

Data Structure

실습 11

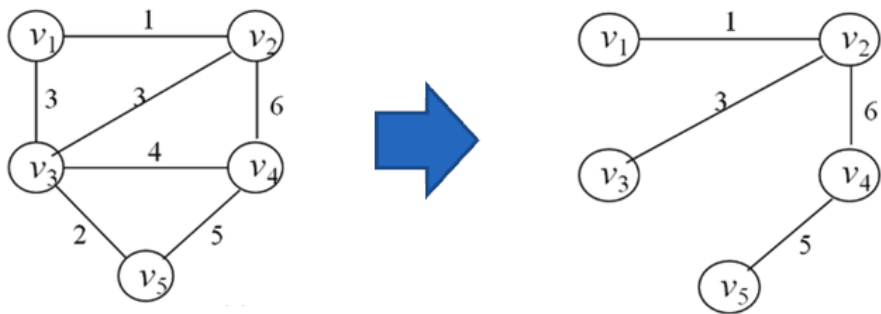
0. 이번 주 실습 내용

- **Graph** 복습
- **Hash**
 - Hash의 정의 & 용어 & 구조
 - Hash의 원리 (Collision)
 - Linear Probing & Chaining
- **Hash 실습**
 - Chaining Hash Table 구현 실습
- **과제 안내**

1. Graph 복습

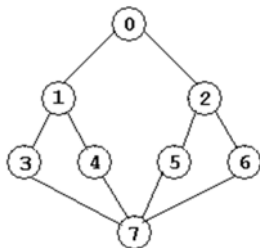
- **Spanning Tree**

- 그래프 내의 모든 Vertex들을 포함하는 트리
- Spanning Tree의 조건
 - 모든 Vertex들이 서로 연결되어 있어야 함
 - 사이클(Cycle)이 생겨서는 안됨

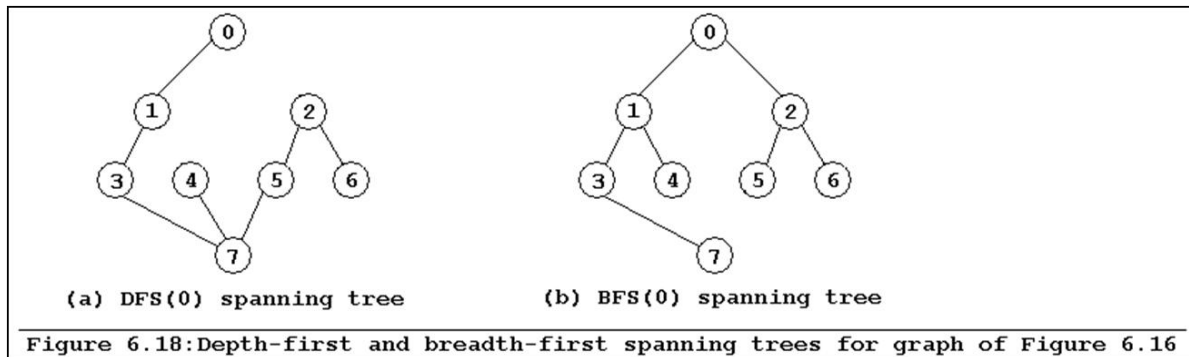


1. Graph 복습

- Spanning Tree



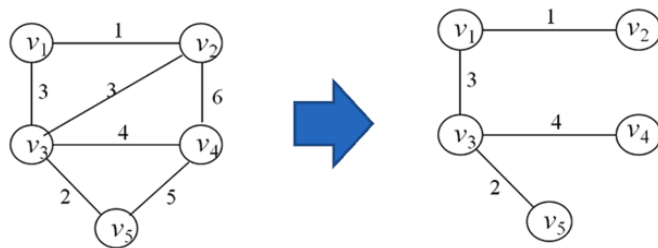
- DFS Spanning Tree / BFS Spanning Tree



1. Graph 복습

• Minimum cost Spanning Tree

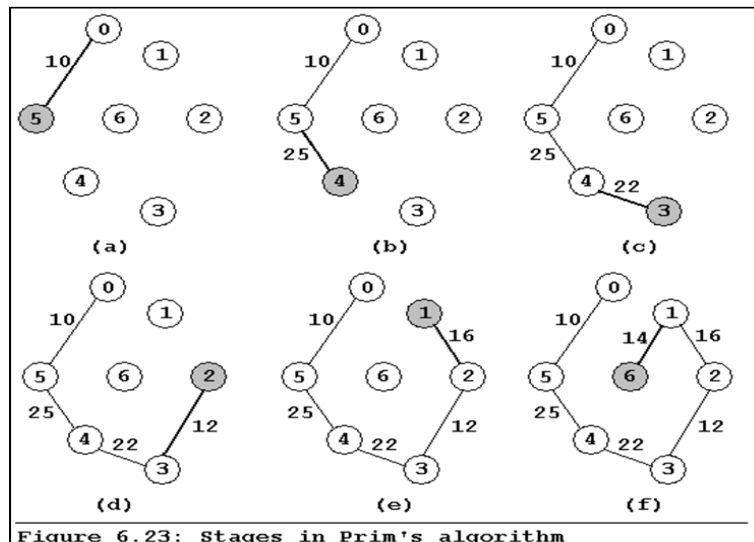
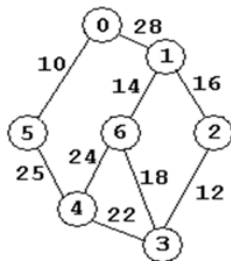
- Spanning Tree 중에서 Edge들의 가중치 합이 최소인 Tree (in Weighted Graph)
- Minimum cost Spanning Tree 조건
 - Edge들의 Weighted 값의 합이 최소
 - 반드시 $n-1$ 개의 edge만 사용
 - 사이클(Cycle)이 생겨서는 안됨
- Minimum cost Spanning Tree 생성 알고리즘
 - Prim algorithm (Using Vertex)
 - Kruskal algorithm (Using Edge)



