자료구조

2018학년도 2학기 기말시험

핵심체크 및 출제예상문제

범위 : 교재 전 범위

제1장 자료구조란 무엇인가

1. 자료와 정보 사이의 관계

1) 자료의 정의

① 현실 세계에서 관찰이나 측정을 통해서 수집된 값(value)이나 사실(fact)

② 우리의 생활에서 실제로 만질 수 있거나 볼 수 있거나 하는 것(길이, 무게, 부피 등을 측정할 수 있는 대상)에 대해서 물리적인 단위로 표현하여 얻어낼 수 있는 내용

2) 정보의 정의

① 어떤 상황에 대해서 적절한 의사결정(decision)을 할 수 있게 하는 지식(knowledge)으로서 자료의 유효한 해설(interpretation)이나 자료 상호 간의 관계(relationship)를 표현하는 내용

② 어떠한 상황에 적절한 결정이나 판단에 사용될 수 있는 형태로 가공되거나 분류되기 위해 '처리 과정'을 거쳐서 정리되고 정돈된 '자료'의 2차 처리 결과물

3) 자료와 정보의 관계

컴퓨터

자료 → [처리] → 정보

2. 추상화의 개념

1) 추상화

① 공통적인 개념을 이용하여 같은 종류의 다양한 객체를 정의하는 것

② 추상화를 통해 간결하게 말하는 사람의 의사를 전달할 수 있게 되는 것

2) 자료의 추상화

① 자료의 추상화 : 다양한 객체를 컴퓨터에서 표현하고 활용하기 위해 필요한 자료의 구조에 대해서 공통의 특징만을 뽑아 정의한 것

② 자료의 추상화에는 컴퓨터 내부의 이진수의 표현 방법, 저장 위치 등은 포함되지 않고 단순하게 개발자의 머릿속에 그림을 그리는 것처럼 개념화하는 것

3. 자료구조의 개념

1) 자료구조

① 추상화를 통해 자료의 논리적 관계를 구조화한 것

② 자료의 추상화와 구조화가 적절히 이루어지지 못하면 소프트웨어는 비효율적으로 수행되거나 소프트웨어의 확장성에 문제가 생길 수 있음

2) 자료구조와 알고리즘의 협동

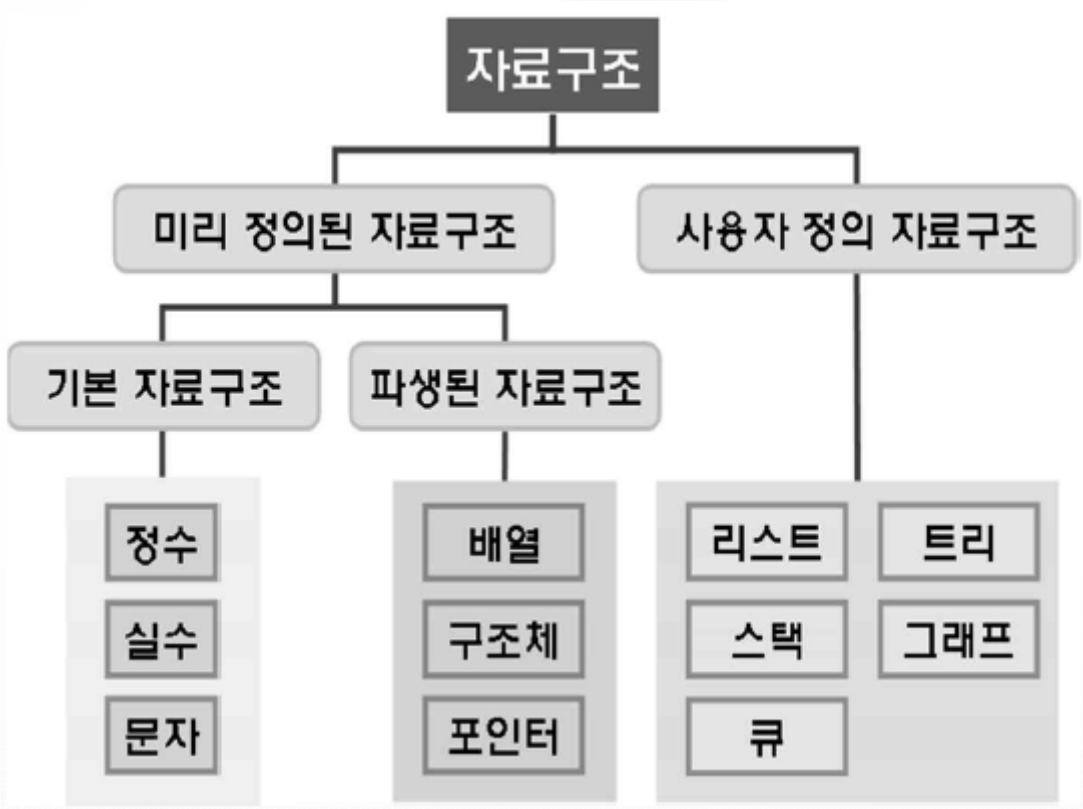
자료구조는 입력값의 추상화된 상태라면, 알고리즘은 컴퓨터가 수행해야 할 명령의 추상화

3) 자료구조의 두 가지 측면

자료구조는 입력값의 추상화된 상태라면, 알고리즘은 컴퓨터가 수행해야 할 명령의 추상화

4) 자료구조와 알고리즘의 추상화 / 구체화
입력될 값을 머릿속에서 추상화된 형태(자료구조)로 구조화하고, 수행되어야 할 명령어를 머릿속에서 추상화된 형태(알고리즘)로 체계화

5) 자료구조의 종류와 관계



4. 자료구조와 알고리즘의 관계

1) 알고리즘
① 컴퓨터에게 일을 시키는 명령들의 덩어리
② 컴퓨터에 의해 수행되기 위해 필요한 명령어들의 유한 집합이 람의 머릿속에 추상화되어 존재하는 것
③ 사람(개발자)이 컴퓨터에게 일을 시키기 위해서는 사람의 의도와 명령을 전달해 줄 수 있는 방법(프로그래밍언어)

2) 알고리즘의 조건
① 출력: 적어도 한 가지의 결과를 생성함
② 유효성: 원칙적으로 모든 명령들은 종이와 연필만으로 수행될 수 있도록 기본적 이어야함
③ 입력: 외부에서 제공되는 자료가 있을 수 있음
④ 명확성: 각 명령들은 명확하고, 모호하지 않아야 함
⑤ 유한성: 알고리즘의 명령대로 수행하면, 어떤 경우에도 한정된 수의 단계 뒤에는 반드시 종료함

5. 알고리즘 성능의 분석과 측정

1) 실행시간 분석

(1) 알고리즘을 실행하는데 필요한 실행시간을 추정하여 알고리즘의 성능을 분석(performance analysis)

(2) 실행시간의 예측

① 프로그램의 시간 복잡도는 프로그램을 실행시켜 완료하는 데 걸리는 시간을 의미. 일반적으로 어떤 프로그램 P를 실행하는 데 필요한 시간을 Tp라 할 때 이것을 다음과 같이 표현할 수 있음

- Tp = Tc + Te
- Tc = 컴파일 시간, Te = 실행 시간

② 컴파일 시간은 소스 프로그램을 컴파일하는데 걸리는 시간으로서 프로그램의 실행 특성에 의존하지 않기 때문에 고정적

③ 프로그램이 일단 정확하게 수행된다는 것이 검증되면, 그 프로그램을 재 컴파일하지 않고도 계속해서 반복해 실행시킬 수 있음. 그러므로 시간 복잡도를 비교할 때는 프로그램의 실제 실행 시간 Te만 고려하면 됨. 프로그램의 시간 복잡도를 분석하기 위해서는 프로그램 단계의 실행 빈도수를 계산함

④ 연산 시간 표기법: Big-oh(O)

- f와 g를 각각 양의 정수를 갖는 함수라 하면, 만일 어떤 두 양의 상수 a와 b가 존재하고, 모든 $n \geq b$에 대하여, $f(n) \leq a \cdot g(n)$이면 $f(n) = O(g(n))$임
- 만일 f(n)이 O (g(n))의 범위에 들어가면, 우리는 f(n)의 차수가 g(n)이라고 말하거나 f(n)는 O (g(n))이라고 함. 이 Big-oh 표기법의 기본 아이디어는 만일 f(n)가 O (g(n))이라면, n이 계속적으로 무한히 커질 때 f(n)의 값은 결국 g(n)를 상한으로 점점 가깝게 점근적으로 한정되므로, g(n)를 f(n)의 어림값으로 볼 수 있다는 것. 그런 의미에서 Big-oh(O)를 점근식 표기법(asymptotic notation)이라 함

2) 실행메모리 분석

(1) 알고리즘을 실행하는데 필요한 공간(메모리)을 추정하여 알고리즘의 성능을 분석(performance analysis)

(2) 실행 메모리의 예측

① 프로그램을 실행시켜 완료하는 데 필요한 총 저장 공간. 일반적으로 어떤 프로그램 P가 필요로 하는 총 저장 공간을 Sp라 할 때 이것은 다음과 같이 표현할 수 있음

- Sp = Sc + Se

② 고정 공간(Sc): 프로그램의 크기나 입출력의 횟수에 관계없이 고정적으로 필요한 저장 공간

③ 가변 공간(Se): 여기에는 실행 과정에서 자료 구조와 변수들이 필요로 하는 저장 공간이 포함됨. 또한 함수가 순환 호출을 할 때마다 추가로 필요로 하는 런타임 스택을 위한 저장 공간도 포함함

3) 성능 측정

(1) 컴퓨터가 실제로 프로그램을 실행하는데 걸리는 시간을 측정하여 알고리즘의 성능을 측정(performance measurement)

(2) 실행 시간의 측정

① 알고리즘의 성능을 측정한다는 것은 실제로 실행시간을 시계로 잰다는 것을 의미함

② 알고리즘의 실행파일을 얻기 위해 실제 프로그램을 작성해야 함

③ 일반적으로 프로그램의 실행은 운영체제에서 제공하는 시스템 시계를 이용함

④ 시스템 시계에 접근하는 시스템 연산 호출을 이용하면 쉽게 프로그램의 실행을 측정할 수 있음

<1장 출제예상문제>

1. 다음 중 알고리즘의 설명으로 적합한 것은 무엇인가?
① 경우에 따라 처리 결과가 생각될 수 있다.
② 입력은 반드시 내부에서만 제공되어야 한다.
③ 특정한 일을 수행하는 명령어의 유한 집합이다.
④ 경우에 따라 무한 수행이 가능해야 한다.

[정답] 3
[해설] 알고리즘은 컴퓨터가 수행해야 할 명령의 추상화이다.

2. 어떤 알고리즘의 분석 결과가 각각 다음과 같을 때, 가장 효율적인 결과는?
① O(n^2)
② O(n)
③ O(2^n)
④ O($nlogn$)

[정답] 2
[해설] O(1) < O(logn) < O(n) < O(nlogn) <O(n^2)O(n^3) < O(2^n) < O(n!)

3. 알고리즘의 성능 분석을 위한 공간 복잡도를 바르게 나타낸 것은?
① 컴파일 시간과 실행 시간의 합계
② 고정 공간과 명령어 공간의 합
③ 고정 공간과 가변 공간의 합
④ 명령어의 빈도수 계산

[정답] 3
[해설] Sp(공간 복잡도) = Sc(고정 공간) + Se(가변 공간)

4. 다음에서 선형 자료 구조의 종류가 아닌 것은?
① 배열
② 큐
③ 스택

④ 트리

[정답] 4
[해설] 선형자료 구조에는 배열, 레코드, 스택, 큐, 연결 리스트가 있다.

5. 자료구조에서 의미하는 자료와 정보에 대한 설명으로 잘못된 것은?
① 자료는 컴퓨터 처리에 의해 얻어진 종합적인 결과를 의미한다.
② 정보를 얻기 위해서는 프로그램이 필요하다.
③ 자료란 현실세계로부터 얻어지는 단순한 사실이나 값을 의미한다.
④ 정보란 어떤 상황에 대한 적절한 의사 결정을 할 수 있는 지식이다.

[정답] 1
[해설] 자료는 현실 세계에서 관찰이나 측정을 통해서 수집된 값(value)이나 사실(fact)을 말함. 이러한 값이나 사실은 수치(number)로 표현하거나 문자(character, string)로 표현됨. 반면에 정보는 어떤 상황에 대해서 적절한 의사결정(decision)을 할 수 있게 하는 지식(knowledge)으로서 자료의 유효한 해석이나 자료 상호간의 관계를 말함. 따라서 정보는 자료를 처리(process)해서 얻어진 결과(result)라고 할 수 있음. 이것을 수식으로 표현하면 I= P(D)로 나타낼 수 있으며, 이때 I는 정보, D는 자료, P는 처리를 의미한다.

6. 다음 중 알고리즘이 갖추어야 할 조건으로 틀린 것은?
① 한 가지 이상의 출력이 결과를 생성해야 한다.
② 각 명령은 명확하고 모호하지 않아야 한다.
③ 명령은 한정된 단계 뒤에는 반드시 종료되어야 한다.
④ 입력으로 외부에서 제공되는 자료가 있을 수 없다.

[정답] 4
[해설] 알고리즘이 갖추어야 할 조건은 다음과 같다.
- 입력: 외부에서 제공되는 자료가 있을 수 있다.
- 출력: 적어도 한 가지의 결과를 생성한다.
- 명확성: 각 명령들은 명확하고, 모호하지 않아야 한다.
- 유한성: 알고리즘의 명령대로 수행하면, 어떤 경우에도 한정된 수의 단계 뒤에는 반드시 종료한다.
- 유효성: 원칙적으로 모든 명령들은 종이와 연필만으로 수행될 수 있도록 기본적이어야 함. 이것은 각 연산이 명확해서만은 안 되고 반드시 실행 가능해야 한다.

7. 다음 빈 칸에 들어갈 말로 적당한 것은?

순환은 (　　) 의 특성을 가진 문제에 적합하다. (　　)는 어떤 복잡한 문제를 직접 간단하게 풀 수 있는 작은 문제로 분할하여 해결하는 방법이다.

① 내부처리

② 분할정복
③ 순차처리
④ 계승함수

[정답] 2
[해설] 순환 방식에는 분할 정복(divide and conquer)의 특성을 가진 문제가 가장 적합. 이 분할 정복은 어떤 복잡한 문제를 직접 간단하게 풀 수 있는 작은 문제로 분할하여 해결하는 방법이다.

8. 추상화와 구체화는 서로 대칭되는 표현으로 알고리즘의 구체화 표현은?
① 추상형자료형
② 데이터
③ 프로그램
④ 자료형

[정답] 3
[해설] 알고리즘의 구체화 표현은 프로그램이다.

9. 다음이 설명하는 것은 무엇인가?

현실 세계에서 관찰이나 측정을 통해서 수집된 값이나 사실이다. 일반적으로 눈으로 보거나 귀로 듣거나 코로 냄새를 맡거나 해서 얻게 된다.

① 자료
② 배열
③ 정보
④ 의사

[정답] 1
[해설] 자료에 대한 설명이다.

10. 공통적인 개념을 이용하여 같은 종류의 다양한 객체를 정의하는 것을 다음 중 무엇이라고 하는가?
① 자료
② 알고리즘
③ 추상화
④ 출력

[정답] 2
[해설] 추상화란 공통적인 개념을 이용하여 같은 종류의 다양한 객체를 정의하는 것이다.

제2장 배열

1. 배열의 정의

1) 정의

① 배열은 일정한 차례나 간격에 따라 벌여 놓음이라고 표준국어사전에 정의되어 있음

② 배열 내에서 원소의 상대적 위치를 나타내는 것이 인덱스

③ 인덱스가 하나의 값으로 표현되면 1차원 배열, 2개의 값으로 표현되면 2차원 배열, n개의 값으로 표현되면 n차원 배열이라 함

④ 인덱스가 몇 개의 값으로 구성되든지 간에 원소의 위치를 나타내는 값이라는 의미에서 배열에서는 인덱스를 하나의 논리적 단위로 취급하고 있음

2) 배열의 특성

① 순차 표현이 연속된 메모리 블록을 이용하여 데이터를 표현하는 대표적인 예가 배열

② 배열은 순차적 메모리 할당 방식으로 <인덱스(index), 원소(element)> 쌍의 집합으로, 각 쌍은 어느 한 인덱스가 주어지면 그와 연관된 원소값이 결정되는 대응 관계를 나타냄

③ 인덱스는 순서를 나타내는 원소의 유한 집합

④ 배열의 원소들은 모두 같은 타입 즉, 동질의 값으로 그 크기도 같음

⑤ 배열의 접근 방법은 직접 접근

2. 배열 추상 자료형

1) 배열 추상 데이터 타입 설명

① 행의 연산 create(n)는 n개의 원소를 저장할 수 있는 공백 배열을 생성함. 초기에는 모든 원소값이 정의되지 않은 상태임

② 행의 연산 retrieve(a,i)는 Array a와 Index I를 매개 변수로 전달받아 Index를 검사하여 유효하면 이 Index에 대응되는 원소 e를 반환하고, 무효이면 error를 반환함

③ 행의 연산 store(a,i,e)는 Array, Index, Element를 매개 변수로 전달받아 Index를 검사하여 유효하면 <인덱스, 원소>쌍이 되게 원소를 저장한 뒤 Array를 반환함

3. 배열의 연산의 구현

1) 연산의 구현

① 배열을 공백 배열로 생성하는 연산(create)

② i 번째 인덱스에 저장되어 있는 원소 값을 반환하는 연산(retrieve)

③ i 번째 인덱스에 e 원소값을 저장하는 연산(store)

4. 1차원 배열

1) 1차원 배열의 표현

① 배열 중에 가장 단순한 배열은 1차원 배열임. 이 1차원 배열을 A라고 할 때 배열 원소 A(i 1,i 2,…i n)의 메모리 위치가 효율적으로 결정될 수 있는 표현 방법을 하나 택하여야 함

② 배열의 순서나 위치를 표시할 때는 첨자나 인덱스를 사용함. 위 그림의 배열 이름은 A이고 n개의 크기를 가짐. 1차원 배열의 또 다른 표현은 배열 범위, 즉 하한선(low bound)과 상한선(high bound)으로 표현하

는 것
- A(L:U)={A(i)}
- i=L, L+1, …, U-1, U

③ 예) 1차원 배열 A를 연속적인 주소의 개념을 사용하여 기억 장치에 표현
- 배열원소: A[0],A[1],A[2],…A[i],...,A[u 1-1]
- 주소: a ,a+1,a+2,…a+i,…a+u 1-1
- 총 원소수 = u 1

④ 위에서 배열 원소 A[0]의 주소가 a라면 임의의 A[i]의 주소는 a+i가 되지만, 원소의 크기를 L이라 할 때 주소는 a+i·0이 됨. C언어에서 일차원 배열만을 생각함. C언어 에서 일차원 배열은 변수의 이름 끝에 중괄호를 추가 하여 묵시적으로 선언 (int list[5], *plist[5];)

5. 배열의 확장

1) 2차원 배열의 표현

① 2차원 배열 A[ ]는 A[m, n] 형식으로 선언된다. 여기서 m은 행(row)을 나타내는 행의 수, n는 열(column)을 나타내는 열의 수임. 2차원 배열을 위의 그림과 같이 A[3, 4]로 선언하였다고 함

② 먼저, 행우선 순서로 차례대로 배열의 원소를 A[0, 0], A[0, 1], A[0, 2], A[0, 3], A[1, 0], A[1, 1], ..., A[2, 3]이 됨. 따라서 2차원 배열을 A[m, n]으로 선언했을 때, 임의의 A[i, j]의 주소를 구하는 일반식은 A[i, j]=α i×n+j가 됨

2) 3차원 배열의 표현

3차원 배열 A[m, n, p]의 경우에는, 차원이 [n, p]인 2차원 배열이 u0개모인 것으로 해석하면 됨. 임의의 A[i, j, k]의 주소를 찾아내기 위해서 우선 A[0, 0, 0]의 주소를 α 라 할 때, 행우선 순서에서의 A[i, j, k]의 주소=α+i×n×p+j×p+k가 되고, 열우선 순서에서의 A[i, j, k]의 주소=α+k×m×n+j×m+i가 됨

6. 희소행렬의 개념

1) 희소 행렬의 개요

① 일반적으로 행렬은 m개의 행과 n개의 열로 구성되는데 이것을 m × n(m by n이라 읽음)으로 표현하고 행렬의 차수(dimension)라 함

② m×n 행렬의 원소 수는 (m·n)개가 됨. 만일 m과 n이 같으면 정방 행렬(square matrix)이라 함

③ 행렬 B는 7개의 행과 6개의 열로 된 7 × 6 행렬로서 42개의 원소를 가지고 있음

2) 희소 행렬의 표현 방법

이제 이러한 연산을 구현하기 위해서는 희소 행렬의 표현 방법을 결정해야 함. 행렬은 기본적으로 행과 열로 원소를 식별할 수 있기 때문에, 희소 행렬의 0이 아닌 원소에 대한 <행, 열, 값>의 3원소 쌍을 열이 3인 2차원 배열로 표현하면 됨

3) 희소 행렬의 전치 알고리즘

① 전치 행렬(transposed matrix)은 원소의 행과 열이 서로 교환된 행렬 즉, 원소 <i, j, v>가 <j, i, v>로 변환되어 만들어진 행렬

② 예를 들면 원소 <0, 4, 13>과 <2, 5, 3>은 전치 행렬에서 <4, 0, 13>과 <5, 2, 3>으로 각각 변환됨

③ 만일 주어진 행렬이 일반 m×n 행렬로서 a[m, n] 배열로 표현되었다면, 전치 행렬을 표현하는 배열 b[n,

m]을 만들 수 있음. 이 방법에서는 배열 a[ ]를 열 별로 처리하고 결과는 배열 b[ ]에 행별로 저장

```
for ( j ← 0; j <= n-1 ; j ← j+1) do
for (i ← 0; i <= m-1; i ← i+1) do
b[ j, i] ← a[i, j];
```

4) C에서의 희소 행렬 구현

① C언어에서 함수의 매개변수 전달은 원칙적으로 값 호출(call by value)임

② 매개변수가 배열인 경우 transposeS( ) 함수 정의에서와 같이 배열로 선언된 형식 매개변수는 실제로 포인터임

③ 이에 대응한 함수 호출문에서 배열이 인자(argument)로 함수에 전달될 때는 그 배열의 기본 주소 (base address)가 전달되며 배열의 내용 즉, 원소들의 값은 복사되거나 전달되지 않음

④ 배열 매개변수는 참조 호출(call by reference)로 전달되는 것과 같음

⑤ 함수 호출문 sum(a, SIZE)에서 a는 배열의 시작 주소를 나타내고 sum() 함수의 형식 매개변수로 명세된 b[ ]는 배열을 지시하는 포인터로서 인자 배열 a[ ]의 시작 주소 값을 전달 받음

<2장 출제예상문제>

1. 다음에서 2차원 배열 K(4:6, -2:1)의 원소의 개수는?

① 9

② 10

③ 12

④ 15

[정답] 3

[해설] 2차원 배열의 표현은 2차원 배열 a[ ]는 a[n1, n2] 형식으로 선언된다. n1은 행(row)을 나타내는 행의 수, n2는 열(column)을 나타내는 열의 수이며 2차원 배열 a[ ]의 원소 수는 n1・n2이고 각 원소는 <행 인덱스(i), 열 인덱스(j)>를 기초로 a[i , j]로 표현된다. 2차원에서 1차원으로 사상시키는 방법으로 행우선(row major order) 방법과 열우선(column major order) 방법이 있으며 2차원 배열 B[m][n]일 경우, 처음원소 : B(0,0) 주소가 α일 때, 임의의 원소 B(i, j)의 주소는 행우선 : α +i *n +j, 열우선 : α +j *m +i이다.

2. 다음 배열 N(1:3, 1:2, 1:4)의 열우선 순서에 의해 기억 장소 할당을 바르게 기술한 항은?

