promising(index i) {
     index k = 1;
     bool switch = TRUE;
   while(k < i && switch) {
        if(_____㉠_____)
             switch = FALSE;
             k++;
   }
   return switch;
}
```

① col[i] == col[k] || |col[i]+col[k]| == |i+k|

② col[i] != col[k] && |col[i]-col[k]| == |i-k|

③ col[i] != col[k] || |col[i]-col[k]| == |i-k|

④ col[i] == col[k] && |col[i]-col[k]| == |i-k|

⑤ col[i] == col[k] || |col[i]-col[k]| == |i-k|

[24번 문항 정답] ⑤

[24번 문항 해설] 길라잡이 259~261페이지

백트래킹(Backtracking)

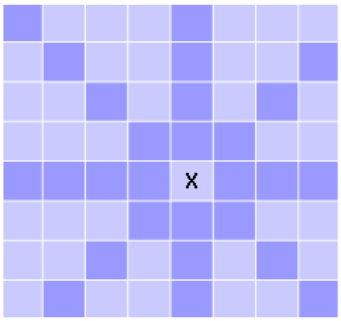
미로 찾기를 할 때 갈림길을 만나면 그중 하나를 선택하여 따라간다. 벽이 나오면 다시 되돌아가서 갈림길 중 가보지 않은 다른 길로 간다.

이와 같은 방법을 반복하다 보면 결국 출구에 도착하게 된다. 이렇게 모든 경우를 탐색하는 방

법을 백트래킹이라고 하는데 모든 경우를 탐색하므로 많은 시간이 걸리긴 하지만 아주 특별한 경우를 제외하고는 꼭 답이 나오게 된다.

모든 경우를 탐색하는 데 많은 시간이 걸리므로 시간 초과에 걸리는데 미로 찾기와 같이 벽이 있는 길을 미리 알아내서 그런 경우는 탐색하지 않는 설정을 해주어 시간을 절약할 수 있다.

백트래킹 하면 가장 많이 등장하는 체스 문제에 대해 생각해 보자. 체스에서 queen의 가로, 세로, 대각선 방향으로 어느 곳이나 한 번에 움직일 수 있다. 즉 다음과 같은 체스판에서 queen이 X라고 표시된 위치에 있을 때, 그다음 queen이 움직여 갈 수 있는 부분은 어둡게 칠해진 부분 중의 하나이다.



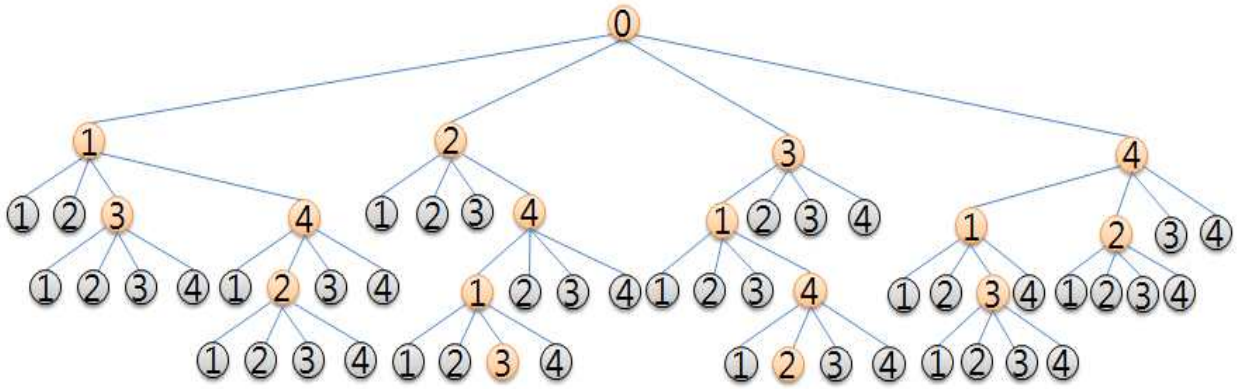
[체스 queen의 행마]

N X N 크기의 정방형 체스판이 주어졌다. 우리는 여기에 N개의 queen을 배치하려고 하는데, 모든 queen들은 서로 잡아먹을 수 없어야 한다. 그렇다면 queen들을 어떻게 배치해야만 할까? 가능한 모든 경우의 개수를 출력하려고 한다.

먼저 상태 공간 트리를 작성하여 문제 상태를 이해한다. 아래의 상태 공간 트리에서 보면 1행에서는 1열, 2열, 3열, 4열을 모두 선택할 수 있다.

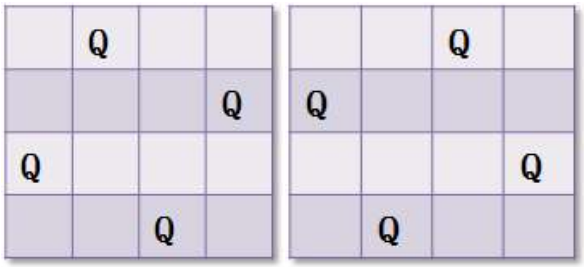
1행에서 1열을 선택한 경우 2행에서 1열은 1행에서 이미 선택하였으므로 선택할 수 없고 2열은 1행에서 선택한 1열과 대각선상에 있기 때문에 선택할 수 없다. 3, 4열을 선택할 수 있는데 3열을 선택하고 나면 3행에서는 선택할 수 있는 자리가 없다. 2행에서 4열을 선택하면 3행에서는 2열을 선택할 수 있으나 4행에서 선택할 수 있는 경우가 없다.

이런 과정을 거쳐 가로, 세로, 대각선으로 같은 선상에 놓이는 경우가 없는 것은 1행에서 2열, 2행에서 4열, 3행에서 1열, 4행에서 3열을 선택하는 한 경우와 1행에서 3열, 2행에서 1열, 3행에서 4열, 4행에서 2열을 선택하는 경우 단 두 가지가 있다.



[체스 queen의 행마에 대한 모든 경우의 수]

모든 경우를 다 고려해서 해를 찾아보면 N이 4일 때 나오는 경우 아래와 같은 두 가지로 나타난다.



[N이 4일 때 체스 queen의 행마]