1. Graph 복습

• Prim algorithm

- 시작 Vertex에서 출발하여 Vertex Set을 단계적으로 확장해 나가면서 Spanning Tree를 구축하는 방법
- 현재 Vertex Set과 인접한 Vertex들 중 연결된 Edge의 Weight가 가장 작은 Vertex를 선택하여 Vertex Set에 추가
- 구현 방법: priority queue || array



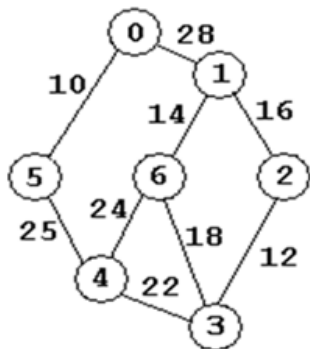
1. Graph 복습

- Prim algorithm

1. 시작 Vertex 선택
2. 시작 Vertex의 거리는 0, 나머지는 INF로 초기화
3. 시작 Vertex에 연결된 Edge들의 Weight를 이용해 거리(distance[])를 갱신
4. distance[]값이 최소인 정점 u 를 선택
5. u 를 Vertex Set에 추가
6. Vertex Set에 포함되지 않은 모든 Vertex들에 대하여 u 와 인접한 Vertex들의 거리를 갱신
7. 4번으로 돌아가 반복

1. Graph 복습

- Prim algorithm



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

#define TRUE 1
#define FALSE 0
#define MAX 7
#define INF 1000

int graph[MAX][MAX] = {
    {0, 28, INF, INF, INF, 10, INF},
    {28, 0, 16, INF, INF, INF, 14},
    {INF, 16, 0, 12, INF, INF, INF},
    {INF, INF, 12, 0, 22, INF, 18},
    {INF, INF, INF, 22, 0, 25, 24},
    {10, INF, INF, INF, 25, 0, INF},
    {INF, 14, INF, 18, 24, INF, 0}
};

int selected[MAX]; //Vertex Set
int dist[MAX];     //distance[]
```


1. Graph 복습

- `int getMinVertex(int n)`
 - Vertex Set의 Vertex들 중에서 연결된 Edge들 중 최소 Weight 값을 갖는 Vertex를 반환하는 함수
 - n은 총 Vertex 개수

```
int getMinVertex(int n) {
    int v, i;

    for (i = 0; i < n; i++)
    {
        if (!selected[i]) {
            v = i;
            break;
        }
    }
    for (i = 0; i < n; i++)
        if (!selected[i] && (dist[i] < dist[v]))
            v = i;
    return v;
}
```

- `void prim(int s, int n)`
 - prim algorithm을 수행하는 함수
 - s는 시작 Vertex, n은 총 Vertex 개수

```
void prim(int s, int n) {
    int i, u, v;

    for (u = 0; u < n; u++) dist[u] = INF;

    dist[s] = 0;
    for (i = 0; i < n; i++)
    {
        u = getMinVertex(n);
        selected[u] = TRUE;

        printf("%d->", u);
        for (v = 0; v < n; v++)
            if (graph[u][v] != INF)
                dist[v] = graph[u][v];
    }
}
```

-> 가장 짧은 노선의 노드에서 dist[] 업데이트

1. Graph 복습

• 실행 결과

```
int main() {
    prim(0, MAX);

