CSE2010 자료구조론

Week 8: Priority Queue, Heap 2

ICT융합학부 한진영

응용: 힙 정렬

- 힙을 이용하면 정렬 가능: 힙 정렬
  - 힙 정렬이 최대로 유용한 경우는 전체 자료를 정렬하는 것이 아니라 가장 큰 값 몇 개만 필요할 때
- 알고리즘
  - 먼저 정렬해야 할 n개의 요소들을 최대 힙에 삽입
  - 한번에 하나씩 요소를 힙에서 삭제하여 저장하면 됨
- 복잡도
  - 하나의 요소를 힙에 삽입하거나 삭제할 때 시간이 O(logn) 만큼 소요되고 요소의 개수가 n개이므로 전체적으로 O(nlogn)시간이 걸림
    - 빠른편

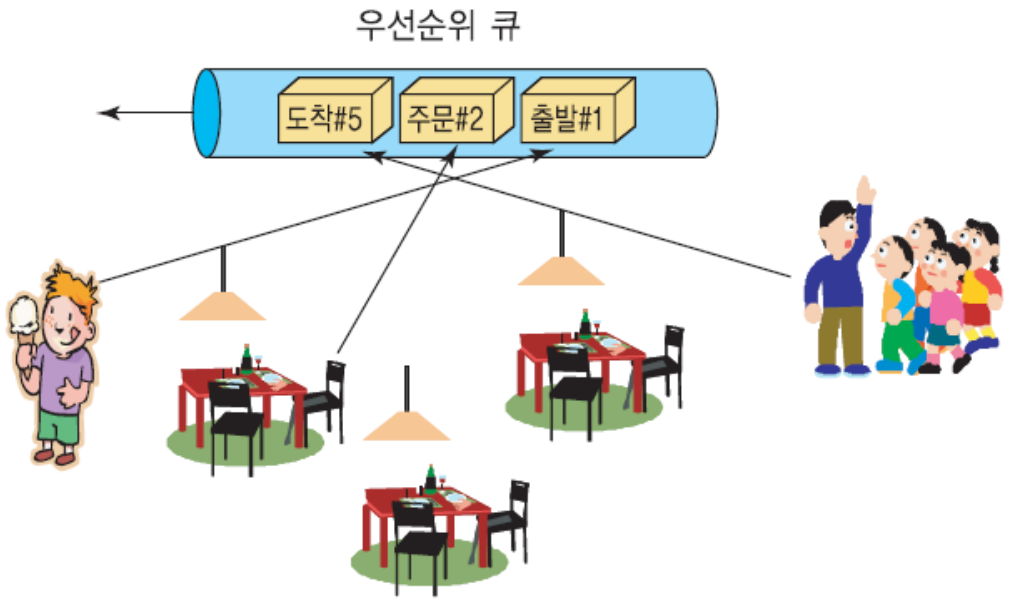
힙 정렬 알고리즘

```
// 우선 순위 큐인 힙을 이용한 정렬
void heap_sort(element a[], int n)
{
	int i;
	HeapType h;

	init(&h);
	for(i=0;i<n;i++){
		insert_max_heap(&h, a[i]);
	}
	for(i=(n-1);i>=0;i--){
		a[i] = delete_max_heap(&h);
	}
}
```

응용: 이산 이벤트 시뮬레이션

- 이산 이벤트 시뮬레이션
  - 모든 시간의 진행은 이벤트의 발생에 의해서 이루어짐
  - 우선순위 큐를 이용하여 이벤트를 저장하고 이벤트의 발생 시각을 우선순위로 하여 이벤트를 처리하는 간단한 시뮬레이션을 생각해 볼 수 있음



이산 이벤트 시뮬레이션 예

```
현재 시간 = 0
3명의 고객 도착
현재 시간 = 1
4명의 고객 도착
현재 시간 = 4
아이스크림 1개 주문 받음
아이스크림 3개 주문 받음
아이스크림 3개 주문 받음
현재 시간 = 4
아이스크림 1개 주문 받음
아이스크림 1개 주문 받음
아이스크림 1개 주문 받음
아이스크림 2개 주문 받음
현재 시간 = 4
2명의 고객 도착
현재 시간 = 5
아이스크림 3개 주문 받음
아이스크림 2개 주문 받음
현재 시간 = 6
4명의 고객 도착
자리가 없어서 떠남
현재 시간 = 6
4명이 매장을 떠남
현재 시간 = 7
2명이 매장을 떠남
현재 시간 = 10
3명이 매장을 떠남
전체 순이익은 = 5.950000입니다.
```

이산 이벤트 시뮬레이션 구현(1)

```
#define ARRIVAL 1
#define ORDER 2
#define LEAVE 3
int free_seats=10;
double profit=0.0;
#define MAX_ELEMENT 100
typedef struct {
    int type;  // 이벤트의 종류
    int key;   // 이벤트가 일어난 시각
    int number;    // 고객의 숫자
} element;
typedef struct {
    element heap[MAX_ELEMENT];
    int heap_size;
} HeapType;
```