① N(1,1,1) -> N(2,1,1) -> N(3,1,1)

② N(1,1,1) -> N(1,2,1) -> N(1,1,2)

③ N(1,1,1) -> N(1,1,2) -> N(1,1,3)

③ N(1,1,1) -> N(2,1,1) -> N(1,3,1)

[정답] 1
[해설] 열우선 순서에 의해 기억장소의 할당은 N(1,1,1) -> N(2,1,1) -> N(3,1,1)으로 할당된다.

3. 3차원 배열 A(1:2, 1:2, 1:3)의 12개 원소에 행우선 순서로 저장할 때 A(1,2,1)는 몇 번째가 되겠는가?
① 두 번째
② 세 번째
③ 네 번째
④ 다섯 번째

[정답] 3
[해설] 3차원 배열 표현으로 3차원 배열 a[n1,n2,n3]의 메모리 표현은 n1개의 2차원 배열(크기가 n2 ×n3)을 다시 차례로 1차원 메모리에 순차적으로 사상시키는 것으로 생각하면 되며 원소 수는 당연히 n1·n2·n3이 되고, 첫 번째 원소 a[0, 0, 0]의 주소를 α라 할 때 임의의 원소 a[i 1,i 2, i 3]의 주소는 α +i 1 · n2 · n3 +i 2 · n3 +i 3이 된다.

4. 다음 중에 희소 행렬에 대한 내용으로 맞게 기술한 것은?
① 다항식의 표현 방법으로 가장 적절한 행렬을 희소 행렬이라고 한다.
② 희소 행렬은 정방 배열에서만 나타날 수 있다.
③ 희소 행렬은 원소값이 대부분 0인 경우를 말한다.
④ 희소 행렬의 모든 원소를 메모리에 저장하는 것이 가장 효율적인 방법이다.

[정답] ③
[해설] 희소행렬은 (b)의 배열 b[ ]의 원소값도 대부분 0으로 채워져 있다. 전체 원소 수에 비하여 극소수의 원소만 0이 아닌 행렬을 희소 행렬(sparse matrix)이라 한다.

5. 배열의 개념에 대한 설명은?
① 불연속적인 메모리 블록에 할당하고, [인덱스, 원소] 쌍으로 표현된다.
② 불연속적인 메모리 블록에 할당하고, [매개변수, 원소] 쌍으로 표현된다.
③ 연속적인 메모리 블록에 할당하고, [매개변수, 원소]쌍으로 표현된다.
④ 배열은 연속적인 메모리 블록에 할당하고, [인덱스, 원소] 쌍으로 표현된다.

[정답] 2
[해설] 배열의 특성으로는 순차 표현이 연속된 메모리 블록을 이용하여 데이터를 표현하는 대표적인 예가 배열이며 배열은 순차적 메모리 할당 방식으로 <인덱스(index), 원소(element)> 쌍의 집합으로, 각 쌍은 어느 한 인덱스가 주어지면 그와 연관된 원소값이 결정되는 대응 관계를 나타내고 인덱스는 순서를 나타내는 원소의 유한 집합이다. 배열의 원소들은 모두 같은 타입 즉, 동질의 값으로 그 크기도 같으며 배열의 접근 방법은 직접 접근이다.

6. 배열 A[3][4]는 A[0][0]부터 B[2][3]까지이다. B[0][1]의 열우선 위치는?
① 1
② 3
③ 4
④ 5

[정답] 3
[해설] 행우선과 열우선 방식의 주소계산 공식은 행우선 방식에서 배열 A(l1:u1,l2:u2)에서, 처음 주소 A(l1,l2)=α라 할 때, A(l1,2)=α+1 이고, 임의의 주소 A(i, j)=α+[(i-l1)u2 +(j-l2)]L (여기에서 L은 단위 원소의 크기)이며 열우선 방식에서 배열 A(l1:u1,l2:u2)에서, 처음주소 A(l1,l2)=α라 할 때, A(2, l2)=α +1이고, 임의의 주소 A(i, j)=α+[(j-l2)u1 +(i-l1)]L

7. 배열을 생성하는 함수의 이름은 다음 중 무엇인가?
① create
② define
③ index
④ error

[정답] 1
[해설] 배열을 생성하는 함수의 이름은 create로 정의된다.

8. 배열이 인덱스 값을 이용해서 배열의 원소값에 접근하는 방식은 다음 중 무엇인가?
① 인접접근
② 직접접근
③ 도구접근
④ 순차접근

[정답] 2
[해설] 배열에 인덱스 값을 이용해서 배열의 원소값에 접근하기 때문에 직접 접근하게 된다.

9. 다음의 빈칸에 들어갈 알맞은 말은 무엇인가>

create(n)는 n개의 원소들을 저장할 수 있는 (　　)을 생성한다. 배열을 생성할 때 n개의 원소들을 저장할 수 있는 공간은 만들어지지만 그 안에 채워진 원소 값 들이 아직은 없다는 것을 의미한다.

① 희소배열
② 직접배열
③ 인덱스
④ 공백배열

[정답] 4
[해설] 공백배열에 대한 설명이다.

10. 행이 모두 연속적으로 메모리 영역을 할당박고 다음 행이 메모리 영역을 연속적으로 할당받는 방식의 행 우선 저장 방식을 가지는 배열은 다음 중 무엇인가?

① 1차 배열
② 2차 배열
③ 다중 배열
④ 단일 배열

[정답] 2
[해설] 2차 배열에 대한 설명이다.

제3장 스택

1. 스택의 개념

① 0개 이상의 원소를 갖는 유한 순서 리스트
② push(add)와 pop(delete)연산이 한곳에서 발생되는 자료구조
③ 객체와 그 객체가 저장되는 순서를 기억하는 방법에 관한 추상 자료형
④ 가장 먼저 입력된 자료가 가장 나중에 출력되는 관계를 표현함

2. 스택의 추상 자료형

1) 추상 자료형
① 행의 createStack() 연산자는 공백 스택을 생성함
② 행의 push(stack,item) 연산자는 스택의 top에 item을 삽입
③ 행의 isEmpty(stack) 연산자는 스택이 공백이면 true를 반환하고, 아니면 false를 반환
④ 행의 pop(stack) 연산자는 만약 스택이 공백이면 error를 반환하고, 아니면 스택의 top에서 한 item을 삭제하고 item 값을 반환
⑤ 행의 delete(stack) 연산자는 스택이 공백이면 error를 반환하고, 아니면 스택의 top이 가리키는 item를 삭제
⑥ 행의 peek(stack) 연산자는 스택이 공백이면 error를 반환하고, 아니면 스택의 top이 가리키는 item를 반환

3. 스택의 응용

1) 응용
① 컴퓨터 시스템 측면에서 아주 많은 분야에서 사용되고 있고, 따라서 응용분야가 다양함
② 변수에 대한 메모리의 할당과 수집을 위한 시스템 스택이 있음
③ 스택은 서브루틴의 수행이 끝난 후에 되돌아갈 함수 주소를 저장하기 위한 서부루틴 호출 관리를 위해 사용됨
④ 연산자들 간의 우선순위에 의해 계산 순서가 결정되는 수식계산이 있음
⑤ 프로그램이 수행 도중 발새오디는 인터럽트의 처리와 인터럽트 처리가 끝난 후에 되돌아갈 명령 수행지접을 정하기 위해 스택이 사용됨
⑥ 한 문자 한 문자씩 입력 받은 후에 명령어의 문법이 옳은지 검사하기 위한, 컴파일러가 있음
⑦ 함수가 자신을 함수로 되부르는 순환 호출의 실행이 끝나고 되돌아갈 실행 주소를 저장하기 위한 순환호출 관리 등에 스택은 매 우 유용하게 사용됨

4. 스택의 연산

1) 스택의 삭제 연산
① pop 연산자는 스택에서 원소를 하나 제거하는 연산인데, 만일 스택이 비어 있는 상태이면 stackEmpty 메시지를 출력한다는 것을 의미함
② 스택이 비어있는 상태가 아니라면, 스택에는 어떤 원소가 있다는 것을 의미함
③ *top이 가리키는 주소 값을 하나 감소(-1)시키는 것임
④ *(top)--에서 사용된 '--' 연산자는 C/C++ 프로그래밍 언어에서만 사요오디는 독특한 단항연산자임

2) 스택의 삽입 연산
① 메인프로그램에서 Push를 호출하면 *top에 해당하는 실 매개변수는 주소를 나타내는 포인터 변수이고, 함수 push의 형식 매개변수에는 주소 값이 전달됨
② 스택에서 *top이 가리키는 주소 값을 하나 증가(+1)시키고, top이 가리키는 메모리 주소에 item으로 전달된 값을 저장함
③ '++'연산자가 사용ㅇ되며 ++(*top)에서 사용된 '++'연산자는 C/C++ 프로그래미 언어에서만 사용되는 독특한 단항 연산자임
④ *top이 가리키는 주소 값을 1만큼 증가(+1) 시킨 후에 top이 가리키는 주소 위치에 가서 값을 저장한다고 해석하면 됨

5. 배열을 이용한 스택의 구현
1) 스택 삭제 연산
① 스택에서 데이터를 삭제하고 반환하는 연산함수를 pop으로 정의함
② 스택의 공백 여부를 확인하기 위해 top 변수의 값이 -1인지 비교함
③ top의 값이 -1인 경우에는 스택이 비어 있다는 내용을 출력함, 재대로 연산이 이루어지지 않았음을 나타내는 0의 값을 반환하고 연산을 종료함
④ top의 값이 -1이 아닌 경우에는 데이터가 들어 있는 경우이므로 top 위치에 있는 데이터의 값을 반환하고 top 변수의 값을 하나 감소시킴

2) 스택 삽입 연산
① 스택에서 데이터를 삽입하는 연산 함수를 push로 정의함
② 스택에 데이터를 저장할 공간이 있는지를 확인하기 위해서 top 변수의 값이 스택의 사이즈보다 1 작은 값과 비교하여 크거나 같은지를 비교함
③ top의 값이 스택의 사이즈보다 1 작은 값과 비교하여 크거나 같은 경우에는 스택이 full 상태로 꽉 차 있다는 내용을 출력하고 연산을 종료함
④ 스택에 빈 공간이 있는 경우에는 top의 위치를 하나 증가시킨 후에 매개변수로 전달받은 데이터 item을 변경돼 top의 위치에 저장함

6. 사칙연산식의 전위/후위/중위 표현
1) 전위 표기법
① 'A+B'를 +AB와 같이 연산자 '+'를 피연산자인 A와 B앞에 놓고 식을 표현하는 방법으로 폴란드 수학자 루카지비치에 의해서 처음 소개됨
② polish notation이라고 함

2) 후위 표기법
① A+B를 AB+와 같이 표현하는 기법임
② '+'를 피연산자인 A와 B 뒤에 놓고 식을 표현하는 방법으로 우리가 일반적으로 사용하는 수식표현 방법인 중위표기법과 달리 후위 표기법이 컴퓨터가 해석하기에 훨씬 빠르고 간결한 표현법임

3) 스택을 이용한 후위 표기식의 계산
① 수식 계산을 하려면 일상생활 속에서 사용하는 중위 표기식을 후위 표기식으로 변화해야 함

② 후위 표기식으로 변환하는 과정에서 기억해야 할 것은 연산자(+,-,/,*)와 피연산자가 언제나 하나의 뭉치로 움직임
③ 스택을 이용한 후위 표기식의 생성과정에서 명심해야 할 것은 새로 입력되는 연산자와 스택의 가장 위에 저장되어 있는 연산자의 우선순위를 비교해야 한다는 것임
④ 스택의 가장 위에 저장되어 있는 연산자의 우선순위가 저장하려는 연산자의 우선순위보다 높거나 같으면, 스택의 가장 위에 저장된 연산자를 스택에서 삭제하고 출력해야 함
⑤ 입력된 연산자가 현재 스택의 가장 위에 저장되어 있는 연산자의 우선순위보다 높은 경우에만 새롭게 입력된 연산자가 스택에 저장됨

4) 후위 표기식의 계산 알고리즘과 설명
① 알고리즘

```
- element evalPostfix(char *exp) {      // 후위 표기식(369*+)을 계산하는 연산
-     int oper1, oper2, value, i=0
-     int length = strlen(exp);
-     char symbol;
-     top = -1
-     for(i=0; i<length; I++) {
-      symbol = exp[i];
-      if(symbol != '+' && symbol != '-'&& symbol != '*' && symbol != '/') {
-             value = symbol - '0';
-             push(value);
-      }
-        else {
-              oper2 = pop( );
-              oper1= pop( );
-              switch(symbol) {
-                     case '+': push(oper1 + oper2); break;
-                     case '-': push(oper1 - oper2); break;
-                     case '*': push(oper1 * oper2); break;
-                     case '/': push(oper1 / oper2); break;
-              }
-         }
-     }
-     return pop( );
- }
```

② 알고리즘 설명
- 스택을 사용하여 후위 표기식을 계산하는 함수를 evalPostfix라고 정의함
- 피연산자의 변수를 정의하며 사칙연산을 하기 위해서는 연산자 두 개가 필요함
- char형 포인터 매개변수로 전달받은 후위 표기식의 저장된 배열의 길이를 계산하여 length 변수에 저장함
- 배열에서 값을 가져와 저장할 변수 symbol을 정의함

- 스택의 위치를 가리키는 top 변수를 -1로 초기화함
- 반복문이 시작됨
- 처음 i가 0이므로 exp[0]의 값 '3'을 symbol 변수에 저장함
- symbol 변수의 값이 연산자가 아닌 피연산자인지 비교함
- 피연산자를 스택에 저장함

<3장 출제예상문제>

1. 다음 중 LIFO 알고리즘을 갖는 순서 리스트는?
① 큐
② 스택
③ 배열
④ 연결 리스트

[정답] ②
[해설] 스택은 제일 나중에 삽입된 원소가 제일 먼저 삭제되므로 후입 선출(LIFO:Last-In-First-Out) 리스트라고 하며 스택에서의 삽입은 push() 삭제는 pop()이라고 한다.

2. 다음 중위 표기식을 후위 표기식으로 바르게 나타낸 항은?

A*(B+C)/D-P

① ABC+D/*P-
② BC+D/A*P-
③ ABC+*D/P-
④ BC+A*D/P-

[정답] ③
[해설] 괄호에 싸여있는 수식을 가장 먼저 계산화고 +보다는 우선순위가 높은 *나 /가 계산되어야 한다.

3. 다음 중위 표기식을 전위 표기식으로 바르게 나타낸 항은?

A-B*C/D

① -A/*BCD
② -*AB/CD
③ -*BC/DA
④ */-ABCD

[정답] ①
[해설] 가장 먼저 괄호로 둘러싸인 부분을 먼저 계산하고 그러고 난 후에 나누기 연산자가 빼기 연산자보다 우선순위가 높기 때문에 나누기 연산자를 먼저 계산해야 한다.

4. 다음 중 스택의 응용 분야로서 부적합한 항은?
① 순환 호출
② 인터럽트 처리
③ 서브루틴 호출
④ 작업의 스케줄링

[정답] ④
[해설] 스택의 응용 분야로는 우리 주위의 일상생활에서 택시 기사들이 사용하는 동전 보관통 외에 뷔페식당에 쌓아 둔 접시들이 좋은 예이며 컴퓨터 시스템 측면에서는 응용 분야가 다양한데, 운영 체제가 사용하는 시스템 스택(system stack), 서브루틴 호출(subroutine call), 수식 계산(evaluation of expression), 인터럽트(interrupt) 처리, 컴파일러(compiler), 순환 호출(recursion call) 등에 매우 유용하게 사용되고 있다.

5. 스택이 저장할 수 있는 최대개수의 element를 의미하는 것은 무엇인가?
① maxStack
② stack
③ IsFull
④ Boolean

[정답] 1
[해설] 매개변수인 maxStack은 스택이 저장할 수 있는 최대 개수의 element를 의미한다.

6. 다음이 설명하는 것은 무엇인가?

스택이 빈 상태라면 삭제 할 원소가 없으므로 'stackEmpty'를 출력한다. 하지만 빈 상태가 아니라면 삭제할 원소가 있으므로, 스택의 top이 가리키는 원소를 삭제하고 그 원소를 반환한다.

① maxStack
② Stack push
③ Element Pop
④ inflx notation

[정답] 3
[해설] Element Pop에 대한 설명이다.

7. 연산자를 연산자 앞에 표기하는 방법은 다음 중 무엇인가?
① 중의 표기법
② 후위 표기법
③ 전위 표기법
④ 선행 표기법

[정답] 3
[해설] 전위표기법은 연산자를 피연산자 앞에 표기하는 방법(+AB)이다.

8. 다음의 용어에 대해서 올바르게 설명하지 않은 것은?
① 스택: 객체와 그 객체가 저장되는 순서를 기억하는 방법에 관한 추상자료형
② 중위표기법: 변수에 대한 메모리의 할당과 수집을 위해 운영체제가 관리하는 스택
③ 전위표기법: 연산자를 피연산자의 앞에 표기하는 방법
④ 후기표기법: 연산자를 피연산자의 뒤에 표기하는 방법

[정답] 2
[해설] ②는 시스템 스택에 대한 설명이다.

9. 땅속에 박혀 있는 관에서 공을 넣을 수 있는 입구 부분이라고 정의하는 스택의 부분은 다음 중 무엇인가?
① 시스템 스택
② 전위표기법
③ XML
④ 스택의 톱

[정답] 4
[해설] 스택의 톱에 대한 설명이다.

10. 다음 중 스택을 생성하는 연산은 무엇인가?
① CreateS
② element
③ maxStracksize
④ IsFull

[정답] 1
[해설] CreateS은 스택을 생성하는 연산이다.

제4장 큐

1. 큐의 개념

1) 개념

① 큐의 응용분야는 우리의 일상생활에서 많이 볼 수 있음

② 큐는 가장 처음에 제출되어 작업대기 줄에 들어간 작업이 가장 처음에 처리되어 작업 스케줄이 만들어짐

③ 큐는 양쪽이 터진 관이라고 생각하면 됨

2. 큐의 추상 자료형

① 큐(queue)는 한쪽 끝에서는 항목들(items)이 삭제되고 다른 한쪽 끝에서는 항목들이 삽입되는 1차원 배열의 선형 리스트(linear list)를 의미

② 항목들이 삭제되는 끝을 앞(front)이라 하고 삽입되는 끝을 뒤(rear)라고 함

③ 큐에 먼저 삽입된 항목이 먼저 삭제되므로 선입 선출(FIFO : First-In-First-Out) 또는 선착순 서브(FCFS : first-come-first-serve) 알고리즘을 갖는 순서 리스트임

④ 큐를 컴퓨터 시스템에 표현하는 가장 간단한 방법은 1차원 배열을 사용하는 것

⑤ 큐 추상 데이터 타입

```
ADT Queue
데이터 : 0개 이상의 원소를 가진 유한 순서 리스트
연산 :
queue ∈ Queue; item ∈ Element;
① createQ() ::= create an empty queue;
② enqueue(queue, item) ::= insert item at the rear of queue; //큐의 rear에 item을 삽입//
③ isEmpty(queue) ::= if (queue is empty) then return trueelse return false;
//큐가 비었으면 true를 반환하고, 아니면 false를 반환//
④ dequeue(queue) ::= if (isEmpty(queue)) then return error
else {delete and return the front item of queue};//만약 큐가 공백(IsEmpty(queue))
이면 error을 반환하고, 아니면 큐의 front가 가리키는 item을 삭제하고 그 값을 반환//
⑤ delete(queue) ::= if (isEmpty(queue)) then return error
else {delete the front item of queue}; //만약 큐가 공백(IsEmpty(queue))이면 error
을 반환하고, 아니면 큐의 front item을 삭제//
⑥ peek(queue) ::= if (isEmpty(queue)) then return error
else {return the front item of queue}; //만약 큐가 공백(IsEmpty(queue))이면 error
을 반환하고, 아니면 큐의 front item을 반환//
End Queue
```

3. 큐의 응용

1) 응용

① 중앙처리장치(CPU)는 컴퓨팅 자원 중에서 아주 중요한 자원임

② FCFS스케줄링은 작업이 준비 큐에 도착한 순서대로 CPU를 할당받도록 해주는 기법임

③ RR 스케줄링 기법은 대화형 시스템에 사용되는 스케줄링 방식임

4. 배열을 이용한 큐의 구현

1) 큐의 생성

변수 rear의 초기값은 큐의 공백 상태를 나타내는 '-1'로 시작함

2) 큐의 삽입 연산

① 삽입 연산이 발생하면 rear 변수만 오른쪽으로 이동함

② 메인 프로그램에서 Add_q(int *rear, element item)를 호출하면 '*rear'에 해당하는 실 매개변수는 메모리에서 큐의 처음을 나타내는 주소를 가리키는 포인터 변수를 전달하고, 함수 Add_q(int *rear, element item)의 형식 매개변수에서는 rear 변수의 조소로 전달함

③ 함수 Add_q는 먼저 *rear변수의 상태를 검사함

④ 큐가 full 상태가 아니라면 큐의 원소를 삽입할 수 있는 공간이 있음 따라서 원소를 삽입하기 위해 *rear 값을 하나 증가시킨 위치에 item 값을 저장하게 됨

3) 큐의 삭제 연산

① 메인 프로그램에서 delete_q(int *front, in rear)를 호출하면, '*front'에 해당하는 실 매개변수는 메모리에서 큐의 삭제가 발생하는 위치를 가리키는 포인터 변수로 전달됨

② 함수 Delete_q는 먼저 큐의 상태를 검사함

③ 큐가 빈 상태가 아니라며 큐에서 원소를 삭제할 수 있다는 것을 의미함

5. 원형 큐

1) 정의

① 원형 큐는 파이프의 입구와 출구 부분을 연결시킨 형태임, 연결된 부분의 데이터 공간을 연속적으로 사용하기 위해 '나머지 연산자'를 사용함

② 원형 큐는 mod 연산자를 사용하여 n 개의 공간을 (0 : n-1)로 운용함

③ 원형 큐에서의 삽입(add)과 삭제(delete) 알고리즘은 일반 큐의 삽입과 삭제 알고리즘과 약간 다른데 새로운 항목의 삽입을 위해서는 rear를 시계 방향으로 하나 이동시켜야 함

④ 즉, rear의 값을 1 증가시켜야 하는데 rear를 Q(n-1)에서 시계 방향으로 하나 이동시키면 Q(0)가 되어야 하므로, 이를 수식으로 표현하기 위해 나머지를 계산하는 mod(modulus) 연산자를 사용함

⑤ 원형 큐에 항목을 삽입하는 경우 : rear ← (rear+1) mod n

⑥ 원형 큐에 항목을 삭제하는 경우 : front ← (front+1) mod n

<4장 출제예상문제>

1. 큐가 저장할 수 있는 최대 개수의 원소를 의미하는 것은 무엇인가?

① MaxQueue

② FCFS

③ rear

④ queueFull

[정답] 1
[해설] 매개변수인 maxQueue는 큐가 저장할 수 있는 최대 개수의 원소를 의미한다.

2. 큐의 상태가 빈 상태인지 확인하는 연산은 다음 중 무엇인가?
① MaxQueue
② FCFS
③ rear
④ Boolean IsFull_q

[정답] 4
[해설] Boolean IsFull_q는 큐의 상태가 빈 상태인지를 우선적으로 확인하는 연산이다.

3. 다음이 설명하는 것은 무엇인가?

작업이 도착한 순서대로 CPU가 할당되지만, CPU의 시간 할당량 또는 시간 간격에 의해 제한을 받으며, 일정한 크기의 시간 할당량을 모든 작업에 주고 그 시간 동안 작업이 완료되지 못하면 준비 큐의 맨 뒤에 다시 배치되는 기법이다.

① RR 스케줄링 기법
② FCFS 스케줄링 기법
③ 큐
④ 원형 큐

[정답] 1
[해설] RR 스케줄링 기법에 대한 설명이다.

4. 큐에 원소를 삽입하는 기본연산은 다음 중 무엇인가?
① FCFS
② rear
③ Boolean IsFull_q
④ Add_q

[정답] 4
[해설] Add_q연산자는 큐에 원소를 하나 삽입하는 연산자이다.

