한국정보올림피아드 경시부문 전국본선 교재개발연구 김병수 2010년

석 사 학 위 논 문

한국정보올림피아드 경시부문 전국본선 교재 개발 연구
- 초등부를 중심으로 -

A Study of Develping Teaching Material for the KOI final test of Elementary students
- Focusing on Elementary Students -

제주대학교 교육대학원

초등컴퓨터교육전공

김 병 수

2010년 8월

석 사 학 위 논 문

한국정보올림피아드 경시부문 전국본선 교재 개발 연구
- 초등부를 중심으로 -

A Study of Develping Teaching Material for the KOI final test of Elementary students
- Focusing on Elementary Students -

제주대학교 교육대학원

초등컴퓨터교육전공

김 병 수

2010년 8월

한국정보올림피아드 경시부문 전국본선 교재 개발 연구
- 초등부를 중심으로 -

A Study of Develping Teaching Material for the KOI final test of Elementary students
- Focusing on Elementary Students -

지도교수 김 종 훈

이 논문을 교육학 석사학위 논문으로 제출함

제주대학교 교육대학원

초등컴퓨터교육전공

김 병 수

2010년 5월

김병수의

교육학 석사학위 논문을 인준함

심사위원장 김 종 우 인

심사위원 김 종 훈 인

심사위원 박 찬 정 인

제주대학교 교육대학원

2010년 6월

목 차

국문 초록 ····· i

Ⅰ. 서론 ····· 1
1. 연구의 필요성 ····· 1
2. 연구의 방법 ····· 1

Ⅱ. 이론적 배경 ····· 2
1. 한국정보올림피아드 ····· 2
2. 한국정보올림피아드 경시대회 출제 주요 알고리즘 ····· 3
3. 관련연구 ····· 4

Ⅲ. 교재 개발 ····· 7
1. 교재개발의 방향 ····· 7
2. 교재내용 ····· 11

Ⅳ. 적용 및 연구결과의 해석 ····· 88
1. 연구대상 및 연구방법 ····· 88
2. 연구가설 ····· 88
3. 검사도구 및 연구설계 ····· 88
4. 연구결과 및 해석 ····· 90

Ⅴ. 결론 및 제언 ····· 90

참고 문헌 ····· 92
ABSTRACT ····· 94
부 록 ····· 95

표 목 차

<표 Ⅱ-1> 경시대회 소개 ····· 2
<표 Ⅱ-2> IOI Syllabus ····· 6
<표 Ⅳ-1> 연구대상 ····· 88
<표 Ⅳ-2> 채점기준표 ····· 89
<표 Ⅳ-3> 대응표본 상관계수 ····· 90
<표 Ⅳ-4> 대응표본 검정 ····· 90

국문초록

한국정보올림피아드 경시부문 전국본선 교재 개발 연구 - 초등부를 중심으로 -

김 병 수

제주대학교 교육대학원 초등컴퓨터교육전공

지도교수 김 종 훈

초등학생들의 컴퓨터 과학적 사고력, 논리력, 창의력을 기르고자 시행되는 한국정보올림피아드를 현재처럼 사교육에 의존할 수는 없다. 현재 교육현장에서는 이 대회가 단순히 교사가 사교육을 통해 교육받은 어린이를 대회장으로 인솔하는 것으로 공교육의 역할을 다하고 있는 것처럼 생각되어지고 있는 게 현실이다. 이는 학교 현장에서의 컴퓨터 교육이 정보 소비적이며 소프트웨어 활용 교육에 국한되어 있으며 그것을 당연하게 받아들이고 있는 교육과정의 문제라고 할 수 있겠다. 세계의 흐름과 맞추어 우리도 이제 초등교육에서 컴퓨터 과학교육, 로직교육이 필요한 시점이다. 이를 위해 한국정보올림피아드의 문제를 콘텐츠로 교재를 구성하여 학습자의 문제해결 능력 신장과 컴퓨터 과학교육을 하고자 했다. 본 연구는 개발 교재의 완성도를 높이기 위해 J대학교 주관의 초등정보 영재 A반 9명을 실험집단 대상으로 정하고 약 3개월 동안 1회 3시간, 총 7회의 집체교육으로 학습이 이루어졌다. 본 연구의 교재를 통해 컴퓨터 과학교육의 재고와 올바른 방향 설정에 밑거름이 되길 바란다.

핵심어: 한국정보올림피아드, 초등 컴퓨터 과학 교육

I. 서 론

1. 연구의 필요성

교육정보화 및 컴퓨터 교육의 목표는 학습자로 하여금 디지털 정보 지식의 생산자이자 소비자가 되게 하는 것이지 단순한 정보 소비자를 만들자는 것이 아니다. 학생을 정보 생산자가 되게 하려면 학생에게 컴퓨터 관련 기능 기술과 함께 그 기능 기술이 있게 한 컴퓨터 논리, 디지털 논리 더 나아가 디지털 문화논리 까지도 함께 가르쳐야 한다. 디지털 논리를 습득할 때 학생은 컴퓨터식 사고를 할 수 있게 되고, 컴퓨터식 사고를 할 수 있는 상태에서 습득한 기능 기술일 때 그 기능 기술은 디지털 정보 지식, 더 나아가 디지털 매체에 관련한 프로그램을 개발할 수 있는 능력이 된다. 컴퓨터식 사고가 뒷받침되지 않는 컴퓨터 관련 기능 기술은 정보 소비능력 밖에는 되지 못하는 것이다(목영해, 2001).

우리나라에서 초등학생이 컴퓨터식 사고 능력을 개발하기 위한 교육과정은 현재 정보 영재 교육에서 찾아볼 수 있다. 정보 영재 육성을 위한 교육기관의 노력과 많은 관심에도 불구하고, 적합한 교재가 없어 독학으로 공부하기에 많은 무리가 있는 분야가 정보 영재 분야이다. 전 세계적으로 정보 영재에 대한 중요성을 인식하고 있으며 국내에서도 과학 영재를 육성하기 위해 최근 교육기관이 설립되고 있다. 정보 영재 교육은 주로 Basic, C, C++, Java를 가르치는 언어중심의 이벤트성 프로그래밍 교육이나 홈페이지 만들기 등과 같은 툴 위주의 수업으로 이루어지고 있다. 실제 영재 교육은 툴을 잘 다루는 학습에 초점을 맞추기보다는 주어진 문제를 해결하기 위한 창의력을 키울 수 있는 프로그래밍 로직교육이 지속적으로 이루어져야 한다(하성욱 등, 2002).

본 연구에서는 이러한 컴퓨터식 사고 능력을 개발하기 위해서 한국정보올림피아드 경시대회 초등부 전국대회 수준의 로직 문제를 해결해나가기 위한 교재를 개발하는 것을 목적으로 한다.

2. 연구의 방법

J.S.Bruner는 EIS 이론에서 어떤 연령의 아동이든지 간에 그 연령에 맞는 언어로 표현되기만 하면, 어떤 과제이든지 학습이 가능하다고 전제하고 있다. 그의 이론에 따르면 표상은 활동적 표상, 영상적(심상적) 표상, 상징적 표상의 3단계로 발달한다고 한다. 따라서 지식의 전달이 이러한 표상 발달 순서에 맞추어 적절히 제공되기만 하면 어느 연령에서든지 학습될 수 있다는 것이다(조한혁 외, 2002).

따라서 본 연구에서는 컴퓨터 알고리즘을 표, 도식, 그림을 이용하여 영상적(심상적) 표상의 단계로 쉽게 설명할 수 있도록 하고 작은 영역에서 큰 영역으로, 낮은 수준에서 높은 수준으로의 단계적 절차를 가져 하나의 상징적 표상인 프로그래밍 언어를 이용하여 알고리즘을 구현할 수 있는 체계적인 교재 개발을 하는데에 그 목적이 있다.

Ⅱ. 이론적 배경

1. 한국정보올림피아드

한국정보올림피아드(KOI, Korea Olympiad in Informatics)는 행정안전부에서 주최하고 한국정보문화진흥원에서 주관하는 대회이다. 국내 최고의 IT영재들이 참가하여 실력을 겨루는 대회로써 국내 최고의 권위를 가진 대회이며 지역(시도)에서 선발된 학생들이 주어진 문제해결 능력을 겨루는 경시대회와 학생이 스스로 개발한 소프트웨어의 작품성을 평가하는 공모대회로 진행이 된다. 경시부분 우수 입상자에게는 국제정보올림피아드(IOI, International Olympiad in Informatics) 참가 후보자격을 부여하고 있다(한국정보화진흥원, ¶ 1).

경시대회의 주요 내용은 <표 Ⅱ-1>과 같다.

<표 Ⅱ-1> 경시대회 소개

• 개최일시
- 매년 7월 초 ~ 7월 중순
• 출제방향

- 수학적 지식 및 논리적 사고능력을 필요로 하는 알고리즘과 그 구현을 경시하는 문제출제 (IOI 출제경향)

• 경시내용
  - 각 분야별 수준에 맞는 문제해결 및 프로그램 작성 능력 평가

• 사용언어
  - 비주얼베이직, 비주얼 C++

2. 한국정보올림피아드 경시대회 출제 주요 알고리즘

가. 시간복잡도

1) 개념 : 문제의 크기에 따라 걸리는 상대적인 시간

2) 의미 : O-notation(big-Oh notation) : 함수가 들어가 최대에 비례하는 시간이 걸림

나. 정렬 및 순열, 조합

1) 개념 : 정렬 - 임의의 순서로 구성되어 있는 다수의 자료를 일정한 순서로 재배열함(김순근 외, 2005).

순열(Permutation) - 서로 다른 n개의 원소 중에서 r개를 뽑아서 한 줄로 세울 때의 경우의 수를 의미하며 P(n,r)로 나타낸다.

조합(Combination) - n개의 원소를 가지는 집합에서 k개의 부분집합을 고르는 경우의 수를 의미하며 C(n,k)로 나타낸다.

2) 내용 : 선택정렬, 버블정렬, 삽입정렬, 쉘정렬, 퀵정렬, 순열, 조합

다. 그래프 알고리즘

1) 내용 : 너비 우선 탐색, 깊이 우선탐색, 최단 경로 찾기

라. 욕심쟁이 알고리즘

1) 개념 : Greedy 알고리즘이라 하며 여러 선택에 있어서 앞으로의 선택을 고려하지 않고 각각의 선택마다 가장 최선의 선택을 하여 최적의 해를 찾기를

기대하는 알고리즘(이영호 외, 2004).

마. 동적계획법

1) 개념 : 하나의 문제를 작은 부분으로 쪼개어 해결하되 더 큰 부분의 문제해결을 위해서 이미 작은 부분의 문제해결에서 얻은 값을 이용해 나가며 해답을 찾아가는 알고리즘이다.

바. 분할정복

1) 개념 : 작은 사례에 대해서 해답을 바로 구할 수 있으면 바로 구하고 그렇지 않으면 다시 문제를 2개 이상으로 분할하는 알고리즘(경상남도교육청, 2008).

사. 백트래킹

1) 개념 : 검색 공간의 모든 가능한 배치 방법에 대해 어떤 작업을 반복하기 위한 조직정직 방법.(스티븐 스키에나, 2004).

3. 관련연구

가. 한국정보올림피아드 교재 개발의 필요성

본 연구에서는 가장 먼저 초등학생들에게 컴퓨터 알고리즘 교육이 필요한가에 대한 고찰부터 시작된다.

한국의 현행 컴퓨터 교육은 응용 프로그램과 관련한 정보통신기술을 습득하여, 실생활 및 교과학습에 활용한다는 긍정적인 측면도 갖고 있지만, 문제해결력 및 논리적 사고력과 같은 지적 능력의 신장에 한계점을 내재하고 있다는 지적을 받고 있다(김상현, 2002).

이는 지식 정보 사회를 선도할 인재 양성과 개인적 관점에서 급변하는 사회를 살아가는데 필요한 자율적이고 창의적인 현대사회인의 자질 육성에 문제점으로 작용하게 된다(백선련 등, 2008).

이에 대한 근본적인 대안으로 이태욱, 박정호, 백선련(2008)과 CSTA(2003)에서는 학교 현장에서의 컴퓨터 교육에서 컴퓨터의 효과적 사용과 더불어 컴퓨터원리, 알고리즘, 프로그래밍과 같은 컴퓨터 과학 교육을 강조하고 있다.

그러나 현재 초등학교에서 컴퓨터 원리 지도가 어려운 이유 중에 하나가 전공 용어로 구성되어 있는 내용을 초등학생 수준에 맞게 가공하여 구성된 교수설계 및 교재의 부재(임화경, 2005; 백선련 등, 2008)라고 볼 수 있다. 이에 따라 학습자의 수준이 고려된 알고리즘 교재를 개발하는 것이 가장 시급한 문제이다.

본 연구에서는 이러한 알고리즘 교재를 한국정보올림피아드 경시부문 대비용으로 초점을 맞춘데에는 두가지 이유가 있다.

첫째, Papert(1980)와 최성규(2003)의 견해와 마찬가지로 알고리즘이란 문제해결의 절차이며, 프로그래밍이 알고리즘을 기술하는 표현 도구라는 관심에서 보았을 때 알고리즘은 프로그래밍의 전(前) 단계로서 문제해결을 위해 필수적으로 거쳐야 하는 과정이라는 것이다.

즉 개인의 문제해결력을 평가할 수 있고 알고리즘의 학습이 효율적으로 이루어졌는지를 알 수 있는 방법은 정선된 문제를 해결할 수 있는지에 대한 여부를 판단하면 된다. 이러한 이유로 본 연구에서 개발된 교재에서는 한국정보올림피아드 경시부분의 문제를 해결하는 목적으로 알고리즘을 공부하게 된다.

둘째, 한국정보올림피아드 경시부분 대비용 서적의 부재이다. 사실 KOI에서 상위 입상한 학생의 거의 대부분이 사교육을 통하여 컴퓨터 프로그래밍에 대한 지식을 습득하고 있는 현실이다. 그러나 사교육 기관에서 제공되는 교육이 경시결과에 치중하여 올바른 문제 해결방법을 교육하기 보다는 편법을 통하여 점수 일부만이라도 얻을 수 있는 교육이 시행되고 있음이 관찰되고 있어, 유망한 IT 꿈나무들이 초기에 정보과학에 대한 그릇된 인식과 문제 해결에 대한 나쁜 습관에 물들게 되는 실정이다(장직현, 2007). 따라서 한국정보올림피아드 경시부분의 문제를 주요부분으로 하고 이를 해결할 수 있도록 알고리즘 교재를 개발하는 것이다.

한국정보올림피아드는 현재 행정안전부에서 주최하고 한국정보화진흥원에서 주관하는 대회이다. 지역 예·본선은 해당 지역교육청에서 치러지고 전국본선 또한 지역교육청에서 치러진 상위 성적 우수자가 참여하게 된다. 대회는 공교육에서 담당하고 있지만 학생들은 이 대회를 위해 사교육에 의존하고 있는 것이다. 이러한 비판을 위해서라도 누구나 쉽게 이 대회를 준비할 수 있는 교재가 필요한 것이다.

나. 국제정보올림피아드

세계 각국의 중·고등학생을 대상으로 하는 정보과학의 국제적인 경시 대회인 국제정보올림피아드(International Olympiad in Informatics, IOI)는 1989년에 시작되었다. 제1회 IOI는 불가리아의 Pravetz에서 개최되었고 동구권 6개국, 각각 3명의 대표학생들이 참가하였다. 제2회 IOI서부터 각 나라에서 4명의 대표학생들이 참가하게 되었으며 우리나라는 1991년 그리스 Athen에서 개최된 3회 IOI에 참관단을 보냈으며, 1992년 독일 Bonn에서 개최된 4회 IOI부터 현재까지 매년 4명의 대표학생들이 참가하고 있다. 2002년에는 우리나라 용인에서 개최되기도 하였다(장직현, 2007).

IOI 경시가 계속됨에 따라 경시에 참가하는 학생들의 문제 해결 능력이 향상되고 있으며, 이에 따라 출제되는 문제들의 난이도도 계속 높아지고 있다. IOI ISC에서는 IOI가 대학교에 입학하기 전의 학생들을 대상으로 하는 경시인 만큼 참가 학생들이 선행학습에 의한 경쟁 보다는 각자의 지적 능력과 창의력으로 경쟁할 수 있도록 하기 위하여 2006년도 멕시코 Merida에서 개최된 제18회 IOI에서 경시 문제 출제에 대한 수준과 그에 관련된 정보과학의 교육을 위한 교과와 내용을 제시하기 위하여 IOI Syllabus안을 발표하여 앞으로의 IOI 경시 문제 출제에 반영할 것을 추진하고 있다(An official IOI Syllabus, 2006; Tom Verhoeff 등, 2006).

다. IOI Syllabus

다음의 내용은 앞서 설명한 IOI Syllabus 대한 교과 내용들을 요약한 것이다.

<표 Ⅱ-2> IOI Syllabus

수학	수와 기하	정수, 분수와 관련 연산들과 성질들과 평면에서의 점, 선분, 다각형과 거리 및 피타고라스 정리	
	이산 구조	함 수	관계와 집합
		기본 논리학	명제논리, 술어논리, 논리 연산자, 진리표 등
		증명기법	재귀호출의 정의, 연역법, 대우, 부정, 모순들과 증명기법들과 수학적 귀납법 등
		순열과 조합	순열과 조합과 같은 경우의 수 세기
		그래프와 트리	그래프와 트리에 대한 기본적인 성질과 탐색 기법 등
컴	프로그	기초프로그래밍	프로그래밍 언어에 대한 모든 구조와 기능에 대한 지식

퓨터학	래밍의 기본	알고리즘과 문제해결	문제 해결의 전략, 알고리즘의 구현, 디버깅 전력과 알고리즘의 개념 및 성질을 포함
		기초 자료구조	추상 데이터 타입과 여러 기본 자료구조에 대한 지식
		재귀호출	재귀호출의 개념, 구현, 재귀호출에 의한 백트래킹의 구현
	알고리즘과 복잡도	기본적인 알고리즘	알고리즘 명세, 사전 및 사후 조건, 최악의 경우에 대한 복잡도의 계산
		알고리즘 설계전략	간단한 반복 설계, 욕심쟁이 기법, 분할정복, 동적계획법
		기본계산 알고리즘	정수에 관한 간단한 수치계산 알고리즘
		기하 알고리즘	선분에 관한 성질과 교차, 단순다각형에 대한 점의 위치 잡기와 볼록다각형을 구하는 알고리즘
	소프트웨어 공학		작은 규모의 단독 개발 프로젝트의 기본적 기법
	컴퓨터 사용 능력		컴퓨터의 기본 구조와 작동에 대한 이해

Ⅲ. 교재개발

1. 교재개발의 방향

본 연구에서 개발하는 교재는 정보올림피아드 경시대회 본선 수준의 문제를 잘 풀 수 있도록 도움을 줄 수 있는 교재를 만들기 위한 것이다. 장직현(2007)은 현재 KOI와 IOI의 문제들은 컴퓨터 구조의 이해와 높은 사고력, 창의력, 논리력, 알고리즘의 이해를 기본으로 해야만 풀 수 있는 문제로 출제되고 있다고 밝혔다. 즉 본 연구에서 개발된 교재를 통해 높은 점수를 얻을 수 있다는 것은 그만큼 앞서 말한 여러 가지 요소에 대한 능력이 신장되었다고 볼 수 있다.

즉 본 교재는 시중에 출판된 한국정보올림피아드의 교재보다 학습자의 흥미도와 점수 향상면에 초점을 맞추어 개발한다.

2. 교재내용

다음은 본 연구에서 실제 개발한 교재이다.

알고리즘으로 컴퓨터 과학 배워요.

목 차

1 **알고리즘** 9
▪ 성능분석 11 / ▪ 파일입출력 13 / ▪ 국제정보올림피아드 19
2 **배 열** 20
▪ 1차원배열 20 / ▪ 2차원배열 21
3 **스택과 큐** 23
▪ 스택 23 / ▪ 큐 26
4 **정렬과 재귀함수** 30
▪ 버블정렬 31 / ▪ 선택정렬 34 / ▪ 삽입정렬 37 / ▪ 재귀함수 41
5 **그래프** 43
▪ 깊이우선탐색 43 / ▪ 너비우선탐색 48
6 **욕심쟁이알고리즘** 54
7 **백트래킹** 59
8 **분할정복** 65
9 **동적계획법** 72
▪ 동적계획법 맛보기 72 / ▪ 최대 부분 증가 수열 77
▪ 최대 공통 부분수열 80

1 알 고 리 즘

알고리즘이란 무엇일까? 철수와 영희는 '알고리즘'이라는 단어를 국어사전에서 찾아보기로 하였다. 철수는 국어사전의 처음부터 한 쪽씩 넘기며 '알고리즘'이라는 단어가 나올 때까지 찾기로 하였고 영희는 국어사전의 가운데 쪽부터 단어를 찾되 한번은 앞쪽으로 한번은 뒤쪽으로 반복해서 찾으려고 한다.

[그림1_1] 철수와 영희

누가 빨리 찾을 수 있을까? 만약 '알고리즘'이 아닌 '가로수'라는 낱말을 찾는다면 누가 먼저 찾게 될까?

이에 대한 해답은 나중에 애기 하도록 하고 컴퓨터 과학에서는 이 이야기 속의 철수의 방법을 순차검색, 영희의 방법을 이분검색이라고 한다. 이러한 검색의 종류는 지금 알 필요는 없다. 이 이야기에서 말하고자 하는 것은 어떠한 문제에 대하여 전혀 다른 여러개의 접근 방법이 있을 수 있다는 것이다.

어떤 문제에 대하여 어떤 하나의 방법(전략)을 적용하게 될 때 그 문제에 대한 답을 찾는 일반적인 절차 또는 과정이 나온다. 이러한 일반적인 절차 또는 과정을 그 문제에 대한 **알고리즘(Algorithm)**이라고 한다.

영희의 알고리즘의 일반적인 절차는 다음과 같다고 할 수 있다.

단계① 국어사전의 가운데 쪽을 A(=B)쪽이라고 하자(A와 B는 처음에는 같은 쪽을 가리킨다).
단계② A(=B)쪽에서 단어를 찾는다.