    return 0;
}
```

C:\> 선택 C:\WINDOWS\system32\cmd.exe

0-> 5-> 4-> 3-> 2-> 1-> 6->

계속하려면 아무 키나 누르십시오 . . .

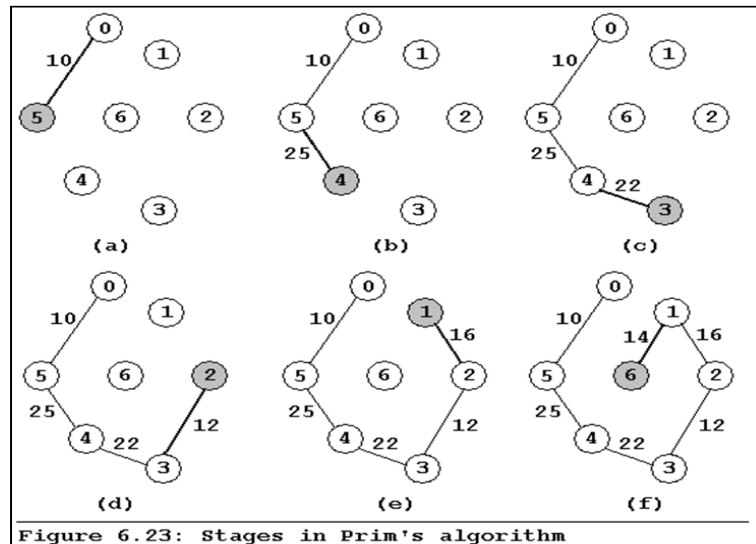
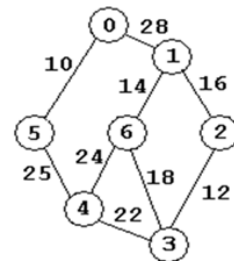
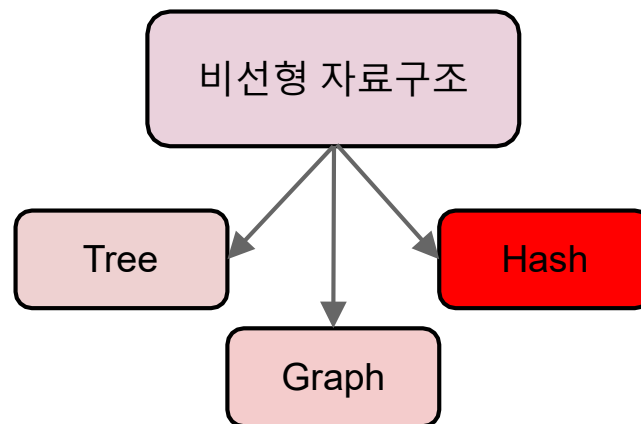
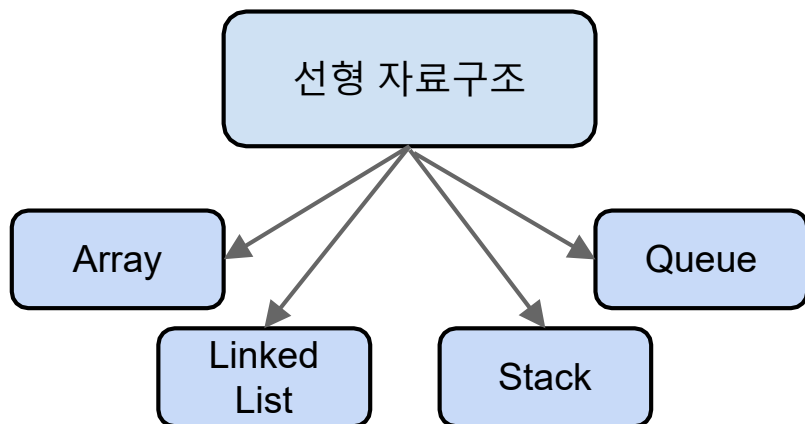


Figure 6.23: Stages in Prim's algorithm

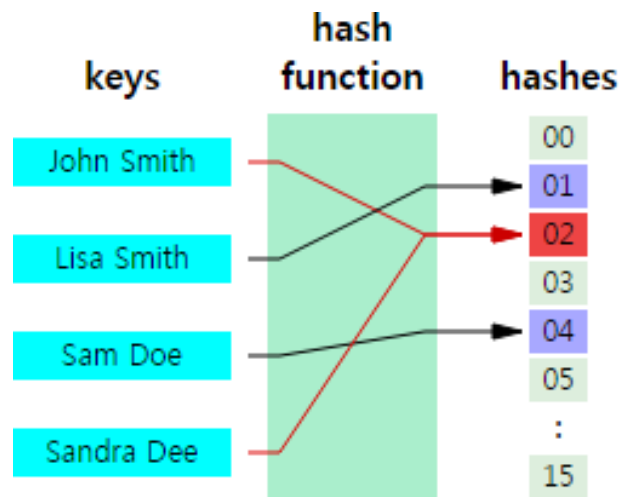
2. Hash

- Data Structure



2. Hash

- **Hash (정의):** 많은 양의 데이터들을 그보다 작은 크기의 테이블로 대응시켜 저장할 수 있는 알고리즘(like Dictionary)
 - **Hash Table** : Hash 기법을 이용해서 데이터들을 저장하는 자료구조
 - 임의의 데이터를 Hash 값으로 대응(mapping)
 - 데이터가 같으면 대응되는 Hash 값은 무조건 같다.
 - 데이터가 달라도 Hash 값은 같을 수 있다.
 - 용도
 - 데이터베이스 내의 항목들을 색인(indexing)하고 검색하는 데 사용
 - 전자서명을 암호화하고 복호화 하는데 사용



2. Hash - 용어

- 키(Key)

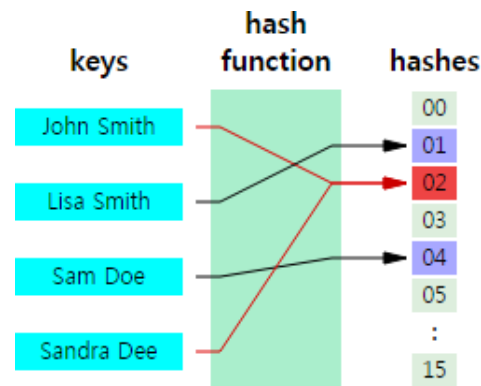
- 탐색, 삽입, 삭제를 하기위한 입력 데이터

- 해싱(Hashing)

- 키 값에 직접 산술 연산(**hash function**)을 적용하여 항목이 저장되어 있는 테이블의 주소를 계산하여 접근하는 방법
 - 이론적으로는 $O(1)$ 시간 내에 탐색이 종료될 수 있음

- 해시 테이블(Hash Table)

- 키 값들이 해싱을 통해 나온 결과대로 저장을 해 두는 자료구조
 - 키 값들이 해시 테이블에 저장될 때 위치가 겹치는 경우(Collision) 여러 방법으로 이를 해결
 - Linear Probing, Chaining