5. 원형 큐의 특징은?
① 빈 공간을 없애기 위해 front, rear를 재설정한다.
② 빈 공간을 없애기 위해 앞으로의 이동이 필요하다.
③ 만원 규이지만 빈 공간이 있을 수 있다.
④ mod 연산자를 이용하여 삽입 삭제가 이루어진다.

[정답] 4
[해설] 원형 큐는 mod 연산자를 사용하여 n개의 공간을 (0 : n-1)로 운용하며 원형 큐에서의 삽입(add)과 삭제(delete) 알고리즘은 일반 큐의 삽입과 삭제 알고리즘과 약간 다른데 새로운 항목의 삽입을 위해서는 rear를 시계 방향으로 하나 이동시켜야 한다. 즉, rear의 값을 1 증가시켜야 하는데 rear를 Q(n-1)에서 시계 방향으로 하나 이동시키면 Q(0)가 되어야 하므로, 이를 수식으로 표현하기 위해 나머지를 계산하는 mod(modulus) 연산자를 사용한다.

6. 큐의 논리 구조는?
① 후입선출
② 선입후출
③ 선입선출
④ 임의입출

[정답] 3
[해설] 큐에 먼저 삽입된 항목이 먼저 삭제되므로 선입 선출(FIFO : First-In-First-Out) 또는 선착순 서브(FCFS : first-come-first-serve) 알고리즘을 갖는 순서 리스트이다.

7. 다음 중 데이터의 삽입 삭제가 양쪽 끝에서 이루어지는 자료구조는?
① front
② pointer
③ array
④ deque

[정답] 4
[해설] 데크(deque)은 스택과 큐의 동작을 복합시킨 방식으로 수행하는 선형 리스트이다. ‘double ended queue’의 약자로서 deque라고 하며, 데크의 새로운 변수로는 큐의 front 대신에 left-pointer와 rear 대신에 right-pointer 변수가 있다. 어느 한쪽으로만 삽입하고 양쪽으로 삭제할 수 있는 자료 구조를 스크롤(scroll) 리스트라 하고, 반대로 삽입은 양쪽으로 하고 삭제는 어느 한쪽으로 할 수 있는 자료 구조를 쉘프(shelf) 리스트라고 한다.

8. 다음이 설명하는 용어에 대해 올바르지 않은 것은 무엇인가?
① FCFS 스케줄링: 작업이 준비 큐에 도착한 순서대로 CPU를 할당받도록 해 주는 기법

② RR 스케줄링: 작업이 도착한 순서대로 CPU가 할당되지만, CPU의 시간 할당량 또는 시간 간격에 의해 제한을 받으며, 일정한 크기의 시간 할당량을 모든 작업에 주고 그 시간 동안 작업이 완료되지 못하면 준비 큐의 맨 뒤에 다시 배치되는 기법
③ 원형큐: 파이프의 입구와 출구 부분을 연결시킨 형태이며, 큐의 양 끝을 연결시켜서 원으로 만든 형태의 큐
④ 큐: 원소의 삭제 연산이 이루어지는 곳

[정답] 4
[해설] 큐는 한쪽에서 삽입이 발생하고 다른 한쪽에서 삭제가 발생하도록 정의되었으며, 먼저 삽입된 원소가 먼저 삭제되므로 선입선출 또는 선착순 서브 알고리즘을 갖는 순서 리스트이다.

9. 다음 중 큐에서 원소를 하나 삭제하는 연산자는 무엇인가?
① Delete_q
② rear
③ Boolean IsFull_q
④ Add_q

[정답] 1
[해설] Delete_q에 대한 설명이다.

10. 작업이 준비 큐에 도착한 순서대로 CPU를 할당받도록 해 주는 기법은 무엇인가?
① FCFS 스케줄링
② RR 스케줄링
③ 원형큐
④ 큐

[정답] 1
[해설] FCFS 스케줄링에 대한 설명이다.

제5장 연결 리스트

1. 리스트의 개념

1) 개념

① '일정한 순서'의 나열

② 어떤 정의에 의해서 결정된 '논리적인 순서'의 나열

③ 리스트의 '순서'는 데이터가 저장되는 물리적인 위치와 상관없이 사람들의 머릿속에 인식되는 '논리적인 순서', 혹은 리스트에 나타나는 원소들 간의 '의미적인 순서'를 의미함

④ 물품이나 사람의 이름 따위를 일정한 순서로 적어 놓은 것

⑤ 배열은 인덱스로 표현되는 '순서'가 배열 원소의 메모리 공간(주기억 장치, DDR)에서의 물리적인 위치를 의미함

⑥ 하지만 리스트의 '순서' 개념은 어떤 정의에 의해서 결정된 '논리적인 순서' 임

⑦ 원소들의 물리적인 저장 순서나 위치와는 무관하게 원소들 간의 논리적인 순서만 유지함

2) 리스트의 구현 방법

① 포인터를 이용한 리스트의 구현 방법 : 원소값을 저장하는 공간과 다음 원소를 가리키는 위치 정보를 저장하는 공간을 함께 구현하는 방법

② 배열을 이용하여 리스트의 구현 방법

2. 배열을 이용한 리스트의 구현

1) 개념

배열을 이용한 리스트를 구현하는 방법은 리스트의 원소값을 순서대로 배열에 저장함

2) 배열을 이용한 리스트의 원소 삽입

배열의 확장 : 초기 배열 선언에서 충분히 크게 하면 어느 정도는 피할 수 있겠지만, 원소를 리스트의 중간에 삽입하기 위해서는 리스트의 원소값을 하나씩 뒤로 밀어야 하는 상황이 발생함

3) 배열을 이용한 리스트의 원소 삽입 / 삭제

'배열로 구현된 리스트'는 원소의 순서가 연속적인 물리적 주소에 저장됨

⇒ 원소를 삽입하거나 삭제하기 위해서는 해당 원소의 위치 뒤에 있는 모든 원소를 뒤로 물리거나 앞으로 당겨야만 됨

⇒ 리스트 원소값의 이동은 원소수가 많을수록 프로그램의 수행시간을 증가시킴

4) 배열을 이용한 리스트의 원소 삽입 / 삭제 시 발생하는 문제

① 리스트의 원소 삽입은 프로그램의 실행 중에 메모리 할당을 필요로 하는 경우도 발생시킴

② 배열을 이용한 리스트의 구현은 실제 IT 서비스 환경에서는 자주 사용되지 않고 있음

③ 자료의 삽입과 삭제가 빈번히 발생하는 상황에서 리스트를 배열로 구현하는 것은 빈번한 자료 이동으로 인한 비효율적인 컴퓨팅 성능을 유발함

3. 포인터를 이용한 리스트의 구현

1) 개념

① 원소의 자리에는 원소값을 저장하고, 다음 원소를 가리키는 정보의 자리에는 다음 원소가 저장될 위치의 주소 값을 저장함

② 리스트의 원소의 자리와 다음 원소를 가리키는 정보의 자리를 합쳐서 노드라고 함

③ 노드에는 데이터 요소와 리스트의 다음 원소를 지시하는 포인터가 있다고 생각하면 됨

④ 연결 리스트는 리스트 원소들의 논리적 순서만을 지원하며 실제로 리스트 원소의 저장 위치는 순차적으로 이루어지지 않음

⑤ 실제로 연결 리스트는 컴퓨터 메모리에서 저장되고, 원래 프로그래머가 추상적으로 생각하는 리스트 원소의 순서와는 다르게 컴퓨터 메모리에 저장됨

4. 포인터 변수

1) 구조체 포인터 타입

① 다양한 데이터형 변수를 하나의 상자 안에 넣어서 선언하거나 사용하는 C프로그래밍 문법이 구조체임

② 구조체는 다양한 데이터형을 하나의 단위로 다룰 수 있어서 하나의 객체에 대한 다양한 정보를 모아서 사용할 경우에 유용함

③ 연결리스트의 원소를 하나의 구조체로 정의하고, 여러 개의 구조체를 링크로 연결하여 연결리스트를 구현함

④ struct의 멤버는 같은 타입의 또 다른 struct를 지시하는 포인터도 될 수 있음. 이것을 자체 참조 구조라 하는데 이것은 리스트의 노드를 정의하는 데 유용함

```
struct char_list_node  {
char letter;
struct char_list_node *next;
};
struct char_list_node *p;
```

⑤ 이 선언문은 struct 형의 char_list_node를 선언하는 동시에 p를 이 struct char_list_node형에 대한 포인터로 선언하고 있음

⑥ 이 선언은 char_list_node 형의 노드로 구성된 연결 리스트를 형성할 수 있게 하고 포인터 p는 이 리스트의 노드를 지시할 수 있게 함

⑦ 이 포인터 p가 가리키는 노드에 연결된 다음 노드를 접근하기 위해서는 다음과 같이 포인터를 순회시키면 됨

```
p = p->next;
```

⑧ 이렇게 하면 포인터 p는 다음 노드를 지시하게 되는데 이 struct는 멤버인 next에 다음 노드의 주소를 저장하게 되어 있기 때문임

⑨ 리스트 처리를 위해 노드와 포인터를 정의할 때 typedef를 이용하면 간결해짐

⑩ 포인터 형 list_pointer는 아직 정의 되지 않은 char_list_node라는 struct 형을 이용하여 정의하였음. C언어는 아직 정의되지 않은 자료형에 대한 포인터를 미리 선언할 수 있게 허용함

⑪ 노드 구조가 정의된 뒤에 포인터 p는 NULL 값으로 초기화 되었는데 이는 p가 지시하는 주소가 없다는 것을 명시적으로 표현하는 것

2) 프로그램 실행 중의 구조체 메모리 할당
① 배열을 이용한 리스트의 구현과 포인터를 이용한 리스트의 구현은 확장성에서도 차이가 있을 수 있음
② 프로그램 실행 중에 동적으로 메모리 공간을 할당받아 리스트 원소를 새롭게 할당받은 메모리 공간에 저장하고 새롭게 할당받은 메모리 공간을 연결할 수 있는 구현방법이 C 프로그래밍 언어에서 제공함
① 포인터 변수를 이용한 연결 리스트는 프로그램 실행 도중에 값을 저장할 메모리가 필요하게 될 때 메모리를 할당받아 데이터를 처리하고, 사용이 끝난 메모리를 반환함
② C언어에서는 malloc( )이란 함수를 이용하여, 메모리를 할당받고, 더 이상 필요 없게 되면 free( )라는 함수를 이용해 이 메모리 영역을 시스템에 반환함
③ malloc( ) 함수를 호출할 때는 int나 float와 같은 자료형을 인자로 갖는 sizeof( ) 함수를 사용하여 필요한 기억장소의 크기에 대한 정보를 제공함
④ malloc( ) 함수의 반환 값은 요청한 크기의 메모리 영역에 대한 첫 번째 주소가 되므로 이를 요청한 자료형과 일치하도록 형 변환(type cast)을 하고 포인터 변수에 지정. 함수 free( )는 이전에 malloc( )으로 할당된 메모리 영역을 더 이상 사용하지 않을 때 자유 공간 리스트(free space list)에 반환

5. 연결 리스트에서 노드의 삽입과 삭제
1) 개념
① 연결리스트의 원소에 대한 대표적인 연산은 원소(노드)의 삽입과 삭제임
② 리스트 원소 삭제는 원하는 리스트의 원소를 리스트로부터 제거하는 것임
③ 리스트 원소 삽입은 원하는 리스트의 원소를 리스트의 적절한 위치에 추가하는 것임

6. 연결리스트의 여러 가지 연산프로그램
1) 연결리스트의 생성
① 연결리스트에서 원소는 연결될 다음 원소에 대한 주소를 저장해야 함
② 노드는 원소의 값을 저장하는 데이터 필드와 다음 노드의 주소를 저장할 수 있는 링크 필드로 구성됨

2) 연결리스트의 노드 삽입
① 노드를 추가하는 함수의 이름을 addNode라고 정의함
② 새롭게 만든 노드를 NewNode라고 정의함
③ 연결 리스트의 노드들이 많은 경우 마지막 노드를 LastNode라고 정의함
④ Mallic 함수를 사용하여 NewNode의 데이터 공간을 메모리에 할당받음

3) 연결리스트의 노드 삭제
① 레드 노드에서 시작하여 링크 노드의 값을 따라가면서 마지막 노드를 찾아서 삭제를 해주어야 함
② 맨 뒤의 노드가 삭제되면서 삭제되는 노드 앞의 선행 노드의 링크 필드 값은 NULL이 되어야 함
③ 삭제할 때는 연결리스트의 노드가 존재하는지, 노드가 한 개만 존재하는지, 노드가 여러 개 존재하는지를 고려해야 함

4) 연결리스트의 특정 노드 위에 삽입
① 특정 노드의 위치는 prevNode라고 정의하고, 삽입할 노드의 데이터 값과 함께 매개변수로 전달받음
② 특정노드의 삽입연살일 경우도 연결 리스트의 공백 여부 및 삽입할 노드의 위치를 찾기 위한 탐색과정이 필요함

5) 연결리스트의 특정 노드 검색
① 특정노드를 검색하려면 연결 리스트의 헤드 노드로부터 출발함
② 링크를 따라가면서 노드의 데이터 값을 원하는 데이터 값과 비교하여 같을 경우 존재 여부를 출력함
③ 간단하게 연결리스트에는 노드들이 여러 개 존재하며 찾고자 하는 노드가 반드시 있는 것으로 가정 함

<5장 출제예상문제>

1. 연결 리스트에 관한 설명으로 틀리게 기술된 항은?
① 연결 리스트는 자료 필드와 링크 필드로 구성된다.
② 자료 필드에 기억 장소의 주소가 기억될 수 있다.
③ 연결 리스트는 기억 공간상의 불연속한 기억 장소를 활용하는 자료 구조이다.
④ 연결 리스트의 끝은 일반적으로 NULL값(혹은 0값)이 기억된다.

[정답] ②
[해설] 주소 값은 링크 필드에서 저장한다.

2. 이중 연결 리스트의 가장 큰 장점은?
① 다음 노드로의 이동을 쉽게 할 수 있다.
② 자료의 삭제를 쉽게 할 수 있다.
③ 기억 장소의 낭비가 적다.
④ 현재 노드의 선행 노드를 쉽게 찾을 수 있다.

[정답] ④
[해설] 이중 연결 리스트는 단순 연결 리스트의 장점은 포인터가 현재 어떤 노드 p를 가리키고 있을 때 링크의 방향, 즉 다음 노드로의 이동은 쉽게 이루어질 수 있으며 노드 p의 선행(좌측) 노드를 찾아내려면 리스트의 처음부터 다시 찾거나 원형 연결 리스트의 경우, 리스트를 한 바퀴 순회해야만 하는 단점이 있다. 데이터 값이 오름차순으로 정렬되어 있는 단순 연결 리스트 t에서 삽입할 데이터(x)가 어떤 노드 p의 데이터 값보다 작고 p의 앞쪽 노드 값보다는 클 때, 노드 p 앞에 삽입하기 위해서 노드 p의 선행 노드의 링크 필드를 변경시켜야 하므로 p의 선행 노드를 처음부터 찾아야 한다.

3. 연결 리스트 구성을 위한 노드 구조를 바르게 설명한 항은?
① 자료 부분과 링크 부분으로 구성된다.
② 자료 부분으로만 구성된다.
③ 링크 부분으로만 구성된다.
④ storage-pool로 구성된다.

[정답] ①

[해설] 노드 구조의 논리적 표현은 데이터-링크 이다.

4. 다음은 이중 연결 리스트의 노드 x를 삭제하는 프로그램의 일부이다. 프로그램 7번째 행에 있는 괄호 (A) 속에 들어가야 할 적합한 내용은?

```
void delete(list_pointer x, list_pointer L){
        if(x == L){
                no_more_nodes();
                return;
        }

        x -> llink -> rlink = x -> rlink;
        (               A               )

        free(x);
}
```

① x -> llink -> rlink = x -> rlink;
② x -> llink -> rlink = x -> llink;
③ x -> rlink -> llink = x -> llink;
④ x -> rlink -> llink = x -> rlink;

[정답] ③
[해설] 이중 연결 원형 리스트 L에서 임의의 노드 x를 삭제하는 알고리즘으로서 다음 과 같이 풀이된다.

```
        typedef struct list_node *list_pointer;
        typedef struct list_node {
                char data;
                struct list_node *rlink;
                struct list_node *llink;
        } list_node;
①       void delete(list_pointer x, list_pointer L)  {
②               if (x == L)  {
                        no_more_nodes();
                        return;
                }
③               x->llink->rlink=x->rlink;
④               x->rlink->llink=x->llink;
⑤               free(x);
⑥               }
```

5. 다음 중 연결리스트에 대한 설명으로 잘못된 것은?
① 원소는 자료이고, 주소는 링크이다.
② 연결리스트의 노드구조는 [원소, 주소] 쌍이다.
③ 링크 필드는 노드의 자료(data) 값을 저장한다.
④ 연결 리스트는 배열과는 다른 비순차적 표현이다.

[정답] 3
[해설] ③ 일반적으로 노드는 데이터 필드와 링크 필드로 구분된다. 데이터 필드(data field)는 리스트의 원소 즉, 데이터 값을 저장하는 곳이고, 링크 필드(link field)는 포인터(pointer) 즉, 다음 노드의 주소 값을 저장하는 곳이다.

6. 다음 빈 칸에 들어갈 말로 알맞은 것은?

연결리스트에서 노드의 사용을 위한 할당과 사용이 끝난 노드의 반환을 위해 ( ) 리스트가 사용된다.

① 이중연결
② 단순연결
③ 원형연결
④ 자유공간

[정답] 4
[해설] 자유공간리스트는 자유 공간(free space)은 필요에 따라 요구하는 노드를 할당할 수 있도록 준비해둔 자유 메모리 풀(memory pool)이며 자유공간을 연결 리스트 구조를 이용하여 관리할 수 있는데 이를 위해서 초기에 사용할 수 있는 모든 메모리를 자유 공간 리스트(free space list)로 만들어 놓는다고 가정한다. 노드 할당 요청이 들어오면 자유 공간 리스트 앞에서부터 공백 노드를 할당하고 프로그램을 실행하는 기간 동안 메모리를 일종의 용품으로 취급한다.

7. 리스트의 원소의 자리와 다음 원소를 가리키는 정보의 자리는 다음 중 무엇인가?
① 포인터
② 노드
③ 리스트
④ 링크

[정답] 2
[해설] 노드는 리스트의 원소의 자리와 다음 원소를 가리키는 정보의 자리를 합쳐 이르는 말이다.

8. 다음 용어의 설명으로 적절하지 않은 것은 무엇인가?
① 리스트: 원소값과 다음 원소를 가리키는 위치의 주소 값으로 구성된 자료 단위
② 포인터: 메모리에 저장되는 값의 저장 위치에 대한 주소를 가리키는 데이터 형

③ 단항연산자: 피연사자 하나만 갖는 연산자
④ 구조체: 다양한 데이터형의 변수를 하나의 상자 안에 넣어서 선언하거나 사용하는 C프로그래밍 문법

[정답] 1
[해설] 리스트는 원소들 간의 순서가 지켜지며 유지되는 자료구조이다.

9. 다음이 설명하는 것은 무엇인가?

메모리에 저장되는 값의 위치이다. 메모리에 저장되는 값은 저장위치에 대한 주소를 가지며, 이 저장 위치를 이용해서 리스트의 원소값을 찾아갈 수 있다.

① 리스트
② 포인터
③ 메모리 주소 값
④ 구조체

[정답] 3
[해설] 메모리 주소 값에 대한 설명이다.

10. 다양한 데이터형의 변수를 하나의 상자 안에 넣어서 선언하거나 사용하는 C프로그래밍 문법은 다음 중 무엇인가?
① 리스트
② 포인터
③ 메모리 주소 값
④ 구조체

[정답] 4
[해설] 구조체에 대한 설명이다.

제6장 연결 리스트의 응용

1. 연결 리스트의 변형

1) 연결 리스트의 종류

① 단순연결리스트

- 링크가 하나만 있고, 각각의 노드는 후행 노드만을 가리키는 구조
- 특정노드의 후행 노드는 쉽게 접근할 수 있지만 특정 노드의 선행 노드에 대한 접근은 헤드노드부터 재검색해야 함

② 이중연결리스트

- 특정노드의 선행노드를 가리키는 링크와 후행노드를 가리키는 링크를 가짐
- 이중연결리스트의 특정노드에서 선행노드와 후행 노드에 간단한 프로그램 코드를 통해 접근할 수 있음

③ 원형연결리스트

- 한 방향으로 모든 노드가 계속 연결되어 있기 때문에 한 노드에서부터 다른 어떤 노드로도 접근할 수 있는 이점이 있음
- 동적으로 할당된 메모리 공간을 재상요하는 경우 높은 효율을 제공함

2. 원형 연결 리스트

1) 원형 연결 리스트의 생성

① 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 리스트의 구조를 원형으로 만든 것임

② 공백리스트에 처음으로 노드를 삽입할 경우에는 자기 자신을 가리키게 됨

③ 추가로 노드가 삽입되는 경우에는 첫 번째 노드를 가리키게 함

④ 정의 및 생성

```
typedef struct ListNode { // 원형 연결 리스트의 노드 구조 정의
        Int data[10];
        struct ListNode* link;
} listNode;

typedef struct { // 원형 연결 리스트의 헤드 노드 구조 정의
        listNode* head;
} linkedList_h;
linkedList_h* createLinkedList_h(void) { // 원형 연결 리스트의 헤드 노드 생성
        linkedList_h* H;
        H = (linkedList_h*)malloc(sizeof(linkedList_h));
        H → head = NULL;
return H; }
```

2) 원형 연결 리스트의 노드 삽입

① 원형 연결 리스의 첫 번째 노드를 삽입하는 함수의 이름을 addFirstNode라고 정의함

② 마지막 노드를 가리킬 수 있도록 하는 변수가 필요함

③ 새롭게 생성할 노드의 이름을 NewNode라고 정의함

④ malloc 함수를 사용하여 NewNode의 데이터 공간을 메모리에 할당받음
⑤ NewNode의 데이터 값에는 100의 값을 저장하고 링크 값에는 아직 가리키는 곳이 없으므로 null 값을 저장함
⑥ 헤드 노드가 가리키는 노드가 없는 경우는 현재 리스트의 노드들이 없는 경우임
⑦ 헤드노드의 링크 값에 NewNode가 가리키는 노드의 주소 값을 저장하여 가리키게 함
⑧ NewNode가 첫 번째 노드이자 마지막 노드가 되므로 자기 자신으로 연결되도록 하여 원형 연결 리스트가 되도록 함
⑨ 삽입연산

```
void addFirstNode(linkedList_h* H, int x) {
//원형 리스트 첫 번째 노드 삽입 연산, x값은 100이라고 가정함
    listNode* tempNode;
    listNode* NewNode;
    NewNode = (listNode*)malloc(sizeof(listNode));
    NewNode → data = x;
    NewNode → link = NULL;

void addFirstNode(linkedList_h* H, int x) {
if (H → head == NULL) { // 현재 리스트가 공백인 경우
    H → head = NewNode;
    NewNode → link = NewNode;
    return; }
tempNode = H → head;
while(tempNode → link != H → head)
            tempNode = tempNode → link;
NewNode → link = tempNode → link;
tempNode → link = NewNode;
H → head = NewNode;
}
```

3) 원형 연결 리스트의 노드 삭제
① 여러 개의 노드들이 존재하는 가운데 원하는 데이터 값을 가진 노드를 삭제하고자 함
② 찾고자 하는 노드를 탐색하는 과정은 별도로 존재한다고 생각하고 삭제하고자 하는 노드의 선행 노드 위치를 매개 변수로 전달하여 연산을 수행함
③ 원형연결리스트가 공백인지 확인해야 함

3. 이중 연결 리스트

1) 이중 연결 리스트의 노드 구조
① 리스트 구조상 한 노드에서 후속 노드 및 선행 노드를 가리키는 포인터를 갖고 있는 자료 구조를 이중 연결 리스트라고 함
② 각 노드에는 2개의 링크 필드가 있어서 하나는 우측 방향(forward direction, RLINK)을, 또 하나는 좌측 방

향(backward direction, LLINK)을 가리키게 됨
③ 이중 연결 리스트는 적어도 DATA, LLINK(left link), RLINK(right link)의 세 가지 필드를 가지고 있음
④ p가 이중 연결 원형 리스트의 임의의 노드를 가리킨다면 p = RLINK(LLINK(p)) = LLINK(RLINK(p))가 성립. 헤드 노드는 항상 있으므로 공백 리스트도 실제로는 하나의 노드를 갖게 됨

2) 이중 연결 리스트의 노드 삽입
① 행은 노드 x의 우측에서 새로운 노드 p를 삽입하는 프로그램이다. (a)
② 행은 삽입할 노드 p의 llink가 노드 x를 가리키도록 한다. (b)
③ 행은 삽입할 노드 p의 rlink가 노드 x의 우측 노드를 가리키도록 한다. (c)
④ 행은 노드 x의 우측 노드의 llink가 노드 p를 가리키도록 한다. (d)
⑤ 행은 노드 x의 rlink가 노드 p를 가리키도록 한다. (e)

3) 이중 연결 리스트의 노드 삭제
이중 연결 원형 리스트 L에서 임의의 노드 x를 삭제하는 알고리즘

```
typedef struct list_node *list_pointer;
typedef struct list_node {
  char data;
  struct list_node *rlink;
  struct list_node *llink;
} list_node;
void ddelete(list_pointer x, list_pointer L)  {
if (x == L)  {
        no_more_nodes( );
        return;
  }
x->llink->rlink=x->rlink;
x->rlink->llink=x->llink;
free(x);
 }
```

<6장 출제예상문제>

1. 다음 중 링크 부분이 하나만 있고, 각각의 노드는 후행 노드만을 가리키는 구조의 리스트는 무엇인가?
① 단순 연결 리스트
② 이중 연결 리스트
③ 원형 연결 리스트
④ 단형 연결 리스트

[정답] 1

[해설] 단순 연결 리스트에 대한 설명이다.

