Hash Search

Date	
작성일시	@2022년 8월 3일 오전 10:29
강의 번호	
유형	
강사명	
강의자료	☐
노션 복습	☐
코딩 복습	☐
주말숙제(교제)	☐
정리	☐

해시 탐색법_Hash Search

👉 선형 탐색이나 이진 탐색의 전제 조건은 어떤 데이터가 어떤 요소에 들어 있는지 전혀 모르는 상태에서 검색을 했다. 그러나 해시 탐색은 데이터의 내용과 저장한 곳의 요소를 미리 연계해 둠으로써 극히 짧은 시간안에 탐색할 수 있도록 고안된 알고리즘이다.

해시 탐색은 '데이터를 데이터와 같은 첨자의 요소에 넣어두면 한 번에 찾을 수 있다는 아이디어에서 시작한다.

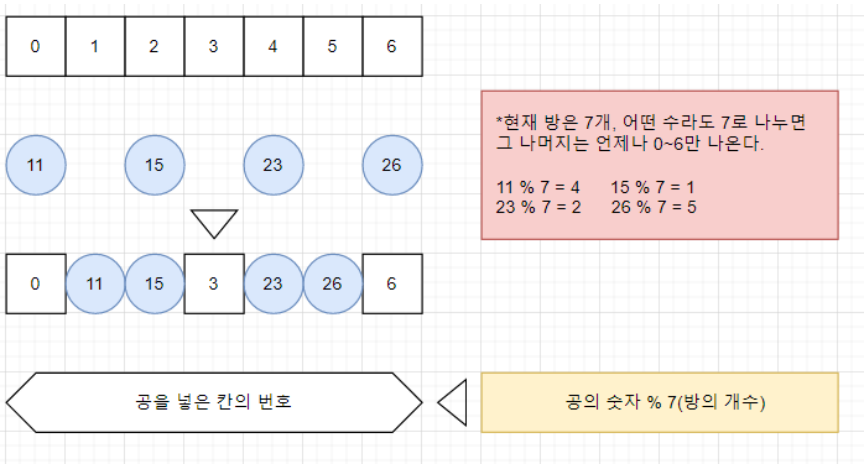
확실하고 쉽고 빠르게 찾을 수는 있지만 문제는 단 1개 즉 10000데이터를 저장하려면 공간이 9999의 불필요한 공간을 준비해야 한다.

따라서 좀 더 효율적으로 저장하기 위해서 일정한 계산을 하여 그 계산 결과값과 같은 첨자의 요소에 보관하는 방법을 고안하였다.

▼ 해시탐색법으로 데이터를 보관하는 알고리즘

- 해시 함수는 데이터릐 저장소의 첨자를 계산하는데 사용한다.

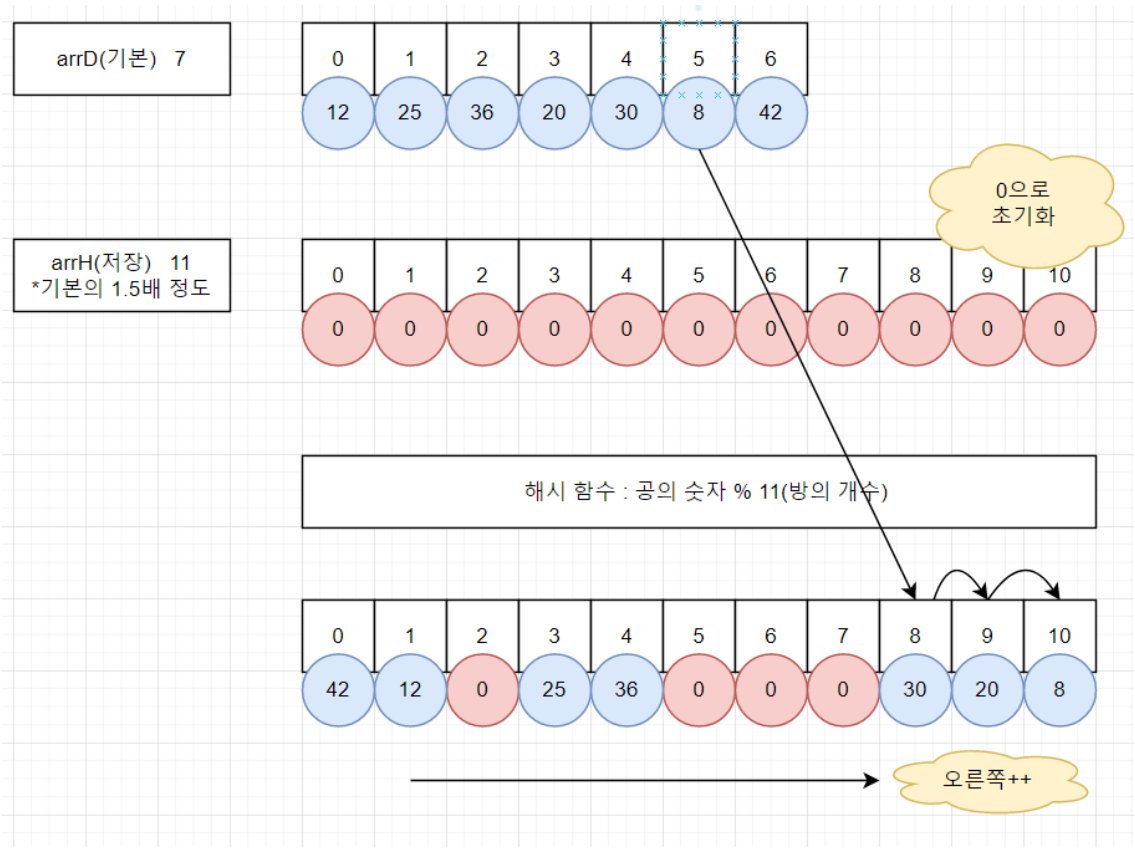
- 저장고의 첨자가 겹치는 것을 ‘충돌’이라고 한다.
- 충돌이 발생하면 옆의 빈자리에 보관한다.