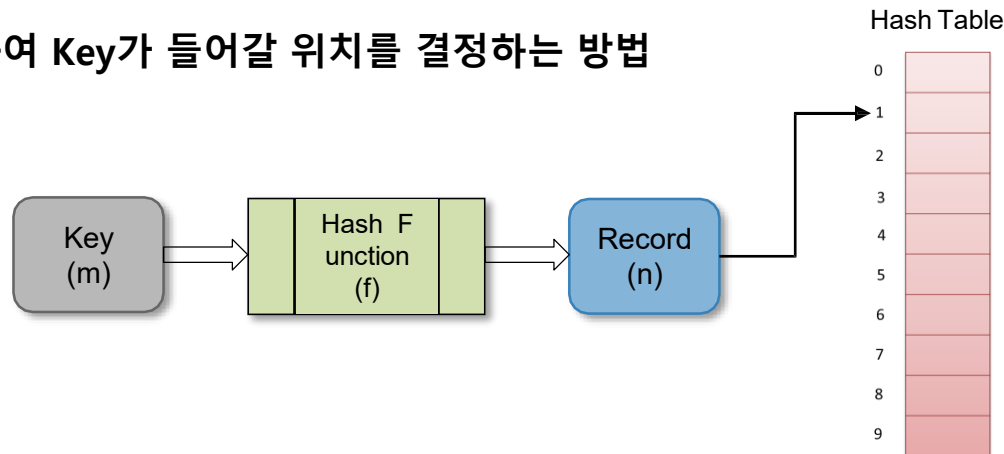


2. Hash - 구조

- Hash 구현에 필요한 자료구조

- Key: 해당 항목 탐색을 위해 사용되는 값
- Mapping function(Hash function): 탐색 키를 작은 정수로 대응시키는 함수 (즉, 탐색 key \rightarrow Mapping function \rightarrow Hash Table 배열의 인덱스)
- Record: Key값을 Hashing 하였을 때 나오는 값
- Hash Table: 일반적으로 배열을 사용

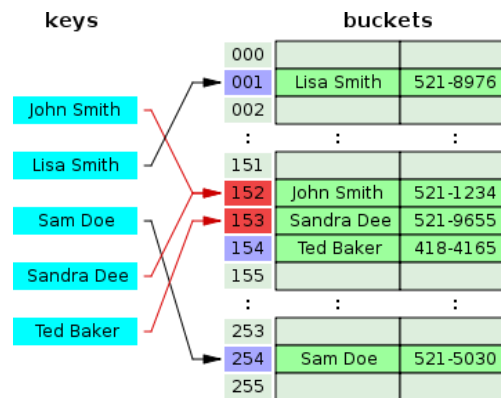
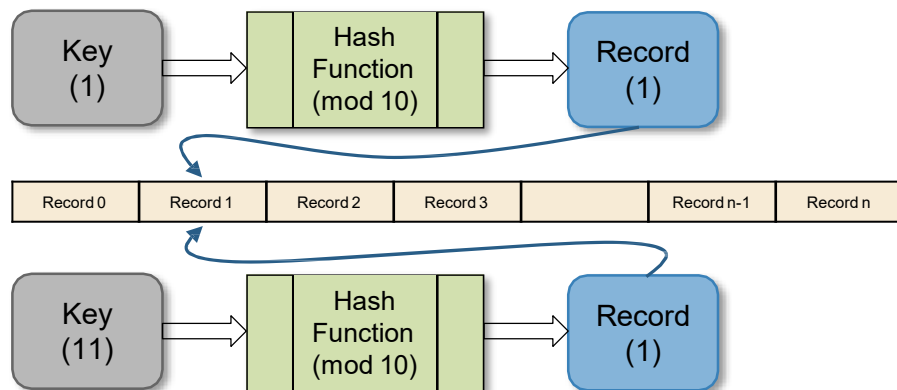
- Hashing = Hash function을 이용하여 Key가 들어갈 위치를 결정하는 방법



2. Hash - 원리

• Collision

- 서로 다른 Key 값을 갖는 항목들이 같은 Hashing 결과 값(해시 주소)를 가지는 현상



- 충돌(Collision) 발생시의 해당 항목을 Hash Table에 추가하는 방법(즉, 해결하는 방법)
 - Hash Table의 다른 위치에 추가하는 방법([Linear probing](#))
 - Hash Table의 하나의 위치가 여러 개의 항목을 저장할 수 있도록 Hash Table의 구조를 변경하는 방법([Chaining](#))

2. Hash

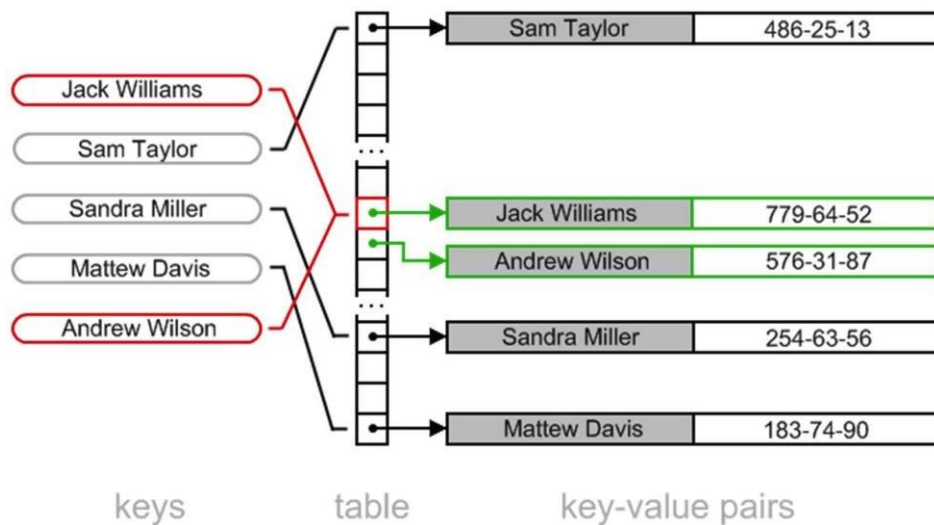
- **Linear probing(선형 조사법)**

- Hash Table의 특정 Bucket에서 충돌 발생 시, 비어 있는 Bucket을 탐색(Probing)하는 방법
- 만일 충돌 위치가 $HT[k]$ 인 경우
 - $HT[k+1]$ 이 비어 있는지 확인
 - $HT[k+1]$ 이 비어 있다면 $\rightarrow HT[k+1]$ 에 항목 삽입
 - $HT[k+1]$ 이 비어 있지 않다면 $\rightarrow HT[k+2]$ 가 비어 있는지 확인
 - ...
- 따라서, Hash function $h(k)$ 에 대하여,
 - $h(k), h(k)+1, h(k)+2, \dots h(k)+n$ 의 위치를 탐색함 (n : hash table 크기)

