DATA
STRUCTURES
USING C

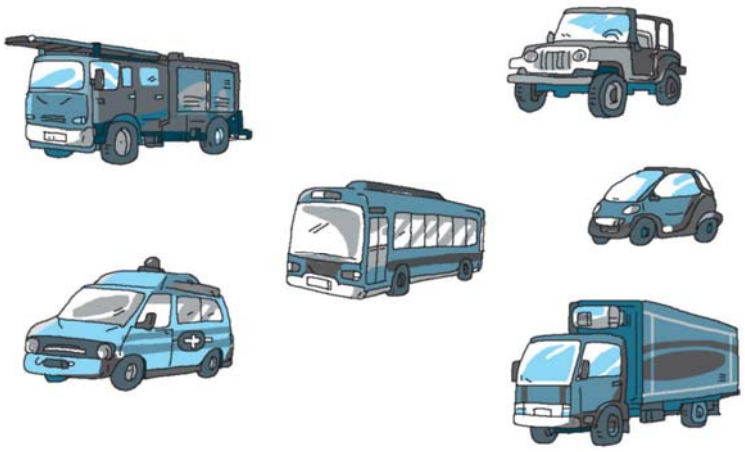
- 우선순위 큐의 개념을 이해한다.
- 다양한 우선순위 큐 구현 방법의 장단점을 이해한다.
- 힙의 동작 원리를 이해한다.
- 힙의 효율성을 이해한다.
- 힙의 배열을 이용한 구현을 이해한다.
- 힙을 이용한 허프만 코드 구현을 이해한다.

CHAPTER 10

우선순위 큐

실생활에서의 우선순위

- 도로에서의 자동차 우선순위



우선순위 높음 우선순위 낮음

우선순위 큐

- Priority queue
  - 우선순위를 가진 항목들을 저장하는 큐
  - 우선 순위가 높은 데이터가 먼저 나가게 됨
  - 가장 일반적인 큐로 생각할 수 있음
    - 스택이나 큐를 우선순위 큐로 구현할 수 있다.

자료구조	삭제되는 요소
스택	가장 최근에 들어온 데이터
큐	가장 먼저 들어온 데이터
우선순위큐	가장 우선순위가 높은 데이터

- 응용분야
  - 시뮬레이션, 네트워크 트래픽 제어, OS의 작업 스케쥴링 등

우선순위 큐의 추상 자료형

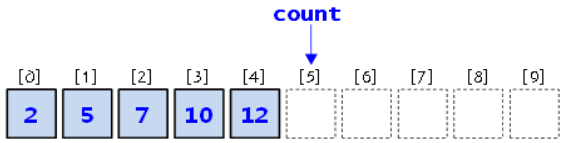
- ***데이터***
  - 우선순위를 가진 요소들의 모음

- ***연산***
  - init(): 우선순위 큐를 초기화 한다.
  - **insert(item)**: 우선순위 큐에 항목 item을 추가한다.
  - **delete()**: 가장 우선순위가 높은 요소를 꺼내서 반환한다.
  - find(): 가장 우선순위가 높은 요소를 삭제하지 않고 반환한다.
  - is_empty(): 우선순위 큐가 공백상태인지를 검사한다.
  - is_full(): 우선순위 큐가 포화상태인지를 검사한다.

- 다음에서 우선순위 큐 추상 자료형의 연산에 속하지 않는 것은?
  1. 특정한 값 탐색하기
  2. 최대값이나 최소값 삭제하기
  3. 주어진 값을 삽입하기
  4. 공백 상태 검사하기