2. 다음이 설명하는 것은 무엇인가?

null 값을 갖는 마지막 노드의 링크 부분을 활용하면서도 프로그램 성능에 도움을 주기 위해 제안되었으며, 한 방향으로 모든 노드가 원형으로 계속 연결되어 있기 때문에 한 노드로부터 다른 어떤 노드로도 접근이 가능한 구조이다.

① 단순 연결 리스트
② 이중 연결 리스트
③ 원형 연결 리스트
④ 단형 연결 리스트

[정답] 3
[해설] 원형 연결 리스트에 대한 설명이다.

3. NewNode의 데이터 공간을 메모리에 할당 받는 함수는 다음 중 무엇인가?
① Null
② addFirstNode
③ return
④ malloc

[정답] 4
[해설] malloc 함수를 사용하여 NewNode의 데이터 공간을 메모리에 할당 받는다.

4. 노드 구조의 자료형을 정리하는 포인터는?
① Null
② addFirstNode
③ ListNode
④ malloc

[정답] 3
[해설] ListNode 포인터는 ListNode 노드 전체를 가리키는 포인터 변수이다.

5. 다음이 설명하는 것은 무엇인가?

특정 노드에서는 선행 노드와 후행 노드에 대해 간단한 프로그램 코드를 통해 접근할 수 있는 구조이다.

① 단순 연결 리스트

② 이중 연결 리스트
③ 원형 연결 리스트
④ 단형 연결 리스트

[정답] 2
[해설] 이중연결리스트에 대한 설명이다.

6. 연결리스트의 마지막 노드의 링크의 값은 다음 중 무엇인가?
① Null
② addFirstNode
③ ListNode
④ malloc

[정답] 1
[해설] 연결 리스트를 살펴보면 가장 마지막 노드의 링크는 언제나 null 값이라는 것이다. 즉, 리스트의 마지막 뒤에는 아무 원소도 없기 때문에 연결 리스트의 마지막 노드의 링크는 언제나 null 값을 가진다.

7. 다음 괄호에 들어갈 알맞은 말은 무엇인가?

헤드 노드는 데이터 부분이 필요 없고, 리스트의 첫 번째 원소를 가리키기 위한 링크 부분만 필요하므로 여기서 자료형 ListNode를 가리킬 수 있도록 (　　)라고 정의한다.

① Null
② addFirstNode
③ ListNode*head
④ malloc

[정답] 3
[해설] ListNode*head에 대한 설명이다.

8. 다음 중 이중 연결 리스트의 노드 구조로 올바른 것은?
① 세 개의 링크
② 두 개의 링크 와 한 개의 데이터
③ 한 개의 링크와 두 개의 데이터
④ 세 개의 링크

[정답] 2
[해설] 이중 연결 리스트의 노드 구조는 두 개의 링크와 한 개의 데이터로 정의해야 한다.

9. 다음 중 단순 연결 리스트의 단점은 무엇인가?
① 특정 노드의 선행 노드에 대한 접근은 헤드 노드로부터 재검색해야 한다.
② 링크 부분이 하나만 있다.
③ 각각의 노드는 후행 노드만을 가리키는 구조이다.
④ 특정 노드의 후행 노드는 쉽게 접근할 수 있다.

[정답] 1
[해설] 단순 연결 리스트는 특정 노드의 선행 노드에 대한 접근은 헤드 노드로부터 재검색해야 하는 단점이 있다.

10. 연결 리스트의 처음 노드를 가리키는 역할을 하는 것은 무엇인가?
① 선행노드
② 후행노드
③ 공백리스트
④ 헤드 노드

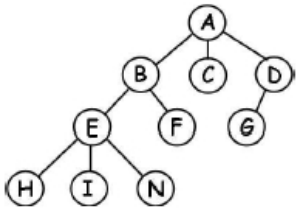
[정답] 4
[해설] 맨 앞부분은 일반적으로 헤드노드라고 불리며 연결 리스트의 처음 노드를 가리키는 역할을 한다.

제7장 트리

1. 트리

1) 정의

① 트리는 컴퓨터 분야에서 자료들 간의 관계를 나타내거나 알고리즘 문제를 해결하는 데 매우 중요한 요소로서 비선형 구조를 말함



② 트리 구조는 주변의 나무나 족보처럼 노드 사이의 관계가 계층적 관련성을 가짐

- 맨 위의 A는 선조이고, B, C, D는 A의 자식들임. 한편 E와 F는 B의 자식이며, G는 C의 자식이고, H와 I는 D의 자식임. 트리는 선조들로 거슬러 올라가는 것이 아니라 자손들을 보여 주는 표이므로 한 개체가 여러 개의 가지(branch)를 가질 수 있음
- 트리는 각 노드 사이에 사이클이 형성되지 않고, 루트 노드(root node)라고 하는 한 정점에서 계속 가지를 치는 형식을 갖고 있음

2. 용어와 논리적 표현 방법

1) 용어

① 노드(node)란 한 정보 아이템과 이로부터 다른 아이템으로 뻗은 가지를 합쳐 말함

② 일반적으로 루트(root)는 트리의 시작 노드로 맨 꼭대기에 그림

③ 어떤 노드의 서브트리 수를 그 노드의 차수(degree) 또는 분기 수라고 함

④ 차수가 0인 노드를 리프(leaf)노드 또는 단말(terminal) 노드라 하고, 그 이외의 나머지 노드들을 비단말(non terminal) 노드라고 함

⑤ 어떤 노드 X의 서브트리들의 루트들은 X의 자식(child) 노드라 하고, X는 이 자식의 부모(parent) 노드라 함

⑥ 동일한 부모의 자식들은 형제(sibling)라고 부름

⑦ 어떤 트리의 차수는 그 트리 내의 노드 차수 가운데 최대 차수를 말함

⑧ 한 노드의 조상(ancestor)이라 하면 루트에서부터 그 노드에 이르는 경로 상에 있는 모든 노드들을 말함

⑨ 어떤 노드의 레벨(level)이란 기본적으로 루트의 레벨을 1로 가정한 후에 정의. 즉, 어떤 노드가 i번째 레벨에 있다면, 그 자식 노드들의 레벨은 i + 1이 됨

⑩ 어떤 트리의 높이 또는 깊이란 그 트리 속에 노드가 가질 수 있는 최대 레벨을 말함

⑪ 포리스트(forest)란 n≥0개의 분리된 트리들의 집합을 말함. 트리와 포리스트는 아주 밀접한 관계가 있어서 트리에서 루트를 제거하면 포리스트로 됨

2) 트리의 표현

① 트리를 나타내는 방법 중 가장 유용한 것은 리스트를 이용하는 방법. 리스트 이용 방법은 루트 노드 정보를 제일 먼저 기술한 다음 그 서브트리들의 리스트가 따라 나옴

(A(B(E(K, L), F), C(G), D(H(M), I, J)))

② 연결 리스트를 사용한다면 한 개의 노드에는 그 차수만큼의 링크 필드가 필요하게 됨

자료	링크1	링크2	…	링크n

- 노드의 크기가 일정해야 데이터를 표현하는 데 적절한 알고리즘을 간단히 만들 수 있는데 데이터와 포인터 필드를 사용하면 일정한 노드 크기의 리스트 형태로 트리를 나타낼 수 있기 때문

③ 트리의 노드들의 배열순서(특히 레벨이 같은 노드들의 좌우 순서) 가 고정되어 위치의 의미가 중요한 트리를 순서 트리라고 하며, 그렇지 않은 트리를 비순서 트리라고 함

3. 추상 자료형

1) 내용

① 큐와 스택은 1차원 기억장소에서 데이터를 어떻게 넣고 빼는지에 따라 규정한 단순한 자료구조이므로 정의할 연산의 개수도 적고 각 연산이 수행하는 내용도 비교적 간단함

② 트리는 데이터의 계층관계, 포함관계 등을 나타내는 자료구조

4. 이진트리

1) 이진트리 정의

① 트리에 속한 모든 노드의 차수가 2이하인 트리를 이진트리라고 함

② 수학적으로 이론을 정리하기 쉽고 컴퓨터 내부에 구현하기도 쉬워 자주 사용함

③ 모든 노드가 두 개 이하의 자식을 가지므로 일반성을 잃지 않고 오른쪽, 왼쪽이라는 방향 개념을 부여할 수 있음

2) 완전 이진트리의 정의

이진트리에서 각 레벨에서 허용되는 최대 개수 노드를 가지는 트리

3) 포와 이진트리의 정의

높이가 k인 이진 트리가 '0 레벨'부터 'k-2'까지 다 채우고 마지막 'k-1 레벨'에서 왼쪽부터 오른쪽으로 노드들이 차례로 채워진 이진트리

5. 이진트리 구현

1) 배열을 이용한 이진트리의 구현

① 트리가 완전 이진트리 또는 포화 이진트리인 경우 낭비되는 공간이 없어 효율적임

② 트리가 깊어질수록 기억장소 낭비가 2의 거듭제곱에 비례하며 심해짐

2) 포인터를 이용한 이진트리의 구현

포인터를 이용한 이진 트리의 노드 생성

6. 이진트리 연산

1) 이진트리 순회

이진트리의 각 노드를 (빠짐없이 그리고 중복 없이)한 번씩 방문하는 것

2) 이진트리의 순회 단위
① 루트 방문(P)
② 왼쪽 서브트리 순회(L)
③ 오른쪽 서브트리 순회(R)

3) 이진트리의 순회 종류
(1) 전위 순회(PLR)
① 루트를 방문
② 왼쪽 서브트리를 전위 순회로 순회
③ 오른쪽 서브트리를 전위 순회로 순회

```
struct node *nodeptr ;
void preorder(struct node *tree_ptr) {
if (tree_ptr) {
printf("%d", tree_ptr->info) ;
preorder(tree_ptr->left) ;
preorder(tree_ptr->right) ;
} }
```

(2) 중위 순회(LPR)
① 왼쪽 서브트리를 중위 순회로 순회
② 루트를 방문
③ 오른쪽 서브트리를 중위 순회로 순회

```
struct node *nodeptr ;
void inorder(struct node *tree_ptr) {
if (tree_ptr) {
inorder(tree_ptr->left) ;
printf("%d", tree_ptr->info) ;
inorder(tree_ptr->right) ;
} }
```

(3) 후위 순회(LRP)
① 왼쪽 서브트리를 후위 순회로 순회
② 오른쪽 서브트리를 후위 순회로 순휘
③ 루트를 방문

```
struct node *nodeptr ;
void postorder(struct node *tree_ptr) {
if (tree_ptr) {
postorder(tree_ptr->left) ;
postorder(tree_ptr->right) ;
printf("%d", tree_ptr->info) ;
} }
```

2) 이진트리 생성, 삽입, 삭제

① 일반적인 이진 트리를 생성하는 것은 이중 연결 리스트 연산을 사용함

② 첫 노드를 생성하면 루트 노드가 되고, 새로운 노드를 추가하려면 이중 연결 리스트의 삽입 연산을 사용함

③ 노드를 삭제할 때, 삭제하려는 노드가 리프노드인 경우는 해당 노드를 가리키는 포인터를 null로 지정하면 됨

④ 리프노드가 아닌 경우에는 삭제하려는 노드의 자식 노드에 대한 처리를 추가로 해주어야 함

3) 이진트리 노드 개수 세기

(1) 이진트리의 노드 개수 세는 연산

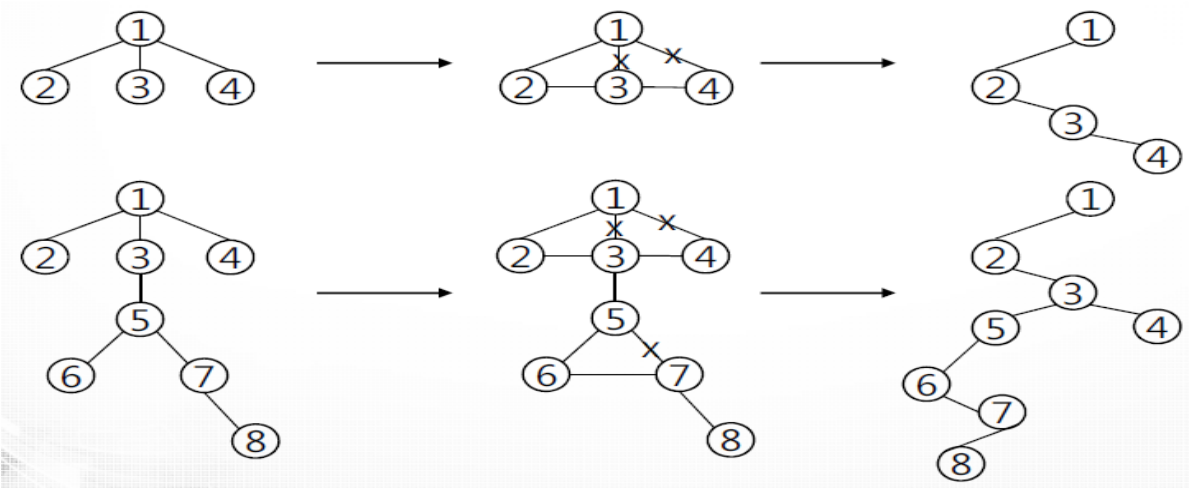
```
int node_count(nodeptr *root) {
    if (root == null) return 0;
    int result = 1;
    result += get_node_count((nodeptr*)root->left)
            +
            get_node_count((nodeptr*)root->right);
    return result;
}
```

(2) 이진트리의 리프 노드 개수 세는 연산

```
int leaf_count(nodeptr *root){
     int result = 0;
     if (root == null) {
         return 0;
     } else if (root->left == null && root->right == null){
         return 1;
     }
     result += get_leaf_count((nodeptr*)root->left)
             +
             get_leaf_count((nodeptr*)root->right);
     return result;
}
```

6. 일반 트리를 이진트리로 변환

① 주어진 트리에 대하여 각 노드의 형제들을 연결함
② 각 노드에 대하여 가장 왼쪽 링크만 남기고 모두 제거함
③ 루트 노드는 반드시 왼쪽 자신 하나만 가지도록 함



<7장 출제예상문제>

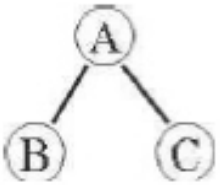
1. 다음 빈 칸에 들어갈 말은?

깊이가 k이고 노드 수가 n인 이진트리가 만일 이 트리의 각 노드들이 깊이 k인 포화 이진 트리에서 1부터 n까지의 번호를 붙인 노드들과 1대 1로 일치하면, 이 트리는 ( )이다.

① 경사 이진 트리
② 균형 이진 트리
③ 완전 이진 트리
④ 스레드 이진 트리

[정답] 3
[해설] 완전 이진 트리는 깊이가 k이고 노드 수가 n인 이진트리가 만일 이 트리의 각 노드들이 깊이 k인 포화 이진트리에서 1부터 n까지의 번호를 붙인 노드들과 1대 1로 일치하면, 이 트리는 완전 이진 트리이다. 또 그 역도 성립한다.

2. 이진트리의 전위 순회, 중위 순회, 후위 순회 결과를 순서대로 기술한 것은?



① BAC, ABC, BCA
② ABC, BAC, BCA
③ BCA, ABC, BAC
④ ABC, BCA, BAC

[정답] 2
[해설] 전위순회 ABC, 중위 순회 BAC, 후위순회 BCA이다.

3. 다음 중 트리 차수에 대한 설명은?
① 그 트리 내의 노두 차수의 평균 차수를 말한다.
② 그 트리 내의 노드 차수 가운데 최대 차수를 말한다.
③ 그 트리 내의 노드 차수 가운데 최소 차수를 말한다.
④ 그 트리 내의 노드 차수 가운데 가장 많은 차수를 말한다.

[정답] 2
[해설] 어떤 트리의 차수는 그 트리 내의 노드 차수 가운데 최대 차수를 말한다.

4. 다음 중 트리에서 노드의 배열순서가 고정되어 위치의 의미가 중요한 트리는?
① oriented 트리
② balanced 트리
③ ordered 트리
④ heap 트리

[정답] 3
[해설] 트리의 노드들의 배열순서(특히 레벨이 같은 노드들의 좌우 순서)가 고정되어 위치의 의미가 중요한 트리를 순서(ordered)트리라고 하며, 그렇지 않은 트리를 비순서(oriented)트리라고 한다.

5. 다음의 용어의 설명 중 올바르지 않은 것은?
① 루트: 트리에서 부모를 같지 않은 노드
② 진입차수: 트리에 있는 어떤 노드에 대해 그 노드로 들어오는 선의 개수
③ 내부노드: 루트도 잎도 아닌 노드
④ 완전이진트리: 트리의 각 노드를 빠짐없이 한 번씩만 방문하는 것

[정답] 4
[해설] 순회에 대한 설명이다.

6. 다음이 설명하고 있는 것은 무엇인가?

높이가 k인 이진트리가 레벨 0으로부터 k-2까지 다 채우고 마지막 k-1레벨에서 왼쪽부터 오른쪽으로 노드들이 차례로 채워진 트리

① 완전이진트리
② 포화이진트리
③ 순회트리
④ 이진트리

[정답] 1
[해설] 완전이진트리에 대한 설명이다.

7. 다음 중 트리를 구성하는 항목을 무엇이라고 하는가?
① 순회
② 데이터
③ 노드
④ 숲

[정답] 3
[해설] 트리를 구성하는 항목을 노드 혹은 정점이라고 한다.

8. 부모를 갖지 않는 노드는 다음 중 무엇인가?
① 트렉
② 레벨
③ 루트
④ 차수

[정답] 3
[해설] 루트는 트리에서 부모를 갖지 않은 노드이다.

9. 다음이 설명하는 것은 무엇인가?

트리의 각 노드(빠짐없이 그리고 중복 없이) 한 번씩 방문하는 것

① 순회
② 데이터
③ 노드
④ 숲

[정답] 1
[해설] 순회에 대한 설명이다.

10. 더 이상 사용하지 않는 트리의 기억장소를 시스템에 반환하는 연산자는 다음 중 무엇인가?
① Tree Creat( )
② Destroy(Tree)
③ Traverse( )
④ Root( )

[정답] 2
[해설] Destroy(Tree)는 더 이상 사용하지 않은 트리의 기억장소를 시스템에 반환하는 연산자이다.

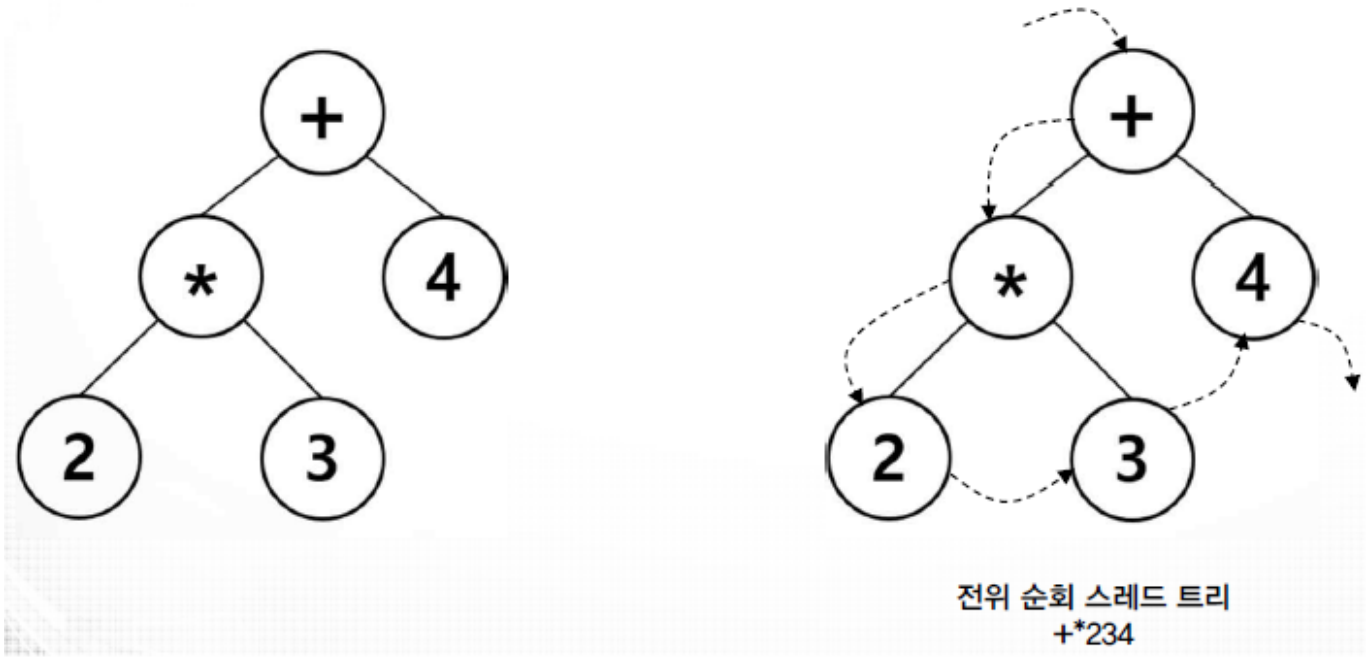
제8장 스레드 트리

1. 스레드 트리

1) 내용

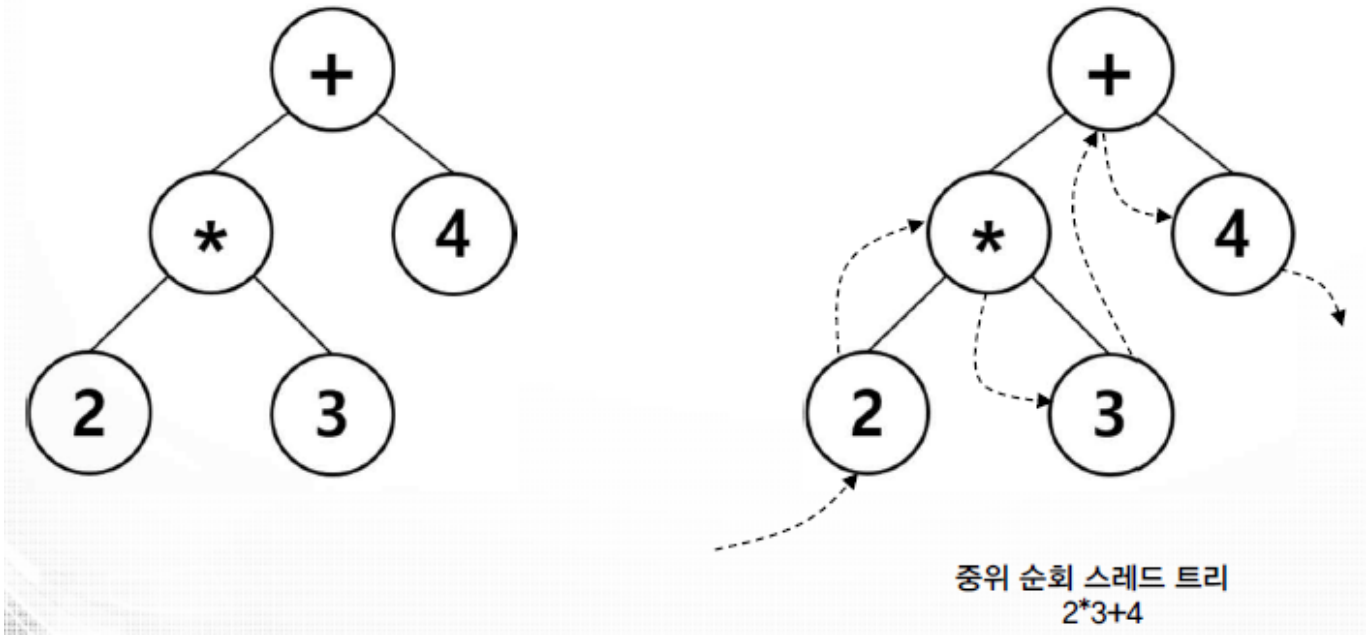
① 스레드 포인터를 갖는 이진트리를 스레드 트리라고 함
② 스레드라는 포인터를 추가하여 트리 순회를 편리하게 한 것임
③ 오른쪽 스레드는 정해진 순해 순서에 따른 그 노드의 후속 노드를 가리킴
④ 왼쪽 스레드는 그 노드의 선행 노를 가리킴
⑤ 스레드 트리는 어떤 한 가지 방법을 적용했을 때 순서를 유지하며 필요하다면 다중 스레드를 사용하여 모든 순회에 따른 순서를 유지하는 스레드 트리를 구성할 수 있음
⑥ 이진 트리의 노드 순회 : 전위 순회, 중위 순회, 후위 순회

