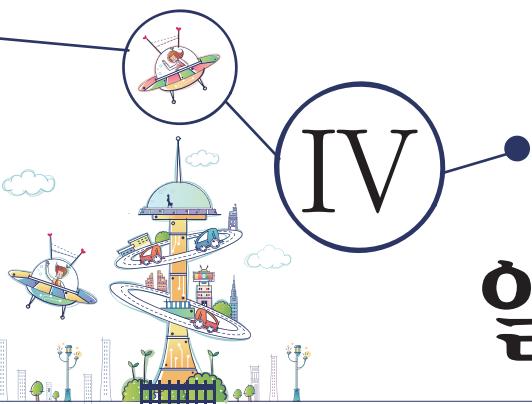


• 정보올림피아드 및 프로그래밍 교육 교재



알고리즘

제IV장에서는 알고리즘의 정의에 대해 알아보고 올림피아드 및 교원프로그래밍 경진대회에서 자주 이용되는 알고리즘들에 대해서 소개한다.

1. 알고리즘

컴퓨터 프로그래밍을 공부하다보면 알고리즘이란 말을 자주 접할 수 있다. 하지만 대부분의 사람들은 알고리즘을 어려워한다. 특히 프로그래밍 공부를 시작하여 중급 정도의 실력을 가지게 되면 알고리즘이라는 벽을 만나게 되고 이 벽을 넘지 못하고 중도 포기하는 경우도 많다. 하지만 알고리즘은 반드시 넘어야 할 벽이고, 정확하게 이해하면 그다지 어려운 것이 아니다.

알고리즘은 수학으로 말하자면 공식이다. 직각 삼각형의 직각을 끼고 있는 두 변의 길이를 알면 $a^2 + b^2 = c^2$ 과 같은 공식을 이용해서 쉽게 빗변의 길이를 구할 수 있다. 이와 마찬가지로 다양한 알고리즘들을 익히고 있으면 각종 경시대회의 문제들을 해결하는 데 많은 도움이 되며, 정보올림피아드에 참가하는 학생들의 지도에도 많은 도움이 될 것이다.

이 책에서 설명하는 알고리즘은 모두 실전에서 바로 사용할 수 있는 함수 형태로 구성하였으며, VC와 VB코드를 모두 수록하였다.

가. 알고리즘의 정의

알고리즘을 정의하는 말은 다양하다. 하지만 이 책에서는 알고리즘의 학문적인 정의에 대해서는 논하지 않을 것이다.

다음은 가장 일반적인 알고리즘의 정의이다.

“알고리즘이란 주어진 문제를 해결하기 위한 절차 혹은 방법”

위의 정의에 의하면 알고리즘이란 모든 문제에 대해서 모든 사람이 처리해 나가는 방법들은 모두 알고리즘이라고 할 수 있다. 이는 누구나 자기 자신만의 알고리즘 설계 방법을 이미 가지고 있다는 것이다. 따라서 알고리즘이란 그렇게 어려운 것이 아니다. 물론 알고리즘을 작성하는 사람에 따라서 알고리즘의 성능은 달라질 것이다. 어떤 알고리즘은 효율이 좋을 것이고, 또 다른 알고리즘은 특

정 경우에는 효율이 매우 좋으나, 그 외의 경우에는 효율이 좋지 않을 수도 있다. 이와 같이 알고리즘의 성능은 상황에 따라 달라질 수 있다.

이번 장에서 일반적으로 공식처럼 사용되고 있는 알고리즘을 소개하고, 알고리즘의 성능을 평가하는 방법에 대해서 다룬다.

나. 알고리즘의 예

알고리즘의 예를 3가지 보여주고자 한다. 첫 번째는 두 정수를 곱하는 알고리즘은 *a la russe* 알고리즘, 두 번째로는 최대공약수 알고리즘, 마지막으로 소수를 구하는 알고리즘을 소개한다. 특히 최대공약수 알고리즘과 소수를 구하는 알고리즘은 올림피아드 및 교원프로그래밍 경진대회의 실기문제로 자주 출제되므로 반드시 익혀둘 필요가 있다.

1) 두 정수의 곱셈

두 정수를 곱셈하는 것은 초등학교 때 이미 배웠을 것이다. 이러한 과정도 하나의 알고리즘이라고 할 수 있다. 그렇다면 우리가 알고 있는 곱셈 알고리즘을 자세히 분석해 보자. 만일 18과 32를 곱하는 과정을 보면 다음과 같다.

$$\begin{array}{r} 18 \\ \times 32 \\ \hline 36 \\ 54 \\ \hline 576 \end{array}$$

위의 곱셈 방법도 일종의 알고리즘이라고 할 수 있다. 이 알고리즘은 다음과 같은 수학적인 원리에서 나온 것이다.

$$18 \times 32 = 18 \times (30 + 2)$$

$$\begin{aligned}
 &= (18 \times 2) + (18 \times 30) \\
 &= 36 + 540 \\
 &= 576
 \end{aligned}$$

이 알고리즘을 분석해 보면 기본 동작은 다음과 같다.

- ① 임의의 정수 \times 한 자리의 정수의 반복
- ② 구한 정수들의 덧셈

<곱셈 알고리즘의 동작>

이 알고리즘으로 곱셈의 과정은 충분히 설명되고 우리는 누구나 위의 알고리즘으로 곱셈을 해왔다. 하지만 위의 방법은 사람들이 손으로 쉽게 계산하기 위해서 만들어진 알고리즘이다. 이 알고리즘을 컴퓨터로 처리할 수도 있지만 곱셈의 수가 많아서 효율이 떨어진다. 컴퓨터는 일반적으로 곱셈의 속도가 덧셈의 속도보다 느린다.

다음 알고리즘을 살펴보자.

- ① 3열을 만들고 첫 번째 수를 1번 열에 두 번째 수를 2번 열에 적는다.
- ② 만약 첫 번째 수가 홀수라면 두 번째 수를 3번 열 위치에 또 쓴다. 짝수라면 3번 열을 비워둔다.
- ③ 첫 번째 수를 2로 나누고(나머지는 버림), 두 번째 수에 2를 곱한다.
- ④ 1번 열의 수가 1이 될 때까지 ②~③과정을 반복한다.
- ⑤ 3번 열의 수들을 모두 더한다.

<*a la russe* 곱셈 알고리즘>

위 알고리즘을 *a la russe* 알고리즘이라고 한다. 다음은 18×32 의 계산을 *a la russe* 알고리즘으로 처리한 과정이다.

$$\begin{array}{r}
 18 & 32 & - \\
 9 & 64 & 64 \\
 4 & 128 & - \\
 2 & 256 & - \\
 1 & 512 & + 512 \\
 \hline
 576
 \end{array}$$

이 알고리즘의 계산 과정을 단순히 보면 복잡해 보인다. 하지만 계산과정을 자세히 살펴보면 다음 3가지 연산과정만으로 구성되어 있다는 것을 알 수 있다.

- 정수를 2로 나눈다.
- 정수에 2를 곱한다.
- 정수를 더한다.

<*a la russe* 알고리즘 연산>

컴퓨터의 연산에서 2를 곱하거나 나누는 연산은 곱셈이나 나눗셈으로 처리하지 않는다는 것은 이미 알려진 사실이다. 이는 10진수에서 10을 나누거나 곱하는 연산과 같은 의미를 가진다. 10진수에서 10을 곱하는 연산을 생각해보면 곱셈을 하지 않고 바로 맨 뒷자리에 0을 하나 추가하는 것으로 처리가 된다. 이진수에서도 같은 원리가 적용되므로 *a la russe* 알고리즘은 컴퓨터가 곱셈을 처리하기에 적합한 알고리즘임을 알 수 있다.



예제IV.1.1

이진수 1001×2^3 은 얼마인가?

<풀이> 1001000

이 문제는 *a la russe* 알고리즘이 얼마나 빠른지를 알 수 있게 해주는 문항이다. 2를 3번 곱하는 것으로 뒤에 0을 3개 추가하면 된다.

정답은 1001000

2) 최대공약수 구하기

두 정수 A, B의 최대공약수란 임의의 정수 A와 임의의 정수 B의 공약수 중 가장 큰 공약수를 의미한다. 예를 들어 130과 20의 최대공약수를 구해보자. 130의 약수는 1, 2, 5, 10, 13, 26, 65, 130이고, 20의 약수는 1, 2, 5, 10, 20이다. 여기서 130과 20의 공약수는 1, 2, 5, 10이다. 이 중 가장 큰 값인 10이 바로 130과 20의 최대공약수인 것이다.

최대공약수를 구하는데 이처럼 모든 약수를 구해서 구하는 알고리즘은 효율적이지 않다. 일반적으로 우리는 최대공약수를 구할 때에는 소인수분해를 이용하여 구한다. 다음 알고리즘을 살펴보자.

$$\begin{array}{r} 2 | \quad 130 \quad 20 \\ 5 | \quad 65 \quad 10 \\ \hline & 13 \quad 2 \end{array}$$

이렇게 1이외에 더 이상 나누어떨어지는 수가 없을 때 까지 소인수분해한 뒤 왼쪽의 수를 곱하면 그 값이 바로 최대공약수가 된다. 하지만 이 알고리즘은 손으로 풀기에는 좋은 알고리즘이지만 컴퓨터로 구현하기 쉽지 않은 알고리즘이다.

컴퓨터를 이용해서 최대공약수를 구하는 알고리즘 중 가장 효율이 좋은 것으로 알려진 것이 바로 유클리드 알고리즘(Euclid's Algorithm)이다. 이 유클리드 알고리즘에 대해서 알아보자.

유클리드의 알고리즘은 최대공약수의 성질을 이용하여 뱘셈과 두 값의 교환이라는 기본 동작의 반복으로 최대공약수를 구할 수 있다. 우선 최대공약수의 성질을 알아보자. 두 정수 A와 B가 있을 때, 이 A와 B의 최대공약수를 GCD라고 하면, A와 B는 다음과 같이 표현된다.

$$A = a \times GCD \quad (130 = 13 \times 10)$$

$$B = b \times GCD \quad (20 = 2 \times 10)$$

위의 a, b는 각 A와 B에서 GCD이외의 모든 인수들을 곱한 값을 의미하고, a와 b는 서로소인 경우이다. 그렇다면 A - B와 B의 최대공약수는 얼마일까?

$$A - B = a \times GCD - b \times GCD = (a - b) \times GCD$$

(a - b)와 b는 서로소 이므로 A - B와 B의 최대공약수 역시 GCD이다. 즉 쉽게 말로 표현한다면 $GCD(x, y)$ 라는 함수가 정수 x와 y의 최대공약수를 구하는 함수라고 할 때 이 함수는 다음과 같은 성질을 가진다.

$$GCD(x, y) = GCD(x - y, y) \quad \dots \dots \dots \quad ①$$

그리고 최대공약수는 두 정수의 순서에 관계없이 같으므로 다음과 같은 성질도 가진다.

$$GCD(x, y) = GCD(y, x) \quad \dots \dots \dots \quad ②$$

0이 아닌 정수 k와 0과의 최대공약수는 0이다. 따라서 다음이 성립한다.

$$GCD(k, 0) = k \quad \dots \dots \dots \quad ③$$

위의 성질 ①, ②, ③을 이용해서 130과 20의 최대공약수는 다음과 같은 과정을 통해서 구할 수 있다.

$$\begin{aligned}
 GCD(130, 20) &= GCD(110, 20) \\
 &= GCD(90, 20) \\
 &= GCD(70, 20) \\
 &= GCD(50, 20) \\
 &= GCD(30, 20) \\
 &= GCD(10, 20) && \text{여기까지는 식 ①에 의해} \\
 &= GCD(20, 10) && \text{식 ②에 의해} \\
 &= GCD(10, 10) && \text{식 ①에 의해} \\
 &= GCD(0, 10) && \text{식 ①에 의해} \\
 &= GCD(10, 0) && \text{식 ②에 의해} \\
 &= 10 && \text{식 ③에 의해}
 \end{aligned}$$

보기에는 매우 긴 과정으로 보이지만 컴퓨터가 처리하기에는 적합한 구조이다. 이유는 연산 과정이 빨셀과 교환밖에 없기 때문이다.

이 유클리드 알고리즘을 표현하자면 다음과 같다.

- ① 임의의 두 정수 A와 B를 입력받는다.
- ② B가 A보다 크면 두 수를 교환한다.
- ③ $A \leftarrow A - B$
- ④ A가 0이면 B가 최대공약수이고, 0이 아니면 ②의 과정으로 돌아간다.

〈유클리드 알고리즘〉



예제IV.1.2

두 정수 280과 30의 최대공약수를 구하시오.

<풀이> 10

여러 가지 방법이 있지만 여기서는 소인수분해를 이용한 알고리즘으로 풀어본다. 답은 $2 \times 5 = 10$

$$\begin{array}{r} 2 | 280 \quad 30 \\ 5 | 140 \quad 15 \\ \quad \quad 28 \quad 3 \end{array}$$



문제IV.1.1

유클리드 알고리즘을 함수로 작성하시오. (단, 반복문을 이용하시오.)

<풀이>

<VC>

```
int GCD(int x, int y)
{
    int temp;
    while(x != 0){
        if(x < y){
            temp = x;
            x = y;
            y = temp;
        }
        x = x - y;
    }
    return y;
}
```

<VB>

```
Function GCD
    (x as integer, y as integer) as integer
    Dim temp as integer
    Do while x <> 0
        If x < y then
            temp = x
            x = y
            y = temp
        End If
        x = x - y
    Loop
    GCD = y
End Function
```

3) 소수 구하기

소수란 1과 자기 자신 이외에는 약수를 가지지 않는 수이다. 일반적으로 소수를 구하기 위해서는 약수의 개수를 먼저 구해야 한다. 하지만 약수를 구하는 과정은 최대공약수 구하는 문제에서도 생각해보았지만 이는 컴퓨터로 처리하기에 적절한 방법이 아니다. 컴퓨터로 소수를 구하는 알고리즘을 만들 때에는 약수를 구하는 방법을 개선해서 구할 수 있다.

개선하는 방법은 다음과 같다. 기본적으로 임의의 정수 A 가 소수인지 아닌지 판단하는 방법은 1부터 A 까지의 수들로 A 를 나누어서 나누어떨어지는 수가 2개만 존재한다면 A 는 소수이다. 이 말은 A 의 약수가 2개라는 의미와 같다. 이때 존재하는 2개의 약수는 바로 1과 자기 자신이 될 것이다. 하지만 이 과정은 A 가 1000이라면 1000번을 검사해야 한다.

이 방법을 좀 더 효율적으로 고쳐보자.

임의의 정수 A 를 2부터 $A-1$ 인 정수로 나누어 본다면 어떻게 될까? 만약 A 가 소수라면 나누어떨어지는 수가 하나도 없을 것이다. 이를 이용한다면 임의의 정수 A 를 2부터 $A-1$ 까지 나누어 보면서 하나라도 나누어떨어지는 수가 나온다면 A 는 소수가 아니라는 의미와 같다. 이 방법으로 알고리즘을 만들면 검사 횟수가 크게 줄어든다. 이 알고리즘을 표현하면 다음과 같다.

- ① 임의의 정수 A 를 입력받는다.
- ② k 의 초기값을 2로 한다.
- ③ A 를 k 로 나누어 떨어지면 A 는 소수가 아니다.(끝)
- ④ k 를 1증가시킨다.
- ⑤ k 가 $A-1$ 이 아니면 다시 ③번으로 간다.
- ⑥ A 는 소수이다. (끝)

〈소수 구하기 알고리즘〉

위의 알고리즘을 언어별로 코딩해보면 다음과 같다.

```
<VC>
int prime(int n)
{
    int k;
    for(k=2 ; k<n ; k++){
        if( n % k == 0 )
            return 0; //0은 거짓
    }
    return 1; // 1은 참
}
```

```
<VB>
Function prime(n as integer) as boolean
    Dim k as integer
    For k=2 to n-1
        If n mod k = 0 Then
            prime = False
        End If
    Next k
    prime = True
End Function
```



문제IV.1.2

n을 입력받아서 2에서 입력받은 수까지 소수만을 출력하는 프로그램을 언어별로 작성하시오.

<풀이> 앞에서 작성한 prime함수를 이용해서 작성했다.

```
<VC>
void main(void)
{
    int n, i;
    printf("n입력 : ");
    scanf("%d", &n);
    for(i=2;i<=n;i++){
        if(prime(i)==1)
            printf("%d ", i);
    }
    printf("\n");
}
```

```
<VB>
Sub main()
    Dim n as integer, k as integer
    n=val(Inputbox("n입력","입력창"))
    For k=2 to n
        If prime(k) = True Then
            Debug.Print k + " ";
        End If
    Next k
    Debug.Print
End Sub
```

다. 알고리즘의 분석

한 문제를 해결하는데 다양한 알고리즘이 존재한다는 것을 알았다. 그렇기 때문에 문제를 해결하는데 어떤 알고리즘을 선택할 것인가 하는 고민이 생긴다. 그렇기 때문에 알고리즘의 분석 단계가 필요하다. 알고리즘의 객관적인 분석은 경험적 분석과 수학적인 분석이 있다.

경험적인 분석은 실제 알고리즘을 실행시키고 그 실행시간을 직접 비교해 보는 것이다. 이 방법은 특별한 수학적 지식이 없이도 쉽게 할 수 있다는 장점이 있다. 하지만 직접 알고리즘을 작성해야 한다는 것이 단점이다.

수학적인 분석은 직접 알고리즘을 작성하지 않고 알고리즘 자체를 수학적으로 분석하는 것이다. 이 부분은 어렵고 깊은 수학적인 지식을 요하므로 이 책에서는 언급하지 않는다. 경험적인 분석도 잘 활용하여 입력 데이터의 크기와 실행시간의 함수관계를 유추할 수 있다.

이번 장에서는 알고리즘의 유형을 알아보고 알고리즘 실행시간을 측정하는 방법에 대해서 알아본다.

1) 알고리즘의 유형

알고리즘의 유형은 입력 데이터의 크기 n 과 알고리즘의 실행시간 t 간의 함수관계로 표현한다. 이 관계는 보통 접근법으로 O , Θ , Ω 로 표현할 수 있는데 이 접근법은 수학적인 분석에서 사용되는 깊은 지식이 요구되므로 설명은 생략하고 일반적으로 많이 활용되는 $O(Big-O)$ 를 사용한다.

① $O(1)$

입력 자료의 수 n 에 관계없이 항상 일정한 시간을 가지는 알고리즘이다. 이러한 알고리즘은 대단히 실행효율이 높은 알고리즘이다. 검색의 예를 들자면 1개의 자료 중 내가 원하는 자료를 찾을 때 걸리는 시간이나 1000개의 자료 중 내가 원하는 자료를 찾을 때 걸리는 시간이 같은 검색 알고리즘을 작성하면 $O(1)$ 알고리즘 유형이 된다.

② $O(\lg n)$

일반적으로 전산학에서는 밑이 2인 로그를 이용한다. 밑이 2인 로그를 앞으로 \lg 로 표현하겠다. 이 알고리즘은 입력 자료의 수가 증가하는 수에 비해 실행시간의 증가 폭이 적다. 이 정도 알고리즘도 상당히 실행 효율이 좋은 알고리즘이다. 검색을 예로 들자면 1개의 자료 중 원하는 자료를 찾는데 1

초가 걸린다면 1000개의 자료를 찾는 경우는 대략 $LG\ 1000$ 초 즉 대략 10초의 시간이 걸린다는 의미이다.

③ $O(n)$

입력 자료의 수에 따라 시간이 비례적으로 증가하는 알고리즘이다. 알고리즘 내부에 입력 자료의 수와 관계있는 반복문을 사용하는 알고리즘이 대부분 이 부류에 속한다. 검색의 경우 1개 중 원하는 자료를 찾는데 1초가 걸린다면 1000개의 자료 중 원하는 자료를 검색하는데 1000초 가량 걸리게 되는 알고리즘 유형이다.

④ $O(n \lg n)$

이 알고리즘 유형은 커다란 문제를 독립적인 작은 문제로 나누어 해결할 경우에 나타날 수 있는 유형이다. 쿼팅렬의 최악의 경우를 제외하고는 이 유형을 보인다. $O(n^2)$ 의 알고리즘을 개량할 경우 나타나는 대표적인 유형이기도 하다.

⑤ $O(n^2)$

이 유형의 알고리즘이 가장 많이 등장한다고 할 수 있다. 주로 이중 루프로 알고리즘을 작성한 경우가 이 유형이 된다. 이 유형은 입력 자료가 많아질 경우 제한된 시간 내에 해를 구할 수 없다. 검색의 경우 1개 중 원하는 자료를 찾을 때 1초라면 자료의 수가 1000개가 되면 1000^2 즉 1,000,000초가 걸린다는 의미이다.

⑥ $O(2^n)$

이 유형은 입력 자료가 늘어남에 따라 급격히 실행시간이 늘어나는 유형으로 거의 실용성이 없는 알고리즘이다. 대부분 이 형태로 작성되었다가 효율이 좋은 알고리즘으로 개선되는 것이 일반적이다.

2) 알고리즘의 실행시간을 측정하는 방법

알고리즘의 실행시간을 측정하는 방법은 크게 2가지가 있다.

첫 번째 방법은 수학적인 분석이다. 수학적인 분석은 알고리즘을 실제로 작성하지 않아도 측정할 수 있다는 장점은 있지만 정확한 측정이 어렵고 알고리즘이 복잡해지면 간단히 구할 수 없다는 단점이 있다. 다음 예는 간단한 알고리즘을 수학적으로 분석하여 알고리즘의 유형을 알아내는 과정이다.

아래 알고리즘은 1부터 n 까지 더하는 알고리즘이다. 다음 알고리즘을 수학적으로 분석해보면 다음과 같다. (이 부분은 Basic은 생략한다. Basic도 같은 방법으로 분석할 수 있다.)

알고리즘	빈도수
#include<stdio.h>	-
void main(void)	-
{	-
int i, sum, n;	0
sum = 0;	1
scanf("%d", &n);	1
for(i=1;i<=n;i++){	$n+1$
sum += i;	n
}	-
printf("%d\n", sum);	1
}	-

〈알고리즘 시간 복잡도 분석〉

오른쪽의 빈도수는 명령어가 실행되는 횟수이다. 선언문은 실제 실행되는 문장이 아니므로 빈도수가 0이다. 그 이외의 수들을 모두 합해보면 다음과 같은 n 에 대한 식을 구할 수 있다.

$$2n + 4$$

이 값을 시간에 관한 함수로 표현하면 $T(n) = 2n + 4$ 가 된다.

이 함수는 입력값 n 에 대한 1차 함수이다. 따라서 대략적인 개형은 $O(n)$ 유형

의 알고리즘이라고 할 수 있다. 점근접근법에서 중요한 것은 계수나 상수항이 아니라 n 의 최고차항이다. 이와 같은 방법이 수학적으로 분석하는 한 예라고 할 수 있다. 더 자세한 내용은 이 책의 내용에서 벗어나므로 생략한다.

두 번째 방법은 실제로 알고리즘을 작성해서 실행시간을 측정하는 방법이다. 이 방법은 간단하게 실행시간을 알아볼 수 있지만 알고리즘을 작성해야만 측정을 해볼 수 있다는 단점이 있다.

프로그램의 실행시간은 다음과 같이 측정할 수 있다. 원리는 C언어로 설명하고 Basic은 나중에 예제 파일로 다루도록 하겠다.

일단 C언어 함수 중에 *clock()*과 *GetTickCount()*라는 함수가 있다. 이 두 함수는 현재의 tick값을 반환한다. (각각 *time.h*, *windows.h*에 정의되어 있음) tick값은 1/1000초마다 1씩 증가한다. 따라서 프로그램 시작 시의 tick값을 저장해 두었다가 마지막 종료시의 tick값을 구해서 시작시의 tick값을 빼주면 실행시간을 알 수 있다. 이 때 구한 값에서 1000을 나누어 주면 초 단위로 환산된다.

```
#include<stdio.h>
#include<time.h>      //clock() 함수를 사용하기 위해
#include<windows.h>   //GetTickCount() 함수를 사용하기 위해
void main(void)
{
    int start_time = clock(); // = GetTickCount()를 사용해도 같은 결과임
    int end_time;

    // 이 부분에 알고리즘을 작성한다.

    end_time = clock(); // = GetTickCount()를 사용해도 같은 결과임
    printf("실행시간 : %.3lf초\n", (double)(end_time - start_time) / 1000.0);
}
```

〈프로그램의 실행시간 측정 방법〉

완성된 알고리즘에 위의 이탤릭체로 표시된 코드를 추가하면 된다. 다시 실행하면 알고리즘의 실행시간이 초 단위로 표시된다. mili-second단위로 알고 싶다면 1000을 나누는 과정을 생략하면 된다.

실행시간이 너무 짧은 알고리즘(20ms 이내)으로 확인하면 정확한 시간을 구하기가 어렵다. (0~20ms까지는 오차라고 봐도 무방하다. 이 정도의 오차가 생기는 이유는 윈도의 다중프로그래밍 방식 때문인 것으로 판단된다.)

위의 방법으로 입력 값 n의 수를 증가시켜가면서 실행시간과의 함수관계를 유추할 수 있다. 이 방법은 수학적인 방법보다 더 정확한 값을 얻을 수 있고 수학적인 지식이 없어도 된다는 장점이 있지만 실제로 알고리즘을 작성해야 한다는 것이 단점이다.



연습문제



예제의 유클리드 알고리즘을 더욱 더 개량하는 방법이 있다. 빈도수를 줄일 수 있는 방법을 생각해보고 유클리드 알고리즘을 개량해 보시오.



다음 알고리즘의 빈도수를 적고 입력 값 n에 대한 식으로 나타내시오.

<pre>#include <stdio.h> void main(void) { int S[100][100], i, j, n; scanf("%d", &n); for(i = 0 ; i < n ; i++) for(j = 0 ; j < n ; j++) scanf("%d", &S[i][j]); printf("종료\n"); }</pre>	- - - () () () () () () -
---	--



풀

이



① 유클리드 알고리즘은 식 ①의 정리에 의해 뺄셈을 반복하게 된다. $\text{GCD}(x, y) = \text{GCD}(x - y, y)$ 의 식을 $\text{GCD}(x, y) = \text{GCD}(x \% y, y)$ 로 고쳐서 사용하면 뺄셈을 여러 번 반복하던 것을 한 번의 연산으로 처리가능하다. 프로그램의 코드 부분도 $x = x - y$; **부분을** $x = x \% y$ 로 고치면 된다.



<pre>#include <stdio.h> void main(void) { int S[100][100], i, j, n; scanf("%d", &n); for(i = 0 ; i < n ; i++) for(j = 0 ; j < n ; j++) scanf("%d", &S[i][j]); printf("종료\n"); }</pre>	$(\begin{array}{c} 0 \\ 1 \\ n+1 \\ n(n+1) \\ n \times n \\ 1 \end{array})$
---	---

$$T(n) = (n+1) + n(n+1) + n \times n + 2$$

$$= 2n^2 + 2n + 3$$

2. 검색 알고리즘

이번 단원에서는 검색 알고리즘에 대해서 알아본다. 검색 알고리즘은 문제를 푸는 과정에서 자주 이용되는 알고리즘으로 습득하고 있으면 정보올림피아드나 교원프로그래밍경진대회의 문제를 푸는데 많은 도움이 된다. 이번 단원을 통해서 순차 검색, 이분 검색, 해시에 대해서 이론적인 원리를 학습하고 직접 구현해 본다.

가. 검색의 개요

검색과 정렬은 다른 알고리즘에서 가장 많이 이용되는 알고리즘이다. 검색이란 어떤 집합에서 원하는 키 값을 가진 레코드를 찾는 작업을 의미한다. 쉽게 말하면 원하는 값이 집합 내에 있는지, 있다면 어디에 있는지를 찾는 것을 검색이라고 한다.

이 단원에서는 일반적으로 키를 이용해서 레코드를 찾는 검색은 하지 않는다. 올림피아드나 교원프로그래밍경진대회 등에서는 레코드를 검색하는 경우는 거의 없다. 따라서 이 단원에서는 단일 자료를 배열에서 검색하는 내용에 대해서 다룬다.

나. 순차 검색

순차 검색은 매우 단순하다. 이는 처음부터 원하는 값을 찾을 때 까지 순차적으로 검색을 하는 것이다. S라는 배열이 있다고 가정하자. 이 배열은 10개의 원소로 이루어져 있다.

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

이 배열 S 의 원소 중에 76이 있는지 알고 싶다. 원하는 값인 76을 찾는데 순

차검색 알고리즘을 이용하면 다음과 같다.

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

76을 찾는데 6번의 검색을 했다. 순차 검색은 원하는 자료를 찾을 때 평균적으로 $n/2$ 번을 검색한다. 최악의 경우는 n번의 검색과정을 거치게 된다. 따라서 이 알고리즘의 시간복잡도(시간에 따른 알고리즘의 유형)는 $O(n)$ 이라고 할 수 있다. 검색효율은 그렇게 좋은 편은 아니지만 구현이 간단하기 때문에 각종 대회에서는 자주 이용된다.

순차 검색 알고리즘을 s_search라는 함수로 만들어 보자.

s_search(자료의 수, 원하는 자료)

위의 형태로 함수를 사용할 것이다. 만약 원하는 자료를 찾았다면 s_search()함수는 원하는 자료가 있는 배열의 방 번호를 반환하고, 자료를 찾지 못했다면 -1을 반환하는 함수로 만들 것이다. 함수의 내용은 다음과 같다.

그리고 배열 S는 전역(main()함수 위에)으로 미리 선언되어 있어야 한다.

<VC>	<VB>
int S[배열의 최대 수];	Dim S(배열의 최대 수) as integer

<VC>

```
int s_search(int n, int value)
{
    int i;
    for(i=0 ; i<n ; i++){
        if(S[i]==value) return i;
    }
    return -1;
}
```

<VB>

```
Function s_search(n as integer, value as integer) as integer
    Dim i as integer
    For i = 1 to n
        If S(i) = value Then
            s_search = I
            End Function
        End If
    Next I
    s_search = -1
End Function
```



예제IV.2.1

100개의 자료를 가진 집합에서 원하는 자료 k를 순차 검색으로 찾고자 한다. 집합에 원하는 자료 k가 존재하지 않을 경우 검색횟수는 몇 회인가?

<풀이> 100회

순차 검색이므로 최악의 경우는 100회를 검색해야 자료의 유무를 알 수 있다. 정답은 100회

다. 이분 검색

이분 검색은 순차 검색보다 효율이 좋은 알고리즘이다. 이분 검색법은 여러분야에 많이 활용되는 알고리즘이므로 익혀두는 것이 좋다. 이분 검색은 처음부터 차례로 검색하는 것이 아니라. 배열의 가운데 값부터 검색을 시작한다. 배열의 가운데 값이 찾고자 하는 값이면 검색이 종료되고 찾고자 하는 값이 아니면 배열을 2부분으로 분할하여 다시 검색한다. 가운데 값이 찾고자 하는 값보다 크면 배열의 전반부 반을 다시 검사하고 그렇지 않으면 배열의 후반부 반을 검색하는 방법으로 원하는 값을 찾는다. 이때 전반부나 후반부의 반을 검색하는 방법은 처음에 검색 방법과 같이 다시 그 부분의 가운데 값을 검색하는 과정을 반복한다. 따라서 이분 검색을 하기 전에 배열의 값들이 정렬이 되어있어야 한다.

다음 배열에서 93을 찾는 과정을 통해 이분 검색법의 원리를 알아보자.

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

원하는 값인 93을 찾는데 이분 검색 알고리즘을 이용하면 처음에 가운데 값을 검색한다. 가운데 값이라고 하는 것은 배열의 첨자를 기준으로 한다. C언어를 기준으로 한다면 10개의 원소를 가지게 되면 첨자는 0~9가 된다. 여기에서는 가운데 값을 첨자의 처음과 마지막 값은 더한 후 2로 나눈 결과로 정의한다. 그러므로 C언어의 경우에는 $\frac{0+9}{2}$ 이므로 4가 된다. (C언어의 정수 연산의 결과는 소수점 이하를 버린다.) 즉 4번 방에 있는 값을 처음으로 검색을 하게 된다.

$$S = \{ 4, 7, 9, 12, \boxed{23}, 34, 76, 88, 93, 100 \}$$

찾고자 하는 값이 93이므로 23과는 다르다. 찾고자 하는 값인 93은 23보다 큰 값이므로 다음으로 후반부를 검색하게 된다. 후반부의 배열 첨자는 5~9가 된다. 따라서 2번째 검색하게 되는 값은 $\frac{5+9}{2}$ 가 되므로 7번 방의 값인 88을 검색하게 된다.

$$S = \{ 4, 7, 9, 12, 23, 34, 76, \boxed{88}, 93, 100 \}$$

이와 같은 과정을 반복하면 다음 단계에서 93을 검색할 수 있다.

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, \boxed{93}, 100 \}$$

위의 과정에서는 3번의 검색으로 원하는 결과를 찾았다. 이분 검색의 실행을 분석해 보면 최악의 경우 $\ln n$ 번에 원하는 자료를 찾을 수 있다는 것을 알 수

있다. 따라서 실행 시간복잡도로 보면 $O(\lg n)$ 알고리즘이다. 알고리즘의 효율이 좋기 때문에 올림피아드를 준비하는 학생들에게는 꼭 필요한 검색법이다.

이번 검색 알고리즘을 정의해보자.

b_search(시작첨자, 끝첨자, 원하는 자료)

위의 형태로 함수를 사용할 것이다. 만약 원하는 자료를 찾았다면 b_search() 함수는 원하는 자료가 있는 배열의 방 번호를 반환하고, 자료를 찾지 못했다면 -1 을 반환하는 함수로 만들 것이다. 함수의 내용은 다음과 같다.

그리고 배열 S는 전역(main()함수 위에)으로 미리 선언되어 있어야 한다.

<VC>

```
int S[배열의 최대 수];
```

<VB>

```
Dim S(배열의 최대 수) as integer
```

<VC>

```
int b_search(int s, int e, int value)
{
    int mid;
    while(s <= e){
        mid = (s + e) / 2;
        if( S[mid]==value )
            return mid;
        if( S[mid] > value )
            e = mid - 1;
        if( S[mid] < value )
            s = mid + 1;
    }
    return -1;
}
```

<VB>

```
Function b_search _
    (s as integer, e as integer, value as integer) as integer
    Dim mid as integer
    Do While( s <= e )
        mid = (s + e) / 2
        If S(mid) = value Then
            s_value = mid
            Exit Function
        End If
        If S(mid) > value Then e = mid - 1
        If S(mid) < value Then s = mid + 1
    Loop
    b_search = -1
End Function
```



예제IV.2.2

집합 S의 원소는 다음과 같다.

$$S = \{ 4, 7, 9, 12, 23, 34, 76, 88, 93, 100 \}$$

위의 집합에서 이분 검색방법을 이용하여 9를 검색하고자 한다. 몇 회만에 검색할 수 있는가? 그리고 9를 검색하는 과정에서 찾게되는 값들을 순서대로 나열하시오.