단계③ 단어를 찾지 못했다면 A보다 1쪽 앞 부분을 A로 정하고 그곳에서 단어를 찾는다.
단계④ 단어를 찾지 못했다면 B보다 1쪽 뒷 부분을 B로 정하고 그곳에서 단어를 찾는다.
단계⑤ 해당하는 단어를 찾을 때까지 단계③~④를 반복한다.
만약 찾았다면 '성공!'이라고 외치고 멈춘다.

이러한 일반적인 과정은 국어사전에서 '알고리즘'이라는 단어를 찾을 때 다음과 같이 단계적으로 이용된다.

영희가 국어사전에서 '알고리즘'을 찾을 때의 과정

단계① 국어사전의 가운데 쪽은 880쪽이다. A와 B는 880쪽이다.
단계② 880쪽(=A=B)에서 단어 찾어를 찾는다.
단계③ 단어를 찾지 못했다면 879(=A)쪽에서 단어를 찾는다.
단계④ 단어를 찾지 못했다면 881(=B)쪽에서 단어를 찾는다.
단계③ 단어를 찾지 못했다면 878(=A)쪽에서 단어를 찾는다.
단계④ 단어를 찾지 못했다면 882(=B)쪽에서 단어를 찾는다.
⋮
단계④ 단어를 찾지 못했다면 886(=B)쪽에서 단어를 찾는다.
단계⑤ 해당하는 단어를 찾았다. 성공!

위와 같은 일반적인 절차를 알고리즘이라 할 수 있는 것이다.

사실 우리는 이미 이러한 알고리즘을 배웠을 수도 있다. 단지 그것이 알고리즘이란 것을 모르고 있을 뿐이다. 예를 들자면 수학시간에 주어진 문제를 ① 그림을 그려 문제 해결하기 ② 예상하고 확인하며 문제 해결하기 ③ 거꾸로 생각하며 문제 해결하기 등의 방법을 사용해 본적이 있다면 이미 알고리즘을 사용해 본 적이 있는 것이다.

어떠한 문제는 그림을 그려 문제를 해결할 수도 있으며 동시에 거꾸로 생각하며 문제를 해결할 수도 있다. 즉 하나의 문제가 하나의 알고리즘으로만 풀 수 있는 것은 아니다.

그러면 알고리즘은 왜 필요할까?

앞서 예로 든 국어사전에서 단어를 찾을 때에 아무런 전략, 즉 알고리즘이 없이 단어를 찾는다면 시간만 낭비하는 일이다.

다른 예를 들어보기로 하자. 다음의 그림[1_2]과 같이 번호가 적힌 구슬이 6개가 책상 위에 있다고 상상해보자.

[그림1_2] 구슬

이 6개의 구슬을 낮은 번호에서 높은 번호 순서로 나열할 수 있을까? 별로 어렵지 않을 것이다. 다음 [그림1_3]와 같은 순서로 나열할 수 있을 것이다.

[그림1_3] 정렬된 구슬

이렇게 정렬하는 데에는 굳이 알고리즘이 필요하지 않을 수도 있을 것이다. 왜냐하면 누구든 쉽게 위의 구슬을 보자마자 가장 작은 수와 가장 큰 수가 한눈에 보이고 이 들 사이의 수들도 빠른 시간 안에 크기 비교가 되기 때문이다.

하지만 이러한 구슬이 100개가 있다고 생각해보자. 이 구슬들에 적혀있는 숫자가 1~1000 범위 안에 있는 숫자들이라면 어떤 방법으로, 어떤 전략으로 구슬을 번호 순서대로 나열할 수 있을까?

알고리즘이 필요한 이유는 바로 여기에 있다. 이러한 문제를 효과적으로 해결하기 위해서 단계적인 절차를 필요로 하는 것이다. 효과적이라는 의미는 빠른 시간안에 보다 작은 공간을 사용하는 것을 의미한다. 컴퓨터로 이러한 알고리즘을 실행할 때 시간은 프로그램의 실행시간(처리속도)을 의미하며 공간은 메모리의 용량을 말한다.

1 성능 분석

앞서 제시한 국어사전에서 단어찾기 문제를 해결하는 데 자신만의 새로운 알고리즘을 만들어 볼 수도 있을 것이다. 이렇게 알고리즘을 만들고 난 후에는 그 알고리즘이 주어진 문제를 얼마나 정확히 해결해 나가는지를 확인해야 할 필요가 있다. 또한 나의 알고리즘이 컴퓨터 자원을 얼마나 필요로 하는지도 생각해 봐야 한다. 컴퓨터 과학에서 자원이란 흔히 수행시간과 소요 메모리 양을

말한다. 이들을 각각 시간 복잡도와 공간 복잡도라고 한다.

하지만 「한국정보올림피아드 초등부 본선 」문제에서 이들을 모두 고려해 가면서 프로그래밍을 할 필요는 없다. 문제에서 대부분 주어지는 제한 사항은 '실행시간은 1초를 넘을 수 없다.'식의 총 수행시간에 대한 제한 사항이다.

다음은 1000×1000회의 반복문을 실행시킬 때의 총 수행시간을 구하는 프로그램이다.

[소스1_1]

```
1  #include <stdio.h>
2  #include <time.h>
3  int main(){
4          clock_t sTime,fTime;
5          int i,j;
6          sTime=clock();
7          for(i=1;i<=1000;i++)
8                  for(j=1;j<=1000;j++)
9                          printf("");
10         fTime=clock();
11         printf("%.3lf",(double)(fTime-sTime)/CLOCKS_PER_SEC);
12         return 0;
13 }
```

[소스1_1 출력화면]

```
0.109
```

2	메인함수에서 시간과 관련된 변수형 clock_t, 컴퓨터의 클럭 틱 단위로 현재의 시간을 구하는 clock()함수, 초 단위당 클럭의 수를 의미하는 CLOCKS_PER_SEC 변수를 사용할 수 있는 time.h 헤더파일을 포함시킨다.
4	clock_t 변수형은 현재의 클럭 수를 기억시킵니다. 이 변수형으로 sTime, fTime 변수를 생성한다.
6	반복문 실행 이전, 현재의 클럭 수를 sTime에 저장한다.
7-9	1000×1000회의 printf문을 실행한다. 실제 화면에는 아무것도 출력되지 않는다.

10	반복문 실행 이후, 현재의 클럭 수를 fTime에 저장한다.
11	fTime-sTime으로 반복문 실행 동안의 클럭 수를 계산하고 이를 초당 클럭 수인 CLOCKS_PER_SEC로 나눈다면 실행시간을 구할 수 있다. 예를 들어 fTime=109, sTime=0, CLOCKS_PER_SEC=1000 일 경우 프로그램의 실행시간은 0.109초로 계산되어 출력된다.

컴퓨터는 클럭의 수가 1 증가할 때마다 한번의 명령을 수행한다고 말할 수 있다. 1초에 1000번 클럭 수가 증가하는 컴퓨터에서 내가 만든 프로그램을 실행하는 동안에 클럭 수가 500번 증가했다면 0.5초만에 프로그램이 실행되었다는 의미이다.

[소스1_1]과 같이 실제 자신의 알고리즘이 수행되는 시간을 고려하며 문제를 풀어야 하는 이유는 어떠한 문제에 대한 해답을 찾을 때 중요한 것은 정답보다는 효율적인 알고리즘을 개발하는 것이기 때문이다.

다시 처음의 철수와 영희 이야기 문제로 넘어가 보자. 철수와 영희는 서로 다른 전략을 가지고 단어 찾기를 하고 있다. 철수와 영희는 모두 '알고리즘'과 '가로수' 단어를 찾겠지만 찾을 때까지의 시간은 경우에 따라 다를 것이다. '알고리즘'이란 단어는 영희가 먼저 찾을 수 있을 것이고 '가로수'라는 단어는 철수가 먼저 찾을 가능성이 크다.

모든 문제를 가장 빠른 시간안에 풀 수 있는 알고리즘은 없다. 하나의 문제에 가장 적절한 알고리즘을 개발해 나가는 것이 우리가 컴퓨터 과학을 공부하는 이유라 할 수 있겠다.

2 파일 입출력

한국정보올림피아드의 초등부 본선 문제에서 주로 데이터의 입력과 출력은 화면이 아닌 파일로 이루어진다. 파일 입출력의 장점은 대량의 데이터를 한번에 입력할 수 있고 많은 출력 데이터를 통해 오류를 쉽게 발견할 수 있으며 평균 수행시간을 구할 수 있어 알고리즘의 효율성을 따질 수 있다는 것이다. 또한 자동채점을 할 수 있다는 장점이 있다.

파일 입출력의 간단한 예제를 살펴보자. 다음은 INPUT.TXT 파일에 있는

내용을 이용하여 OUTPUT.TXT 파일을 생성하는 소스이다. 현재 INPUT.TXT 파일에는 [그림1_4]와 같은 내용이 들어있다. 실제 문제에는 다음과 같이 입력형식과 출력형식의 조건이 주어진다.

<입력형식> 입력파일 INPUT.TXT의 첫째 줄에는 학생 수가 입력된다. 다음 줄부터는 학생들의 성적들이 입력된다.

INPUT.TXT
3
97
79
84

[그림1_4]

<출력형식> 출력파일 OUTPUT.TXT의 첫째 줄에는 총점을, 둘째 줄에는 평균을 소수점 이하 2자리로 출력한다.

[소스1_2]

```
1  #include <stdio.h>
2  int main(){
3          int i,no,score,sum=0;
4          FILE *in=fopen("INPUT.TXT","r");
5          FILE *out=fopen("OUTPUT.TXT","w");
6          fscanf(in,"%d",&no);
7          for(i=0;i<no;i++){
8                  fscanf(in,"%d",&score);
9                  sum+=score;
10         }
11         fprintf(out,"%d\n%.2lf",sum,(double)sum/no);
12         fclose(in);
13         fclose(out);
14         return 0;
15 }
```

4-5	FILE은 파일을 다루는 변수를 생성하게 할 수 있는 구조체이다. 이미 <stdio.h> 헤더파일에 정의되어 있다. FILE로 생성된 변수는 포인터형이다. fopen()은 해당하는 파일 구조체에 대한 포인터를 반환해주는 함수이다. 자세한 설명은 아래에서 하기로 하고 여기에서 알고 넘어갈 사항은 in은 INPUT.TXT 파일이 내용을 읽어(r) 들이는 데 사용되고, out은 OUTPUT.TXT 파일에 출력 내용을 쓰는(w) 데 사용된다는 것이다.
6	파일포인터 in에서 읽어 들인 정수(%d)를 변수 no에 저장한다.
7-10	scanf와 마찬가지로 파일에서 입력을 받아들이는 fscanf 함수는 공백(Space Bar)이나 강제개행(Enter↵)이 있을 때까지의 입력된 수를 하나의 수로 인식한다. 즉 여기에서도 fscanf 함수는 자동으로 다음 줄의 수를 읽어 들이게 된다. 파일포인터 in에서 읽어 들인 정수(%d)를 변수 score에 저장하고 이를 변수 sum에 누적하여 합한다. 이를 no의 횟수만큼 반복 실행한다.
11	printf는 화면에 데이터를 출력시키는 것이라면 fprintf는 파일에 출력을 할 때 사용된다. 파일포인터 out이 가리키는 OUTPUT.TXT에 변수 sum과 sum/no의 값을 출력한다. 파일에 출력한다는 의미에 파일에 결과데이터가 해당 파일에 입력된다는 의미이다.
12-13	연결된 파일포인터와의 연결을 닫는 단계이다. 닫지 않아도 프로그램은 실행이 되지만 효율적인 메모리 관리를 위해 fclose() 함수로 닫는 것이 권장된다.

위의 [소스1_2] 프로그램을 실행할 때 해당 소스 파일과 INPUT.TXT 파일은 같은 폴더에 있어야 한다. 실행 후 생성되는 OUTPUT.TXT의 내용은 [그림 1_5]와 같다.

OUTPUT.TXT
260 86.67

[그림1_5]

간단한 예제로 대략적인 파일 입, 출력 기능에 대해 알아 보았다. 실제 파일과 관련된 함수는 다양하며 매우 유용하므로 차근차근 배워 나가보도록 하겠다.

⁂ 파일 포인터 연결과 해제 : fopen(), fclose()

fopen() 함수는 파일포인터를 반환하는 함수이다. fopen() 함수에 어떤 파일을, 어떤 모드(mode)로 접근할지를 정해주어야 한다.

```
fopen ( "파일명" , "접근모드" ) ;
```

접근 모드에 관한 옵션은 다음을 참고한다.

모 드	의 미
r	read(읽다)의 의미로 내용을 읽기 위해 파일을 연다.
w	write(쓰다)의 의미로 내용을 쓰기 위해 파일을 생성한다. 기존에 파일이 이미 있어도 모든 내용을 지우고 내용을 새로 쓴다.
a	append(덧붙이다)의 의미로 파일이 없을 경우는 파일을 생성하고 기존에 해당파일이 있다면 내용을 지우지 않고 덧붙여서 쓴다.

이 외에도 많은 모드들이 있지만 생략하겠다. 이 세 가지의 모드만 알고 있어도 문제를 해결하는데 어려움은 없다. 만약 ABC.TXT 파일의 내용을 읽기 위해서는 파일포인터 fp를 생성하고 싶다면 다음처럼 하면 된다.

```
FILE *fp = fopen ("ABC.TXT", "r");
```

fclose() 함수는 파일포인터와의 연결을 닫는 함수이다. 앞서 말했듯이 메모리의 효율적인 사용을 위해 닫아주는 것을 권장한다. 사용방법은 다음과 같다.

```
fclose ( 파일포인터 ) ;
```

다음은 ABC.TXT를 쓰기 위해 fopen() 함수로 파일포인터 fp에 접근 권한을 받고 이를 닫는 예제이다.

```
FILE *fp = fopen ("ABC.TXT", "w");
```

```
...
fclose(fp);
```

⁂ 입력함수 : fgetc(), fgets(), fscanf()

파일을 열어 파일 안의 내용을 입력받고자 할 때 다양한 함수를 사용할 수 있다. 다음은 이러한 입력함수의 종류이다.

모 드	의 미
fgetc(파일포인터)	파일포인터가 가리키는 파일에서 문자형(unsigned char형)으로 내용을 읽어들인 후, 그것을 정수로 반환한다.

```
#include <stdio.h>
int main(){
        FILE *fp = fopen("ABC.TXT","r");
        printf("%d",fgetc(fp));
        fclose(fp);
        return 0;
}
```

fgets(배열명,입력길이, 파일포인터)	파일포인터가 가리키는 파일에서 배열로 내용을 읽어들인다. fgets()함수는 해당하는 파일을 줄 단위로 읽어들인다.

```
#include <stdio.h>
int main(){
        char text[100];
        FILE *fp = fopen("ABC.TXT","r");
        fgets(text,100,fp);
        printf("%s",text);
        fclose(fp);
        return 0;
}
```

fscanf(파일포인터,데이터형식,배열명)	scanf()와 같은 기능이며 가장 유용하여 널리 쓰인다. 파일포인터가 가리키는 파일에서 내용을 배열에 지정한 데이터형식으로 읽어들인다.

```
#include <stdio.h>
int main(){
        int score;
        FILE *fp = fopen("ABC.TXT","r");
        fscanf(fp,"%d",&score);
        printf("%d",score);
        fclose(fp);
        return 0;
}
```

실제 본선 문제에서는 fscanf() 함수를 사용할 수 있다면 크게 문제될 것은 없기 때문에 fscanf() 함수를 유심히 봐둘 필요가 있다.

▟ 출력함수 : fputc(), fputs(), fprintf()

출력함수는 변수에 내용을 입력받고 파일 안에 쓰기 위해 사용한다. 이 또한 다양한 함수가 있는데 다음을 참고한다.

모 드	의 미
fputc(변수,파일포인터)	변수의 내용을 파일포인터가 가리키는 파일에 출력한다.

```
#include <stdio.h>
int main(){
        char ch='a';
        FILE *fp=fopen("ABC.TXT","w");
        fputc(ch,fp);
        fclose(fp);
        return 0;
}
```

fputs(배열명, 파일포인터)	배열에 들어있는 문자열을 파일포인터가 가리키는 파일에 출력한다.

```
#include <stdio.h>
int main(){
        char ch[10]="Hello";
        FILE *fp=fopen("ABC.TXT","w");
```

```
        fputs(ch,fp);
        fclose(fp);
        return 0;
}
```

fprintf(파일포인터, 데이터형식,배열명)	printf()와 같은 기능이되 화면이 아닌 파일로 출력된다. 파일포인터가 가리키는 파일에 배열에 저장된 문자열을 지정한 데이터형식으로 출력한다.

```
#include <stdio.h>
int main(){
        int no=32;
        FILE *fp=fopen("ABC.TXT","w");
        fprintf(fp,"%d",no);
        fclose(fp);
        return 0;
}
```

정답을 출력하게 될 때 숫자로 출력할 때가 많기 때문에 문자와 숫자의 데이터형식에 구애받지 않는 fprintf() 함수를 사용할 것을 권장한다.

3 국제정보올림피아드

이번에는 국제정보올림피아드(International Olympiad in Informatics, IOI - http://www.ioinformatics.org)와 관련된 이야기를 해보겠다. 물론 수준의 차이는 있겠지만 국제정보올림피아드의 경향과 문제 유형을 알아두면 한국정보올림피아드(Korea Olympiad in Informatics, KOI - http://www.nia.or.kr/koi/)의 본선 문제 유형을 파악하는 실마리가 될 수 있다. 국제정보올림피아드는 유네스코에서 주최하는 국제 과학 올림피아드에 포함되는 정기적인 컴퓨터 과학 대회로, 1989년에 시작되었다. 2010년 올해로 22회째 대회를 치르고 있으며 2002년에는 한국에서 개최되기도 했다.

각 국가에서 보통 2명의 지도자와 4명의 선수로 구성되어 대회에 참가하며 두 번의 경시(2 Round)를 통하여 총 8문제에 대한 점수의 합으로 결과를 얻게 된다. 전체 참가자의 1/12이 금메달, 1/6이 은메달, 1/4이 동메달을 거머쥘 수 있다.

2 배 열

본선 문제에서는 자주 등장하는 문제 중 하나의 유형이 너비, 면적 또는 공간에 대한 활용 문제이다. 이때 사용하는 데이터형식이 배열이다. 배열은 같은 데이터형의 자료를 같은 크기의 공간에 넣을 수 고 순서가 정해져 있어서 쉽게 접근하여 불러다 사용할 수 있다는 장점이 있다.

1 1차원 배열

변수 5개에 숫자 1~5를 차례대로 입력시키기 위해 아래와 같은 코딩을 했다.

```
#include <stdio.h>
int main(){
        int a,b,c,d,e;
        a=1; b=2; c=3; d=4; e=5;
        return 0;
}
```

만약 위와 같은 방법으로 변수 100개에 숫자 1에서부터 100까지 입력시키고 싶다면 100개의 변수명을 모두 적고 일일이 입력하고자 하는 숫자를 넣어줘야 할 것이다. 이러한 불편함 때문에 배열이 생겼다고 생각하면 쉽다.

```
#include <stdio.h>
int main(){
        int a[100],i;
        for(i=0;i<100;i++)
                a[i]=i+1;
        return 0;
}
```

배열 a

a[0]	a[1]	a[2]	a[3]	a[4]	a[…]	a[98]	a[99]
1	2	3	4	5	…	99	100

위와 같은 방법으로 쉽게 정수형 배열 a에 100개의 공간을 만들고 그 곳 모두에 숫자 1에서부터 100까지 입력시킬 수 있다. 여기서 유의해야 할 점은 배열의 순서는 0에서부터 시작한다는 점이다. 즉 위의 소스에서는 마지막 배열이

a[99]라는 점이다. 이러한 점을 자주 헷갈려하는 학생들을 위해 다음과 같은 방법도 좋을 것이다.

```
#include <stdio.h>
int main(){
        int a[101],i;
        for(i=1;i<=100;i++)
                a[i]=i;
        return 0;
}
```

배열 a

a[0]	a[1]	a[2]	a[3]	a[4]	a[··]	a[99]	a[100]
NULL	1	2	3	4	···	99	100

위의 소스는 배열 a에 101개의 공간을 생성하고 a[0]은 사용하지 않는 식으로 코딩되었다. 물론 a[0]을 사용하지 않는 점에서는 비효율적이라고 할 수 있지만 이런 코딩이 쉽게 읽힌다면 사용해도 무방할 것이다.

