데이터구조 선택정렬

20223100 박신조

정렬이란?

물건을 크기순으로 오름차순(adcending order)이나 내림차순(descending order)으로 나열하는 것을 의미합니다.
ex) 책들을 제목 순, 저자 순으로 사람은 키, 이름으로 나열한 것

정렬은 컴퓨터 공학에서 가장 기본적이고 중요한 알고리즘입니다.
사람들이 물건을 구매할 때 가격 순으로 나열하는 것 또한 정렬 알고리즘을 사용합니다.

만약 국어사전이 가나다 순으로 정렬되어 있지 않다면 특정 단어를 찾을 때 오랜 시간이 걸립니다.
그렇지만 국어사전은 정렬이 되어있고 사람들이 단어를 찾을 때 해당 자음이나 모음이 있는 페이지에서 조금의 시간만 들이면 찾을 수 있습니다.
이처럼 정렬은 자료 탐색에 있어서 필수적입니다. 이는 컴퓨터에도 똑같이 적용됩니다.
어떤 자료를 잘 정렬된 자료들 속에서 찾는 것은 높은 효율을 보입니다. 그렇지만 정렬되어 있지 않은 자료들에서 찾는 것은 낮은 효율을 보입니다.

일반적으로 정렬시켜야 하는 대상을 레코드라고 부르고 다시 필드 단위로 나눠집니다.
ex) 레코드를 학생들이라 하고 이름, 학번, 주소, 전화번호는 필드가 됩니다.
이때 레코드와 레코드를 식별시켜 주는 필드를 키(key) 라고 합니다.
결국 정렬이란 레코드들을 키값의 순서로 재배열하는 것입니다.

지금까지 개발된 정렬 알고리즘은 매우 많습니다. 그러나 아직까지 모든 경우에 있어서 최상의 성능을 보여주는 알고리즘은 존재하지 않습니다.
그렇기 때문에 여러 개의 알고리즘 중 자신의 개발 환경에 맞는 최적의 알고리즘을 사용하는 것이 중요합니다.
정렬 알고리즘은 대개 2가지로 나눠집니다.

1. 단순하지만 비효율적인 알고리즘

ex) 삽입정렬, 선택 정렬, 버블정렬 등

2. 복잡하지만 효율적인 알고리즘

ex) 퀵정렬, 히프 정렬, 합병정렬, 기수정렬 등

자료의 개수가 적다면 1번 종류의 알고리즘을 사용하고 일정 개수를 넘어가면 그에 맞는 효율적인 알고리즘을 사용하면 됩니다.
그 중에서 제가 소개할 정렬은 선택 정렬 입니다.

선택정렬이란

숫자만 있는 1차원 배열이 있다고 가정한다면 배열 요소들의 최솟값을 찾아 0번째 인덱스에 저장한 후 0번째 인덱스를 제외하고
다시 최솟값을 찾아 1번째 인덱스에 저장하여 이 과정을 배열의 최댓값이 마지막 인덱스에 올 때까지 반복하는 정렬 기법입니다.

1. 배열 두 개를 사용하는 선택정렬

왼쪽 리스트	오른쪽 리스트	설명
( )	(5,3,8,1,2,7)	초기상태
(1)	(5,3,8,2,7)	1선택
(1,2)	(5,3,8,7)	2선택
(1,2,3)	(5,8,7)	3선택
(1,2,3,5)	(8,7)	5선택
(1,2,3,5,7)	(8)	7선택
(1,2,3,5,7,8)	()	8선택

[그림 12-8] 선택 정렬의 원리

위 그림처럼 배열 두 개를 사용하는 방법은 숫자만 들어있는 1차원 배열을 준비하고 크기가 동일한 빈 배열을 준비합니다.
이때 왼쪽 배열을 빈 배열로 둡니다. 오른쪽 배열에서 가장 작은 최솟값을 구하여 왼쪽 첫 번째 배열로 정합니다.
이후 이 과정을 계속 반복하여 오른쪽 배열의 요소가 없을 때 까지 진행하면 정렬이 완성됩니다.