<풀이> 검색횟수 3회, 순서 $23 \rightarrow 7 \rightarrow 9$

10개의 자료를 가졌으므로 C언어 기준으로 할 경우 첫 번째 검색하는 방이 $\frac{0+9}{2} = 4.5 = 4$ 번 방, 즉 23을 검색하게 된다. 이는 찾고자 하는 값이 아니므로 그 이전 방 0~3사이에서 값을 찾으므로 중간 값은 $\frac{0+3}{2} = 1.5 = 1$ 번 방의 값인 7을 검색하게 되고 구하고자 하는 값이 이보다 크므로 마지막으로 $\frac{2+3}{2} = 2.5 = 2$ 번 방의 값인 9을 검색하면서 검색이 종료된다.

\therefore 검색횟수는 3회, 찾는 원소의 순서는 $23 \rightarrow 7 \rightarrow 9$ 의 순이다.

라. 해시

해시는 $O(1)$ 인 알고리즘이다. 즉 자료의 수 n 에 관계없이 검색시간이 일정한 함수로 효율이 매우 뛰어난 알고리즘이다. 그렇다고 어려운 알고리즘도 아니다. 해시의 원리는 자료를 저장할 때 해시함수를 이용해서 자료를 저장할 위치(버킷

이라고 부른다.)를 결정하여 저장하므로 자료를 검색할 때 저장 시 사용했던 해시 함수를 이용하면 자료를 바로 검색할 수 있는 방법이다. 다음은 간단한 해시 함수를 이용해서 자료가 저장되는 예를 보여준다.

$$h(n) = n \bmod k$$

일단 해시함수 $h(n)$ 을 위와 같이 정의하자. 해시에서 제일 중요한 것이 바로 자료를 저장할 장소를 정하는 해시함수이다. 이 해시함수가 좋아야 해시의 성능이 우수하다. 위에서 정의한 해시 함수는 저장할 숫자를 k 로 나눈 나머지를 공간번호로 이용하는 함수이다. S 라는 배열이 10개를 가진다고 가정하면 k 는 10이 된다. 다음 데이터 5개를 해시함수를 이용해서 저장하고 검색해 보자.

저장될 DATA 7, 44, 76, 128, 35

처음으로 7을 저장해보자.

7을 저장하기 위해서 해시 함수에 대입하면 $h(7) = 7$ 이 되므로 S 라는 배열의 7번 방에 데이터가 저장된다. 나머지도 모두 해시 함수로 저장될 방을 구하면 다음과 같다.

$$h(44) = 4, \quad h(76) = 6, \quad h(128) = 8, \quad h(35) = 5$$

해시에 의해서 저장된 배열 S 는 다음과 같다.

$$S = \{ -, -, -, -, 44, 35, 76, 7, 128, - \}$$

index [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

검색시에도 해시 함수를 이용하면 1번에 검색할 수 있다. 예를 들어 76을 검색하고자 한다면 $h(76) = 6$ 이므로 6번 방을 검색하면 1번의 검색으로 76을 찾을 수 있다.

이와 같은 방법이 해시이다. 간단하면서도 효율이 좋은 알고리즘이다. 하지만 문제점도 존재한다. 76, 66, 96과 같은 데이터가 있다면 모두 6번 방에 저장되어야 한다. 이런 경우를 충돌이라고 한다. 해시에서 충돌이 발생하면 이를 처리하는 과정이 필요하다. 충돌을 처리하는 방법은 주소확장법, 연결법 등이 있으나 여기서는 다루지 않을 것이다.

해시에서 제일 중요한 것은 $h(n)$ 을 정하는 것이다. 이 해시함수는 모든 n 에 대해서 최대한 데이터를 고르게 분포시키면서 충돌이 적게 발생되게 만들어져야만 한다.



예제IV.2.3

해시 테이블의 크기가 m 이고, 해시 함수는 임의의 키에 대하여 균일하게 해시 테이블에 분포시킬 때, n 개의 동일한 키가 동일한 주소로 해시 될 확률은 얼마인가? (2004 교원 예선)

<풀이> $(1/m)^n$

2004년 예선 문제로 출제가 된 문제이다. 테이블의 크기라고 하는 것은 배열의 방의 수를 의미한다. 일단 m 개의 공간이 있으므로 1개의 자료가 특정 공간으로 들어갈 확률이 균일하다고 했으므로 1개의 자료가 특정 공간에 들어갈 확률은 $1/m$ 이 된다. 그리고 다음 자료도 특정 공간에 들어갈 확률은 $(1/m) * (1/m)$ 이 된다. 따라서 n 개의 데이터가 특정 공간에 모두 충돌할 확률은 $(1/m)^n$ 이 된다.

정답은 $(1/m)^n$



연습문제



1~20사이의 값들 중 중복을 허용하지 않고 10개의 값을 뽑는다. 이 뽑은 값들 중 7이 몇 번째로 뽑혔는지를 구하는 프로그램을 작성하시오. 만약 뽑은 값들 중 7이 없다면 “None”이라고 출력하시오.
(프로그램 작성 시 값을 뽑는 과정은 랜덤을 이용하고, 검색은 순차 검색법을 이용하시오.)



풀이



10개의 랜덤 값(1~20)을 뽑아서 차례로 저장한 뒤 7을 몇 번째로 뽑았는지 찾아보면 된다. 찾을 때는 순차검색이므로 첫 번째 값부터 7이 나올 때까지 검사하면 되며, 만약 7이 없으면 "None"을 출력하면 된다. 코드는 다음과 같다.

```

<VC>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int S[10];
void main(void)
{
    int temp, pt = 0;
    srand(time(NULL)); // 랜덤 초기화
    for( int i = 0 ; i < 10 ; i++ ){
        temp = rand() % 20 + 1; // 1~20까지의 수를 랜덤으로 생성
        S[i] = temp; // i번째 생성 값을 저장
        for( int j = 0 ; i < i ; j++ ) // 중복된 값이 있으면 다시 생성
            if( temp == S[j] ) i--;
    }
    for( i = 0 ; i < 10 ; i++ )
        if( S[i] == 7 ) pt = i;
    if( pt )
        printf("%d번 방에 7이 있습니다.\n", pt);
    else
        printf("None\n");
}

```

```

<VB>
Option Explicit
Dim S(10) As Integer
Sub main()
    Dim temp As Integer, pt As Integer
    Dim i As Integer, j As Integer
    Randomize '랜덤 수 초기화
    For i = 1 To 10
        temp = Rnd() * 20 '1~20까지의 랜덤 수 발생
        S(i) = temp
        For j = 0 To i - 1
            If temp = S(j) Then i = i - 1 '중복된 수 제거
        Next j
    Next i
    For i = 1 To 10
        If S(i) = 7 Then pt = i
    Next i
    If pt <> 0 Then
        MsgBox CStr(pt) + "번 방에 7이 있습니다."
    Else
        MsgBox "None"
    End If
End Sub

```

3. 정렬 알고리즘

정렬이란 집합 S 의 원소 $1, 2, 3, \dots, n$ 이 있을 때 이 원소들을 오름차순이나 내림차순으로 재배치를 하는 작업이다.

$S = \{ 7, 3, 2, 5, 9 \}$ 라고 한다면 S 를 오름차순으로 정렬한 결과는

$S_2 = \{ 2, 3, 5, 7, 9 \}$ 이다.

S 를 내림차순으로 정렬한 결과는

$S_3 = \{ 9, 7, 5, 3, 2 \}$ 이다.

위의 과정과 같이 S 를 S_2 로 만드는 과정을 오름차순 정렬, S_3 로 만드는 과정을 내림차순 정렬이라고 한다.

기본적으로 정렬 알고리즘은 $O(n^2)$ 에 구현할 수 있다. 하지만 $O(n^2)$ 의 알고리즘들인 실제로 n 이 100,000이상이 되면 처리하는데 아주 많은 시간이 걸린다. 대표적인 $O(n^2)$ 정렬 알고리즘은 선택정렬, 버블정렬, 삽입정렬 등이 있다.

최근의 각종 경시대회 문제들은 정렬을 하더라도 $O(n^2)$ 로는 해결이 불가능한 문제들이 많이 출제되고 있다. 따라서 $O(n^2)$ 보다 효율이 좋은 정렬 알고리즘을 이용해야하는 경우가 많이 있다. 이러한 경우를 대비해서 $O(n \lg n)$ 의 정렬알고리즘인 퀵정렬, 힙정렬, 합병정렬에 대해서도 알아둘 필요가 있다.

가. 선택정렬

$O(n^2)$ 정렬 중 가장 간단하게 이용할 수 있는 정렬이 선택정렬이다. 선택정렬은 주어진 집합에서 첫 번째 원소를 선택한 후, 다른 모든 원소들과 순차적으로 비교하여 정렬이 안 된 원소와 교환하는 방법으로 정렬이 이루어진다. 오름차순 선택정렬의 알고리즘은 다음과 같다.

- ① 집합 S 의 원소 중 k 번째 원소를 선택한다.
(k 의 초기 값은 1)
- ② 집합 S 원소 중 k 번 째 원소와 $k+j$ 번째 원소를 비교한다.
(j 의 초기 값은 1)
- ③ k 째 원소보다 $k+j$ 번째 원소가 작으면 두 원소의 자리를 교환 한다.
- ④ $k+j$ 번째 원소가 마지막 원소가 아니라면 j 를 1 증가시키고 ③으로 간다.
- ⑤ k 번째 원소가 집합 S 의 마지막에서 두 번째 원소가 아니면, k 를 1증가 시키고 ①로 간다.
- ⑥ 집합 S 의 결과를 출력한다.

〈오름차순 선택 정렬 알고리즘〉

위의 예는 선택정렬 알고리즘으로써 첫 번째 원소부터 $n - 1$ 번 째 원소까지 차례로 선택하고 선택되지 않은 모든 원소들과 크기를 비교하여 현재 선택된 원소보다 값이 작으면 교환한다. 이러한 과정을 거치면 결국은 선택된 원소가 가장 작은 값을 가지게 된다. 이 과정을 $n - 1$ 번째 원소까지 반복하게 되어 전체 집합 S 가 오름차순으로 정렬된다. 오름차순정렬을 내림차순으로 바꾸기 위해서는 ③부분의 원소의 크기비교에서 부등호 방향만 반대로 바꿔주면 된다.

다음은 $S = \{ 15, 22, 13, 24, 12, 10, 20, 25 \}$ 를 오름차순으로 정렬하는 과정을 그림으로 나타낸 것이다.

S	15 22 13 27 12 10 20 25	1단계
S	10 22 15 27 13 12 20 25	2단계
S	10 12 22 27 15 13 20 25	3단계
S	10 12 13 27 22 15 20 25	4단계
S	10 12 13 15 27 22 20 25	5단계
S	10 12 13 15 20 27 22 25	6단계

S [10 12 13 15 20 22 27 25] 7단계

S [10 12 13 15 20 22 25 27] 8단계

< ■ 는 정렬된 원소 ■ 는 선택된 원소 >

위의 그림은 선택정렬의 과정을 보여주고 있다. 선택정렬은 항상 사용할 수 있도록 외워 두는 것이 좋다. 정의는 아래와 같이 하며, 사용 시에도 같은 방법으로 하면 된다. (단, 배열 S는 전역으로 미리 선언되어 있어야 한다.)

selection_sort(정렬할 자료의 수)

아래의 표는 각 언어별로 오름차순 선택정렬을 구현한 소스 코드이다.

<pre><VC> void selection_sort(int n) { int i, j, temp; for(i=0; i<n-1; i++) for(j=i+1; j<n; j++) if(S[i] > S[j]) temp = S[i]; S[i] = S[j]; S[j] = temp; }</pre>	<pre><VB> Sub selection_sort(N As Integer) Dim k as integer, j as integer, temp as integer For k = 1 to N-1 For j = k+1 to N If S(k) > S(j) Then temp = S(k) S(k) = S(j) S(j) = temp End If Next j Next k End Sub</pre>
--	--



예제IV.3.1

주어진 집합 $S = \{ 3, 5, 4, 2 \}$ 를 오름차순으로 정렬하려고 한다. 선택정렬 알고리즘을 사용하였을 경우 3단계까지 진행되었을 때의 원소들의 배치를 적으시오.

<풀이> $S = \{2, 3, 4, 5\}$

1단계 진행시마다 1개의 원소가 제자리를 찾아가게 된다. 따라서 3단계가 진행되면 3개의 원소가 제자리를 찾아갈 것이다. 참고로 단계별 결과는 다음과 같다.

알고리즘을 간단하게 설명하면 첫 번째 원소를 선택하고 2~4번째 원소와 비교하면서 만약 선택한 원소보다 값이 적으면 바꾼다. 따라서 처음에 3을 선택하고 5와 비교, 4와 비교, 2와 비교(여기서 2가 더 적으므로 2와 3을 바꾼다.) 1단계 결과는 2, 5, 4, 3이 되고 다음으로 5를 선택하고 4와 비교, 5와 4교환, 마지막으로 3과 비교 4와 3교환 결과는 2, 3, 5, 4가 되고 계속 진행하면 아래와 같이 된다.

<초기> $S = \{3, 5, 4, 2\}$

<1단계> $S = \{2, 5, 4, 3\}$

<2단계> $S = \{2, 3, 5, 4\}$

<3단계> $S = \{2, 3, 4, 5\}$

따라서 정답은 $S = \{2, 3, 4, 5\}$

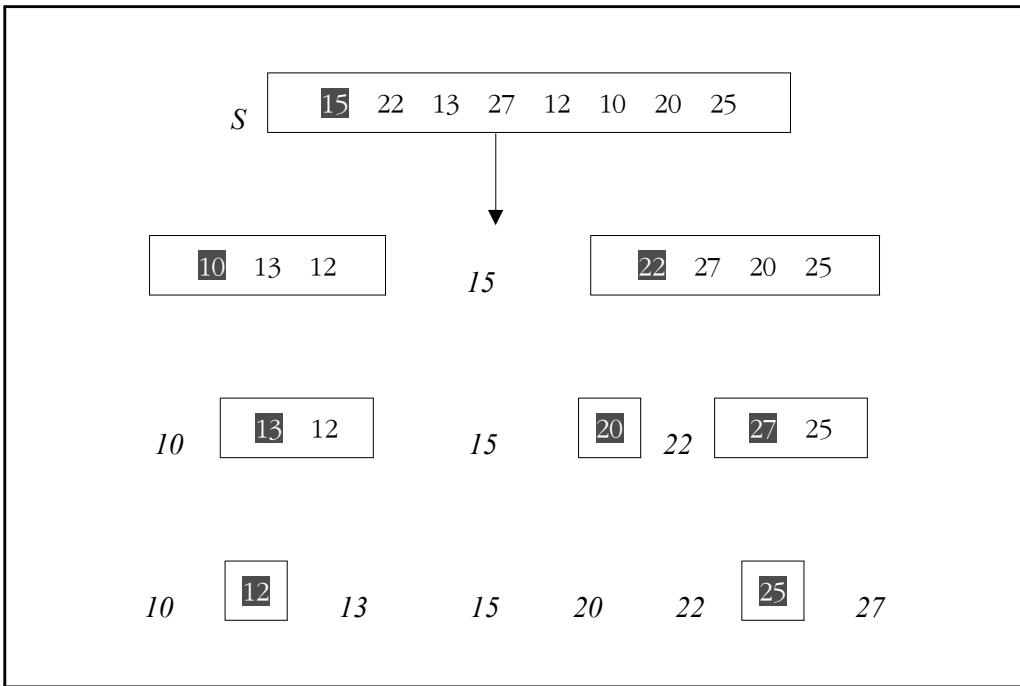
나. 퀵정렬

$O(n \lg n)$ 정렬 알고리즘을 대표하는 것이 퀵정렬이다. 퀵정렬은 10만 개 이상의 원소도 아주 빠른 속도로 처리할 수 있는 정렬이다. 하지만 알고리즘이 복잡하여 각종 경시대회 등에서 잘 활용하지 않는 경향이 있으나 데이터의 수가 10,000개 이상이라면 반드시 이 퀵정렬을 사용해야만 한다.

퀵정렬의 원리는 간단하다. 일단 기준을 정하고, 그 기준 값보다 큰 집합과 기준 값 보다 작은 집합의 두 부분으로 분할한다. 그 다음 기준 값을 그 분할된 두 집합 사이로 옮긴다. 이 작업으로 기준 값은 자기의 위치를 찾게 된다. 그리고 남은 두 집합을 새로운 집합으로 간주하여 재귀적으로 위의 알고리즘을 반복 수행하여 정렬한다. 퀵정렬에서 기준을 정할 때는 특별한 규칙은 없으나 편의상 첫 번째 원소를 기준으로 하는 경우가 많다.

$$S = \{ 15, 22, 13, 27, 12, 10, 20, 25 \}$$

위의 집합을 정렬하는 과정을 살펴보자.



〈퀵 정렬의 단계〉

위 그림에서 ■로 반전된 부분이 기준이 되는 수이다. 현재 □영역의 값들이 이 현재 기준 값으로 두 부분으로 나뉠 영역이다. n 번 검색할 때마다 영역이 둘씩 나뉘어 지므로 $O(n \lg n)$ 이 된다. 자세한 과정 설명은 생략하지만 소스코드는 외우고 있는 것이 좋다.

퀵정렬 함수는 다음과 같이 정의하고 사용한다.

`qsort(배열의 시작 첨자, 배열의 끝 첨자)`

아래 표는 퀵정렬을 구현한 소스코드이다.

<VC>

```

void qsort(int left, int right) {
    int pivot;
    int L, R;
    int ieq;
    int tempEntry;
    if (left >= right) {
        return;
    }
    pivot = S[left];
    ieq = L = left;
    R = right;
    while (L <= R) {
        if (S[R] > pivot) R--;
        else {
            tempEntry = A[L];
            S[L] = S[R];
            S[R] = tempEntry;
            if (S[L] < pivot) {
                tempEntry = S[ieq];
                S[ieq] = S[L];
                S[L] = tempEntry;
                ieq++;
            }
            L++;
        }
    }
    qsort(left, ieq - 1);
    qsort(R + 1, right);
}

```

<VB>

```

Sub qsort(left As Integer, right As Integer)
    Dim pivot As Integer
    Dim L As Integer, R As Integer
    Dim ieq As Integer
    Dim tempEntry As Integer
    If left < right Then
        pivot = S(left)
        ieq = left : L = left
        R = right
        Do While L <= R
            If S(R) > pivot Then
                R=R-1
            Else
                tempEntry = A(L)
                S(L) = S(R)
                S(R) = tempEntry
                If S(L) < pivot Then
                    tempEntry = S(ieq)
                    S(ieq) = S(L)
                    S(L) = tempEntry
                    ieq=ieq+1
                End If
                L=L+1
            Loop
        End If
        qsort left, ieq - 1
        qsort R + 1, right
    End If
End Sub

```

쿠정렬의 소스는 길어서 외우기 어렵지만 차근차근 분석해서 이해하게 되면 쉽게 외울 수 있을 것이다. 프로그래밍 실력이 중급 이상이 되려면 반드시 알고 있어야 할 알고리즘이며 정보올림피아드를 준비하는 학생들은 반드시 외우고 있어야 되는 내용이다.



예제IV.3.2

퀵 정렬의 평균 시간복잡도는 $O(n \lg n)$ 이다. 하지만 최악의 경우에는 $O(n^2)$ 으로 동작한다. $O(n^2)$ 으로 동작하는 경우는 어느 경우인가?

<풀이> 거꾸로 정렬되어 있는 경우

퀵 정렬이 $O(n \lg n)$ 이 되려면 일단 기준을 중심으로 그룹이 2개로 나누어져야 된다. 정렬이 완료되는 동안 단 한번도 그룹이 2개로 나눠지지 않으면 시간복잡도는 $O(n^2)$ 이 된다. 이러한 경우는 거꾸로 정렬이 되어 있는 경우이다. 즉, 오름차순 정렬이라면 내림차순으로 정렬이 되어있는 경우이고, 내림차순 정렬이라면 오름차순으로 정렬되어 있는 경우이다.



연습문제



선택정렬과 유사한 알고리즘 중 대표적인 것이 버블 정렬이다. 다음은 오름차순 버블 정렬의 알고리즘을 나타낸다.

초기상태 : 원소의 수가 n , $k = 1$

- ① 집합 S 의 원소 중 k 번째 원소와 $k+1$ 번째 원소와 비교한다.
- ② k 번째 원소보다 $k+1$ 번째 원소가 적으면 두 원소를 교환한다.
- ③ k 를 1증가시킨다.
- ④ $k+1 \leq n$ 보다 적으면 ①으로 간다.
- ⑤ n 을 1 감소시킨다.
- ⑥ n 이 1보다 크면 k 를 다시 1로 만들고 ①로 간다. (단계 완료)
- ⑦ 집합 S 의 내용을 출력한다.

위 알고리즘대로 아래 배열의 값들을 단계별로 결과를 보면 다음과 같다.

<초기> $S = \{ 3, 5, 4, 2 \}$
 <1단계> $S = \{ 3, 4, 2, 5 \}$
 <2단계> $S = \{ 3, 2, 4, 5 \}$
 <3단계> $S = \{ 2, 3, 4, 5 \}$

버블 정렬 알고리즘 함수를 다음과 같이 정의한다.

`bubble_sort(원소의 수)`

위의 정의대로 버블 정렬 함수를 언어별로 작성하시오.



풀



버블정렬의 소스는 선택정렬의 소스를 약간만 수정해서 완성할 수 있다.
다음 소스를 참고하여 오름차순 버블정렬을 이해하자.

<VC>

```
void bubble_sort(int n)
{
    int i, j, temp;
    for(i=0; i<n-1; i++)
        for(j=0; j<n-i-1; j++)
            if(S[j] < S[j+1])
                temp = S[j];
                S[j] = S[j+1];
                S[j+1] = temp;
}
```

<VB>

```
Sub bubble_sort(N As Integer)
    Dim k as integer, j as integer, temp as integer
    For k = 1 to N - 1
        For j = 1 to N-k-1
            If S(j) < S(j+1) Then
                temp = S(j)
                S(j) = S(j+1)
                S(j+1) = temp
            End If
        Next j
    Next k
End Sub
```

4. 트리 관련 알고리즘

트리에 관한 성질을 비롯한 기본적인 내용은 이미 Ⅲ장의 이산수학 부분에서 다루었다. 이번 단원에서 다루는 트리는 특별한 언급이 없는 한 이진트리를 의미한다. 그리고 이번 단원에서는 트리자료구조를 직접 이용한 알고리즘들에 대해서 알아보고 VC, VB코드로 구현하게 될 것이다. 트리를 이용한 알고리즘은 앞에서 배웠던 정렬, 검색과는 달리 중급 이상의 문제들을 풀 때 이용되는 알고리즘들이 많다. 교원프로그래밍경진대회를 준비하는 선생님들은 이번 단원의 이

론적인 내용만 이해하고 있으면 예선 문제를 푸는데 도움이 될 것이다. 하지만 정보올림피아드를 지도하는 선생님들은 이번 단원의 알고리즘을 익혀두어야 할 것이다. 그 이유는 학생들이 준비하는 정보올림피아드에서 트리 관련 알고리즘은 상당히 중요한 요소이기 때문이다.

가. 이진트리의 표현

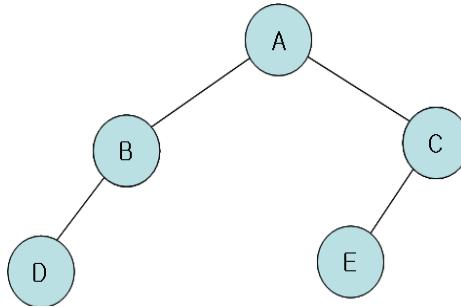
이진트리에 관한 설명은 Ⅲ장의 내용을 참고하고 이번 단원에서는 이진트리를 메모리 상에 표현하는 방법부터 설명한다. 이진트리를 메모리 상에 표현하는 방법은 크게 2가지 방법이 있다.

첫 번째는 배열을 이용한 방법이다. 배열을 이용하면 표현이 쉽고 간결하며 연산속도가 빠르다는 장점이 있으나 메모리 공간을 낭비할 수 있다는 단점을 가지는 방법이다. 하지만 쉽다는 것과 빠르다는 장점 때문에 각종 대회에서 많이 이용되는 방법이기도 하다. 따라서 이번 단원에서 트리를 구현할 때는 배열을 이용한 표현법을 사용한다.

두 번째 방법은 연결리스트를 이용하는 방법이다. 이 방법은 구현하기가 어려우며 속도가 배열에 비해서 느리나 메모리 낭비가 거의 없고 트리를 직관적으로 표현할 수 있으며 트리의 확장, 축소 등의 변환이 용이하다.

● 배열을 이용한 이진트리의 표현

배열로 이진트리를 구현할 경우에 배열의 크기는 이진트리는 항상 포화이진트리의 최대 노드 수를 기준으로 한다. 루트 노드는 배열의 1번방에 저장한다. 그 이유는 배열로 트리를 표현할 경우에 부모 자식 간의 관계 연산에 대한 체계를 확립하기 위해서이다. 아래 그림은 이진트리를 배열로 표현한 것을 나타낸다.



-1	A	B	C	D	-1	E	-1
0	1	2	3	4	5	6	7

위와 같은 방식으로 저장하면 부모 자식 노드 간에는 다음과 같은 식이 성립한다.

임의의 노드 k 의 왼쪽자식 노드는 트리 배열의 $2k$ 번 방

임의의 노드 k 의 오른쪽자식 노드는 트리 배열의 $2k + 1$ 번 방

임의의 노드 k 의 부모노드는 트리 배열의 $\lceil k / 2 \rceil$ 번 방

위의 식을 이용하면 이진트리의 다양한 연산을 배열의 첨자를 통해서 간단하게 구현할 수 있다. 하지만 이 방법으로 트리를 표현하게 되면 경우에 따라서는 많은 메모리 공간의 낭비를 초래할 수도 있다. 사향이진트리가 그 대표적이 예이다. 그러므로 완전이진트리나 포화이진트리가 배열로 표현할 수 있는 최적의 트리의 형태가 된다. 조금만 생각해 보면 왜 완전이진트리, 포화이진트리가 배열로 표현할 수 있는 최적의 형태인지 알 수 있다. 이 두 트리들은 배열로 표현할 때 낭비되는 메모리가 없다. 즉 빈 방이 없이 모든 배열의 첨자에 트리의 노드 값을 저장할 수 있는 것이 특징이다. 따라서 대회에서 사향 이진트리가 나오는 경우만 아니라면 배열로 트리를 구현하는 것이 좋다.

트리로 이용할 배열의 선언

<VC>

```
#define MAX_TREE_SIZE 최대 노드 수
int Tree[MAX_TREE_SIZE];
```

<VB>

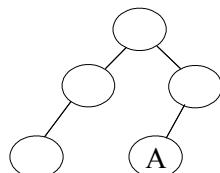
```
const MAX_TREE_SIZE as Integer = 최대 노드 수
Dim Tree(MAX_TREE_SIZE) as Integer
```

위의 내용은 트리로 이용할 배열을 선언한 예이다. 배열을 선언하고 최대 크기를 설정해 주면 트리로 사용할 수 있다. 하지만 트리의 연산은 3장에서 다룬 스택과 큐와는 달리 비선형 자료구조 이므로 연산 시 위에서 언급한 부모와 자식 간의 관계식으로 이루어진다.



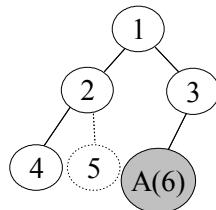
예제IV.4.1

다음 이진트리를 배열로 저장할 때 A노드는 몇 번 방에 저장되는가?



<풀이> 6번 방

트리를 배열에 저장할 때 루트노드를 1번으로 하여 포화이진트리라고 가정하고 차례로 번호를 붙이면 된다. 따라서 각 노드별로 저장되는 방의 번호는 다음과 같다. (5번 노드는 없지만 배열에는 공란으로 비워둬야 각 연산이 가능하다.)

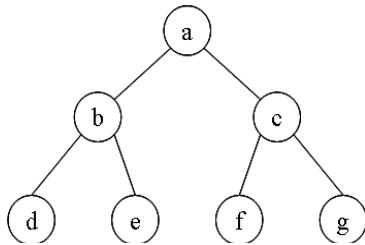


따라서 정답은 6번 방

나. 이진트리의 순회

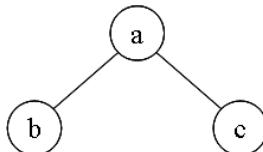
이진트리의 순회란 이진트리의 루트 노드로부터 시작하여 모든 노드를 체계적으로 탐색하는 방법을 의미한다. 트리와 같은 비선형 자료 구조의 순회 방법은 그 자체가 특별한 알고리즘으로 활용되는 경우가 많다. 따라서 순회하는 방법은 반드시 익히고 있어야 한다.

다음 트리를 기본으로 해서 각 순회를 설명한다.



1) 전위 순회

전위 순회의 순서는 다음과 같다.



위의 트리를 기준으로 생각해 보면 방문순서는 **a - b - c** (부모, 왼쪽자식, 오른쪽자식)로 순회하는 것이 전위 순회이다. 즉 부모를 먼저 방문하는 방식이 전위 순회이다. 여기서 b와 c는 노드일 수도 있고 서브 트리일 수도 있다. 만약 b나 c가 서브트리라면 위의 순회 순서를 바탕으로 재귀적으로 다시 순회하게 된다. 앞에 주어졌던 트리를 기준으로 전위 순회 한 결과는 다음과 같다.

a - b - d - e - c - f - g

전위 순회의 함수는 다음과 같이 정의한다.

preorder(순회 시작할 노드 번호)

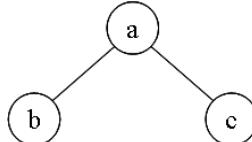
일반적으로 루트노드에서 시작하므로 순회시작 노드 번호는 1이 되며 특별한 경우에는 다른 노드에서 시작할 수도 있다. 다음은 알고리즘을 재귀 호출을 이용하여 표현한 것이다. 노드가 없는 빈 배열에는 -1이 들어있다고 가정한다.

```
void preorder(int cur)
{
    if(Tree[cur]!=-1){
        printf("%d\n", Tree[cur]);
        preorder(cur*2);
        preorder(cur*2+1);
    }
}
```

```
Sub Preorder(cur as integer)
If Tree(cur) <> -1 Then
    print #2, cstr(Tree(cur))
    Preorder cur*2
    Preorder cur*2+1
End If
End Sub
```

2) 중위 순회

중위 순회의 순서는 다음과 같다.



위의 트리를 기준으로 생각해 보면 방문순서는 $b - \boxed{a} - c$ (왼쪽자식, 부모, 오른쪽자식)로 순회하는 것이 중위 순회이다. 왼쪽자식 혹은 서브트리를 방문하고 루트를 방문, 마지막으로 오른쪽 자식 또는 서브트리를 방문하는 방식이 중위 순회이다. 앞에 주어졌던 트리를 기준으로 전위 순회 한 결과는 다음과 같다.

$d - b - e - \boxed{a} - f - c - g$

중위 순회의 함수는 다음과 같이 정의한다.

inorder(순회 시작할 노드 번호)

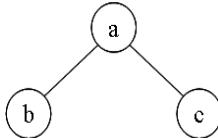
일반적으로 루트노드에서 시작하므로 순회시작 노드 번호는 1이 되며 특별한 경우에는 다른 노드에서 시작할 수도 있다. 다음은 알고리즘을 재귀 호출을 이용하여 표현한 것이다. 마찬가지로 배열의 값이 -1일 경우에는 노드가 존재하지 않는다는 의미이다.

```
void inorder(int cur){
    if(Tree[cur]!=-1){
        inorder(cur*2);
        printf("%d\n", Tree[cur]);
        inorder(cur*2+1);
    }
}
```

```
Sub inorder(cur as integer)
    If Tree(cur) <> -1 Then
        inorder cur*2
        print #2, cstr(Tree(cur))
        inorder cur*2+1
    End If
End Sub
```

3) 후위 순회

중위 순회의 순서는 다음과 같다.



위의 트리를 기준으로 생각해 보면 방문순서는 $b - c - a$ (왼쪽자식, 오른쪽자식, 부모)로 순회하는 것이 후위 순회이다. 왼쪽자식 혹은 서브트리를 방문하고 오른쪽 자식 또는 서브트리를 방문한 다음 마지막으로 루트를 방문하는 방식이 후위 순회이다. 앞에 주어졌던 트리를 기준으로 후위 순회 한 결과는 다음과 같다.

$d - e - b - f - g - c - a$

후위 순회의 함수는 다음과 같이 정의한다.

postorder(순회 시작할 노드 번호)

다음은 알고리즘을 재귀 호출을 이용하여 후위 순회를 표현한 것이다. 마찬가지로 배열의 값이 -1일 경우에는 노드가 존재하지 않는다는 의미이다.

```
void postorder(int cur){
    if(Tree[cur]!=-1){
        postorder(cur*2);
        postorder(cur*2+1);
        printf("%d\n", Tree[cur]);
    }
}
```

```
Sub postorder(cur as integer)
    If Tree(cur) <> -1 Then
        postorder cur*2
        postorder cur*2+1
        print #2, cstr(Tree(cur))
    End If
End Sub
```

다. 이진 트리의 응용

이진 트리의 응용에서는 트리가 실전 문제들을 통해서 트리를 어떻게 프로그래밍에 활용하는지를 알아보도록 한다.

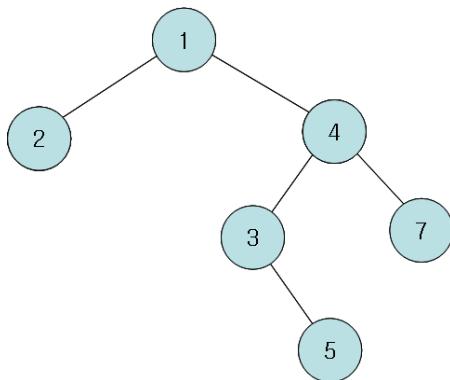
정보올림피아드를 지도하는 선생님들을 위해서 정보올림피아드 기출문제도 다루고 있으므로 익힐 수 있도록 하자.



문제IV.4.1

input.txt에서 트리의 노드 값을 입력받아 전위순회, 중위순회, 후위순회한 결과를 출력하는 프로그램을 작성하시오.

<입력형식> 입력 파일은 input.txt이며, 입력은 2줄로 이루어지며, 첫 번째 줄에는 총 노드 수(완전이진트리로 가정하고, 빈 노드의 수도 포함)가 주어지고, 두 번째 줄에는 노드의 값이 완전이진트리라고 가정하고, 루트로부터 순차적으로 주어진다. 단 트리의 노드가 없는 부분의 값은 -1로 주어진다.