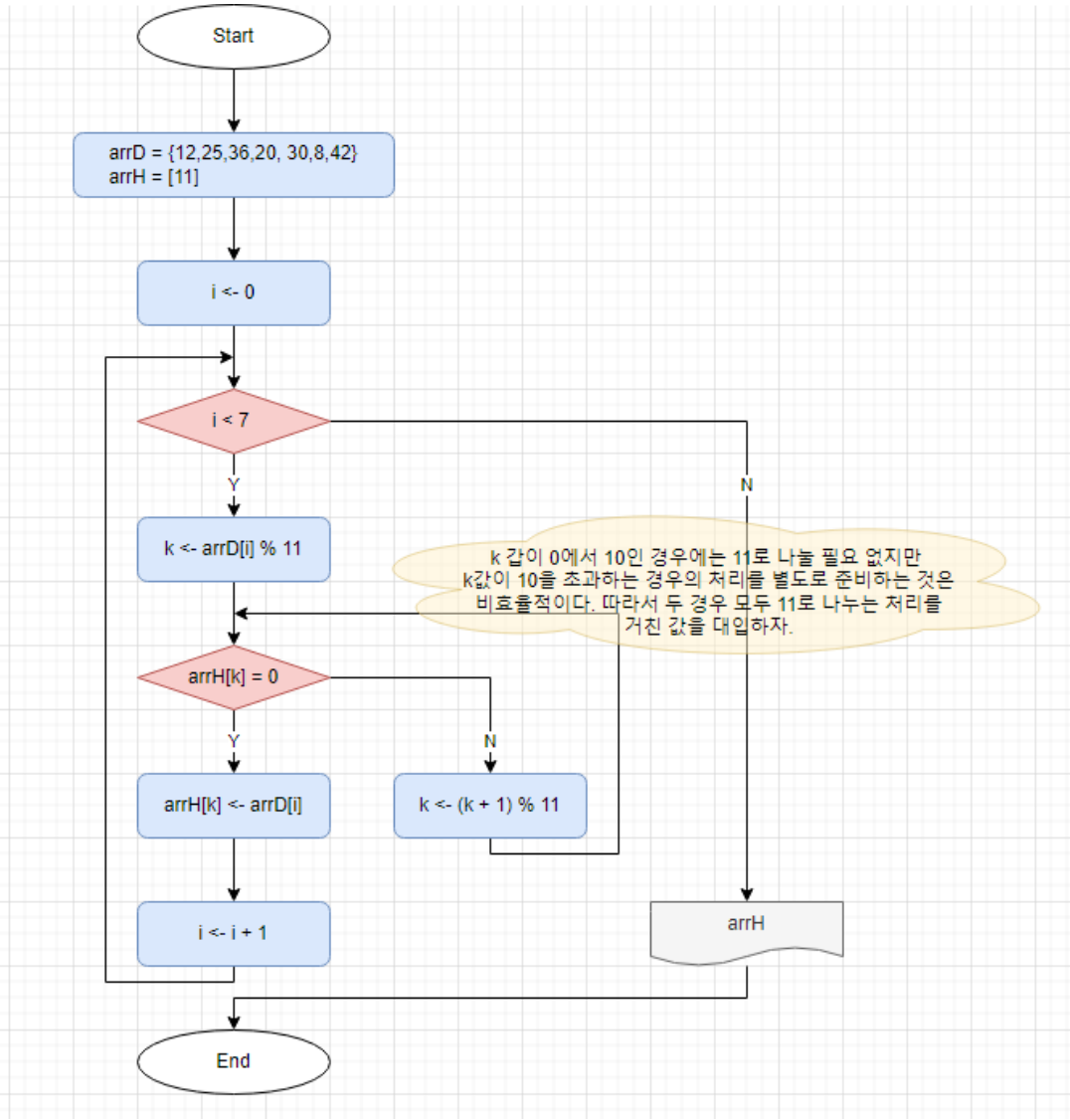
- 타 알고리즘과 차이점

  ▼ 해시 함수로 데이터를 저장하는 알고리즘

  - 배열을 2개 준비한다.
    - 하나는 저장할 데이터 (기본 데이터 배열_초기값),
    - 다른 하나는 실제 데이터(저장 데이터 배열)
  - 해시 함수를 사용하여 데이터를 저장할 때, 같은 나머지가 발생하게 되면 그 다음 인덱스의 위치에 저장하여 해결한다. 그런데 또 다른 데이터가 존재한다면 다시 한 번 다음 인덱의 위치로 이동하여 저장한다.
  - 저장 데이터의 배열의 개수는 기본 데이터의 1.5배가 적당한 것으로 연구되어 있다. 너무 많은 배열을 준비하면 메모리의 소모가 발생하고 적게 배열을 준비하면 충돌이 많이 발생하게 되어 불필요한 검색과정이 추가가 필요해진다.



▼ Flow chart - input