10.2 우선순위 큐 구현방법

- 배열을 이용한 구현



- 연결리스트를 이용한 구현

- 힙(heap)을 이용한 구현
  - 완전이진트리
  - 우선순위 큐를 위해 만들어진 자료구조
  - 일종의 반 정렬 상태를 유지

우선순위 큐 구현방법 비교

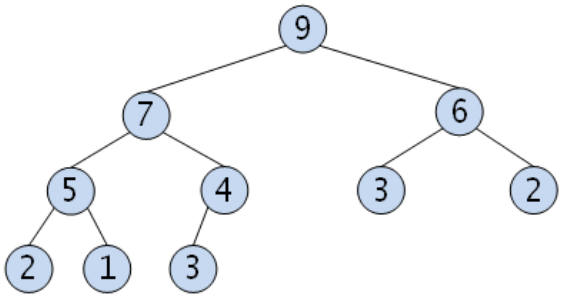
표현 방법	삽 입	삭 제
순서없는 배열	O(1)	O(n)
순서없는 연결 리스트	O(1)	O(n)
정렬된 배열	O(n)	O(1)
정렬된 연결 리스트	O(n)	O(1)
힙	**O(logn)**	**O(logn)**

10.3 힙(heap)이란?

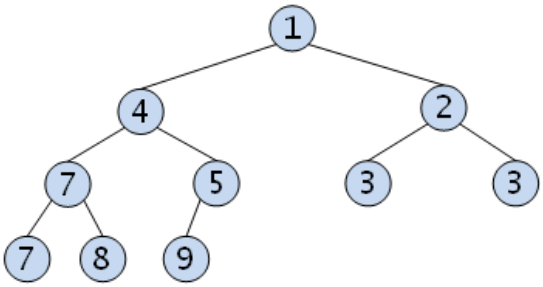
- Heap
  - 더미
  - 완전이진트리
  - 최대 힙, 최소 힙
- 최대 힙(max heap)
  - 부모 노드의 키값이 자식 노드의 키값보다 크거나 같은 완전 이진 트리
- 최소 힙(min heap)
  - 부모 노드의 키값이 자식 노드의 키값보다 작거나 같은 완전 이진 트리

최대힙과 최소힙

- 최대 힙(max heap)
  - key(부모노드) ≥key(자식노드)



- 최소 힙(min heap)
  - key(부모노드) ≤key(자식노드)



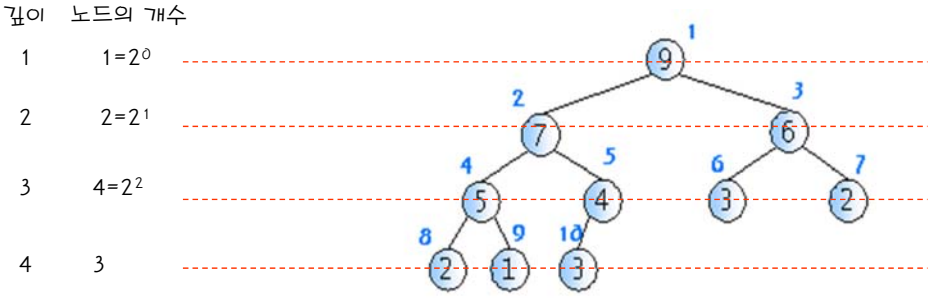
- 히프에서의 마지막 노드는 다음 중 어떤 조건을 만족하는가?
  1. 항상 왼쪽 노드이다
  2. 항상 오른쪽 노드이다
  3. 항상 맨 마지막 레벨에 있다
  4. 형제 노드보다 작지 않다

- 최소 히프에 관한 설명 중 틀린 것은?

1. 왼쪽 노드 값이 오른쪽 노드 값보다 항상 작다
2. 완전 이진 트리이다
3. 항상 삭제될 때는 루트 노드가 삭제된다
4. 부모 노드 값이 자식 노드 값보다 항상 작거나 같다

힙의 높이

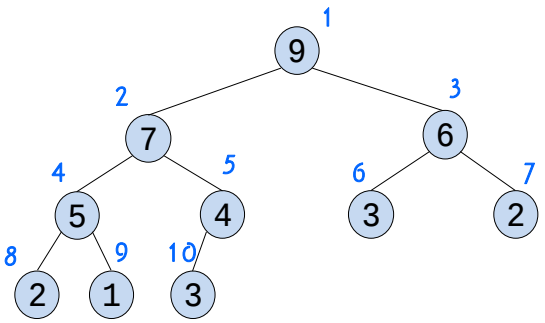
- n개의 노드를 가지고 있는 힙의 높이는 O(logn)
  - 히프는 완전이진트리
  - 마지막 레벨을 제외하고 각 레벨 i에 2^{i-1}개의 노드 존재