<출력형식> 출력은 output.txt로 이루어지며, 총 3줄을 출력한다. 첫 번째 줄에는 전위순회, 둘 째 줄에는 중위순회, 마지막 줄에는 후위순회한 결과를 출력한다.

입력파일(INPUT2.TXT) 예 : 13
1 2 4 -1 -1 3 7 -1 -1 -1 -1 5

출력파일(OUTPUT2.TXT) 예 : 전위 : 1 2 4 3 5 7
중위 : 2 1 3 5 4 7
후위 : 2 5 3 7 4 1

<풀이> 이 문제의 풀이에는 이전에 정의했던 순회 함수들을 그대로 이용하고 있으므로 활용하는 방법을 익힐 수 있을 것이다.

<VC>

```

#include <stdio.h>
#define MAX_TREE_SIZE 512

/* 트리의 최대 크기를 512로 잡은 이유는 트리의 최대 높이가 9층까지를 처리할
수 있게 하기 위해서임 */

int Tree[MAX_TREE_SIZE];
FILE *in=fopen("input.txt","r");
FILE *out=fopen("output.txt","w");

/* 최대 크기로 트리 배열을 선언하고, 입출력 파일에 대한 파일 포인터를 선언*/

void preorder(int c){
    if(Tree[c] != -1){
        fprintf(out,"%d ",Tree[c]);
        preorder(c * 2);
        preorder(c * 2 + 1);
    }
}

/* 전위순회 함수임. if문에서 Tree[c]라는 의미는 Tree[c] != 0과 같은 의미로 트
리에 값이 없으면 0이 있기 때문에 값이 없으면 더 이상 검색을 하지 않게
하기 위한 수단으로 이용 */

void inorder(int c){
    if(Tree[c] != -1){
        inorder(c * 2);
        fprintf(out,"%d ",Tree[c]);
        inorder(c * 2 + 1);
    }
}

/* 중위순회 함수임. if문에서 Tree[c]라는 의미는 Tree[c] != 0과 같은 의미로 트
리에 값이 없으면 0이 있기 때문에 값이 없으면 더 이상 검색을 하지 않게
하기 위한 수단으로 이용 */

```

```
void postorder(int c){
    if(Tree[c] != -1){
        postorder(c * 2);
        postorder(c * 2 + 1);
        fprintf(out,"%d ",Tree[c]);
    }
}

/* 후위순회 함수임. if문에서 Tree[c]라는 의미는 Tree[c] != -1과 같은 의미로 트리에 값이 없으면 0이 있기 때문에 값이 없으면 더 이상 검색을 하지 않게 하기위한 수단으로 이용 */
void main(void){                                // 프로그램의 시작
    int i, n;
    fscanf(in,"%d",&n);                      // 파일로부터 전체 노드 수를 받음
    for(i=1;i<=n;i++)
        fscanf(in,"%d",&Tree[i]);      // 각 노드의 값을 차례로 배열에 입력
    fprintf(out, "전위 : ");
    preorder(1);
    fprintf(out, "\n중위 : ");
    inorder(1);
    fprintf(out, "\n후위 : ");
    postorder(1);
    fprintf(out, "\n");                    // 전위, 중위, 후위 순으로 호출하면서 출력
    fclose(in);
    fclose(out);
}
```

<VB>

```
Option Explicit
```

```
Const MAX_TREE_SIZE As Integer = 512
Dim Tree(MAX_TREE_SIZE) As Integer
```

'MAX_TREE_SIZE를 상수로 선언하고, 512로 한 이유는 트리의 최대 높이를
'9층까지 가능하게 처리하겠다는 의도임.
'최대 크기로 트리 배열을 선언

```
Sub Preorder(c As Integer)
    If Tree(c) <> -1 Then
        Print #2, Tree(c);
        Preorder c * 2
        Preorder c * 2 + 1
    End If
End Sub
```

' 위의 내용은 전위 순회를 하는 모듈

```
Sub Inorder(c As Integer)
    If Tree(c) <> -1 Then
        Inorder c * 2
        Print #2, Tree(c);
        Inorder c * 2 + 1
    End If
End Sub
```

' 위의 내용은 중위순회 하기 위한 모듈

```
Sub Postorder(c As Integer)
    If Tree(c) <> -1 Then
        Postorder c * 2
        Postorder c * 2 + 1
        Print #2, Tree(c);
    End If
End Sub
```

' 위의 내용은 후위순회를 하는 모듈

```
Sub main()
    Dim i As Integer, n As Integer
    Open "input.txt" For Input As #1      '파일 입출력을 위한 준비
    Open "output.txt" For Output As #2
```

```

Input #1, n           '트리의 크기를 입력받음
For i = 1 To n
    Input #1, Tree(i)   '각 트리의 값을 배열에 저장
    Next i
Print #2, "전위 : ";
Preorder 1             '전위순회 시작
Print #2, vbCrLf + "중위 : ";
Inorder 1              '중위탐색 시작
Print #2, vbCrLf + "후위 : ";
Postorder 1            '후위탐색 시작
Print #2, vbCrLf
Close #1, #2
End Sub

```



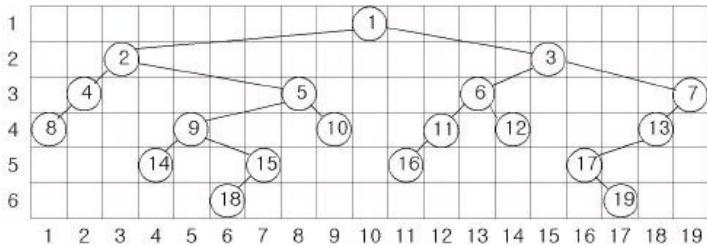
문제IV.4.2 이진트리의 너비 (2003 한국정보올림피아드 고등부 기출)

이진트리를 다음의 규칙에 따라 행과 열에 번호가 붙여진 격자 모양의 틀 속에 그리려고 한다.

- ① 이진트리에서 같은 레벨(level)에 있는 노드는 같은 행에 위치한다.
- ② 한 열에는 한 노드만 존재한다.
- ③ 임의의 노드의 왼쪽 부트리(left subtree)에 있는 노드들은 해당 노드보다 왼쪽의 열에 위치하고, 오른쪽 부트리(right subtree)에 있는 노드들은 해당 노드보다 오른쪽의 열에 위치한다.
- ④ 노드가 배치된 가장 왼쪽 열과 가장 오른쪽 열 사이에는 아무 노드도 없이 비어있는 열은 없다.

이와 같은 규칙에 따라 이진트리를 그릴 때 각 레벨의 너비는 그 레벨에 할당된 노드 중 가장 오른쪽에 위치한 노드의 열 번호에서 가장 왼쪽에 위치한 노드의 열 번호를 뺀 값 더하기 1로 정의한다. 트리의 레벨은 가장 위쪽에 있는 루트 노드가 1이고 아래로 1씩 증가한다.

아래 그림은 어떤 이진트리를 위의 규칙에 따라 그려본 것이다. 첫 번째 레벨의 너비는 1, 두 번째 레벨의 너비는 13, 3번째, 4번째 레벨의 너비는 각각 18이고, 5번째 레벨의 너비는 13이며, 그리고 6번 째 레벨의 너비는 12이다.



우리는 주어진 이진트리를 위의 규칙에 따라 그릴 때에 너비가 가장 넓은 레벨과 그 레벨의 너비를 계산하려고 한다. 아래 그림의 예에서 너비가 가장 넓은 레벨은 3번째와 4번째로 그 너비는 18이다. 너비가 가장 넓은 레벨이 두 개 이상 있을 때는 번호가 작은 레벨을 답으로 한다. 그러므로 이 예에 대한 답은 레벨은 3이고, 너비는 18이다.

임의의 이진트리가 입력으로 주어질 때 너비가 가장 넓은 레벨과 그 레벨의 너비를 출력하는 프로그램을 작성하시오.

실행 파일의 이름은 WIDTH.EXE로 하고 실행 시간은 0.5초를 초과할 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 노드의 개수를 나타내는 정수 N ($1 \leq N \leq 20$) 이 주어진다. 다음 N 개의 줄에는 각 줄마다 노드 번호와 해당 노드의 왼쪽 자식 노드와 오른쪽 자식 노드의 번호가 순서대로 주어진다. 노드들의 번호는 1부터 N 까지로 주어진다. 자식이 없는 경우에는 자식 노드의 번호가 -1로 주어진다. 루트 노드의 번호는 1이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 너비가 가장 넓은 레벨과 그 레벨의 너비를 순서대로 출력한다. 단, 너비가 가장 넓은 레벨이 두 개 이상 있을 때는 번호가 작은 레벨을 출력한다.

입력과 출력의 예

입력(input.txt)

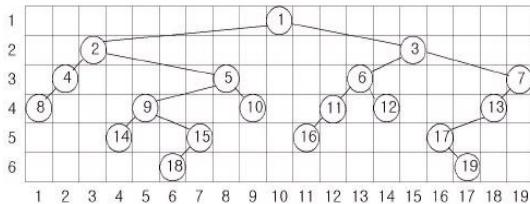
```
19  
1 2 3  
2 4 5  
3 6 7  
4 8 -1  
5 9 10  
6 11 12  
7 13 -1  
8 -1 -1  
9 14 15  
10 -1 -1  
11 16 -1  
12 -1 -4  
13 17 -1  
14 -1 -1  
15 18 -1  
16 -1 -1  
17 -1 19  
18 -1 -1  
19 -1 -1
```

출력(output.txt)

```
3 18
```

<풀이>

이 문제는 한국정보올림피아드의 기출문제이니 만큼 쉽게 해법을 찾기 어렵다. 하지만 자세히 문제를 분석해 보면, 해결의 실마리가 보인다.



위 그림을 잘 살펴보자. 너비를 구하기 위해서는 열 번호가 있는 1~19를 유심히 봐야 한다. 1부터 19까지의 열에 있는 값들만 쭉 나열하면 8-4-2-14-9-18-15-5-10-1-16-11-6-12-3-17-19-13-7의 순이 된다. 이 순서의 의미를 잘 파악해보자.

자세히 살펴보면 중위 우선 탐색 순위와 같다라는 것을 알 수 있다. 이것이 가장 큰 힌트가 될 수 있다. 중위 우선 탐색을 하면서 깊이를 저장하여 깊이별 중위 우선 순위간의 관계를 이용하여 너비를 구할 수 있다.

소스는 다음과 같다.

— ● <VC>

```
#include<stdio.h>
#include<conio.h>
#define MAXN 1200001 // 최대 노드수가 20개 이므로 2^20보다 많아야 함(사항일 경우.)
int Tree[MAXN], N; // 트리와 트리의 노드 수
int mind[MAXN], maxd[MAXN]; // 각 깊이에서의 최대와 최소를 저장할 배열
int mdep, Dep, max = -1, seq = 1; // 최대깊이, 정답깊이 등의 변수
void input(void) // 파일의 입력형식을 Tree에 저장하는 알고리즘 분석요함.
{
    FILE *in=fopen("input.txt","r");
    for(int i=1 ; i<MAXN ; i++)
        mind[i] = 0x7fffffff,
    fscanf(in, "%d", &N);
}
```

```

fscanf(in, "%d %d %d", &Tree[1], &Tree[2], &Tree[3]);
for(i=2 ; i<N ;i++){
    int temp, temp2, temp3;
    fscanf(in, "%d %d %d", &temp, &temp2, &temp3);
    for(int j=1 ; j<i*2+1 ; j++){
        if( Tree[j] == temp ){
            Tree[j*2] = temp2;
            Tree[j*2+1] = temp3;
            break;
        }
    }
}

void inorder(int cur, int dep) // 중위 순회를 활용한 풀이, dep는 현재 깊이
{
    if( Tree[cur] <= 0 ) return;
    if( mdep < dep ) mdep = dep; // 최대 깊이를 저장
    inorder( cur * 2 , dep + 1);
    if( mind[dep] > seq ) mind[dep] = seq; // seq는 중위순회 상의 순서를 나타냄
    if( maxd[dep] < seq ) maxd[dep] = seq; // 각 깊이에서 seq의 최대 최소 저장
    seq++; // 나중에 최대-최소 하면 각 깊이에서 너비를
    inorder( cur * 2 + 1 , dep + 1); // 구할 수 있음.
}

void process(void) // 순회를 마치고 깊이별로 너비를 구하여 최대너비 저장
{
    for(int i=1 ; i<=mdep ; i++)
    if( max < maxd[i] - mind[i] ){
        Dep = i;
        max = maxd[i] - mind[i];
    }
}

void output(void) // 결과를 출력
{
    FILE *out = fopen("output.txt", "w");
    fprintf(out, "%d\n", Dep, max + 1);
}

void main(void) // 메인 함수
{
    input();
    inorder(1, 1);
    process();
    output();
}

```

<VB>

Option Explicit

```

Const MAXN As Long = 1200001
Dim Tree(MAXN) As Integer, mind(MAXN) As Integer, maxd(MAXN) As Integer
Dim mdep As Integer, dep As Integer, max As Integer, seq As Integer, N As Integer

Sub inp()
    Dim i As Integer, j As Integer
    Open "input.txt" For Input As #1
    For i = 1 To MAXN - 1
        mind(i) = 9999
    Next i
    Input #1, N
    Input #1, Tree(1), Tree(2), Tree(3)
    For i = 2 To N
        Dim temp As Integer, temp2 As Integer, temp3 As Integer
        Input #1, temp, temp2, temp3
        For j = 1 To i * 2
            If Tree(j) = temp Then
                Tree(j * 2) = temp2
                Tree(j * 2 + 1) = temp3
                Exit For
            End If
        Next j
    Next i
End Sub

Sub inorder(cur As Integer, dep As Integer)
    If Tree(cur) <= 0 Then Exit Sub
    If mdep < dep Then mdep = dep
    inorder cur * 2, dep + 1
    If mind(dep) > seq Then mind(dep) = seq
    If maxd(dep) < seq Then maxd(dep) = seq
    seq = seq + 1
    inorder cur * 2 + 1, dep + 1
End Sub

```

```
Sub process()
    Dim i As Integer
    For i = 1 To mdep
        If max < maxd(o) - mind(i) Then
            dep = i
            max = maxd(i) - mind(i)
        End If
    End Sub

Sub output()
    Open "output.txt" For Output As #2
    Print #2, CStr(dep) + " " + CStr(max + 1)
End Sub

Sub main()
    inp
    inorder 1, 1
    process
    output
End Sub
```



연습문제

①

문제 IV.4.2의 트리의 너비 문제를 자세히 분석해 보자. 이 문제가 요구하는 답은 최대 너비와 그 깊이를 구하는 것이다. 하지만 최대 너비가 되는 깊이가 2군데 이상 존재할 경우에는 깊이가 낮은 것만 출력하는 문제였다. 이 문제를 약간 고쳐서 최대 너비를 구하고 그 깊이를 출력하되 그 깊이가 2군데 이상 존재할 경우 모두 출력하는 프로그램을 만드시오.



풀



위의 코드에서 약간만 수정하였다. 아래 코드를 살펴보면 모든 레벨을 출력하는 부분을 찾을 수 있을 것이다.

<VC>

```
#include<stdio.h>
#include<conio.h>
#define MAXN 1200001 // 최대 노드수가 20개 이므로 2^20보다 많아야 함(사항일 경우.)
int Tree[MAXN], N; // 트리와 트리의 노드 수
int mind[MAXN], maxd[MAXN]; // 각 깊이에서의 최대와 최소를 저장할 배열
int mdep, Dep, max = -1, seq = 1; // 최대깊이, 정답깊이 등의 변수
void input(void) // 파일의 입력형식을 Tree에 저장하는 알고리즘 분석요함.
{
    FILE *in=fopen("input.txt","r");
    for(int i=1 ; i<MAXN ; i++)
        mind[i] = 0x7fffffff;
    fscanf(in, "%d", &N);
    fscanf(in, "%d %d %d", &Tree[1], &Tree[2], &Tree[3]);
    for(i=2 ; i<N ; i++){
        int temp, temp2, temp3;
        fscanf(in, "%d %d %d", &temp, &temp2, &temp3);
        for(int j=1 ; j<i*2+1 ; j++)
            if( Tree[j] == temp ){
                Tree[j*2] = temp2;
                Tree[j*2+1] = temp3;
                break;
            }
    }
}

void inorder(int cur, int dep) // 중위 순회를 활용한 풀이, dep는 현재 깊이
{
    if( Tree[cur] <= 0 ) return;
    if( mdep < dep ) mdep = dep; // 최대 깊이를 저장
    inorder( cur * 2 , dep + 1);
    if( mind[dep] > seq ) mind[dep] = seq; // seq는 중위순회 상의 순서를 나타냄
    if( maxd[dep] < seq ) maxd[dep] = seq; // 각 깊이에서 seq의 최대 최소 저장
    seq++;
    inorder( cur * 2 + 1 , dep + 1); // 구할 수 있음.
}
```

```

void process(void)           // 순회를 마치고 깊이별로 너비를 구하여 최대너비 저장
{
    for(int i=1 ; i<=mdep ; i++)
    if( max <= maxd[i] - mind[i] ){
        max = maxd[i] - mind[i];
    }
}

void output(void)      // 결과를 출력
{
    FILE *out = fopen("output.txt", "w");
    fprintf(out, "%d", max + 1);
    for( i=0 ; i < mdep ; i++ )
        if( maxd[i] - mind[i] == max ) // 이 부분에서 다중으로 검색하게 됨
            fprintf(out, "%d ", i);
}
void main(void) // 메인 함수
{
    input();
    inorder(1, 1);
    process();
    output();
}

```

<VB>

```

Const MAXN As Long = 1200001
Dim Tree(MAXN) As Integer, mind(MAXN) As Integer, maxd(MAXN) As Integer
Dim mdep As Integer, dep As Integer, max As Integer, seq As Integer, N As Integer

Sub inp()
    Dim i As Integer, j As Integer
    Open "input.txt" For Input As #1
    For i = 1 To MAXN - 1
        mind(i) = 9999
    Next i
    Input #1, N
    Input #1, Tree(1), Tree(2), Tree(3)
    For i = 2 To N
        Dim temp As Integer, temp2 As Integer, temp3 As Integer
        Input #1, temp, temp2, temp3
        For j = 1 To i * 2
            If Tree(j) = temp Then
                Tree(j * 2) = temp2
                Tree(j * 2 + 1) = temp3
                Exit For
            End If
        Next j
    Next i
End Sub

```

```
Sub inorder(cur As Integer, dep As Integer)
    If Tree(cur) <= 0 Then Exit Sub
    If mdep < dep Then mdep = dep
    inorder cur * 2, dep + 1
    If mind(dpe) > seq Then mind(dep) = seq
    If maxd(dep) < seq Then maxd(dep) = seq
    seq = seq + 1
    inorder cur * 2 + 1, dep + 1
End Sub

Sub process()
    Dim i As Integer
    For i = 1 To mdep
        If max < maxd(o) - mind(i) Then
            dep = i
            max = maxd(i) - mind(i)
        End If
    End Sub

Sub output()
    Open "output.txt" For Output As #2
    Print #2, CStr(dep) + " " + CStr(max + 1)
    For i = 0 to mdep - 1
        If maxd(i) - mind(i) = max then      '이 부분을 추가하여야 됨
            Print #2, Cstr(i) + " "
        End If
    Next i
End Sub

Sub main()
    inp
    inorder 1, 1
    process
    output
End Sub
```

<VB>

```

#include<stdio.h>
#include<conio.h>
#define MAXN 1200001 // 최대 노드수가 20개 이므로 2^20보다 많아야 함(사향일 경우.)
int Tree[MAXN], N; // 트리와 트리의 노드 수
int mind[MAXN], maxd[MAXN]; // 각 깊이에서의 최대와 최소를 저장할 배열
int mdep, Dep, max = -1, seq = 1; // 최대깊이, 정답깊이 등의 변수
void input(void) // 파일의 입력형식을 Tree에 저장하는 알고리즘 분석요함.
{
    FILE *in=fopen("input.txt","r");
    for(int i=1 ; i<MAXN ; i++)
        mind[i] = 0x7fffffff;
    fscanf(in, "%d", &N);
    fscanf(in, "%d %d %d", &Tree[1], &Tree[2], &Tree[3]);
    for(i=2 ; i<N ; i++){
        int temp, temp2, temp3;
        fscanf(in, "%d %d %d", &temp, &temp2, &temp3);
        for(int j=1 ; j<i*2+1 ; j++)
            if( Tree[j] == temp ){
                Tree[j*2] = temp2;
                Tree[j*2+1] = temp3;
                break;
            }
    }
}

void inorder(int cur, int dep) // 중위 순회를 활용한 풀이, dep는 현재 깊이
{
    if( Tree[cur] <= 0 ) return;
    if( mdep < dep ) mdep = dep; // 최대 깊이를 저장
    inorder( cur * 2 , dep + 1 );
    if( mind[dep] > seq ) mind[dep] = seq; // seq는 중위순회 상의 순서를 나타냄
    if( maxd[dep] < seq ) maxd[dep] = seq; // 각 깊이에서 seq의 최대 최소 저장
    seq++;
    inorder( cur * 2 + 1 , dep + 1 ); // 구할 수 있음.
}

```

위의 코드를 실행하면 그 결과는 다음과 같이 출력된다.

18 3 4

위의 답에서 18은 최대 너비를 의미하고 3, 4는 깊이 3과 깊이 4에서의 너비가 최대가 된다는 의미이다.

5. 그래프 관련 알고리즘

그래프도 트리와 마찬가지로 성질을 비롯한 기본적인 내용은 Ⅲ장의 이산수학 부분에 잘 설명되어 있다. 이 단원에서는 그래프를 실제 프로그래밍으로 구현하는 방법과 탐색 알고리즘을 작성하는 방법과 같이 실제로 그래프를 다루는 방법에 대해서 알아본다. 마지막으로 그래프의 활용 부분에서는 그래프를 활용한 알고리즘 중에 대표적이라고 할 수 있는 최단경로탐색 알고리즘에 대해서도 다룰 것이다.

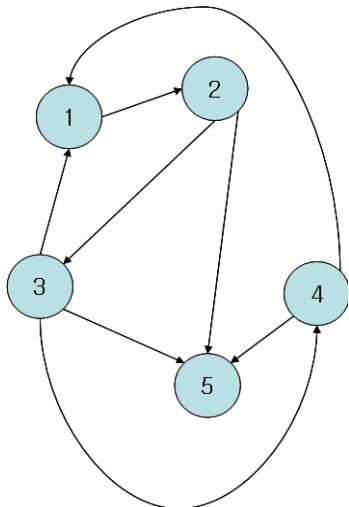
가. 그래프의 표현

그래프를 메모리 상에 저장하는 방법은 배열을 이용하는 방법과 연결리스트를 이용하는 두 가지 방법이 있다. 이 단원에서는 배열을 이용하는 방법 즉, 인접 행렬로써 그래프를 표현하는 방법에 대해서 알아본다.

인접행렬은 그래프의 정점의 수가 N 개라면 $N \times N$ 개의 2차원배열을 만들어서 각 정점간의 연결정보를 저장하는 방법을 말한다. 정보를 저장할 때 일반적으로는 1번 정점에서 2번 정점으로 연결되는 길이 있다면 인접행렬 1행 2열의 내용에 1을 입력하고 길이 없다면 0을 입력하는 것이 가장 일반적인 방법이다. 하지만 문제의 특성에 따라서 얼마든지 입력 값이나 저장 방법을 활용할 수 있다. 예를 들어서 이동 시간을 가중치로 가지는 그래프라고 한다면 1번 도시에서 2번 도시로 가는데 23분이 걸린다면 인접행렬 1행 2열에 저장될 내용은 23이 될 것이다. 만약 길이 없다면 0이 되는 것은 동일하다. 그리고 무향 그래프일 경우에는 1행 2열에 a 라는 값을 입력하면 2행 1열에도 a 라는 값을 입력해야 한다. 무향그래프는 1번 정점에서 2번 정점으로 가는 길이 2번 정점에서 1번 정점으로 가는 길과 같기 때문이다.

인접행렬의 기본 저장 방법을 정확하게 이해하면 주어진 문제의 특성에 따라서 적합한 인접행렬 구조를 만들 수 있다. 다음은 방향그래프를 인접행렬로

저장한 예이다.



	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	0	1
3	1	0	0	1	1
4	1	0	0	0	1
5	0	0	0	0	0

〈그래프의 인접행렬 표현〉



예제IV.5.1

입력 파일로부터 그래프의 정점과 간선에 대한 정보를 입력받아서, 이를 인접행렬로 저장하는 프로그램을 작성하시오.

<입력형식> 첫 줄에는 정점의 개수 N이 입력되고, 둘째 줄에는 간선의 개수 M이 입력된다. 셋째 줄부터 M+2째 줄까지는 간선이 2개의 정수로 표현되는데 앞의 정수가 출발 정점이고 뒤에 나오는 정수는 도착 정점이 된다.

(일반적으로 그래프 문제가 출제될 경우에 입력형식은 위와 같이 주어지는 경우가 대부분이므로 이 형식을 익혀두도록 하자.)

입력 (INPUT.TXT) 예 :

```

5
8
1 2
2 3
2 5
3 1
3 4
3 5
4 1
4 5

```

<풀이> 위에 예제의 형식이 일반적으로 각종 대회의 문제에서 주어지는 형식과 유사하다. 따라서 그래프를 입력하는 방법에 대해서 정확하게 이해해 둘 필요가 있다. 위 입력 값을 분석해 보면 가중치가 없는 그래프이므로 정점 간에 길이 있으면 1 없으면 0으로 저장하는 방식으로 코딩한다.
각 언어별 코드는 다음과 같다.

```

<VC>
#include <stdio.h>
#define MAX 100 //최대 정점 수
int G[MAX][MAX]; // 인접행렬
void main(void){
    FILE *in=fopen("input.txt", "r");
    int N, M, i, a, b;
    fscanf(in, "%d %d", &N, &M);
    for( i=0 ; i<M ; i++ ){
        fscanf(in, "%d %d", &a, &b);
        G[a][b] = 1;
    }
}

```

```

<VB>
Const MAX As Integer = 100 '최대 정점의 수
Dim G(MAX, MAX) As Integer '인접행렬 선언
Sub main()
    Dim N as Integer, M As Integer, I As Integer
    Dim a as Integer, b as Integer
    Open "input.txt" For Input As #1
    Input #1, N, M      'M, N을 입력받음
    For I = 1 to M      ' M개의 간선을 입력받음
        Input #1, A, B
        G(A, B) = 1;
    Next I
End Sub

```

나. 그래프의 탐색

일반적으로 그래프를 탐색하는 방법은 크게 깊이우선탐색(DFS)과 너비우선탐색(BFS)의 2가지로 나뉜다. 이 두 가지 탐색 법은 이산수학에 자세히 설명되어 있으므로 여기서는 생략하도록 한다.

이 단원에서는 인접행렬로 그래프를 저장한 상태에서 깊이우선탐색과 너비우선탐색을 각 언어로 구현한다.

1) 깊이우선탐색의 구현

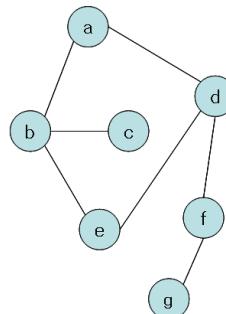
깊이우선탐색은 스택을 이용한다. 스택을 이용할 경우에도 2가지 방법이 있는데 첫 번째는 직접 스택을 만들어서 사용하는 방법이며, 두 번째 방법은 재귀 호출을 이용해서 시스템 스택을 이용하는 방법이다. 이 단원에서는 비교적 간단한 재귀를 이용한 깊이우선탐색을 다룬다.

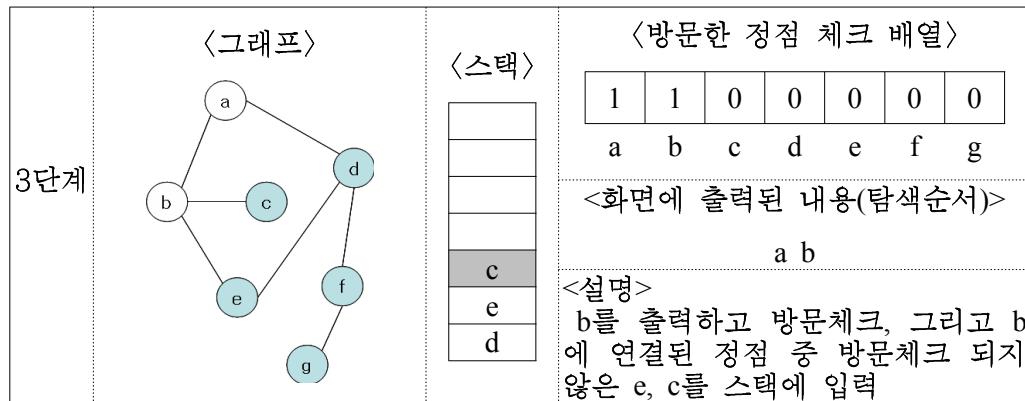
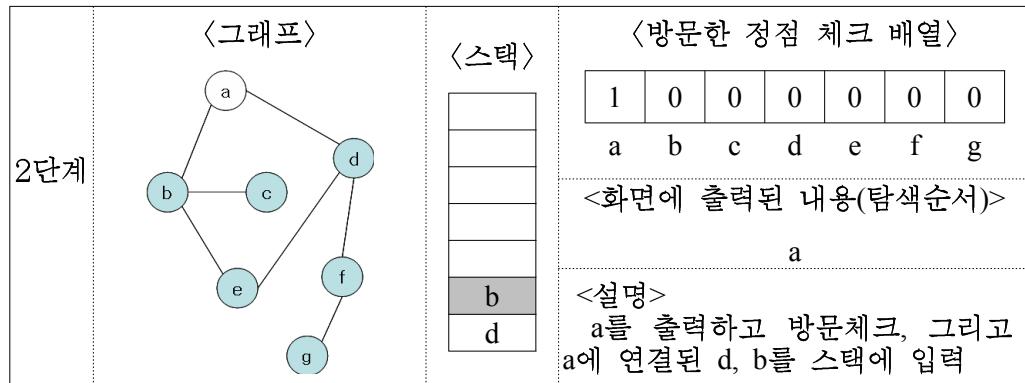
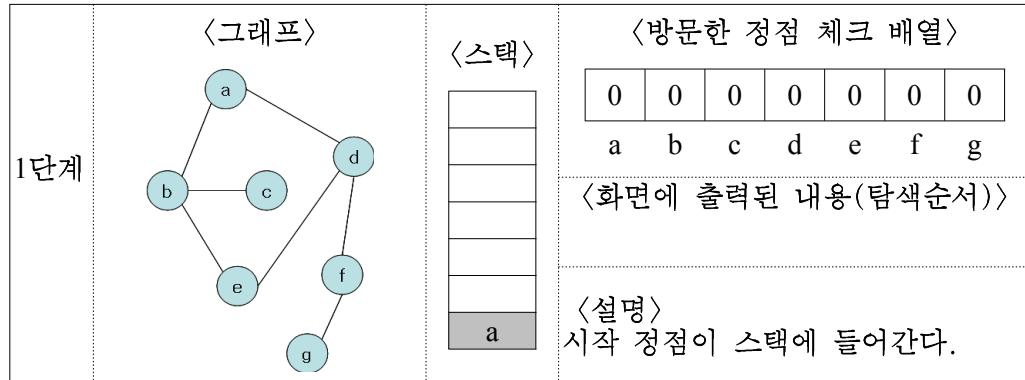
다음은 스택을 이용한 깊이우선탐색 원리를 설명한다.

- ① 시작 정점을 스택에 넣는다.
- ② 스택에서 정점 하나를 꺼내서 출력하고, 방문 정점을 체크한다.
- ③ 출력한 정점에 직접 연결된 정점들 중 방문 체크되지 않은 정점들만 스택에 차례로 넣는다. (여기서는 내림차순으로 넣는 것으로 정의한다.)
- ④ 모든 정점이 방문체크 되지 않았으면 ②번 과정으로 간다.

〈깊이우선탐색 알고리즘〉

아래 그래프를 이용해서 깊이우선탐색의 과정을 자세히 살펴보자.





4단계	〈그래프〉 	〈스택〉 <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td>e</td></tr> <tr><td>d</td></tr> </table>						e	d	〈방문한 정점 체크 배열〉 <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> </table> a b c <설명> c를 출력하고 c에 연결된 정점 중 방문하지 않은 정점이 없으므로 스택에 아무 값도 넣지 않음	1	1	1	0	0	0	0
e																	
d																	
1																	
1																	
1																	
0																	
0																	
0																	
0																	

5단계	〈그래프〉 	〈스택〉 <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td>d</td></tr> <tr><td>d</td></tr> </table>						d	d	〈방문한 정점 체크 배열〉 <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> </table> a b c e <설명> e를 출력하고 e에 연결된 정점들 중 방문하지 않은 정점 d를 스택에 입력한다.	1	1	1	0	1	0	0
d																	
d																	
1																	
1																	
1																	
0																	
1																	
0																	
0																	

6단계	〈그래프〉 	〈스택〉 <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td>f</td></tr> <tr><td>d</td></tr> </table>						f	d	〈방문한 정점 체크 배열〉 <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> </table> a b c e d <설명> d를 출력하고 d에 연결된 정점들 중 아직 체크되지 않은 f를 입력	1	1	1	1	1	0	0
f																	
d																	
1																	
1																	
1																	
1																	
1																	
0																	
0																	

7단계	<p>〈그래프〉</p>	<p>〈스택〉</p> <table border="1"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td>g</td></tr> <tr><td>d</td></tr> </table>						g	d	<p>〈방문한 정점 체크 배열〉</p> <table border="1"> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td><td>g</td></tr> </table> <p>〈화면에 출력된 내용(탐색순서)〉</p> <p>a b c e d f</p> <p>〈설명〉</p> <p>f를 출력하고 f에 연결된 정점들 중 아직 방문하지 않은 g를 스택에 입력</p>	1	1	1	1	1	1	0	a	b	c	d	e	f	g
g																								
d																								
1	1	1	1	1	1	0																		
a	b	c	d	e	f	g																		

8단계	<p>〈그래프〉</p>	<p>〈스택〉</p> <table border="1"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> </table>							<p>〈방문한 정점 체크 배열〉</p> <table border="1"> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td><td>g</td></tr> </table> <p>〈화면에 출력된 내용(탐색순서)〉</p> <p>a b c e d f g</p> <p>〈설명〉</p> <p>g를 출력하고 연결된 정점들 중 아직 방문한 정점들이 없으므로, 다시 마지막으로 스택에서 d를 꺼내고 d는 이미 체크 되었으므로 출력하지 않는다. 더 이상 스택에 내용이 없으므로 깊이우선탐색은 종료</p>	1	1	1	1	1	1	1	a	b	c	d	e	f	g
1	1	1	1	1	1	1																	
a	b	c	d	e	f	g																	

위의 과정으로 깊이우선탐색을 진행하고 깊이우선탐색의 결과는 다음과 같다.

a - b - c - e - d - f - g

위의 알고리즘을 언어별로 코딩한 것은 다음과 같다.

```

int Check[MAX]; // 각 정점의 방문여부를 판단
                // (1은 방문 0은 미방문)
void DFS(int v)           // 깊이우선탐색 함수
{
    int i;
    if(Check[v]) return; // 정점 v가 체크면 복귀
    Check[v] = 1;         // 정점 v방문 체크
    fprintf(out, "%d\n", v); // 파일로 출력
    for(i=1;i<=MAX;i++) // 현 정점에서 연결된
        if(G[v][i] && !Check[i]) // 미방문 정점을
            DFS(i); // 시작점으로 DFS재귀
}
// 이 DFS함수는 재귀 호출을 이용

```

```

Dim Check(MAX) As Integer ' 방문체크배열
                           '(1은 방문 0은 미방문)
Sub DFS(v As Integer)   ' 깊이우선탐색 함수
    Dim i as integer
    If Check(v) = 0 Then
        Check(v) = 1      ' 정점 v방문 체크
        Print #2, Cstr(v) ' 파일로 출력
        For i = 1 to N   ' 현 정점에서 연결된
            If G(v, i) And Check(i)<>1 Then
                DFS(i)      ' 미방문 정점을 다시 DFS로..
            End If
        Next i
    End If
End Sub

```

2) 너비우선탐색의 구현

너비우선탐색은 깊이우선탐색과는 달리 큐를 이용한다. 너비우선탐색은 현 정점에서 제일 가까운 정점들을 먼저 탐색하고 점점 더 멀리 탐색해나가는 기법으로 다양한 형태로 활용되므로 잘 익혀두어야 한다.