2 2차원 배열

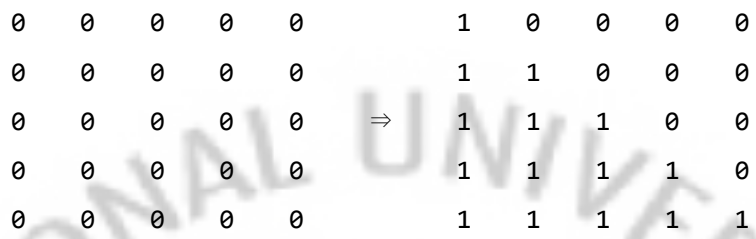
면적 또는 길찾기 등과 관련된 문제에서 2차원 배열은 자주 사용된다. 아래의 [그림2_1]은 2차원 배열을 표현한 것이다.

```
int a[5][5];
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]
a[4][0]	a[4][1]	a[4][2]	a[4][3]	a[4][4]

[그림2_1] 2차원 배열

이를 응용하여 [그림2_2]와 같은 숫자 0이 입력된 2차원 배열 a에 삼각형의 모양으로 숫자 1을 입력하여 보자.



[그림2_2] 초기화시 [그림2_3] 1 입력후

[소스2_1]

```
1  #include <stdio.h>
2  int main(){
3          int a[5][5]={0,};
4          int x,y;
5          for(x=0;x<5;x++)
6                  for(y=0;y<=x;y++)
7                          a[x][y]=1;
8          for(x=0;x<5;x++){
9                  for(y=0;y<5;y++){
10                         printf("%d",a[x][y]);
11                 }
12                 printf("\n");
13         }
14         return 0;
15 }
```

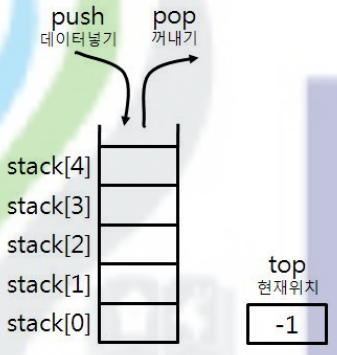
3	5×5 크기의 2차원 배열 a를 정의하고 모든 배열요소에 숫자 0을 입력한다.
5-7	삼각형 모양으로 숫자 1을 입력한다.
8-13	출력을 5개씩 할 때마다 뉴라인(\n)으로 줄넘김을 한다.

3 스택과 큐

스택과 큐를 이미 알고 있겠지만 다시 확인하여 넘어가자. 스택과 큐는 이후에 공부하게 될 여러 가지 알고리즘에서 기본적으로 사용되는 구조이기 때문에 확실히 알아둘 필요가 있다.

1 스택

스택은 가장 나중에 삽입된 데이터가 제일 먼저 꺼내어지는 후입선출(LIFO : Last In First Out) 구조이다. 스택은 주로 배열로 구현하게 된다.



[그림3_1] 스택

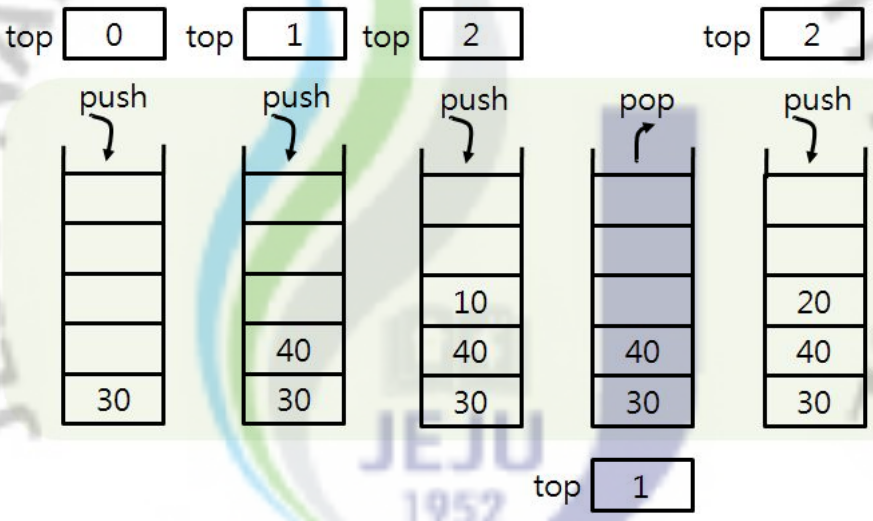
스택을 배열로 구현할 때의 일반적인 알고리즘은 다음과 같다.

단계① 스택의 크기를 정하고 가장 상위에 있는 데이터를 가리키는 변수 top에 -1을 저장한다.
단계② 사용자가 push, pop, view 중 하나의 메뉴를 선택한다.
단계③ push를 선택한 경우 top이 스택의 크기보다 작다면 top을 1 증가시키고 데이터를 삽입한다. (클 경우 "스택이 찼음" 출력)
단계④ pop을 선택한 경우 top이 -1보다 큰 경우 top에서 1 감소시킨다. (top이 -1일 경우는 "데이터 없음" 출력)
단계⑤ view를 선택한 경우는 stack[top]에서부터 내림차순으로 출력한다. (top이 -1일 경우는 출력이 되지 않는다.)
단계⑥ 단계②부터 반복 시행

예를 들어 [그림3_1]과 같은 스택의 구조에서 다음과 같은 명령을 했다고 한다면 현재 스택에는 어떤 자료가 어떤 구조로 남아 있을지 생각해 보아야 한다.

```
push(30);
push(40);
push(10);
 pop();
push(20);
```

위의 명령은 [그림3_2]와 같은 과정으로 동작될 것이다.



[그림3_2] 스택의 동작

이러한 기본적 지식을 이용하여 스택을 구현해보자.

[소스3_1]

```
1  #include <stdio.h>
2  int main(){
3          int stack[5], top=-1, choice, i;
4          while(1){
```

```
5                printf("1.push 2.pop 3.view : ");
6                scanf("%d",&choice);
7                if(choice==1){
8                        if(top<4){
9                                printf("push number : ");
10                               scanf("%d",&stack[++top]);
11                       }else{
12                               printf ("Stack is full\n");
13                       }
14               }else if(choice==2){
15                       if(top>-1) top--;
16                       else printf ("Stack is empty\n");
17               }else if(choice==3){
18                       for(i=top;i>-1;i--)
19                               printf("%d\n",stack[i]);
20               }
21       }
22       return 0;
}
```

3	스택으로 사용할 정수형 배열 stack[5]를 생성하고, 현재 데이터가 저장된 가장 상위 부분을 가리키는 변수 top에는 -1을 저장한다.
5	메뉴(push, pop, view)를 출력하여 선택할 수 있게 한다.
8-13	top의 값이 4보다 작다면 stack[] 배열에 값이 들어갈 공간이 있다는 의미이다. 그렇지 않다면 “Stack is full"을 출력한다. scanf("%d",&stack[++top]);는 입력받은 값을 직접 stack[top]에 입력하되 먼저 top의 값을 1 증가시킨 후 입력될 수 있도록 stack[++top]으로 사용한다.
15-16	top의 값이 -1보다 크다면 stack[] 배열에 적어도 한 개의 값이 저장되어 있다는 의미이므로 pop(데이터 꺼내기)이 가능하다. 여기에서는 pop을 top에서 1을 감소시키는 것으로 한다. 만약 top의 값이 -1 이하라면 “Stack is empty"를 출력한다.
18	스택의 자료를 stack[top]~stack[0]으로 출력한다. 이는 화면에 보

여질때 마치 스택과 같이 가장 나중에 입력된 데이터가 가장 상위에 출력되어 보여지기 위함이다.

여기에서 알고 넘어가야할 부분은 pop의 기능이 실제 데이터를 삭제하는 것이 아니라 top의 값에서 1을 감소시키는 것이다. 이는 이후에 view 메뉴에서 스택의 자료 중 stack[0]~stack[top]의 값만 출력시키면 되기 때문이다. 이는 이미 int stack[5]; 이라는 명령으로 5개의 정수형 변수가 들어갈 공간을 메모리에 확보해 놓았기 때문에 굳이 그 안의 내용을 지우고 다시 쓰는 작업이 불필요하다. 좀 더 효율적인 메모리 관리를 위해서는 스택을 연결 리스트(Linked List)를 활용하여 구현할 수도 있다.

2 큐

큐는 가장 먼저 삽입된 데이터가 제일 먼저 삭제되는 선입선출(FIFO : First In First Out) 구조이다. 큐 또한 배열로 구현할 수 있다.

[그림3_3] 큐

[그림3_3]에서 front(큐의 머리)는 데이터가 삭제되어 나갈 부분을 가리키는 변수이며 rear(큐의 꼬리)는 데이터가 삽입될 공간을 가리키는 변수이다. 스택에서 현재 가장 상위 부분을 가리키는 변수 top이 큐에서는 변수 rear라고 생각하면 쉽다. front는 데이터가 빠져나갈 부분을 가리키며 현재의 값에서 1을 증가시키고 해당 부분에서 데이터가 삭제되는 것이다.

큐를 배열로 구현할 때의 일반적인 알고리즘은 다음과 같다.

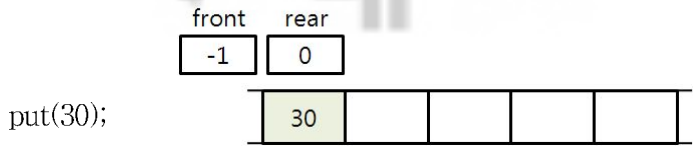
단계① 큐의 크기를 정하고 큐의 머리, 큐의 꼬리를 가리키는 front, rear 변수에 각각 -1 을 저장하여 생성한다.
단계② 사용자가 put, get, view 중 하나의 메뉴를 선택한다.
단계③ put을 선택한 경우 rear의 값이 큐의 크기보다 작다면 rear를 1 증가시키고 데이터를 삽입한다. (클 경우 "큐가 찼음" 출력)
단계④ get을 선택한 경우 front의 값이 rear값보다 작다면 front를 1 증가시킨다. (front의 값이 rear보다 작지 않다면 "데이터 없음" 출력)
단계⑤ view를 선택한 경우 stack[front+1]~stack[rear]까지 출력한다.
단계⑥ 단계②부터 반복 시행

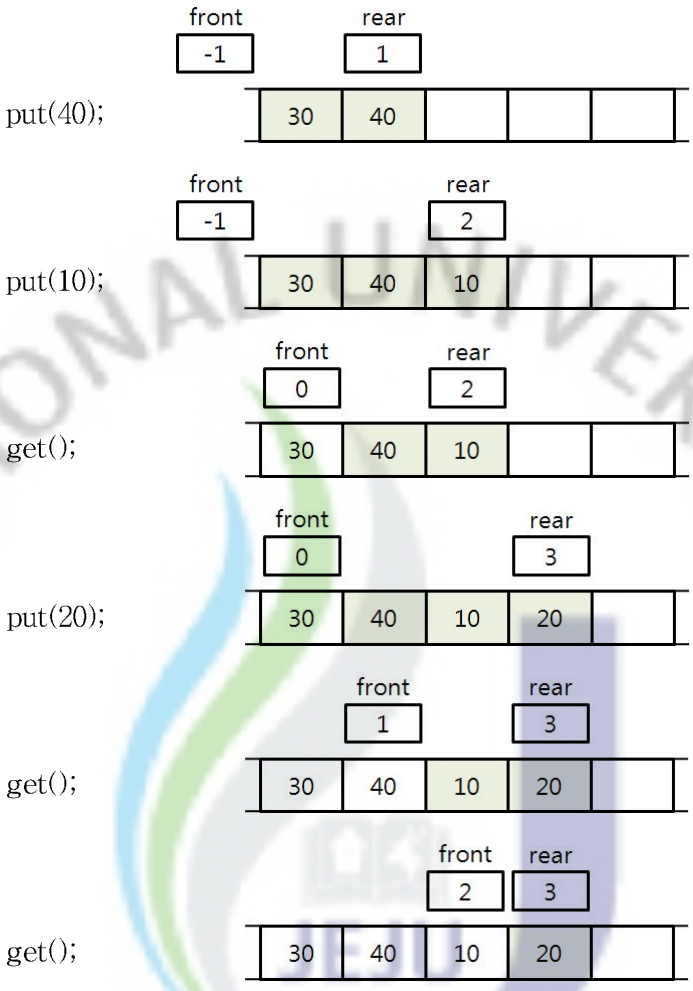
위의 알고리즘에서 단계①에서 처음부터 front의 값을 0으로 정의하고 단계⑤에서 view를 선택한 경우 stack[front]~stack[rear]까지 출력할 수도 있다. 이는 프로그래머가 이해가 쉬운 쪽으로 이용하면 되겠다.

예를 들어 [그림3_3]과 같은 큐의 구조에서 다음과 같은 명령을 했다고 한다면 현재 큐에는 어떤 자료가 어떤 구조로 남아 있을지, 또 front와 rear의 값은 무엇일지 생각해보자.

```
put(30);
put(40);
put(10);
 get();
put(20);
 get();
 get();
```

위의 명령은 [그림3_4]와 같은 과정으로 동작될 것이다. 이 때 색칠된 배열은 view를 선택할 경우 출력되는 부분이다.





[그림3_4] 큐의 동작

이러한 큐의 구조를 실제로 구현해보자.

[소스3_2]

```
1  #include <stdio.h>
2  int main(){
3          int que[5], front=-1, rear=-1, choice, i;
4          while(1){
```

```
5               printf("1.put 2.get 3.view : ");
6               scanf("%d",&choice);
7               if(choice==1){
8                       if(rear<4){
9                               printf("put number : ");
10                              scanf("%d",&que[++rear]);
11                      }else{
12                              printf("Que is full\n");
13                      }
14              }else if(choice==2){
15                      if(front<rear) front++;
16                      else printf("Que is empty\n");
17              }else if(choice==3){
18                      for(i=front+1;i<=rear;i++)
19                              printf("%d\t",que[i]);
20                      printf("\n");
21              }
22      }
23      return 0;
24  }
```

3	큐로 사용할 정수형 배열 que[5]를 생성하고 큐의 머리 front, 큐의 꼬리 rear에 -1을 저장한다.
5	메뉴(push, pop, view)를 출력하여 선택할 수 있게 한다.
8-13	rear의 값이 4보다 작다면 que[] 배열에 값이 들어갈 공간이 있다는 의미이다. 그렇지 않다면 “Qeu is full"을 출력한다. scanf("%d",&que[++rear]);는 입력받은 값을 직접 que[rear]에 입력하되 먼저 rear의 값을 1 증가시킨 후 입력될 수 있도록 que[++rear]으로 사용한다.
15-16	front의 값이 rear보다 작다면 que[] 배열에 적어도 한 개의 값이 저장되어 있다는 의미이므로 get(데이터 꺼내기)이 가능하다. 여기에서는 get을 front에서 1을 증가시키는 것으로 한다. 만약 front의 값이 rear보다 작지 않다면 “Que is empty"를 출력한다.

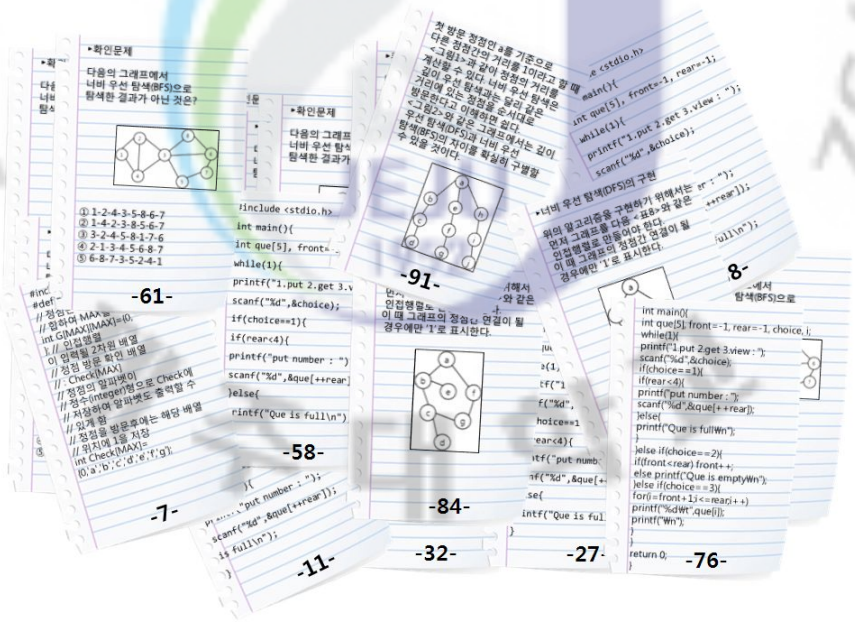
18-20	큐의 자료를 que[front+1]~que[rear]로 출력한다.

스택에서와 마찬가지로 큐의 구현에서도 get의 경우 실제 데이터를 지우면서 빼내는 것이 아니라 단지 front의 값을 1증가시키고 출력시킬때 이미 삭제된 부분은 출력하지 않는 것으로 동작된다. 이러한 메모리의 비효율성 때문에 다양한 구조의 큐가 존재한다. 하지만 한국정보올림피아드에서는 메모리의 효율성보다는 프로그램의 성능에 더욱 초점을 맞추므로 다른 구조의 큐에 대한 설명은 생략하겠다.

4 정렬과 재귀함수

프로그래밍을 공부하기 위해서 프린터기에서 100쪽의 유인물을 뽑아냈다. 유인물을 옮기는 과정에서 실수로 100쪽의 유인물이 뒤죽박죽이 되어 버렸다. 다행히 유인물의 하단에 쪽수는 적혀져 있기 때문에 정리는 가능할 것 같다.

어떤 방법으로 정리해야 할까?



이렇게 순서없이 구성된 자료를 일정한 순서로 다시 재배열하는 것을 정렬이라고 한다. 정렬의 방법은 다양하므로 차근차근 하나씩 익혀나가도록 하자.

1 버블정렬

버블정렬은 보글보글하며 유연하게 움직이는 비누거품처럼 버블정렬도 동작되기 때문에 붙여진 이름이다. 버블정렬에서 핵심 알고리즘은 바로 옆 자리의 데이터와 비교하여 자리를 바꾼다는 것이다.

먼저 [그림4_1]과 같이 초기 데이터가 있다고 생각하자. 그리고 사용자는 오름차순으로 이 데이터를 정렬하고자 한다. 이 때 버블 정렬은 다음의 단계와 같이 동작된다.

[그림4_1] 초기 데이터

1회전 [단계1]

9는 5보다 크므로 자리를 바꾼다.

1회전 [단계2]

9는 7보다 크므로 자리를 바꾼다.

1회전 [단계3]

9는 3보다 크므로 자리를 바꾼다.

1회전 [단계4]

9는 1보다 크므로 자리를 바꾼다.

1회전 [최종데이터]

초기 데이터에서 제일 앞에 있던 자료 9이 [단계1]~[단계4]를 거쳐 제일 뒤로 움직이게 되었다. 이러한 과정을 4회 더 하게 되면 최종적으로 정렬이 끝나게 된다. 앞서 했던 1회전처럼 2~4회전을 할 경우 각 회전의 최종 데이터는 다음과 같다.

구 분	2 회 전	3 회 전	4 회 전
단계1	⑤ ⑦ ③ ① ⑨	③ ⑤ ① ⑦ ⑨	① ③ ⑤ ⑦ ⑨
단계2	⑤ ③ ⑦ ① ⑨	③ ① ⑤ ⑦ ⑨	① ③ ⑤ ⑦ ⑨
단계3	⑤ ③ ① ⑦ ⑨	③ ① ⑤ ⑦ ⑨	① ③ ⑤ ⑦ ⑨
단계4	⑤ ③ ① ⑦ ⑨	③ ① ⑤ ⑦ ⑨	① ③ ⑤ ⑦ ⑨

버블정렬을 구현할 때의 알고리즘은 다음과 같다.

> 단계① 크기가 n인 배열 data를 생성하고 배열안에 데이터가 삽입된다.
> 단계② data[0]~data[n-2]번째 배열을 기준배열로 오른쪽 배열(data[1]~data[n-1])의 자료와 비교하여 값이 더 크면(오름차순) 교환한다.
> 단계③ 단계②를 n-1번 반복 실행한다.

실제 구현은 다음과 같다.

[소스4_1]

```
1  #include <stdio.h>
2  int main(){
3          int data[5]={9,5,7,3,1}, i, j, temp;
4          for(i=0;i<4;i++){
5                  for(j=0;j<4;j++){
6                          if(data[j]>data[j+1]){
7                                  temp=data[j];
8                                  data[j]=data[j+1];
9                                  data[j+1]=temp;
10                         }
11                 }
12         }
13         for(i=0;i<5;i++) printf("%d\t",data[i]);
14         return 0;
15 }
```

3	정수형 배열 data[5]를 생성하고 9,5,7,3,1을 순서대로 삽입한다.
4	for(i=0;i<4;i++)는 for문 안에 있는 명령을 4회 반복 실행하겠다는 의미이다. data 배열의 크기는 5이므로 버블정렬에서는 총 4회전의 정렬이 필요하다.
5	for(j=0;j<4;j++)는 기준배열을 지정해주기 위한 것이다. j가 1씩 증가하되 data[3]까지만 기준배열로 지정하고 for문은 종료된다.
6-10	data[j]가 data[j+1]보다 크다면 교환한다.

```
13      정렬된 배열의 값을 순서대로 출력한다.
```

다시 처음의 이야기로 돌아가서 생각해 본다면 버블정렬은 그다지 매력있는 정렬 방법은 아니다. 왜냐하면 100쪽의 종이를 나열한 상태에서 제일 처음에서부터 99번째의 종이까지 오른편의 종이와 쪽번호를 비교해야 할 뿐만 아니라 이러한 과정을 총 99회 해야하는 것이다.

이러한 버블정렬을 좀 더 개선할 수도 있다. 현재의 회전에서 이전 회전에서 마지막으로 배열값이 교환된 위치까지만 비교하게 하고 만약 이전 회전에서 아무런 교환이 없다면 정렬이 모두 끝난 상태이므로 더 이상의 실행을 중단하게 한다. 이러한 개선된 버블정렬을 스스로 구현해 보는 것도 재미있는 일일 것이다.

2 선택정렬

선택정렬은 말 그대로 선택하여 정렬하게 된다. 예를 들어 무작위로 된 번호가 나열되어 있을 경우 이들을 오름차순으로 정렬하고 싶다고 하자. 이 번호 중 제일 작은 수를 선택해서 첫째 자리에 놓고 그 다음 작은 수를 선택하여 두 번째 자리에 놓는 방법이다.

[그림4_2] 초기 데이터

[그림4_2]에서 5개의 초기 데이터 중에서 사람은 쉽게 무엇이 가장 작은 수인지 쉽게 알 수 있고 빠르게 정렬할 수도 있다. 하지만 컴퓨터는 그렇지 못하다. 프로그래밍의 성격상 한번에 하나의 계산으로만 비교하여 찾게 된다. 즉 가장 작은 수를 찾는 과정은 다음과 같은 단계와 같다.

1회전 [단계1]

9는 5보다 크므로 값을 바꾼다.

1회전 [단계2]

5는 7보다 작으므로 값을 바꾸지 않는다.

1회전 [단계3]

5는 3보다 크므로 값을 바꾼다.

1회전 [단계4]

3은 1보다 크므로 값을 바꾼다.

1회전 [최종데이터]

1회전에서 계속적으로 찾으려고 하는 것은 가장 작은 값이다. 1회전의 각 단

계마다 제일 앞자리의 값보다 현재 비교하는 부분의 배열값이 작으면 교환하고자 한다. 이러한 방법으로 2회전에서는 두 번째 자리에 두 번째 작은 수를 찾아서 갖다 놓을 것이고 3회전, 4회전에서도 같은 이것은 반복된다. 그러면 굳이 5회전을 하지 않아도 4회전 마지막에 5번째 자리에는 당연히 가장 큰 값이 위치하게 될 것이다.

2~4회전의 과정은 다음과 같다.

구 분	2 회 전	3 회 전	4 회 전
단계1	① ⑦ ⑨ ⑤ ③	① ③ ⑦ ⑨ ⑤	① ③ ⑤ ⑦ ⑨
단계2	① ⑤ ⑨ ⑦ ③	① ③ ⑤ ⑨ ⑦	-
단계3	① ③ ⑨ ⑦ ⑤	-	-
단계4	-	-	-

선택정렬에서는 매 회전마다 단계가 하나씩 줄어들고 있음을 볼 수 있다. 또한 N번째 회전에서는 N번째 자리의 수가 결정된다.

선택정렬을 구현할 때의 알고리즘은 다음과 같다.

