CHAP 8. Search Structure (탐색구조)

- Searching - 저장된 자료중에서 특정한 자료를 찾는 방법 (예: 데이타들이 X_1, X_2, ...X_n 으로 구성되어 있을때, 특정값 X_i 를 찾는 것)

1) 선행조건: 검색이 효율적으로 수행되려면, 검색대상이 되는 자료는
   - 적절한 자료구조에 의해 저장되어야 하고,
   - 자료는 식별하기 위한 Key 가 지정되어야 하며,
   - 능률적인 검색기법이 적용되어야 한다.

2) Key 와 Record 를 연관 시키는 방법:
   - 독립적인 Key Table 구성, Table 로 부터 특정키를 가르키는 외부키의 설정방법

- 검색방법 (Searching method)

1) **선형검색 (Linear Search / sequential search)**
2) **이진검색 (Binary Search)**
3) **Fibonacci 검색**
4) **보간검색 (Interpolation Search)**

5) 트리검색 (BST, AVL, B-TREE, DFS, BFS, …)
6) Hashing

1) 순차탐색 (Linear/Sequential Search)

가장 간단한 탐색기법 (첫번째부터 차례로 Key 와 비교해 가면서 마지막 데이터까지 찾는 방법)

(만약 데이터가 8 개라면, 최악의 경우 8 번 비교)

Method 1)

```
Seqsrch (S, i, key)
{
    i = 1;
    while (S[i] != key) && (i <= arraysize)    do
            i = i +1;
    if (i > arrsize)    print (“NOT found...”);
 }
```

Method 2) // recursion

```
Seqsrch(S, i, n, key)
{
    if (i > n)
          { print (“NOT FOUND”);
            return;
          }
    elseif   (S[i] == key)
          {   print(“FOUND”);
              return;
           }
    else     Seqsrch(S, i+1, n, key);
 }
```

2) 이진탐색 (Binary Search)

- Record 들이 오름차순/내림차순으로 정리되어 있을 때, 원하는 key 를 찾는 경우, 우선 중간에 위치한 값 S[mid]을 선택하여 찾는 key 와 비교한다.

  ⇨ 3 가지 case 발생

  - key < S[mid] => S[1] ...S[mid-1]. 에 찾는값이 있음
  - key > S[mid] => S[mid+1] ,... S[mid] 에 찾는값이 있음
  - key = S[mid] => S[mid] 이 찾는값. 탐색끝

  ⇨ 비교 대상 record 는 매번 반 이상 줄어든다.

- Algorithm:

  1) Initialize: Low = 1; High = n; Found = false
  2) While NOT FOUND and Low < High
     mid = ⌊(Low + High)/2⌋
     if key > S[mid] Low = mid+1
     else if key < S[mid] High = mid-1
     else found = true
     endWhile

- Recursive version of Binary Search

```
BSH(S, Low, High, Key)  {
  if  (Low < High)  {
    mid = ⌊(Low+High)/2⌋
    if  (key  > S[mid])  BSH(S, (mid+1), High, key);
    elseif (key < S[mid])  BSH(S, Low, (mid-1), key);
    else found = true;
  }
    else  print("NOT FOUND...");  }
```

3) 피보나치 탐색 (FIBONACCI Search)

	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9
수열:	0,	1,	1,	2,	3,	5,	8,	13,	21,	34,

$$\begin{cases} F_0 = 0, \quad F_1 = 1 \\ F_i = F_{i-1} + F_{i-2}, \quad n \geq 2 \end{cases}$$

- Advantage: 다음 비교할 원소의 위치 선택시
  (이진탐색 -> 나눗셈
  피보나치 -> 덧셈/뺄셈 이용 => 빠르다)

- Algorithm: (Assume **$n = F_i - 1$**)
  첫번째 비교는 F_{i-1} 번째 원소와 비교, 그 결과에 따라
  1) key < A[F_{i-1}] 일때, A 의 index 1 부터 F_{i-1} - 1 까지 recursive 하게 다시 탐색
  2) key = A[F_{i-1}] 일때, success
  3) key > A[F_{i-1}] 일때, A 의 index F_{i-1} + 1 부터 F_i - 1 까지 recursive 하게 다시 탐색

Let **$n = F_i - 1$**, 초기화: **$F_{i-1} = i$**, **$F_{i-2} = p$**, **$F_{i-3} = q$**

If (key < A[i]) if q=0, then key is not in the data
else { **i= i-q, t=p; p=q; q= t-q;** }
else (key> A[i]) if p=1, then key is not in the data
else { **i= i+q; p= p-q; q= q-p}**
else (key=A[i]) then found.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	7	11	16	20	25	26	29	30	33	35	46	50	53	54	62	64	75	88	92

ex) Find 16?
$n = 20$ 일때, $20 = F_8 - 1$ ($F_8 = 21$)
F_{8-1} => F_7 => 13 (i = 13, p= 8, q= 5)

- (A[13] > 16): (**i** = 13-5=**8**, t=8, **p** => q=5 **q** => t-q = 8-5=3)
- (A[8] > 16): (**i** = 8-3=5, t= 5, **p** =>3 **q** => 5-3 =2)
- (A[5] > 16): (**i** = 5-2=3, t= 3, **p** =>2 **q** => 3-2 =1)
- (A[3] < 16): (i= 3+1=4, **p** =2-1=1 **q** = 1-1=0)
- (A[4] = 16) Found.