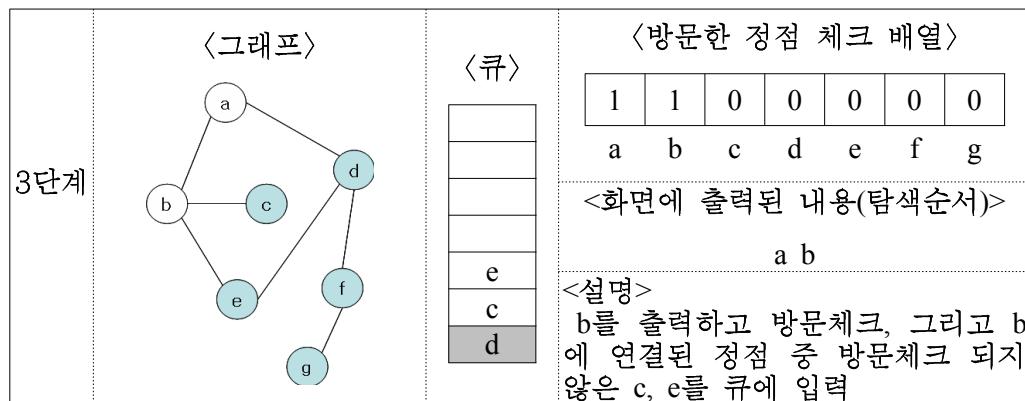
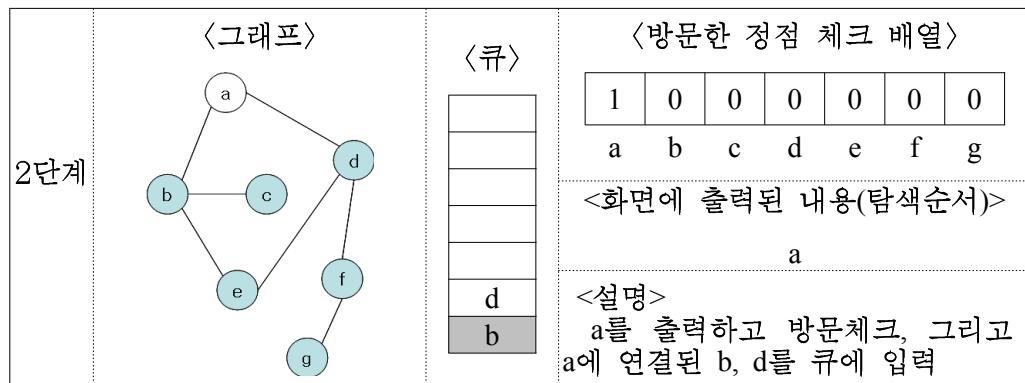
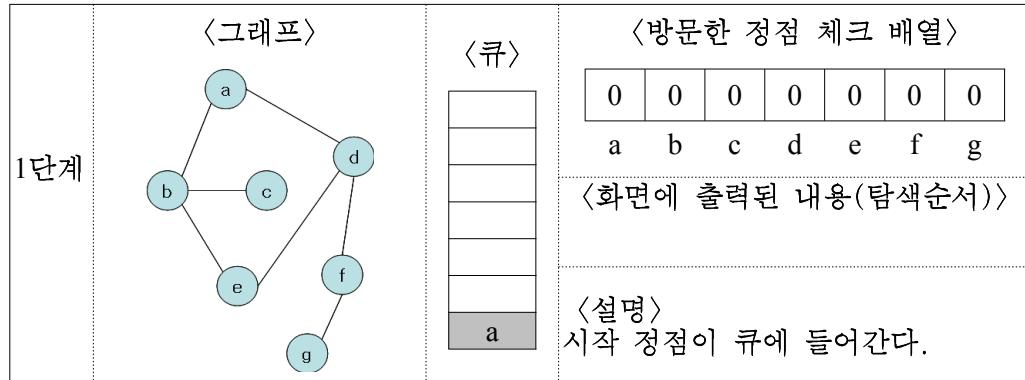
다음은 너비우선탐색의 원리를 설명한다.

- ① 시작 정점을 큐에 넣는다.
- ② 큐에서 정점 하나를 꺼내서 출력하고, 방문 정점을 체크한다.
- ③ 출력한 정점에 직접 연결된 정점들 중 방문 체크되지 않은 정점들만 큐에 차례로 넣는다. (여기서는 오름차순으로 넣는 것으로 정의한다.)
- ④ 모든 정점이 방문체크 되지 않았으면 ②번 과정으로 간다.

〈너비우선탐색 알고리즘〉

적용하는 그래프는 깊이우선탐색에서와 같은 그래프를 이용한다. 깊이우선탐색과 너비우선탐색의 차이를 살펴보면 스택과 큐를 사용한다는 것만 제외하고는 알고리즘이 거의 동일하다. 두 탐색법의 탐색결과차이는 결국 스택과 큐라는 자료구조의 차이로 이해하면 될 것이다. 탐색 과정을 자세히 살펴보자.

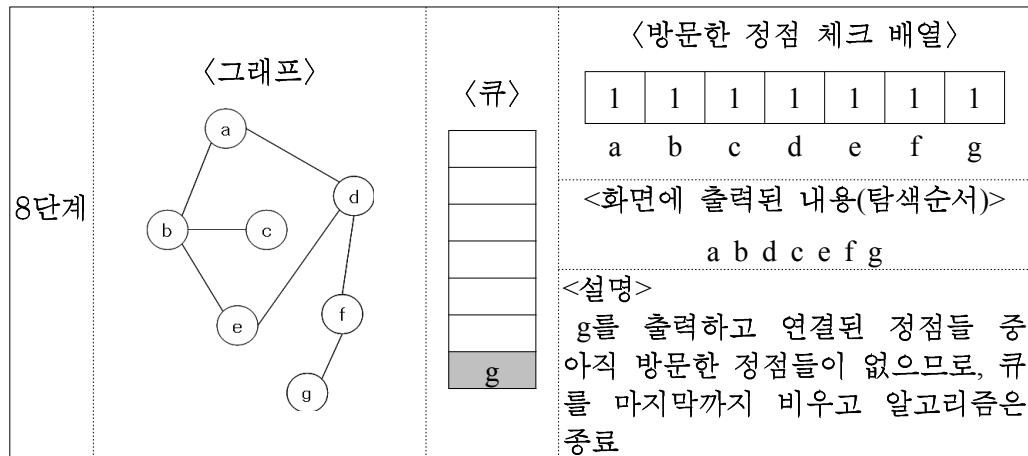
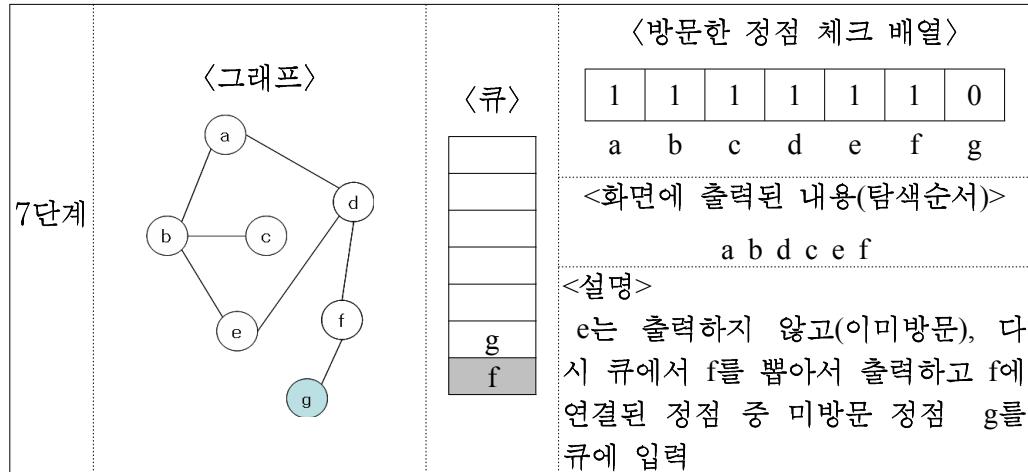
아래 구조도에서 사용된 큐는 위를 입구로, 아래를 출구로 생각하면 된다.



4단계	<p>〈그래프〉</p>	<p>〈큐〉</p> <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td>f</td></tr> <tr><td>e</td></tr> <tr><td>e</td></tr> <tr><td>c</td></tr> </table>					f	e	e	c	<p>〈방문한 정점 체크 배열〉</p> <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> </table> <p>a b c d e f g</p> <p><화면에 출력된 내용(탐색순서)></p> <p>a b d</p> <p><설명> d를 출력하고 d에 연결된 정점 중 방문하지 않은 정점 e, f를 큐에 입력</p>	1	1	0	1	0	0	0
f																		
e																		
e																		
c																		
1																		
1																		
0																		
1																		
0																		
0																		
0																		

5단계	<p>〈그래프〉</p>	<p>〈큐〉</p> <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td>f</td></tr> <tr><td>e</td></tr> <tr><td>e</td></tr> </table>						f	e	e	<p>〈방문한 정점 체크 배열〉</p> <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> </table> <p>a b c d e f g</p> <p><화면에 출력된 내용(탐색순서)></p> <p>a b d c</p> <p><설명> c를 출력하고 c에 연결된 정점은 모두 방문했으므로 입력은 없음</p>	1	1	1	1	0	0	0
f																		
e																		
e																		
1																		
1																		
1																		
1																		
0																		
0																		
0																		

6단계	<p>〈그래프〉</p>	<p>〈큐〉</p> <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td></td></tr> <tr><td>f</td></tr> <tr><td>e</td></tr> </table>							f	e	<p>〈방문한 정점 체크 배열〉</p> <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> </table> <p>a b c d e f g</p> <p><화면에 출력된 내용(탐색순서)></p> <p>a b d c e</p> <p><설명> e를 출력하고 e에 연결된 정점은 모두 방문했으므로 입력 없음</p>	1	1	1	1	1	0	0
f																		
e																		
1																		
1																		
1																		
1																		
1																		
0																		
0																		



위의 과정으로 너비우선탐색을 진행하고 너비우선탐색의 결과는 다음과 같다.

a - b - d - c - e - f - g

위의 알고리즘을 언어별로 코딩한 것은 다음과 같다.

```

<VC>
#include <queue.h>
int G[MAX][MAX]; //그래프 선언
std::queue<int> Q; // 객체 큐의 선언
int Check[MAX];// 각 정점의 방문여부를 판단

void BFS(int v){ // BFS를 v정점으로부터 탐색
    int i;
    while(1){
        if(!Check[v]){
            //방문 안한 정점의
            Check[v] = 1;
            fprintf(out, "%d\n", v);
            for(i=1;i<=N;i++)
                //인접 정점을
                if(G[v][i] && !Check[i])
                    Q.push(i); // 큐에 저장
        }
        if(!Q.empty()){
            v=Q.front(); // 큐가 빌 때 까지 반복
            Q.pop();
        }
        else break;
    }
    //VC에서는 객체큐를 이용했으므로 VB보다
    //코드가 짧고 간결함
}

```

```

<VB>
Dim Check(MAX) As Integer
Dim G(MAX,MAX) As Integer

Function Empty() As Integer '큐가 비었으면 1 아니면 0을 반환
    If Rear = Front Then
        Empty = 1
    Else
        Empty = 0
    End IF
End Function

Sub BFS(v As Integer){ '정점 v에서 너비우선 탐색 시작
    Dim i As Integer
    Do While TRUE
        If Check(v) <> 0 Then '인접정점 중
            Check(v) = 1;
            Print #2, CStr(v) ' 방문안한 정점은 큐에
            For i = 1 To N ' 삽입하는 과정
                If G(v, i) And Check(i) <> 0 Then Push i
            Next I
        End If
        If Empty() = 1 Then ' 큐가 빌 때 까지 수행
            v=Pop()
        Else
            Exit Do
        End IF
    Loop
End Sub

```

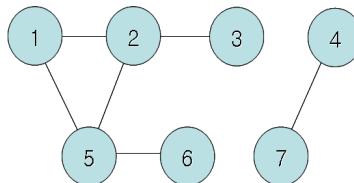
'여기서 사용된 Push, Pop 등의 함수는 앞의 큐에서 구현한 것
'을 이용한 것이고, empty()함수는 큐가 비었으면 1 아니면 0을
'반환하는 함수이다.

다. 그래프의 활용

그래프의 응용에서는 그래프에 관련 되는 실전 문제들을 통해서 그래프가 어떻게 응용되는지를 알아본다. 그리고 그래프 알고리즘 중에 대표적인 최단 경로 알고리즘도 소개한다.



문제IV.5.2 바이러스 (2004 한국정보올림피아드 지역본선기출)



신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 위의 그림과 같이 네트워크 상에서 연결되어 있다고 하자. 위의 그림을 살펴보자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크 상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.

어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

<입력형식> 입력 파일은 input.txt로 한다. 입력파일의 첫째 줄에는 컴퓨터의 수가 주어진다. 컴퓨터의 수는 100이하이고, 각 컴퓨터에는 1번부터 차례대로 번호가 매겨진다. 둘째 줄에는 네트워크 상에서 직접 연결되어 있는 컴퓨터 쌍의 수가 주어진다. 이어서 그 수만큼 한 줄에 한 쌍씩 네트워크 상에서 직접 연결되어 있는 컴퓨터의 번호 쌍이 주어진다.

<출력형식> 출력 파일은 output.txt로 한다. 1번 컴퓨터가 웜 바이러스에 걸렸을 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 첫째 줄에 출력한다.

입력파일(INPUT.TXT) 예 :
 7
 6
 1 2
 2 3
 1 5
 5 2
 5 6
 4 7

출력파일(OUTPUT.TXT) 예 : 4

<풀이> 이 문제는 그래프의 탐색으로 해결할 수 있는 문제이다. 각 컴퓨터들을 정점으로 생각하고 네트워크 관계를 노드라고 생각해서 인접행렬에 저장한 다음, 1번 컴퓨터와 연결된 컴퓨터를 찾으면 되는 문제이다.

1번 컴퓨터와 연결된 컴퓨터를 어떻게 조사할까? 생각해 보면 간단하게 해결 가능하다. 컴퓨터가 바이러스에 걸리는 순서는 관계없이 연결 관계만 조사하면 된다. 즉 1번에서 출발해서 모든 연결된 정점을 탐색하면 되는 것이다. 이 문제의 해결법은 깊이우선탐색, 너비우선탐색의 어느 방법을 이용해도 무방하다.

하지만 깊이우선탐색이 구현이 간단하므로, 깊이우선탐색으로 구현하는 것이 쉬울 것이다. 앞에 예로 주어진 탐색 프로그램을 그대로 적용시켜도 쉽게 풀 수 있다.

각 언어별로 소스 코드를 살펴보자.

— ● <VC>

```
#include <stdio.h>
#define MAX 100

int GRAPH[MAX][MAX], virus_count, n;
FILE *in=fopen("input.txt", "r"), *out=fopen("output.txt", "w");
int Check[MAX];           // 각 정점의 방문여부를 판단
                           // (1은 방문 0은 미방문)

void DFS(int v)           // 깊이우선탐색 함수
{
    int i;
    if(Check[v]) return;   // 현재 정점 v를 방문했으면 복귀
    Check[v] = 1;          // 정점 v방문 체크
    virus_count++;         // 감염 컴퓨터 계산
    for(i=1;i<=n;i++)      // 현 정점에서 연결된
        if(GRAPH[v][i] && !Check[i]) // 미방문 정점을
            DFS(i);           // 다시 DFS로 탐색
}
void main(void){
    int m, s, e, i;
    fscanf(in,"%d %d", &n, &m); //정점과 간선의 수를 받음
    for(i=0;i<m;i++){
        fscanf(in,"%d %d", &s, &e);
        GRAPH[s][e] = 1;      // 방향성이 없으므로 양쪽 다 1을 채움
        GRAPH[e][s] = 1;
    }
    DFS(1);
    fprintf(out,"%d\n", virus_count-1);
}
```

<VB>

```

Option Explicit
Const MAX As Integer = 100
Dim Check(MAX) As Integer, G(MAX, MAX) As Integer, Virus_cnt As Integer
Dim n as integer

Sub main()
    Dim i As Integer, m As Integer, s As Integer, e As Integer
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2
    Input #1, n, m           '그래프의 간선 수를 입력받음
    For i = 1 To m
        Input #1, s, e       '인접행렬 구현
        G(s, e) = 1
        G(e, s) = 1
    Next i
    DFS 1                   '깊이우선탐색
    Print #2, CStr(Virus_cnt - 1)
End Sub

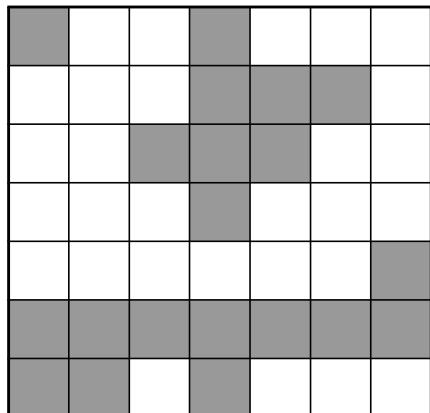
Sub DFS(v As Integer)      ' 깊이우선탐색 함수
    Dim i As Integer
    If Check(v) = 0 Then    '정점 v가 미방문 정점이면 수행
        Check(v) = 1         ' 정점 v방문 체크
        Virus_cnt = Virus_cnt + 1   ' 감염 컴퓨터 계산
        For i = 1 To n        ' 현 정점에서 연결된
            If G(v, i) And Check(i) <> 1 Then
                DFS (i)
            End If             '미방문 정점을 다시 DFS로.
        Next i
    End If
End Sub

```

2

문제IV.5.3 가장 큰 빙하는? (2004 마산시 컴퓨터 활용경진대회)

인공위성으로 남극상공을 위성사진으로 촬영하였다. 현재 우리나라의 위성 성능이 그리 좋지 않아서, 세밀한 촬영은 불가능하다. 촬영되는 내용은 다음과 같다. 2차원 격자의 형태로 촬영이 되며 0은 바다를 1은 빙하를 의미한다. 이 문제의 목적은 현재 남극에서 가장 큰 빙하의 둉어리를 찾고자 하는 것이다. 인공위성사진이 주어질 때 가장 큰 빙하의 둉어리를 찾고자 하는 것이다. 인공위성사진이 주어질 때 가장 큰 빙하의 둉어리 개수를 출력하는 프로그램을 작성하시오. (아래의 그림은 위성에서 찍은 빙하의 사진 예시이다.)



(□ : 바다(0), ■ : 빙하(1))

<입력형식> 파일명 input.txt이고, 첫 번째 줄에는 총 사진의 가로, 세로의 크기가 입력되고(3~100), 둘째 줄부터는 각 사진의 내용이 0과 1로 표현된다. 여기서 0은 바다를 1은 빙하 둉어리를 의미한다. 각 값들 사이에는 공백으로 구분된다.

<출력형식> 파일명은 output.txt로 하고 출력은 출력파일에 가장 큰 빙하의 둉어리 수를 출력 한다.

입력(INPUT.TXT) 예 :

7

```

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

```

출력(OUTPUT.TXT) 예 :

11

<풀이> 이 문제는 Flood Fill이라는 알고리즘을 사용하면 되는 문제이다. 여기서 Flood Fill이라는 방법이 그래프의 탐색을 바탕으로 한 알고리즘이라는 것을 이해하면 쉽게 구현할 수 있다. 위의 사진에서 1이 그래프의 정점이고 0은 점점이 없는 그래프로 생각한다.(실제 그래프 형태는 아니지만 그래프로 생각하고 풀면 쉽다.)

그리고 이 그래프가 연결될 수 있는 위치는 상, 하, 좌, 우 4곳으로 생각하고 만약 상, 하, 좌, 우에 연결된 1이 있으면 두 정점은 연결된 것으로 가정한다. 이렇게 이해하면 위의 예시는 3개의 그래프로 구성된 맵이라는 것을 알 수 있다. 첫 번째 그래프는 1개의 정점으로, 두 번째 그래프는 8개의 정점, 마지막 그래프는 11개의 정점으로 구성된다.

여기서 문제는 다음과 같은 방법으로 해결하였다.

- ① 배열의 (0, 0)에서 순차적으로 1이 나올 때까지 검색한다.
- ② 배열의 방에서 1이 검색되면 하나의 그래프라고 생각하고 이 정점을 출발점으로 하여 DFS를 시작한다. (이 때 방문한 정점은 모두 0로 고친다.)
- ③ 각 정점을 방문할 때마다 정점의 수를 카운트한다.
- ④ DFS가 끝나면 정점의 수를 저장한다.
- ⑤ 다시 ②번 과정으로 가서 남은 배열의 부분을 검색한다.
- ⑥ 배열의 모든 1을 다 검색하면 알고리즘을 마친다.

이렇게 모든 배열을 검사하면 부분 그래프를 모두 검사하게 되고 각 그래프의 정점의 수를 알 수 있다. 이 정점의 수들 중 가장 많은 정점의 수가 바로 우리가 구하고자 하는 정답이 된다.

위의 알고리즘은 앞 단원에서 언급한 DFS를 응용한 것이므로 잘 분석해 둘 필요가 있다.

언어 별 소스 코드를 직접 분석해 보고 이해할 수 있도록 하자.

————— ● <VC>

```
#include <stdio.h>
#define MAX 100

int ice_map[MAX][MAX], max_ice, cur; // 기본 변수 선언
void DFS(int a, int b){ // 범람 채우기 알고리즘
    ice_map[a][b] = 0; // 검색한 빙하는 바다로 바꿔서 재검색 피함
    cur++; // 빙하 수 증가
    if(ice_map[a+1][b]) DFS(a+1, b); // 맵 값이 1이면 다시 재귀 호출
    if(ice_map[a][b+1]) DFS(a, b+1);
    if(ice_map[a-1][b]) DFS(a-1, b);
    if(ice_map[a][b-1]) DFS(a, b-1);
}
void main(void){
    int n, i, j;
    FILE *in=fopen("input.txt", "r"), *out=fopen("output.txt","w");
    fscanf(in,"%d", &n);
    for(i=0;i<n;i++) // 빙하 입력 받음
        for(j=0;j<n;j++)
            fscanf(in,"%d", &ice_map[i][j]);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(ice_map[i][j]){
                cur = 0;
                DFS(i, j);
                max_ice = (max_ice < cur)? cur : max_ice;
            } // 최대 빙하 수를 저장
    fprintf(out, "%d" , max_ice); // 최대 빙하 수 출력
}
```

<VB>

```

Option Explicit
Const MAX As Integer = 100
Dim ice_map(MAX, MAX), max_ice, cur ' 기본 변수 선언

Sub DFS(a As Integer, b As Integer) ' 범람 채우기
    ice_map(a, b) = 0
    cur = cur + 1
    If ice_map(a + 1, b) = 1 Then DFS a + 1, b '4 방향 검색
    If ice_map(a, b + 1) = 1 Then DFS a, b + 1 '10면 재귀 호출
    If ice_map(a - 1, b) = 1 Then DFS a - 1, b
    If ice_map(a, b - 1) = 1 Then DFS a, b - 1
End Sub

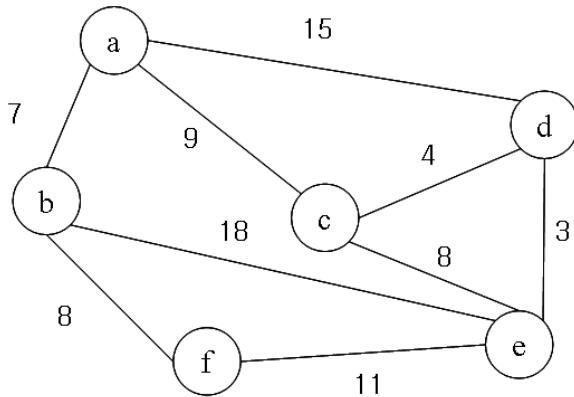
Sub main()
    Dim n As Integer, i As Integer, j As Integer
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2
    Input #1, n
    For i = 1 To n
        For j = 1 To n
            Input #1, ice_map(i, j) '파일에서 입력
        Next j, i
    For i = 1 To n
        For j = 1 To n
            cur = 0
            DFS i, j ' 범람 채우기 시작(깊이우선탐색)
            max_ice = IIf(max_ice < cur, cur, max_ice) '최대값 저장
        Next j, I
        Print #2, CStr(max_ice - 1) ' 최대값 파일로 출력
    End Sub

```

— ● 최단경로 알고리즘

Dijkstra의 최단경로 알고리즘은 가중치 그래프 상에서 특정 출발점에서 다른 모든 정점들까지의 최단경로를 구하는 알고리즘이다. 이 알고리즘의 원리는 다음과 같다.

다음은 $a \sim f$ 도시를 연결한 도로망을 가중치 그래프로 나타낸 것이다. 가중치는 두 도시간의 이동 시간을 나타낸다. 이 그래프를 이용해서 최단경로 알고리즘을 이해해보자. (여기서는 출발점을 a 라고 가정한다.)



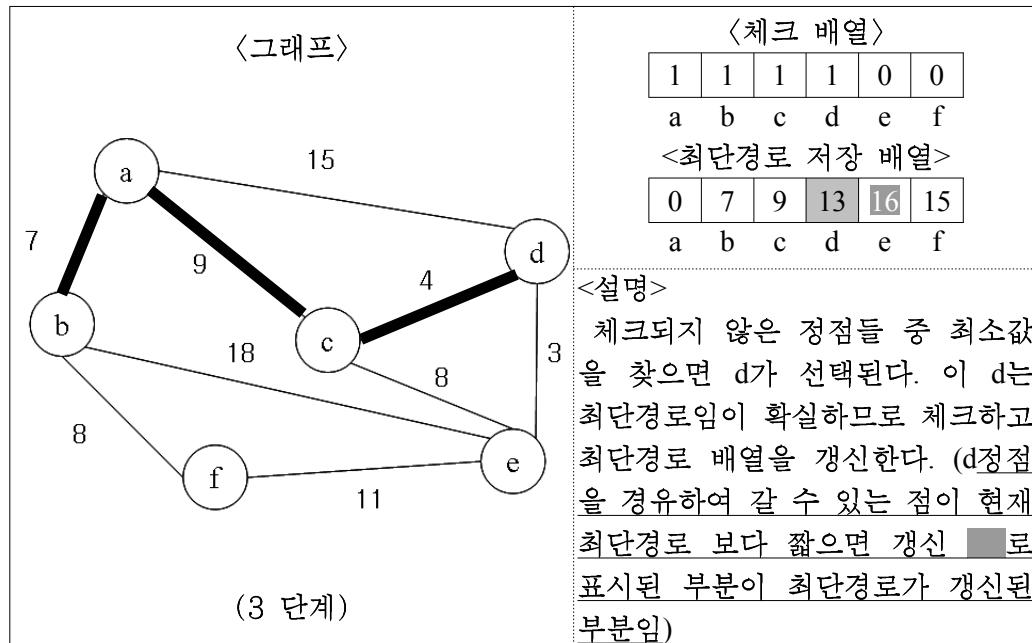
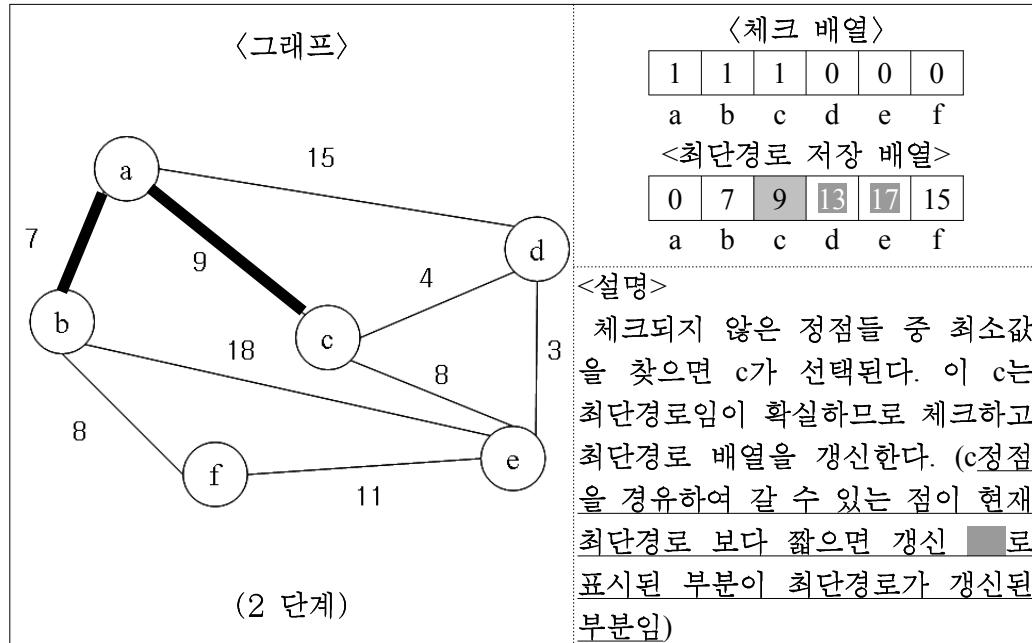
- ① 최단경로를 찾았는지 체크할 배열을 준비한다.
- ② 출발점 a 에서 직접 연결된 도시들 중 가장 가중치가 적은 b 는 최단경로임이 확실하므로 b 도시까지의 최단경로는 찾은 것이다.(b 도시 체크)
- ③ 최단 경로로 확정된 도시를 k 라고 두고 a 에 연결된 나머지 도시들과 도시 k 를 경유해서 갈 수 있는 도시들도 a 에서 직접 연결된 것과 같은 도시들로 가정하여 다시 가중치를 계산
- ④ a 에서 직접 연결된 도시들(새로 추가된 도시들 포함) 중 가중치가 제일 적은 도시를 $k2$ 라고 하면 a 에서 $k2$ 까지도 최단 경로임이 확실하다.
- ⑤ 모든 정점이 체크될 때까지 ③~④의 과정을 반복한다.

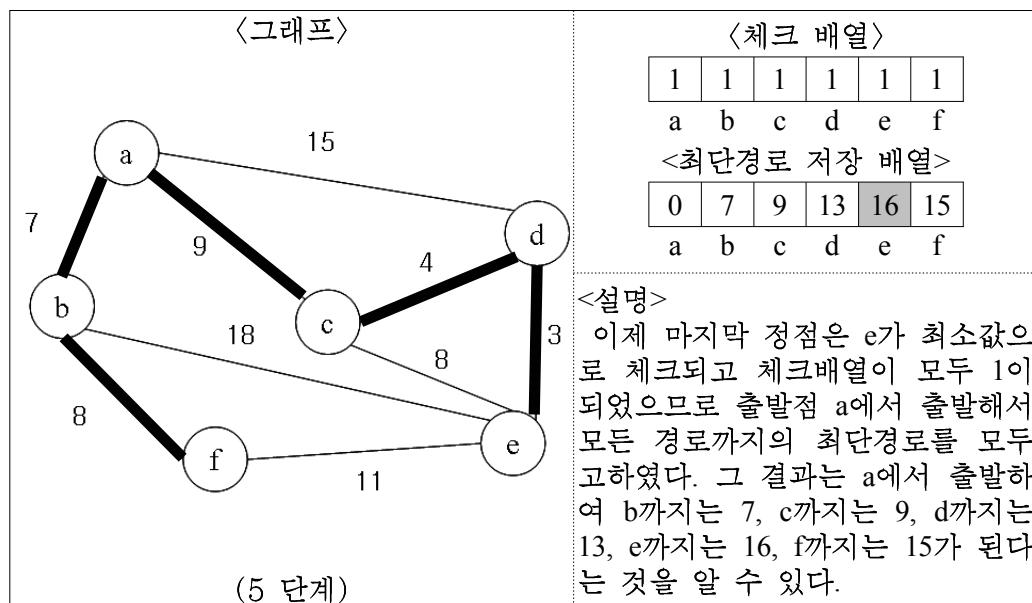
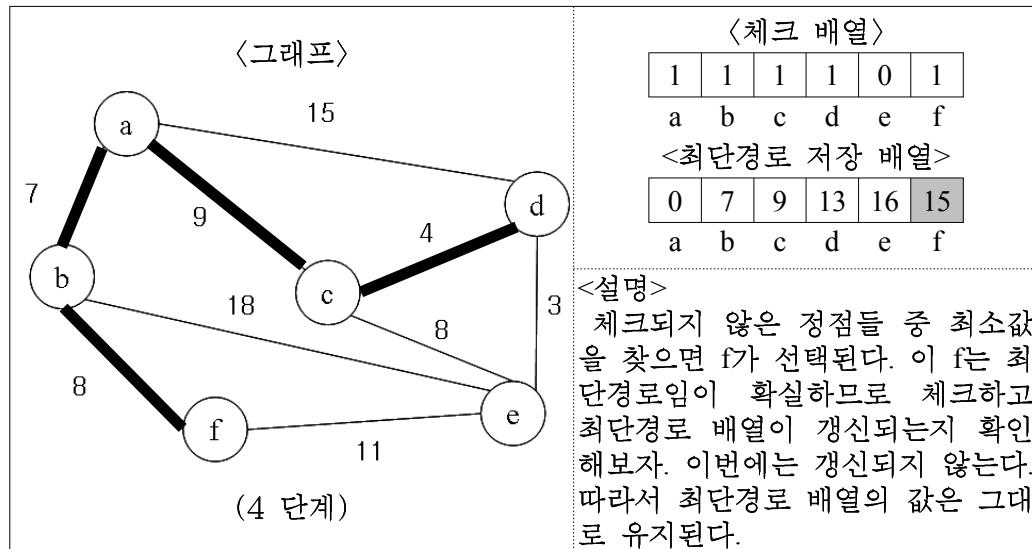
<Dijkstra의 최단경로 알고리즘>

위의 알고리즘을 자세히 분석해보자.