위처럼 배열 2개를 사용하면 숫자가 저장된 배열과 크기가 같은 빈 배열이 1개 더 필요합니다. -> 메모리를 많이 차지 함
그렇기에 메모리를 절약하기 위해 배열 1개만 사용하는 선택 정렬을 설명하겠습니다.

2. 배열 1개만 사용하는 선택정렬

[그림 12-10] 선택 정렬의 과정

위 그림처럼 배열 한 개를 사용하는 방법은 숫자로 이루어진 1차원 배열 1개를 준비합니다.
이후 배열에서 최솟값을 구합니다. 이 최솟값을 첫 번째 요소와 교환합니다.
다음은 첫 번째 요소를 제외한 배열에서 최솟값을 구합니다. 이후 두 번째 요소와 교환합니다.
이렇게 해서 마지막 요소까지 저장되고 나면 전체 요소들이 정렬됩니다.

선택 정렬의 알고리즘

알고리즘 12.1 선택 정렬 알고리즘

```
selection_sort(A, n):

for i←0 to n-2 do
    least ← A[i], A[i+1],..., A[n-1] 중에서 가장 작은 값의 인덱스;
    A[i]와 A[least]의 교환;
    i++;
```

위 배열 1개를 사용하는 선택정렬을 코드로 구현하면 이런 식의 알고리즘이 됩니다.

c언어를 사용하여 선택정렬 구현

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_SIZE 10
#define SWAP(x, y, t) ( (t)=(x),(x)=(y),(y)=(t))

int list[MAX_SIZE];
int n;
void selection_sort(int list[] , int n){
    int i , j , least, temp;
    for (i = 0; i< n - 1; i ++){
        least = i;
        for(j = i+1; j<n; j++)
            if(list[j]<list[least]) least = j;
        SWAP(list[i], list[least], temp);
    }
}

int main(void)
{
    int i;
    n = MAX_SIZE;
    srand(time(NULL));
    for (i = 0; i<n; i++)
        list[i] = rand() % 100;

    selection_sort(list, n);
    for (i = 0; i<n; i++)
        printf("%d ", list[i]);
    printf("\n");
    return 0;
}
```

프로그램 입력

프로그램 출력

```
15 17 33 37 58 73 92 92 94 97

[Execution complete with exit code 0]
```

위에서 말한 선택정렬 알고리즘을 활용하여 c언어로 선택정렬을 구현한 코드입니다.
seslection_sort 함수는 이중 for문을 활용하여 배열의 최솟값을 찾고 0번째 인덱스부터 n-2번째 인덱스까지 값을 저장해줍니다.
이때 배열은 전역변수로 선언해주어 seslection_sort 함수와 main 함수에서 같이 사용할 수 있도록 했습니다.
그리고 SWAP이라는 매크로를 설정해 찾은 최솟값과 i 번째 배열을 교환했습니다.
main 함수는 배열의 요소를 0부터 99까지의 난수로 설정하여 값이 랜덤하게 배열에 저장되도록 하였습니다.
이후 seslection_sort함수를 선언하여 배열을 정렬했습니다.

선택정렬의 시간복잡도
외부루프는 n-1번 실행되고
내부루프는 0에서 n-2까지 변하는 i에 대하여 (n-1)-i번 반복됩니다.

$$(n-1)+(n-2)+\cdots+1=n(n-1)/2=O(n2)$$

선택정렬의 시간복잡도는 O(n2) 이다.

선택 정렬의 전체 이동 횟수는 외부 루프 n-1에 SWAP에서 3번의 교환이 일어남으로 총 3(n-1)입니다.
자료가 정렬된 경우에는 불필요하게 자신 자신과의 이동을 하게 됩니다.

따라서 이 문제를 약간 개선하려면 다음과 같은 if 문을 추가하면 됩니다.