```
단계① 크기가 n인 배열 data를 생성하고 배열안에 데이터가 삽입된다.
단계② data[i]와 data[i+1]~data[n-1]를 비교하며 가장 작은 수를 찾아낸다
      (초기 i의 값은 0으로 한다).
단계③ 단계②를 i를 1씩 증가시키며 n-1번 반복 실행한다.
```

실제 구현은 다음과 같다.

[소스4_2]

```
1  #include <stdio.h>
2  int main(){
3          int data[5]={9,5,7,3,1}, i, j, temp;
4          for(i=0;i<4;i++){
5                  for(j=i+1;j<5;j++){
6                          if(data[i]>data[j]){
```

```
7                                   temp=data[i];
8                                   data[i]=data[j];
9                                   data[j]=temp;
10                              }
11                      }
12              }
13              for(i=0;i<5;i++) printf("%d\t",data[i]);
14              return 0;
15      }
```

3	정수형 배열 data[5]를 생성하고 9,5,7,3,1을 순서대로 삽입한다.
4	for(i=0;i<4;i++)는 for문 안에 있는 명령을 4회 반복 실행하겠다는 의미이다. data 배열의 크기는 5이므로 선택정렬에서는 총 4회전의 정렬이 필요하다. 여기서 주목해야 할 점은 변수 i가 단지 반복만을 사용된 것이 아니라 6번째 줄에서 비교시 기준배열의 위치로도 사용된다는 것이다.
5	for(j=i+1;j<5;j++)는 비교배열의 범위를 지정해주기 위한 것이다. j는 data[i+1]에서 data[4]끝까지 증가되며 반복 실행된다.
6-10	data[i]가 data[j]보다 크다면 서로의 값을 교환한다.
13	정렬된 배열의 값을 순서대로 출력한다.

이러한 선택정렬은 간단하다는 장점이 있고 데이터의 량이 적을 때 유용하다.

3 삽입정렬

삽입정렬은 말 그대로 순서없이 나열된 자료에서 순서에 맞게 자료들을 중간 중간에 삽입해 나가며 정렬하는 방법이다.

[그림4_3] 초기 데이터

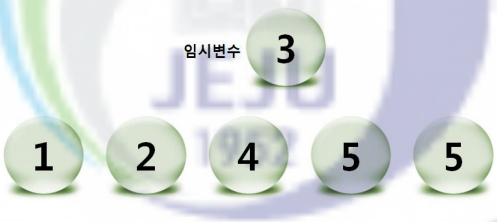
삽입정렬은 쉽게 사람의 정렬과도 비슷하다. [그림4_3]을 보면 쉽게 사람들은 3번 구슬을 2번과 4번 구슬 사이에 끼워 넣을 것이다. 그러면 쉽게 정렬을 할 수 있다. 하지만 컴퓨터는 이렇게 직관적인 사고를 할 수 없고 일정한 규칙을 정해주어야 한다.

기준데이터 : 1번, 2번, 4번, 5번구슬의 경우는 생략

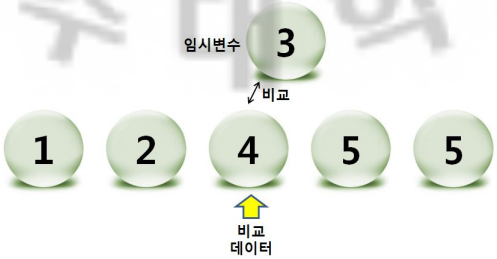
기준데이터 : 3번 구슬

[1단계]

기준데이터의 값을 임시변수에 저장시켜둔다. 그리고 이 임시변수를 활용하여 값을 비교한게 된다. 먼저 기준데이터의 왼쪽으로 나아가며 값을 비교하되 임시변수의 값보다 크면 아래의 그림과 같이 하나씩 값을 오른쪽에 덮어쓴다. 이는 나중에 현재 임시활용 변수의 값을 끼워넣을 자리를 만들기 위한 것이다.



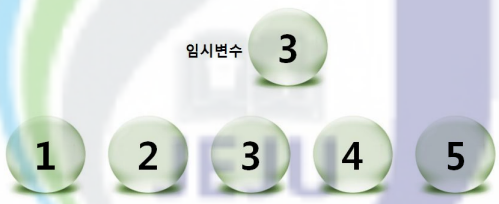
[2단계]



임시변수의 값 3과 4를 비교하면 4가 크므로 4번 구슬 오른쪽 구슬에 4를 덮어쓴다.

[3단계]

임시변수의 값 3과 2를 비교하면 임시변수의 값 3이 더 크므로 2번 구슬 오른쪽에 값을 덮어쓰고 회전을 중단한다.



이렇게 해서 최종데이터를 얻게 되었다. 이러한 방법으로 삽입정렬은 진행된다. 삽입정렬을 구현할 때의 알고리즘은 다음과 같다.

단계① 크기가 n인 배열 data를 생성하고 배열 안에 데이터가 삽입된다.
단계② data[i]의 값을 변수 value에 저장해둔다(초기 i의 값은 0이다).
단계③ value와 data[i-1]~data[0]의 값을 순서대로 비교하되 value값보다 크면 (오름차순으로 정렬) 한자리 앞 배열의 값을 덮어쓴다.
단계④ i값이 n-1이 될 때까지 을 1을 증가시키며 단계②부터 반복 실행한다.

실제 구현은 다음과 같다.

[소스4_3]

```
1  #include <stdio.h>
2  int main(){
3          int data[5]={5,1,3,2,4};
4          int i,j,k,value;
5          for(i=0;i<5;i++){
6                  value=data[i];
7                  for(j=i;j>0;j--){
8                          if(value<data[j-1]) data[j]=data[j-1];
9                          else break;
10                 }
11                 data[j]=value;
12         }
13         for(k=0;k<5;k++){
14                 printf("%d ",data[k]);
15         }
16         return 0;
17 }
```

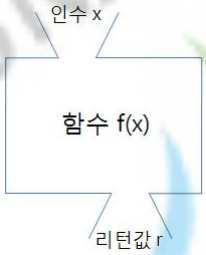
3	정수형 배열 data[5]를 생성하고 5,1,3,2,4를 순서대로 삽입한다.
5-6	i값을 0~4까지 1씩 증가 시키며 반복 실행한다. 이 때마다 해당 data[i]의 값을 value에 저장시킨다.
7-10	해당 data[i]보다 왼쪽 순서대로 value의 값과 비교하여 크다면 data[j]에 data[j-1]의 값을 덮어 쓰게 된다. 그렇지 않으면 for문을 빠져나가게 break 명령을 쓴다.
11	7번 줄에서 for문 실행이 다 되거나 9번 줄에서 break 문으로 for문을 빠져나오면 data[j]에는 현재 value가 들어갈 자리를 찾게 된다.
13-15	현재 배열 data의 값을 순서대로 출력한다.

삽입정렬은 쉽게 사용된다기 보다는 거의 정렬이 되어 있는 데이터에서 사용될 때 효율적이다.

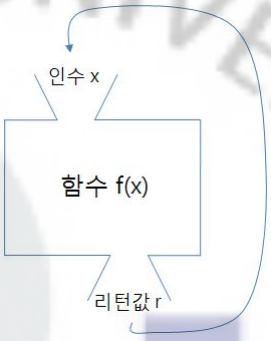
4 재귀함수

재귀함수란 '재귀'라는 단어를 해석하면 쉽게 이해를 할 수 있다. 재귀는 '다시 돌아간다'라는 뜻으로 어떠한 함수가 자기 자신을 다시 호출할 때 이 함수를 재귀함수라고 하고 이러한 호출을 재귀호출이라고 한다.

일반적으로 리턴값이 있는 함수를 [그림4_4]로 재귀함수를 [그림4_5]로 생각할 수 있다.

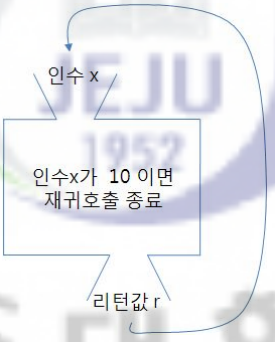


[그림4_4] 함수



[그림4_5] 재귀함수

위 그림의 재귀함수를 보면 마치 네버엔딩 스토리처럼 멈추지 않고 계속 돌아갈 것이다. 그렇기 때문에 재귀호출을 멈출 수 있는 조건이 필요할 것이다.



[그림4_5] 재귀함수의 종료 조건

다음의 [소스4_4]는 1부터 10까지의 수를 출력하는 재귀함수이다.

[소스4_4]

```
1  #include <stdio.h>
2  int f(int x){
3          if(x>=10) return 0;
4          x++;
5          printf("%d\n",x);
6          return f(x);
7  }
8  int main(){
9          f(0);
10         return 0;
11 }
```

2	int값을 리턴하는 함수 f를 정의한다. 인수를 x로 받는다.
3	x가 10보다 크거나 같으면 종료(return 0)된다.
4-5	현재 x값이 1증가되고 이를 출력한다.
6	리턴값으로 자기 자신의 함수 f를 호출한다.
8-11	메인함수에서는 f함수에 인수 1을 넘겨주며 호출한다.

이러한 재귀함수는 앞서 설명한 스택의 구조로 진행된다. 이를 쉽게 이해하기 위해서는 다음 함수를 보면 도움이 될 것이다.

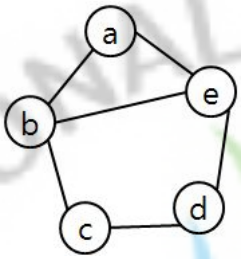
[소스4_5]

```
1  int f(int x){
2          if(x>=10) return 0;
3          x++;
4          f(x);
5          printf("%d\n",x);
6          return 0;
7  }
```

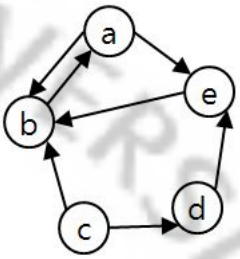
위 소스는 10부터 1을 출력하는 함수 f이다. 위 함수는 스택에 f(1)→f(2)→…→f(10)로 입력되어 출력은 10부터 1까지 출력되게 된다.

5 그래프

그래프에서 중요한 두 가지 개념은 정점과 간선이다. 다음 [그림6_1]을 보자.



[그림5_1] 무방향 그래프



[그림5_2] 방향 그래프

[그림5_1]에서 ⓐ, ⓑ, ⓒ, ⓓ, ⓔ를 정점이라고 하고 이를 잇는 선을 간선이라고 한다. 간선은 (시작점, 도착점)으로 나타낸다. 이와 같은 그래프에서는 (시작점, 도착점)이 (도착점, 시작점)과 같다. 예를 들어 (a, b)와 (b, a)는 같은 간선이다. 이러한 그래프를 '무방향 그래프'라고 한다. 위 그래프에서는 (a, b), (a, e), (b, c), (b, e), (c, d), (d, e)가 6개의 간선이라고 할 수 있다.

[그림5_2]는 '방향 그래프'이다. 이 그래프에서는 (a, b)와 (b, a)는 같은 간선이라고 할 수 없다. 이 그래프에서는 총 7개의 간선이 있고 (a, b), (b, a), (a, e), (c, b), (c, d), (d, e), (e, b)가 그것이다.

그래프에서 모든 정점을 방문하는 것을 그래프의 탐색이라 하는데, 깊이 우선 탐색(DFS, depth first search)과 너비 우선 탐색(BFS, breadth first search)이 있다.

1 깊이 우선 탐색(DFS)

깊이 우선 탐색은 주로 스택을 이용하여 구현한다. 직접 스택을 만들어 사용하기도 하며 재귀호출을 이용하여 시스템 스택을 이용하기도 한다. 흔히 깊이

우선 탐색을 백트래킹과 혼용하여 사용하기도 한다.

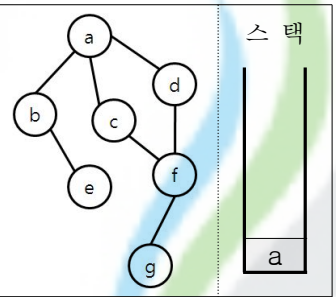
다음의 [표5_1]은 깊이 우선 탐색의 원리를 순서대로 설명한 것이다.

[표5_1] DFS 알고리즘

① 시작 정점을 스택에 넣는다.
② 스택에서 정점 하나를 꺼내어 출력하고, 방문 정점을 확인한다.
③ 방문한 정점에서 직접 연결된 정점들 중 방문하지 않은 정점들을 스택에 차례로 넣는다.(여기에서는 인접한 정점이 복수 개일 경우에는 알파벳 내림차순으로 스택에 저장)
④ 모든 정점을 방문할 때까지 ②~③번 과정을 반복한다.

1단계

시작정점을 스택에 넣는다.

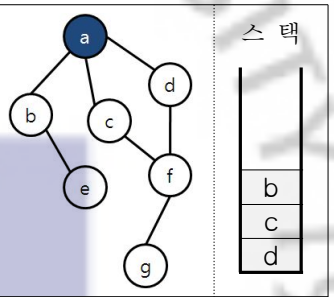


방문정점	a	b	c	d	e	f	g
확인배열	0	0	0	0	0	0	0

화면출력:

2단계

스택에서 자료(a)를 꺼내어 출력, a와 연결된 d, c, b를 스택에 입력

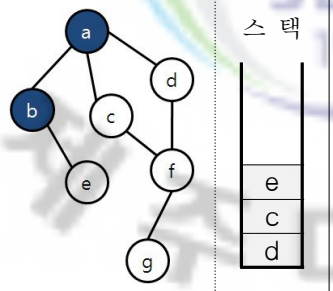


방문정점	a	b	c	d	e	f	g
확인배열	1	0	0	0	0	0	0

화면출력: a

3단계

스택에서 자료(b)를 꺼내어 출력, b와 인접한 e를 스택에 입력, 이미 방문한 a는 입력하지 않음

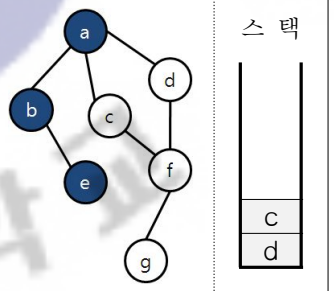


방문정점	a	b	c	d	e	f	g
확인배열	1	1	0	0	0	0	0

화면출력: b

4단계

스택에서 자료(e)를 꺼내어 출력, e와 인접한 정점이 없으므로 스택에 아무것도 입력하지 않음

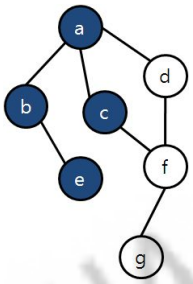


방문정점	a	b	c	d	e	f	g
확인배열	1	1	0	0	1	0	0

화면출력: e

5단계

스택에서 자료(c)를 꺼내어 출력, c와 인접한 f를 스택에 입력, 이미 방문한 a는 입력하지 않음



스 택

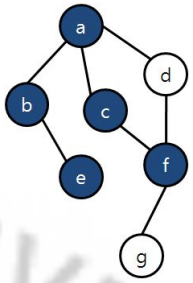
f
d

방문정점	a	b	c	d	e	f	g
확인배열	1	1	1	0	1	0	0

화면출력: c

6단계

스택에서 자료(f)를 꺼내어 출력, f와인접한g d를 스택에 입력, 이미 방문한c는 입력하지 않음



스 택

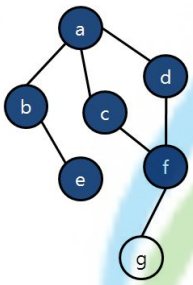
d
g
d

방문정점	a	b	c	d	e	f	g
확인배열	1	1	1	0	1	1	0

화면출력: f

7단계

스택에서 자료(d)를 꺼내어 출력, d와 인접한 a, f는 이미 방문했기 때문에 입력하지 않음



스 택

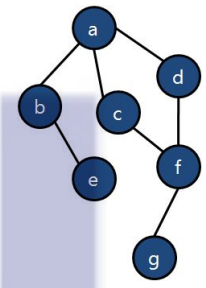
g
d

방문정점	a	b	c	d	e	f	g
확인배열	1	1	1	1	1	1	0

화면출력: d

8단계

스택에서 자료(g)를 꺼내어 출력, g와 인접한 f는 이미 방문했기 때문에 입력하지 않음



스 택

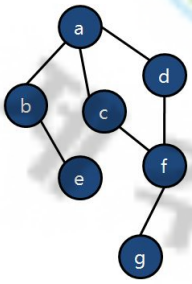
d

방문정점	a	b	c	d	e	f	g
확인배열	1	1	1	1	1	1	1

화면출력: g

9단계

스택에서 자료(d)를 꺼내어 이미 방문한 정점임을 확인하고 출력하지 않음

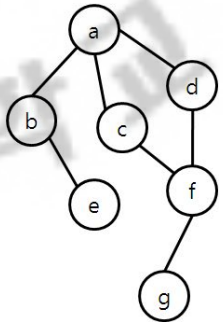


스 택

방문정점	a	b	c	d	e	f	g
확인배열	1	1	1	1	1	1	1

화면출력:

이러한 과정으로 깊이 우선 탐색의 진행 과정은 다음과 같다.

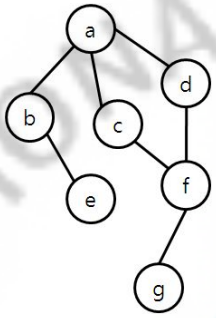


a - b - e - c - f - d - g

▟ 깊이 우선 탐색(DFS)의 구현

이를 C언어로 코딩하기 위해서는 먼저 그래프를 다음 [표5_2]와 같은 인접 행렬로 만들어야 한다. 이 때 그래프의 정점간 연결이 될 경우에만 '1'로 표시한다.

[표5_2] 인접행렬



	a	b	c	d	e	f	g
a	0	1	1	1	0	0	0
b	1	0	0	0	1	0	0
c	1	0	0	0	0	1	0
d	1	0	0	0	0	1	0
e	0	1	0	0	0	0	0
f	0	0	1	1	0	0	1
g	0	0	0	0	0	1	0

이를 2차원 배열로 나타내면 다음 [표5_3]와 같다.

[표5_3] 2차원 배열

	1	2	3	4	5	6	7
1	0	1	1	1	0	0	0
2	1	0	0	0	1	0	0
3	1	0	0	0	0	1	0
4	1	0	0	0	0	1	0
5	0	1	0	0	0	0	0
6	0	0	1	1	0	0	1
7	0	0	0	0	0	1	0

이를 바탕으로 다음의 [소스5_1]과 같은 소스를 만들 수 있다. 이 소스는 재귀호출을 이용하였다. 재귀호출은 스택 구조를 이용하게 되므로 굳이 스택을 구현할 필요는 없다.

[소스5_1]