〈그래프〉	〈체크 배열〉																								
<p>(초기상태)</p>	<table border="1" style="margin-bottom: 10px;"> <tr> <td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td></tr> </table> <p>〈최단경로 저장 배열〉</p> <table border="1" style="margin-bottom: 10px;"> <tr> <td>0</td><td>7</td><td>9</td><td>15</td><td>-</td><td>-</td></tr> <tr> <td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td></tr> </table> <p>〈설명〉</p> <p>체크배열은 출발점만 체크해두고, 최단경로 저장배열에는 a에서 직접 연결된 정점들까지의 가중치를 저장하고 직접 연결되지 않은 값들은 충분히 큰 정수를 저장해 둔다. 여기서는 ‘-’로 표현</p>	1	0	0	0	0	0	a	b	c	d	e	f	0	7	9	15	-	-	a	b	c	d	e	f
1	0	0	0	0	0																				
a	b	c	d	e	f																				
0	7	9	15	-	-																				
a	b	c	d	e	f																				
	<table border="1" style="margin-bottom: 10px;"> <tr> <td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td></tr> </table> <p>〈최단경로 저장 배열〉</p> <table border="1" style="margin-bottom: 10px;"> <tr> <td>0</td><td>7</td><td>9</td><td>15</td><td>25</td><td>15</td></tr> <tr> <td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td></tr> </table>	1	0	0	0	0	0	a	b	c	d	e	f	0	7	9	15	25	15	a	b	c	d	e	f
1	0	0	0	0	0																				
a	b	c	d	e	f																				
0	7	9	15	25	15																				
a	b	c	d	e	f																				

〈그래프〉	〈체크 배열〉																								
<p>(1 단계)</p>	<table border="1" style="margin-bottom: 10px;"> <tr> <td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td></tr> </table> <p>〈최단경로 저장 배열〉</p> <table border="1" style="margin-bottom: 10px;"> <tr> <td>0</td><td>7</td><td>9</td><td>15</td><td>25</td><td>15</td></tr> <tr> <td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td></tr> </table> <p>〈설명〉</p> <p>체크되지 않은 정점들 중 최소값을 찾으면 b가 선택된다. 이 b는 최단경로임이 확실하므로 체크하고 최단경로 배열을 갱신한다. (b정점을 경유하여 갈 수 있는 점이 현재 최단경로 보다 짧으면 갱신 로 표시된 부분이 최단경로가 갱신된 부분임)</p>	1	1	0	0	0	0	a	b	c	d	e	f	0	7	9	15	25	15	a	b	c	d	e	f
1	1	0	0	0	0																				
a	b	c	d	e	f																				
0	7	9	15	25	15																				
a	b	c	d	e	f																				





위의 결과를 직접 눈으로 확인해보자. 알고리즘이 정확하게 동작한다는 것을 알 수 있을 것이다. 이 알고리즘은 다양한 분야에 활용되므로 꼭 익혀두어야 한다. 2004년 교원프로그래밍경진대회 5번 문항에도 출제된 바가 있다. 학생 정보올림피아드에도 출제되는 경우가 있으므로 익혀두면 여러모로 편리하다.



연습문제



Dijkstra 알고리즘을 바로 이용할 수 있는 함수형태로 작성하시오.

(단, 그래프에 가중치가 있는 정점은 가중치로 저장된 상태이고 연결되지 않은 정점은 0이 저장된 상태이다.)

함수의 Proto Type는 다음과 같다.

dijkstra(출발 정점)



풀



문제의 조건대로 함수를 만든 결과이다.

— ● <VC>

```

int map[MAX][MAX];           //전체 그래프를 저장할 인접행렬
int chk[MAX], distance[MAX]; //layover은 경유하는 경로 저장하는 배열

void dijkstra(int st){        // 최단경로 알고리즘 Dijkstra
    int min, i, j, p, c=0;
    for(i=1;i<=n;i++){
        chk[i] = 0;
        distance[i] = (map[a][i])? map[a][i] : 0x7fffffff;
    }                                //0은 길이 없으므로 최대정수를 저장
    chk[st] = 1; //출발 정점은 최단경로이므로 체크
    while(1){
        min=0x7fffffff;           // 0x7fffffff는 최대 정수값
        for(j=1;j<=n;j++)
            if(min>distance[j] && !chk[j]){ //체크되지 않은 정점 중 최소 값
                min = distance[j];
                p = j;
            }                                // 최소 정점을 p로 둠
        if(min==0x7fffffff) break; // 모든 정점이 체크라면 종료
        chk[p] = 1; // p가 최소임이 확실하므로 체크
        for(j=1;j<=n;j++)
            if(map[p][j] && distance[j] > distance[p] + map[p][j])
                distance[j] = distance[p] + map[p][j]; // 최단경로 갱신
    }
}

```

— ● <VB>

```
Dim map(MAX, MAX) As Integer    '인접행렬
Dim chk(MAX) As Integer, distance(MAX) As Integer  '필요한 배열 선언
Dim n As Integer, m As Integer, w As Integer
Sub dijkstra(a As Integer, b As Integer)      '다익스트라 알고리즘
    Dim min As Integer, i As Integer, j As Integer, c As Integer, p As Integer
    For i = 1 To n
        chk(i) = 0      '체크배열 초기화
        distance(i) = If(map(a, i) <> 0, map(a, i), 9999)  '최단경로 초기화
    Next I
    Do While True
        min = 9999
        For j = 1 To n
            If min > distance(j) And chk(j) = 0 Then '체크 되지 않은 최소정점
                min = distance(j) : p = j          '을 찾는다. 그 정점은 p
            End If
        Next j
        If min = 9999 Then Exit For      '모든 정점이 체크되었으면 종료
        chk(p) = 1                      '최단경로가 확실한 p정점을 체크
        For j = 1 To n                  'p를 경유한 정점을 갱신
            If map(p, j) <> 0 And distance(j) > distance(p) + map(p, j) Then
                distance(j) = distance(p) + map(p, j)
            End If
        Next j
    Loop
End Sub
```

정보올림피아드 및 프로그래밍 교육 교재



알고리즘 설계법

제 V 장은 정보올림피아드를 지도하거나 교원프로그
래밍경진대회 4, 5번 문제를 해결하고자 하는 선생님들께
도움이 되는 내용들로 구성되어 있습니다.

1. 알고리즘 설계론

우리는 앞장에서 검색과 정렬을 비롯한 다양한 알고리즘을 배웠다. 이런 알고리즘은 어떻게 만들어질까? 이번 장에서는 알고리즘을 설계하는 방법에 대해서 다룬다. 본 장은 교원프로그래밍경진대회의 고급문제(주로 4, 5번)나 정보올림피아드를 지도하는 교사들에게는 중요한 부분이지만 프로그래밍을 처음 접하는 선생님들은 이 장을 마음 편하게 한 번 쭉 읽어보면 된다.

한국정보올림피아드 지도를 해본 선생님이라면 학생들이 다루고 있는 문제들이 쉽지 않음을 알고 있을 것이다. 풀이를 봐도 동적계획법이니 백트래킹이니 하는 낯선 단어들이 등장하여 이해하기가 쉽지 않았을 것이다. 이번 장에서는 대표적인 알고리즘 설계기법인 그리디, 백트래킹, 분할정복, 메모리핑션, 동적계획 기법에 대해서 알아본다. 그리고 이해를 돋기 위해 하나의 문제를 가지고 각 기법들로 직접 설계 풀이하여 각 기법들의 장점과 단점을 알아본다. 어떤 문제에 어떤 설계기법을 사용해야 할지는 선생님들의 판단에 맡긴다.

다음은 이번 장에서 다룰 문제이다. 이 문제는 아주 단순한 문제이므로 이해하기 어렵지는 않을 것이다. 이번 장에서는 이 문제를 다양한 알고리즘 설계법을 이용해서 풀이할 것이며, 그 내용을 분석할 것이다.



문제 V.1.1 거스름돈

어떤 물건을 사고 남은 돈을 거스름돈이라고 한다. 가게 주인은 거스름돈을 지불할 때 가능한 한 적은 수의 동전으로 지불하고자 한다. 이 문제를 해결할 프로그램을 작성하시오.

<입력형식> 입력파일명은 input.txt이고, 첫 번째 줄에는 거스름돈의 총액수인 M이 입력된다. 둘째 줄에는 동전의 종류 N이 입력되고 세 번째 줄에는 N가지의 동전의 값이 주어진다.

<출력 형식> 파일명은 output.txt이다. 주어진 거스름돈을 지불할 때 가능한 가장 적은 동전의 수를 출력한다.

입력(INPUT.TXT)	예 :	1300
		4
		10 50 100 500

출력(OUTPUT.TXT)	예 :	5
----------------	-----	---

위의 5는 500, 500, 100, 100, 100의 동전을 의미한 이 보다 적은 수의 동전으로 지불할 수 있는 방법은 없다. 위 문제의 풀이는 각 단원별로 다른 알고리즘 설계법으로 설명한다.

2. 그리디(Greedy) 설계 기법

가. 그리디 설계기법이란

스크루지 이야기를 들으면 제일 먼저 떠오르는 단어가 있을 것이다. 그것은 바로 욕심쟁이이다. 그리디 설계 기법은 이러한 욕심쟁이들이 생각하는 과정과 같은 방식으로 알고리즘을 설계하는 방법이다. 즉, 순서대로 어떤 값을 선택해야 하는 경우, 이전에 결정했거나 앞으로 결정할 선택과는 관계없이 오직 그 당시 가장 이득이 되는 선택만을 취하는 방법이다.

이 방법은 최적화문제를 푸는데 종종 있으나 항상 최적해를 보장해주지는 않는다. 하지만 간단히 알고리즘을 설계할 수 있어서 문제를 처음 접근할 때 시도 해보는 방법으로 이용하는 경우가 종종 있으며, 경우에 따라서는 매우 효율적이고 간단한 알고리즘이 만들어지는 경우도 있다. 그리디로 설계했을 경우에는 항상 최적해를 보장해 주지 않으므로 수학적으로 최적해가 됨을 증명하는 과정이

필요하다. 하지만 이 과정은 여기서 다루기에는 적합한 내용이 아니므로 생략한다. 참고로 앞장과 이산수학장에서 소개된 알고리즘들 중 Dijkstra의 최단경로 알고리즘, Prim의 최소비용신장트리 등이 그리디 설계기법으로 만들어진 대표적인 알고리즘으로 효율이 좋으며 이해가 쉬운 알고리즘 중 하나이다.



문제 V.2.1 거스름돈의 그리디 알고리즘 설계

거스름돈 문제를 그리디로 설계해보자.

첫 번째로 생각할 것이 가장 이득이 되는 선택이 무엇인지를 결정하는 것이다. 이 문제의 목적은 가장 적은 동전의 수이므로 거슬러 줄 수 있는 동전 중에 가장 액수가 큰 동전을 지불하는 방법이 가장 이득이 되는 방법이다. 만약 이 방법보다 더 이득이 되는 방법이 있다면 그 방법을 기준으로 알고리즘을 설계하면 된다.

그리디로 설계된 기본적인 알고리즘은 다음과 같다.

(선생님들께서도 다양한 방법으로 알고리즘을 직접 설계해보기 바란다.)

- ① M원을 지불하기 위해 가장 액수가 큰 동전부터 차례로 검사한다.
- ② 만약 액수가 큰 동전 k 원이 거스름돈 M 원 같거나 적다면 그 동전을 선택한다. 그렇지 않다면 다음으로 큰 동전을 고르고 조건 $M \leq k$ 를 만족할 때 까지 반복한다. (만약 끝까지 조건을 만족할 수 없다면 지불 할 수 있는 방법이 없다.)
- ③ 선택한 동전으로 일단 지불하고 남은 거스름돈을 계산한다.

$$M = M - k$$
, 지불한 동전 수를 1증가 시킨다.
- ④ M 이 0이 아니라면 다시 ②과정으로 간다.
- ⑤ 지불한 동전 수를 출력한다.

〈그리디로 설계한 거스름돈 알고리즘〉

알고리즘을 자연어로 표현하다보니 어색한 부분도 있지만 이해하기 어려운 알고리즘은 아니다.

그렇다면 그리디로 알고리즘으로 1300원을 지불하는 과정을 분석해보자.
주어진 금액이 1300원이다. 현재 가장 큰 단위의 동전이 500원이므로 500원부터 지불하기 시작한다.

— ● 동전의 종류



500원



100원



50원



10원

<지불해야 할 돈>

<지불된 돈>

500원을 집는다.

1300원



500원을 집는다.

800원



500원을 집을 수 없다.

300원



100원을 집는다.

300원



<지불해야 할 돈>

<지불된 돈>

500원을 집을 수 없다.

200원



100원



500원을 집을 수 없다.

100원



100원을 집는다.

0원



위의 그림은 그리디 알고리즘을 차례로 설명한 것이다. 지불이 완료된 결과를 보면 5개의 동전을 지불할 수 있음을 알 수 있다. 위의 1300원의 예시에서는 더 이상 적은 동전의 수로 지불할 수 있는 방법은 존재하지 않으므로 최적해를 구하는데 성공했다.

각 언어별 프로그램은 다음과 같다.

— ● <VC>

```
#include <stdio.h>
#define MAX 4

// 이 프로그램은 그리디 기법으로 작성된 프로그램입니다.
// 최적해를 찾을 수 없는 경우도 있습니다.
// 파일입출력을 표준입출력으로 바꿈(실습의 용이성을 위해)
// 이퀄리체로 표시된 부분이 그리디기법을 사용한 부분임.
int coin[MAX] = {500, 100, 50, 10};
// 동전 종류를 바꿔가면서 테스트 가능

int main(void)
{
    int money, i, cnt = 0;
    printf("거스름 돈을 입력하시오. ");
    scanf("%d", &money);

    while(money){          // 거스름돈의 지불이 완료될 때까지 반복
        for( i = 0 ; i < 4 ; i++ )
            if( money - coin[i] >= 0 ){
                // 액수가 큰 동전부터 처리(그리디)
                money = money - coin[i];
                break;
            }
        cnt++; // 사용한 동전 수 증가
    }

    printf("%d개 입니다.\n", cnt);
    return 0;
}
```

 <VB>

```
Option Explicit
```

```
Dim coin As Variant
```

```
Sub main()
```

```
    Dim money As Integer, i As Integer, cnt As Integer
    coin = Array(500, 100, 50, 10)      '기본 돈 단위 입력
    money = InputBox("거스름 돈을 입력하시오.")
    Do While money <> 0                '그리디로 설계된 부분
        For i = 0 To 3
            If money - coin(i) >= 0 Then
                money = money - coin(i)
                Exit For
            End If
        Next i
        cnt = cnt + 1
    Loop
    Debug.Print CStr(cnt) + "개입니다."
End Sub
```



문제 V.2.2

위의 거스름돈 문제가 1300원의 거스름돈을 10, 50, 100, 500의 4종류의 동전을 이용하여 거슬러 줄 경우에 최적해를 구할 수 있음을 보았다. 하지만 대회에서 이 문제를 그리디로 해결하면 모든 경우의 최적해를 구할 수 없다. 그 반례를 제시하고 그 이유를 간단히 설명하시오.

<풀이> 반례

위의 문제에서 동전의 종류만 하나 더 추가한 경우를 생각해 보자. 만약 10, 50, 100, 500원의 동전에 600원 짜리 동전을 추가하여 5가지 동전을 사용할 수 있다고 가정하자. 그리고 거스름돈은 1500원이라고 가정한다. 이 경우에 위의 그리디 알고리즘으로 해결하면 다음과 같은 정답을 출력할 것이다.

{600, 600, 100, 100, 100}으로 지불하고 총 5개의 동전을 사용하게 된다.

하지만 최적해는 {500, 500, 500}의 3개의 동전으로 지불하는 방법이다.

구할 수 없는 이유

주어진 동전의 종류에 따라서 그리디가 성립될 수도 성립되지 않을 수도 있다. 각 동전의 종류를 오름차순으로 정렬할 경우 가장 액수가 큰 동전이 나머지 동전들의 배수가 되면, 즉 액수가 적은 모든 동전이 액수가 큰 동전의 약수가 되는 동전 구성일 경우 이 문제는 그리디로 해결 가능하다.

그 이유는 현재 상태에서 가장 액수가 큰 동전을 지불하는 것이 최적임이 보장되기 때문이다.

이 문제에서 10, 50, 100, 500원 동전 셋은 위 조건을 만족하므로 그리디로 해답을 구할 수 있으나, 여기에 600원 동전이 추가되면 600원과 500원은 서로 약수, 배수 관계가 될 수 없으므로 500원으로 지불해야 할지 600원으로 지불해야 할지를 고려해야 한다. 하지만 그리디는 이러한 과정이 없으므로 최적해를 구할 수 없는 경우가 발생하고 거스름돈 문제를 대회장에서 그리디로 풀었다면 만점을 얻을 수 없을 것이다.

3. 백트래킹(Backtracking) 설계 기법

가. 백트래킹 설계 기법이란

백트래킹은 어떤 집합에서 어떤 기준을 만족하면서 그 집합에 속한 대상의 순서를 선택하는 문제를 푸는 데 유용하다.

백트래킹의 기본은 해가 될 수 있는 모든 조합을 체계적으로 모두 검색하는 방법으로 항상 최적해를 찾을 수 있다. 하지만 모든 조합을 검사하게 되므로 시간이 많이 걸린다는 것이 단점이다. 일반적으로 백트래킹으로 알고리즘을 설계하면 $O(2^n)$ 이 되므로 제한된 시간 내에 답을 찾기가 어렵다. 하지만 문제의 특성에 따라서 잘 활용하면 최적해를 찾는데 도움이 되는 알고리즘 설계법이다.

백트래킹에서 제일 중요한 요소는 가능한 해집합을 나타내는 상태공간트리를 만드는데 있다. 상태공간트리는 실제 트리를 만드는 것이 아니라 함수의 호출관계를 트리처럼 표현한 것을 의미하며, 이렇게 만들어진 트리의 모든 노드를 순회하면서 정답이 되는 조합을 발견하는 방식이다. 일반적으로 백트래킹 설계기법에서 상태공간트리의 순회 시에 주로 이용되는 것이 바로 전위순회이다.



문제 V.3.1 거스름돈의 백트래킹 알고리즘 설계

거스름돈 문제를 백트래킹으로 설계해보자.

백트래킹으로 주로 이용되는 것이 트리의 전위순회법이었다. 전위순회는 일반적으로 재귀호출로 이루어지므로 백트래킹의 설계에 있어서 제일 중요한 점이 재귀호출이다.

재귀호출할 함수의 이름을 back()으로 이 함수를 하나의 노드로 생각한다. 이 노드는 k개의 자식노드를 가지는데 k가 의미하는 것은 지불할 수 있는 동전의 종류의 수를 의미한다. 즉 각 노드 별로 500, 100, 50, 10원을 지불한다는 의미가 된다. 순서는 오름차순이나 내림차순이나 크게 관계는 없다.

기본적인 알고리즘은 다음과 같다.

```

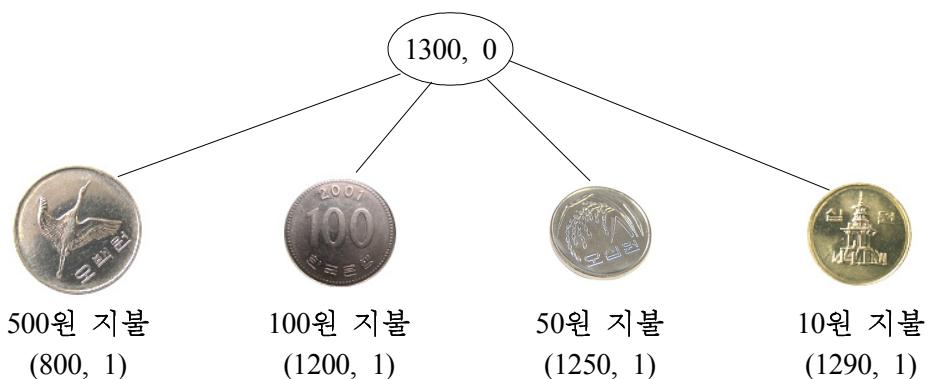
Back( int M, int k ) // M은 지불할 액수, k는 사용한 동전 수
    M < 0 이면 return
    M = 0 이면{
        현재 최소동전 min값보다 k가 적으면 min = k
        return
    }
    Back( M - 500, k + 1) // 4개의 노드를 가지는 트리의 전위순회
    Back( M - 100, k + 1)
    Back( M - 50, k + 1)
    Back( M - 10, k + 1)

```

〈백트래킹 알고리즘〉

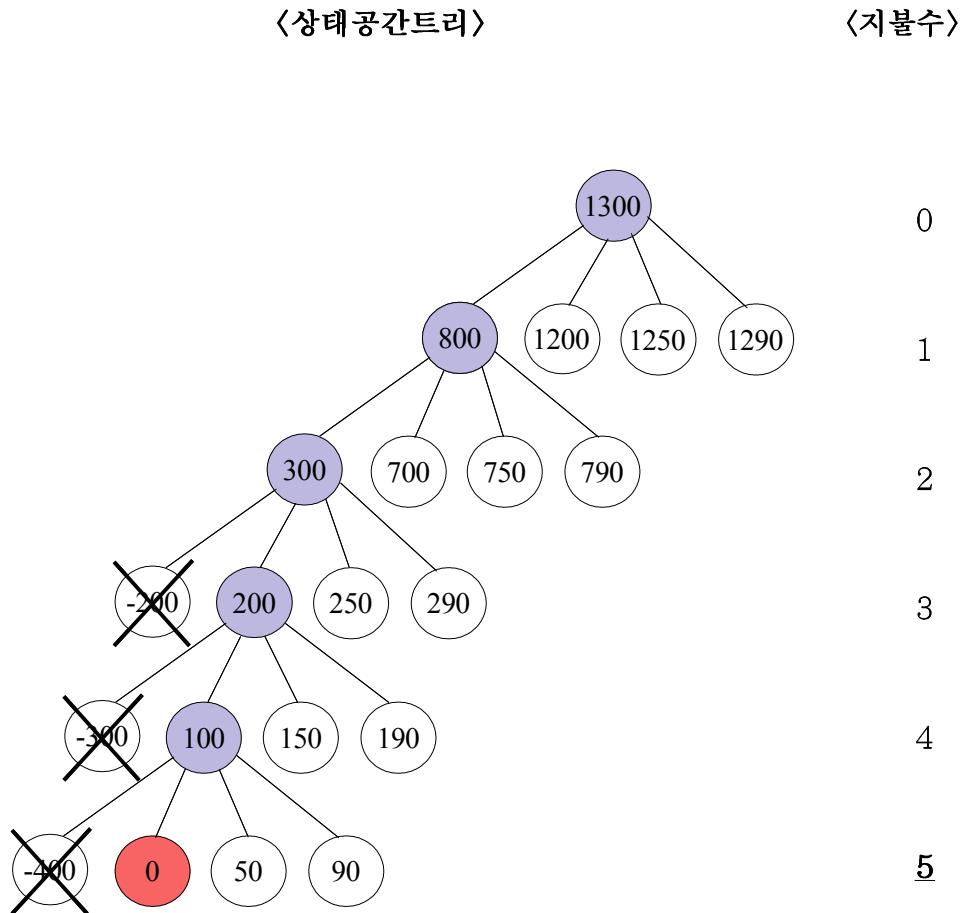
위 알고리즘은 간단하게 중간언어로 표현했다. 출발은 $\text{Back}(1300, 0)$ 으로 출발한다.

상태공간 트리의 탐색 과정을 그림으로 살펴보면 다음과 같다. 다음이 기본 트리이다.



위의 트리를 상태공간트리로 해서 전위순회하면서 지불하는 방법과 동전의 수를 구하면 된다. 위의 각각 $(800, 1)$, $(1200, 1)$, $(1250, 1)$, $(1290, 1)$ 이 각각 다시 새로운 루트가 되어 위의 과정을 반복한다. 자세한 과정을 살펴보자.

아래 과정은 처음으로 지불 가능한 방법을 발견한 것을 보여준다.

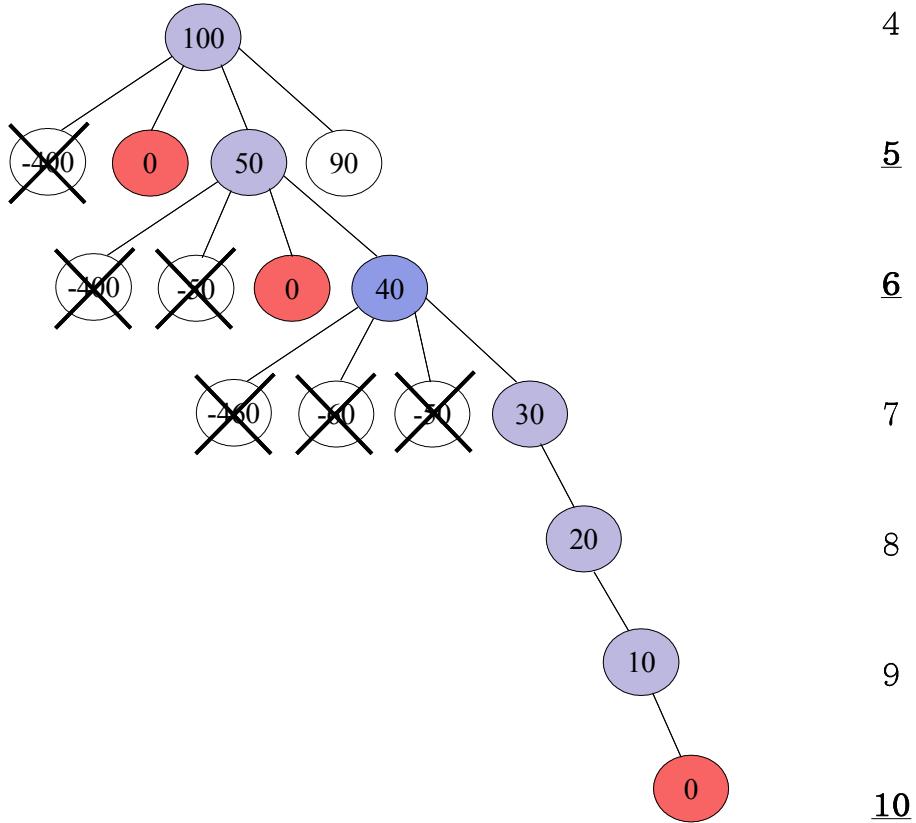


위의 과정은 `back()`의 호출과 복귀 과정을 보여준다. $M \mid 0$ 이고 k 가 5인 점에서 처음으로 지불 방법을 찾았으므로 \min 에 5를 저장하고 계속 백트랙하게 된다.

다음으로 2번째 ~ 3번째 방법을 찾는 상태공간 트리를 계속해서 살펴보자. 아래 트리는 앞페이지의 상태공간트리와 연결되는 것이다. 이 과정을 자세하게 이해하고 있어야 백트래킹 설계기법을 정확하게 이해할 수 있다.

〈상태공간트리〉

〈지불수〉



위의 과정에서 2번째 지불방법은 6개의 동전으로 (500, 500, 100, 100, 50, 50)으로 3번째 방법은 10개의 동전으로 (500, 500, 100, 100, 50, 10, 10, 10, 10, 10)으로 지불 가능한 경로를 보여준다.

위와 같은 방법으로 모든 가능한 노드를 전위탐색으로 검색하게 되면 모든 지불가능한 방법의 수를 구할 수 있음에 틀림없다.

따라서 백트래킹으로 알고리즘을 설계하게 되면 반드시 최적해를 찾을 수 있다. 하지만 실행시간이 상당히 오래 걸린다. 따라서 이 방법으로 그대로 프로그

램을 작성하면 제한 시간 내에 정답을 찾을 수 없는 현상이 발생할 수도 있다.

여러분들이 위의 탐색트리를 잘 분석해 보면 실행시간을 줄일 수 있는 방법을 발견할 수 있을 것이다. 실행시간을 줄이는 방법에 대해서는 문제로 남기도록 한다.

다음은 백트래킹 설계기법으로 설계한 알고리즘의 소스코드이다.

— ● <VC>

```
#include <stdio.h>
#define MAX 4

// 이 프로그램은 백트래킹으로 작성된 프로그램입니다.
// 돈의 수가 조금만 많아져도 실행시간이 기하급수적으로 커지게 됩니다.
// 따라서 풀이로는 부적합 하지만 백트래킹의 원리를 이해할 수 있습니다.
// 파일 입출력은 생략하고 배열로 초기값을 주었다.(실습의 용이성을 위해)
int coin[MAX] = {500, 100, 50, 10};
// 동전 종류를 바꿔가면서 테스트 가능, 단 백트래킹은 시간이 너무 오래 걸림
int min = 0x7fffffff;           // 정수 최대값 저장

void back(int rest, int count)          // 백트래킹의 본체 (되부름)
{
    if( rest == 0 ){                  // 거스름돈을 모두 지불 했을 경우
        if( min > count )           // 동전 수가 현재 최소 동전 보다 적으면 갱신
            min = count;             // 다시 다른 경우를 찾아서 백트래킹
        return;
    }
    for( int i = 0 ; i < MAX ; i++ )
        if( rest - coin[i] >= 0 )
            back( rest - coin[i], count+1); // 백트랙하는 부분임
}

int main(void)
{
    int money;
    printf("거스름 돈을 입력하시오. ");
    scanf("%d", &money);
    back(money, 0);           // 백트래킹 시작( 거스름돈, 동전 카운터);
    printf("%d개 입니다.\n", min);
    return 0;
}
```

— ● <VB>

Option Explicit

- ‘ Backtracking인 만큼 실행시간이 길다. 특히 500원 이상일 경우
- ‘ 생각보다 오래걸릴 것임

```
Dim coin As Variant
Dim min As Integer
Sub back(rest As Integer, count As Integer)
    Dim i As Integer, flag As Integer
    If rest = 0 Then
        If min > count Then
            min = count
        End If
        Exit Sub
    End If
    For i = 0 To 3
        If rest - coin(i) >= 0 Then
            back rest - coin(i), count + 1
        End If
    Next i
End Sub

Sub main()
    Dim money As Integer, i As Integer, cnt As Integer
    coin = Array(500, 100, 50, 10)
    min = 9999
    money = InputBox("거스름 돈을 입력하시오.")
    back money, 0
    Debug.Print CStr(min) + "개입니다."
End Sub
```



문제 V.3.2

위의 문제에서 백트래킹은 실행시간이 너무 오래 걸린다. 따라서 제한시간 내에 답을 구할 수 없는 경우가 있다. 위의 알고리즘을 분석하여 실행시간을 줄일 수 있는 방법을 한 가지만 제시하시오.

<풀이> 위의 탐색트리를 자세히 살펴보면 쉽게 시간을 줄일 수 있는 방법이 몇 가지가 있다. 그 중에 한 가지만 소개한다.

이 문제에서 구하고자 하는 것은 최소 동전 개수이다. 따라서 모든 탐색공간을 탐색할 필요가 없음을 쉽게 알 수 있다.

물론 처음 탐색할 때에는 지불 가능한 경로를 찾을 때 까지 탐색해야한다. 그 결과 처음으로 지불 가능한 경로를 찾았고, 그 때 깊이가 k 였다면, k 개의 동전으로 거스름돈을 지불할 수 있다는 의미가 된다. 계속 검색을 진행하다가 트리의 k 층 이상으로 탐색을 진행하려는 경우 여기서 바로 탐색을 멈추고 다시 백트랙 할 수 있다. 이유는 최소 동전의 수를 구하는 것이므로 k 층 보다 깊은 층에 지불할 수 있는 경우가 있다고 해도 이것은 정답이 될 수 없다. 따라서 탐색할 필요가 없음은 당연하다.

이 원리를 branch and bound 기법이라고 한다. 이 기법은 고급기법 중 하나로 백트래킹의 실행시간을 현실적으로 만들어주는 기법이다. 여기서는 깊이한계(bound)를 현재 까지 지불한 최소의 동전의 수로 보고 더 이상은 탐색하지 않는 방법으로 시간을 획기적으로 단축시킬 수 있다.

단, 이 경우에 그리디 기법이 도입이 되어야 효율이 높다는 점을 명심하기 바란다. 즉 액수가 큰 동전부터 탐색하는게 효율적임은 증명할 필요도 없다. 500, 100, 50, 10으로 검색 할 경우 상당히 bound 값이 적으므로 효율이 좋으나 10, 50, 100, 500으로 탐색할 경우 bound 값이 너무 커져서 크게 효과를 볼 수 없다.

소스는 다음 페이지에 소개한다. 단 한 줄만 추가해 주면 된다.

— ● <VC>

```
void back(int rest, int count)
{
    if( count >= min ) return;      // i] 부분을 추가하면 된다.
    if( rest == 0 ){
        if( min > count ) min = count;
        return;
    }
    for( int i = 0 ; i < 4 ; i++ )
        if( rest - coin[i] >= 0 ) back( rest - coin[i], count+1);
}
```

— ● <VB>

```
Sub back(rest As Integer, count As Integer)
    Dim i As Integer, flag As Integer
    If count >= min Then Exit Sub '이 줄의 추가로 속도가 빨라진다.
    If rest = 0 Then             '제한시간 내에 답을 구할 수 있음
        If min > count Then
            min = count
        End If
        Exit Sub
    End If
    For i = 0 To 3
        If rest - coin(i) >= 0 Then
            back rest - coin(i), count + 1
        End If
    Next i
End Sub
```

나. 백트래킹 설계 기법의 활용

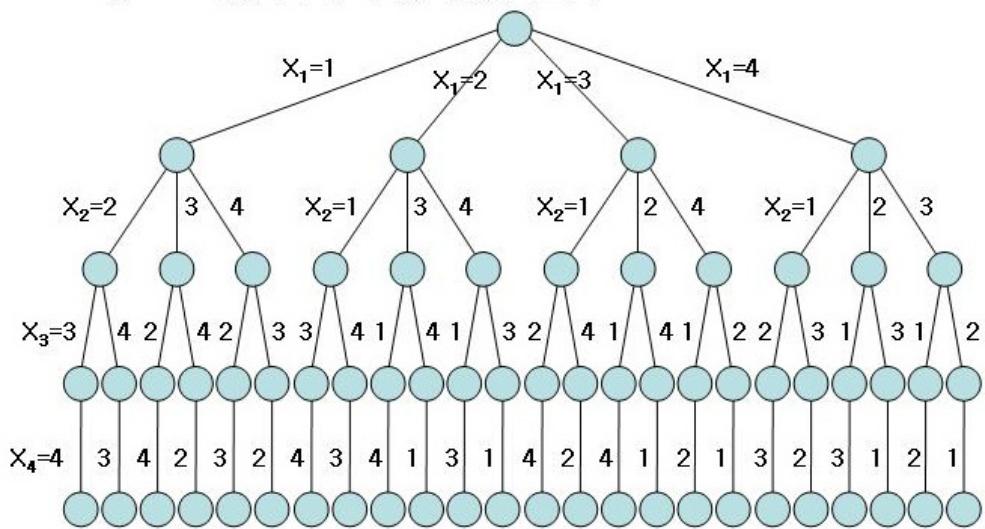
백트래킹으로 유명한 n-Queen문제가 있다. $n \times n$ 의 체스판에 한 줄에 하나씩 퀸을 배치하되, 각 퀸들이 서로 공격하지 못하게 배치하는 방법을 찾는 문제이다. 이 문제는 백트래킹으로 해결할 수 있는 유명한 문제이다. 이러한 문제의 특징은 퀸을 배치할 수 있는 방법 중 하나의 예를 출력하거나, 모든 배치 가능한 경우의 수를 묻는다. 가능한 방법을 구하는 경우에는 백트래킹으로 완벽하게 구할 수 있다.