Bucket: Hash Table에 각각의 데이터들을 넣을 수 있는 공간

2. Hash

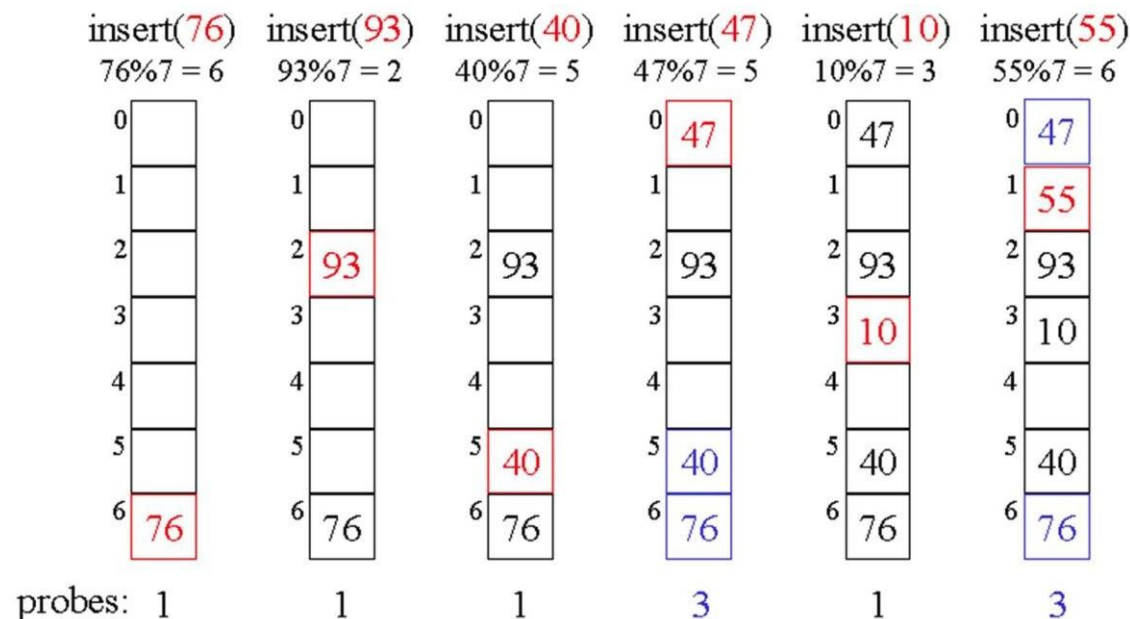
- Linear probing(선형 조사법)
 - Example



2. Hash

- Linear probing(선형 조사법)

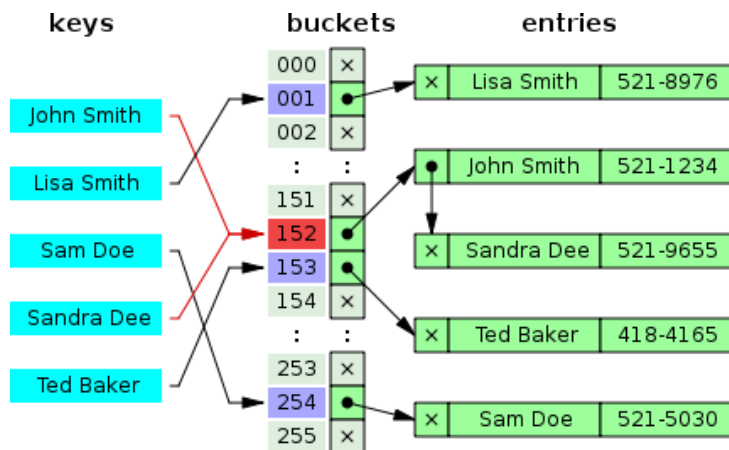
- Example



2. Hash

• Chaining(연쇄법)

- Hash Table의 구조를 변경하여 각 Bucket에 하나 이상의 Key 값을 저장할 수 있도록 하는 방법
- Bucket에 Key 값을 효과적으로 추가/삭제하기 위하여 Linked List 자료구조를 활용함.
- Bucket내에서의 원하는 Key 값의 검색은 Linked List의 linear search를 사용함.



2. Hash

- **Chaining(연쇄법)**

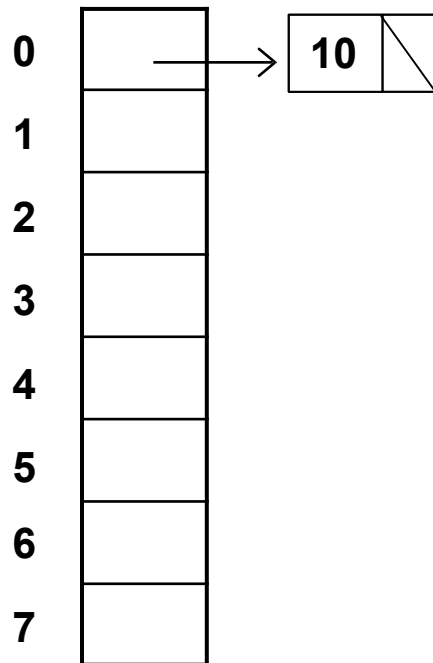
- Hash function: $h(k) = k \% 10$
- Key values: 10, 56, 44, 1, 20, 36
 - Insert 10: $h(10) = 0$
 - Insert 56: $h(56) = 6$
 - Insert 44: $h(44) = 4$
 - Insert 1: $h(1) = 1$
 - Insert 20: $h(20) = 0$
 - Insert 36: $h(36) = 6$

0	
1	
2	
3	
4	
5	
6	
7	

2. Hash

- Chaining(연쇄법)

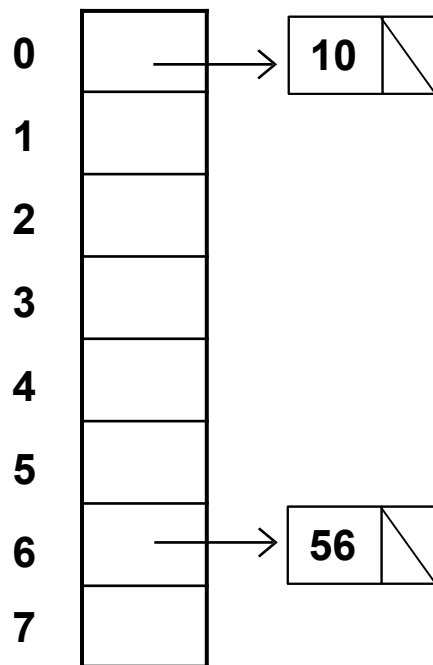
- Hash function: $h(k) = k \% 10$
- Key values: 10, 56, 44, 1, 20, 36
 - Insert 10: $h(10) = 0$
 - Insert 56: $h(56) = 6$
 - Insert 44: $h(44) = 4$
 - Insert 1: $h(1) = 1$
 - Insert 20: $h(20) = 0$
 - Insert 36: $h(36) = 6$



2. Hash