```
1  #include <stdio.h>
2  #define MAX 8  // 정점은 7개, 0번째 배열을 합하여 MAX를 8로 지정
3  int G[MAX][MAX]={0,};  // 인접행렬이 입력될 2차원 배열
```

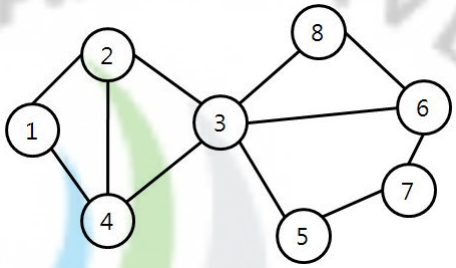
```
4  // 정점 방문 확인 배열 : Check[MAX]
5  // 정점의 알파벳이 정수(integer)형으로 Check에 저장하여 알파벳도
6  // 출력할 수 있게 함
7  // 정점을 방문후에는 해당 배열 위치에 1을 저장
8  int Check[MAX]={0,'a','b','c','d','e','f','g'};
9  void DFS(int v){
10         int i;
11         if(Check[v]==1) return;  // 이미 방문하였으면 리턴
12         printf("%c\t",Check[v]);  // 방문한 배열의 알파벳 출력
13         Check[v]=1;  // 방문한 정점의 알파벳 출력 후 1을 저장
14         // 방문한 정점과 연결된 정점 중 현재 방문하지 않은 정점을
15         // 찾아 재귀호출
16         for(i=1;i<MAX;i++) if(G[v][i] && Check[i]!=1) DFS(i);
17 }
18
19 void main(){
20         int i,j;
21         // 사용자가 인접행렬 입력(올림피아드에서는 파일 입출력 사용함)
22         for(i=1;i<MAX;i++)
23            for(j=1;j<MAX;j++)
24                scanf("%d",&G[i][j]);
25         // 첫 방문 정점을 1(즉, Check[1]='a', 정점 a)를 방문하면서
26         //DFS 함수 호출
27         DFS(1);
28 }
```

3	인접행렬이 들어갈 2차원 배열 G[MAX][MAX]이 생성된다.
8	Check[MAX]에 0,'a','b','c','d','e','f','g'를 순서대로 넣는다. 이 때 문자들은 아스키값으로 변환되어 들어간다.
9-13	확인배열 Check에서 이미 방문한 곳을 제외하고 알파벳을 출력하고 방문했다는 의미로 '1'을 입력한다.
16	방문한 정점과 연결된 정점 중 현재 방문하지 않은 정점을 찾아 재귀호출한다.

22-24	2차원 배열 G를 사용자가 입력하여 값을 얻을 수 있도록 한다. 실제 정보올림피아드에서는 이를 파일 입력을 통해 이루어진다. 이러한 과정은 다음 장에서 자세히 다루어 질 것이다.
27	DFS 함수에 인자 1을 넣어 호출한다.

▶ 확인문제

다음의 그래프에서 깊이 우선 탐색(DFS)으로 탐색한 결과가 아닌 것은?



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

2 너비 우선 탐색(BFS)

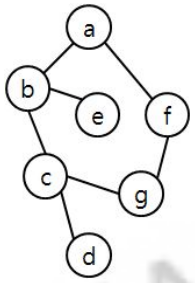
너비 우선 탐색은 깊이 우선 탐색과는 달리 큐를 이용한다. 너비 우선 탐색은 현 정점에서 제일 가까운 정점들을 먼저 탐색하고 점점 더 멀리 탐색하되 같은 기회를 모든 경우의 수에 분배해 나가며 가장 빠른 탐색을 할 수 있는 것에 초점을 맞춘다. 다음의 [표5_4]는 너비 우선 탐색의 원리를 순서대로 설명한 것이다.

[표5_4] BFS 알고리즘

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 정의).
④ 모든 정점이 방문체크 되지 않았으면 ②번 과정으로 간다.

1단계

시작정점이 큐에 들어간다



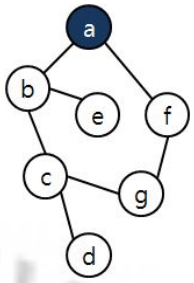
큐: a

방문정점	a	b	c	d	e	f	g
확인배열	0	0	0	0	0	0	0

화면출력:

2단계

a를 출력하고 배열에 1입력, a와 인접한 정점 b, f를 큐에 입력



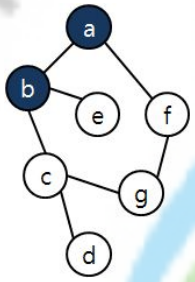
큐: f, b

방문정점	a	b	c	d	e	f	g
확인배열	1	0	0	0	0	0	0

화면출력: a

3단계

b를 출력하고 배열에 1입력, b와 인접한 정점 c, e를 큐에 입력



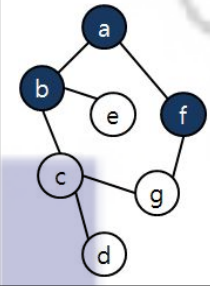
큐: e, c, f

방문정점	a	b	c	d	e	f	g
확인배열	1	1	0	0	0	0	0

화면출력: b

4단계

f를 출력하고 배열에 1입력, f와 인접한 정점 g를 큐에 입력



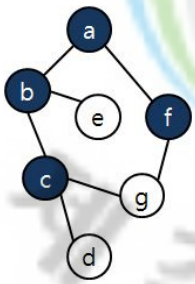
큐: g, e, c

방문정점	a	b	c	d	e	f	g
확인배열	1	1	0	0	0	1	0

화면출력: f

5단계

c를 출력하고 배열에 1입력, c와 인접한 정점 d, g를 큐에 입력



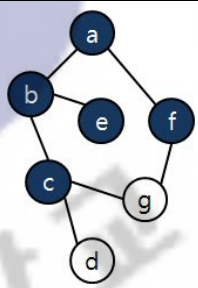
큐: g, d, g, e

방문정점	a	b	c	d	e	f	g
확인배열	1	1	1	0	0	1	0

화면출력: c

6단계

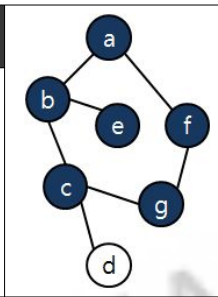
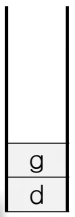
e를 출력하고 배열에 1입력, e와 인접한 정점 b는 이미 방문하여 큐에 입력하지 않음

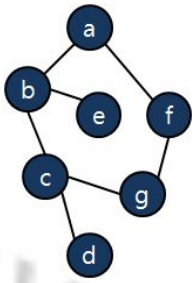
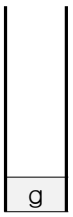


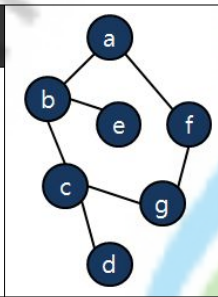
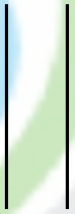
큐: g, d, g

방문정점	a	b	c	d	e	f	g
확인배열	1	1	1	0	1	1	0

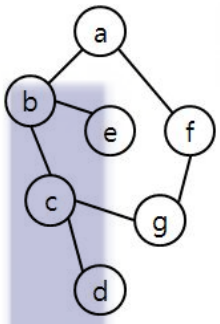
화면출력: e

<table>
<tr><td>7단계</td><td> 
</td><td>큐
 
</td></tr>
<tr><td>g를 출력하고 배열에 1입력, g와 인접한 정점 c, f는 이미 방문하여 큐에 입력하지 않음</td><td colspan="2"></td></tr>
<tr><td>방문정점
확인배열</td><td colspan="2">a b c d e f g
1 1 1 0 1 1 1</td></tr>
<tr><td>화면출력</td><td colspan="2">g</td></tr>
</table>

<table>
<tr><td>8단계</td><td> 
</td><td>큐
 
</td></tr>
<tr><td>d를 출력하고 배열에 1입력, d와 인접한 정점 c, g는 이미 방문하여 큐에 입력하지 않음</td><td colspan="2"></td></tr>
<tr><td>방문정점
확인배열</td><td colspan="2">a b c d e f g
1 1 1 1 1 1 1</td></tr>
<tr><td>화면출력</td><td colspan="2">d</td></tr>
</table>

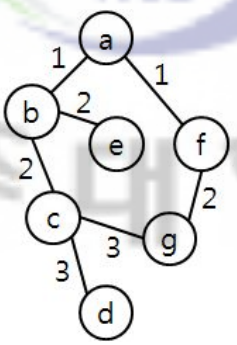
<table>
<tr><td>9단계</td><td> 
</td><td>큐
  </td></tr>
<tr><td>g를 큐에서 꺼냈지만 이미 방문했으므로 출력하지 않음</td><td colspan="2"></td></tr>
<tr><td>방문정점
확인배열</td><td colspan="2">a b c d e f g
1 1 1 1 1 1 1</td></tr>
<tr><td>화면출력</td><td colspan="2"></td></tr>
</table>

이러한 과정으로 깊이 우선 탐색의 진행 과정은 다음과 같다.



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

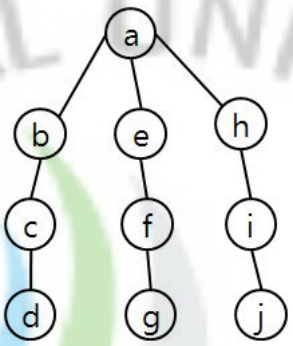
이러한 너비 우선 탐색은 다음과 같은 [그림5_3]로 알고리즘을 쉽게 이해할 수도 있다.



[그림5_3] BFS 탐색

첫 방문 정점인 a를 기준으로 다른 정점간의 거리를 1이라고 할 때 위 그림과 같이 정점의 거리를 계산할 수 있다. 너비 우선 탐색은 깊이 우선 탐색과는 달리 같은 거리에 있는 정점을 순서대로 방문한다고 이해하면 쉽다.

[그림5_4]와 같은 그래프에서는 깊이 우선 탐색(DFS)과 너비 우선 탐색(BFS)의 차이를 확실히 구별할 수 있을 것이다.



[DFS 탐색 순서]
a b c d e f g h i j

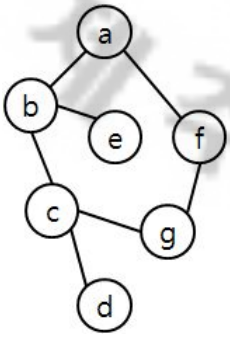
[BFS 탐색 순서]
a b e h c f i d g j

[그림5_4] DFS와 BFS 탐색의 비교

너비 우선 탐색(DFS)의 구현

위의 알고리즘을 구현하기 위해서는 먼저 그래프를 다음 [표6_5]와 같은 인접행렬로 만들어야 한다. 이 때 그래프의 정점간 연결이 될 경우에만 '1'로 표시한다.

[표5_5] 인접행렬



	a	b	c	d	e	f	g
a	0	1	0	0	0	1	0
b	1	0	1	0	1	0	0
c	0	1	0	1	0	0	1
d	0	0	1	0	0	0	0
e	0	1	0	0	0	0	0
f	1	0	0	0	0	0	1
g	0	0	1	0	0	1	0

이를 2차원 배열로 나타내면 다음 [표5_6]과 같다.

[표5_6] 그래프 G[v][i]

	1	2	3	4	5	6	7
1	0	1	0	0	0	1	0
2	1	0	1	0	1	0	0
3	0	1	0	1	0	0	1
4	0	0	1	0	0	0	0
5	0	1	0	0	0	0	0
6	1	0	0	0	0	0	1
7	0	0	1	0	0	1	0

[소스5_2]

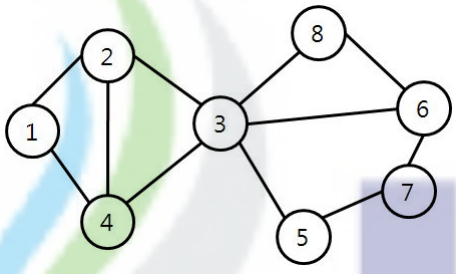
```
1  #include <stdio.h>
2  #define MAX 50 // 최대 정점이 50개까지 가능하도록 함
3  int G[MAX][MAX]={0,}; // 인접행렬
4  int Check[MAX]={0,}; // 정점 방문 확인 배열
5  int Que[(MAX+1)*MAX/2]={0,}; // 큐로 사용될 배열,
6                               // 최악의 경우(완전 그래프) 대비
7  int rows; // 몇줄 몇행인지 변수로 받음
8  int rear=-1,front=-1; // 큐의 입력, 출력 위치를 저장하는 변수
9
10 // 큐에 데이터 삽입시
11 void quePush(int vertexNo){
12         front++;  // 함수가 호출될 때마다 출력 위치를 저장하는
13                   // front 변수가 1씩 증가
14         Que[front]=vertexNo;  // 큐에 현재의 정점 번호 저장
15                               // 예. a=1, b=2, c=3
16 }
17
18 // 큐에서 데이터 출력시
19 int quePop(){
20         int vertexNo;
21         if(rear==front){  // rear와 front가 같은 경우는
```

```
22                              //큐에 자료가 없는 경우
23                      return 0;
24              }else{
25                      rear++;
26                      vertexNo=Que[rear];
27                      if(Check[vertexNo]!=1){  //큐에서 꺼낸 정점이
28                                               // 미방문 정점일 경우
29                              // 방문 정점의 알파벳 출력
30                              printf("%c",Check[vertexNo]);
31                              //방문한 배열 위치에 1 저장
32                              Check[vertexNo]=1;
33                      }
34                      return vertexNo;  // 큐에서 꺼내진 정점을 반환
35              }
36      }
37
38      void BFS(int vertexNo){
39              int j;
40              // 최초 함수 호출시 입력된 정점을 큐에 삽입
41              if(front==-1) quePush(vertexNo);
42              // 큐에서 정점을 하나가 꺼내질 경우
43              while(vertexNo=quePop()){
44                      for(j=1;j<=rows;j++)
45                      // 꺼내진 정점과 연결된 정점 중 미방문 정점을 큐에 삽입
46                              if(G[vertexNo][j]&&Check[j]!=1)
47                                      quePush(j);
48              }
49      }
50
51      void main(){
52              int i,j;
53              // 정점 개수 입력
54              scanf("%d",&rows);
55              // 인접행렬 입력
```

```
56          for(i=1;i<=rows;i++)
57                  for(j=1;j<=rows;j++) scanf("%d",&G[i][j]);
58          // 입력받은 변수만큼의 Check[]배열 생성
59          for(i=1;i<=rows;i++) Check[i]='a'+i-1;
60          // 첫 방문 정점을 1('a')로 지정하여 BFS 함수 호출
61      BFS(1);
    }
```

▶ 확인문제

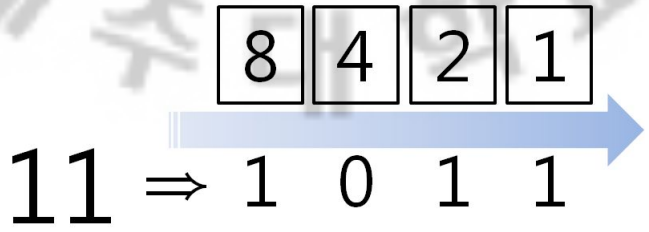
다음의 그래프에서 너비 우선 탐색(BFS)으로 탐색한 결과가 아닌 것은?



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

6 욕심쟁이 알고리즘

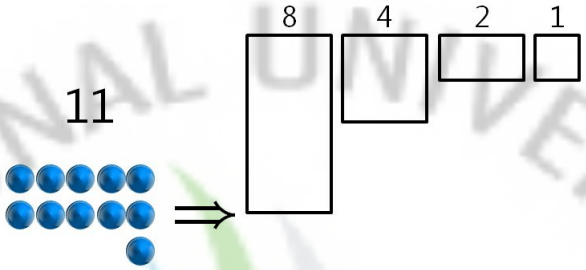
10진수 숫자 10은 2진수로는 10102 이다. 10진수를 2진수로 바꾸는 방법 중 '8421' 코드라는 방법이 있다. 2진수의 각 자리수에 8, 4, 2, 1을 곱하여 10진수로 나타내는 방법이다.



[그림6_1] 8421코드 방법으로 2진수로 변환

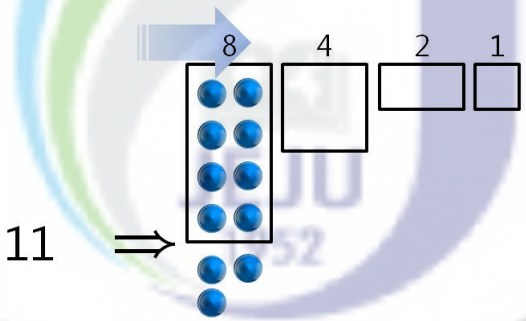
이 방법을 쉽게 이해하기 위해서는 8, 4, 2, 1을 한번씩만 사용하여 이 들의 합을 11로 만들면 된다. 가장 왼쪽 8에서 오른쪽 순서로 계산하게 되는데 먼저 11에서 8을 뺀다. 나머지 3을 2와 1에 넣으면 된다.

[초기상태]

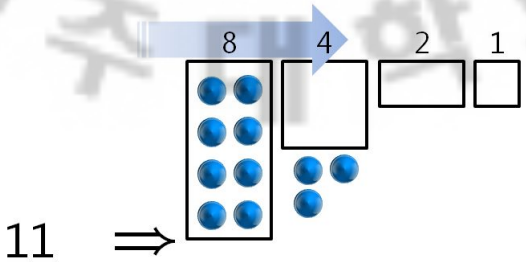


매 단계마다 가장 최고의 이익을 챙기려는 '욕심쟁이 알고리즘'을 이 방법에서 생각해보자. 11개의 구슬이 있을 경우 왼쪽으로 한 단계씩 구슬을 옮길 수 있다고 했을 때 각 단계에서 숫자 8, 4, 2, 1 바구니에 넣을 수 있다면 넣어보자. 단, 각 바구니에 구슬을 넣을 때는 바구니를 가득 채울 수 있어야 한다.

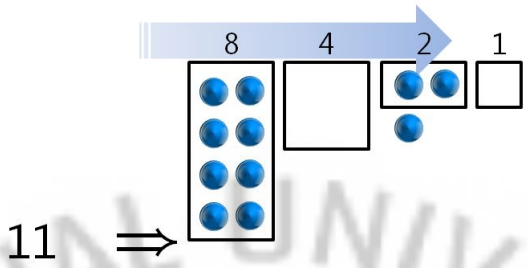
[단계1]



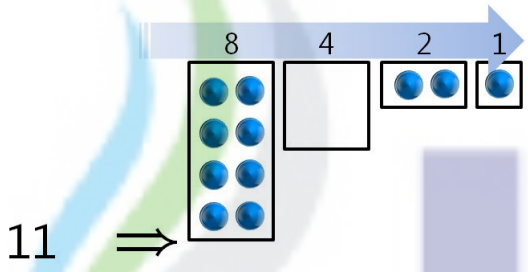
[단계2]



[단계3]



[단계4]



$11_{10}=1101_2$

[단계1]~[단계4]를 거쳐 8, 2, 1 바구니가 가득 찼다. 가득찬 바구니를 1, 빈 바구니를 0으로 표현했을 때 $11_{10}=1101_2$라고 표현할 수 있는 것이다.

위에서 설명한 방법이 바로 '욕심쟁이' 알고리즘이다. 이러한 욕심쟁이 알고리즘은 매 단계마다 나중을 생각하지 않고 최선의 이익을 추구하는 방법이다.

하지만 이러한 욕심쟁이 알고리즘은 항상 최적의 해답을 찾을 수는 없다.

다음의 [소스6_1]은 앞서 본 '8421'를 2진수로 바꾸는 알고리즘을 구현한 것이다.

[소스6_1]

```
1  #include <stdio.h>
2  #define N 4  // 자리값의 개수
3  int main(){
4          int NUM=11;  // 입력할 수
```

```
5         int v8421[N]={8,4,2,1};  // 자리값
6         int binary[N];  // 8421코드로 변환된 2진수의 값
7         int i;
8         for(i=0;i<N;i++){
9                 if(NUM/v8421[i]){
10                        binary[i]=1;
11                        NUM-=v8421[i];
12                }else binary[i]=0;
13        }
14        for(i=0;i<N;i++) printf("%d",binary[i]);
15        return 0;
16  }
```

4-5	NUM은 10진수 11을 저장한 변수, v8421은 8421코드의 자리값을 저장한 배열이다.
6	binary 배열은 2진수가 저장될 배열이다.
8-12	NUM을 8, 4, 2, 1 순서로 나누고 이때 몫이 생기면 binary[i]에는 1을 저장하고 NUM에는 나머지의 값을 저장한다. 만약 몫이 없을 경우는 binary[i]에 0을 저장한다.

한가지 예를 더 들어보겠다.

슈퍼마켓에서 소비자에게 1300원을 주어야 하는 상황이다. 물론 100원짜리로 13개를 줘도 상관은 없겠지만 소비자가 싫어할 뿐만 아니라 다음 차례의 소비자에게 거스름돈을 주기 위해서 최소의 지폐와 동전을 사용해야 한다면 어떤 방법을 사용해야 할까?

이 때에도 '욕심쟁이' 알고리즘을 적용해보자.

[초기상태]

거스름돈 1300원

⇒

[단계1]

[단계2]

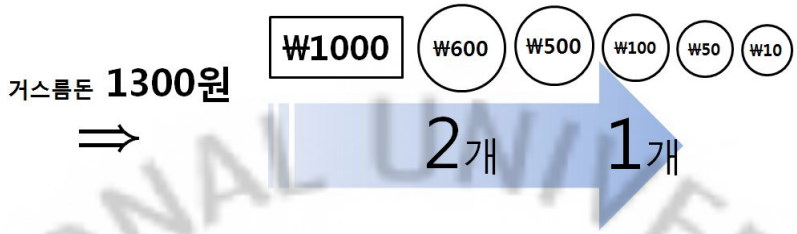
[단계3]

거스름돈이 1300원이므로 1000원 1장, 100원 3개를 이용하여 거스름돈을 주면 '욕심쟁이' 알고리즘을 이용하여 이 문제에 대한 올바른 해답을 찾게 된다.

만약 우리나라에서 600원짜리 동전을 만들었다고 생각해보자. 이 때 거스름돈 1300원을 주게 될 때에도 '욕심쟁이' 알고리즘은 1000원 1장, 100원 3개를 주게 된다.

[그림6_2] '욕심쟁이' 알고리즘

하지만 이것은 적절한 해답이 아니다. 해답은 [그림6_3]과 같이 600원짜리 동전 2개, 100원짜리 동전 1개, 총 3개의 동전만을 이용하여 거스름돈을 줄 수 있다.



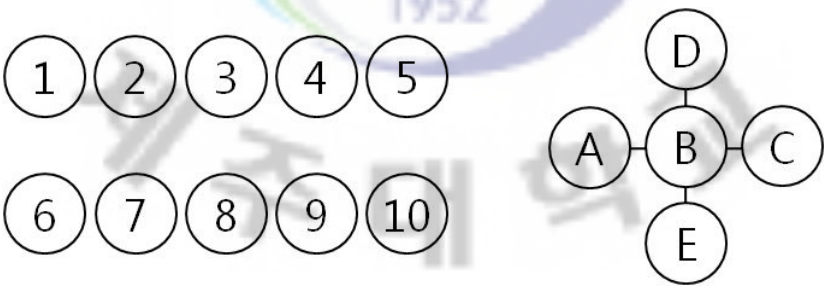
[그림6_3] 해답

즉 '욕심쟁이' 알고리즘은 때로는 빠르고 쉽게 해답을 찾기도 하지만 어떠한 경우에는 그렇지 못할 때도 있다는 것을 유념해 두어야 한다.

7 백트래킹

백트래킹은 어떠한 공간에서 가능한 모든 경우의 배치를 반복하며 해답을 찾기 위한 알고리즘이라고 할 수 있다. 다음 [그림7_1]의 문제를 풀어보자.

아래의 왼쪽 ①~⑩의 숫자를 오른쪽 십자형판 Ⓐ~Ⓔ에 하나씩만 사용하여 세로 3개 수의 합과 가로 3개 수의 합이 같을 경우는 몇 가지인가?



[그림7_1] 숫자넣기 퀴즈

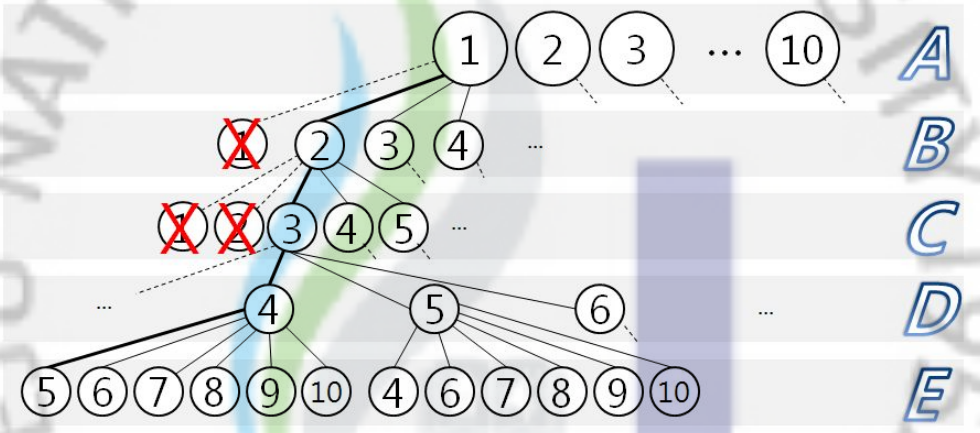
이러한 문제를 사람이 풀때에 가장 쉬운 방법은 종이와 펜을 이용하는 것일 것이다. 그리고는 잠시 생각하고 각 원안에 숫자를 써보고 지우고 하면서 여

러 가지 경우를 찾아낼 것이다.

과연 이런 문제를 컴퓨터는 어떻게 풀 것인가? 백트래킹 알고리즘에서는 위 문제에서 5개의 빈 원에 숫자가 들어갈 수 있는 모든 경우를 찾으면서 이들 중 문제의 조건(세로 3개 수의 합과 가로 3개 수의 합이 같음)을 만족하는 경우를 찾아내가는 과정으로 알고리즘이 진행된다.

예를 들면 Ⓐ~Ⓔ에 들어갈 숫자를 (1, 2, 3, 4, 5)로 정하고 이 경우 문제의 조건을 만족하는지 확인한다. 그리고 다음 경우로 (1, 2, 3, 4, 6), (1, 2, 3, 4, 7) (1, 2, 3, 4, 8), … 를 탐색하게 된다.

이러한 과정을 그림으로 표현하면 [그림7_2]와 같다.



[그림7_2] 백트래킹의 과정

위 그림을 보면 마치 깊이 우선 탐색(DFS)과 같다고 생각할 수 있겠다. 하지만 깊이 우선 탐색(DFS)은 (1, 1, 1, 1, 1), (1, 1, 1, 1, 2) (1, 1, 1, 1, 3) ~ (10, 10, 10, 10, 10)까지 아무런 조건 없이 끝까지 깊이를 따라 내려감에 비해 백트래킹은 알고리즘의 진행에 있어서 불필요하다고 생각되면 더 이상 내려가지 않는 식으로 수정된 것이다(이를 가지치기라고도 말한다). 즉, 백트래킹은 수정된 깊이우선탐색이라고 말해도 좋을 듯하다.

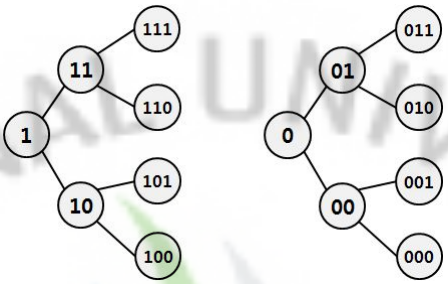
이러한 백트래킹 알고리즘을 이용하여 집합 S의 부분집합을 구해보자.