그럼 n-Queen문제 중 n이 4일 경우의 해를 백트래킹으로 찾는 과정을 살펴보자.

	Q		
			Q
Q			
		Q	

위의 그림과 같이 배치를 하게 되면 서로 어떤 퀸도 공격받지 않게 된다. 즉 조건을 만족하는 해가 될 수 있는 배치이다. 기본적으로 위와 같은 해를 찾아내기 위해서는 우선 첫 번째 줄에 퀸을 왼쪽에 하나 배치한다. 그리고 2번째 줄에 공격받지 않는 위치 중 가장 왼쪽에 퀸을 배치하고 이러한 과정을 반복하여 4째 줄까지 반복하면 된다. 하지만 중간에 4칸 모두 퀸을 놓을 수 없는 경우가 발생한다. 바로 이때 퇴각이 일어난다. 한 단계 이전으로 돌아가서 다시 그 이전 퀸을 다른 가능한 칸에 놓고 다시 그 다음 칸에 퀸을 놓는 과정을 반복한다.

반복하는 과정을 트리로 만들면 다음 그림과 같다.



위의 탐색트리는 4×4 퀸의 모든 경우를 탐색하고 있다. 위의 트리를 전위순회하면서 답을 찾는 과정이 바로 백트래킹이다. 즉 백트래킹은 전위순회를 통해서 구현된다는 점도 반드시 이해하고 있어야 한다. 소스코드는 생략한다. 설계해 보면 백트래킹의 이해에 많은 도움이 될 것이다.



문제 V.3.3 좋은 수열(1997 한국정보올림피아드 중등부 기출)

숫자 1, 2, 3으로만 이루어지는 수열이 있다. 임의의 길이의 인접한 두 개의 부분 수열이 동일한 것이 있으면, 그 수열을 나쁜 수열이라고 부른다. 그렇지 않은 수열은 좋은 수열이다.

다음은 나쁜 수열의 예이다.

33

32121323

123123213

다음은 좋은 수열의 예이다.

2
32
32123
1232123

길이가 N인 좋은 수열들을 N자리의 정수로 보아 그 중 가장 작은 수를 나타내는 수열을 구하는 프로그램을 작성하라. 예를 들면, 1213121과 2123212는 모두 좋은 수열이지만 그 중에서 작은 수를 나타내는 수열은 1213121이다.

<입력형식> 입력파일은 input.txt이며 하나의 정수가 주어진다. 이 정수는 수열의 길이를 의미하며 80이하의 양의 정수 값이다.

<출력형식> 출력파일은 길이가 n인 좋은 수열 중 가장 작은 값을 출력하면 된다.

입력파일(INPUT.TXT) 예 : 7

출력파일(OUTPUT.TXT) 예 : 1213121

<풀이> 전형적인 백트래킹 문제이다. 여러 개의 좋은 수열이 존재하지만 가장 수가 작은 값을 출력하라고 했으므로 백트래킹 순서는 1, 2, 3의 순으로 이루어진다. 처음에 1을 기준으로 검색을 시작 2번째 항도 다시 1 즉 11이 된다. 이 수열은 좋은 수열이 아니므로 퇴각하여 12를 검사 여기까지 좋은 수열이므로 다시 3번째 항으로 진출 121과 같은 식으로 좋은 수열이 되지 않으면 퇴각하여 다시 다음 수를 검사하는 루틴으로 재귀 호출을 이용하면 된다.

다음 코드를 보고 분석해 보자.

— ● <VC>

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_N 161

int n, num[MAX_N]; // 정답을 저장할 배열 num선언
FILE *in = fopen("input.txt", "r"), *out = fopen("output.txt", "w");

int chk(int l) { // 현재까지 만들어진 수열이 좋은 수열인지 검사
    int i, j;
    for(i = 1 ; i <= l / 2 ; i++) {
        for(j = 1 ; j > l - i ; j--)
            if(num[j] != num[j - i])
                break;
        if(j == l - i) return 0; // 그 결과 좋은 수열이면 1 아니면 0을 리턴
    }
    return 1;
}

void back(int pt, int k) { // 백트래킹 법으로 좋은 수열을 1단 계식 검색
    int i;
```

```

if(pt == k + 1) {
    for(i = 1 ; i <= k ; i++)
        fprintf(out, "%d", num[i]);
    fprintf(out, "\n");
    exit(0);
}
for(i = 1 ; i < 4 ; i++) {
    num[pt] = i;           // 1~3까지의 수를 대입해가며 백트래킹
    if( chk(pt) )
        back(pt + 1, n);
}
}

void main() {
    fscanf(in, "%d", &n); // n값을 입력 받음
    back(1, n);          // 백트래킹 시작
}

```

— ● <VB>

```

Option Explicit
Const MAX_N As Integer = 161
Dim n As Integer, num(MAX_N) As Integer

Function chk(L as integer) As Integer   '현재까지의 수열이 좋은 수열인지 검사
    Dim i as Integer
    Dim j as Integer
    For i = 1 to L / 2
        For j = L to L - I
            If num(j) <> num(j-1) Then
                Exit For
            End If
        Next j
    End Function

```

```
If j = L - I Then chk = 0      '좋은 수열이면 1아니면 0을 반환
Next I
chk = 1
End Function

Sub back( pt As Integer, n as Integer )  '좋은 수열을 만들면서 백트래킹
    Dim I as integer
    If pt = n + 1 Then
        For I = 1 to n
            Print #2, num(i)
        Next I
    End If
    For I = 1 to 3
        num(pt) = I
        If chk(pt) = 1 Then      '숫자 1~3까지의 숫자를 차례로 퇴각
            back pt+1, n
        End if
    Next I
End Sub

Sub main()
    Dim n As Integer
    Open "input.txt" for Input As #1
    Open "output.txt" for output As #2
    Input #1, n
    back 1, n      '백트래킹 시작
End Sub
```

4. 분할정복(Divide And Conquer) 설계 기법

가. 분할정복 설계 기법이란

분할정복 설계 기법은 군대의 한 전략으로부터 유래된 알고리즘 설계기법이다. 19세기 아수스터리츠 전투에서 프랑스의 황제 나폴레옹이 한 전략을 사용했다. 오스트리아-러시아 연합군은 나폴레옹의 군대보다 15,000명 정도 많았다. 연합군은 프랑스군의 우측면에서 대규모 공격을 감행했다. 공격을 이미 예상한 나폴레옹은 연합군의 중앙으로 돌격하여 그들 병력을 둘로 갈라놓았다. 둘로 갈라진 소규모 병력은 개별적으로는 나폴레옹과는 상대가 되지 못했기 때문에 그들은 참패를 당하였다.

나폴레옹은 대규모 병력을 두 개의 소규모 병력으로 갈라놓고, 두 소규모 병력을 각각 점령함으로써 대규모의 연합군에 승리할 수 있었던 것이다. 분할정복법은 위의 사례와 같이 문제의 사례를 2개 이상의 더 작은 사례로 나눈다. 이 작은 사례는 주로 원래 문제에서 그대로 따온다. 작은 사례에 대해서 해답을 바로 구할 수 있으면 바로 구하고 그렇지 않으면 다시 문제를 2개 이상으로 분할한다. 따라서 분할정복법은 하향식 접근 방법이다.

앞 장에서 다룬 알고리즘들 중 분할정복법으로 만들어진 것은 퀵정렬과 이분탐색이 대표적이다. 이처럼 분할정복은 다양한 부분에서 이용되고 있는 설계기법이며, 동적계획법으로 넘어가는 중간 단계역할을 하므로 정보올림피아드를 지도하는 교사들이라면 꼭 익혀둘 필요가 있다.

이러한 분할정복법을 이용해서 거스름돈 문제를 해결해보자.

문제의 특성에 따라서 분할정복법은 퇴각검색법과 거의 동일한 알고리즘이 만들어지는 경우도 있다. 이번 문제의 경우에도 퇴각검색법과 비슷한 알고리즘이 만들어진다. 물론 설계하는 사람마다 다르게 설계할 수도 있겠지만 기본적으로 재귀호출을 이용하기 때문에 비슷하게 설계되는 경우가 많다.



문제 V.4.1 거스름돈의 분할정복 알고리즘 설계

거스름돈 문제를 분할정복으로 설계해보자.

일단 함수 DNC 를 정의한다. $DNC(n)$ 은 n 원의 거스름돈을 지불할 때의 가장 작은 동전의 수를 반환하는 함수로 정의한다. 따라서 이 문제에서 구하고자 하는 것은 $DNC(1300)$ 이다. 분할정복의 정의에 의해서 $DNC(1300)$ 을 2개의 사례로 반으로 분할해 보자.

$$DNC(1300) = DNC(650) + DNC(650)$$

위와 같이 반으로 분할할 수 있다. 하지만 함수의 정의에 의해서 위의 분할은 성립할 수 없음을 직감적으로 알 수 있다. 그 이유는 1300원을 거스름돈을 최소의 동전으로 주는 방법이 650원을 거스름돈으로 거슬러 줄 때의 최소의 동전 $\times 2$ 가 성립하지 않기 때문이다. 따라서 위와 같이 분할해서는 정답을 구할 수가 없다.

조금만 생각을 해보면 어떻게 분할을 해야 답을 구할 수 있을지 쉽게 알 수 있을 것이다. 방법은 거슬러 줄 수 있는 동전의 액수만큼과 나머지로 분할 하면 식이 성립함을 알 수 있다. 따라서 다음과 같이 분할 할 수 있다.

$$DNC(1300) \text{ 은 } DNC(500) + DNC(800), DNC(100) + DNC(1200),$$

$$DNC(50) + DNC(1250), DNC(10) + DNC(1290)$$

으로 분할할 수 있다. 위의 4개의 식중 $DNC(1300)$ 이 될 수 있는 것은 최소의 값이다. 따라서 다음과 같은 식이 만들어 진다.

$$DNC(1300) = \min\{ DNC(500) + DNC(800), DNC(100) + DNC(1200),$$

$$DNC(50) + DNC(1250), DNC(10) + DNC(1290) \}$$

Min(a, b, c, d)는 a, b, c, d중의 최소값을 반환하는 함수이다. 위 식을 바탕으로 프로그램을 만들면 다음과 같다.

<VC>

```
#include <stdio.h>
#define MAX 4

// 이 프로그램은 분할 정복법으로 작성된 것입니다.
// 기본 적으로 뱃트래킹처럼 되부름으로 이루어 집니다.
// 돈의 종류가 조금만 많아지면 실행시간이 엄청나게 커짐
// 분할 정복도 이 문제의 풀이로는 적합하지 않습니다.

int coin[MAX] = {500, 100, 50, 10}; // 동전의 종류임

int DNC(int rest)           // 분할정복
{
    int mid_value, min = 0x7fffffff, i;
    for (i = 0 ; i < MAX; i++)
        if( rest - coin[i] == 0 )
            return 1; // 정복가능한 크기라면 정복함.
    for (i = 0; i < MAX; i++)
    {
        if (rest >= coin[i]) // 정복 불능이면 분할 함
            mid_value = DNC(rest - coin[i]) + DNC(coin[i]);
        if (min > mid_value)
            min = mid_value; // 구한 값 중 최소값을 저장
    }
    return min;
}

int main(void)
{
    int money;
    printf("거스름 돈을 입력하시오. ");
    scanf("%d", &money);
    printf("%d가지 입니다.\n", DNC(money));
    return 0;
}
```

— ● <VB>

```
Option Explicit
'역시 DNC도 시간이 오래 걸린다. 500이상이라면 거의 불가능
Dim coin As Variant
Dim min As Integer
Function DNC(ByVal rest As Integer) As Integer
    Dim mid_value As Integer, min As Integer, i As Integer
    min = 9999: mid_value = 10000
    For i = 0 To 3
        If rest - coin(i) = 0 Then
            DNC = 1
            Exit Function
        End If
    Next i
    For i = 0 To 3
        If rest >= coin(i) Then
            mid_value = DNC(rest - coin(i)) + DNC(coin(i))
        End If
        If min > mid_value Then
            min = mid_value
        End If
    Next i
    DNC = min
End Function

Sub main()
    Dim money As Integer
    coin = Array(500, 100, 50, 10)
    min = 9999
    money = InputBox("거스름 돈을 입력하시오.")
    Debug.Print CStr(DNC(money)) + "개입니다."
End Sub
```



문제 V.4.2

위의 문제에서 분할정복법으로 작성된 알고리즘도 앞에서 다루었던 백트래킹과 마찬가지로 실행시간이 너무 오래 걸린다. 따라서 제한시간 내에 답을 구할 수 없는 경우가 있다. 위의 알고리즘을 분석하여 실행시간을 줄일 수 있는 방법을 제시하시오.

<풀이> 위의 거스름돈 문제를 분할정복법으로 작성하게 될 경우 실행시간이 상당히 오래 걸린다. 그 이유는 다음 식을 자세히 분석해 보면 알 수 있다. 위에서 사용했던 다음 식을 분석해 보자.

$$\begin{aligned} \text{DNC}(1300) &= \text{Min}\{ \text{DNC}(500) + \text{DNC}(800), \text{DNC}(100) + \text{DNC}(1200), \\ &\quad \text{DNC}(50) + \text{DNC}(1250), \text{DNC}(10) + \text{DNC}(1290) \} \end{aligned}$$

위의 식을 보면 $\text{DNC}(1300)$ 은 위의 4개의 계산을 해서 구한 최소값을 답으로 한다. 이 4개의 식은 다시 한 번에 정복 가능한 식과 ($\text{DNC}(500)$, $\text{DNC}(100)$, $\text{DNC}(50)$, $\text{DNC}(10)$) 다시 분할해야 하는 식들($\text{DNC}(800)$, $\text{DNC}(1200)$, $\text{DNC}(1250)$, $\text{DNC}(1290)$)로 분할 된다.

여기서 조금만 생각해보면 다시 분할되어야 하는 식 4개는 각각 다시 분할되어야 할 식 4개씩 즉 16개의 분할 해야 할 식들이 만들어지는데, 이렇게 만들어지다 보면 같은 식이 생기게 된다는 것이다. 즉 $\text{DNC}(1290)$ 을 분할하다 보면 언젠가는 $\text{DNC}(800)$, $\text{DNC}(1200)$, $\text{DNC}(1250)$ 과 같은 분할이 생기게 될 것이 이 분할들은 다시 분할이 되어야 한다. 하지만 이미 위의 값들은 앞에서 연산이 되었다. 따라서 다시 분할하지 않아도 답을 구할 수 있음에도 위 알고리즘은 계속 분할을 해서 계산의 중복이 많이 발생되는 단점을 가진다.

<풀이> 이러한 계산의 중복을 없앨 수 있다면 이 알고리즘은 제한된 시간내에 답을 구할 수 있는 알고리즘이 될 것이다.

그렇다면 계산의 중복을 어떻게 줄일 수 있을까?

방법은 간단하다. 다른 메모리 공간에 이미 한번 계산한 값을 저장해 두었다가 다시 그 계산을 할 경우에 분할하지 않고 이미 계산한 값을 그대로 이용하는 것이다.

일단 다음과 같은 배열을 선언한다.

D[MAX]

위 배열은 한번 계산한 값을 저장해 둘 배열이다. MAX지를 가능한 최대 값이 된다. 즉 D[100]에는 100원의 거스름돈을 지불할 수 있는 최소의 동전 수가 저장된다. 처음의 초기값은 모두 0이다. 그리고 DNC(800)이 계산될 때 D[800]을 검사하여 그 값이 0이라면 아직 정답을 알지 못한다는 의미이므로 일단 분할하여 답을 구하고, 구한 값을 다시 D[800]에 저장해 둔다.

다음에 다시 DNC(800)이 계산된다면 이 때는 이를 분할하지 않고 D[800]에 저장된 값을 이용할 수 있다. 이와 같은 방법을 이용하면 실행시간이 엄청나게 짧아진다는 것은 실험을 해보면 쉽게 알 수 있다.

위의 방법을 Memory Function기법이라고 한다. 계산의 중복을 없애기 위해서 한 번 계산된 값을 저장해 두었다가 다시 이용할 때 그 값을 이용하는 방법이다. 이 방법은 다음 단원에서 다루게 될 동적계획법의 일종이기도 하다.

소스파일을 분석해보면 쉽게 이해될 것이다. 소스를 보자.

<VC>

```

#include <stdio.h>
#define MAX 100
#define TMAX 10000

// 기본 분할 정복에서 계산 중복을 없앤 것입니다. 실행속도가 현실적입니다.
// 잘 분석해 보세요. 일명 가지치기 기법과 유사합니다.

int coin[MAX] = {10,50,60,100,500};
int Table[TMAX];           // 이 부분이 추가 됨 (계산 중복 방지)

int DNC(int rest)
{
    int mid_value, min = 0x7fffffff, i;
    for (i = 0 ; i < 5; i++)
        if( rest - coin[i] == 0 )
            return 1; // 정복 가능한 크기라면 정복함.
    if( Table[rest] == 0 ) // 이 부분도 추가( 아직 정복되지 않았다면 분할)
    {
        for (i = 0; i < 5; i++)
        {
            if (rest >= coin[i]) mid_value = DNC(rest - coin[i]) + DNC(coin[i]);
            if (min > mid_value) min = mid_value;
        }
        Table[rest] = min; // 이 부분도 추가됨 (정복 리스트 갱신)
    }
    return Table[rest]; // 값 반환
}

int main(void)
{
    int money;
    printf("거스름 돈을 입력하시오. ");
    scanf("%d", &money);
    printf("%d 가지 입니다.\n", DNC(money));
    // 분할정복 시작 시작( 거스름돈, 동전 카운터);
    return 0;
}

```

— ● <VB>

```

Option Explicit
‘처리속도가 아주 빨라짐, 제한 시간 내에 답을 구할 수 있음
Dim coin As Variant
Dim min As Integer, Table(10000) As Integer ‘새로운 배열 추가
Function DNC(ByVal rest As Integer) As Integer
    Dim mid_value As Integer, min As Integer, i As Integer
    min = 9999: mid_value = 10000
    For i = 0 To 3
        If rest - coin(i) = 0 Then
            DNC = 1
            Exit Function
        End If
    Next i
    If Table(rest) = 0 Then      ‘새로 추가된 줄
        For i = 0 To 3
            If rest >= coin(i) Then
                mid_value = DNC(rest - coin(i)) + DNC(coin(i))
            End If
            If min > mid_value Then
                min = mid_value
            End If
            Table(rest) = min      ‘처리 방법을 바꿈
        Next i
    End If
    DNC = Table(rest)           ‘○] 부분도 바꿈
End Function

Sub main()
    Dim money As Integer
    coin = Array(500, 100, 50, 10)
    min = 9999
    money = InputBox("거스름 돈을 입력하시오.")
    Debug.Print CStr(DNC(money)) + "개입니다."
End Sub

```

나. 분할정복 설계 기법의 활용

이 절에서는 분할정복법으로 해결할 수 있는 다른 문제들을 다루어 본다. 이 문제들을 통해서 분할정복법을 익힐 수 있도록 하자.



문제 V.4.3

$1 + 2 + 3 + \dots + n$ 을 구해서 화면에 출력하는 프로그램을 분할정복법을 이용해서 만들어 보시오.

<풀이> 분할정복법은 일단 문제를 작은 사례로 분할을 해야 한다.

재귀 함수의 이름을 $\text{sum}(n)$ 이라고 두자.

여기서 $\text{sum}(n)$ 의 역할은 $1 + 2 + 3 + \dots + n$ 을 반환하는 함수이다. 이 함수에서 $\text{sum}(n)$ 을 구하는 과정을 $\text{sum}(n-1) + n$ 로 분할한다. 여기서 n 은 이미 구한 값이므로 $\text{sum}(n-1)$ 을 구해야 하는데 이는 지금 구할 수 없으므로 다시 $\text{sum}(n-2) + n-1$ 로 분할한다. 이러한 과정으로 계속 분할하다 보면 결국 모든 항의 값을 구할 수 있다.

재귀 호출로 사례를 분할하여 $\text{sum}(n)$ 을 구하는 과정으로 작성하면 된다. 다음 소스 코드를 살펴보자.

— ● <VC>

```
#include <stdio.h>

int sum(int a){
    if(a==1) return 1;          // 우리가 풀 수 있을 만큼 분할되었을 때를 의미
    else return sum(a-1) + a;  // 재귀 호출로 분할 정복을 구현
}

void main(void){
    int n;
    printf("n을 입력하시오. : ");
    scanf("%d", &n);
    printf("%d\n", sum(n));   //sum(n)을 호출
}
```

— ● <VB>

Option Explicit

```
Function sum(a As Integer) As Integer
    If a=1 Then
        sum = 1      ' 우리가 풀 수 있을 만큼 분할되었을 때를 의미
    else
        sum = sum(a-1) + a  ' 재귀 호출로 분할정복을 구현
    End If
End Function

Sub main()
    Dim n As Integer
    n = InputBox("n값을 입력하시오.")
    MsgBox Cstr(sum(n))  ' sum(n)을 호출
End Sub
```



문제 V.4.4 색종이 만들기(2001 한국정보올림피아드 중등부 기출)

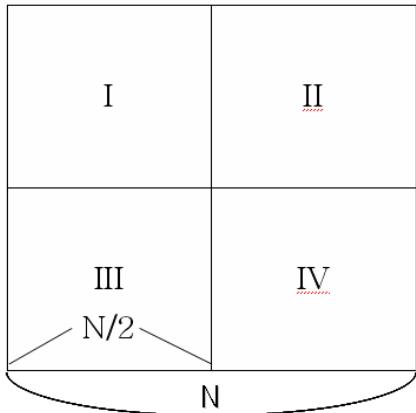
아래 그림과 같이 여러 개의 정사각형 칸들로 이루어진 정사각형 모양의 종이가 주어져 있고, 각 정사각형 칸들은 하얀색으로 칠해져 있거나 파란색으로 칠해져 있다. 주어진 종이를 일정한 규칙에 따라 잘라서 다양한 크기를 가진 정사각형 모양의 하얀색 또는 파란색 색종이를 만들려고 한다.

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

전체 종이의 크기가 $N \times N$ ($N = 2^k$, k 는 1이상 7이하의 자연수) 이라면 종이를 자르는 규칙은 다음과 같다.

전체 종이가 모두 같은 색으로 칠해져 있지 않으면 가로와 세로로 중간 부분을 잘라서 다음 페이지의 첫 번째 그림 I, II, III, IV와 같이 똑같은 크기의 네 개의 $\frac{N}{2} \times \frac{N}{2}$ 색종이로 나눈다. 나누어진 종이 I, II, III, IV 각각에 대해서도 앞에서와 마찬가지로 모두 같은 색으로 칠해져 있지 않으면 같은 방법으로 똑같은 크기의 네 개의 색종이로 나눈다. 이와 같은 과정을 잘라진 종이가 모두 하얀색 또는 모두 파란색으로 칠해져 있거나, 하나의 정사각형 칸이 되어 더 이상 자를 수 없을 때까지 반복한다.

위와 같은 규칙에 따라 잘랐을 때 다음 페이지의 그림들과 같이 분할되면서 최종적으로 만들어진 다양한 크기의 9장의 하얀색 색종이와 7장의 파란색 색종이를 보여주고 있다.



〈분할 요령〉

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

〈1차 분할〉

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

〈2차 분할〉

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

〈3차 분할〉

입력으로 주어진 종이의 한 변의 길이 N 과 각 정사각형 칸의 색 (하얀색 또는 파란색)이 주어질 때 잘라진 하얀색 색종이와 파란색 색종이의 개수를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 COLOR.EXE로 하고, 프로그램의 실행시간은 5초를 초과할 수 없다. 부분 점수는 없다.

<입력형식> 입력 파일명은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에는 전체 종이의 한 변의 길이 N이 주어져 있다. N은 2, 4, 8, 16, 32, 64, 128 중 하나이다. 색종이의 각 가로줄의 정사각형 칸들의 색이 윗줄부터 차례로 입력 파일의 둘째 줄부터 마지막 줄까지 주어진다. 하얀색으로 칠해진 칸은 0, 파란색으로 칠해진 칸은 1로 주어지며 각 숫자 사이에는 빈칸이 하나씩 있다.

<출력형식> 출력 파일명은 OUTPUT.TXT로 한다.

첫째 줄에는 잘라진 하얀색 색종이의 개수를 출력하고 둘째 줄에는 파란색 색종이의 개수를 출력한다.

입력파일(INPUT.TXT) 예 : 8
 1 1 0 0 0 0 1 1
 1 1 0 0 0 0 1 1
 0 0 0 0 1 1 0 0
 0 0 0 0 1 1 0 0
 1 0 0 0 1 1 1 1
 0 1 0 0 1 1 1 1
 0 0 1 1 1 1 1 1
 0 0 1 1 1 1 1 1

출력파일(OUTPUT.TXT) 예 : 9
 7

<풀이> 이 색종이 만들기는 대표적인 분할정복법 문제이다. 이 문제를 해결하기 위해서 일단 전체가 한 색인지 검사를 하고 한 색이 아니면 4부분으로 분할하고, 각각 4부분을 전체적으로 재귀 호출을 이용하여 해결할 수 있는 문제이다.

— ● <VC>

```
#include <stdio.h>

int white, blue; // 각 각의 색깔의 종이 조각의 수를 저장
int paper[128][128];

void DNC(int a, int b, int n){
    int i, j, color, flag=0;
    color = paper[a][b]; // 첫 색을 color에 저장
    for(i=a;i<a+n;i++)
        for(j=b;j<b+n;j++)
            if(paper[i][j]==color) flag++; // 모든 부분의 색이 같은지 검사
    if(flag==n*n){
        if(color) blue++; // 같으면 색을 체크하고 리턴
        else white++;
        return;
    }
    DNC(a, b, n/2); //그렇지 않다면 4부분으로 분할하여 재귀 호출
    DNC(a+n/2, b, n/2);
    DNC(a, b+n/2, n/2);
    DNC(a+n/2, b+n/2, n/2);
}
void main(void){
    int n, i, j;
    FILE *in=fopen("input.txt","r"), *out=fopen("output.txt","w");
    fscanf(in, "%d", &n); // 종이 크기를 입력받음.
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            fscanf(in,"%d", &paper[i][j]); // 데이터 입력
    DNC(0, 0, n); //분할 정복 시작
    fprintf(out, "%d\n%d\n", white, blue); //결과 출력
}
```

<VB>

```
Option Explicit
```

```

Dim white As Integer, blue As Integer
Dim paper(128, 128) As Integer
'a는 시작행 첨자, b는 시작열 첨자, n은 현재 종이 크기
Sub DNC(a As Integer, b As Integer, n As Integer)
    Dim i As Integer, j As Integer, color As Integer, flag As Integer
    color = paper(a, b)      '첫 번째 종이 색깔 저장
    For i = a To a + n - 1      '전체 색깔이 동일한지 검사
        For j = b To b + n - 1
            If paper(i, j) = color Then flag = flag + 1
        Next j, i
        If flag = n * n Then      '전체 색깔이 동일하면
            If color = 1 Then
                blue = blue + 1   '현재 색깔 종이 하나 추가
            Else
                white = white + 1
            End If
            Exit Sub
        End If
        DNC a, b, n / 2          '그렇지 않으면 4분할 후 재귀 호출
        DNC a + n / 2, b, n / 2
        DNC a, b + n / 2, n / 2
        DNC a + n / 2, b + n / 2, n / 2
    End Sub

Sub main()
    Dim n As Integer, i As Integer, j As Integer
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2
    Input #1, n
    For i = 0 To n - 1
        For j = 0 To n - 1
            Input #1, paper(i, j)  '파일로 부터 정보 입력
        Next j, i
        DNC 0, 0, n              '분할 정복 시작
        Print #2, CStr(white)     '결과 출력
        Print #2, CStr(blue)
    End Sub

```

5. 동적계획법(Dynamic Programming)

가. 동적계획법이란

동적계획법은 문제의 사례를 더 작은 사례로 분할한다는 점에서 분할정복법과 비슷하다. 그러나 이 방법은 작은 사례를 먼저 해결한 후, 그 결과를 저장하고 나중에 그 결과가 필요할 때마다 다시 계산하는 대신 저장한 결과를 이용한다. 즉 분할정복법은 하향식 방법인데 반해 동적계획법은 상향식 방법인 것이다.

동적계획법은 해답을 구축하는데 필요한 배열을 사용함으로써 이전에 구했던 답을 저장해서 사용한다. 가장 작은 문제의 해를 구해서 점점 큰 해를 구하므로 동적계획법은 상향식 방법이라고 할 수 있다. 동적계획법의 개발 절차는 다음과 같다.

- ① 문제의 사례에 대해서 해답을 주는 점화식을 구한다.
- ② 점화식을 사용해서 주어진 문제를 차례로 해결한다.

〈동적계획법의 절차〉

동적계획법은 이전의 답을 저장해서 이용하기 때문에 효율이 좋다. 정답을 저장할 배열을 사용하므로 해서 부가적으로 메모리를 이용한다는 단점도 있다.

동적계획법은 설계하기가 상당히 어려운 알고리즘 설계 방법이다. 이는 수많은 경험을 요구하는 알고리즘 설계방법이기도 하다. 그러므로 거의 모든 경시대회에 단골 메뉴로 나온다는 것을 명심하자. 어려운 방법이지만 동적계획법으로 알고리즘을 설계하는 방법을 문제를 통해서 익힐 수 있도록 많은 연습을 하여야 한다.

앞에 주어졌던 거스름돈 문제를 동적계획법으로 만들어 보자.

일단 점화식을 만들어야 한다. 하지만 점화식은 이미 분할정복법의 활용에서 다룬 Memory Function에서 충분한 힌트가 되었을 것이다.

일단 점화식을 다음과 같이 정의한다.

$DP(n)$ 을 n 원을 지불할 때의 최소 동전 수라고 정의한다.

초기값으로

$DP(10), DP(50), DP(100), DP(500)$ 의 값은 모두 1이다.

그 이외의 $DP(k)$ (단 $10 \leq k \leq n$)의 경우에는 다음과 같은 식이 성립한다.

$$\begin{aligned} DP(k) = \min\{ & DP(k-10) + DP(10), DP(k-50) + DP(50), \\ & DP(k-100) + DP(100), DP(k-500) + DP(500) \} \end{aligned}$$

여기서 $DP(10), DP(50), DP(100), DP(500)$ 은 실제 모두 1이므로 위 식을 종합하면 다음과 같이 고칠 수 있다.

$$DP(10) = 1, DP(50) = 1, DP(100) = 1, DP(500) = 1$$

$$DP(k) = \min\{ DP(k-10) + 1, DP(k-50) + 1, DP(k-100) + 1, DP(k-500) + 1 \}$$

(위의 식에서 $DP(p)$ 에서 $p \geq 0$ 일 때만 계산한다.)

$DP(n)$ 을 출력

위의 점화식을 이용해서 k 를 10부터 n 까지 반복문으로 작성하면 $O(n)$ 만에 우리가 원하는 답을 구할 수 있는 아주 속도가 빠른 알고리즘이 된다. 결국 올림피아드에서 가장 많이 등장하는 알고리즘 설계법이 바로 동적계획법이고, 이 동적계획법의 점화식이 쉽게 만들어지지 않으므로, 분할정복법을 활용하면 점화식을 유도하는데 도움이 되는 경우도 있으므로 익혀두도록 하자.

위 점화식을 바탕으로 만들어진 소스는 다음과 같다. 소스의 길이를 보면 정말 짧으면서도 실행 효율이 좋다는 것을 알 수 있을 것이다.

— ● <VC>

```
#include <stdio.h>
#define MAX 100
#define TMAX 10000

// 동적 계획법입니다.
// 실제 올림피아드에 가장 많이 출제되는 방식입니다.
// 꼭 익혀 두시기 바랍니다. 특히 점화식 유도 부분.

int coin[MAX] = {10,50,100,500};
int DP[TMAX];           // 동적계획 테이블

int main(void)
{
    int money, min;
    printf("거스름 돈을 입력하시오. ");
    scanf("%d", &money);
    for( int i = 10 ; i <= money ; i+= 10 )
    {
        for( int j = 0, min = 0xffffffff ; j < 4 ; j++ ) // 4는 동전의 종류
            if( coin[j] <= i && min > DP[i - coin[j]] )
                min = DP[i - coin[j]] + 1;
        DP[i] = min;
    }
    printf("%d가지 입니다.\n", DP[money]);
    return 0;
}
```

<VB>

Option Explicit

'화식 유도부분이 중요하므로 잘 익혀두시기 바랍니다.