<전위 순회>

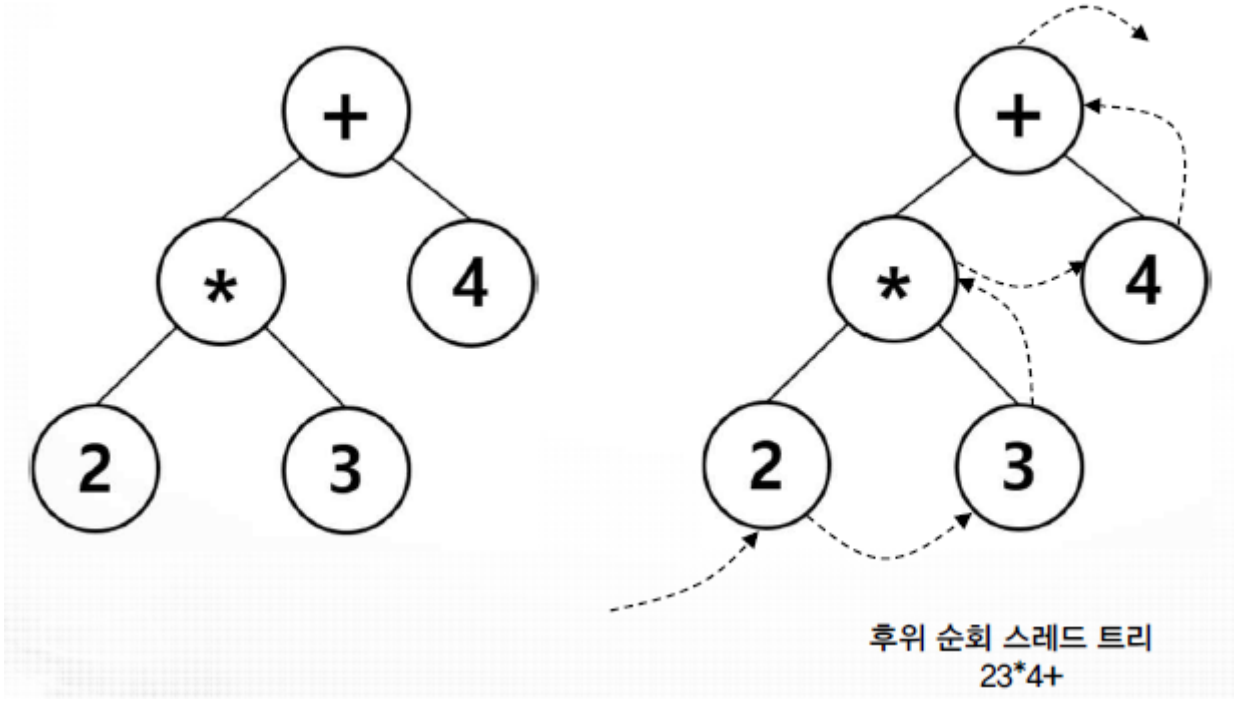


전위 순회 스레드 트리
+*234

<중위 순회>



<후위 순회>



2. 스레드 트리 구현

1) 트리 구형

① 포인터 필드의 추가 : 스레드를 저장하는 포인터를 추가하는 것

- 왼쪽 스레드 포인터, 왼쪽 자식 포인터, 데이터, 오른쪽 자식 포인터, 오른쪽 스레드 포인터 필드로 노드 구조를 정의함

③ 오른쪽 스레드 : 정해진 순회 순서에 따른 그 노드의 후속 노드를 가리키고

④ 왼쪽 스레드 : 그 노드의 선행 노드를 가리킴

⑤ 알고리즘 INSUC(X) : 스레드 이진 트리에서 노드 X의 중위 후속자를 찾는 알고리즘

```
tree_pointer INSUC(tree_pointer X){
// 스레드 이진 트리에서 X의 중위 후속자를 찾음. //
tree_pointer S = X->RLINK
// RTAG(X) = 0이면 S를 RETURN //
// RTAG(X) = 1이면 X의 오른쪽 서브트리의 가장 왼쪽 노드를 찾는다. //
if(X->RTAG == 1){
while(S->LTAG == 1){ // 스레드일 때까지 //
S = S->LLINK // 좌측으로 따라감. //
}
}
return(S)
}
```

- 이진트리의 노드 X에서 RTAG(X) = 0이면 X의 중위 후속자는 스레드 정의에서 RLINK(X)임을 알 수 있음
- RTAG(X) = 1이라면 RLINK(X)는 X의 우측 자식을 가리키는 포인터이므로 X의 중위 후속자는 X의 우측 자식으로부터 시작해서 LTAG가 0이 될 때까지 좌측 자식 링크를 따라가면 찾을 수 있음
- 중위 선행자에 관한 정보나 스택의 도움 없이 어떤 노드의 후속자라도 찾을 수 있음
- 스레드 이진 트리에서 모든 노드를 중위 순서로 나열하고 싶다면 INSUC 함수를 여러 번 반복하여 호출하면 될 것

④ 모든 트리는 헤드 노드의 좌측 서브트리이고, 헤드 노드의 RTAG = 1이므로 다음과 같은 TINORDER 함수는 트리의 모든 노드들을 중위 순서로 나열할 수 있음

```
void TINORDER(tree_pointer T){
// 스레드 이진 트리 T를 중위로 순회함. //
tree_pointer HEAD = T
while(1){
T = INSUC(T)
if(T == HEAD)
return
printf("%c", T->DATA)
}
}
```

3. 스레드 트리 순회, 삽입, 삭제

1) 스레드 트리 순회

① 각 노드의 왼쪽 포인터는 스레드가 아니면 실제 왼쪽 자식을 가리키는 포인터임

② 노드 A와 C의 다음 방문대상인 노드 B, D를 가리키고 있음을 알 수 있음

③ 마지막 방문인 노드의 루트 노드 /의 왼쪽 포인터는 null로 두어 함수를 정상적으로 종료할 수 있음

2) 스레드 트리 노드 삽입 및 삭제

① 스레드 트리에 노드를 삽입하는 것은 스레드가 없는 일반 트리보다 더 복잡함

② 스레드 트리에 속한 것으로 삽입할 위치에 있는 노드 X와 삽입할 노드에 대한 포인터를 매개변수로 하는 삽입함수 Insert()를 정의함

③ 내부 노드를 스레드 트리에서 삭제하는 과정은 훨씬 복잡함

④ 실제로 계층관계에 있는 일련의 프로세스가 트리로 관리되고 있다면 최상위 프로세스를 삭제할 때 그에 딸린 자식 프로세스들을 삭제하는 것은 자연스러운 처리임

⑤ 내부 노드를 수시로 삭제해야 하는 응용문제라면 스레드를 포기하는 것이 현명함

<8장 출제예상문제>

1. 다음 용어의 설명으로 올바르지 않은 것은?

① 스레드: 정해진 순회 방법에 따른 방문 순서를 유지한느 포인터

② 스레드 트리: 정해진 순회 순서에 따른 그 노드의 후속 노드

③ 왼쪽 스레드: 그 노드의 선행 노드를 가리킴

④ 2n-(n-1)=n+1: 이진트리의 null 포인터 개수

[정답] 2

[해설] 스레드 트리는 스레드라는 포인터를 갖는 이진트리이다.

2. 다음이 설명하는 것은 무엇인가?

정해진 순회 방법에 따른 방문 순서를 유지하는 포인터를 뜻하는 말이다.

① 스레드

② 스레드 트리

③ 왼쪽 스레드

④ 오른쪽 스레드

[정답] 1

[해설] 스레드의 정의이다.

3. 다음 중 스레드가 가리키는 것은 무엇인가?
① 포인트 필드
② 데이터 필드
③ 오른쪽 스레드 포인터 필드
④ 같은 구조를 갖는 다른 노드

[정답] 4
[해설] 모든 포인터는 같은 구조를 갖는 다른 노드들을 가리킨다.

4. 노드 포인터 firstin을 매개변수로 하는 함수의 이름은 다음 중 무엇인가?
① while
② info
③ inorder
④ right_thread

[정답] 3
[해설] Inorder는 순회할 트리의 루트 노드를 가리키는 노드포인터 firstin을 매개변수로 하는 함수이다.

5. 다음 중 왼쪽 스레드가 가리키는 것은 무엇인가?
① 순회순서
② 선행노드
③ 후속노드
④ 현재노드

[정답] 2
[해설] 왼쪽 스레드는 그 노드의 선행노드를 가리킨다.

6. 노드가 n개인 이진 트리를 연결리스트로 구현할 때 null 포인터의 값은?
① 2n=1
② 2n-(n-1)=1
③ 2n-(n-1)=n+1
④ n-(n-1)=n+1

[정답] 3
[해설] null 포인터는 항상 2n-(n-1)=n+1개가 존재한다.

7. 각 노드에 대해 포인터가 스레드로 사용 중인지 아니면 서브트리에 대한 포인터인지를 구분하기 위해 사용되는 것은 무엇인가?

① 왼쪽 스레드
② 오른쪽 스레드
③ 루트 노드
④ Tag 필드

[정답] 4
[해설] Tag 필드에 대한 설명이다.

8. 다음 노드의 방문 순서를 가진 순회 방법은 무엇인가?

/ + * + AB/CD%EFG

① 전위 순회
② 중위 순회
③ 후위 순회
④ 반복 순회

[정답] 1
[해설] 전위 순회하는 노드의 방문 순서이다.

9. 다음 중 스레드 트리 순회에서 사용하지 않는 필드는 무엇인가?

① 왼쪽 스레드
② threadFlag 필드
③ 루트 노드
④ Tag 필드

[정답] 2
[해설] 단순하게 트리를 순회하는 스레드 트리에서는 threadFlag 필드는 사용하지 않는다.

10. 오른쪽 스레드가 정해진 순회 순서에 따라 가리키는 것은 다음 중 무엇인가?

① 선행노드
② 반복노드
③ 후속노드
④ 종결노드

[정답] 3
[해설] 오른쪽 스레드는 정해진 순회 순서에 따른 그 노드의 후속 노드를 가리키고, 왼쪽 스레드는 그 노드의

선행 노드를 가리킨다.

제9장 힙

1. 우선순위 큐

1) 힢의 정의

① 피라미드 모양으로 쌓아 올린 더미

② 무엇인가를 쌓아놓은 더미이고 항상 가장 위에 있는 것을 우선 꺼내는 구조

③ 부모-자식 노드사이에서(부분적으로) 정렬된 포화 이진트리로 부모노드는 자식노드보다 우선순위가 높다

2) 우선순위 큐

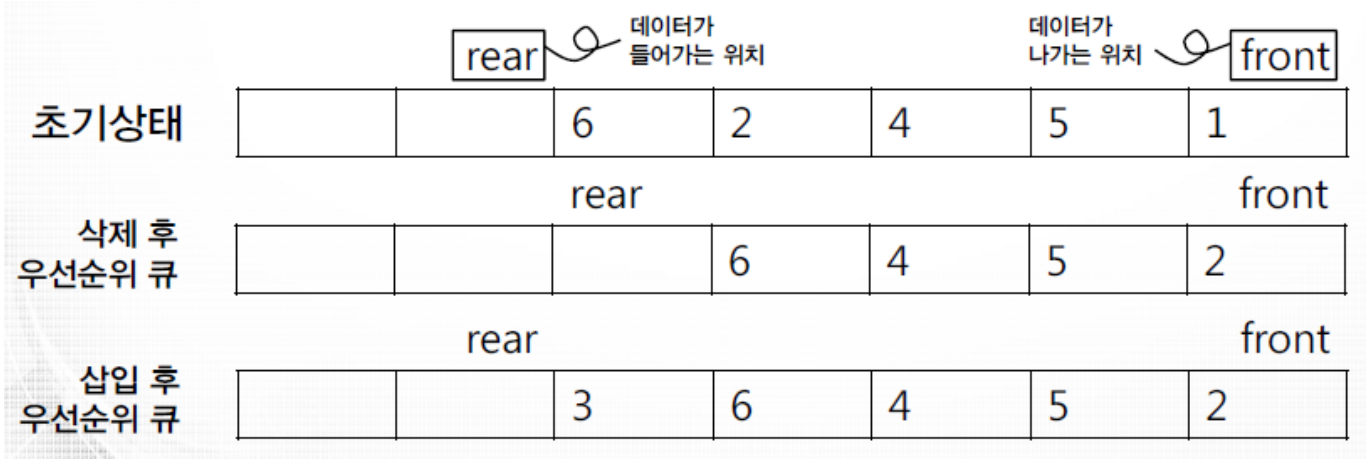
① 큐 : 먼저 들어간 데이터가 먼저 삭제되는 자료구조, 먼저 줄을 선 사람이 먼저 서비스를 받는 구조

② 우선순위 큐 : 대기 리스트에서 항상 우선순위가 높은 것을 먼저 처리하는 구조

3) 우선순위 큐의 배열 구현

(1) 데이터 삭제(Delete_q())와 삽입(Add_q(3))

(2) 데이터 삭제(Delete_q())와 삽입(Add_q(3)) : Delete_q()에 의해 큐의 front에 있던 '1'이 삭제되면서, 나머지 데이터 중에서 가장 작은 값인 '2'가 다음 삭제 위치 즉, front가 가리키는 위치로 이동됨



(3) 우선순위 큐의 작동 방식
① 첫째, 삭제 명령이 실행되면 저장된 데이터 중에서 가장 작은 값(가장 큰 값)이 삭제된다.
② 둘째, 나머지 데이터들은 어떤 순서로 저장되든 문제가 되지 않는다.

2. 힙

1) 힙의 추상자료형
① 힙 객체의 정의: 부분적으로 정렬된 포화 이진트리로 부모노드는 자식노드보다 우선순위가 높다.
② 연산
- inser(element) ::= 힙에 데이터 삽입
- remove( ) ::= 힙(루트)에서 데이터 삭제
- peek( ) ::= 힙(루트)에서 데이터 읽어 오기
- isEmpty( ) ::= 힙이 비었는지 확인
- size( ) ::= 힙에 저장한 데이터 개수 확인

2) 힙의 종류
① 최소힙 : 루트가 전체 노드중에서 최소값인 힙
  - 트리의 모든 노드가 자식 노드보다 작은 값을 가짐
  - 트리의 레벨에 따라 데이터가 순서를 갖지는 않음
  - 탐색 트리처럼 왼쪽 노드와 오른쪽 노드 사이에 크기 제한도 없음
  - 루트가 가장 작은 값을 갖고 부모는 자식보다 작은 값을 가짐
② 최대힙 : 루트가 전체 노드중에서 최소값인 힙
  - 트리의 모든 노드가 자식 노드보다 큰 값을 가짐
  - 트리의 레벨에 따라 데이터가 순서를 갖지는 않음
  - 탐색 트리처럼 왼쪽 노드와 오른쪽 노드 사이에 크기 제한도 없음
  - 루트가 가장 큰 값을 갖고 부모는 자식보다 큰 값을 가지면 됨

3. 힙 노드의 삭제 및 삽입

1) 내용
① 힙은 완전 이진트리이기 때문에 기억장소 낭비가 없음
② 연결리스트로 구현하는 경우 실행 시간 측면에서 비효율적이라는 것이 알려져 있음
③ 힙은 삽입과 삭제가 매우 용이함
④ 우선순위를 관리하는 응용에 흔히 사용하는 중요한 자료구조임
⑤ 힙은 데이터를 정렬하는데도 사용함

2) 힙의 노드 삭제
```
typedef struct {
        int heap[MAX_Data];
        int heap_size;
} HeapType ;

int deleteHeap(HeapType *h) {
```

```
        int parent, child;
        int item, temp;

        item = h->heap[1];
        temp = h->heap[(h->heap_size)];
        parent = 1; child = 2;
        while(child <= h->heap_size) {
                if((child < h->heap_size) && (h->heap[child] > h->heap[child+1]))
                        child++;
                if(temp <= h->heap[child])
                        break;
                h->heap[parent] = h->heap[child];
                parent = child;
                child *= 2;
        }
        h->heap[parent] = temp;
        return item;
}
```

3) 힙의 노드 삽입

```
void insertHeap(HeapType *h, int item) {
        int i;
        i = ++(h->heap_size);  ----> 힙 크기+1
        while((i != 1) && (item < h->heap[i/2])) {  ----> 부모노드와 크기 비교
                h->heap[i] = h->heap[i/2];
                i /= 2;
        }
        h->heap[i] = item;
}
```

<9장 출제예상문제>

1. 다음 중 루트가 가장 큰 값을 갖고 부모는 자식보다 큰 값을 갖는 완전 이진트리는 다음 중 무엇인가?

① 최소힙
② 최대힙
③ 루트
④ 스레드

[정답] 2
[해설] 최대힙에 대한 설명이다.

2. 다음이 설명하는 것은 무엇인가?

대기 리스트에서 항상 우선순위가 높은 것을 먼저 처리한다.

① 최소힙
② 최대힙
③ 노드
④ 우선순위 큐

[정답] 4
[해설] 우선순위 큐에 대한 설명이다.

3. 다음의 빈칸에 들어갈 알맞은 연산은 다음 중 무엇인가?

① inser(element) ::= 힙에 데이터 삽입 ② remove( ) ::= 힙(루트)에서 데이터 삭제 ③ peek( ) ::= 힙(루트)에서 데이터 읽어 오기 ④ [ ] ( ) ::= 힙이 비었는지 확인 ⑤ size( ) ::= 힙에 저장한 데이터 개수 확인

① front
② rear
③ isEmpty
④ heap

[정답] 3
[해설] isEmpty에 대한 내용이다.

4. 다음 중 힙의 연산에서 힙에서 데이터를 읽어오는 연산은 무엇인가?
① inser(element)
② remove( )
③ peek( )
④ size( )

[정답] 3
[해설] peek( )는 힙(루트)에서 데이터 읽어오는 연산이다.

5. 다음 주 힙의 삭제 함수는 무엇인가?
① while
② deleteHeap
③ return item
④ HeapType

[정답] 2
[해설] 삭제 함수 deleteHeap( )은 힙 데이터를 저장한 구조체 포인터를 인수로 받는다.

6. 다음 중 힙에 저장한 데이터 개수 확인하는 연산은 무엇인가?
① inser(element)
② remove( )
③ peek( )
④ size( )
[정답] 4
[해설] Size에 대한 설명이다.

7. 힙의 데이터 삽입 함수는 다음 중 무엇인가?
① while
② deleteHeap
③ inserHeap
④ HeapType

[정답] 3
[해설] 힙의 삽입의 경우는 그 데이터를 삽입 함수 insertHeap( )에 넘겨주어야 한다.

8. 다음이 설명하는 것은 무엇인가?

무엇인가를 쌓아 놓은 더미라고 할 수 있다. 항상 가장 위에 있는 것을 우선 꺼내는 구조를 상징한다.

① 힙
② 우선순위 큐
③ 루트
④ 스레드

[정답] 1
[해설] 힙은 무엇인가를 쌓아 놓은 더미이고, 항상 가장 위에 있는 것을 우선 꺼내는 구조를 상징한다. 그리고 힙은 우선순위 큐의 한 종류이다.

9. 다음의 설명에서 올바르지 않은 것은?
① 힙은 완전 이진 트리이며, 최소힙은 루트가 최소값이고 최대힙은 최대값이다.
② 최대힙은 트리의 모든 노드가 자식 노드보다 큰 값을 갖는 것이다.
③ 최소힙은 트리의 모든 노드가 자식노드보다 작은 값을 갖는 것이다.
④ 힙에서 노드를 삽입한 후에는 단일 트리가 되어야 하며 최대힙 혹은 최소힙의 조건을 만족해야 한다.

[정답] 4
[해설] 힙에서 노드를 삭제한 후에도 완전 이진 트리이어야 하며 최대힙 혹은 최소힙의 조건을 만족해야 한다.

10. 힙의 트리 형태로 올바른 것은?
① 이진트리
② 단일 트리
③ 선택 트리
④ 숲

[정답] 1
[해설] 힙은 완전 이진 트리이다.

제10장 선택트리, 숲, 이진트리 개수

1. 선택트리

1) 합병 정렬

① 합병 정렬 : 차례로 정렬된 데이터 리스트 k 개를 완전한 순서를 유지하는 하나의 리스트로 만드는 과정

② 일반적으로 데이터 리스트가 k 개인 경우 k-1 번 비교함

③ 선택 트리를 이용하여 비교 횟수를 줄일 수 있음

2) 승자트리

① 승자트리는 각 노드가 두 자식 노드보다 더 작은 값을 갖는 완전 이진 트리임

② 승자트리 구축 과정은 작은 값이 승자로 올라가는 토너먼트 경기와 유사함

③ 트리의 각 내부 노드는 두 자식 노드 값의 승자를 자신의 값으로 함

④ 루드 노드는 전체 토너먼트 승자, 즉 트리에서 가장 작은 값을 갖는 노드가 됨

3) 패자트리

① 각 노드가 두 자식 노드 중 더 작은 값을 갖는 완전 이진 트리라는 점은 승자 트리와 같음

② 잎 노드들이 각 리스트의 head가 가리키는 값을 갖는다는 점도 승자트리와 같음

③ 패자트리는 루트 노드 위에 최상위 0번 노드를 가짐

④ 0번 노드에 경쟁에서 이긴 최종 승자를 저장함

⑤ 내부 노드에는 토너먼트 패자를 저장함

⑥ 패자를 부모 노드에 저장하고 승자는 부모의 부모 노드로 올라가서 다시 경쟁함

⑦ 루트에는 마지막 토너먼트 패자를 저장하고 최종 승자는 문제의 0번 노드에 저장하는 것임

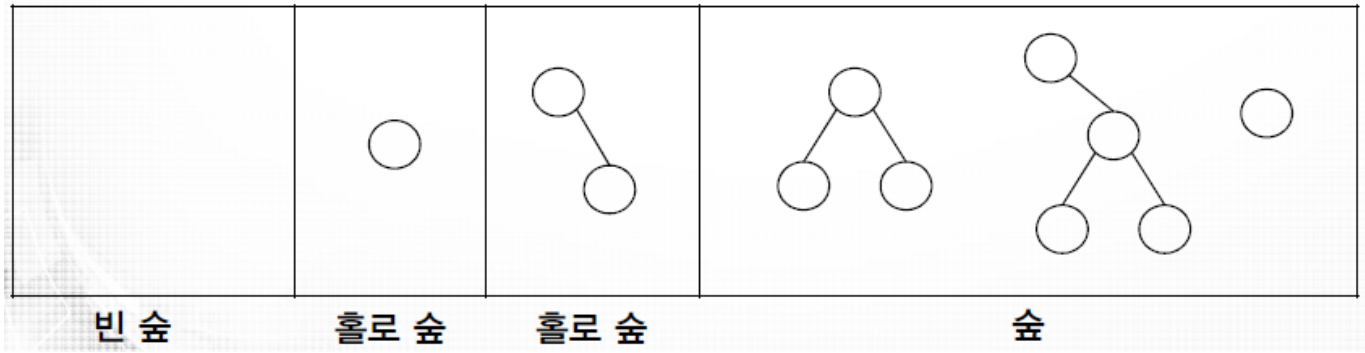
2. 숲

1) 숲의 정의

① 분리된 트리 모임

② 트리의 모임

③ 0개 이상의 분리된 트리 집합



④ 숲 : n (n≧0)개 이상의 분리된 트리 집합

⑤ 트리에서 루트(혹은 다른 노드)를 제거하면 숲을 쉽게 얻을 수 있음

⑥ 반대로 숲을 연결하면 트리를 만들 수도 있음

2) 숲의 이진트리 변환

트리 T1, T2, ... , Tn으로 구성된 숲에 대한 이진트리 BTi~n는 다음과 같다.