```
S[3] = { 1, 2, 3 }
```

집합 S는 1, 2, 3이라는 세 개의 원소를 가지고 있다고 하자. 집합 S의 부

분 집합은 { }, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3} 이다. 이 부분집합을 백트래킹을 이용하여 출력할 수 있을까?

재귀호출을 이용하여 다음과 같은 1과 0의 조합을 만들 수 있다.



[그림7_3] 1과 0의 조합

이 조합을 이용하면 다음 [그림7_4]처럼 집합 S의 부분집합을 쉽게 구할 수 있다.

S= | 1 | 2 | 3 |

1	1	1	=>	{1,2,3}
1	1	0	=>	{1,2}
1	0	1	=>	{1,3}
1	0	0	=>	{1}
0	1	1	=>	{2,3}
0	1	0	=>	{2}
0	0	1	=>	{3}
0	0	0	=>	{ }

[그림7_4] 부분집합

이러한 1, 0의 조합을 통해 집합 S를 배열로 하고 1인 부분은 출력하고 0인 부분은 출력하지 않으면 총 8개의 부분집합을 만들어 낼 수 있을 것이다. 이러한 알고리즘이 백트래킹일 수 있는 이유는 [그림7_3]과 같은 재귀호출에서 언제까지 반복을 계속해야 하는지에 대한 조건을 정해주기 때문이다.

다음 [소스7_1]은 위의 알고리즘을 구현한 것이다.

[소스7_1]

```
1  #include <stdio.h>
2  int S[3]={1,2,3};
3  int tmp[3];
4  void backtrack(int no){
5          int i;
6          if(no==3){
7                  for(i=0;i<3;i++) if(tmp[i]) printf("%d",S[i]);
8                  printf("\n");
9          }else{
10                 tmp[no]=1;
11                 backtrack(no+1);
12                 tmp[no]=0;
13                 backtrack(no+1);
14         }
15 }
16 int main(){
17         backtrack(0);
18         return 0;
19 }
```

2	S[3]은 {1, 2, 3}을 원소로 가진 집합이다.
3	tmp[3]은 재귀호출로 111, 110, 101, 100, … 001, 000 등을 저장하게 될 배열이다.
6-8	함수 backtrack이 인수 no를 받을 때 no가 3일때 tmp[i]가 1일때 그에 맞는 S[i]의 값을 출력시킴.
9-14	tmp[no]에 1을 넣고 backtrack 함수를 호출하고, tmp[no]에 0을 넣어 backtrack 함수를 호출한다.
16-19	메인함수에서는 backtrack 함수를 호출한다.

백트래킹을 이용하여 다른 문제도 풀어보자.

[그림7_5]과 같은 아파트 단지가 있다. '0'은 빈터이고 '1'은 아파트 건물이다. 한 아파트 건물의 상, 하, 좌, 우에 다른 아파트 건물이 있다면 그것은 연결되어 있는 것이고 한 아파트 단지라고 할 수 있다. 이러한 아파트 단지를 [그림7_6]처럼 번호를 붙여 출력시켜보자.

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

[그림7_5] 아파트 단지

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

[그림7_6] 아파트 단지 번호

눈으로 보기에는 너무나도 간단한 문제이다. 연결되어 있는 부분에 아파트 단지 번호를 써놓으면 된다. 하지만 컴퓨터로 이 문제를 해결하려면 어떻게 해야 할까? 이 문제를 깊이 우선 탐색(DFS)로 풀기에는 무리가 있다.

[그림7_5]에서의 전략은 다음과 같다.

단계① count는 아파트 단지 번호를 저장하는 변수이며 초기값은 0이다.
단계② [그림7_5]의 내용을 map[8][8]에 저장하여 두고 임시적으로 사용할 tmp[8][8]의 모든 공간에는 0을 저장한다.
단계③ map[1][1]~[7][7]까지 탐색을 하면서 숫자가 1인 부분을 찾는다. 같은 위치의 tmp배열에 숫자가 0이면 단계④~⑤를 실행하고 아니라면 계속 탐색한다.
단계④ tmp배열의 해당 위치에 count(아파트 단지 번호) 값을 저장한다.
단계⑤ map배열에서 해당 위치보다 상(하, 좌, 우)의 값이 1이고 tmp배열에서 해당 위치보다 상(하, 좌, 우)의 값이 0이라면 단계④~⑤를 실행한다. 모두 실행되었다면 단계③의 호출 지점으로 다시 돌아간다.

다음은 이를 실제 구현한 것이다.

[소스7_2]

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 101
int no;
int map[MAX][MAX]={0,};
int tmp[MAX][MAX]={0,};
int count=0;
void algorithm(int i, int j){
        tmp[i][j]=count;
        if(map[i-1][j] && !tmp[i-1][j]) algorithm(i-1,j);
        if(map[i+1][j] && !tmp[i+1][j]) algorithm(i+1,j);
        if(map[i][j-1] && !tmp[i][j-1]) algorithm(i,j-1);
        if(map[i][j+1] && !tmp[i][j+1]) algorithm(i,j+1);
}
int main(){
        int i,j;
        FILE *in=fopen("INPUT1996-1.TXT","r");
        FILE *out=fopen("OUTPUT1996-1.TXT","w");
        fscanf(in,"%d",&no);
        for(i=1;i<=no;i++){
                for(j=1;j<=no;j++){
                        fscanf(in,"%d",&map[i][j]);
                }
        }
        for(i=1;i<=no;i++){
                for(j=1;j<=no;j++){
                        if(map[i][j] && !tmp[i][j]){
                                count++;
                                algorithm(i,j);
                        }
                }
        }
        for(i=1;i<=no;i++){
```

```
34                  for(j=1;j<=no;j++){
35                          printf("%d ",tmp[i][j]);
36                  }
37                  printf("\n");
38          }
39          return 0;
40  }
```

8-14	algorithm 함수는 tmp[i][j]에 count(아파트 단지 번호)를 저장하고 map배열의 상(하, 좌, 우)에 1, tmp배열의 상(하, 좌, 우)에 0이라면 다시 재귀함수로 algorithm을 호출한다.
25-32	map배열의 모든 부분을 탐색하게 되는데 특정 위치의 값이 1일 때, tmp배열의 같은 위치의 값이 0이라면 count(아파트 단지 번호)를 1증가시키고 algorithm함수를 호출시키고 인자로 해당 배열의 위치를 넘기게 된다.

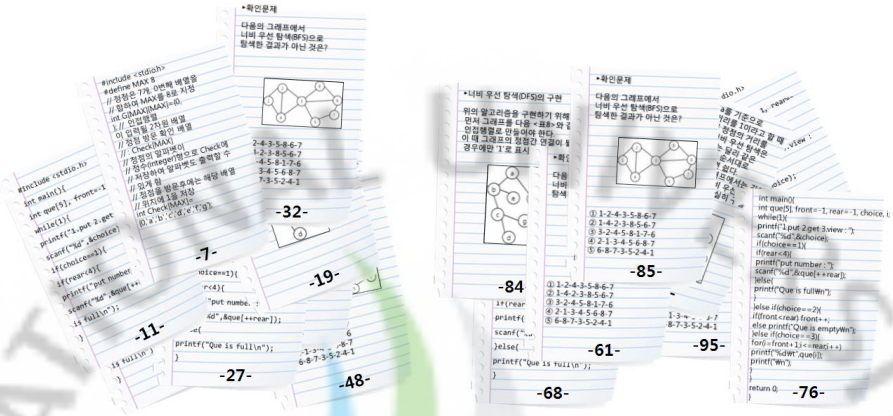
이렇게 백트래킹은 DFS를 기본으로 적절한 조건을 곁들여 복잡한 문제를 쉽게 해결해 나갈 수 있는 강력한 알고리즘이다. 하지만 모든 알고리즘이 그러하듯 백트래킹은 모든 문제의 최적의 알고리즘은 아니다.

8 분할정복

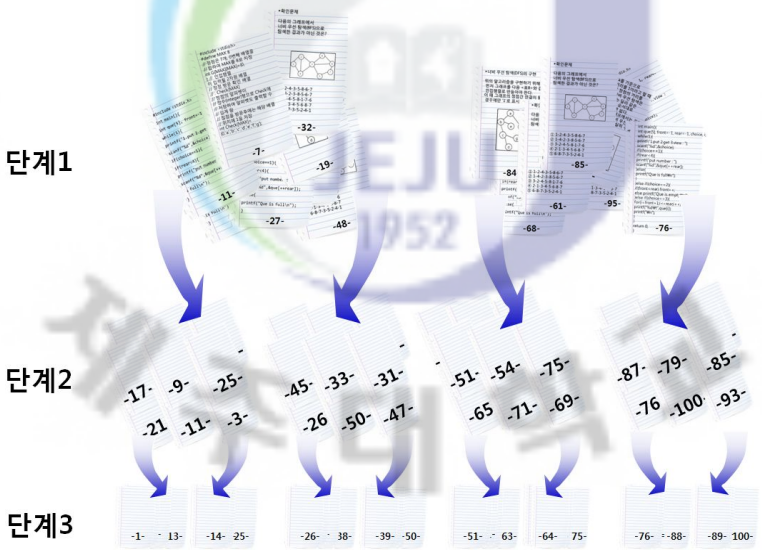
4단원에서 흩어진 종이들에 대해서 이야기했다. 100장의 흩어진 종이를 한 번에 정렬하기는 힘들지만 10장 정도의 종이는 한눈에 쪽수 비교가 될 것 같아 다음과 같은 방법을 생각해보았다.

[그림8_1] 흩어진 종이들

[단계1] 흩어진 종이의 쪽수를 순서대로 읽으며 50쪽 이하의 것과 50쪽보다 큰 것으로 분류한다.



[단계2] 단계①의 50쪽 이하의 종이를 25쪽 이하의 것과 25쪽보다 큰 것으로 분류하고 단계①의 50쪽 보다 큰 종이를 75쪽 이하의 것과 75쪽보다 큰 것으로 분류한다.



[단계3] 단계②의 25쪽 이하의 종이를 13쪽 이하의 것과 13쪽보다 큰 것으로 분류하고 25쪽보다 큰 종이를 38쪽 이하의 것과 38쪽보다 큰 것으로 분류한

다. 또, 단계②의 75쪽 이하의 종이를 63쪽 이하의 것과 63쪽보다 큰 것으로 분류하고 75쪽보다 큰 종이를 88쪽 이하의 것과 88쪽보다 큰 것으로 분류한다.

단계3까지 분류하자 한 그룹마다 12~13장 정도 때문에 쉽게 정렬할 수 있을 것 같았다. 그룹마다 정렬을 마치고 일렬로 나열하자 총 100장의 흩어진 종이를 모두 정렬한 것이나 다름없었다.

이렇게 하나의 큰 문제를 바로 풀기 힘들때 이 문제를 2개 이상의 더 작은 문제로 나누게 되고 바로 해답을 찾을 수 없으면 이러한 나누기 과정을 계속하게 된다. 이러한 방법을 바로 '분할정복' 알고리즘이라고 하는 것이다.

분할정복 알고리즘을 이용하는 하나의 예가 '병합정렬'이다. 앞서 설명했던 흩어진 종이를 다시 원래 쪽번호 순서대로 정리하는 예를 병합정렬을 이용하여 해결해보자.

병합정렬의 단계는 다음과 같다고 말할 수 있다.

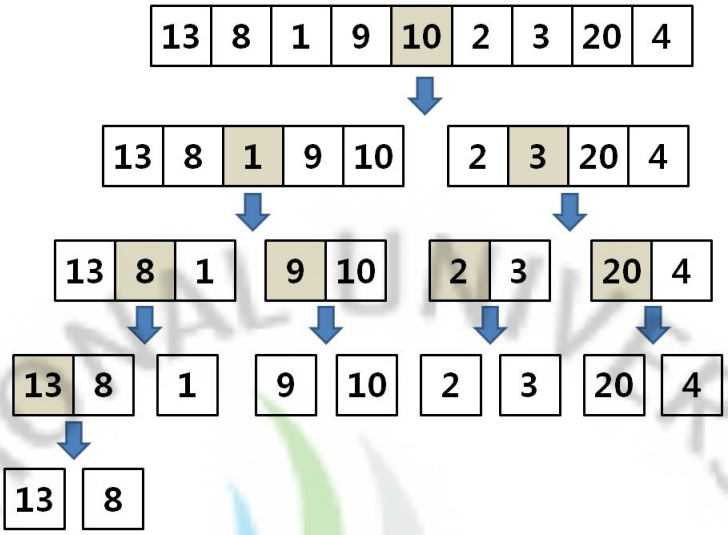
단계① 정렬하고자 하는 데이터를 반으로 나눕니다.(홀수일 경우는 $\frac{n}{2}$+1개, $\frac{n}{2}$ 개로 나눔)
단계② 나뉘어진 데이터의 크기가 2이상일 경우는 단계①을 반복합니다.
단계③ 원래 같은 데이터에서 나뉘어진 데이터 둘을 합쳐 하나의 데이터로 만들어 갑니다. 이때 정해진 기준에 의해 오름차순, 또는 내림차순으로 정렬합니다.
단계④ 데이터 집합이 원래 크기의 데이터가 될 때까지 단계③을 반복합니다.

만약 흩어진 종이의 번호가 다음과 같은 순서로 있다고 생각해보자.

13	8	1	9	10	2	3	20	4

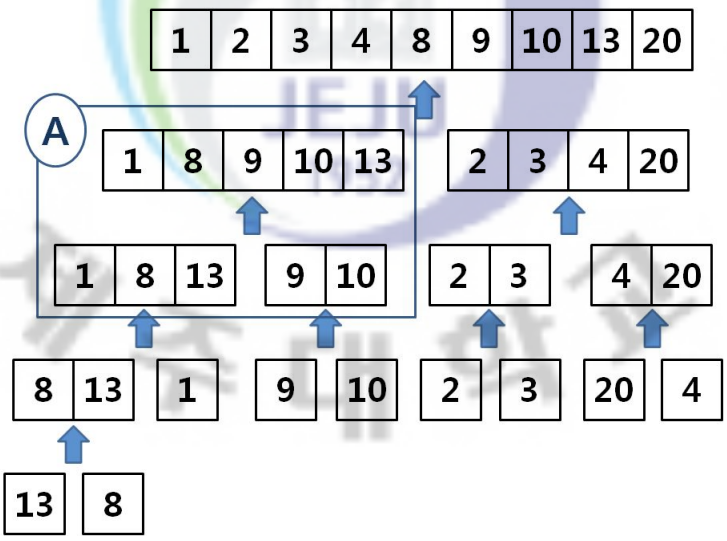
[그림8_2] 흩어진 데이터

이 데이터를 앞서 설명한 병합정렬의 단계로 분할(단계①~②)합니다. [그림 8_3]은 이 과정을 표현한 것입니다.



[그림8_3] 분할 과정

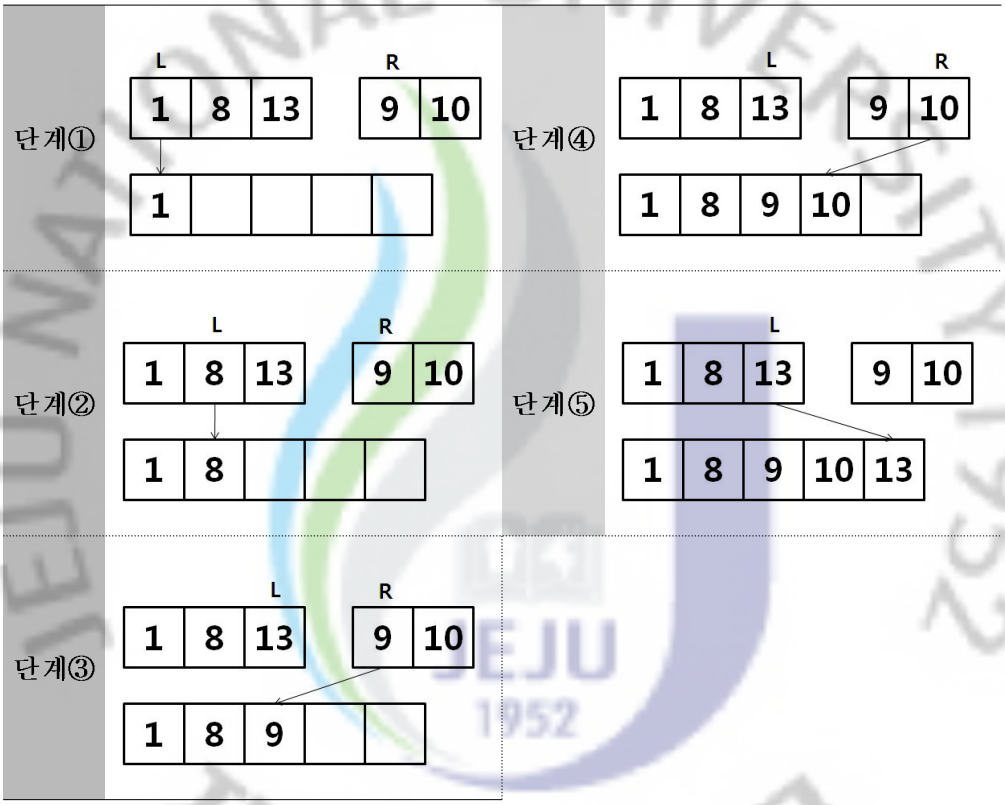
이러한 분할과정이 다 끝나면 이번에는 분할된 데이터를 정렬하는 순서다. 즉 문제를 실질적으로 해결하는 이 부분을 분할정복 알고리즘에서는 '정복'이라고 한다. [그림8_4]은 병합정렬의 단계③~④를 표현한 것이다. 이때 각 데이터의 병합시 기준은 오름차순이다.



[그림8_4] 정복 과정

위의 정복 과정에서 정렬 알고리즘을 Ⓐ부분에서 살펴보자.

이들을 병합하는 과정에서 정렬하는 방법은 총 3부분으로 나누어 설명할 수 있다. 한 부분에는 {1, 8, 13}이 있고 또 다른 한 부분에는 {9, 10}을 넣고 이 값들이 들어갈 여분의 배열 하나를 더 만들어 놓고 순서대로 비교하며 정렬하는 것이다. [그림8_5]을 보면 쉽게 이해할 수 있을 것이다. 여기서 L, R은 현재 비교할 위치를 나타내는 변수이다.



[그림8_5] 병합정렬시 예

이러한 병합정렬은 다음의 [소스8_1]과 같이 구현할 수 있다.

[소스8_1]

```
1  #include <stdio.h>
2  #define N 9
```

```
3  int data[N]={13,8,1,9,10,2,3,20,4};
4
5  void conquer(int S,int M, int E){
6          int tmp[N];  // temporary array
7          int L=S, R=M+1;  // Left, Right Position
8          int index=S, i;
9
10         while(L<=M && R<=E){
11                 if(data[L]<data[R]){
12                         tmp[index]=data[L];
13                         L++;
14                 }else{
15                         tmp[index]=data[R];
16                         R++;
17                 }
18                 index++;
19         }
20
21         if(L<=M) for(i=L;i<=M;i++,index++) tmp[index]=data[i];
22         else for(i=R;i<=E;i++,index++) tmp[index]=data[i];
23
24         for(i=S;i<=E;i++) data[i]=tmp[i];
25 }
26
27 void divide(int S, int E){
28         int M;  // Middle Position
29         if(S>E-1) return;
30         M = (S+E)/2;
31         divide(S,M);
32         divide(M+1,E);
33         conquer(S,M,E);
34 }
35
36 int main(){
```

```
37          int S=0;  // Start Position
38          int E=N-1;  // End Position
39          int i;
40
41          divide(S,E);
42
43          for(i=S;i<=E;i++) printf("%d ",data[i]);
44
45          return 0;
46  }
```

2-3	총 9개의 값을 갖는 data 배열을 생성한다.
5	conquer는 모두 쪼개어진 부분들을 다시 합치며 정렬하는 함수이다.
10-19	[그림6_11]과 같은 과정으로 두 부분의 배열을 비교하며 tmp 배열에 숫자가 작은 순서(오름차순)대로 정렬한다.
21-22	[그림6_11]과정에서 반드시 마지막 부분에는 비교할 수 없는 1개 이상의 데이터가 남게 되는데 이를 처리해 주기 위한 부분으로 남은 한쪽의 데이터를 모두 tmp 배열에 순서대로 저장한다.
24	정렬된 tmp 배열의 값을 같은 위치의 data 배열에 집어 넣는다.
27	divide는 배열을 쪼개는 함수이다.
28-30	재귀함수로 사용되기 때문에 종료 조건을 배열을 쪼갤 때 S>E-1 로 준다. 이는 배열을 1개 이하로 자를 수 없게 하는 조건이다.
31-33	잘린 배열을 divide(S,M)과 divide(M+1,E)로 다시 재귀함수로 호출하고, 이들에 대한 계산이 다 끝난 후에 이를 합쳐서 정렬하는 conquer(S,M,E)함수를 호출한다.