- 노드가 18개인 히프의 높이를 구하라

힙의 구현: 배열을 이용

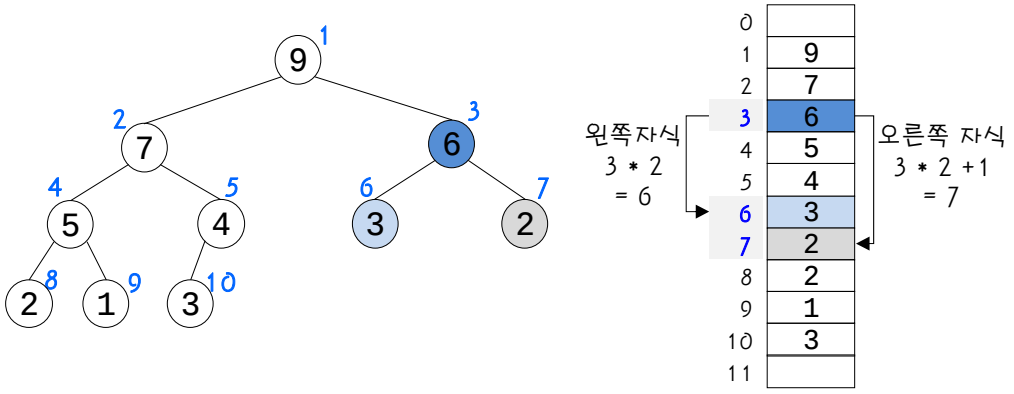
- 힙은 보통 배열을 이용하여 구현
  - 완전이진트리 ➔ 각 노드에 번호를 붙임 ➔ 배열의 인덱스



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

힙의 구현

- 부모노드와 자식노드의 관계
  - 왼쪽 자식의 인덱스 = (부모의 인덱스)*2
  - 오른쪽 자식의 인덱스 = (부모의 인덱스)*2 + 1
  - 부모의 인덱스 = (자식의 인덱스)/2



- 히프가 배열로 표현될 수 있는 이유는 무엇인가?
  1. 완전 이진 트리이기 때문에
  2. 어느 정도 정렬되어 있기 때문에
  3. 이진 트리이기 때문에
  4. 히프 조건을 만족하기 때문에

힙의 구현

- 예: 정수 저장 힙

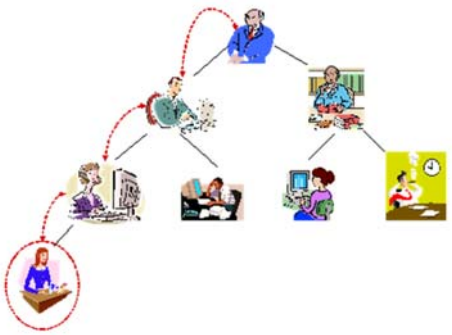
```
typedef int HNode;                  // 힙에 저장할 항목의 자료형
#define Key(n) (n)                  // 힙 노드 n의 키값

HNode heap[MAX_HEAP_NODE];          // 배열을 이용해 구현한 힙(힙노드 배열)
int heap_size;                      // 힙의 크기

#define Parent(i)       (heap[i/2])     // i의 부모 노드
#define Left(i)         (heap[i*2])     // i의 왼쪽 자식 노드
#define Right(i)        (heap[i*2+1])   // i의 오른쪽 자식 노드

void init_heap()        { heap_size = 0; }
int is_empty_heap()     { return heap_size == 0; }
int is_full_heap()      { return (heap_size==MAX_HEAP_NODE-1); }
HNode find_heap()       { return heap[1]; }
```