▼ Java code - input

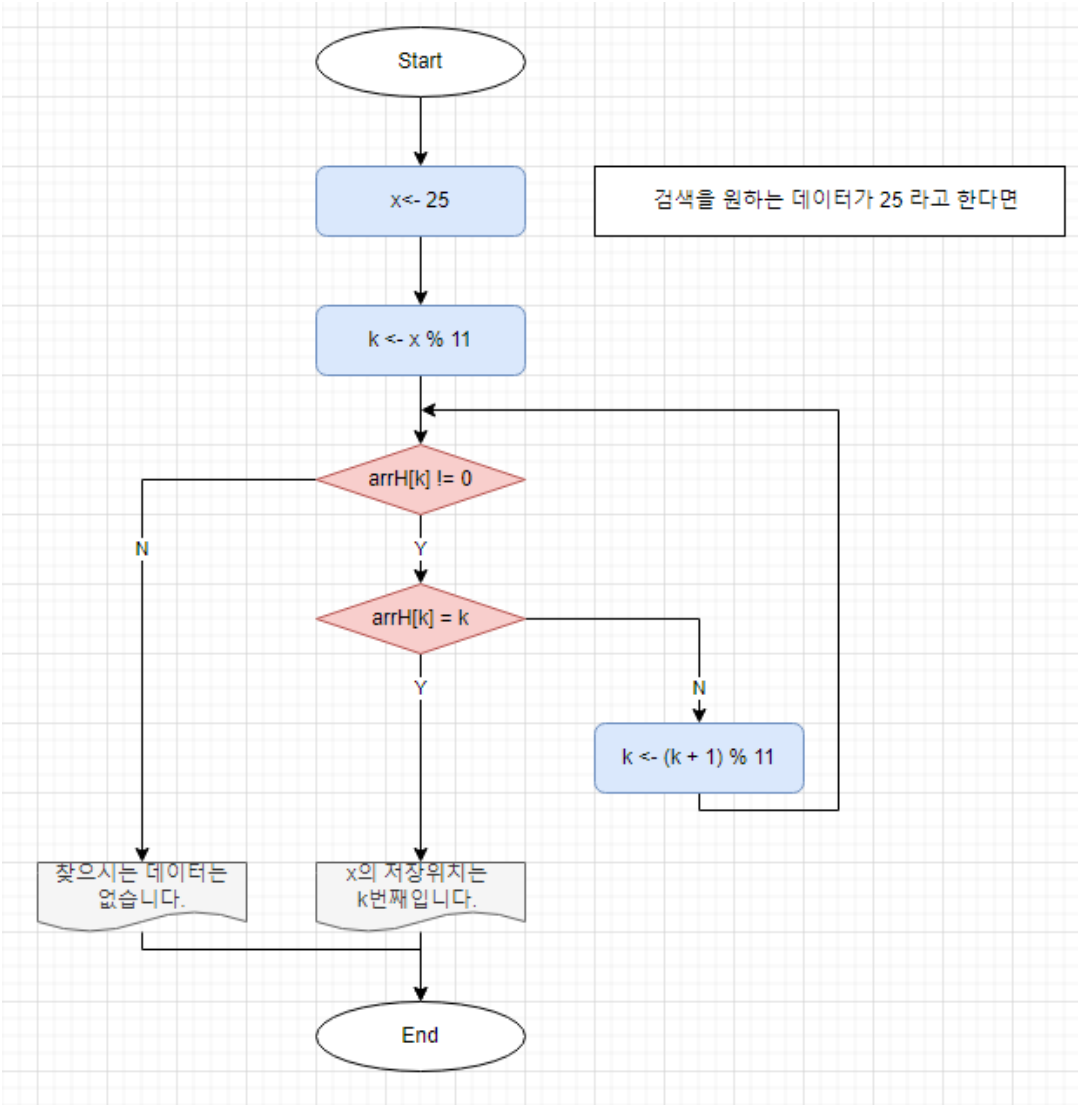
```
package HashSearch;

public class Input {

  public static int[] Inputs(int[] arrD, int[] arrH){

  for (int i = 0; i < arrD.length; i++) {
    int k = arrD[i] % 11;

    while (true) {
      if (arrH[k] != 0) {
        k = (k + 1) % 11;
      } else {
```

```
          arrH[k] = arrD[i];
          break;
        }
      }
    }
    return arrH;
  }
}
```