• Chaining(연쇄법)

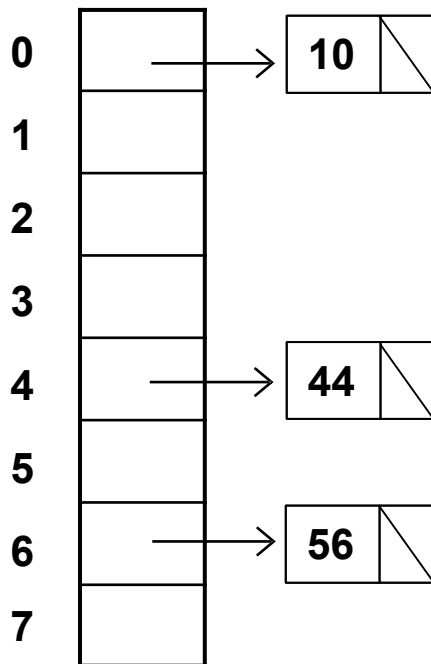
- Hash function: $h(k) = k \% 10$
- Key values: 10, 56, 44, 1, 20, 36
 - Insert 10: $h(10) = 0$
 - Insert 56: $h(56) = 6$
 - Insert 44: $h(44) = 4$
 - Insert 1: $h(1) = 1$
 - Insert 20: $h(20) = 0$
 - Insert 36: $h(36) = 6$



2. Hash

• Chaining(연쇄법)

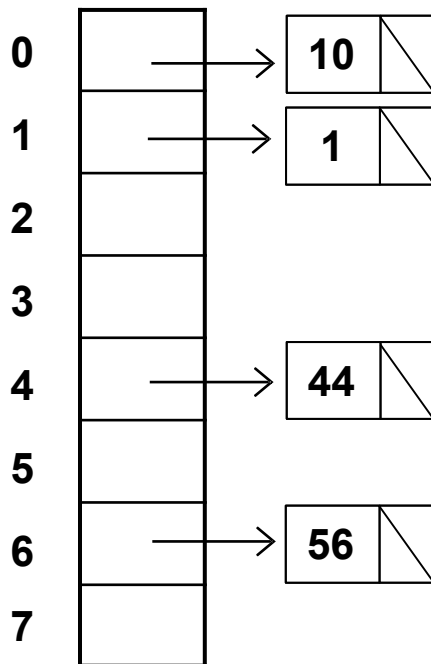
- Hash function: $h(k) = k \% 10$
- Key values: 10, 56, 44, 1, 20, 36
 - Insert 10: $h(10) = 0$
 - Insert 56: $h(56) = 6$
 - Insert 44: $h(44) = 4$
 - Insert 1: $h(1) = 1$
 - Insert 20: $h(20) = 0$
 - Insert 36: $h(36) = 6$



2. Hash

- **Chaining(연쇄법)**

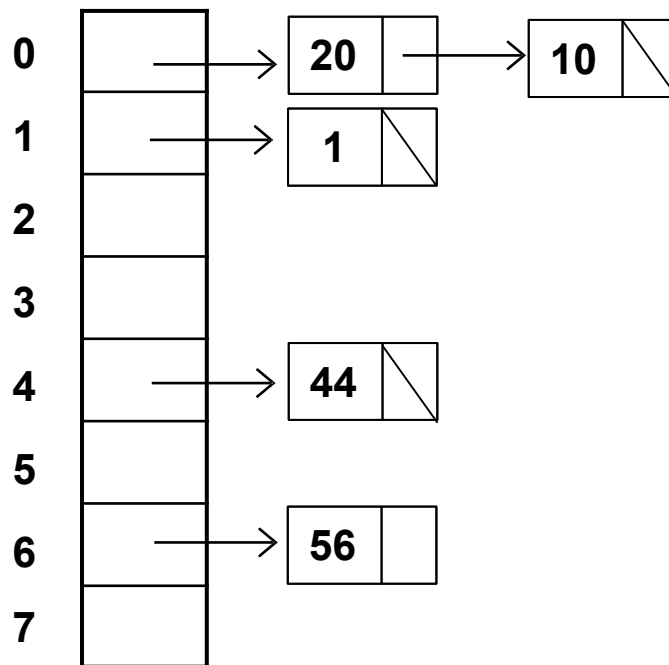
- Hash function: $h(k) = k \% 10$
- Key values: 10, 56, 44, 1, 20, 36
 - Insert 10: $h(10) = 0$
 - Insert 56: $h(56) = 6$
 - Insert 44: $h(44) = 4$
 - Insert 1: $h(1) = 1$
 - Insert 20: $h(20) = 0$
 - Insert 36: $h(36) = 6$



2. Hash

• Chaining(연쇄법)

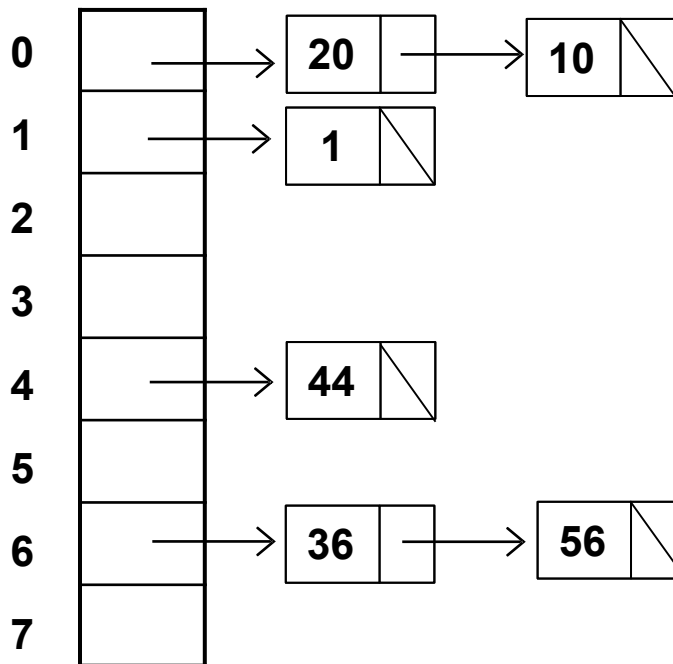
- Hash function: $h(k) = k \% 10$
- Key values: 10, 56, 44, 1, 20, 36
 - Insert 10: $h(10) = 0$
 - Insert 56: $h(56) = 6$
 - Insert 44: $h(44) = 4$