10.4 삽입 연산

- Upheap
  - 회사에서 신입 사원이 들어오면 일단 말단 위치에 앉힘
  - 신입 사원의 능력을 봐서 위로 승진시킴

(1)히프에 새로운 요소가 들어 오면, 일단 새로운 노드를 히프의 마지막 노드에 이어서 삽입

(2)삽입 후에 새로운 노드를 부모 노드들과 교환해서 히프의 성질을 만족

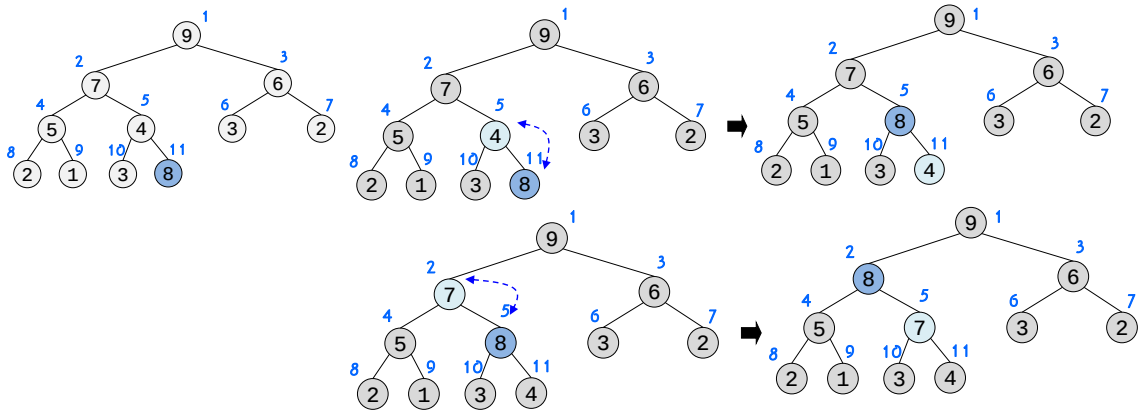


Upheap

- 삽입된 노드에서 루트까지의 경로에 있는 노드들을 비교/교환
- 히프의 성질을 복원
  - 삽입된 노드 키가 부모노드보다 작거나 같으면 upheap는 종료한다

```
insert(node)

heapSize ← heapSize + 1;
i ← heapSize;
heap[i] ← node;
while i ≠ 1 and KEY(heap[i]) > KEY(Parent(i)) do
      heap[i] ⇔ Parent(i);
      i ← Parent(i);
```


Upheap 과정 예



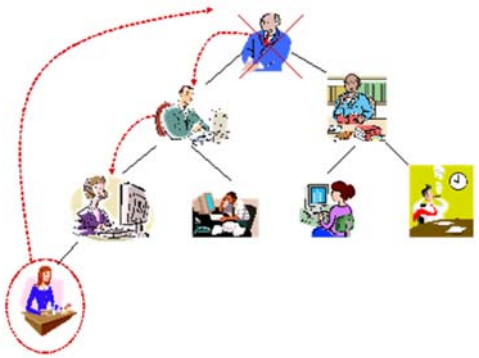
- 힙의 높이: O(logn) ➔ upheap연산은 O(logn)

삽입 함수

```
void insert_heap( HNode n )
{
        int i;
        if( is_full_heap() ) return;
        i = ++(heap_size);
        while( i!=1 && Key(n) > Key(Parent(i))) {
                heap[i] = Parent(i);
                i /= 2;

        }
        heap[i] = n;
}
```