이산 이벤트 시뮬레이션 구현(2)

```
// 삽입 함수
void insert_min_heap(HeapType *h, element item)
{
    int i;
    i = ++(h->heap_size);
    // 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정
    while((i != 1) && (item.key < h->heap[i/2].key)){
        h->heap[i] = h->heap[i/2];
    i /= 2;
    }
    h->heap[i] = item; // 새로운 노드를 삽입
}
```

이산 이벤트 시뮬레이션 구현(3)

```
// 삭제 함수
element delete_min_heap(HeapType *h)
{
        int parent, child;
        element 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].key) > h->heap[child+1].key)
            child++;
            if( temp.key <= h->heap[child].key ) break;
            h->heap[parent] = h->heap[child];
            parent = child;
            child *= 2;
        }
        h->heap[parent] = temp;
        return item;
}
```

이산 이벤트 시뮬레이션 구현(4)

```
// 0에서 n사이의 정수 난수 생성 함수
int random(int n)
{
    return rand() % n;
}
// 자리가 가능하면 빈 자리수를 사람수만큼 감소시킨다.
int is_seat_available(int number)
{
    printf("%d명의 고객 도착₩n", number);
    if( free_seats >= number ){
        free_seats -= number;
        return TRUE;
    }
    else {
        printf("자리가 없어서 떠남₩n");
        return FALSE;
    }
}
```

이산 이벤트 시뮬레이션 구현(5)

```
// 주문을 받으면 순익을 나타내는 변수를 증가시킨다.
void order(int scoops)
{
    printf("아이스크림 %d개 주문 받음₩n", scoops);
    profit += 0.35 * scoops;
}
// 고객이 떠나면 빈자리수를 증가시킨다.
void leave(int number)
{
    printf("%d명이 매장을 떠남₩n", number);
    free_seats += number;
}
```

이산 이벤트 시뮬레이션 구현(6)

```
// 이벤트를 처리한다.
void process_event(HeapType *heap, element e)
{
    int i=0;
    element new_event;

    printf("현재 시간=%d₩n", e.key);
    switch(e.type){
    case ARRIVAL:
        // 자리가 가능하면 주문 이벤트를 만든다.
        if( is_seat_available(e.number) ){
            new_event.type=ORDER;
            new_event.key = e.key + 1 + random(4);
            new_event.number=e.number;
            insert_min_heap(heap, new_event);
        }
        break;
```

이산 이벤트 시뮬레이션 구현(7)

```
case ORDER:
        // 사람수만큼 주문을 받는다.
        for (i = 0; i < e.number; i++){
            order(1 + random(3));
        }
        // 매장을 떠나는 이벤트를 생성한다.
        new_event.type=LEAVE;
        new_event.key = e.key + 1 + random(10);
        new_event.number=e.number;
        insert_min_heap(heap, new_event);
        break;
    case LEAVE:
        // 고객이 떠나면 빈자리수를 증가시킨다.
        leave(e.number);
        break;
    }
}
```

이산 이벤트 시뮬레이션 구현(8)

```
int main()
{
        element event;
        HeapType heap;
        unsigned int t = 0;
        init(&heap);
        // 처음에 몇개의 초기 이벤트를 생성시킨다.
        while (t < 5) {
        t += random(6);
                event.type = ARRIVAL;
                event.key = t;
                event.number = 1 + random(4);
                insert_min_heap(&heap, event);
        }
        while (!is_empty(&heap)) {
                event = delete_min_heap(&heap);
                process_event(&heap, event);
        }
        printf("전체 순이익은 =%f입니다.₩n ", profit);
}
```

이산 이벤트 시뮬레이션 예

```
현재 시간 = 0
3명의 고객 도착
현재 시간 = 1
4명의 고객 도착
현재 시간 = 4
아이스크림 1개 주문 받음
아이스크림 3개 주문 받음
아이스크림 3개 주문 받음
현재 시간 = 4
아이스크림 1개 주문 받음
아이스크림 1개 주문 받음
아이스크림 1개 주문 받음
아이스크림 2개 주문 받음
현재 시간 = 4
2명의 고객 도착
현재 시간 = 5
아이스크림 3개 주문 받음
아이스크림 2개 주문 받음
현재 시간 = 6
4명의 고객 도착
자리가 없어서 떠남
현재 시간 = 6
4명이 매장을 떠남
현재 시간 = 7
2명이 매장을 떠남
현재 시간 = 10
3명이 매장을 떠남
전체 순이익은 = 5.950000입니다.
```