```

Dim coin As Variant
Dim min As Integer, DP(10000) As Integer

Sub main()
    Dim money As Integer, i As Integer, j As Integer
    coin = Array(500, 100, 50, 10)
    money = InputBox("거스름 돈을 입력하시오.")
    For i = 10 To money Step 10
        min = 9999
        For j = 0 To 3
            If coin(j) <= i Then
                If min > DP(i - coin(j)) Then
                    min = DP(i - coin(j)) + 1
                End If
            End If
        Next j
        DP(i) = min
    Next i
    Debug.Print CStr(DP(money)) + "개입니다."
End Sub

```

나. 동적계획법의 활용

동적계획법은 활용분야가 워낙 넓기 때문에 쉽게 설명하기 어렵다. 일단 간단한 동적계획법 문제를 풀어보고 원리를 익힐 수 있도록 하자. 마지막 단원의 기출문제 풀이에서도 동적계획법으로 푼 문제들이 있으므로 참고하기 바란다.



문제 V.5.1 광석 수집 문제

달에서 광석을 수집하는 수집차량이 있다. 하지만 달에 불시착하면서 이 수집차량이 고장이 났다. 그래서 이 차량은 아래와, 오른쪽으로만 이동이 가능하다. 한 지역에서 아래와 오른쪽으로 이동해 가면서 광석을 모아야 하는데 아래 그림에서 가장 많이 얻을 수 있는 광석은 몇 개인가? 출발 지점은 가장 왼쪽 위가 되고, 도착점은 가장 오른쪽 아래 지점이 된다.

		◆		
	◆		◆	
◆	◆		◆	
◆	◆		◆	◆

(◆ : 광석)

<입력형식> 입력파일은 input.txt이다. 이 파일의 첫째 줄에는 행이나 열의 크기를 나타내는 100이하의 양의 정수가 표시된다. 둘째 줄부터 0과 1이 표시되는데 0은 일반 땅을 표시하고 1은 광석이 있는 땅을 표시한다.

<출력형식> 출력파일은 output.txt이다. 출력파일에 주어진 맵에서 얻을 수 있는 가장 많은 광석을 정수하나로 표현하면 된다.

입력파일(INPUT.TXT) 예 : 5
0 0 1 0 0
0 1 0 1 0
0 0 0 0 0
1 1 0 1 0
1 1 0 1 1

출력파일(OUTPUT.TXT) 예 : 5

<풀이> 우선 동적계획법에서 제일 먼저 생각해야하는 것이 주어진 문제를 분할하여 분할된 내용들 사이의 점화식을 만들어야 한다. 위의 문제는 일반적으로 모든 가능한 답을 탐색해서는 원하는 결과를 얻기 힘들고, 시간이 많이 걸린다. 주어진 문제에 n 이 최대 100이므로 $O(n^2)$ 의 알고리즘으로 충분히 해결이 가능하다. 이제 점화식을 만들어 보자.

$T(n, n)$ 의 배열을 만들고 이 배열은 정답을 저장하기 위해 사용한다.

$T(i, j)$ 가 의미하는 것은 광석수집차량이 출발점에서 i, j 까지 왔을 때의 최대 광석 량을 저장할 것이다.

$T(1, 1)$ 은 주어진 맵의 (1, 1)의 값이 된다. 이를 $map(1, 1)$ 이라고 하겠다. 왜냐면 시작 지점이 골인 지점이라고 생각하면 최대 광석 량은 시작점의 광석 량이 되기 때문이다. 이제 점화관계를 만들어 나가자.

$map(i, j)$ 로 광석차가 오기 위해서는 반드시 $map(i-1, j)$ 또는 $map(i, j-1)$ 두 곳 중 한 곳은 반드시 지나야 한다. (이동 가능 경로가 오른쪽과 아래 이므로) 여기서 나오는 점화식은

$$T(i, j) = \max\{ T(i-1, j), T(i, j-1) \} + map(i, j)$$

가 된다. 이 식을 이용해서 1, 1부터 n, n 까지 T 를 채우면서 마지막에 $T(n, n)$ 값을 출력하면 된다. 실제 풀이에서는 T 라는 배열을 따로 만들지 않고 map 배열을 그대로 개신하면서 사용했다. 왜냐하면 이 문제에서는 map 배열이 나중에 따로 사용되지 않으므로, 원본 데이터를 보존할 필요가 없기 때문이다.

————— ● <VC>

```
#include<stdio.h>
int map[101][101];
void main(void){
    int i, j, n;
    FILE *in=fopen("input.txt","r"), *out=fopen("output.txt","w");
    fscanf(in,"%d",&n);
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            fscanf(in,"%d",&map[i][j]);
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            map[i][j] += (map[i][j-1]>map[i-1][j])? map[i][j-1], map[i-1][j];
    fprintf(out,"%d\n", map[n-1][n-1]);
}
```

————— ● <VB>

```
Option Explicit
Dim map(100, 100) As Long, i As Integer, j As Integer, n As Integer
Sub Main()
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2
    Input #1, n
    For i = 1 To n
        For j = 1 To n
            Input #1, map(i, j)
        Next j, i
        For i = 1 To n
            For j = 1 To n
                map(i, j)=IIf(map(i-1, j)<map(i, j-1), map(i, j-1), map(i-1, j))+map(i, j)
            Next j, i
            Print #2, CStr(map(n, n))
        End Sub
```

6. 교원컴퓨터프로그래밍경진대회 본선 기출 문제 풀이

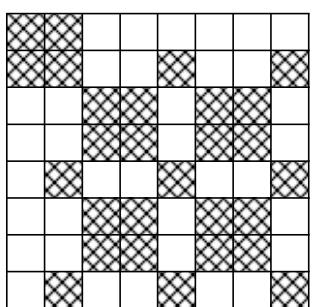
이 장에서는 제1,2,3회 교원컴퓨터프로그래밍 경진대회의 기출문제 중 주로 4번부터 5번까지의 문제들을 살펴보고 해결책을 제시하며 분석하고자 한다. 각 문제의 난이도를 ★의 수를 기준으로 표현해 보았다. ☆는 반개의 별을 의미한다. 기출 문제들을 직접 풀어보고, 이해되지 않는 부분은 해당 풀이를 참조할 수 있도록 하자.

가. 제1회 교원컴퓨터프로그래밍 경진대회 문제 풀이

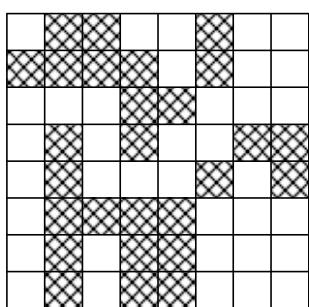


문제 V.6.1 (제1회 4번, 난이도 : ★★)

흑과 백의 셀들로 구성된 $n \times n$ ($n \leq 20$)의 정사각형에서 백색의 셀들로만 연결된 영역의 개수와 그 영역 안에 있는 백색의 셀 개수를 계산하는 프로그램을 작성하시오. 예를 들면 그림 a에는 10개의 백색 영역이 있고 그 중 2개는 10개의 백색 셀들로 구성되어 있고 나머지 8개는 2개의 셀들로 이루어져 있다. 입력 자료는 정사각형의 둘레를 ‘b’값으로 추가한 $(n+2) \times (n+2)$ 의 배열로 주어진다. 그림 b의 정사각형은 그림 c와 같은 $(8+2) \times (8+2)$ 의 배열로 표현된다. 실행파일은 PR4.EXE라 한다.



(a)



(b)

b	b	b	b	b	b	b	b	b	b
b	w	b	b	w	w	b	w	w	b
b	b	b	b	w	b	w	w	w	b
b	w	w	w	b	b	w	w	w	b
b	w	b	w	b	w	w	b	b	b
b	w	b	w	w	w	b	w	b	b
b	w	b	b	b	w	w	w	b	b
b	w	b	w	b	b	w	w	w	b
b	w	b	w	b	b	w	w	w	b
b	b	b	b	b	b	b	b	b	b

(c)

<입력형식> 입력파일 INPUT4.TXT의 첫 줄(라인)에는 n의 값이, 두 번째 줄부터 (n+3)번째 줄까지 데이터가 존재한다. 두 번째 줄부터는 각 줄에 (n+2)개의 문자(b 또는 w)가 입력된다.

<출력형식> 출력파일 OUTPUT4.TXT의 첫 줄에는 영역의 개수를 출력하고, 두 번째 줄에는 각 영역의 넓이(셀의 개수)를 출력하는데, 출력순서는 영역의 넓이 순서로 출력한다. 즉 좁은 영역을 먼저 출력하고 넓은 영역의 넓이를 나중에 출력한다. 각 영역의 넓이 사이에는 한 칸 이상의 빈칸을 두고, 필요하면 여러 줄을 사용하여 출력한다.

<그림 b인 경우>

입력파일(INPUT4.TXT) 예 :

8
b b b b b b b b
b wb b wwb wwb
b b b b wb wwb
b wwwb b wwwb
b wb wb wwb b b
b wb wwwb wb b
b wb b b wwwb
b wb wb b wwwb
b wb wb b wwwb
b b b b b b b b

출력파일(OUTPUT4.TXT) 예 :

5
1 2 3 10 21

<풀이> 이 문제는 대표적인 범람 채우기 문제이다. 여기서는 깊이우선탐색으로 범람 채우기를 구현하고 있다. 우선 각 칸들을 그래프의 정점이라고 생각을 하고, 각 정점들은 상·하, 좌·우 4 방향 중 연결된 부분이 w라면 이동이 가능하다. 이렇게 보면 여러 개의 그래프로 볼 수 있다. 이 문제를 바꾸어 말한다면 주어진 그래프에서 독립된 각 그래프의 수를 구하고 각 그래프가 가지는 정점의 수를 오름차순으로 출력하라는 문제로 바꿀 수 있다.

이렇게 바꿔서 생각하고 나면 해결 방법은 간단하다. 맨 처음 정점에서 출발해서 깊이우선탐색으로 탐색을 하면서 정점을 수를 센다. 이 상태가 그래프 1개를 찾은 것이다. 여기서 그래프 총 개수를 1증가시키고, 다음 방문 안한 정점을 또 찾아서 다시 깊이우선탐색을 실시, 모든 정점을 방문했으면 다시 한 개의 그래프를 찾은 결과가 되므로 그래프 수를 1만큼 증가시킨다. 이 작업을 방문 안한 정점이 없을 때까지 반복한다. 알고리즘이 완료되면 총 그래프의 수가 정해질 것이고, 각 그래프 별로 정점의 수를 저장하게 될 것이다. 여기서 각 그래프별 정점의 수를 오름차순으로 정렬해서 출력하면 이 문제는 해결이 된다. 각 언어별 풀이를 보면서 이해해 보자.

— ● <VC>

```
#include <stdio.h>
#include <stdlib.h>

int map[22][22], cnt = 2, result[1000]; // 입력값을 저장할 배열

int Compare( const void *a, const void *b ){
    return *(int *)a - *(int *)b;
}

// 쿼소트를 사용하기 위한 Compare함수
void flood_fill(int i, int j){ // 이 프로그램의 중요 알고리즘 Flood_Fill
    map[i][j] = cnt;
    if(map[i][j+1]==0) flood_fill(i,j+1);
    if(map[i+1][j]==0) flood_fill(i+1,j);
}
```

```

        if(map[i][j-1]==0) flood_fill(i,j-1);
        if(map[i-1][j]==0) flood_fill(i-1,j);
    } // 4방 검색 후 이동가면하면 DFS하는 함수임(상세한 분석 필요!!!)

void main(void){
    FILE *in=fopen("input4.txt", "r"), *out=fopen("output4.txt", "w");
    int i, j, n;
    char temp[30];
    fscanf(in,"%d",&n);
    for(i=0;i<n+2;i++){
        fscanf(in,"%s", temp);
        for(j=0;j<n+2;j++)
            if(temp[j]=='b') map[i][j] = 1;
            else map[i][j] = 0;
    } // 파일서 입력받아 b면 1 w면 0으로 대치
    for(i=0;i<n+2;i++)
        for(j=0;j<n+2;j++)
            if(map[i][j]==0){
                flood_fill(i,j); //flood_fill 알고리즘 구현
                cnt++; //그래프 개수 증가
            }
    for(i=0;i<n+2;i++)
        for(j=0;j<n+2;j++) // 독립 그래프 블
            if(map[i][j]>1) result[map[i][j]-2]++;
    qsort(result, cnt-2, sizeof(int), Compare); // 오름차순정렬(퀵정렬)
    fprintf(out,"%d\n", cnt-2);
    for(i=0;i<cnt-2;i++) // 결과 출력
        fprintf(out,"%d", result[i]);
}

```

— ● <VB>

```

Option Explicit
Dim i As Long, j As Long, cell(22, 22) As String, n As Long, msg As String
Dim count As Long, result(100) As Long

Sub main()
    input_data
    process
    output_data
End Sub

Sub input_data()
    Open App.path + "\input4.txt" For Input As #1
    Input #1, n
    For i = 1 To n + 2      '파일에서 cell배열로 읽기
        For j = 1 To n + 1
            cell(i, j) = Input(1, #1)
        Next j
        Line Input #1, cell(i, j)
    Next i
    Close #1
End Sub

Sub process()
    For i = 1 To n + 2
        For j = 1 To n + 2
            If cell(i, j) = "w" Then      '각 데이터를 W로 둑기
                count = count + 1
                Flood_Fill i, j          'DFS로 범람 채우기구현
            End If
        Next j
    Next I
    For i = 1 To n + 2
        For j = 1 To n + 2
            result(Val(cell(i, j))) = result(Val(cell(i, j))) + 1
        Next j
    Next i
    sort
End Sub

Sub output_data()
    Open App.path + "\output4.txt" For Output As #1
    Print #1, CStr(count)
    For i = 1 To count
        Print #1, CStr(result(i)) + " "; '완성된 결과
    Next i
End Sub

```

```

Next i
Close #1
End Sub

Sub Flood_Fill(i As Long, j As Long) '넓은 채우기 순서는 우 하 좌 상
    cell(i, j) = CStr(count)           ' DFS방법으로 진행
    If cell(i, j + 1) = "w" Then Flood_Fill i, j + 1
    If cell(i + 1, j) = "w" Then Flood_Fill i + 1, j
    If cell(i, j - 1) = "w" Then Flood_Fill i, j - 1
    If cell(i - 1, j) = "w" Then Flood_Fill i - 1, j
End Sub

Sub sort()                      '결과를 오름차순으로 선택정렬
    Dim temp As Long
    For i = 1 To count - 1
        For j = i + 1 To count
            If result(i) > result(j) Then
                temp = result(i) : result(i) = result(j) : result(j) = temp
            End If
        Next j, i
    End Sub

```



문제 V.6.2 (제1회 5번, 난이도 : ★★★★☆)

경상남도 K시는 시민들에게 보다 나은 교통 서비스를 제공하기 위하여 K시의 주요 지점별 소요시간을 측정하였고, 어떤 두 지점간의 최단 경로를 제공하는 서비스를 구축하려고 한다. 문제는 경상남도 K시의 인접한 주요 지점별 교통 소요시간 정보와 두 지점을 입력으로 받아 두 지점간의 최단경로를 찾아내고 소요시간을 출력하는 프로그램을 작성하는 것이다. K시의 주요 지점별 교통소요시간 정보는 다음 그림과 같이 주요 지점들은 A부터 Z까지의 문자들로 표시되며, A에서 B까지의 소요 시간은 A에서 B까지의 간선에 레이블로 표시할 수 있다. 이런 그래프 정보를 표현하는 방법에는 여러 가지가 있지만, 여기서는 표와 같은 형식으로써 표현된다(입력파일 INPUT5.TXT의 두 번째 줄 참조). 표의 첫 번째 열은 시작 지점, 두 번째 열은 끝 지점, 그리고 세 번째 열은 시작 지점에서 끝 지점까지의 소요 시간을 나타낸다. 입력으로 이런 교통소요시간에 대한 정보와 최단 경로를 알고 싶은 지점들을 입력으로 받아, 두 지점 사이의 최단 경로를 출력하는 프로그램을 작성하시오(단, 실행파일은 PR5. EXE로 하고, 프로그램의 실행시간은 3초를 넘을 수 없다).

<입력형식> 입력파일 INPUT5.TXT의 첫째 줄에는 주요지점의 개수(m)와 소요 시간정보의 개수(n)가 주어진다. 둘째 줄부터 n개의 줄에는 주요 지점간의 교통 정보가 주어진다. 다음 줄에는 알고 싶은 최단 경로의 총 수(k)가 주어지며, 그 다음 k줄에는 시작 지점과 끝 지점을 나타내는 영문자가 주어진다.(즉, 주요 지점은 대문자 A부터 알파벳 순서로 m개까지의 문자로 표시되며, $1 \leq m \leq 26$ 이다.)

<출력형식> 출력파일은 OUTPUT5.TXT라 한다. k개의 줄에 총소요시간, 시작 지점부터 끝 지점까지의 모든 경로를 문자들로 나타낸다. (최단경로가 여러 개 존재할 때는 하나만 찾아내어도 정답으로 인정한다. 또한, 경로가 존재하지 않을 때에는 “경로가 존재하지 않음”이라고 출력한다.)

입력화일(INPUT5.TXT) 예 :

```
7 15
B A 7
B C 2
C B 10
C A 4
A G 10
A D 4
D C 8
C E 6
E D 3
E F 7
D F 5
F D 1
D G 2
F G 6
G F 11
5
A D
A G
B G
G C
G G
```

출력화일(OUTPUT5.TXT) 예 :

```
4 AD
6 ADG
12 BCADG
20 GFDC
0 G
```

<풀이> 이 문제는 전형적인 그래프 문제이다. 그래프 중에서 가중치그래프의 최단경로를 구하는 문제이다. 만약 이 문제의 그래프에 가중치가 없다면 단순히 너비우선탐색으로도 해결할 수 있으나, 가중치가 존재한다는 것은 정점을 지나는 횟수(가장 적은 정점을 거치고 목적지로 가는 행위)로는 최단 경로를 보장받을 수 없다. 그래서 이 문제는 최단경로 알고리즘인 다익스트라의 최단경로 알고리즘 또는 플로이드(Floyd)알고리즘으로 구할 수 있다. 다익스트라의 알고리즘은 $O(n^2)$ 이고 플로이드알고리즘은 $O(n^3)$ 이므로 여기에서는 다익스트라의 방법으로 풀이하였다. 하지만 플로이드알고리즘도 간단하게 최단경로를 찾을 수 있는 알고리즘이므로 간단하게 살펴보자.



플로이드의 최단경로 (동적계획법)

이 알고리즘은 모든 정점에서 모든 정점으로의 최단경로를 구할 수 있는 알고리즘이다. 인접행렬로 그래프를 표현한 다음에 아래의 알고리즘을 실행하게 되면 인접행렬 값이 각 정점에서 정점으로의 최단경로가 저장이 되게 된다. 여기서 이용한 인접행렬은 $map[n][n](map(n,n))$ 이 된다. 이 알고리즘은 구현이 간단하여 많이 이용되기 때문에 꼭 알아둘 필요가 있다. 하지만 모든 경로를 다 구하므로 시간이 많이 걸린다는 단점이 있다.

● <VC>

```
for(k=0;k<n;k++){
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            if(map[i][j]>map[i][k]+map[k][j])
                map[i][j]=map[i][k]+map[k][j];
        }
    }
}
```

● <VB>

```
For k=0 to n
    For i=0 to n
        for j=0 to n
            If map(i, j)>map(i, k) + map(k, j) Then
                map(i, j) = map(i, k) + (k, j)
            End If
    Next k, j, i
```

— ● <VC>

```

#include<stdio.h>
#include<string.h>
#define MAX 27

int map[MAX][MAX];           //전체 그래프를 저장할 인접행렬
int chk[MAX], distance[MAX], layover[MAX];
                           //layover은 경유하는 경로 저장하는 배열
int n, m, w;
FILE *in=fopen("input5.txt", "r"), *out=fopen("output5.txt", "w");

void dijkstra(int a, int b){
    // 최단경로 알고리즘 Dijkstra, 탐욕적인 방법을 바탕으로 한
    int min, i, j, p, c=0;           // 동적계획
    char temp[MAX];
    if(a==b){
        fprintf(out, "0 %c\n", a+64);
        return;
    }
    for(i=1;i<=n;i++){
        chk[i] = 0;
        distance[i] = (map[a][i])? map[a][i] : 0x7fffffff;
    }
    layover[a] = 0;
    for(i=1;i<=n;i++){
        min=0x7fffffff;           // 0x7fffffff는 최대 정수값
        for(j=1;j<=n;j++){
            if(min>distance[j] && !chk[j]){
                min = distance[j];
                p = j;
            }
        }
        if(i==1) layover[p] = a;
        if(min==0x7fffffff) break;
        chk[p] = 1;
        for(j=1;j<=n;j++){
            if(map[p][j] && distance[j] > distance[p] + map[p][j]){
                distance[j] = distance[p] + map[p][j];
                // 최단경로 갱신
                layover[j] = p;
                // 경유경로 저장
            }
        }
    }
}

```

```

if(distance[b]!=0x7fffffff){           // 경로를 찾아주는 루틴임.
    fprintf(out, "%d ", distance[b]);
    sprintf(temp, "%c", b+64);
    for(i=b;i!=a;i=layover[i])
        sprintf(temp, "%s%c", temp, layover[i]+64);
    for(i=strlen(temp)-1; i>=0; i--)
        fprintf(out, "%c", temp[i]);
    fprintf(out, "\n");
}
else fprintf(out, "경로가 존재하지 않음!\n");
}

void main(void){
    int i, k;
    char st, ed;          // 사용할 변수의 선언
    fscanf(in,"%d %d\n", &n,&m);      // 정점, 간선수를 받음
    for(i=0;i<m;i++){
        fscanf(in,"%c %c %d\n", &st, &ed, &w);
        //시작정점, 끝정점, 가중치 입력
        st -= 64;           //입력 값이 문자이기 때문에 숫자로 바꿈
        ed -= 64;           // A->1, B->2, ..., Z->26로 바뀜
        map[st][ed] = w;    // 인접행렬 제작
    }
    fscanf(in,"%d\n", &k);
    for(i=0;i<k;i++){
        fscanf(in, "%c %c\n", &st, &ed);
        st -= 64;
        ed -= 64;           //각 노선 정보를 저장 st 출발점, ed 도착점
        dijkstra(st, ed);
    }
}

```

————— ● <VB>

```

Option Explicit
Const MAX As Integer = 27, MAXINT As Integer = 32700
Dim map(MAX, MAX) As Integer  '인접행렬
Dim chk(MAX) As Integer, distance(MAX) As Integer, layover(MAX) As Integer
'각 필요한 배열 선언
Dim n As Integer, m As Integer, w As Integer
Sub dijkstra(a As Integer, b As Integer)  '다익스트라 알고리즘
    Dim min As Integer, i As Integer, j As Integer, c As Integer, p As Integer
    Dim temp As String
    If a = b Then
        Print #2, "0 " + Chr(a + 64)
        Exit Sub
    End If

```

```

For i = 1 To n
    chk(i) = 0
    distance(i) = If(map(a, i) <> 0, map(a, i), MAXINT)
Next i
layover(a) = 0
For i = 1 To n
    min = MAXINT
    For j = 1 To n
        If min > distance(j) And chk(j) = 0 Then
            min = distance(j) : p = j      '그리디 기법 도입
        End If
    Next j
    If i = 1 Then layover(p) = a
    If min = MAXINT Then Exit For
    chk(p) = 1
    For j = 1 To n          '그리디를 활용한 동적계획
        If map(p, j) <> 0 And distance(j) > distance(p) + map(p, j) Then
            distance(j) = distance(p) + map(p, j)
            layover(j) = p
        End If
    Next j, i
    If distance(b) <> MAXINT Then
        Print #2, CStr(distance(b)) + " ";
        temp = Chr(b + 64) : i = b
        Do While i <> a
            temp = temp + Chr(layover(i) + 64) : i = layover(i)
        Loop
        For i = Len(temp) To 0 Step -1
            Print #2, Mid(temp, i + 1, 1);   '경로 출력 부
        Next i
        Print #2,
    Else
        Print #2, "경로가 존재하지 않음!"
    End If
End Sub

Sub main()
    Dim i As Integer, k As Integer, st As Integer, ed As Integer, temp As String
    Open "input5.txt" For Input As #1
    Open "output5.txt" For Output As #2
    Input #1, n, m
    For i = 0 To m - 1
        st = Asc(Trim(input(2, #1))) - 64
        ed = Asc(Trim(input(2, #1))) - 64      '맵을 입력 받는 부분
        Input #1, w
        map(st, ed) = w
    Next i
    Input #1, k
    For i = 0 To k - 1
        Line Input #1, temp
        st = Asc(Mid(temp, 1, 1)) - 64
        ed = Asc(Mid(temp, 3, 1)) - 64
        dijkstra st, ed
    Next i
End Sub

```

나. 제2회 교원컴퓨터프로그래밍 경진대회 문제 풀이



문제 V.6.3 (제2회 4번, 난이도 : ★★)

물류보관창고에 n 개의 상자가 존재한다. ($0 \leq i \leq n-1$, n 은 자연수) 이 n 개의 상자들은 특정한 순서에 따라 초기에는 “A” 위치에 쌓여 있으며, 출고지 시에 따라 창고 외부로 옮겨지게 된다. 맨 위에 쌓여있는 상자는 바로 옮겨질 수 있지만, 중간에 위치한 상자를 옮기기 위해서는 옮기고 싶은 상자 위의 상자들을 옆에 (“A” 위치 또는 “B” 위치) 다시 쌓아 올린 후, 옮기고 싶은 상자가 맨 위에 놓여 질 때 옮길 수 있다. 그리고 한번에 하나의 상자만이 이동이 가능하다. 아래의 입력 및 출력 예, 수행과정 예 등을 참고하여 프로그램을 작성하시오. (단, 실행파일은 PR4.EXE로 한다.)

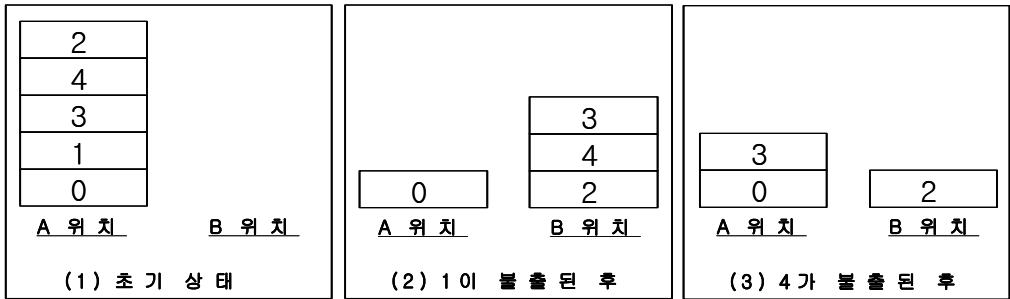
<입력형식> 상자의 개수, 초기 상자가 쌓여 있는 순서, 출고될 상자들의 번호와 순서는 “INPUT4.TXT” 파일에서 입력받는다.

(단, 각각의 숫자는 공백으로 구별한다.)

- 1) 입력 파일의 첫 번째 줄은 상자의 개수
- 2) 입력 파일의 두 번째 줄은 초기 상자가 놓인 순서
- 3) 입력 파일의 세 번째 줄은 출고지시(상자가 출고되는 순서)

<출력형식> 출고 지시에 의해 출고된 후 “A” 위치와 “B” 위치에 쌓여있는 상자들을 아래쪽에 있는 상자부터 순서대로 출력하고, “A” 위치에서 “B” 위치로 이동한 회수와 “B” 위치에서 “A” 위치로 이동한 회수의 합을 출력하라.(단, 출력은 “OUTPUT4.TXT” 파일로만 가능, 각각의 숫자는 공백으로 구별한다.)

- 1) 출력 파일의 첫 번째 줄은 “A” 위치에 놓인 상자가 쌓여있는 순서
- 2) 출력 파일의 두 번째 줄은 “B” 위치에 놓인 상자가 쌓여있는 순서
- 3) “A”에서 “B”로 “B”에서 “A”로의 이동 회수의 합



<수행과정의 예>

그림 (1)은 5개의 상자가 초기 위치 “A”에 “0 1 3 4 2” 순서대로 쌓여 있는 것을 보여준다.

그림 (2)는 상자 “1”이 출고된 후의 상태를 보여 준다. 즉, 상자 “1”이 출고되기 위해서는 “1” 위의 상자들이 “B” 위치로 옮겨져야 한다. 한번에 하나의 상자만이 이동 가능하므로 “B” 위치에 상자 “2”를 옮기고, 다음으로 상자 “4”를 상자 “2” 위에 쌓고, 마지막으로 상자 “3”을 상자 “4”위에 쌓게 된다.

그림 (3)은 상자 “4”가 출고된 후의 상태를 보여 준다. 즉, 상자 “4”가 출고되기 위해서는 “4”위에 있는 상자 “3”을 “A”위치의 상자 “0” 위에 옮겨야 한다.

입력화일(INPUT4.TXT) 예 :

```
5
0 1 3 4 2
1 4
```

출력화일(OUTPUT4.TXT) 예 :

```
0 3
2
4
```

<풀이> 전형적인 스택 문제입니다. 스택을 이용해서 푸는 방법에 대해서 소개하고 있습니다. 만약 C++언어를 사용한다면 객체로 제공되는 스택을 이용하면 간단하게 구현할 수 있다.
(코드의 길이가 VB보다 많이 짧다는 것을 알 수 있다.)

— ● <VC>

```
#include <stdio.h>
#include <stack>
#define MAX 1000

using namespace std;
stack<int> S1, S2;      //2개의 스택 객체를 선언
int work[MAX], n, count, w;

void main(void){
    int i, j;
    FILE *in=fopen("input4.txt", "r"), *out=fopen("output4.txt", "w");
    fscanf(in, "%d", &n);
    for(i=0; i<n; i++){
        fscanf(in, "%d", &temp);
        S1.push(temp);      // 입력 자료 모두를 1스택에 푸시
    }
    for(i=0; fscanf(in, "%d", &work[i])!=EOF; i++);
    w = i;                  // 작업 수를 w에 저장
    for(i=0; i<w; i++){
        if(search(work[i], 1)){      //스택 1에 원하는 작업이 있으면
            while(temp!=work[i]){
                temp=S1.pop();
                S2.push(temp);    //작업 전 항목은 스택2로
                count++;
            }
            temp = S2.pop();      // 찾은 작업은 없앰
        }
        if(search(work[i], 2)){      //스택 2에 원하는 작업이 있으면
            while(temp!=work[i]){
                temp=S2.pop();
                S1.push(temp);    //작업 전 항목은 스택1로
                count++;
            }
            temp = S1.pop();      // 찾은 작업은 없앰
        }
    }
}
```

```

while(!S1.empty()){           // S1스택의 값 모두 출력
    fprintf(out, "%d ", S1.front());
    S1.pop();
}   fprintf(out, "\n");
while(!S2.empty()){           //S2스택의 값 모두 출력
    fprintf(out, "%d ", S2.front());
    S2.pop();
}   fprintf(out, "\n");
fprintf(out, "%d\n", count - w); //작업횟수 출력
}

int search(int item, int mode){ // 스택 내에 원하는 자료가 있는지 검사
    int I, j, flag, temp[MAX];
    if(mode==1){
        for(i=0;!S1.empty();i++){
            if(S1.front()==item) return 1;
            temp[i] = S1.pop();
        }
        for(j=i-1;j>=0;j--) S1.push(temp[j]);
        return 0;
    }
    else{
        for(i=0;!S2.empty();i++){
            if(S2.front()==item) return 1;
            temp[i] = S2.pop();
        }
        for(j=i-1;j>=0;j--) S2.push(temp[j]);
        return 0;
    }
}

```

— ● <VB>

```

Option Explicit
Const MAX As Integer = 1000
Dim stackA(MAX) As Integer, stackB(MAX) As Integer,
      work(MAX) As Integer, n As Integer, temp As Integer
Dim topA As Integer, topB As Integer, w As Integer,
      count As Integer
Sub main()
    Dim i As Integer, j As Integer
    Open "input4.txt" For Input As #1
    Open "output4.txt" For Output As #2
    Input #1, n
    For i = 1 To n
        Input #1, temp
        Push temp, 1           '입력자료를 모두 받아 스택A에 모두 저장
    Next i

```

```

i = 1
Do
    Input #1, work(i)
    i = i + 1
Loop While Not EOF(1)
w = i - 1          '작업 수를 w에 저장
For i = 1 To w      '작업을 수행
    If search(work(i), 1) = 1 Then
        Do
            temp = Pop(1)
            Push temp, 2
            count = count + 1          '작업횟수 증가
        Loop While temp <> work(i)
        '만약 A스택에 있으면 원하는 작업이 있을 때 까지 B스택으로 옮기고
        temp = Pop(2)                '작업한 결과는 버림
    End If
    If search(work(i), 2) = 1 Then
        Do
            temp = Pop(2)
            Push temp, 1
            '만약 B스택에 작업이 있으면 작업까지 A스택에 옮긴다.
            count = count + 1          '작업횟수 증가
        Loop While temp <> work(i)
        temp = Pop(1)                '작업한 결과는 버림
    End If
Next i
For i = 1 To topA
    Print #2, CStr(stackA(i)) + " ";      '스택A에 있는 값을 모두 출력
Next i
Print #2,
For i = 1 To topB
    Print #2, CStr(stackB(i)) + " ";      '스택B에 있는 값을 모두 출력
Next i
Print #2,
Print #2, CStr(count - w)                '작업횟수 출력

End Sub

Sub Push(ByVal item As Integer, ByVal mode As Integer)
'스택의 삽입 연산을 담당 mode가 1이면 스택A 2이면 스택B를 의미
    If mode = 1 Then
        topA = topA + 1
        stackA(topA) = item
    Else
        topB = topB + 1
        stackB(topB) = item
    End If
End Sub

```

```

        End If
End Sub
Function Pop(ByVal mode As Integer) As Integer
'스택의 제거(출력)연산을 담당 mode가 1이면 스택A 2이면 스택B를 의미
    If mode = 1 Then
        Pop = stackA(topA)
        topA = topA - 1
    Else
        Pop = stackB(topB)
        topB = topB - 1
    End If
End Function