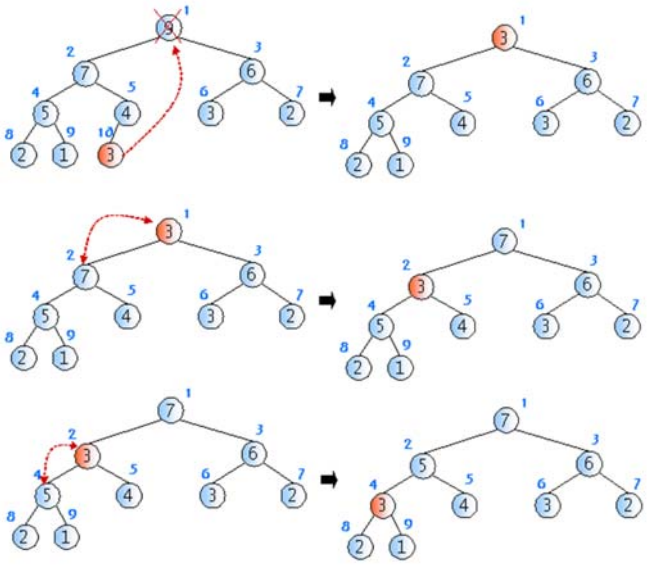

삭제 연산

- 최대힙에서의 삭제 ➔ 항상 루트가 삭제됨
  - 가장 큰 키값을 가진 노드를 삭제하는 것
- 방법: downheap
  - 루트 삭제
  - 회사에서 사장의 자리가 비게 됨
  - 말단 사원을 사장 자리로 올림
  - 능력에 따라 강등 반복



루트에서부터 단말노드까지의 경로에 있는 노드들을 교환하여 히프 성질을 만족시킨다.

Downheap 과정



- 힙의 높이: O(logn) ➔ downheap연산은 O(logn)

Downheap 알고리즘

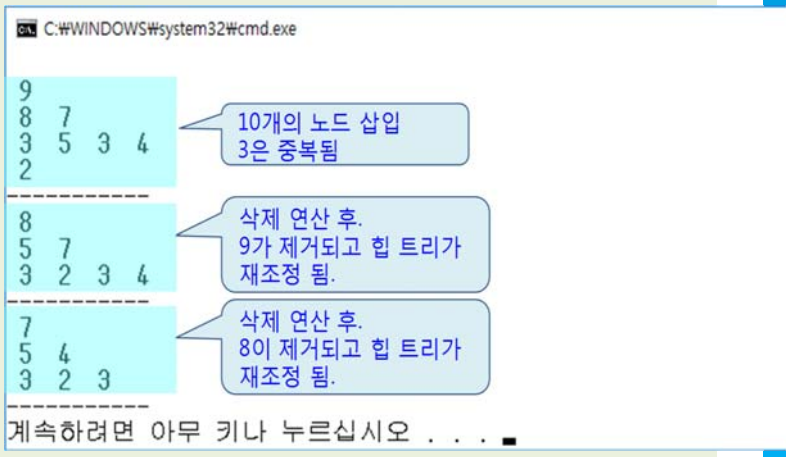
```
remove()

root ← A[1];
A[1] ← A[heapSize];
heapSize ← heapSize-1;
i ← 2;
while i ≤ heapSize do
      if i < heapSize and A[LEFT(i)] > A[RIGHT(i)]
         then largest ← LEFT(i);
         else largest ← RIGHT(i);
      if A[PARENT(largest)] > A[largest]
         then break;
      A[PARENT(largest)] ↔ A[largest];
      i ← LEFT(largest);
return root;
```

삭제 함수

```
HNode delete_heap()
{
        HNode hroot, last;
        int parent = 1, child = 2;
        if( is_empty_heap()) error("힙 트리 공백 에러");
        hroot = heap[1];
        last = heap[heap_size--];
        while( child <= heap_size ){
                if( child < heap_size
                && Key(Left(parent)) < Key(Right(parent)))
                        child++;
                if( Key(last) >= Key(heap[child]) ) break;
                heap[parent] = heap[child];
                parent = child;
                child *= 2;
        }
        heap[parent] = last;
        return hroot;
}
```


전체 프로그램

```
void main()
{
        init_heap();
        insert_heap(2 );
        insert_heap(5 );
        insert_heap(4 );
        insert_heap(8 );
        insert_heap(9 );
        insert_heap(3 );
        insert_heap(7 );
        insert_heap(3 );
        print_heap();

        delete_heap( );
        print_heap( );
        delete_heap( );
        print_heap( );
        printf("\n");
}
```

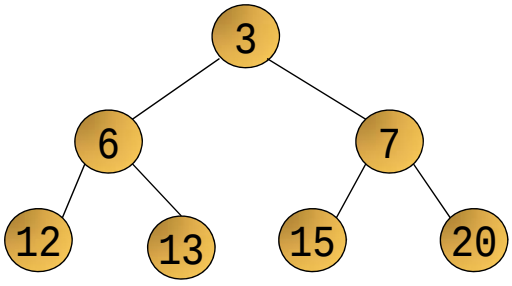


- 히프를 구현한 배열의 내용이 다음과 같을 때 물음에 답하라.