```
4
1: (1, 2) (2, 4) (3, 1) (4, 3)
2: (1, 3) (2, 1) (3, 4) (4, 2)

Process returned 0 (0x0)   execution time : 1.448 s
Press any key to continue.
```

d

```
#include <stdio.h>
#define ABS(x) (x > 0 ? x : -(x))

//col[i] : i번째 행에서 퀸이 놓여있는 열
int n, cnt=0, col[100] = { 0, };

void printResult() {
    printf("%d: ", ++cnt);
    for(int i = 1; i <= n; i++)
```

```
        printf("(%d, %d) ", i, col[i]);
    printf("\n");
    return;
}

bool promising(int i) {
    int k = 1;
    bool sw = true;
    while(k < i && sw) {
        if(col[i] == col[k] || ABS(col[i]-col[k]) == ABS(i-k))
            sw = false;
        k++;
    }
    return sw;
}

void queens(int i) { //i: 배치한 퀸의 수
    int j;
    //i행에 퀸을 배치할 수 있는지 판단
    if (promising(i))
        if (i == n) //퀸이 모두 배치되었으면 출력
            printResult();
        else
            //현재 행에 1개씩 퀸을 배치하고 다음 행에 퀸을 배치할 수 있는 열을 탐색
            for (j=1; j<=n; j++) {
                col[i+1] = j;
                queens(i+1);
            }
    return;
}

int main()
{
    scanf("%d", &n);
    queens(0);
    return 0;
}
```

25. 다음 입력에 따른 프로그램의 실행 결과는? (4.5점)