▼ 해시 함수로 데이터를 검색하는 알고리즘

- 데이터 탐색도 저장할 때 사용한 동일 해시함수를 사용한다.
- 탐색 데이터가 존재하지 않을 경우의 처리가 중요하다.

▼ Flow chart - Outputs



▼ Java code -Outputs

```
package HashSearch;

import java.util.*;
import HashSearch.Input;

public class Outputs {

  public static void main(String[] args) {
    int[] arrD = { 12, 25, 36, 20, 30, 8, 42 };
    int[] arrH = new int[11];

    Input ip = new Input();
    int[] arr = ip.Inputs(arrD,arrH);
    System.out.println(Arrays.toString(arr));

    System.out.println();

    int x = 8;
    int k = x;
    k = k % 11;

    while (arrH[k] != 0) {
      if (arrH[k] == x) {
        System.out.println(x + "의 저장 위치는 " + k + "번째 입니다.");
        break;
      }
      else {
        k = (k + 1) % 11;
      }
    }
    if (arrH[k] == 0) {
      System.out.println(x + "값은 존재하지 않습니다.");
    }
  }
}
Outputs :
[42, 12, 0, 25, 36, 0, 0, 0, 30, 20, 8]

8의 저장 위치는 10번째 입니다.
```

▼ Java code

```
package HashSearch;

public class HashSearch {

  public static void main(String[] args) {
```

```
    int[] arrD = { 12, 25, 36, 20, 30, 8, 42 };
    int[] arrH = new int[11];

          //해시태그 저장

    int i = 0;

    for (i = 0; i < arrD.length; i++) {
      int k = arrD[i] % 11;

      while (true) {
        if (arrH[k] != 0) {
          k = (k + 1) % 11;
        } else {
          arrH[k] = arrD[i];
          break;
        }
      }
    }
    for (int j = 0; j < arrH.length; j++) {
      System.out.print(arrH[j] + " ");

    }

        // 해시태그 출력

    System.out.println();

    int x = 8;
    int k = x;
    k = k % 11;

    while (arrH[k] != 0) {
      if (arrH[k] == x) {
        System.out.println(x + "의 저장 위치는 " + k + "번째 입니다.");
        break;
      }
      else {
        k = (k + 1) % 11;
      }
    }
    if (arrH[k] == 0) {
      System.out.println(x + "값은 존재하지 않습니다.");
    }
  }
}
Outputs :
42 12 0 25 36 0 0 0 30 20 8
8의 저장 위치는 10번째 입니다.
```