인덱스	0	1	2	3	4	5	6
데이터	0	2	5	6	8	9	10

1. 위의 히프에 해당하는 히프 트리를 그려라.
2. 이 히프에서 한 번의 삭제 연산을 수행한 다음의 배열 내용을 쓰라.
3. 데이터 7을 삽입한 다음의 배열의 내용을 쓰라.

- 다음의 최소 히프트리에 답하라.



1. 2을 삽입하였을 경우, 히프 트리를 재구성하는 과정을 보여라.
2. 삭제 연산이 한 번 이루어진 다음에 히프를 재구성하는 과정을 보여라.

- 히프 트리가 비어 있는 상태에서 다음의 연산들을 차례대로 수행한 후의 최소 히프 트리의 모습을 그려라.

  insert(20), insert(12), insert(3), insert(2), delete(), insert(5), insert(16), delete(), insert(1), is_empty()

힙의 복잡도 분석

- 삽입 연산에서 최악의 경우
  - 루트 노드까지 올라가야 하므로 트리의 높이에 해당하는 비교 연산 및 이동 연산이 필요하다.

  ➔ O(logn)

- 삭제연산 최악의 경우
  - 가장 아래 레벨까지 내려가야 하므로 역시 트리의 높이 만큼의 시간이 걸린다.

  ➔ O(logn)

10.5 힙 정렬

- 힙을 이용하면 정렬 가능: 힙 정렬
  - 먼저 정렬해야 할 n개의 요소들을 최대 힙에 삽입
  - 한번에 하나씩 요소를 힙에서 삭제하여 저장하면 된다.
  - 삭제되는 요소들은 값이 증가되는 순서(최소힙의 경우)
- 시간 복잡도: O(nlogn)
  - 하나의 요소의 삽입 삭제가 O(logn)
  - 요소의 개수가 n개 ➔ O(nlogn)
- 특히 유용한 경우
  - 전체의 정렬이 아니라 가장 큰 값 몇 개만 필요할 때이다.

힙 정렬

```
void main()
{
        int     i, data[10];
        // 난수 배열 생성
        for( i=0 ; i<10 ; i++ )
                data[i] = rand() % 100;
        print_array(data, 10, "정렬 전");
        init_heap();
        for (i = 0; i<10; i++)
                insert_heap(data[i]);
        for (i = 9; i >= 0; i--)
                data[i] = Key(delete_heap());
        print_array(data, 10, "정렬 후");
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
정렬 전:  41 67 34  0 69 24 78 58 62 64
정렬 후:   0 24 34 41 58 62 64 67 69 78
계속하려면 아무 키나 누르십시오 . . .
```