이렇게해서 우리는 분할정복의 한 예인 병합정렬로 분할정복에 대해서 알아 보았다. 말 그대로 분할하고 작게 나뉜 부분들을 순서대로 해결(정복)한 후에 다시 이를 합쳐 넣는 것과 같다. 빙하는 쉽게 녹일 수 없지만 잘게 쪼개진 얼음 조각들은 순식간에 녹는 것과 같은 이치라고 할 수 있다.

9 동적계획법

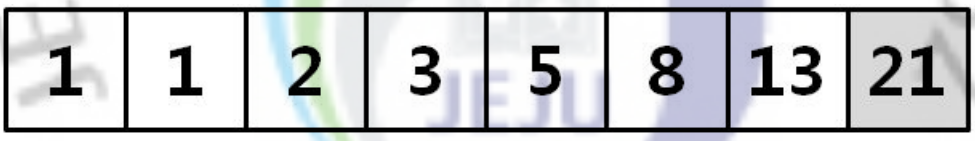
동적계획법이란 큰 문제를 작은 문제로 쪼개어 해결한다는 점에서 분할정복법과 비슷한 점이 많다. 문제를 해결할 때 분할정복법은 제일 큰 문제에서 시작하여 작은 문제로 푸는 방법을 택하고 동적계획법은 제일 작은 문제를 먼저 풀고 이를 활용하여 점점 큰 문제를 해결하는 방법을 선택한다.

동적계획법은 본선 대회에서 매우 자주 나오는 알고리즘이므로 자세히 공부해둘 필요가 있다.

1 동적계획법 맛보기

분할정복 알고리즘에서 우리는 40번째 피보나치 수열을 계산하기 위해서 총 1억번이 넘는 재귀호출이 이루어짐을 알 수 있었다. 이러한 비효율성을 다른 접근방법으로 해결할 수는 없을까?

동적계획법은 분할정복 알고리즘의 접근방법과 정반대라고 생각하면 된다. 예를 들어 피보나치 수열 8번째의 값을 구하기 위해 1~7번째 까지의 수열의 값을 구하고 Fibo(6)과 Fibo(7)의 값을 더하여 Fibo(8)의 값을 구하게 된다.



[그림9_1] 피보나치 수열 배열

이런 예는 동적계획법이 아니라 단순히 배열을 사용한 것이 아닌가 혼동하게 된다. 하지만 이것은 배열을 사용하여 동적계획법으로 구현한 문제 풀이임에 틀림없다.

다른 예로 이해를 확실하게 해보자. [그림9_2]는 5×5 크기의 방에 치즈(©)가 있는 곳과 빈 방을 표현했다. 실험용 쥐는 시작점에서 출발하여 끝점까지 치즈를 찾아 먹으며 이동하게 되는데 오른쪽, 아래로만 이동할 수 있다. 이 경우 실험용 쥐가 가장 많은 치즈를 먹고 끝점에 도달할 때의 치즈의 수를 구해보자.

시작	©	©	©	
				©
	©	©		
		©	©	
	©		©	끝

[그림9_2] 치즈방

만약 이 문제를 욕심쟁이 알고리즘으로 해결한다면 다음 [그림9_3]과 같은 경로를 예상 할 수 있다(현재 방의 아래와 오른쪽 방 모두가 빈 방일 경우는 아래로 우선 탐색).

시작	©	©	©	
				©
	©	©		
		©	©	
	©		©	끝

[그림9_3] 욕심쟁이 알고리즘

시작	©	©	©	
				©
	©	©		
		©	©	
	©		©	끝

시작	©	©	©	
				©
	©	©		
		©	©	
	©		©	끝

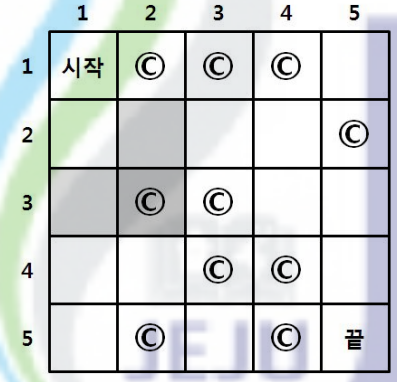
[그림9_4] 치즈방 문제의 해답

욕심쟁이 알고리즘은 현재에서 가장 이득이 되는 해결방법을 찾기 때문에 이 문제를 정확히 풀 수 없다. 또 깊이 우선 탐색은 5×5 치즈방은 쉽게 해답을 찾을 수 있겠지만 치즈방의 크기가 커질수록 계산 횟수가 엄청나게 증가하기 때문에 적절하지 못하다.

그러면 이 문제를 동적계획법으로 풀어보자. 동적 계획법의 일반적인 알고리즘을 정리해본다면 다음과 같다.

[표9_1] 동적계획법 알고리즘

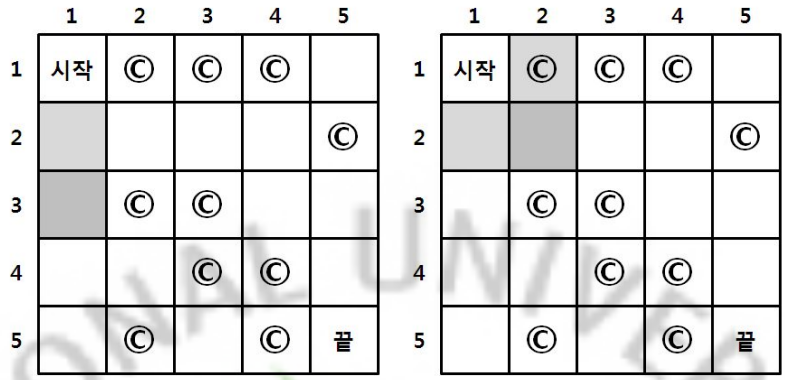
① 최소값, 최대값을 구하는 최적화 문제에 적용되는 문제인지 확인한다. ② 주어진 문제를 더 작은 문제로 분해할 수 있는지 확인한다. ③ 작게 나눠진 문제의 해답을 이용하여 더 큰 문제를 풀 수 있는 식을 만들어 이용한다.



[그림9_5] 식 만들기

이러한 조건들이 모두 맞는지 확인하고 치즈방 문제에 도입하여 식을 만들어보자. 치즈방[X][Y] 지점에 도달하기 위해서는 치즈방[X-1][Y] 또는 치즈방[X][Y-1]은 반드시 지나고 와야하기 때문에 치즈방[X-1][Y]와 치즈방[X][Y-1]에서의 치즈를 가장 많이 먹고 오는 값을 계산하면 되는 것이다.

분할정복법에서 큰 문제를 작게 나누었던 기법 그대로 이 문제도 그런 과정이 필요하다.



[그림9_6] 치즈방[3][1], 치즈방[2][2]에서의 도착 경로 찾기

예를 들자면 [그림9_6]에서 시작점을 치즈방[1][1]이라고 할 때 치즈방[3][2]에 도달하기 위해서는 실험용 생쥐는 반드시 치즈방[3][1] 또는 치즈방[2][2]를 지나야 한다. 이러한 과정을 염두해두고 아래와 같은 식을 만들어낼 수 있을 것이다.

치즈방[X][Y] = 최대값 { 치즈방[X-1][Y], 치즈방[X][Y-1] }

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

[그림9_7] 치즈방 2차 배열

0	1	2	3	3
0	1	2	3	4
0	2	3	3	4
0	2	4	5	5
0	3	4	6	6

[그림9_8] 동적계획법 알고리즘

[그림9_2]의 치즈방을 [그림9_7]과 같은 배열로 나타내고 위의 식을 이용하면 [그림9_8]과 같은 배열을 만들 수 있을 것이다.

동적계획법 알고리즘으로 [그림9_8]과 같은 2차원배열을 생성하고 결과값은 치즈방[5][5]에 쓰여진 숫자가 실험용 생쥐가 먹을 수 있는 치즈의 최대값이 된다. 위의 문제 해결방법을 구현하면 아래와 같다. INPUT.TXT 파일의 값이 [그

림9_9]와 같다고 할 때 소스는 [소스9_1]과 같다.

INPUT.TXT

```
5
0 1 1 1 0
0 0 0 0 1
0 1 1 0 0
0 0 1 1 0
0 1 0 1 0
```

[그림9_9]

[소스9_1]

```
1  #include <stdio.h>
2  #define X 101
3  #define Y 101
4  int cheese[X][Y]={0,};
5  int MAX_way(int x, int y){
6          if(cheese[x-1][y]>cheese[x][y-1]) return cheese[x-1][y];
7          else return cheese[x][y-1];
8  }
9  int main(){
10         int i,j,n;
11         FILE *in=fopen("INPUT.TXT","r");
12         FILE *out=fopen("OUTPUT.TXT","w");
13         fscanf(in,"%d",&n);
14         for(i=1;i<=n;i++){
15                 for(j=1;j<=n;j++){
16                         fscanf(in,"%d",&cheese[i][j]);
17                 }
18         }
19         for(i=1;i<=n;i++){
20                 for(j=1;j<=n;j++){
21                         cheese[i][j]+=MAX_way(i,j);
22                 }
```

```
23          }
24          fprintf(out,"%d ",cheese[n][n]);
25          fclose(in);
26          fclose(out);
27          return 0;
28  }
```

4	최대 cheese[101][101]에 치즈방 2차 배열을 입력받는다.
5-8	MAX_way라는 함수는 현재 치즈방의 위치 x, y를 받아 치즈방[x-1][y]의 값과 치즈방[x][y-1]의 값 중 최대값을 반환한다.
19-23	MAX_way함수에서 반환된 값을 현재 치즈방[i][j]에 있는 수와 합하여 저장한다.
24	치즈방[끝][끝]의 값을 OUTPUT.TXT 파일에 저장한다.

2 LIS(최대 부분 증가수열)

수열은 '숫자의 나열'을 뜻한다. LIS(Longest Increasing Subsequence)는 최대 부분 증가수열을 의미한다. 최대 부분 증가수열은 어떠한 수열의 부분수열(부분집합이라고 생각하면 됨) 중 증가되는 수로만 이루어진 최대(가장 긴) 수열을 말한다. 다음의 예를 보면 쉽게 이해할 수 있을 것이다.

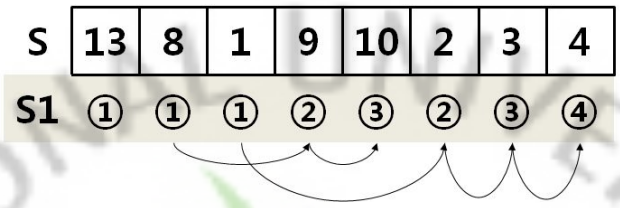
예를 들어 13, 8, 1, 9, 10, 2, 3, 4라는 수열이 있다고 할 때 부분수열은 다양하게 존재한다. {13, 8}, {13, 1, 10}, {10, 2, 3, 4} … 이 존재한다. 이러한 부분수열 중에 부분 증가수열은 {8, 9}, {8, 10}, {9, 10}, {2, 3, 4} … 처럼 점점 수가 증가되는 수열을 말한다. 따라서 최대 부분 증가수열은 이러한 부분 증가수열 중에서 가장 최대(가장 긴)인 {1, 2, 3, 4}가 된다.

이러한 LIS(최대 부분 증가수열)를 구할 수 있는 방법은 많다. 반복문을 여러번 사용한다든지, 재귀함수를 이용한 해결방법도 있지만 동적계획법을 사용하면 효율적으로 문제를 해결할 수 있다.

13	8	1	9	10	2	3	4

[그림9_10] 수열 S

[그림9_10]과 같은 배열을 S라고 하자. 이 배열에서 증가 부분 수열을 어떻게 하면 구할 수 있을까? 기준을 S[0]~S[6]까지 한번씩 증가시키면서 기준보다 오른쪽에 있는 다른 수들과 비교하면서 증가수열이 되는지 파악하는 것이다. 결과적으로 이러한 탐색 후에 [그림9_11]과 같은 값을 얻어내면 되는 것이다.



[그림9_11] 부분 증가 수열 구하기

[그림9_11]에서는 부분 증가 수열로 {8, 9, 10}과 {1, 2, 3, 4}를 구했다. 배열 S1을 하나 더 만들어 부분 증가 수열의 순서를 저장하게 하였다. 하지만 [그림9_11]의 화살표와 같은 다음 수열의 순서를 저장하기 위해서는 어떻게 하여야 할까? [그림9_12] 또는 [그림9_13]과 같은 해결방법이 있을 것이다.

	0	1	2	3	4	5	6	7
S	13	8	1	9	10	2	3	4
S1	①	①	①	②	③	②	③	④
S2	-	3	5	4	-	6	7	-

[그림9_12] S2에 다음 수열 배열주소 저장하기

	0	1	2	3	4	5	6	7
S	13	8	1	9	10	2	3	4
S1	①	①	①	②	③	②	③	④
S2	-1	-1	-1	1	3	2	5	6

[그림9_13] S2에 이전 수열 배열주소 저장하기

기준을 S[0]~S[6]까지 한번씩 증가시키면서 탐색하기 때문에 그 이후에 나올 수열의 배열주소를 알 수 없다. 따라서 [그림9_12]와 [그림9_13] 중에서 [그림9_13]이 더 효율적일 것이다. 이를 실제로 프로그램으로 구현하기 위해서는 식을 먼저 세워볼 필요가 있겠다.

$$S[i]<S[j] \Rightarrow S1[j]=S1[i]+1,\ S2[j]=i$$

위와 같은 식으로 해결할 경우 [그림9_14]와 같은 결과를 볼 수 있다.

	0	1	2	3	4	5	6	7
S	13	8	1	9	10	2	3	4
S1	①	①	①	②	③	②	③	④
S2	-1	-1	-1	2	3	2	5	6

[그림9_14] 동적계획법에 의한 결과

[그림9_14]에서 S2[3]이 [그림9_13]의 S2[3]의 값과 다른 이유는 위의 식으로 해결했기 때문이다. 즉 9는 1보다 크므로 S1[3]에 S1[2]+1의 값이 저장된 것일뿐 잘못된 것이 아니다.

이를 실제로 구현하면 [소스9_2]와 같다.

[소스9_2]

```
1  #include <stdio.h>
2  int S[8]={13,8,1,9,10,2,3,4};
3  int S1[8],S2[8];
4  void printS(int lastNo){
5         if(S2[lastNo]==-1){
6                printf("%d ",S[lastNo]);
7         }else{
8                printS(S2[lastNo]);
```

```
9                printf("%d ",S[lastNo]);
10       }
11 }
12 int main(){
13       int i,j,lastNo;
14       for(i=0;i<8;i++){
15              S1[i]=1;
16              S2[i]=-1;
17       }
18       for(i=0;i<7;i++){
19              for(j=i+1;j<8;j++){
20                     if(S[i]<S[j]){
21                            S1[j]=S1[i]+1;
22                            S2[j]=i;
23                            lastNo=j;
24                     }
25              }
26       }
27       printS(lastNo);
28       printf("\n%d",S1[lastNo]);
29       return 0;
30 }
```

4-11	재귀를 이용하여 최대 부분 증가수열의 처음 수부터 끝까지를 출력한다.
14-17	S1배열에는 1로 초기화, S2배열에는 -1로 초기화 한다.
18-26	앞서 설명한 알고리즘에 의한 식에 의해 구현한 것이다.
27-28	printS는 최대 부분 증가수열을 순서대로 출력하는 함수이고, S1[lastNo]는 최대 부분 증가수열의 개수를 나타낸다.

3 LCS(최대 공통 부분수열)

사건 현장에 형사가 출동하였다. 사건 현장에서 발견한 머리카락과 용의자의 DNA를 비교하려고 한다. 만약 DNA가 [그림9_15]과 같이 A~D의 영문 알

파벳 20개로 표현된다고 할 때 두개의 DNA가 90%(18개)이상 같다면 한 사람의 것으로 볼 수 있다고 하자.

DNA ① : CDBADCDAABDBACADBCBA
DNA ② : CBACDDAABDCBACADBBAA

[그림9_15] 두 개의 DNA

참고로 두 개의 DNA가 ABCD와 ADCB라면 두 개의 DNA가 일치한다고 할 수 없다. 이것들은 수열이므로 순서는 바꿀 수가 없다. 이 두 개의 DNA에서 공통된 부분 AC(ABCD, ADCB)만이 공통 부분 수열이다. 이런 방법으로 [그림9_15]를 다시 살펴보자.

DNA①, DNA②에서 가장 긴 공통 부분 수열은 무엇일까? 그 해답은 [그림9_16]과 같다.

DNA ① :	CDBADCDAABDBACADBCBA
DNA ② :	CBACDDAABDCBACADBBAA
최대공통부분	CBADDAABDBACADBBA

[그림9_16] 최대 공통 부분수열

결과적으로 위의 두 DNA는 최대 공통 부분수열의 길이가 17이므로 한 사람의 것이라고 보기는 힘들겠다.

다시 알고리즘에 초점을 맞추어 생각해보자.

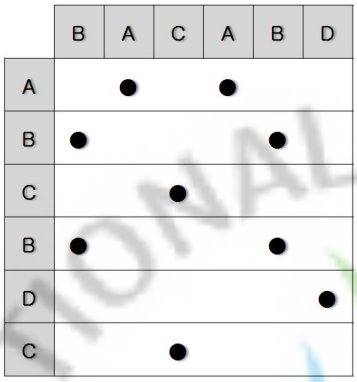
최대 공통 부분 수열은 어떤 알고리즘을 이용해야 효율적으로 구현할 수 있을까? 이 또한 동적계획법으로 구현 가능하다.

앞서 설명한 바에 의하면 동적계획법에는 식이 필요하다고 했다. 이 식을 유도하기 위해서 다음과 같은 예를 들어보자.

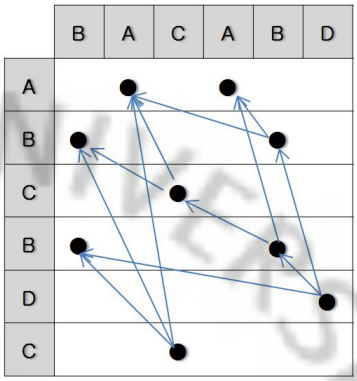
문자열 ① : BACABD
문자열 ② : ABCBDC

[그림9_17] 두 개의 문자열

[그림9_17]의 두 개의 문자열에 대한 최대 공통 부분수열을 구하기 위해서 다음과 같은 표를 그려보자.



[그림9_18] 체크리스트



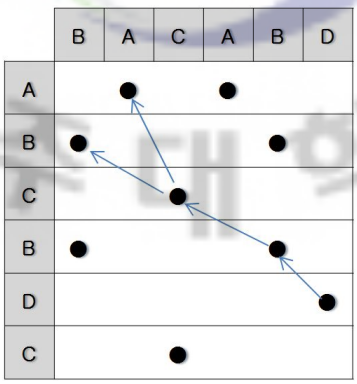
[그림9_19] 공통 부분수열 찾기

위의 [그림9_18]의 체크리스트는 문자열①의 알파벳을 가로로 씨놓고 문자열② A, B, C, B, D, C 의 순서대로 문자열①에서 찾아가며 ●를 해 놓는다. 이 ●를 이용하여 공통 부분 수열을 찾을 수 있다. [그림9_19]를 보자.

위의 그림에서 연결된 선은 공통 부분수열이다. 선을 연결하는 데에는 가장 뒤의 ●에서 부터 ↖(왼쪽 위)로만 움직일 수 있다는 규칙이 있다. 그러한 방법으로 얻은 공통 부분수열은

{B,C} {B,C} {A,C} {B,D} **{B,C,A,D}** **{A,C,A,D}** {A,B,D} {A,B,D} {A,B,D}

이다.

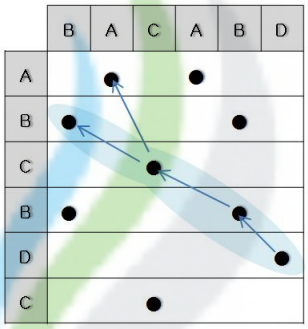


[그림9_20] 최대 공통 부분수열 찾기

사실 이 방법은 사람이 직접 공통된 부분을 뽑아가며 부분 수열을 찾는 것과 똑같은 과정이라고 생각해도 된다. 이 중 최대(가장 긴) 공통 부분수열은 {B,C,A,D} {A,C,A,D}이다. 이 수열을 구하기 위한 규칙을 생각해보자.

이 문제에 대한 동적계획 알고리즘은 다음과 같다.

단계① 문자열①의 제일 뒤의 ●를 찾는다.
단계② 문자열①의 앞 글자의 ●를 탐색하고 ●가 없을 시에는 하나 더 앞 글자를 탐색한다.
단계③ 문자열①의 제일 앞 글자에 도착할 때까지 단계②를 반복한다.



[그림9_21] 최대 공통 부분수열

이 알고리즘의 결과로 우리는 {B,C,B,D}라는 최대 공통 부분수열을 얻을 수 있을 것이다. 물론 {A,C,B,D}도 최대 공통 부분수열이지만 이 알고리즘은 최대 공통 부분 수열의 길이에 초점을 맞추었기 때문에 문제 될 것이 없다.