1) n = 0 이면 BT0는 빈 이진트리

2) n = 1 이면 BT1 = T1 이면

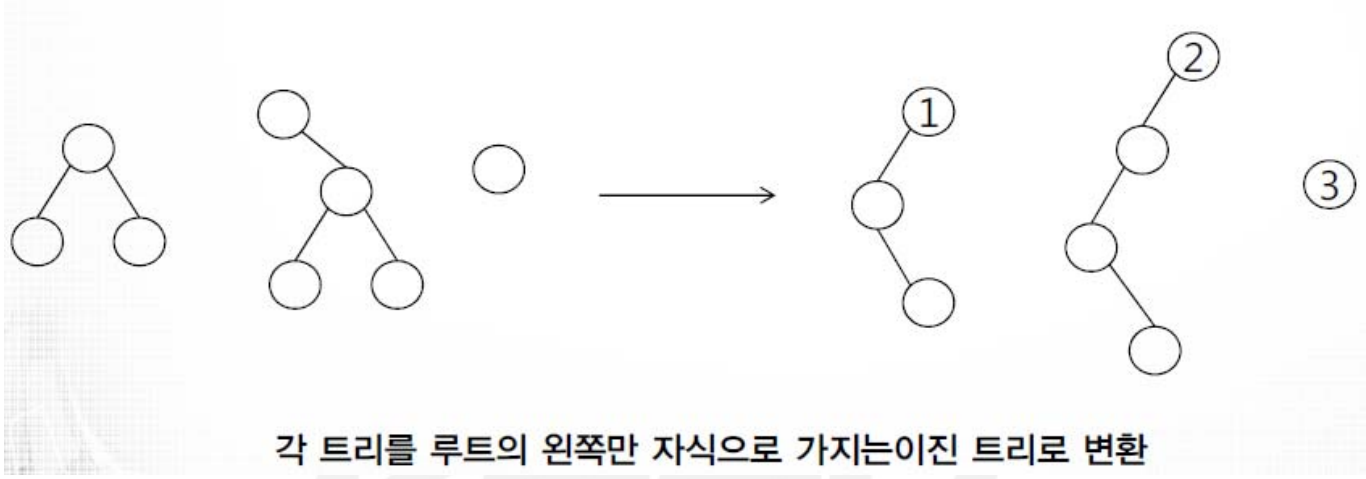
3) n ≥ 2 이면

```
                rootT1
                /  \
      T1 왼쪽 서브트리     BT2~n
```

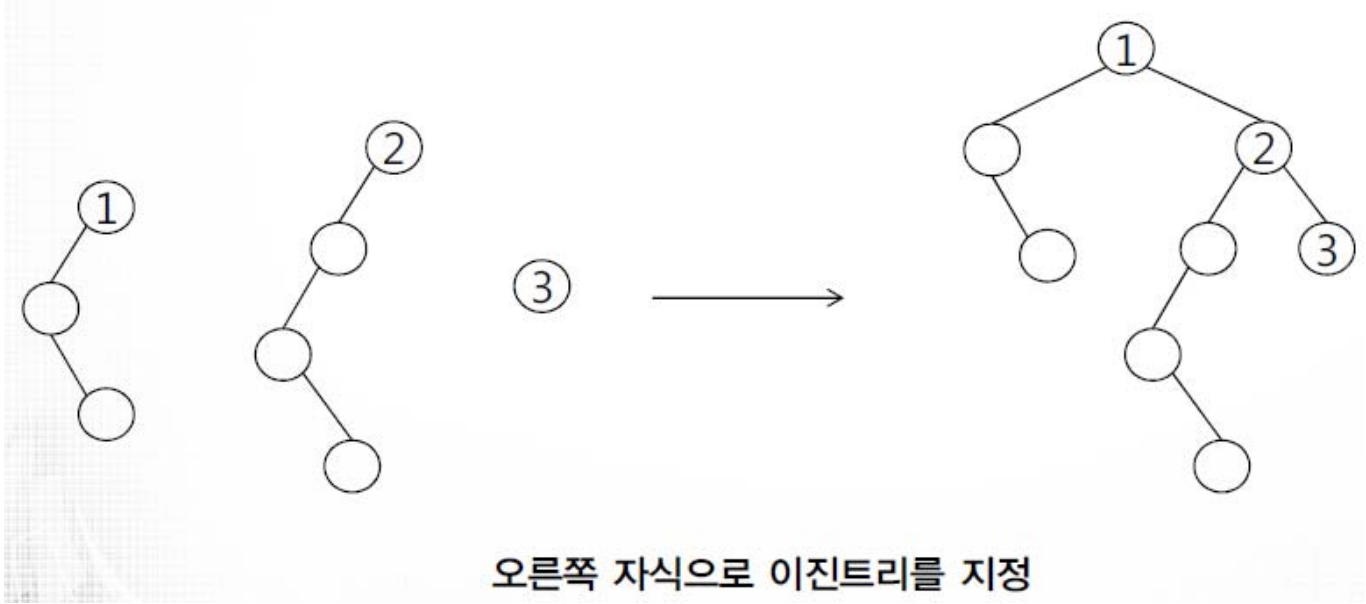
여기서 Ti 는 트리 Ti 를 (루트가 왼쪽 서브 트리만 갖도록) 이진 트리로 변환한 것이다.

① 먼저 각 트리 (T_i)를 이진 트리(T_i^{BT})로 바꿈 (이때 T_i^{BT} 의 루트는 왼쪽 서브트리만 가짐)

② 다음은 T_i^{BT} 의 루트를 최상위 루트로 하고, 왼쪽 자식은 그 왼쪽 서브트리, 오른쪽 자식은 나머지들의 이진 트리(BT_{2-n})가 되도록 함



각 트리를 루트의 왼쪽만 자식으로 가지는이진 트리로 변환



오른쪽 자식으로 이진트리를 지정

3. 이진트리 개수

① 노드가 어떤 값을 갖느냐와 무관하게 가능한 구조의 이진트리가 몇 개나 있는가를 묻는 것임
② 어떤 이진트리에 대한 전위-중위 순회 방문 순서가 주어지면 트리 구조를 유일하게 정할 수 있음
③ 1부터 n까지 수를 스택에 넣었다가 가능한 모든 방법으로 삭제하여 생성할 수 있는 경우의 수와 n개의 노드를 가진 상이한 이진트리의 수는 같음
④ 카탈랑이라는 수학자가 노드 n개인 서로 다른 이진트리의 개수가 $\frac{(2n)!}{n!(n+1)!}$과 같음을 증명함

<10장 출제예상문제>

1. 다음 용어의 설명으로 올바르지 않은 것은?
① 합병정렬: 차례로 정렬한 데이터 리스트 k개를 완전한 순서를 유지하는 하나의 리스트로 만드는 과정
② 선택트리: 합병 정렬에 사용하는 특수한 트리
③ 승자트리: 각 노드가 두 자식 노드보다 더 큰 작은 값을 갖는 완전 이진트리
④ 숲: 각 노드가 두 자식 노드보다 더 큰 값을 가지며 최종 승자는 0번 노드에 저장하는 트리

[정답] 4
[해설] ④는 패자 트리에 대한 설명이다.

2. 다음이 설명하는 것은 무엇인가?

분리된 트리 모임, n(≧0)개 이상의 분리된 트리 집합

① 합병 정렬
② 숲
③ 승자트리
④ 패자트리

[정답] 2
[해설] 숲에 대한 설명이다.

3. 각 노드가 두 자식 노드보다 더 작은 값을 갖는 완전 이진 트리는 다음 중 무엇인가?
① 선택트리
② 승자트리
③ 패자트리
④ 단일트리

[정답] 2

[해설] 승자트리에 대한 설명이다.

4. 어떤 이진트리에 대한 전위-중위 순회 방문 순서가 주어지면 정할 수 있는 트리의 구조는 몇 개인가?
① 1개
② 2개
③ 4개
④ 6개

[정답] 1
[해설] 어떤 이진트리에 대한 전위-중위 순회 방문 순서가 주어지면 트리 구조를 유일하게(한개) 정할 수 있다.

5. 합병 정렬에 사용하는 특수한 트리는 다음 중 무엇인가?
① 단일트리
② 승자트리
③ 선택트리
④ 패자트리

[정답] 3
[해설] 합병정령에 사용하는 특수한 트리가 선택트리이다.

6. 다음이 설명하는 것은 무엇인가?

차례로 정렬한 데이터 리스트 k개가 있다고 가정할 때 그것들은 완전한 순서를 유지하는 하나의 리스트로 만드는 과정

① 합병 정렬
② 숲
③ 승자트리
④ 패자트리

[정답] 1
[해설] 합병정렬에 대한 설명이다.

7. 숲을 이진트리로 바꾸려면 선행되어야 하는 것은 무엇인가?
① 0번 노드에 경쟁에서 이긴 최종 승자를 저장한다.
② 각 트리를 이진트리로 바꾼다.
③ 오른쪽 자식은 나머지들의 이진트리가 되도록 함

④ 트리의 각 내부 노드에서 자식 노드 값의 승자를 자신의 값으로 정한다.

[정답] 2
[해설] 숲을 이진트리로 바꾸려면 먼저 각 트리를 이진트리로 바꾼다.

8. 다음 중 트리에서 숲을 쉽게 얻을 수 있는 방법은 무엇인가?
① 숲을 연결하면서 트리를 만든다.
② 리스트 head에 위치한 값을 패자트리에 올려 보내고 경쟁 시킨다.
③ 트리에서 루트를 제거한다.
④ 트리를 재구성한다.

[정답] 3
[해설] 트리에서 루트를 제거하면 또는 노드를 제거하면 숲을 쉽게 얻을 수 있다.

9. 노드 n 개인 서로 다른 이진트리의 개수를 증명한 학자는 다음 중 누구인가?
① 피타고라스
② 멘델
③ 뉴턴
④ 카탈랑

[정답] 4
[해설] 카탈랑이라는 수학자가 노드 n개인 서로 다른 이진트리의 개수가 $\frac{(2n)!}{n!(n+1)!}$ 과 같음을 증명하였다.

10. 패자트리에서 최종 승자를 저장하는 노드는 몇 번인가?
① 1
② 2
③ 0
④ -1

[정답] 3
[해설] 패자트리에서 최종 승자는 0번 노드에 저장한다.

제11장 이진탐색, Splay, AVL, BB 트리

1. 이진 탐색 트리(BS 트리)

1) 이진 탐색 트리

① 이진 탐색 트리 : 트리에 특정 데이터가 있는지를 검색하고, 노드를 자주 삽입, 삭제하는 응용문제에 가장 효과적인 이진 트리 - BS 트리

② 탐색에 최적화된 이진 트리

③ 키 : 이진 트리의 노드의 데이터

2) BS 트리 탐색

① BS 트리는 탐색에 최적화 된 트리임

② 한 번의 비교로 전체 데이터의 절반을 탐색 대상에서 제외하므로 매우 빠르게 탐색을 수행할 수 있음

3) BS 트리 순회

(1) BS 트리를 위한 노드 정의

```
struct KNode {
        struct KNode *left ; // 왼쪽 자식을 위한 포인터
        char key[10] ; // 키값을 위한 문자 배열
        int info ; // 데이터 필드
        struct Knode *right ; // 오른쪽 자식을 위한 포인터
}
```

(2) BS 트리의 중위 순회

```
void Inorder(struct KNode *rootptr) {
        if(rootptr ! = null) {
                Inorder(rootptr->left) ;
                printf("%d", rootptr->info) ;
                Inorder(rootptr->right) ;
        }
}
```

3) BS 트리의 노드 탐색

```
struct KNode *Search (char k[], struct KNode *r) {
        If (r == null)
                return(null) ;
        else
                If (strcmp(k, r->key) == 0)
                        return(r) ;
                else if (strcmp(k, r->key) < 0) // k < r -> key
                        return(Search(k, r->left)) ;
                        else return(Search(k, r->right)) ;
}
```

4) BS 트리의 노드 삽입 연산

```
struct KNode *Insert(struct KNode *newptr, struct KNode *r) {
        if(r == null)
                return(newptr) ;
        else
                if(strcmp(newptr->key, r->key) == 0)
                        DUPLICATE_ENTRY() ;
                else
                        if(strcmp(newptr->key, r->key) < 0)
                        r->left = Insert(newprt, r->left);
                        else
                        r->right = Insert(newptr, r->right);
        return(r) ;
}
```

5) BS 트리의 노드 삭제

① 자식을 하나만 갖는 노드를 삭제하는 경우, [삭제한 노드를 가리키던 포인터]가 [그 노드의 null이 아닌 서브트리의 루트]를 가리키게 할당함 Ex) 노드 H나 I의 삭제

② 두 개의 자식 노드를 가지는 노드는 삭제 시 항상 특정 방향의 자식 노드를 올리는 방법으로 구현 (즉, 항상 오른쪽(혹은 왼쪽) 자식 노드를 삭제 위치로 이동) Ex) 노드 L이나 F의 삭제

2. Splay 트리

1) 이진 탐색 트리의 단점

① BS 트리의 성능은 트리의 구조와 특정 노드에 접근할 확률에 영향을 받음

② 트리의 성능이 구조에 영향을 받기 때문에 어떤 정점이 탐색될지, 삽입되거나 삭제될지에 대한 정보를 예측할 수 없는 상황에서는 최적의 BS 트리 구조를 결정하기 어려움

③ 휴리스틱 알고리즘을 사용하여 구축한 BS 트리의 성능이 최적에 가까움

2) 이진 탐색 트리의 성능

좋은 성능의 BS 트리를 구축하는 방법 (휴리스틱)

- 자주 탐색하는 키를 가진 노드를 트리의 루트에 가깝게 놓는다.
- 트리가 균형(balance)이 되도록 유지한다. 즉, 각 노드에 대해 왼쪽과 오른쪽 서브트리가 가능한 한 같은 수의 노드를 갖게 한다.

3) Splay 트리

① 자주 탐색하는 키를 가진 노드를 루트에 가깝게 위치하도록 구성한 BS 트리

② Splay 연산 : 최근에 (사용하려고) 접근한 노드 x를 (곧 사용할 것으로 보고) 루트에 위치시켜 트리를 재구성하는 연산

③ Splay 트리 : 가장 최근에 사용한 노드 x가 트리의 루트에 올 때까지 Splay 연산을 반복하여 적용하여 이진 트리를 구축함

4) Splay 트리의 연산

노드 x는 최근에 접근한 노드, 노드 p는 x의 부모 노드, g는 x의 조부모(부모의 부모) 노드

① Zig : p가 트리의 루트면 p와 x의 간선 연결(edge joining)을 회전시킨다.

② Zig-Zig : p가 루트가 아니고 x와 p가 동일하게 왼쪽 자식들 또는 오른쪽 자식들이면 p와 조부모 g와의 간선 연결을 회전시키고 그 다음에 x와 p의 간선 연결을 회전시킨다.

③ Zig-Zag : 만약 p가 루트가 아니고 x가 왼쪽 자식, p가 오른쪽 자식(또는 그 반대)이면 x와 p의 간선 연결을 회전시키고 그 다음에 x와 x의 새로운 부모 p와의 간선 연결을 회전시킨다.

3. AVL 트리

① AVL은 이 개념을 제안한 러시아 수학자의 이름 앞 글자를 따서 붙인 것임

② 트리는 거의 완전한 균형 트리의 한 형태로 높이가 균형 잡힌 높이 균형트리임

③ AVL 조건: 노드 v_i 의 왼쪽 서브트리 높이와 v_i의 오른쪽 서브트리 높이가 최대 1만큼 차이가 남

④ 직접 탐색 성능이 매우 좋음

⑤ 완전히 균형 잡힌 트리에 거의 가까운 좋은 성능을 가짐

⑥ 삽입이나 삭제 수행이 AVL 트리의 균형을 깨면 균형 잡힌 상태로 다시 재구성해야 함

4. BB 트리

① 무게가 균형 잡힌 무게 균형트리라고도 부름

② 노드의 양쪽 서브트리 무게가 균형을 유지하는 트리임

③ 무게가 균형 잡힌 트리의 기본 개념은 완전히 균형 잡힌 것으로부터 얼마나 벗어날 수 있는가에 제한 조건을 준 것임

④ 서브 트리의 높이가 아닌 서브트리의 노드 개수에 제한을 둔 것임

⑤ BB 트리는 균형을 제어하기 위해 인수 β를 사용함

⑥ BB 트리의 평균 탐색 길이 역시 이상적인 균형트리와 유사함

<11장 출제예상문제>

1. AVL에 대한 다음 설명 중 맞는 것은?
① 삽입 연산은 단순히 원소를 삽입하는 것으로 끝나는 것이 아니라 트리의 균형이 깨질 때 그것을 복구해야 한다.
② AVL 트리는 m-원 탐색 트리이다.
③ 루트 노드에서 단말 노드까지의 경로의 길이는 모두 같아야 한다.
④ 왼쪽 서브트리의 높이와 오른쪽 서브트리의 높이 차는 2 이상이다.

[해설] AVL 트리는 모든 노드들이 AVL 성질을 만족하는 2-원 탐색 트리이며 루트 노드의 왼쪽 서브트리의 높이와 오른쪽 서브트리의 높이 차이가 1이기 때문에 루트 노드는 AVL 성질을 만족한다.
[정답] 1

2. 다음의 용어의 설명으로 올바르지 않은 것은?
① 트리의 무게: 트리에 속한 잎 노드의 개수
② 트리의 높이: 각 노드를 식별하기 위해 별도의 간단한 이름을 붙여 준 것
③ 이진탐색트리: 빠르게 탐색할 수 있도록 구성한 이진트리
④ Splay 트리: 자주 탐색하는 키를 가진 노드를 루트에 가깝게 위치하도록 구성한 이진 탐색 트리

[정답] 2
[해설] ②는 키에 대한 설명이다.

3. 다음이 설명하는 것은 무엇인가?

자주 탐색하는 키를 가진 노드를 루트에 가깝게 위치하도록 구성한 이진 탐색 트리

① 트리의 무게
② 트리의 높이
③ 이진탐색트리
④ Splay 트리

[정답] 4
[해설] Splay 트리에 대한 설명이다.

4. 노드가 가질 수 있는 가장 높은 레벨에 1을 더한 값으로 루트로부터 잎까지의 가장 긴 경로 길이는 다음 중 무엇인가?
① 트리의 무게
② 트리의 높이

③ 이진탐색트리
④ Splay 트리

[정답] 2
[해설] 트리의 높이에 대한 설명이다.

5. 다음의 설명하는 트리는 무엇인가?

노드v_i의 왼쪽 서브트리 높이와 v_i의 오른쪽 서브트리 높이가 최대 1만큼 차이가 난다는 조건을 만족하는 트리

① AVL 트리
② 트리의 높이
③ 이진탐색트리
④ Splay 트리

[정답] 1
[해설] AVL 트리에 대한 설명이다.

6. BS 트리의 성능에 영향의 미치는 요인은 다음 중 무엇인가?
① 휴리스틱
② 트리의 구조
③ 시간
④ 속도

[정답] 2
[해설] BS트리의 성능은 트리의 구조와 특정 노드에 접근할 확률에 영향을 받는다.

7. 다음 중 무게가 균형 잡힌 트리에 속하는 트리는 무엇인가?
① AVL 트리
② BB 트리
③ 이진탐색트리
④ Splay 트리

[정답] 2
[해설] BB트리는 무게균형트리에 속한다.

8. Splay 트리가 트리를 재구성하기 위해 노드 x를 어디에 위치시키는 곳은 무엇인가?
① 루트

② 인덱스
③ 힙
④ 스레드

[정답] 1
[해설] 루트에 대한 설명이다.

9. 각 노드를 식별하기 위해 별도의 간단한 이름을 붙여 준 것으로 노드의 데이터라고 생각할 수 있는 것은 다음 중 무엇인가?
① 루트
② 키
③ 힙
④ 스레드

[정답] 2
[해설] 키는 각 노드를 식별하기 위해 별도의 간단한 이름을 붙여 준 것으로 노드의 데이터라고 생각 할 수 있다.

10. 다음이 설명하는 것은 무엇인가?

트리에 특정 데이터가 있는지를 검색하고, 노드를 자주 삽입, 삭제하는 응용문제에 가장 효과적인 이진트리이다.

① AVL 트리
② BB 트리
③ 이진탐색트리
④ Splay 트리

[정답] 3
[해설] 이진탐색트리에 대한 설명이다.

