**———————————————————————————————————————————————————**

[1] 기초 문제

———————————————————————————————————————————————————

1. **탐색 용어와 관련된 문제다. 빈 칸에 들어가는 단어를 맞춰보도록 한다.**
2. ① 순차 ② 순서 ③ 차례 ④ 0 ⑤ 크기-1
3. ① 이진 ② 두 개 ③ 이진 탐색 ④ 정렬
4. **탐색을 사용하고 있는 곳을 세 가지씩만 나열해 보자.**
5. 체인점 연결, 미아 찾기, 범인 찾기, 음주 단속, 숨은 그림 찾기, 직업 찾기
6. 구글 검색, P2P 검색, 바이러스 검색, 찾기 명령, 무선 랜 검색

|  |
| --- |
| 해답  정답으로 간주할 수 있는 것들이 너무 많아서 이번 장에서 설명한 것만 실었습니다. |

1. 탐색과 관련해서 나머지 자료 구조들과 비교해 보도록 하자.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 구분 | 스택 | 큐 | 리스트 | 트리 | 우선 순위 큐 | 그래프 | 해시 |
| 정렬 | YES | YES | NO | YES | YES | NO | NO |
| 방식 | 후입선출 | 선입선출 | 순차 탐색 | 이진 탐색 | 히프 구조 | 순차 탐색 | 해시 함수 |
| 탐색 | 1등 | 1등 | 4등 | 3등 | 1등 | 5등 | 2등 |

|  |
| --- |
| 해답  표에 대한 완전한 설명은 4장의 연계 학습에 있습니다. 설명이 너무 길어 기본적인 답안만 제공합니다. |

**———————————————————————————————————————————————————**

[2] 기본 실습 문제

———————————————————————————————————————————————————

1. **이진 탐색을 완전하게 이해했는지 검증해 보는 시간이다. 코드보다 일단 그림을 통해 진행 방식을 이해했는지 살펴보도록 한다.**
2. 574 182 128 145 157
3. 574 849 590 683 700
4. 574 849 885 920 962
5. 574 182 510 306
6. **저자는 여러분이 복잡한 코드까지 직접 구현하기를 바라지는 않는다. 대신 비어있는 곳의 코드를 채울 정도는 되어야 한다고 생각한다. 빈 칸에 들어가는 코드는 무엇인가?**
   1. ① if(array[i] >= value)

② if(array[i] == value)

* 1. ① left = 0, right = size-1;

② middle = (left+right)/2;

③ if(array[middle] < value)

**———————————————————————————————————————————————————**

[3] 응용 실습 문제

———————————————————————————————————————————————————

1. 본문에서 만들었던 함수와 똑같은 매개 변수를 사용하기 위해 래퍼 함수를 두었습니다. 실제 작업은 binarySearchReal 함수가 담당합니다. 재귀 함수로 꾸몄음에도 그다지 재미를 보지는 못했습니다.

int binarySearch(int\* array, int size, int value)

{

return binarySearchReal(array, 0, size-1, value);

}

int binarySearchReal(int\* array, int left, int right, int value)

{

if(left <= right)

{

int middle = (left+right)/2;

if(value == array[middle])

return middle;

if(value < array[middle]) right = middle-1;

else left = middle+1;

return binarySearchReal(array, left, right, value);

}

return -1;

}

조금이라도 간결히 만들려고 했지만, 별다른 방법은 없어 보입니다. 매개 변수를 포인터로 줄여서 처리해 봤습니다. 래퍼 함수에서 -1을 반환하기 위해 코드가 추가됐습니다.

int binarySearch(int\* array, int size, int value)

{

int\* p = binarySearchReal(array, array+size-1, value);

return p ? (int) (p-array) : -1;

}

int\* binarySearchReal(int\* left, int\* right, int value)

{

if(left <= right)

{

int\* middle = left+(right-left)/2; // 포인터 뺄셈을 이용해서 가운데 위치를 구합니다.

if(value == \*middle)

return middle;

if(value < \*middle) right = middle-1;

else left = middle+1;

return binarySearchReal(left, right, value);