Ex) Find 62?

n = 20 일때, 20 = F_8 - 1 (F_8 = 21)

F_{8-1} => F_7 => 13 **(i = 13, p= 8, q= 5)**

{ i= i-q, t=p; p=q; q= t-q; }

{ i= i+q; p= p-q; q= q-p}

- (A[13] < 62): (**i** = 13+5=**18**, **p** => 3 **q** => 5-3 =2)
- (A[18] > 62): (**i** = 18-2=16, t= 3, **p** =>2 **q** => 5-3 =2)
- (A[16] = 62): found

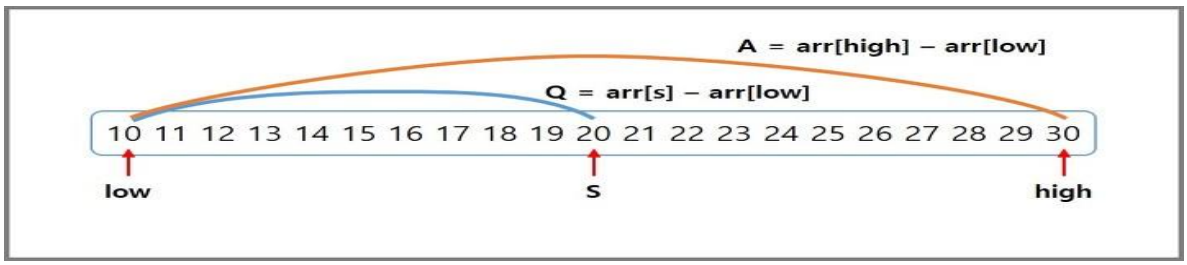
```
int fibSearch (int key) {    // key = A[x],    0 < x < A[i+1]
    int mid = A[i];       int p = A[i-1]       int q    = A[i-2]
    for (;;) {
        if (key == A[mid) return mid; //key found
        else if (key < A[mid]) {
            if (q == 0) return    Not found;
            mid = mid - q;
            temp = p;
            p = q;
            q = temp - q;
        }
        else if (key > A[mid]) {
            if (p == 1) return Not found;
            mid = mid + q;
            p = p - q;
            q = q - p;
        }
    }
```

5) 보간 탐색 (Interpolation Search)

- 사전에서 색인을 보고 찾듯이 찾는 방법 - 있음직한 부분을 계산해서 비교하는 방법
- 이진 탐색의 비효율성(무조건 반씩 줄여 나감)을 개선시킨 알고리즘
- 데이터의 값과 그 데이터가 저장된 위치의 index 값이 비례한다고 가정.

- Low: 탐색대상의 시작 인덱스 값
- High: end 인덱스값
- s: 찾는 데이터가 저장된 위치의 인덱스 값

- A:Q = (high-low): (s-low);
- s = Q/A(high-low):(s-low)
- s = low + (high-low) * (key - A[low]) / (A[high]- A[low])




```
int search(key, a[], n) {
  int low=0,   high= n-1,   mid;
  while (low <= high) {
        mid= low + (high-low) * (key - A[low]) / (A[high]- A[low])
        if (key < A[mid])   high= mid-1;
        else if (key > A[mid]) low= mid+1;
        else return mid;
   }
  return -1; }
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	3	5	7	10	11	12	14	23	24	25	30	33	38	45	49	55	56	66	77

- Find 38? **0**+ (19-0)* (38-1)/(77-1) = 37/4 = 9.. ≒ => **9(mid).**
  => (A[9] < 38); low = 9+1= 10, high = 19,
  mid= **10** + (19-10)*(38-25)/(77-25) =12.2.. => **12(mid)**

  => (A[12] < 38); low = 12+1 =13, high = 19,
  mid= **13**+ (19 -13)*(38 – 33)/(77-33) = 13… =>**13 (mid)**

  ⇨ (A[13] = 38) found

- Find 23 ?
  0+ (19-0)* (23-1)/(77-1) = 5 ≒ => **5(mid).**
  => (A[5]=11 < 23); low = 5+1=6 high = 19
  mid= **6** + (19 - 6)*(23-12)/(77-12) =8.... => **8(mid)**

  => (A[8] =23) found

- Final Notice

1. Lab14 Hashing – chaining method
2. Final Exan – Quiz (범위: 전체)
   . 6 월 22 일 (9 시 – 10 시 15 분)