제12장 멀티웨이 탐색 트리 I

1. m원 탐색 트리

1) 내용

① 이진 탐색 트리를 확장한 것임

② 탐색트리의 제한을 따르되 두 개 이상 자식을 가질 수 있는 것임

③ 탐색트리의 요건, 즉 왼쪽은 작고 오른쪽은 큼

④ 실제 응용에서 킷값은 보통 문자열임

⑤ m원 트리를 위한 노드는 중첩 구조체로 정의 함

⑥ 일반적으로 노드의 가지 개수가 많을수록, 즉 서브트리를 많이 가질수록 최대 탐색 길이는 짧아짐

2. B트리

1) B 트리 노드 삽입

① 삽입할 위치를 찾기 위해 노드의 킷값을 좌에서 우로 탐색함

(B트리에서 모든 노드는 잎에서 삽입이 개시됨)

② 노드에 빈자리가 있으면 삽입 후 종료함

③ 노드가 꽉 찼으면 노드를 두 개로 분리하고 키와 포인터를 새 노드에 반씩 할당함

- 잎 노드 키값과 삽입 노드 킷값 중에서 중간 값을 선택함
- 선택한 값보다 작은 키값을 갖는 것은 왼쪽 노드에 넣고 큰 것은 오른쪽 노드에 넣음
- 새 노드의 키와 포인터를 부모 노드에 삽입함, 만일 부모 노드가 루트 노드이면 두 노드를 가리키도록 수정함

2) B 트리 노드 삭제

① 삭제할 키값을 포함한 노드를 찾음

② 잎 노드에서 삭제하는 경우

- 노드에서 키값을 삭제함
- 필요하면 재배열함

③ 내부 노드에서 삭제하는 경우

- 내부노드 킷값은 하위 노드에 대한 기준 값이기 때문에 삭제 시 대체할 수 있는 적절한 값을 찾아야 함
- 보통 왼쪽 서브트리의 가장 큰 킷값 또는 오른쪽 서브트리의 가장 작은 킷값으로 대체할 수 있음
- 새로운 기준 값을 선택하여 해당 노드에서 삭제하고 그 값을 현재 키 값을 삭제한 지리로 옮김
- 기준 값으로 대체하기 위해 키를 삭제한 잎 노드가 정해진 개수의 킷값을 갖지 않으면 트리를 재배열함

3) 재배열

① 킷값이 부족한 노드의 오른쪽 형제가 존재하고 키가 정해진 개수보다 많다면 왼쪽으로 회전함

- 부모 노드의 기준 값을 개수가 부족한 노드의 끝으로 이동함
- 부모 노드의 기준 값을 오른쪽 형제의 첫 번째 키로 수정해 균형을 맞춤

② 킷값이 부족한 노드의 왼쪽 형제가 존재하고 키가 정해진 개수보다 많다면 오른쪽으로 회전함

- 부모 노드의 기준 값을 개수가 부족한 노드의 끝으로 이동함
- 부모 노드의 기준 값을 오른쪽 형제의 첫 번째 키로 수정해 균형을 맞춤

③ 좌우 형제가 최소 개수의 키를 가지고 있다면, 좌우 형제를 합침
- 부모 노드의 기준 값을 가지고 있다면, 좌우 형제를 합침
- 오른쪽 노드의 모든 키값을 왼쪽 노드로 옮김
- 키를 갖지 않는 오른쪽 노드를 삭제함
- 부모 노드가 루트이면서 키를 더 이상 갖지 않으면 합쳐진 노드가 새로운 루트가 됨

3. B^*, B^+ 트리

1) B^* 트리
① 노드의 약 2/3 이상이 차야하는 B트리를 B^* 트리라고 함
② B^* 트리는 노드가 꽉 차면 분리하지 않고 키와 포인터를 재배치하여 다른 형제 노드로 옮김
- 공집합이거나 높이가 1이상인 m원 탐색 트리임
- 루트 노드는 2개 이상 $2\lfloor(2m-2)/3\rfloor+1$ 이하의 자식 노드를 가짐
- 내부 노드는 최소한 $\lceil(2m-1)/3\rceil$ 개의 자식 노드를 가짐
- 모든 단말 노드는 동일한 레벨에 놓임
- 포인터가 $\mathcal{R}$개인 잎이 아닌 노드의 $\mathcal{R}-1$개의 키를 가짐

2) B^+ 트리
① 인덱스 된 순차 파일을 구성하는데 사용하는 것임
② 인덱스 된 순차 파일은 데이터를 차례로 처리하는 순차처리와 특정 데이터를 직접 찾아 처리하는 두 가지를 모두 효율적으로 할 수 있는 구조임
③ B트리와 같이 각 노드가 적어도 1/2가 차야 하는 점은 같으나 잎 노드를 순차적으로 연결하는 포인터 집합이 있다는 점에서 다름
④ 새로운 키값을 삽입하는 과정은 B트리 경우와 거의 같음
⑤ B^+트리에서 키값을 삭제하는 과정은 다른 트리에 비해 비교적 쉬움
⑥ 모든 키값이 잎에 있고 그 킷값에 대응하는 실제 데이터에 대한 주소를 잎 노드만이 가지 고 있음

<12장 출제예상문제>

1. 다음이 설명하는 용어로 올바르지 않은 것은 무엇인가?
① m원 탐색 트리: 인덱스 구조를 구현하는 데 가장 일반적으로 사용하는 방법으로 차수가 m인 트리
② B^+ 트리: 노드의 약 2/3 이상이 차야 하는 B 트리
③ B^* 트리: 모든 키값이 잎에 있고 그 키값에 대응하는 실제 데이터에 대한 주소
④ B 트리: 루트는 최소한 두 개의 서브트리를 갖는 트리이다.

[정답] 1
[해설] B트리에 대한 설명이다.

2. 다음의 B트리에 키를 삽입하는 알고리즘으로 올바르지 않은 것은?
① 노드에 빈자리가 있으면 삽입 후 종료한다.
② 노드가 꽉 찼으면 노드를 두 개로 분리하고 키와 포인터를 새 노드에 반씩 할당한다.
③ 삽입할 위치를 찾기 위해 노드의 키값을 우에서 좌로 탐색한다.
④ 새 노드의 키와 포인터를 부모 노드에 삽입한다.

[정답] 3
[해설] 삽입할 위치를 찾기 위해 노드의 키값을 좌에서 우로 탐색한다 .

3. 노드의 약 2/3 이상이 차야 하는 B트리는 다음 중 무엇인가?
① B^+ 트리
② B^* 트리
③ m원 탐색 트리
④ B 트리

[정답] 2
[해설] B^* 트리에 대한 설명이다.

4. 다음은 무엇에 대한 설명인가?

이진 탐색 트리를 확장한 것으로 탐색 트리의 제한을 따르되 두 개 이상 자식을 가질 수 없는 트리이다.

① B^+ 트리
② B^* 트리
③ m원 탐색 트리
④ B 트리

[정답] 3
[해설] m원 탐색 트리에 대한 설명이다.

5. 다음 중 탐색을 위한 키 값은 무엇인가?
① r
② skey
③ BS
④ AVL

[정답] 2
[해설] skey는 탐색을 위한 키 값이다.

6. 인덱스 된 순차 파일을 구성하는데 사용하는 트리는 무엇인가?
① B^+ 트리
② B^* 트리
③ m원 탐색 트리
④ B 트리

[정답] 1
[해설] B^+ 트리에 대한 설명이다.

7. 다음의 조건을 만족하는 m원 탐색 트리를 무엇이라고 하는가?

- 루트와 단말 노드를 제외한 트리의 각 노드는 최소 $\lceil m/2 \rceil$ 개의 서브트리를 갖는다.
- 트리의 루트는 최소한 두 개의 서브 트리를 갖는다.
- 트리의 모든 단말 노드는 같은 레벨에 있다.

① 차수 m인 B 트리
② 서브트리
③ 16원 트리
④ 이원집합 트리

[정답] 1
[해설] 차수 m인 B 트리에 대한 설명이다.

8. 다음 중 루트 노드를 가리키는 포인터는 무엇인가?
① r
② skey
③ BS
④ AVL

[정답] 1
[해설] r은 루트 노드를 가리키는 포인터이다.

9. 다음은 차수가 m인 B^* 트리에 대한 설명으로 올바르지 않은 것은?
① 공집합이거나 높이가 1이상인 m원 탐색 트리이다.
② 루트 노드는 2개 이상 $2\lfloor(2m-2)/3\rfloor + 1$개 이하의 부모 노드를 갖는다.
③ 모든 단말 노드는 동일한 레벨에 놓인다.
④ 포인터가 k개인 잎이 아닌 노드는 k-1개의 키를 갖는다.

[정답] 2

[해설] 루트 노드는 2개 이상 2⌊(2m-2)/3⌋ + 1개 이하의 자식 노드를 갖는다.

10. 다음이 설명하는 것은 무엇인가?

트리에서 모든 키값이 잎에 있고, 그 키값에 대응하는 실제 데이터에 대한 주소를 잎 노드만이 가지고 있다.

① B^+ 트리
② B^* 트리
③ m원 탐색 트리
④ B 트리

[정답] 1
[해설] B^+ 트리에 대한 설명이다.

제13장 멀티웨이 탐색 트리 II

1. 2-3 트리

1) 2-3 트리

① 차수가 2 또는 3인 내부 노드를 갖는 탐색 트리

② 2-노드와 3-노드만으로 구성되는 특수한 형태의 B 트리임

③ 2-노드 혹은 3-노드라는 제약이 내부 노드에만 해당함

2) 2-3트리의 조건

① 각각의 내부 노드는 2-노드이거나 3-노드이다. 2-노드는 1개의 키값을, 3-노드는 2개의 키값을 갖는다.

② lchild, mchild를 각각 2-노드의 오른쪽 자식 및 중간 자식이라 하고, lkey가 이 노드의 키값이라 하자. 그러면 루트가 lchild인 모든 2-3 서브트리 키값은 lkey 보다 작고, 루트가 mchild인 모든 2-3 서브 트리 키값은 lkey 보다 크다.

③ lchild, mchild 및 rchild를 각각 3-노드의 왼쪽, 중간 및 오른쪽 자식이라 하고, lkey 및 rkey를 이 노드의 두 키값이라 하면, 다음이 성립한다.

ⓐ lkey < rkey

ⓑ 루트가 lchild인 모든 2-3 서브 트리 키값은 lkey 보다 작다.

ⓒ 루트가 mchild인 모든 2-3 서브 트리 키값은 rkey 보다 작고, lkey 보다 크다.

ⓓ 루트가 rchild인 모든 2-3 서브 트리 데이터는 rkey보다 크다.

④ 모든 잎 노드는 같은 레벨에 있다.

3) 2-3트리의 삽입 알고리즘

```
void insert23(two_three_ptr t, int y) {
        two_three_ptr q, p, r ;
        if (! (*t))
                new_root(t, y, NULL) ;
        else {
                p = find_node(*t, y) ;
                if ( !p ) {
                        fprintf(stderr,"The key is currently in the tree₩n") ;
                        exit(1) ;
}

q = NULL ;
        for ( ; ; )
                if ( p->rkey == INT_MAX ) {
                        put_in(&p, y, q) ;
                        break ; }
                else {
                        split(p, &y, &q) ;
```

```
                if (p == *t) {
                        new_root(t, y, q) ;
                        break ; }
                else
                        p = delete(); }
        }
}
```

4) 2-3트리의 삭제

① 2-3 트리의 삭제에서 잎 노드가 아닌 노드의 키를 삭제하면 그곳을 다른 키로 대체해야 함

② 일반적으로 삭제한 키의 왼쪽 서브트리에서 가장 큰 킷값이나 오른쪽 서브트리에서 가장 작은 키값를 선택하여 대체함

2. 2-3-4 트리

1) 내용

① 2-3 트리를 확장하여 4개의 자식을 가진 4-노드를 허용하는 탐색 트리

② 2-3-4 트리는 2-3 트리보다 삽입과 삭제가 용이함

③ 2-3-4 트리에서는 2-노드와 3-노드에 대한 트리 재구성 확률이 2-3 트리에서보다 작기 때문에 삽입 및 삭제 연산을 수행하는 데 더 효율적임

2) 조건

① 각각의 내부 노드는 2-노드, 3-노드 및 4-노드이다. 이때 2-노드는 1개의 키값, 3-노드는 2개의 키값, 4-노드는 3개의 키값을 갖는다.

② lchild, lmchild는 각각 2-노드의 왼쪽 자식 노드 및 좌중간 자식 노드이라 하고, lkey를 키값이라 하자. 그러면 루트가 lchild인 모든 2-3-4 서브트리 키값은 lkey보다 작고, 루트가 rmchild인 모든 2-3-4 서브트리 키값은 lkey보다 크다.

③ lchild, lmchild 및 rmchild를 각각 3-노드의 왼쪽 자식 노드, 좌중간 자식노드 및 우중간 자식 노드라 하고 lkey, mrkey를 이 노드의 키값이라 하자. 그러면 다음이 성립한다.

- Ⓐ lkey < mkey
- ⓑ 루트가 lchild인 모든 2-3-4 서브트리 키값은 lkey 보다 작다.
- Ⓒ 루트가 lmchild인 모든 2-3-4 서브트리 키값은 lkey 보다 크고 mkey보다 작다.
- ⓓ 루트가 rmchild인 모든 2-3-4 서브트리 키값은 mkey보다 크다.

④ lchild, lmchild, rmchild 및 rchild를 각각 4-노드의 왼쪽, 좌중간, 우중간 및 오른쪽 자식이라 하고, lkey, mkey 및 rkey를 이 노드의 키값이라 하면, 다음이 성립한다.

- ⓐ lkey < mkey < rkey
- ⓑ 루트가 lchild인 모든 2-3-4 서브트리 키값은 lkey 보다 작다.
- ⓒ 루트가 lmchild인 모든 2-3-4 서브트리 키값은 mkey 보다 작고 lkey 보다 크다.
- ⓓ 루트가 rmchild인 모든 2-3-4 서브트리 키값은 rkey 보다 작고 mkey 보다 크다.
- ⓔ 루트가 rchild인 모든 2-3-4 서브트리 키값은 rkey 보다 크다.

⑤ 모든 잎 노드들은 같은 레벨에 있다.

3) 2-3-4 트리의 삽입 연산
① 2-3-4 트리에서는 2-노드와 3-노드에 대한 트리 재구성의 확률이 2-3 트리에서 보다 작기 때문에 삽입/삭제 연산이 효율적임
② 삽입 연산 시 문제가 되는 4-노드는 그 위치에 따라 세 가지 경우로 구분하여 분리함
- 4-노드가 루트인 경우
- 4-노드의 부모가 2-노드인 경우
- 4-노드의 부모가 3-노드인 경우

4) 2-3-4 트리의 삭제 연산
① 잎 노드에 있는 키 값은 단순히 삭제하면 됨
② 잎 노드가 아닌 경우에는, 삭제 연산에서 4-노드를 만났을 때 3가지 경우에 따라 노드를 분리해야 함
- 4-노드가 2-3-4 트리의 루트인 경우
- 4-노드의 부모가 2-노드인 경우
- 4-노드의 부모가 3-노드인 경우

3. 레드 블랙 트리
1) 내용
① 2-3-4 트리가 편리하기는 하지만 4-노드를 무조건 분리하기 때문에 채워지지 않은 기억장소가 많을 수 있음
② 2-3-4 트리를 이진트리로 나타낸 것을 레드 블랙 트리라고 함
③ 레드간선, 블랙간선의 서브트리가 있음
④ 레드 간선은 2-3-4 트리의 한 노드 내에 있던 항목이고, 블랙 간선은 2-3-4 트리의 포인터임
⑤ 레드 블랙 트리의 탐색 보통의 이진 탐색 트리 탐색 알고리즘과 동일함

<13장 출제예상문제>

1. 다음 중 차수가 2-노드와 3-노드만으로 구성한 트리는 무엇인가?
① 2-노드
② 3-노드
③ 2-3 트리
④ 2-3-4트리

[정답] 3
[해설] 2-3트리에 대한 설명이다.

2. 다음이 설명하는 것은 무엇인가?

2-3-4 트리를 이진트리로 나타낸 것으로 기억장소를 효율적으로 사용할 수 있다.

① 2-노드
② 3-노드
③ 2-3 트리
④ 레드 블랙 트리

[정답] 4
[해설] 레드 블랙 트리에 대한 설명이다.

3. 다음 용어의 설명으로 올바르지 않은 것은?
① 2-노드: 차수가 2인 노드
② 2-3-4트리: 2-노드와 3-노드만으로 구성한 트리로 B트리 계열과 거의 같은 성능을 유지하면서 상대적으로 삽입 삭제가 용이한다.
③ 3-노드: 차수가 3인 노드
④ 레드 블랙 트리: 2-3-4 트리를 이진트리로 나타낸 것으로 기억장소를 효율적으로 사용할 수 있다.

[정답] 2
[해설] 2-3-4트리는 2-노드, 3-노드, 및 4-노드만으로 구성한 트리로 2-3 트리와 같은 특성이 있다.

4. 2-3 트리의 첫 노드를 생성할 때 호출하는 함수는 다음 중 무엇인가?
① new_root
② put_in
③ rotation
④ rkey

[정답] 1
[해설] new_root는 2-3 트리의 첫 노드를 생성할 때 호출하는 함수이다.

5. 다음의 2-3 트리 연산은 무엇인가?

3-노드인 형제가 있는 경우 키값 중 하나를 부모로 올리고 대신 부모로부터 키값 하나를 가져다 빈 노드를 채우는 것이다.

① 중복
② 순회
③ 회전
④ 삽입

[정답] 3
[해설] 회전에 대한 설명이다.

6. 다음 중 2-3트리를 확장하여 네 개의 자식을 가진 4-노드를 허용하는 트리는 다음 중 무엇인가?
① 2-노드
② 3-노드
③ 2-3 트리
④ 2-3-4 트리

[정답] 4
[해설] 2-3-4 트리에 대한 설명이다.

7. 다음 중 2-3-4 트리의 한 노드 내에 있던 항목은 다음 중 무엇인가?
① 2-노드
② 레드 간선
③ 블랙 간선
④ 서브 트리

[정답] 2
[해설] 레드 간선은 2-3-4트리의 한 노드 내에 있던 항목이다.

8. 다음이 설명하는 것은 무엇인가?

2-3-4 트리의 포인터

① 2-노드
② 레드 간선
③ 블랙 간선
④ 서브 트리

[정답] 3
[해설] 블랙 간선에 대한 설명이다.

9. 4-노드의 위치에 따라 구분되는 경우가 아닌 것은?
① 4-노드가 2-3-4 트리의 루트인 경우
② 4-노드의 부모가 2노드인 경우
③ 4-노드의 부모가 3-노드인 경우
④ 4-노드의 자녀가 3-노드인 경우

[정답] 4
[해설] 4-노드는 그 위치에 따라 ① 4-노드가 2-3-4 트리의 루트인 경우, ② 4-노드의 부모가 2노드인 경우
③ 4-노드의 부모가 3-노드인 경우로 구분 할 수 있다.

10. 노드 p에 y를 삽입하는 함수는 다음 중 무엇인가?

① new_root
② put_in
③ rotation
④ rkey

[정답] 2
[해설] put_in 함수에 대한 설명이다.

제14장 그래프 I

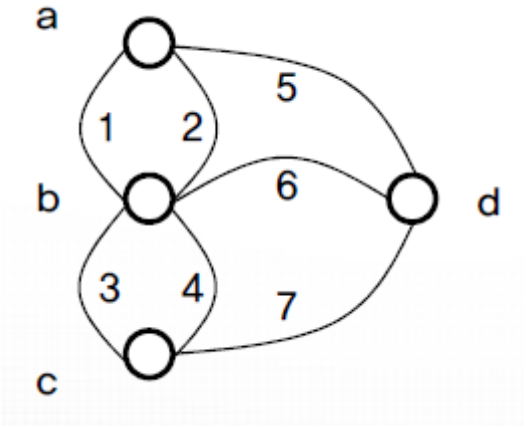
1. 개념 및 용어

1) 그래프의 정의

① 그래프 는 하나 이상의 정점(혹은 노드)을 포함하는 집합 V 와 두 정점의 쌍으로 구성되는 간선을 포함하는 집합 E 의 순서쌍으로 정의함

② 그래프 = (V, E)

- V = {a, b, c, d} (정점의 집합)
- E = {1, 2, 3, 4, 5, 6, 7} (간선의 집합)



2) 그래프의 용어 정의

① 간선은 두 정점을 연결하는 선

② 그래프는 연결(간선)에 방향성이 없는 무방향 그래프와 방향성을 갖는 방향 그래프가 있음

③ 무방향 그래프는 간선이 방향성이 없음

④ 방향 그래프는 간선이 방향성이 있음

⑤ 무방향 그래프의 간선은 실선으로 나타내고 방향 그래프의 간선은 화살표로 나타냄

⑥ 간선은 두 정점 쌍으로 나타냄

⑦ 무방향 그래프일 때는 정점 쌍이 순서를 나타낼 필요가 없으므로 어떤 간선이 두 정점 v_1 , v_2 를 잇는 것이라면 $\{v_1, v_2\}$로 나타냄

⑧ 방향 그래프의 간선은 순서쌍 (v_1, v_2) 로 표시함

3) 다중 그래프

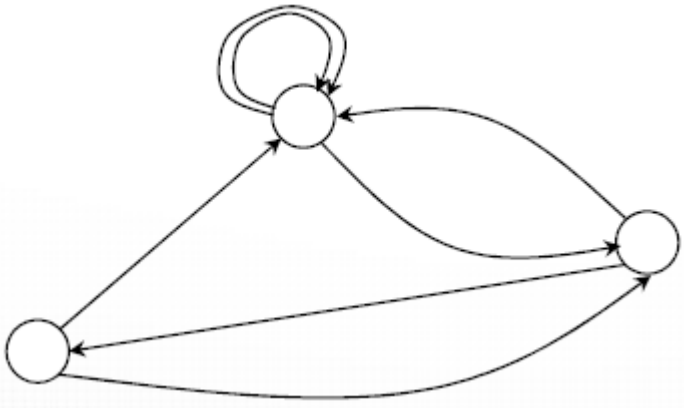
① 방향 다중 그래프

- 다중 그래프 : 두 정점을 잇는 간선이 여러 개인 그래프
- 방향 다중 그래프 : 방향성을 갖는 간선으로 이루어진 다중 그래프

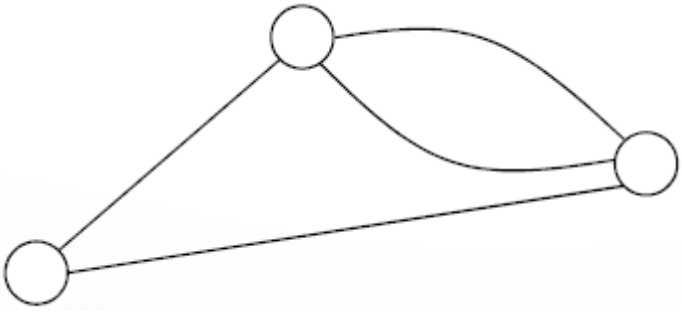
② 무방향 다중 그래프

- 다중 그래프 : 두 정점을 잇는 간선이 여러 개인 그래프
- 무방향 다중 그래프 : 방향성이 없는 간선으로 이루어진 다중 그래프

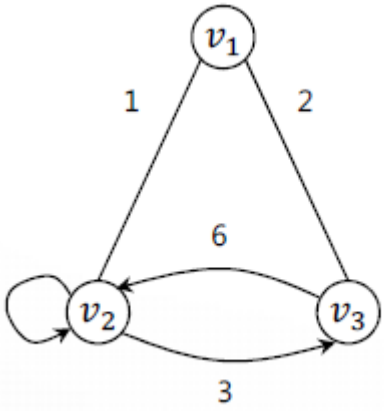
<방향 다중 그래프>

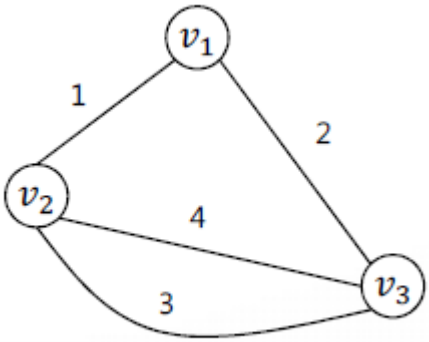


<무방향 다중 그래프>



4) 가중 그래프: 간선이 가중치를 갖는 그래프





5) 그래프의 성질

① n 개의 정점을 갖는 무방향 그래프에서 $v_i \neq v_j$ 인 서로 다른 무순서쌍$\{v_i,\ v_j\}$ 의 최대 개수

$nC_2 = \frac{n(n-1)}{2}$

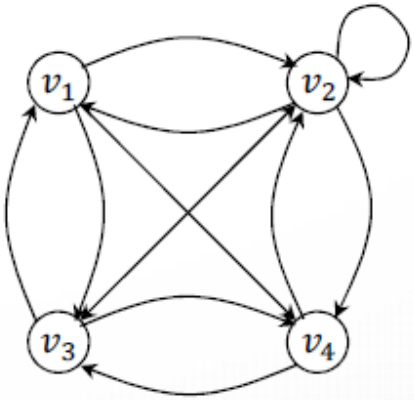
② n 개의 정점을 갖는 방향 그래프에서 $v_i \neq v_j$ 인 서로 다른 순서쌍 $(v_i,\ v_j)$의 최대 개수

$nP_2 = n(n-1)$

③ 완전 그래프(Complete graph) : 모든 정점들이 간선으로 서로 연결된 그래프
④ 완전 그래프인 (만일 정점의 수를 n 이라고 하면) 무방향 그래프의 간선 개수
$\frac{n(n-1)}{2}$
⑤ 두 정점 v_i와 v_j가 서로 인접한다 : 무방향 그래프 간선 e ∈ E 가 $\{v_i, v_j\}$으로 표현될 때 즉, 두 정점 v_i와 v_j를 연결하는 간선인 경우를 말함
⑥ 독립 정점 : 다른 어떤 정점과도 인접하지 않은 정점
⑦ 널(null) 그래프 : 독립 정점만으로 구성한 그래프이며, 간선의 집합 E 는 공집합임 ⇨ 정점만 있는 그래프
⑧ 경로(path) : 임의의 두 정점을 연결하는 어떤 간선의 끝 정점(해당 간선의 머리)이 이어진 간선의 시작 정점(해당 간선의 꼬리)이 되는 간선의 열
⑨ 경로 길이 : 경로에 있는 간선의 수
⑩ 단순 경로 : 경로 상에 있는 모든 정점이 서로 다른 경로
⑪ 기본 경로 : 경로 상에 있는 모든 간선이 서로 다른 경로