}

return NULL;

}

1. 이번 문제는 linearSearch 함수가 반환한 값을 다시 linearSearch 함수에 재사용할 수 있는지 보는 문제였습니다. 자신이 가진 것만 제대로 사용해도 충분하다는 것을 알려주고 싶었습니다. 배열의 첫 번째와 마지막을 식별하는 것이 번거로운데, 쉽게 구현하기 위해 임시 변수를 사용했습니다.

int printMatchAll(int\* array, int size, int value)

{

int\* start = array,

\* end = array+size-1, pos, i;

for(i = 0; ; i++)

{

pos = linearSearch(array, size, value);

if(pos == -1)

break;

printf("[%2d ", (array+pos == start) ? -1 : array[pos-1]);

printf( "%2d ", array[pos ]);

printf("%2d] ", (array+pos == end ) ? -1 : array[pos+1]);

array += pos+1;

size -= pos+1;

}

printf("\n");

return i;

}

1. 정렬에서 모든 자료형에 대해 동작하는 정렬 함수를 만들어 봤습니다. 이번 문제는 앞에서 풀었던 문제를 다시 한번 확인하는 것에 지나지 않습니다. 이진 탐색을 처리해야 하므로, 요소 개수는 9개로 늘렸습니다. 반복문은 배열에 포함된 모든 요소를 검색해서 성공하는 것을 보여주고, 반복문을 벗어나서는 존재하지 않는 요소를 탐색해서 실패한다는 것을 보여줍니다.

#include <stdio.h>

#include <string.h>

#include <assert.h>

int generalSearch(void\* find, void\* array, int arraySize, int elementSize, int (\* cmp)(const void\*, const void\*));

int cmp1(const void\* p1, const void\* p2);

int cmp2(const void\* p1, const void\* p2);

int cmp3(const void\* p1, const void\* p2);

void main(void)

{

int array1[9] = { 10, 20, 30, 40, 50, 60, 70, 80, 90 };

char array2[9][32] = { "black", "cereal", "naver", "net", "page", "phone", "ranger", "twin", "web" };

char\* array3[9] = { "black", "cereal", "naver", "net", "page", "phone", "ranger", "twin", "web" };

int i, value = 55;

char\* google = "google";

for(i = 0; i < 9; i++)

{

assert(generalSearch(array1+i, array1, 9, sizeof(\*array1), cmp1) != -1);

assert(generalSearch(array2+i, array2, 9, sizeof(\*array2), cmp2) != -1);

assert(generalSearch(array3+i, array3, 9, sizeof(\*array3), cmp3) != -1);

}

if(generalSearch(&value, array1, 9, sizeof(\*array1), cmp1) != -1)

printf("결과: 성공\n");

else

printf("결과: 실패[%d]\n", value);

if(generalSearch(google, array2, 9, sizeof(\*array2), cmp2) != -1) // & 없음. cmp2 함수에서 \* 안 붙임.

printf("결과: 성공\n");

else

printf("결과: 실패[%s]\n", google);

if(generalSearch(&google, array3, 9, sizeof(\*array3), cmp3) != -1) // & 있음. cmp3 함수에서 \* 붙임.

printf("결과: 성공\n");

else

printf("결과: 실패[%s]\n", google);

}

int cmp1(const void\* p1, const void\* p2)

{

return \*(int\*) p1 - \*(int\*) p2;

}

int cmp2(const void\* p1, const void\* p2)

{

return strcmp(p1, p2);

}

int cmp3(const void\* p1, const void\* p2)

{

return strcmp(\*(char\*\*) p1, \*(char\*\*) p2);

}

int generalSearch(void\* find, void\* array, int arraySize, int elementSize, int (\* cmp)(const void\*, const void\*))

{

int i, left, right, middle, result;

left = 0, right = arraySize-1;

for(i = 0; left <= right; i++)

{

middle = (left+right)/2;

result = cmp((char\*) array+middle\*elementSize, find);

if(result == 0)

return middle;

if(result < 0) left = middle+1;

else right = middle-1;

}

return -1;

}