```
int n, v[101], g[101][101];
void f( int p ) {
 int i;
 v[p] = 1;
 printf("%c ", p + 64);
 for(i = 1; i <= n; i++) {
    if( ( v[i] == 0 ) && ( g[p][i] == 1 ) )
       f(i);
 }
}
int main() {
 int i, j;
 scanf("%d\n", &n);
 for(i = 1; i <= n; i++) {
    for(j = 1; j <= n; j++)
       scanf("%d", &g[i][j]);
 }
 f(1);
 return 0;
}
```

입력	5 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0

① A B C D E

② A E D B E

③ A D C E B

④ A E B E C

⑤ A D E C B

[25번 문항 정답] ③

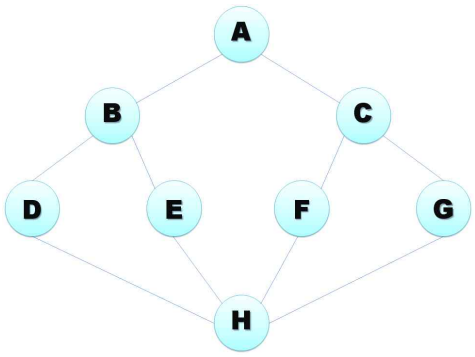
[25번 문항 해설] 길라잡이 225~227페이지

깊이우선탐색(DFS)

깊이 우선 탐색은 그래프의 한 정점(Vertex)을 시작으로 그 정점으로부터 인접한 정점을 선택하고, 선택된 정점으로부터 다시 인접한 정점을 선택하는 과정으로 시작 정점으로부터 인접한 다수의 정점들이 있더라도 그중에서 하나의 정점만을 선택하고 선택된 정점으로부터 더 이상 인접한 정점이 없을 때, 이전 선택된 정점으로부터 인접된 정점을 선택해 나간다.
자료구조의 **스택(Stack)** 구조와 같이 재귀적으로 정점을 방문한다. 다음은 DFS 알고리즘의 의사코드이다.

① 임의 정점 v 선택
② 정점 v의 방문되지 않는 인접 노드가 있으면 v을 스택에 저장
③ 정점 v에 인접하고 아직 방문되지 않은 정점 w 선택
④ 정점 w을 깊이 우선 탐색
⑤ 정점 v의 방문되지 않은 인접 노드가 없으면 스택의 노드를 pop하고 스택이 비어 있으면 종료
⑥ ② - ⑤ 과정을 반복

인접행렬을 이용한 깊이우선탐색의 프로그램을 살펴보자.



[그래프 G]

위의 그래프 G의 정점들을 DFS 방식으로 검색하는 프로그램을 작성하려고 한다.

입력과 출력의 예

입력	출력

```
8
0 1 1 0 0 0 0 0
1 0 0 1 1 0 0 0
1 0 0 0 0 1 1 0
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 0 1 1 1 1 0
```

운행 결과 : A B D H E F C G

```
01  #include <stdio.h>
02
03  int n, v[101], g[101][101];
04  void dfs( int p ) {
05      int i;
06      v[p] = 1;
07      printf("%c ", p + 64);
08      for(i = 1; i <= n; i++) {
09          if( ( v[i] == 0 ) && ( g[p][i] == 1 ) )
10              dfs(i);
11      }
12  }
13
14  int main() {
15      int i, j;
16      scanf("%d₩n", &n);
17      for(i = 1; i <= n; i++) {
18          for(j = 1; j <= n; j++)
19              scanf("%d", &g[i][j]);
20      }
21      printf("운행결과 : ");
22      dfs(1);
23      return 0;
24  }
```

< 프로그램 해설 >

03 : 1차원 배열 v[101]은 정점의 방문 여부를 저장하며 2차원 배열 g[101][101]은 정점들간의 인접행렬을 저장하기 위한 배열로 최대 정점의 수는 100개 이하로 설정해 놓았다. 단 여기서 배열을 dfs 함수와 main 함수 위에 전역으로 선언하여 모든 값은 0으로 초기화한다.

dfs 함수

04 : 깊이우선탐색의 재귀함수는 dfs이고 인자는 정수로 되어 있다. 메인 함수에서 1을 호출하고 있는데 여기서 1은 그래프에서 정점 A를 나타낸다.

06 : 정점이 방문되었음을 나타내는 것으로 배열 v[p]에 1을 대입한다. 여기서 p의 값은 dfs 함수를 호출하면서 넘겨주는 정점의 값이 된다.

07 : 방문된 정점을 출력한다. 여기서 p + 64는 변수 p의 값이 정수이므로 이를 알파벳으로 표현하기 위해 아스키코드의 문자값으로 변환하기 위해 사용되었다. (알파벳 대문자 A의 아스키코드값은 65이다.)

08 : i는 1에서 n까지 9행에서 10행의 과정을 반복한다. 여기서 i는 정점을 나타낸다.

09 : 정점 i가 방문된 정점이 아니고(v[i]==0) 정점 p로부터 정점 i가 연결되어 있는지(g[p][i]==1)를 체크하는 조건문이다.

10 : 09의 조건문을 만족하면 함수 dfs를 호출한다.

main 함수

16 : 정점의 개수를 변수 n으로 입력받는다.

17 ~ 20 : 인접행렬을 받아들인다.

22 : 정점 A를 시작으로 깊이우선탐색한다.