6) 방향 그래프 – 사이클을 포함한 경로
① 사이클 : 출발점과 도착점이 동일한 단순 경로
② 사이클 그래프 : 사이클이 있는 그래프

$C_1 = (v_2, v_2)$
$C_2 = (v_1, v_2), (v_2, v_1)$
$C_3 = (v_2, v_3), (v_3, v_1), (v_1, v_2)$
$C_4 = (v_2, v_1), (v_1, v_4), (v_4, v_3), (v_3, v_2)$



7) 그래프 이론에서 사용하는 용어 정리
① 방향 그래프에서 진입 차수 : 주어진 정점으로 향한 간선의 개수
② 진출 차수 : 주어진 정점에서 시작하는 간선의 개수
③ 정점의 차수 : 진출 차수와 진입 차수의 합
④ 무방향 그래프에서 차수 : 정점에 연결된 간선들의 개수
⑤ 루프 : 간선의 시작점과 끝점이 같은 정점인 길이가 1인 경로
⑥ 무사이클 그래프 : 사이클이 없는 그래프를 '무사이클 그래프' 혹은 '트리' 라고함 (방향이 있는 무사이클 그래프를 dag, directed acyclic graph라고 부름)

2. 추상 자료형
1) 내용
① 그래프는 트리보다 더 광범위하게 응용되는 자료구조임
② 그래프 객체의 정의: 정점과 간선의 유한 리스트

③ 연산
- Graph Creat( ) ::= 그래프 생성
- Destory(Graph)::= 그래프 기억장소 반납
- Graph copy_Tree(Graph):: 그래프 복사
- InsertVertex( )::= 그래프에 정점 삽입
  InsertEdge( )::= 그래프에 간선 추가
- DeleteVertex( ) ::= 그래프에 정점 삽입
  DeleteEdge( ) ::= 간선삭제
- Search ( ) ::= 정점 탐색
- IsAdjacent( ) ::= 인접 정점 여부 확인
- ExistPath( ) ::= 경로가 존재하는지 확인
- PathLength( ) ::= 경로 길이 계산
- BFS( ) ::= 너비 우선 탐색
- DFS( ) ::= 깊이 우선 탐색

3. 그래프 표현법

1) 인접 행렬에 의한 그래프 표현

① 인접 행렬은 행의 수와 열의 수가 같은 정방 행렬이고 모든 원소는 0이거나 1의 값을 가짐

② 인접행렬로 그래프를 표현한다는 것은 임의의 두 정점 v*i*, v*j*를 연결하는 간선이 존재하는지를 나타내는 것임

③ 가중 그래프도 인접행렬로 나타낼 수 있음

2) 인접 리스트에 의한 그래프 표현

① 인접 리스트는 정점의 개수가 n인 그래프에 대하여 인접 행렬의 n행을 n개의 연결리스트로 나타내는 방법임

② 리스트 *i*의 각 노드들은 정점 *i*에 인접되어 있는 장점들을 나타냄

③ 어떤 임의의 정점에 대한 인접 리스트에 임의로 접근할 수 있음

<14장 출제예상문제>

1. 평행한 간선을 포함하는 그래프를 무엇이라 하는가?

① 다중 그래프

② 널 그래프

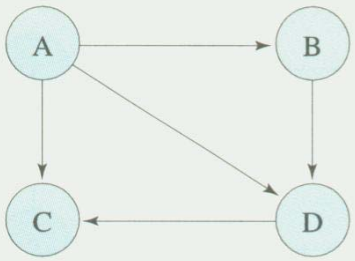
③ 가중 그래프

③ 단순 그래프

[정답] 1

[해설] 그래프는 같은 간선을 중복해서 가질 수 없으며, 그래프에 이러한 제한이 없을 때 만들어지는 데이터 객체를 다중 그래프(multigraph)라 한다.

2. 다음 그래프의 인접 행렬을 완성하여라.

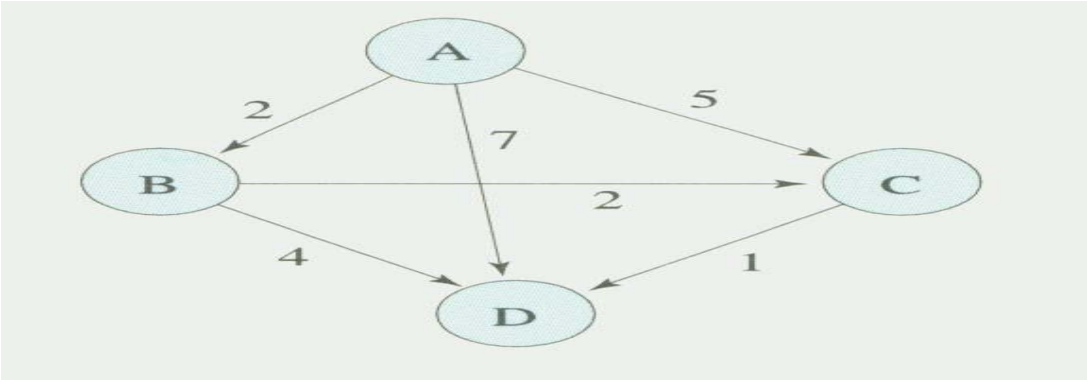


	A	B	C	D
A	0	1	1	1
B	0	0	0	0
C	0	0	0	0
D	(			)

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

[정답] 2
[해설] 인접행렬로 0 0 1 0 이다.

[3~4] 다음 그래프와 관련하여 물음에 답하여라.



3. 위의 그래프에서 정점 c의 진입 차수는 얼마인가?
① 1
② 2
③ 5
④ 7

[정답] 2
[해설] 정점 C의 진입 차수는 2이다.

4. 위에 그래프에서 TE(earliest completion time)와 TL(latest completion time)의 조건이 TE=TL이 되는 경로는?

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

[정답] 4
[해설] TE의 조건에 따라 A-D의 경로를 따른다.

5. 그래프의 간선이 모든 정점으로 구성된 트리를 무엇이라 하는가?

① 완전 트리
② 탐색 트리
③ 신장 트리
④ 선택 트리

[정답] 3
[해설] 신장 트리 그래프 G가 연결되어 있을 때, 즉 그래프 G의 서로 다른 정점 V_i , Vj의 모든 쌍에 대해 V_i에서 Vj로의 경로가 존재하면, DFS나 BFS는 어떤 점에서 시작하든 G에 있는 모든 정점을 방문함. 이 경우 G의 간선들은 2개의 집합 T와 B로 나눌 수 있고, 여기서 T는 탐색 중에 순회한 간선들의 집합, B는 순회하지 않은 나머지 간선들의 집합이다.

6. 다음 빈 칸에 들어갈 말로 알맞은 것은?

한 경로 상에 있는 모든 정점들이 서로 다를 경우 그 경로를 ( )라 하고, 또한 처음 정점과 마지막 정점이 같은 경로를 ( )이라 한다.

① 다중 경로, 사이클
② 단순 경로, 사이클
③ 단순 경로, 연결 그래프
④ 다중 경로, 연결 그래프

[정답] 2
[해설] 한 경로 상에 있는 모든 정점들이 서로 다를 때, 그 경로를 단순 경로(simple path)라 함. 단, 단순 경로에서 처음과 마지막 정점은 같을 수 있음. 경로는 단순히 정점들을 나열하여 표기할 수 있으며 처음과 마지막 정점이 같은 단순 경로를 사이클(cycle)이라 한다. 방향 그래프의 경우에는 사이클과 경로라는 용어 앞에 '방향(directed)'이라는 접두사를 사용한다.

7. 다음 빈 칸에 알맞은 말은?

( )는 가중 방향그래프에서 두 정점 사이의 가장 긴 경로로서, 프로젝트 수행 시 많은 상황을 적용할 수 있는 경영도구로 사용된다.

① 최대경로
② 주경로
③ 최소경로
④ 가중경로

[정답] 2
[해설] 주경로에 대한 설명이다.

8. 방향그래프를 인접행렬로 나타낼 때, 행의 합은 ( )이며, 열의 합은 ( )이다. ( )에 적합한 내용은?
① 진출차수, 진입차수
② 진출경로수, 진입경로수
③ 진입차수, 진출차수
④ 진입경로수, 진출경로수

[정답] 1
[해설] 방향 그래프의 경우 인접행렬을 나타낼 때 행의 합은 진출 차수이며 열의 합은 진입 차수이며 방향 그래프의 경우, 임의의 정점 v가 머리가 되는 간선들의 수를 정점 v의 진입 차수(in-degree)라 하고, v가 꼬리가 되는 간선들의 수를 정점 v의 진출 차수(out-degree)라 한다.

9. 다음 중 정점의 수가 n인 무방향 그래프에서는 최대 간선 수는?
① n(n+1)
② n(n-1)/2
③ n(n-1)
④ n(n+1)/2₩

[정답] 2
[해설] (V_i, V_j)의 수로 n(n-1)/2. 정점의 수가 n인 방향 그래프에서는 최대 간선 수가 n(n-1)이 된다.

10. A(i,j)는 정점 V_i 와 V_j간의 최단 경로 비용으로 알고리즘의 빈 칸에 들어갈 내용은?

```
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            A(i , j) ← min {A(i , j), _____㉠_____ }
        end
    end
end
```

① A(i,k)+A(k,j)
② A(i,k)−A(k,j)
③ A(j,k)+A(k,i)
④ A(j,k)−A(k,i)

[정답] 1

[해설] $A^n(i,j)$를 계산하는 알고리즘의 ⑨A(i , j) ← min {A(i , j), A(i , k) + A(k, j) } 이다.

제15장 그래프 II

1. 그래프 탐색

1) 깊이 우선 탐색

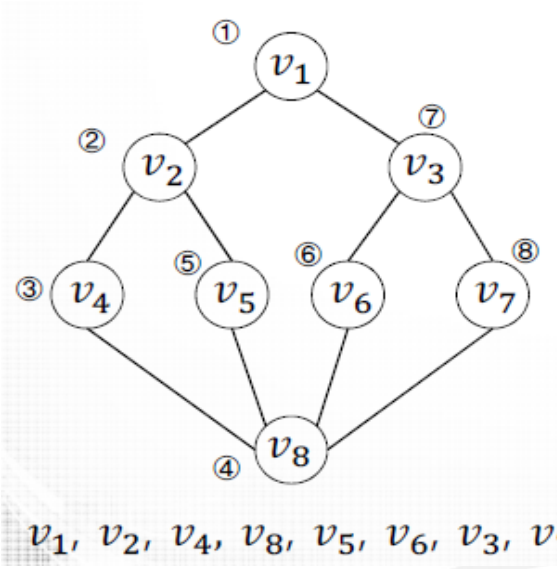
① 주어진 출발점 v를 방문하는 것으로 시작함

② v에 인접하고 아직 방문하지 않은 정점 w를 선택하여 w를 출발점으로 다시 깊이 우선 탐색을 시작함

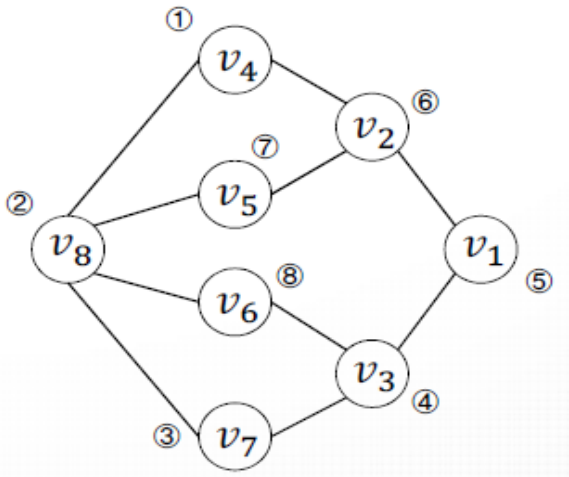
③ 깊이 우선 탐색을 수행하는 알고리즘은 순환 호출을 이용하면 쉽게 구현할 수 있고, 아니면 스택을 직접 운영하여 구현할 수 도 있음

④ 무방향 그래프

<출발점이 v_1, 혹은 v_4일 경우>



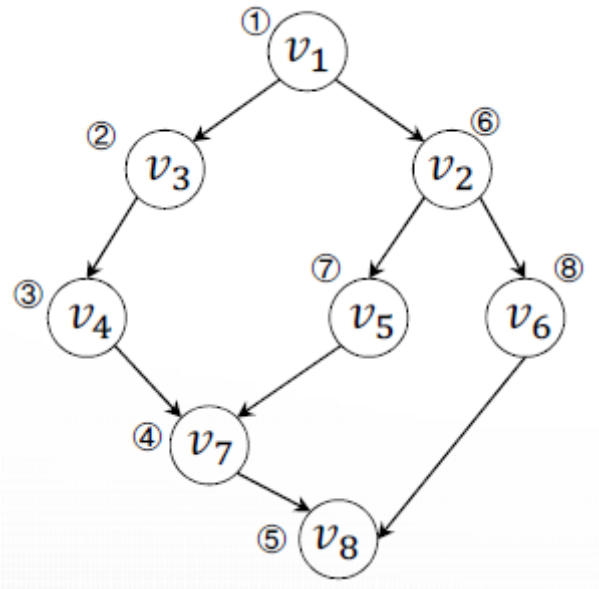
$v_1, v_2, v_4, v_8, v_5, v_6, v_3, v_7$



$v_4, v_8, v_7, v_3, v_1, v_2, v_5, v_6$

⑤ 방향 그래프

<출발점 v_1>



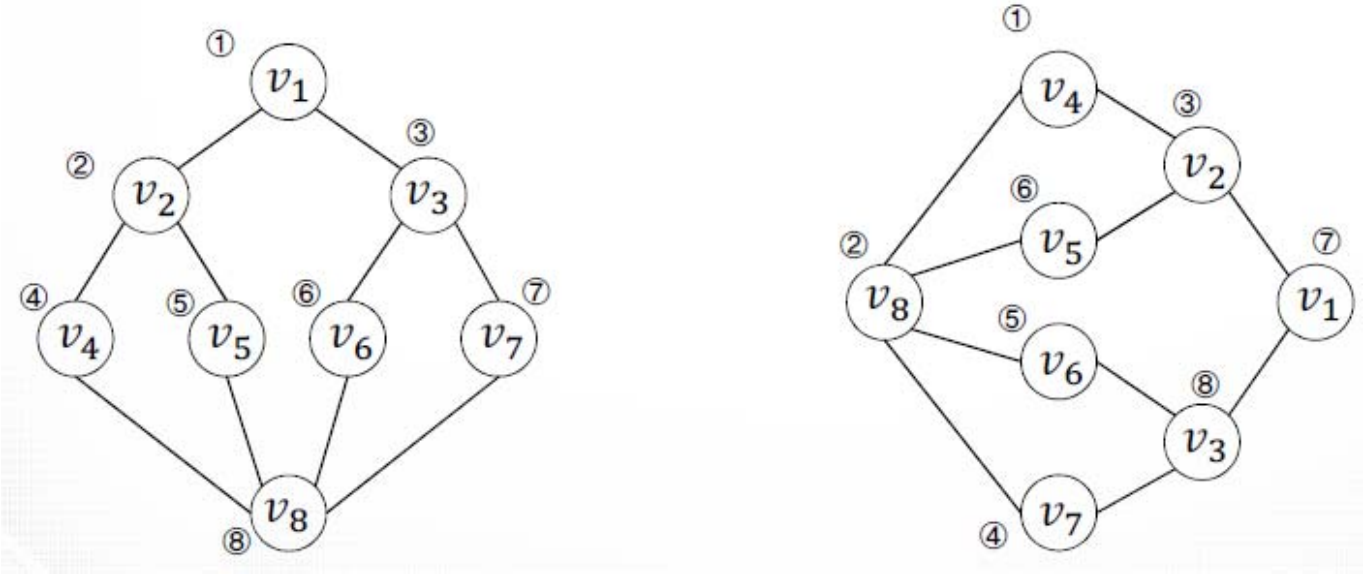
2) 너비 우선 탐색

① 깊이 우선 탐색과는 다르게 v에 인접한 정점 w를 모두 방문 후 , 다시 w에 인접한 방문하지 않은 정점들을 차례로 방문함

② 현재 방문 정점의 모든 인접 정점, 즉 모든 형태를 먼저 방문하므로 아래로 내려가지 않고 먼저 옆을 탐색함

③ 인접정점을 모두 방문하기 때문에 스택이 필요하지 않으며 대신 큐를 사용함

<출발점이 v_1, 혹은 v_4일 경우>



2. 최소 신장 트리

1) 최소신장 트리의 정의

① 트리 : 사이클이 없는 단순 그래프 -> 트리는 그래프이기는 하지만 루트를 가지기 때문에 (일반 그래프에는없는) 계층 개념이 있고, 사이클이 없어서 한 정점에서 다른 정점으로 가는 경로가 유일한 독특한 구조

② 신장 트리(spanning tree) : 그래프 G 의 모든 정점과 간선의 일부(또는 전부)를 포함하는 트리

- 주어진 그래프의 정점을 모두 포함함
- 최소 n-1 개의 간선으로 구성한 그래프

③ G 의 최소 부분 그래프 : 그래프 G 의 부분 그래프 중에서 간선의 수가 가장 작은 것

2) 최소 비용 신장 트리

(1) 프림 알고리즘

① 최소 비용을 갖는 간선을 차례로 추가하는 방법으로 트리를 구축함

② 사이틀이 형성되면 해당 간선은 포기함

③ 비용이 적은 것을 합치면 그들의 합이 최소가 될 것이라는(항상 옳지 않은)가정을 근거로 함

(2) 크루스컬 알고리즘

① 프림 알고리즘처럼 현재 완성한 트리에 간선을 붙여 트리를 키워 나가는 것이 아님

② 알고리즘은 매 단계 최소 비용 간선을 선택해 사이클만 형성하지 않으면 받아들이는 것임
③ 중간과정은 숲일 수 있음

(3) 솔린 알고리즘
① 매 단계에 다수의 간선을 선택함
② 간선이 하나도 없고 그래프의 모든 정점으로 구성된 숲에서 시작함
③ 단계가 거듭되면서 숲 내의 트리를 최소 비용을 갖는 간선으로 연결함
④ 남은 간선이 없거나 완전한 트리가 생성될 때까지 반복함으로써 신장 트리를 얻음

<15장 출제예상문제>

1. 탐색 기법 중 나눗셈은 사용하지 않고 덧셈과 뺄셈만 사용함으로써 평균 효율이 좋은 특성을 갖는 방법은?
① 이진 탐색
② 순차 탐색
③ 피보나치 탐색
④ 비순차 탐색

[정답] 3
[해설] 피보나치 탐색의 장점은 나눗셈을 사용하지 않고 단지 덧셈과 뺄셈만 사용하므로 피보나치 탐색의 평균 효율은 이진 탐색보다 더 좋은데, 이는 컴퓨터에서 나눗셈의 실행이 덧셈이나 뺄셈보다 훨씬 더 많은 시간을 소모하기 때문이다.

2. 해싱 함수 값을 구한 결과, 서로 다른 레코드가 동일한 홈 주소를 가지는 레코드의 집합을 무엇이라고 하는가?
① clustering
② chaining
③ synonym
④ collision

[정답] 3
[해설] 동일한 주소로 해싱되는 두 개의 데이터를 동거자(synonym)라고 하며 버킷에 빈 슬롯이 남아있는 경우 동거자를 계속 저장할 수 있어서 문제가 없지만, 데이터가 가득찬 버킷에 해싱 시키는 경우는 오버플로(overflow)가 발생된다. 버킷의 크기가 1인 경우는 충돌과 오버플로가 동시에 발생하며, 오버플로가 발생할 가능성을 줄이기 위해 버킷의 크기, 즉 슬롯의 개수를 증가하게 되면 해당 버킷에 저장되어 있는 자료를 탐색하는 시간이 길어지므로 적절한 버킷의 크기를 결정할 필요가 있다. 자료를 삽입할 때 두 개의 서로 다른 데이터가 해싱 함수에 의해 똑같은 주소의 버킷으로 해싱되는 경우가 발생하는데 이 경우를 충돌(collision)이라 한다.

3. 해시 테이블에서 슬롯(slot)의 의미는?
① 레코드 내의 자료 보관 장소
② 충돌 발생 시 오버플로를 위한 기억 공간
③ 동일한 주소를 갖는 해시 테이블의 한 구역
④ 해시 함수의 의한 홈 주소

[정답] 3
[해설] 해시 테이블에서 슬롯은 동일한 주소를 갖는 해시 테이블의 한 구역이다.

4. 킷값을 주소와 같은 자릿수를 갖는 몇 개의 부분으로 나눈 다음 그 부분을 서로 더하여 주소를 만들어 내는 해싱 함수는?
① 폴딩 해싱
② 제산 잔여 해싱
③ 중간 제곱 해싱
④ 숫자 분석 해싱

[정답] 1
[해설] 폴딩 해싱에 대한 설명이다.

5. 다음 중 순차탐색의 성공적인 탐색을 위한 평균 비교 횟수로 알맞은 것은?
① (n+1)/2
② (n-1)*2
③ (n-1)/2
④ (n+1)*2

[정답] 1
[해설] 성공적인 탐색의 경우에 수행되는 키 비교 횟수는 화일에서 찾고자 하는 레코드의 위치에 따라 다름. 모든 키가 서로 다르다고 가정하고 키 K_1을 찾으려면 $n-i+1$번의 키 비교가 필요함. 성공적인 탐색을 위한 균 비교 횟수는 $\sum_{i=1}^{n}(n-i+1)/n=(n+1)/2$이 된다.

6. 다음이 설명하는 용어에서 올바르지 않은 것은?
① 그래프 탐색: 그래프 내 특정 정점을 찾는 연산으로 트리와 다르게 루트 노드가 없으므로 특정 정점에서 시작한다.
② 그래프 순회: 특정 정점에서 시작하여 모든 형제를 방문한 후 자손을 방문한다.
③ 깊이 우선 탐색: 그래프 순회 알고리즘의 하나로 특정 점정에서 시작하여 자손을 먼저 방문한 후 전 단계 형제를 방문한다.

④ 최소비용 신장트리: 그래프의 모든 정점을 포함하는 사이클이 없는 부분 그래프이다.

[정답] 2
[해설] 그래프는 특정 정점에서 시작하여 그래프의 모든 정점을 빠짐없이 그리고 중복 없이 방문하는 것이다.

7. 다음이 설명하는 것은 무엇인가?

그래프의 모든 정점을 포함하는 사이클이 없는 부분, 그래프, 즉 트리를 신장 트리라 한다. 가중 그래프에서 비용이 최소인 신장 트리이다.

① 너비 우선 탐색
② 그래프 탐색
③ 최소 비용 신장 트리
④ 깊이 우선 탐색

[정답] 3
[해설] 최소 비용 신장 트리에 대한 설명이다.

8. 그래프 순회 알고리즘의 하나로 특정 점정에서 시작하여 자손을 먼저 방문한 후 전 단계 형제를 방문하는 것은 무엇인가?
① 그래프 탐색
② 최소비용신장트리
③ 그래프 순회
④ 너비 우선 탐색

[정답] 4
[해설] 너비 우선 탐색에 대한 설명이다.

9. 다음이 특징을 가지는 것은 무엇인가?

그래프이기는 하지만 루트를 가지기 때문에 계층 개념이 있고 사이클이 없어서 한 정점에서 다른 정점으로 가는 경로가 유일한 독특한 구조이다.

① 순회
② 트리
③ 노드
④ 단순 그래프

[정답] 2
[해설] 트리에 대한 설명이다.