문자열①과 ②가 같은 부분은 '1'로 채우기

	B	A	C	A	B	D
A	0	1	0	1	0	0
B	1	0	0	0	1	0
C	0	0	1	0	0	0
B	1	0	0	0	1	0
D	0	0	0	0	0	1
C	0	0	1	0	0	0

[그림9_22] map[6][6]

이를 실제 구현할 때에는 [그림9_22]와 같은 과정을 먼저 생각해 볼 수 있겠다. 하지만 이것만으로는 [그림9_21]과 같은 역추적을 하기는 힘들 것이다. 즉 [단계2]에 적힌 숫자들 사이를 잇는 무엇인가가 필요하다. 이런 수를 넣을 때도 어떠한 규칙이 있어야 할 것이다.

규칙1) 문자열①과 ②가 같을 때에는 map[x-1][y-1]의 값보다 1이 큰 값을 입력한다.
규칙2) 현재 위치가 map[x][y]이고 문자열①과 ②의 공통된 문자가 없을 때에는 map[x-1][y]과 map[x][y-1] 중 큰 수를 입력한다.

이러한 규칙으로 [그림9_22]를 [그림9_23]처럼 바꿀 수 있다.

문자열①과 ②가 같은 부분은 '1'로 채우기

		B	A	C	A	B	D
A	0	1	0	1	0	0
B	1	0	0	0	1	0
C	0	0	1	0	0	0
B	1	0	0	0	1	0
D	0	0	0	0	0	1
C	0	0	1	0	0	0

[그림9_22] map[6][6]

규칙1), 2)를 적용한 map 배열

		B	A	C	A	B	D
A	0	1	1	1	1	1
B	1	1	1	1	2	2
C	1	1	2	2	2	2
B	1	1	2	2	3	3
D	1	1	2	2	3	4
C	1	1	2	2	3	4

[그림9_23] map[6][6]

[그림9_23]과 같은 map배열로 우리는 map[5][5]의 값이 '4'를 통해 문자열①과 ②의 공통 문자가 4개임을 알 수 있다. 즉 LCS(최대 공통 부분수열)의 길이를 찾아낸 것이다.

그러면 과연 LCS(최대 공통 부분수열)이 무엇인지 얻어낼 수는 없을까?
이를 위해서는 앞서 설명한 규칙에 다음을 덧붙이면 될 것이다.

규칙3) 문자열①과 ②가 같을 때에는 dir[x-1][y-1]로 돌아갈 수 있도록 (↖) 숫자 1을 입력한다.

규칙4) 현재 위치가 map[x][y]이고 문자열①과 ②의 공통된 문자가 없을 때에는 map[x-1][y]과 map[x][y-1] 중 큰 수를 입력한다.
map[x][y-1]가 크다면 왼쪽(←)으로 추적할 수 있도록 tmp[x][y]에 숫자 '2'를 입력한다. 그렇지 않다면 위(↑)로 추적할 수 있도록 숫자 '3'을 입력한다.

이러한 규칙에 따라서 만들어진 tmp 배열은 [그림9_24]과 같을 것이다.

	B	A	C	A	B	D
A	←2	↖1	←2	↖1	←2	←2
B	↖1	←2	←2	←2	↖1	←2
C	↑3	←2	↖1	←2	←2	←2
B	↖1	←2	↑3	←2	↖1	←2
D	↑3	←2	↑3	←2	↑3	↖1
C	↑3	←2	↖1	←2	↑3	↑3

[그림9_24] tmp[6][6]

이와 같은 방법으로 동적계획법으로 만들어진 화살표를 이용하여 역추적이 가능해졌다. 이렇게 문자열 ①과 ②를 비교하여 최대 공통 부분 수열을 출력하는 프로그램을 구현해보자.

[소스9_3]

```
1  #include <stdio.h>
2  #define MAX 7
3  int map[MAX][MAX]={0,};
4  int dir[MAX][MAX]={0,};
5  char a[MAX]=" bacabd";
6  char b[MAX]=" abcbdc";
```

```
7
8   void LCS(int x, int y){
9           if(!y){
10                  return;
11          }else if(dir[x][y]==1){
12                  LCS(x-1,y-1);
13                  printf("%c",a[y]);
14          }else if(dir[x][y]==2){
15                  LCS(x,y-1);
16          }else{
17                  LCS(x-1,y);
18          }
19  }
20
21  int main(){
22          int i,j;
23          for(i=1;i<MAX;i++){
24                  for(j=1;j<MAX;j++){
25                          if(b[i]==a[j]){
26                                  map[i][j]=map[i-1][j-1]+1;
27                                  dir[i][j]=1;
28                          }else if(map[i][j-1]>=map[i-1][j]){
29                                  map[i][j]=map[i][j-1];
30                                  dir[i][j]=2;
31                          }else{
32                                  map[i][j]=map[i-1][j];
33                                  dir[i][j]=3;
34                          }
35                  }
36          }
37          LCS(MAX-1,MAX-1);
38          return 0;
39  }
40
```

5-6	문자열①은 a[7]=" bacabd", 문자열②는 b[7]=" abcbdc" 를 저장한다.
8-19	LCS함수는 동적계획법으로 방향을 추적할 수 있게 구성된 dir[7][7] 배열을 이용하여 앞자리 문자열부터 출력하기 위해 재귀함수로 이용된다. 이 때 dir[x][y]가 1일 경우에만 문자가 출력되게 한다.
25-27	문자열②의 i번째 문자가 문자열①의 j번째 문자와 같을 경우 map[i][j]는 map[i-1][j-1]의 값보다 1이 큰 값을 저장한다. 이때 dir[i][j]의 값에는 1을 저장한다. 이는 LCS에서 문자열 출력을 위한 것이다.
28-30	문자열①과 ②가 같지 않을 경우 map[i][j]에 어떤 수를 넣을 지를 결정하기 위한 부분이다. map[i][j-1]가 map[i-1][j]보다 크거나 같을 경우에는 map[i][j]에 map[i][j-1]를 저장하고 나중에 역추적으로 문자열을 출력하기 위해 dir[i][j]에는 2를 저장한다.
31-33	문자열①과 ②가 같지 않고 map[i][j-1]가 map[i-1][j]보다 작을때에는 map[i][j]에 map[i-1][j]를 저장하고 나중에 역추적으로 문자열을 출력하기 위해 dir[i][j]에는 3를 저장한다.
37	LCS(6,6)으로 함수에 인수를 전달한다. 이는 dir의 우측 제일 하단의 숫자를 이용하여 역추적하며 문자를 출력해가게 된다.

이렇게 해서 우리는 동적계획법이 어떠한 알고리즘인지를 알게 되었다. 어떠한 일정한 규칙 또는 식이 세워지면 다른 어떠한 알고리즘보다 효율적으로 프로그램을 구현시킬 수 있는 알고리즘이다.

Ⅳ. 적용 및 연구결과의 해석

1. 연구대상 및 연구방법

본 연구에서 개발한 교재가 실제 성적면에서 어느 정도의 효과가 있는지 실험연구를 통해 검증해 보고자 다음과 같이 진행하였다.

본 연구는 <표 Ⅳ-1>과 같이 J대학교 주관의 초등정보 영재 A반 9명을 실험집단 대상으로 정하고 2009년 4월 4일부터 6월 20일까지 약 3개월 동안 7회의 집체교육으로 학습이 이루어졌다. 1회의 교육은 총 3시간이며 본 연구에서 개발한 교재로 교수·학습이 이루어졌다.

<표 Ⅳ-1> 연구대상

구 분	아 동 수				
	남		여		계
학 년	5학년	6학년	5학년	6학년	
실 험 집 단	2	3	·	4	9

2. 연구가설

본 연구에서는 다음과 같은 가설을 설정하였다.

영가설 : '한국정보올림피아드 경시대회 초등부 본선 교재'의 적용 전후 학생들의 한국정보올림피아드 문제 해결능력에는 차이가 없다.

대립가설 : '한국정보올림피아드 경시대회 초등부 본선 교재'의 적용 전후 학생들의 한국정보올림피아드 문제 해결능력에는 차이가 있다.

3. 검사도구 및 연구설계

검사도구는 [부록1]과 같이 한국정보올림피아드 경시대회에 기출된 3문제로 하고 사전·사후 검사 모두 동형인 검사지를 사용한다.

한국정보올림피아드의 답은 정해져 지만 답에 근접한 값이 정답이 되는 경우도

있을 수 있다. 또한, 답을 도출해 내는 방법은 다양할 수 있다. 문제 조건에 맞춰 문제에서 요구하는 사항들을 다 만족을 해야 100점이 나올 수 있다. 대표적으로 정보올림피아드 경시대회에서는 결과값을 내기까지의 시간을 정하게 된다. 이 시간안에 출제자가 제시한 답이 나오면 점수를 부여받을 수 있고, 반대의 경우에는 점수가 없다. 알고리즘의 효율성에 따라 제한된 시간 내에 결과를 낼 수도 있고, 제한 시간에 답을 못 낼 수도 있게 된다. 문제에서 1초안에 결과가 나올 것이라고 했는데 프로그램의 효율성이 낮은 관계로 5초에 답이 나왔다고 하면 점수를 얻을 수가 없다(한국정보화진흥원, ¶ 2).

하지만 본 연구에서는 위와 같은 기준으로는 학습 향상에 대한 결과를 도출하기 힘들기 때문에 <표 Ⅳ-2>와 같이 채점기준표를 재구성하였다.

<표 Ⅳ-2> 채점기준표

구 분	내 용	배 점
1	문제1, 문제2	각 30점
2	문제3	40점
계	총점	100점

구 분	감 점 내 용	감 점
1	모든 조건을 만족하며 모든 테스트 데이터의 결과값이 올바르다.	0점
2	결과값은 올바르나 시간이 초과된다.	-10점
3	일부의 결과값은 올바르지 못하다.	-10점
4	일부의 결과값은 올바르지 못하며 때로는 시간이 초과된다.	-20점
5	알고리즘 설계는 올바르나 결과값이 모두 올바르지 못하다(프로그래밍 과정상의 실수).	-25점

본 연구에서 개발한 교재를 투입하여 교육하기 시작하기 전인 2009년 4월 4일에 검사지를 통해 사전평가가 이루어졌고 교육을 끝낸 2009년 6월 20일에 사후평가가 이루어졌다. 채점기준표에 의해 검사 실시 결과로 얻어진 점수를 종속변인으로 설정하였다. 연구에서 α레벨은 $p<.05$로 검증하였다.

4. 연구결과 및 해석

본 연구에서 개발한 교재의 적용 결과, 다음과 같은 결과를 볼 수 있었다.

<표 Ⅳ-3> 대응표본 상관계수

		N	상관계수	유의확률
대응 1	사전검사 & 사후검사	9	.965	.000

<표 Ⅳ-4> 대응표본 검정

		대응차					t	자유도	유의확률 (양쪽)
		평균	표준편차	평균의 표준오차	차이의 95% 신뢰구간				
					하한	상한			
대응1	사전검사-사후검사	-8.33333	10.00000	3.33333	-16.02001	-.64665	-2.500	8	.037

두 변수의 대응표본 상관계수는 0.965로 강한 상관관계를 보이고 있으며 유의 확률(양쪽) p=0.037이므로 유의수준 0.05에서 사전검사와 사후검사의 평균차이는 유의한 것으로 나타나 영가설은 기각된다.

즉 본 연구에서 개발한 교재로 효과적인 학습이 이루어졌음을 알 수 있었다.

Ⅴ. 결론 및 제언

초등학생들의 컴퓨터 과학적 사고력, 논리력, 창의력을 기르고자 시행되는 한국정보올림피아드를 현재처럼 사교육에 의존할 수는 없다. 현재 교육현장에서는 이 대회가 단순히 교사가 사교육을 통해 교육받은 어린이를 대회장으로 인솔하는 것으로 공교육의 역할을 다하고 있는 것처럼 생각되어지고 있는 게 현실이다.

대회는 교육청 주관하에 이루어지고 교육은 사교육이 담당하니 참으로 아이러니한 일이다. 현재의 이러한 일련의 사태를 올바르게 정리할 필요가 있다. 이러한 목적으로 본 연구에서는 많은 교사의 지도용 또는 학습자의 자기주도적 학습이 이루어질 수 있는 교재를 개발하고자 한 것이다.

관련 연구(장직현, 2007)에서도 언급했듯이 한국정보올림피아드의 문제는 단순히 프로그래밍 언어 습득이나 훈련만으로 높은 점수를 받을 수 없다. 즉 한국정보올림피아드 경시대회에서 높은 점수를 받는다는 것은 경시대회 참가자가 그만큼의 논리력과 사고력, 창의력을 인정받는다는 것과 같다.

궁극적으로 본 연구에서 개발하는 교재는 한국정보올림피아드 경시대회에서 높은 점수를 받기 위함이며, 이를 다시 말하자면 컴퓨터 과학적 사고력, 논리력, 창의력을 향상시키기 위한 것이다.

교재 내용을 효과적으로 전달하기 위해 학습자의 수준에 맞게 설명할 수 있도록 J.S.Bruner는 EIS이론을 활용한 그림, 사진, 도식, 표를 적극적으로 활용하였고 실험집단을 통해 이 교재가 효과적임이 증명되었다.

향후 연구에서는 본 연구가 기반이 되어 더 좋은 교재가 개발되어 공교육에서의 컴퓨터 과학 교육의 역할을 다하기를 바라며 전국 각 교육청 또는 관련 연구회와 공동연구를 통해 교재가 개발되기를 기대한다.

참 고 문 헌

경상남도교육청. (2008). 정보올림피아드 및 프로그래밍 교육 교재(경남교육 2008-015). 경상남도교육청.

김상현. (2002). 21세기 정보화에 따른 컴퓨터교육의 실태와 개선방안에 관한 연구. 경기대 교육대학원 석사학위 논문.

김순근 외. (2005). 속전속결 알고리즘 입문. 영진닷컴.

목영해. (2001). 디지털 문화와 교육. 문음사.

백선련, 송정범, 박정호, 이태욱. (2008). 초등학생의 문제해결력을 위한 놀이 중심 알고리즘 교재 개발 및 적용. 한국컴퓨터교육학회 눈문지, 11(1), p. 87.

스티븐 스키에나 외. (2004). Programming Challenges : 알고리즘 트레이닝 북. 한빛미디어.

이영호 외. (2004). 월드와 함께하는 정보올림피아드 알고리즘 이론편. yeswoa 닷컴.

임화경. (2005). 알고리즘의 개념 학습에 대한 교수적 동기전략. 부산교육대학교 교육대학원 논문집, 7, 193-207.

장직현. (2007). 정보올림피아드와 정보과학 교육. 정보과학회지, 25(7).

조한혁 외. (2002). 인터넷 상의 조작 도구를 이용한 수학교육 프로그램 개발(E-수학교육 논문집). 한국수학교육학회.

최성규. (2003). 컴퓨터 애니메이션을 이용한 수업이 초등학생의 창의성 신장에 미치는 효과. 대구교육대학교 석사학위논문.

하성욱 외. (2002). 쉽게 배우는 실전 알고리즘 & 정보올림피아드 도전하기. 영진닷컴.

한국정보화진흥원. (n.d.). 자주묻는질문 보기. 2010. 4. 2, http://www.nia.or.kr/koi/F03000000000/F03020000000V.asp?BoardIDX=20

한국정보화진흥원. (n.d.). KOI 소개. 2010. 4. 2, http://www.nia.or.kr/KOI/F010000000000/F01020000000.asp

An official IOI Syllabus. (2006). IOI ISC 제안서, 제18회 국제 올림피아드참가보

고서, p. 233.

CSTA. (2003). Curriculum for K-12 Computer Science : Final Report of the ACM K-12 Task Force Curriculum committee. http://csta.acm.org/ Curriculum/sub/ CurrFiles/K-12ModelCurr2ndEd.pdf

Papert, S. (1980). Mindstorms. Brighton:Harvester Press.

Tom Verhoeff, Gyula Horvath, Krzysztof Diks and Gordon Cormack. (2006). A proposal for and an IOI Syllabus, Teaching Mathematics and Computer Science 4/1, pp. 193-216.

ABSTRACT*

A Study of Develping Teaching Material for the KOI final test of Elementary students

Kim, Byeongsu Su

Major in Elementary Computer Education

Jeju National University

Supervised by Professor Kim, Jong Hoon

The purpose of this study is the development of textbook for the elementary students' final of Korea Olympiad in Informatics. This textbook is designed with the pictures of algorithm's process for easy understanding by J.S.Bruner's EIS theory. It makes learners can express their own logics in computer programming languages. and questions can confirm if learners are understanding the algorithm or not. This study may give a lot of chances to many elementary students who cannot prepare the competitions because of the money and find any books about it.

---

* A thesis submitted to the committee of Graduate School of Education, Jeju National University in partial fulfillment of the requirements for the degree of Master of Education conferred in August, 2010.

부 록

[부록 1] 검사지

<그림 부록-1> 문제 1

1. 단지 번호 붙이기

<그림1>과 같이 정사각형 모양의 지도가 있다. 1은 집이 있는 곳을, 0은 집이 없는 곳을 나타낸다. 철수는 이 지도를 가지고 연결된 집들의 모임인 단지를 정의하고, 단지에 번호를 붙이려 한다. 여기서 연결되었다는 것은 어떤 집이 좌우, 혹은 아래위로 다른 집이 있는 경우를 말한다. 대각선상에 집이 있는 경우는 연결된 것이 아니다. <그림2>는 <그림1>을 단지별로 번호를 붙인 것이다. 지도를 입력하여 단지 수를 출력하고, 각 단지에 속하는 집의 수를 오름차순으로 정렬하여 출력하는 프로그램을 작성하시오.

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

< 그림1 >

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

< 그림2 >

실행 파일의 이름은 MAP.EXE로 하시오.

입력 형식

입력 파일명은 INPUT.TXT로 한다. 첫 번째 줄에는 지도의 크기 N(정사각형이므로 가로와 세로의 크기는 같으며 5≤N≤25)이 입력되고, 그 다음 N줄에는 각각 N개의 자료(0 혹은 1)가 입력된다.

출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 첫 번째 줄에는 총 단지 수를 출력하시오. 그리고 각 단지내 집의 수를 오름차순으로 정렬하여 한 줄에 하나씩 출력하시오.

입력과 출력의 예

입력(INPUT.TXT)	출력(OUTPUT.TXT)
7 0110100 0110101 1110101 0000111 0100000 0111110 0111000	3 7 8 9

<그림 부록-2> 문제 2

2. 숫자 고르기

세로로 두 줄, 가로로 N개의 칸으로 이루어진 표가 있다. 첫째 줄의 각 칸에는 정수 1, 2, …, N이 차례대로 들어 있고, 둘째 줄의 각 칸에는 1이상이고 N이하인 정수가 들어 있다. 첫째 줄에서 숫자를 적절히 뽑으면, 그 뽑힌 정수들이 이루는 집합과, 뽑힌 정수들의 바로 밑의 둘째 줄에 들어있는 정수들이 이루는 집합이 일치한다. 이러한 조건을 만족시키도록 정수들을 뽑되, 최대로 많이 뽑는 방법을 찾는 프로그램을 작성하시오. 예를 들어, N=7인 경우 아래와 같이 표가 주어졌다고 하자.

1	2	3	4	5	6	7
3	1	1	5	5	4	6

이 경우에는 첫째 줄에서 1, 3, 5를 뽑는 것이 답이다. 첫째 줄의 1, 3, 5 밑에는 각각 3, 1, 5가 있으며 두 집합은 일치한다. 이 때 집합의 크기는 3이다. 만약 첫째 줄에서 1과 3을 뽑으면, 이들 바로 밑에는 정수 3과 1이 있으므로 두 집합이 일치한다. 그러나, 이 경우에 뽑힌 정수의 개수는 최대가 아니므로 답이 될 수 없다.

실행 파일의 이름은 NUM.EXE로 하시오.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 첫째 줄에는 N을 나타내는 정수 하나가 주어진다(1≤N≤100). 그 다음 줄부터는 표의 둘째 줄에 들어가는 정수들이 순서대로 한 줄에 하나씩 입력된다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 첫째 줄에는 뽑힌 정수들의 개수를 출력하고, 그 다음 줄부터는 뽑힌 정수들을 작은 수부터 큰 수의 순서로 한 줄에 하나씩 출력한다.

입력과 출력의 예

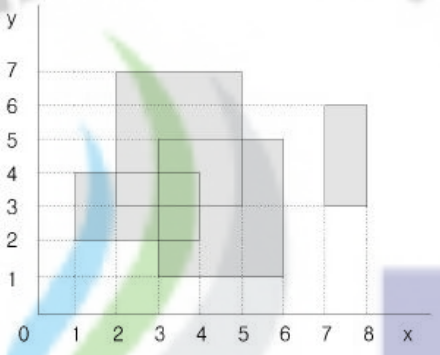
입력(INPUT.TXT)	출력(OUTPUT.TXT)
7 3 1 1 5 5 4 6	3 1 3 5

<그림 부록-3> 문제 3

3. 직사각형 네 개의 합집합의 면적 구하기

평면에 네 개의 직사각형이 놓여 있는데 그 밑변은 모두 가로축에 평행하다. 이 네 개의 직사각형들은 서로 떨어져 있을 수도 있고, 겹쳐 있을 수도 있고, 하나가 다른 하나를 포함할 수도 있으며, 변이나 꼭지점이 겹칠 수도 있다.

이 직사각형들이 차지하는 면적을 구하는 프로그램을 작성하시오.



실행 파일의 이름은 RECT.EXE로 하시오.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 입력은 네 줄이며, 각 줄은 직사각형의 위치를 나타내는 네 개의 정수로 주어진다. 첫 번째와 두 번째의 정수는 사각형의 왼쪽 아래 꼭지점의 x좌표, y좌표이고 세 번째와 네 번째의 정수는 사각형의 오른쪽 위 꼭지점의 x좌표, y좌표이다. 모든 x좌표와 y좌표는 1이상이고 100이하인 정수이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 첫 줄에 네 개의 직사각형이 차지하는 면적을 출력한다.

입력과 출력의 예

입력(INPUT.TXT)	출력(OUTPUT.TXT)
1 2 4 4 2 3 5 7 3 1 6 5 7 3 8 6	26