```
if( i != least )
    SWAP(list[i], list[least], temp);
```

아래 코드는 기존의 코드에서 배열의 값을 랜덤이 아니라 선언과 동시에 초기화를 주었습니다.
그리고 이동이 총 몇 번 일어나는지 체크하기 위해 count 라는 변수로 for문이 한번 반복될 때마다 카운드를 해줬습니다.
아래 결과에서 정확한 전체 이동 횟수를 체크 하려면 SWAP의 이동까지 계산하여 count값에 3을 곱해줘야 합니다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_SIZE 10
#define SWAP(x, y, t) ( (t)=(x),(x)=(y),(y)=(t))
int list[MAX_SIZE] = {10,2,4,3,5,2,4,7,11,1};
int n, count;
void selection_sort(int list[] , int n){
    int i , j , least, temp;
    for (i = 0; i< n - 1; i ++){
        least = i;
        for(j = i+1; j<n; j++)
            if(list[j]<list[least]) least = j;
        SWAP(list[i], list[least], temp);
        count ++;
    }
}
int main(void)
{
    int i;
    count = 0;
    n = MAX_SIZE;
    srand(time(NULL));
    /*for (i = 0; i<n; i++)
        list[i] = rand() % 10;*/
    selection_sort(list, n);
    for (i = 0; i<n; i++)
        printf("%d ", list[i]);
    printf("\n%d",count);
    return 0;
```

프로그램 입력

프로그램 출력

```
1 2 2 3 4 4 5 7 10 11
9
[Execution complete with exit code (
```

아래의 코드는 if문을 추가하여 같은 값이 있을 때 몇 번 반복되는지 카운트를 해줬습니다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_SIZE 10
#define SWAP(x, y, t) ( (t)=(x),(x)=(y),(y)=(t))
int list[MAX_SIZE] = {10,2,4,3,5,2,4,7,11,1};
int n, count;
void selection_sort(int list[] , int n){
    int i , j , least, temp;
    for (i = 0; i< n - 1; i ++){
        least = i;
        for(j = i+1; j<n; j++)
            if(list[j]<list[least]) least = j;
        if( i != least ){
            SWAP(list[i], list[least], temp);
        count ++;}
    }
}
int main(void)
{
    int i;
    count = 0;
    n = MAX_SIZE;
    srand(time(NULL));
    /*for (i = 0; i<n; i++)
        list[i] = rand() % 10;*/
    selection_sort(list, n);
    for (i = 0; i<n; i++)
        printf("%d ", list[i]);
    printf("\n%d",count);
    return 0;
```

프로그램 입력

프로그램 출력

```
1 2 2 3 4 4 5 7 10 11
5
[Execution complete with exit code 0
```

두 코드의 결과를 비교 결과

최소값이 자기 자신이면 자료이동을 하지 않는다는 것을 확인하였습니다.
이때 자료 이동을 하지 않는다는 것은 코드 작동시간이 단축될 수 있음을 의미하고
조금 더 효율적인 코드라고 생각할 수 있을 것 같습니다.

선택정렬의 장단점

장점
자료 이동 횟수가 미리 정해진다는 장점이 있습니다.

단점
안정성을 만족하지 않는다는 것입니다.
값이 같은 레코드가 있는 경우에 상대적인 위치가 변경될 수 있습니다.

(Dave, A) (Alice, B) (Ken, A) (Eric, B) (Carol, A)	왼쪽 이름 순으로 정렬 ->	(Alice, B) (Carol, A) (Dave, A) (Eric, B) (Ken, A)	오른쪽 알파벳 순으로 정렬 ->	(Carol, A) (Dave, A) (Ken, A) (Eric, B) (Alice, B)

위에 있는 자료들을 이름 순으로 정렬한 뒤 오른쪽 알파벳 순으로 정렬하면
알파벳으로 정렬 전에는 Alice가 Eric가 앞에 있었지만 정렬 후 Alice가 뒤로 이동했습니다.
이는 정렬 이전의 순서는 보장되지 않는다는 것을 의미합니다. 그렇기 때문에 선택 정렬을 불안정 정렬에 속합니다.
안정 정렬로 정렬을 진행한다면 아래와 같은 결과가 나옵니다.

(Alice, B) (Carol, A) (Dave, A) (Eric, B) (Ken, A)	오른쪽 알파벳 순으로 정렬 ->	(Carol, A) (Dave, A) (Ken, A) (Alice, B) (Eric, B)

자료 출처 : https://wisdom-and-record.tistory.com/57