 - Insert 1: $h(1) = 1$
 - Insert 20: $h(20) = 0$
 - Insert 36: $h(36) = 6$



2. Hash

- **Chaining(연쇄법)**

- Hash function: $h(k) = k \% 10$
- Key values: 10, 56, 44, 1, 20, 36
 - Insert 10: $h(10) = 0$
 - Insert 56: $h(56) = 6$
 - Insert 44: $h(44) = 4$
 - Insert 1: $h(1) = 1$
 - Insert 20: $h(20) = 0$
 - Insert 36: $h(36) = 6$



3. Hash 실습

- Hash Data Type

Hash Table			
ListNode	[0]	→ Element	Null
ListNode	[1]	→ Element	Null
ListNode	[2]	→ Element	Null
ListNode	[3]	→ Element	Null
ListNode	[4]	→ Element	Null
ListNode	[5]	→ Element	Null
ListNode	[6]	→ Element	Null
ListNode	[7]	→ Element	Null

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define KEY_SIZE 10
6  #define TABLE_SIZE 5
7  #define equal(e1,e2) (!strcmp(e1.key,e2.key))
8
9  typedef struct Element {
10     char key[KEY_SIZE];
11 }Element;
12
13 typedef struct ListNode {
14     Element item;
15     struct ListNode *link;
16 }ListNode;
17
18 ListNode *hashTable[TABLE_SIZE];
19
20 void initTable(ListNode* ht[]) {
21     int i;
22     for (i = 0; i < TABLE_SIZE; i++)
23         ht[i] = NULL;
24 }
25