Function search(ByVal item As Integer, ByVal mode As Integer) As Integer
'스택내에 원하는 자료가 있는지 검사 mode가 1이면 스택A 2이면 스택B를 의미
    Dim i As Integer, flag As Integer
    If mode = 1 Then
        For i = 1 To topA
            If stackA(i) = item Then flag = 1
        Next i
    Else
        For i = 1 To topB
            If stackB(i) = item Then flag = 1
        Next i
    End If
    If flag = 1 Then search = 1 Else search = 0
End Function

```

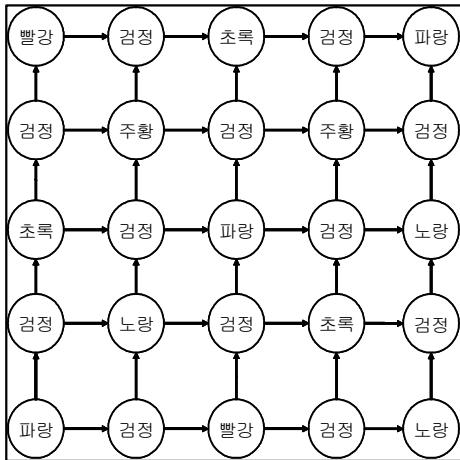


문제 V.6.4 (제2회 5번, 난이도 : ★★)

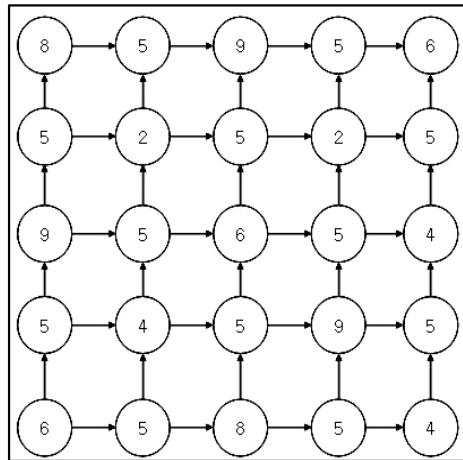
아래의 [그림 5-1] 은 6가지의 색(검정, 주황, 파랑, 초록, 노랑, 빨강)으로 이루어져 있으며 원 안에 있는 색들은 입력파일의 입력값(5 2 6 9 4 8)에 따라[그림 5-2]와 같이 정수 값을 가질 수 있다.

이때, 원쪽 가장 아래의 색(파랑)부터 시작하여 화살표를 따라 움직이면서 오른쪽 가장 위쪽의 색(파랑)까지 갈 수 있는 경우(예를 들어, 파랑-검정-노랑-검정-주황-검정-주황-검정-파랑)는 여러 가지가 있다.

여러 경우 중에서 원쪽 가장 아래의 색(파랑)에서 시작하여 원 안에 있는 색의 값을 더하여 오른쪽 가장 위쪽에 있는 색(파랑)까지의 합 중에서 최소값을 구하는 프로그램을 작성하시오.(단, 실행파일은 PR5.EXE로 한다.)



[그림 5-1]



[그림 5-2]

<입력형식> 각 색들의 값(정수)은 입력파일(INPUT5.TXT)에서 입력받는다. 예를 들어, 입력파일에 입력 값이 5 2 6 9 4 8 과 같이 주어졌다면 각 색들의 값(정수)들은 다음과 같다.

입력 순서	1	2	3	4	5	6
색	검정	주황	파랑	초록	노랑	빨강
입력 값	5	2	6	9	4	8

즉, 검정은 5의 값을 가지며, 주황은 2, 파랑은 6, 초록은 9, 노랑은 4, 빨강은 8의 값을 가진다. 참고로, 각 원 안의 색들에 대한 값이 주어지고 나면 [그림 5-1]은 [그림 5-2]로 바뀌게 될 것이다.

<출력형식> 출력은 원 안의 각각의 값을 더하여 최소값을 출력한다.

예를 들어, 입력파일(INPUT5.TXT)에서 5 2 6 9 4 8 를 입력받았다면 최소값은 $6+5+4+5+2+5+2+5+6=40$ 이 된다.

출력파일은 OUTPUT5.TXT 이다.

입력화일(INPUT5.TXT)

예 :

5 2 6 9 4 8

출력화일(OUTPUT5.TXT)

예 :

40

<풀이> 동적계획법을 연습해 볼 수 있는 좋은 문제이다. 막연히 접근하기에는 상당히 어려워 보이나, 동적계획법 문제의 특성상 점화식만 만들어 내면 아주 간단하게 해결되는 재미있는 문제이다.

이 문제의 화살표 방향을 잘 살펴보면 항상 위쪽과 오른쪽으로만 갈 수 있다는 것을 알 수 있다. 이를 약간 활용하면 현 위치로 오는 방법은 아래쪽과 왼쪽뿐이라는 결론을 얻는다. 여기서 힌트를 얻어서 점화식을 만들어 보자.

점수 정보를 가지고 있는 2차원 배열을 T라고 하자. 앞으로 이 T를 정답저장 배열로 그대로 이용하겠다.

- ① $T(i, j)$ 는 i, j 까지 오면서 얻을 수 있는 최소값을 저장한다
- ② 출발점인 $T(5, 1)$ 은 $T(5, 1)$ 의 원래 값이 가질 수 있는 최소값이다.
- ③ 우리가 구하고자하는 최종 값은 $T(1, 5)$ 이다.
- ④ 여기서 $T(i, j)$ 에 대한 점화식을 만들어 보면, $T(i, j)$ 로 올 수 있는 위치는 $T(i+1, j)$ 또는 $T(i, j-1)$ 이다. $T(i, j)$ 가 최소가 되기 위해서는 $T(i+1, j)$, $T(i, j-1)$ 중 더 적은 값을 선택하는 것이 최선의 방법이다. 참고로 다음과 같은 점화식이 만들어 진다.

$$\pi(i, j) = \min \{ \pi(i+1, j), \pi(i, j-1) \} + T(i, j)$$

우리는 위에서 구한 점화식을 가지고 반복문을 이용해서 $T(5, 1)$ 에서부터 $T(1, 5)$ 의 방향으로 맵을 채워서 $T(1, 5)$ 의 값을 출력하면 원하는 정답을 얻을 수 있다.

<VC>

```
#include<stdio.h>
int map[7][7], c[6];
void map_init(void){
    int i;
    map[1][1]=c[5]; map[1][2]=c[0]; map[1][3]=c[3]; map[1][4]=c[0]; map[1][5]=c[2];
    map[2][1]=c[0]; map[2][2]=c[1]; map[2][3]=c[0]; map[2][4]=c[1]; map[2][5]=c[0];
    map[3][1]=c[3]; map[3][2]=c[0]; map[3][3]=c[2]; map[3][4]=c[0]; map[3][5]=c[4];
    map[4][1]=c[0]; map[4][2]=c[4]; map[4][3]=c[0]; map[4][4]=c[3]; map[4][5]=c[0];
    map[5][1]=c[2]; map[5][2]=c[0]; map[5][3]=c[5]; map[5][4]=c[0]; map[5][5]=c[4];
    for(i=0;i<7;i++){
        map[i][0] = 100;
        map[6][i+2] = 100;
    }
}
int min(int a, int b){
    return (a>b)? b : a;
}
void main(void){
    int i, j;
    FILE *in=fopen("input5.txt", "r"), *out=fopen("output5.txt", "w");
    for(i=0;i<6;i++)
        fscanf(in, "%d", &c[i]);
    map_init();
    for(i=5;i>0;i--){
        for(j=1;j<6;j++)
            map[i][j] = min(map[i][j-1], map[i+1][j]) + map[i][j];
    }
    fprintf(out, "%d\n", map[1][5]);
}
```

<VB>

```
Option Explicit
Dim map(6, 6) As Integer, c(6) As Integer

Sub main()
    Dim i As Integer, j As Integer
    Open "input5.txt" For Input As #1
    Open "output5.txt" For Output As #2
```

```

For i = 1 To 6
    Input #1, c(i)
Next i
mapinit c(1), c(2), c(3), c(4), c(5), c(6)
For i = 5 To 1 Step -1
    For j = 1 To 5
        map(i, j) = min(map(i, j - 1), map(i + 1, j)) + map(i, j)
        '점화식을 이용한 동적계획법으로 해결하는 부분(분석 필요)
    Next j
Next i
Print #2, CStr(map(1, 5))
End Sub

Sub mapinit(a As Integer, b As Integer, c As Integer, d As Integer, e As Integer,
f As Integer)
    Dim i As Integer
    map(1, 1) = f : map(1, 2) = a : map(1, 3) = d : map(1, 4) = a : map(1, 5) = c
    map(2, 1) = a : map(2, 2) = b : map(2, 3) = a : map(2, 4) = b : map(2, 5) = a
    map(3, 1) = d : map(3, 2) = a : map(3, 3) = c : map(3, 4) = a : map(3, 5) = e
    map(4, 1) = a : map(4, 2) = e : map(4, 3) = a : map(4, 4) = d : map(4, 5) = a
    map(5, 1) = c : map(5, 2) = a : map(5, 3) = f : map(5, 4) = a : map(5, 5) = e
    '입력받은 값으로 맵에 대입
    For i = 1 To 4
        map(i, 0) = 100
        map(6, i + 1) = 100
    Next i
    '맵의 외부를 최대값으로 보초를 세움
End Sub

Function min(ByVal a As Integer, ByVal b As Integer) As Integer
    If a > b Then min = b Else min = a
End Function

```

다. 제3회 교원컴퓨터프로그래밍 경진대회 문제 풀이



문제 V.6.5 (제3회 4번, 난이도 : ★★)

스케줄링이란 특정 자원에 대해 그 자원을 요청하고 있는 대상들 중 누구에게 먼저 그 자원을 할당해 줄 것인가를 결정하는 일을 말하며, 그 중에서 특히 프로세스(Process) 스케줄링 기법은 프로세스들을 대상으로 프로세서(Processor) 자원을 할당해 주는 일을 말한다. 스케줄링 기법 중에 우선순위 알고리즘은 각 프

로세스에 우선순위가 주어지고, 우선순위가 제일 높은 프로세스에 프로세서 자원이 할당된다. 우선순위가 같은 경우에는 은행의 번호대기표와 같이 FCFS(First-Come-First-Served)방식으로 처리된다. 우선순위 알고리즘에서 우선순위가 낮은 프로세스는 우선순위가 높은 프로세스에 밀려 무기한 연기(indefinite postponement)가 발생될 수 있으며, 이를 해결하기 위해 대기 시간이 경과할수록 대기하는 프로세스의 우선순위를 높여줌으로써 결국은 자원을 할당 받아 사용할 수 있도록 하는 에이징(aging) 기법 등이 사용된다.

본 문제는 우선순위 알고리즘과 에이징 기법을 응용하여 간단한 작업 스케줄링 프로그램을 작성하는 것이다. 입력데이터는 “프로세스 번호”, “그룹별 가중치”, “서비스 가중치”로 구성된다. 우선순위는 가중치의 합으로 결정되고, 가중치는 “그룹별 가중치”와 “서비스 가중치”의 합이다. 여기서 “그룹별 가중치”는 우선순위 알고리즘에서의 프로세스 우선순위이며, “서비스 가중치”는 에이징 기법에서 대기 시간이 경과할수록 대기하는 프로세스의 우선순위를 높여주기 위한 것이다.

다음의 조건, 입력 형식, 수행 과정, 출력 형식 등을 참조하여 작업 스케줄링 프로그램을 작성하시오.(단, 실행파일은 “PR4.EXE”로 한다.)

조 건

- ① 우선순위는 가중치의 합으로 결정하고 가중치의 합은 “그룹별 가중치”와 “서비스 가중치”의 합으로 한다.
- ② 가중치의 합이 높은 프로세스가 먼저 처리된다. (우선순위 알고리즘)
- ③ 가중치가 같을 경우에는 입력데이터의 순서에서 앞에 있는 프로세스가 먼저 처리된다. (First-Come-First-Served)
- ④ 입력데이터의 순서에서 뒤에 있는 프로세스가 먼저 처리된 경우에는 처리된 프로세스보다 앞에 있는 프로세스들의 “서비스 가중치”를 1씩 증가시킨다.(에이징 기법)
- ⑤ 모든 입력데이터가 처리될 때까지 ①부터 ④의 조건을 반복하여 적용한다.

<입력형식> 입력파일(INPUT4.TXT)에서 처리할 데이터를 입력받는다.

- 1) 입력파일의 첫 번째 행은 처리할 데이터의 개수이다.
- 2) 두 번째 행부터 처리할 데이터이다.

3) 처리할 데이터의 형식은 다음과 같다.

전체 7자리	처음 3자리	다음 1자리	마지막 3자리
예) 101C002	프로세스 번호	그룹별 가중치	서비스 가중치
	101	C	002

〈데이터 형식〉

4) 그룹별 가중치의 값은 다음과 같다.

그룹별 가중치	A	B	C	D	E
값	4	3	2	1	0

〈그룹별 가중치 값〉

5) 가중치를 계산한 예는 다음과 같다.

입력데이터	프로세스 번호	그룹별 가중치	서비스 가중치	가중치 합
101C002	101	2	2	4
102D001	102	1	1	2
103A000	103	4	0	4
104B002	104	3	2	5
105C003	105	2	3	5
106D002	106	1	2	3

〈가중치 계산 예〉

<수행과정> 가중치 계산표는 아래의 입력 예 데이터(INPUT4.TXT)로 가중치의 합을 계산한 것이다. 프로세스 번호 104와 105가 가중치의 합이 5로 우선순위가 가장 높다(조건 2 우선순위 알고리즘). 둘 중 프로세스 번호 104가 먼저 입력되었으므로 먼저 처리된다(조건 3 FCFS). 그 후 프로세스 번호 101, 102, 103은 104보다 먼저 입력되었지만 104가 먼저 처리되었으므로 “서비스 가중치”가 1씩 증가된다(조건 4 에이징 기법).

다음의 표는 프로세스 번호 104가 처리된 후 변경된 가중치의 합을 보여준다.

입력데이터	프로세스번호	그룹별 가중치	서비스 가중치	가중치 합	비고
101C003	101	2	3	5	
102D002	102	1	2	3	
103A001	103	4	1	5	
104B002	104	3	2	5	처리
105C003	105	2	3	5	
106D002	106	1	2	3	

〈프로세스 번호 104 처리 후 변경된 데이터〉

이때 프로세스 번호 101, 103, 105가 가중치의 합이 5로 우선순위가 가장 높다(조건 2 우선순위 알고리즘). 셋 중 프로세스 번호 101이 가장 먼저 입력되었으므로 먼저 처리된다(조건 3 FCFS). 프로세스 번호 101은 제일 먼저 입력된 것이므로 다른 프로세스들의 “서비스 가중치” 변화는 없다. 이후 모든 데이터들이 처리될 때까지 위의 과정을 반복적으로 수행한다.

<출력형식> 작업 스케줄링 프로그램에 의해 처리된 데이터는 출력파일(OUT PUT4.TXT)에 각 행마다 처리된 순서대로 출력한다.(출력예 참조)

입력화일(INPUT4.TXT) 예 :

6
101C002
102D001
103A000
104B002
105C003
106D002

출력화일(OUTPUT4.TXT) 예 :

104B002
101C003
103A001
105C003
102D004
106D002

<풀이> 이 문제는 특별한 방법이 존재하는 것은 아니다. 위의 자료를 정확하게 받아서 시뮬레이션 하는 방법으로 해결하였다. 우선순위 큐를 두게 되면 조금 더 편하게 풀 수 있지만 여기서는 정렬을 통해 우선순위 순으로 정렬하고, 그 결과를 가지고 프로세스를 실행한다.

— ● <VC>

```
#include <stdio.h>
#include <algorithm>

struct process{
    int id, gp, sp, sum;
    bool operator()(const process &a, const process &b){
        return (a.sum * 100 - a.id) < (b.sum * 100 - b.id);
    }
};                                //프로세스를 저장할 구조체 선언

process p[100];                    //프로세스를 100개 까지 사용가능

void main(void){
    FILE *in=fopen("input4.txt", "r"), *out=fopen("output4.txt", "w");
    int i, n;
    char temp[10];
    fscanf(in,"%d", &n);
    for(i=0;i<n;i++){
        fscanf(in,"%s",temp);
        p[i].id = ((temp[0]-'0')*100) + ((temp[1]-'0')*10) + (temp[2]-'0');
        //프로세스 번호
        p[i].gp = 'E' - temp[3];          //그룹 우선순위
        p[i].sp = ((temp[4]-'0')*100) + ((temp[5]-'0')*10) + (temp[6]-'0'); //서비스 우선순위
        p[i].sum = p[i].gp + p[i].sp;     //총괄 우선순위
    }                                //파일로부터 프로세스를 받으면서 바로 분리
    while(n){
        std::sort(p, p+n, process());
        //우선순위에 따른 정렬(순위 동일할 경우 FIFO까지) 구현
        sprintf(temp, "%03d%c%03d", p[n-1].id, 'E'-p[n-1].gp, p[n-1].sp);
        fprintf(out, "%s\n", temp);      //최고 우선순위 출력
        for(i=0;i<n;i++)
            if(p[i].id < p[n-1].id){    //예이징기법 도입
                p[i].sp++;
                p[i].sum++;
            }
        n--;                            //처리할 프로세스 수 1개 감소
    }
}
```

<VB>

```

Option Explicit
Dim id(100) As Integer, gp(100) As Integer, sp(100) As Integer, sum(100) As Integer, n As Integer
Sub Swap(i As Integer, j As Integer)
    Dim temp As Integer
    temp = sum(i) : sum(i) = sum(j) : sum(j) = temp
    temp = id(i) : id(i) = id(j) : id(j) = temp
    temp = gp(i) : gp(i) = gp(j) : gp(j) = temp
    temp = sp(i) : sp(i) = sp(j) : sp(j) = temp
End Sub

Sub sort()
    Dim i As Integer, j As Integer
    For i = 0 To n - 2
        For j = i + 1 To n - 1
            If sum(i) > sum(j) Then
                Swap i, j
            ElseIf sum(i) = sum(j) And id(i) < id(j) Then Swap i, j
            Next j, i
    End Sub

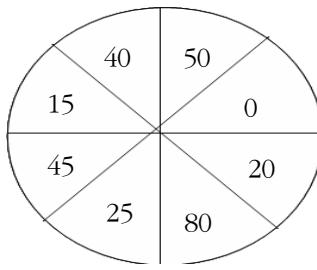
Sub main()
    Open "input4.txt" For Input As #1
    Open "output4.txt" For Output As #2
    Dim i As Integer, temp As String
    Input #1, n
    For i = 0 To n - 1
        Line Input #1, temp
        id(i)=Val(Mid(temp, 1, 1))*100+Val(Mid(temp, 2, 1))*10+Val(Mid(temp, 3, 1))
        gp(i) = Asc("E") - Asc(Mid(temp, 4, 1)) '그룹 우선순위 숫자 변환
        sp(i)=Val(Mid(temp,5,1))*100+Val(Mid(temp,6,1))*10+Val(Mid(temp,7,1))
        sum(i) = gp(i) + sp(i) '총괄 우선 순위 구함
    Next i           '파일로부터 한 줄 받으면 바로 분리
    Do While n <> 0
        sort      '우선순위에 따른 정렬 합수 호출(위에 작성)
        temp = Format(Str(id(n - 1)), "000") +
               Chr(Asc("E") - gp(n - 1)) +
               Format(Str(sp(n - 1)), "000") '출력 형식에 맞게 조립
        Print #2, temp          '최고 우선순위 출력
        For i = 0 To n - 1
            If id(i) < id(n - 1) Then
                sp(i) = sp(i) + 1
                sum(i) = sum(i) + 1
            End If
        Next i                  '이어짐 기법 도입
        n = n - 1                '프로세스 수 1개 감소
    Loop
End Sub

```



문제 V.6.6 (제3회 5번, 난이도 : ★★)

임의의 8개의 숫자가 적혀있는 다프 판이 있다. 동시에 4개의 다프를 던져서 나올 수 있는 숫자의 합을 S라고 한다면, 합(S)을 만족하는 경우의 수와 총 개수를 구하는 프로그램을 작성하시오.(단, 실행 파일은 “PR5.EXE” 이다.)



〈다트 판〉

위의 그림처럼 다프 판에 임의의 숫자 8개가 적혀있다고 가정하자. 만약 동시에 4개의 다프를 던져서 그 합이 100이 되는 경우의 수와 총 개수는 다음과 같다.

80 20 0 0	80에 1개, 20에 1개, 0에 2개의 다프가 꽂힌 경우
50 50 0 0	50에 2개, 0에 2개의 다프가 꽂힌 경우
50 25 25 0	50에 1개, 25에 2개, 0에 1개의 다프가 꽂힌 경우
50 20 15 15	45, 40, 15, 0에 각각 1개의 다프가 꽂힌 경우
45 40 15 0	
45 25 15 15	
45 20 20 15	
40 40 20 0	
40 25 20 15	
40 20 20 20	40에 1개, 20에 3개의 다프가 꽂힌 경우
25 25 25 25	25에 4개의 다프가 꽂힌 경우
11	합(S) 100을 만족하는 총 개수

다트 판에 적혀있는 8개의 숫자와 4개의 다프를 던져 나올 수 있는 숫자의 합(S)은 입력파일에서 각각 입력받는다고 할 때, S를 만족하는 경우의 수를 모두 구하고, 총 개수를 구하면 된다. (단, 다프가 바닥에 떨어져 실패하는 경우는 없고 반드시 8개의 숫자판 중 어느 하나에 꽂힌다고 가정하며, 4개의 다프가 모두 한곳에 꽂힐 수도 있다고 가정한다.)

<입력형식> 다트 판에 적힐 8개의 숫자와 동시에 4개의 다트를 던져 나올 수 있는 합(S)을 “INPUT5.TXT” 파일에서 반드시 입력받는다.
입력조건은 다음과 같다.

- 1) 입력파일의 첫 번째 줄에는 다트 판에 적힐 숫자 8개 입력한다.(입력 예 참조)
(단, 8개의 임의의 숫자는 빈칸으로 구별하며, 각 숫자는 1000 이하로 입력한다.)
- 2) 입력파일의 두 번째 줄에는 다트를 던져 나올 수 있는 합(S)을 입력한다.(입력 예 참조)

<출력형식> 다트를 던져 나올 수 있는 합(S)을 만족하는 모든 경우의 수와 총 개수를 “OUTPUT5.TXT” 파일에 출력한다.
출력조건은 다음과 같다.

- 1) 각 행마다 S를 만족하는 경우의 수를 출력한다.(출력 예 참조)
(각 줄에는 다트 판에 꽂힌 다트의 값을 빈 칸으로 구분하여 숫자 4개를 출력)
- 2) 마지막 줄에는 총 개수를 출력한다.(출력 예 참조)
- 3) 만약, S를 만족하는 경우의 수가 없을 때에는 0을 출력한다.

입력화일(INPUT5 . T X T)

예 :

0 15 20 25 40 45 50 80
100

출력화일(OUTPUT5.TXT)

예 :

80 20 0 0
50 50 0 0
50 25 25 0
50 20 15 15
45 40 15 0
45 25 15 15
45 20 20 15
40 40 20 0
40 25 20 15
40 20 20 20
25 25 25 25
11

<풀이> 기본적으로 이 문제는 백트래킹으로 접근할 수 있다. 다트수가 4개로 정해져있기 때문에 백트래킹이 아니라 4중 for문으로 해결 가능하지만, 다트수가 4개가 아니라 임의의 n개의 다트를 던져서 100을 맞추는 문제로 바뀐다면 백트래킹이나 동적계획법이 아니면 정답을 구하기 어렵기 때문에 백트래킹으로 풀이하였다.

————— ● <VC>

```
#include <stdio.h>
#include <stdlib.h>
int score[8], d[4], cnt;
char temp[1000];
FILE *in=fopen("input5.txt", "r"), *out=fopen("output5.txt", "w");

int Compare( const void *a, const void *b ){
    return *(int *)b - *(int *)a;
}

void back_tracking(int n, int depth, int start){
    int i;
    if(depth==4){           // 4단까지 팠으면
        if(!n){
            cnt++;          //합이 100이면 출력
            fprintf(out, "%d %d %d %d\n", d[0], d[1], d[2], d[3]);
        }
    }
    else
        for(i=start;i<8;i++)
            if(n-score[i]>=0){
                d[depth] = score[i];
                back_tracking(n-score[i], depth + 1, i);
                //답을 찾을 때 까지 재귀 호출(DFS)
            }
}
void main(void){
    int i, sum;
    for(i=0;i<8;i++)
        fscanf(in,"%d",&score[i]); // 다트 점수판을 입력받음
    qsort(score, 8, sizeof(int), Compare);
    // 다트 점수가 오름차순이나 내림차순으로 입력된다고 정해지지 않았으므로,
    fscanf(in,"%d",&sum);      // 따로 정렬함
    back_tracking(sum, 0, 0);   // 백트래킹 시작
    fprintf(out, "%d\n",cnt);
}
```

<VB>

```

Option Explicit
Dim score(8) As Integer, d(4) As Integer, cns As Integer, temp As String, cnt As Integer

Sub back_tracking(n As Integer, depth As Integer, start As Integer)
    Dim I As Integer
    If depth = 4 Then      ' 4단까지 끝으면...
        If n = 0 Then
            cnt = cnt + 1      '합이 100이면 출력
            Print #2, CStr(d(3)) + " " + CStr(d(2)) + " " +
                CStr(d(1)) + " " + CStr(d(0))
        End If
    Else
        For I = start To 7
            If n - score(I) >= 0 Then
                d(depth) = score(I)      '합을 구할 때 까지 재귀 호출(DFS)
                back_tracking n - score(I), depth + 1, I
            End If
        Next I
    End If
End Sub

Sub main()
    Dim I As Integer, sum As Integer
    Open "input5.txt" For Input As #1
    Open "output5.txt" For Output As #2
    For I = 0 To 7
        Input #1, score(I)
    Next I
    Input #1, sum
    back_tracking sum, 0, 0      '백트래킹 시작
    Print #2, CStr(cnt)
End Sub

```

'입력값이 정렬되었다는 말이 없으므로 파일의 데이터를 받을 때 정렬해 주는 것이 좋다. 교원컴퓨터프로그래밍경진대회에서는 정렬을 안해도 만점이 나왔기 때문에 정렬하는 부분은 생략했음.

라. 제4회 교원컴퓨터프로그래밍 경진대회 문제 풀이



문제 V.6.7 (제4회 4번, 난이도 : ★★★)

길동이가 길을 걷고 있는데, 산신령이 나타나 길에 동전을 일렬로 놓으면서 “길동아, 네 마음껏 동전을 주워가라. 그런데 연속해서 3개의 동전을 줍지는 못 한다.”고 하였다.

길동이가 가장 많은 금액의 동전을 주울 수 있도록 도와주는 프로그램을 작성해 시오.

예를 들어, 길에 일렬로 놓은 동전이 아래와 같다면 길동이는 최대 38원 ($7+10+10+11$)을 주울 수 있다.

- 5원 7원 10원 1원 2원 10원 11원 6원

<입력 형식>

입력파일은 “INPUT.TXT”이다.

첫째 줄은 동전의 갯수가 입력되고 동전의 수는 100이하의 자연수이다. 다음 줄에는 동전의 값들이 입력된다. 입력된 값들은 자연수이고 합해서 30,000을 초과하지 않는다.

<출력 형식>

출력파일은 “OUTPUT.TXT”이다.

출력내용은 가장 많이 주울 수 있는 동전의 총 합계를 출력한다.

<입출력 예>

입력 파일(INPUT.TXT)

8

5 7 10 1 2 10 11 6

출력 파일(OUTPUT.TXT)

38

— ● <VC>

```
#include<stdio.h>
// 동적계획법으로 해결했음
int dt[102], i, n, in[102];

int max(int a, int b)
{
    if( a>b ) return a;
    return b;
}

int main(void)
{
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    scanf("%d", &n);
    for(i=1;i<=n;i++)
        scanf("%d", &in[i]);
    for(i=3, dt[1]=in[1], dt[2]= in[1] + in[2]; i<=n; i++){           // 점화식
        dt[i] = max( max( dt[i-2], in[i-1] + dt[i-3]) + in[i], dt[i-1]);
    }
    printf("%d\n", dt[n]);
    return 0;
}
```

— ● <VB>

```
Function max(ByRef a As Integer, ByRef b As Integer)
    If a > b Then max = a Else max = b  ' 동적계획법으로 해결했음.
End Function

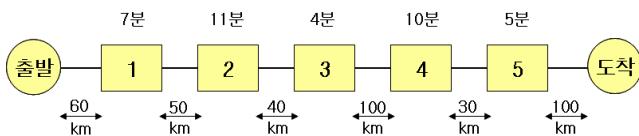
Sub main()
    Dim dt(102) As Long, inp(102) As Long
    Dim i As Long, n As Long
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2
    Input #1, n
    For i = 1 To n
        Input #1, inp(i)
    Next i
    dt(1) = inp(1) : dt(2) = inp(1) + inp(2)
    For i = 3 To n
        dt(i) = max(max(dt(i - 2), inp(i - 1) + dt(i - 3)) + inp(i), dt(i - 1))
        '점화식
    Next i
    Print #2, dt(n)
End Sub
```



문제 V.6.8 (제4회 5번, 난이도 : ★★★)

헬기로 출발지에서 목적지까지 원하는 짐을 운송하려고 한다. 하지만 거리에 따라 필요한 경우 경유지에 들러서 연료를 보충해야 한다. 경유지들은 출발지에서 도착지로 가는 경로에 있으며 1번부터 차례로 번호가 붙어 있다. 이 헬기는 한 번의 연료 보급으로 최대로 갈 수 있는 거리와 경유지마다 연료를 보충하는 데 걸리는 시간이 정해져 있다. 우리의 목적은 경유지에서 연료를 넣는 시간을 최소화 하면서 물건을 운송할 수 있는 방법을 결정하는 것이다. 이를 해결할 수 있는 프로그램을 작성하시오.(출발점은 연료가 최대로 주유된 상태임.)

예를 들어, 아래의 그림과 같이 경유지가 5개 있고, 한 번 주유를 하면 최대 140km를 갈 수 있는 경우를 생각해 보자.



헬기가 출발지에서 경유지 1, 3, 5번을 차례로 방문하여 도착지까지 갈 수도 있고, 경유지 2, 4번을 방문하여 갈 수도 있다. 이 때 1, 3, 5번을 방문하는 경우에는 16분($7+4+5$)이 걸리는데 2, 4번을 방문하게 되면 21분($11+10$)이 걸리게 되므로 1, 3, 5번을 방문하는 것이 주유 시간을 최소화 하게 된다.

<입력 형식>

입력파일은 “INPUT.TXT”이다.

첫째 줄에는 주유를 하지 않고 갈 수 있는 최대 거리가 주어진다. 둘째 줄에는 경유지의 수가 입력되는데 경유지의 수는 100이하이다. 셋째 줄에는 인접한 경유지 사이의 거리(km)가 차례로 주어진다. 넷째 줄에는 경유지별 주유 시간(분)이 차례로 주어진다. 모든 입력은 양의 정수이다.

<출력 형식>

출력파일은 “OUTPUT.TXT”이다.

출력내용은 경유지에서 주유하는데 걸리는 총 주유 시간(분)과 방문한 경유지의 수를 출력한다.

<입출력 예>

입력파일(INPUT.TXT)

```
140  
5  
60 50 40 100 30 100  
7 11 4 10 5
```

출력파일(OUTPUT.TXT)

```
16 3
```

<풀이> 동적계획법으로 풀 수 있는 문제이다.

문제의 조건에 따라서 점화식을 세우면 된다.

점화식은 소스를 참고할 수 있도록 하자.

<VC>

```
#include <stdio.h>
#define MAXN 110
struct lay{
    int time, distance;
};
struct table{
    int min_time, times;
};
lay inp[MAXN];
table T[MAXN];
int N, FMAX;
int main(void){
    int i, k;
    freopen("input.txt" , "r", stdin);
    freopen("output.txt", "w", stdout);
    scanf("%d %d", &FMAX, &N);
    for( i = 1, inp[0].distance = 0 ; i <= N + 1 ; i++ ){
        scanf("%d", &inp[i].distance);
        if( FMAX < inp[i].distance ){
            printf("0 0\n");
            return 0;
        }
        inp[i].distance += inp[i-1].distance;
    }
    for( i = 1, inp[0].time = 0, inp[N+1].time = 0 ; i <= N ; i++ ) scanf("%d", &inp[i].time);
    for( i = 1 ; i <= N+1 ; i++ ){
        int min = 0xffffffff, pt;
        for( k = 0 ; k < i ; k++ ){
            if( inp[i].distance - inp[k].distance <= FMAX ){
                if( min > T[k].min_time ){
                    min = T[k].min_time;
                    pt = k;
                }
            }
        }
        T[i].min_time = min + inp[i].time;
        T[i].times = T[pt].times + 1;
    }
    printf("%d %d\n", T[N+1].min_time, T[N+1].times - 1);
    return 0;
}
```

— ● <VB>

```

Sub Main()
    Dim inp_time(110) As Long, inp_distance(110) As Long
    Dim T_min(110) As Long, T_time(110) As Long
    Dim min As Long, pt As Long
    Dim N As Long, FMAX As Long
    Dim i As Long, k As Long
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2
    Input #1, FMAX, N
    For i = 1 To N + 1
        Input #1, inp_distance(i)
        If FMAX < inp_distance(i) Then
            Print #2, "0 0"
        End If
        inp_distance(i) = inp_distance(i) + inp_distance(i - 1)
    Next i
    For i = 1 To N
        Input #1, inp_time(i)
    Next i
    For i = 1 To N + 1
        min = 999999
        For k = 0 To i - 1
            If inp_distance(i) - inp_distance(k) <= FMAX Then
                If min > T_min(k) Then
                    min = T_min(k)
                    pt = k
                End If
            End If
        Next k
        T_min(i) = min + inp_time(i)
        T_time(i) = T_time(pt) + 1
    Next i
    Print #2, CStr(T_min(N + 1)) & " " & CStr(T_time(N + 1) - 1)
End Sub

```



참 고 문 헌

- ① 비주얼 베이직의 이해, 이건익, 차승윤, 김성락, 남두도서, 2006.
- ② Visual Basic Programming Bible Ver.6.x, 주경민, 박성완, 김민호, 영진닷컴, 1998.
- ③ C로 배우는 프로그래밍 기초, 강환수, 강환일, 학술정보, 2003.
- ④ C 언어 기초, 컴퓨터교육연구소, (주)교학사, 2004.
- ⑤ 이산수학(5판), Judith L. Gersting, 사이텍미디어, 2004.
- ⑥ 이산수학(5판), Kenneth H. Rosen, 인터비젼, 2006.
- ⑦ 이산수학: 논리 · 명제에서 알고리즘까지, 함미옥, 홍영진, 한빛미디어, 2006.
- ⑧ 정보올림피아드 지도 자료, 경상남도교육청, 2004.
- ⑨ 정보올림피아드 및 교원프로그래밍 교육교재, 경상남도교육청, 2006.
- ⑩ 한글비주얼 베이직 6.0 프로그래밍, 이윤수, 정일, 2006.
- ⑪ C로 배우는 알고리즘 1권 이재규, 세화, 2006
- ⑫ Introduction to Algorithms, 문병로 역, 한빛미디어, 2005
- ⑬ Foundation of algorithms, 도경구역, 사이텍미디어, 2006



참고사이트

- ① <http://koi.kado.or.kr/> (한국정보올림피아드 공식사이트)
- ② <http://ACM.uva.es/problemset/> (ACM사이트, 다양한 문제 및 자동채점)
- ③ <http://ace.delos.com/usacogate/> (USACO사이트, 다양한 문제 및 풀이 제공)
- ④ <http://www.dovelet.com/> (루키, 한국정보올림피아드 기출문제 및 자동채점)
- ⑤ <http://www.koi4u.net> (한국정보올림 피아드 학습 사이트)



지/도/위/언

경상남도교육청	교육정보화과	과장 정영규
경상남도교육청	교육정보화과	장학관 남창일
경상남도교육청	교육정보화과	장학사 하현희



집/필/위/언

반성중학교	교사 김성희
경남과학고등학교	교사 정종광
양덕여자중학교	교사 하석봉



검/토/위/언

남해정보산업고등학교	교사 배준호
안골포초등학교	교사 백상준
함양제일고등학교	교사 정화영

2008 정보올림피아드 및 프로그래밍 교재

• 발행일 : 2008년 5월 일

• 발행처 : 경상남도교육청

• 인쇄처 : 세양인쇄사