응용: 허프만 코드

- 이진 트리는 각 글자의 빈도가 알려져 있는 메시지의 내용을 압축하는데 사용될 수 있음
  - 이런 종류의 이진트리를 허프만 코딩 트리라고 함

빈도수 분석 →

A	80
B	16
C	32
D	36
E	123
F	22
G	26
H	51
I	71
...	
Z	1

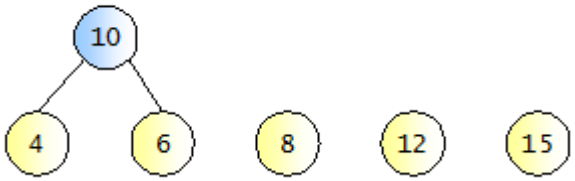
허프만 코드: 글자 빈도수

- 예: 만약 텍스트가 e, t, n, i, s의 5개의 글자로만 이루어졌다고 가정하고 각 글자의 빈도수가 다음과 같다고 가정

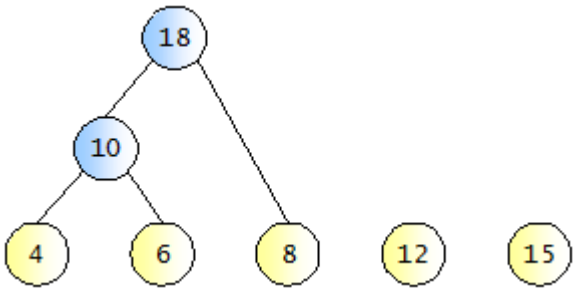
글자	비트 코드	빈도수
e	00	15
t	01	12
n	10	8
i	110	6
s	111	4
합계		

허프만 코드 생성 절차(1)

- 빈도수에 따라 5개의 글자 나열 (4,6,8,12,15)
- 가장 작은 빈도수를 가진 글자 2개(4,6) 추출하여 이진트리 구성

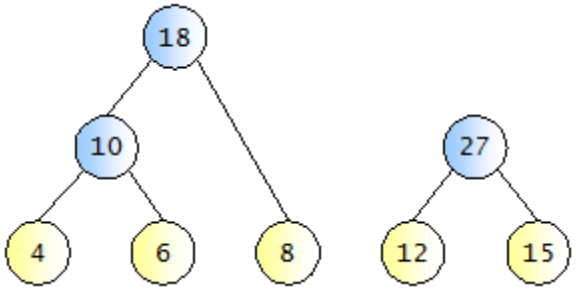


- 합쳐진 글자를 리스트에 삽입하여 (10,8,12,15) 얻음
- 이 빈도수를 정렬하여 (8,10,12,15) 얻음
- 이중 가장 작은 값 2개를 단말노드로 하여 이진트리 구성

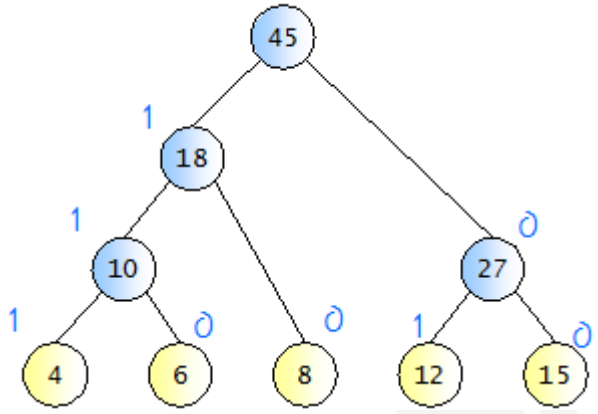


허프만 코드 생성 절차(2)

- 합쳐진 글자를 리스트에 삽입후 정렬하여 (12,15,18) 얻음
- 가장 작은 빈도수를 가진 글자 2개 (12,15) 추출하여 이진트리 구성



- (18, 27) 을 단말 노드로 하여 이진트리 구성



- 최종 트리
  - 왼쪽 edge는 1, 오른쪽 edge는 0
  - 빈도수 6에 해당하는 글자 i의 허프만 코드: 110

허프만 코드 구현(1)

```
// 이진 트리 생성 함수
TreeNode *make_tree(TreeNode *left, TreeNode *right)
{
    TreeNode *node= (TreeNode *)malloc(sizeof(TreeNode));
    if( node == NULL ){
        fprintf(stderr,"메모리 에러₩n");
        exit(1);
    }
    node->left_child = left;
    node->right_child = right;
    return node;
}
// 이진 트리 제거 함수
void destroy_tree(TreeNode *root)
{
    if( root == NULL ) return;
    destroy_tree(root->left_child);
    destroy_tree(root->right_child);
    free(root);
}
```

허프만 코드 구현(2)

```
// 허프만 코드 생성 함수
void huffman_tree(int freq[], int n)
{
    int i;
    TreeNode *node, *x;
    HeapType heap;
    element e, e1, e2;
    init(&heap);
    for(i=0;i<n;i++){
        node = make_tree(NULL, NULL);
        e.key = node->weight = freq[i];
        e.ptree = node;
        insert_min_heap(&heap, e);
     }
```

허프만 코드 구현(3)

```
for(i=1;i<n;i++){
        // 최소값을 가지는 두개의 노드를 삭제
        e1 = delete_min_heap(&heap);
        e2 = delete_min_heap(&heap);
        // 두개의 노드를 합친다.
        x = make_tree(e1.ptree, e2.ptree);
        e.key = x->weight = e1.key + e2.key;
        e.ptree = x;
        insert_min_heap(&heap, e);
    }
    e = delete_min_heap(&heap); // 최종 트리
    destroy_tree(e.ptree);
}
```

Week 8: Priority Queue, Heap 2