```

3. Hash 실습

- Hash Data Type

- Hash Table의 Linked List 구현을 위한 Node 구조체 선언
- Hash Table 선언
- Hash Table 초기화

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define KEY_SIZE 10
6  #define TABLE_SIZE 5
7  #define equal(e1,e2) (!strcmp(e1.key,e2.key))
8
9  typedef struct Element {
10     char key[KEY_SIZE];
11 }Element;
12
13 typedef struct ListNode {
14     Element item;
15     struct ListNode *link;
16 }ListNode;
17
18 ListNode *hashTable[TABLE_SIZE];
19
20 void initTable(ListNode* ht[]) {
21     int i;
22     for (i = 0; i < TABLE_SIZE; i++)
23         ht[i] = NULL;
24 }
25
```

3. Hash 실습

- Chaining

- 문자열 입력 시 숫자 형태(ASCII)로 변환하기 위한 함수
 - 해시 함수 정의 (mod Table size)
- > key[i]는 ASCII 코드를 반환(문자열 경우)

```
26 int transform(char *key) {  
27     int i;  
28     int number = 0;  
29     int size = strlen(key);  
30     for (i=0; i<size; i++)  
31         number = number + key[i];  
32     return number;  
33 }  
34  
35 int hashFunction(char *key) {  
36     return transform(key) % TABLE_SIZE;  
37 }  
38
```

3. Hash 실습

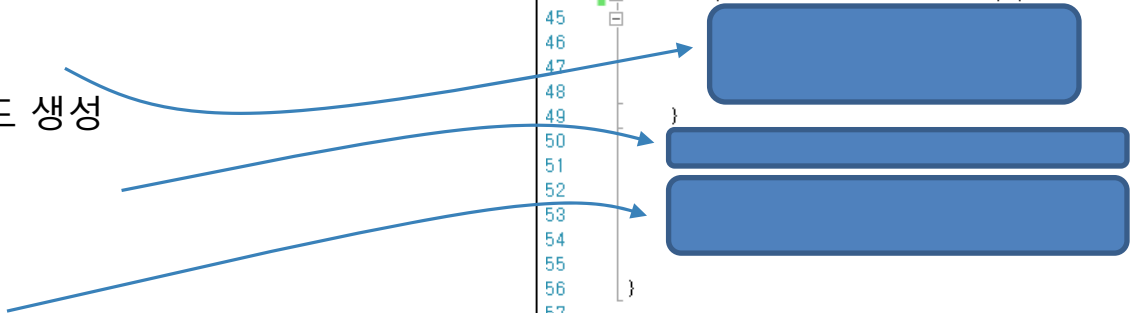
• Chaining

- Hashing 을 통해 Hash Table에 데이터를 넣어주는 함수
- 해당 데이터가 중복되어 있는지 리스트 내에서 탐색
- 추가를 위해 새로운 노드 생성
- 리스트의 맨 앞에 추가

```

39 void addHashTable(Element item, ListNode* ht[]) {
40     int hashValue = hashFunction(item.key);
41     ListNode *ptr;
42     ListNode *node = ht[hashValue];
43
44     for (; node; node = node->link) {
45
46     }
47
48 }
49
50
51
52
53
54
55
56
57

```



3. Hash 실습

• Chaining

- Hashing 을 통해 Hash Table에 데이터를 넣어주는 함수
- 해당 데이터가 중복되어 있는지 리스트 내에서 탐색
- 추가를 위해 새로운 노드 생성
- 리스트의 맨 앞에 추가

```

39 void addHashTable(Element item, ListNode* ht[]) {
40     int hashValue = hashFunction(item.key);
41     ListNode *ptr;
42     ListNode *node = ht[hashValue];
43
44     for (; node; node = node->link) {
45         if (equal(node->item, item)) {
46             printf("중복 삽입 에러\n");
47             return ;
48         }
49     }
50     ptr = (ListNode*)malloc(sizeof(ListNode));
51
52     ptr->item = item;
53     ptr->link = ht[hashValue];
54     ht[hashValue] = ptr;
55
56 }
57

```

3. Hash 실습

• Chaining

- Key 값이 들어 있는 node를 찾는 함수
- Hash Table에 들어있는 값들을 출력하는 함수

```

58 void hashSearch(Element item, ListNode* ht[]) {
59     ListNode *node;
60     [redacted]
61     for (node = ht[hashValue]; node; node = node->link) {
62         [redacted]
63     }
64     printf("탐색 실패\n");
65 }
66
67
68
69
70

```

```

71
72 void printHashTable(ListNode* ht[]) {
73     int i;
74     ListNode *temp;
75     for (i = 0; i < TABLE_SIZE; i++)
76     {
77         [redacted]
78     }
79 }
80
81
82
83

```


3. Hash 실습

• Chaining

- Key 값이 들어 있는 node를 찾는 함수
- Hash Table에 들어있는 값들을 출력하는 함수

```

58 void hashSearch(Element item, ListNode* ht[]) {
59     ListNode *node;
60
61     int hashValue = hashFunction(item.key);
62     for (node = ht[hashValue]; node; node = node->link) {
63         if (equal(node->item, item)) {
64             printf("탐색 성공: 존재합니다.\n");
65             return;
66         }
67     }
68     printf("탐색 실패\n");
69 }
70

```

```

71
72 void printHashTable(ListNode* ht[]) {
73     int i;
74     ListNode *temp;
75     for (i = 0; i < TABLE_SIZE; i++)
76     {
77         temp = ht[i];
78         for (; temp; temp = temp->link)
79             printf("%s\t", temp->item.key);
80         printf("\n");
81     }
82 }
83

```

3. Hash 실습

- main process

- Key 값을 Hash Table에 추가
- Key 값을 Hash Table에서 탐색
- Hash Table에 저장된 모든 Key 값을 출력

```
84 int main()
85 {
86     Element temp;
87     int op;
88
89     while (1) {
90         printf("연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = ");
91         scanf("%d", &op);
92
93         if (op == 3) break;
94         if (op == 2) printHashTable(hashTable);
95         else {
96             printf("키 입력: ");
97             scanf("%s", temp.key);
98             if (op == 0)
99                 addHashTable(temp, hashTable);
100             else if (op == 1)
101                 hashSearch(temp, hashTable);
102         }
103     }
104
105     return 0;
106 }
107 }
```

3. Hash 실습

- main process
 - 결과

```

C:\WINDOWS\system32\cmd.exe
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 0
키 입력: Hello
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 0
키 입력: abc
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 0
키 입력: def
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 2
Hello

def
abc
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 1
키 입력: def
탐색 성공: 존재합니다.
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 0
키 입력: World
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 2
World Hello

def
abc
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 0
키 입력: data
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 1
키 입력: datas
탐색 실패
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 2
data World Hello

def
abc
연산 입력(0: 추가, 1: 탐색, 2: 출력, 3: 종료) = 3
계속하려면 아무 키나 누르십시오 . . .
    
```

감사합니다.

과제 제출 기한: 2025년 6월 18일 23:59분 (LMS 제출 시간 기준)

제출형식:

- 실습6 BST2 ~ 실습11 Hashing까지 진행한 실습코드를 하나의 압축파일로 제출
- 과제 탭에 안내된 실습 코드를 압축하여 LMS 과제 탭에 제출

궁금한 것이 생기면 언제든지 질문하시면 됩니다 ☺

- 공업센터본관 304호로 방문하시거나
- jeongiun@hanyang.ac.kr나 speedpaul@hanyang.ac.kr로 연락 바랍니다.
- 담당조교: 이범기, 이상윤