10.6 허프만 코드

- 이진 트리는 각 글자의 빈도가 알려져 있는 메시지의 내용을 압축하는데 사용될 수 있음
- 이런 종류의 이진트리 ➔ 허프만 코딩 트리

A	80
B	16
C	32
D	36
E	123
F	22
G	16
H	51
I	71
...	
Z	1

문자의 빈도수

- 빈도수가 알려진 문자에 대한 고정길이코드와 가변길이코드의 비교

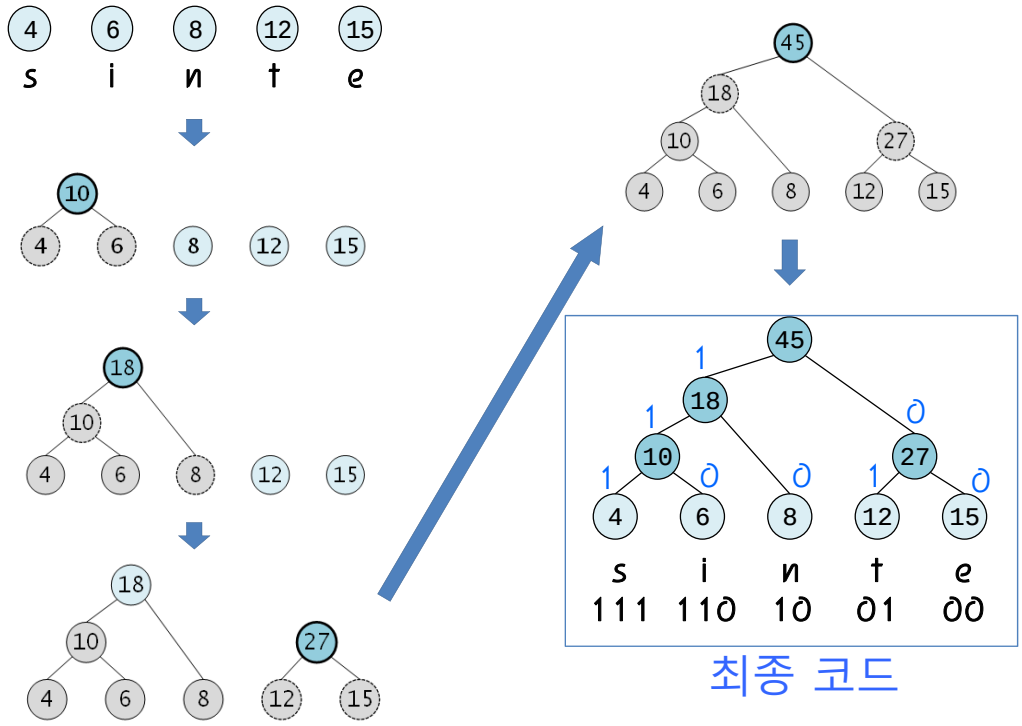
글자	빈도수	고정길이코드			가변길이코드		
		코드	비트수	전체 비트수	코드	비트수	전체 비트수
A	17	0000	4	68	00	2	34
B	3	0001	4	12	11110	5	15
C	6	0010	4	24	0110	4	24
D	9	0011	4	36	1110	4	36
E	27	0100	4	108	10	2	54
F	5	0101	4	20	0111	4	20
G	4	0110	4	16	11110	5	20
H	13	0111	4	52	010	3	39
I	15	1000	4	60	110	3	45
J	1	1001	4	4	11111	5	5
합계	100			400			292

- 코드 읽기

	F	A	C	E
고정길이코드:	0101	0000	0010	0100

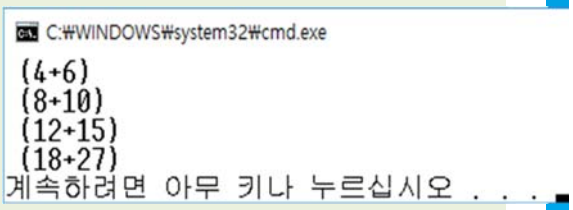
	F	A	C	E
가변길이코드:	0111	00	0110	10

허프만 코드 생성 절차



허프만 코딩 트리 생성 프로그램

```
void make_tree(int freq[], int n)
{
        HNode e1, e2;
        int i;
        init_heap();
        for (i = 0; i<n; i++)
                insert_heap(freq[i]);
        for (i = 1; i<n; i++){
                e1 = delete_heap();
                e2 = delete_heap();
                insert_heap(Key(e1) + Key(e2));
                printf(" (%d+%d)\n", Key(e1), Key(e2));
        }
}
int main()
{
        char label[] = { 'A', 'B', 'C', 'D', 'E' };
        int freq[] = { 15, 12, 8, 6, 4 };
        make_tree(freq, 5);
}
```



실습

- 허프만 코드 프로그램에서 생성된 허프만 트리에서 허프만 코드를 얻는 부분을 추가하라. 루트에서 출발하여 왼쪽 노드로 갈 때는 1을 출력하고 오른쪽 자식으로 갈 때는 0을 출력하라.