자료구조 Lab#11 Report

2017038102
최용석

```
/*
* hw5-sorting.c
*
*  Created on: May 22, 2019
*
*  Homework 5: Sorting & Hashing
*  Department of Computer Science at Chungbuk National University
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h> /*배열의 값을 랜덤으로 초기화하기 위한 헤더파일*/

#define MAX_ARRAY_SIZE                  13      /* prime number */
#define MAX_HASH_TABLE_SIZE     MAX_ARRAY_SIZE

/* 필요에 따라 함수 추가 가능 */
int initialize(int **a);
int freeArray(int *a);
void printArray(int *a);

int selectionSort(int *a);
int insertionSort(int *a);
int bubbleSort(int *a);
int shellSort(int *a);
/* recursive function으로 구현 */
int quickSort(int *a, int n);


/* hash code generator, key % MAX_HASH_TABLE_SIZE */
int hashCode(int key);

/* array a에대 한 hash table을 만든다. */
int hashing(int *a, int **ht);

/* hash table에서 key를 찾아 hash table의 index return */
int search(int *ht, int key);


int main()
{
    char command;
    int *array = NULL;
    int *hashtable = NULL;
    int key = -1;
    int index = -1;

    srand(time(NULL));

    do{
        printf("----------------------------------------------------------------\n");
        printf("                        Sorting & Hashing                       \n");
        printf("----------------------------------------------------------------\n");
        printf("===========[2017038102]==================[최용석]==============\n");
        printf(" Initialize        = z          Quit              = q\n");
        printf(" Selection Sort    = s          Insertion Sort    = i\n");
        printf(" Bubble Sort       = b          Shell Sort        = l\n");
        printf(" Quick Sort        = k          Print Array       = p\n");
        printf(" Hashing           = h          Search(for Hash) = e\n");
        printf("----------------------------------------------------------------\n");

        printf("Command = ");
        scanf(" %c", &command);

        switch(command)
                {
        case 'z': case 'Z':
            initialize(&array);
            break;
        case 'q': case 'Q':
            freeArray(array);
            break;
        case 's': case 'S':
            selectionSort(array);
            break;
         case 'i': case 'I':
            insertionSort(array);
            break;
        case 'b': case 'B':
            bubbleSort(array);
            break;
        case 'l': case 'L':
            shellSort(array);
            break;
```

```
        case 'k': case 'K':
         printf("Quick Sort: \n");
            printf("--------------------------------------------------------------------\n");
            printArray(array);  /*정렬 전 배열 출력*/
            quickSort(array, MAX_ARRAY_SIZE);
            printf("--------------------------------------------------------------------\n");
            printArray(array);  /*정렬 이후 배열출력.*/

            break;

        case 'h': case 'H':
            printf("Hashing: \n");
            printf("--------------------------------------------------------------------\n");
            printArray(array);
            hashing(array, &hashtable);
            printArray(hashtable);
            break;

        case 'e': case 'E':
            printf("Your Key = ");
            scanf("%d", &key);
            printArray(hashtable);
            index = search(hashtable, key);
            printf("key = %d, index = %d,  hashtable[%d] = %d\n", key, index, index, hashtable[index]);
            break;

        case 'p': case 'P':
            printArray(array);
            break;
        default:
            printf("\n         >>>>>    Concentration!!    <<<<<       \n");
            break;
                }

    }while(command != 'q' && command != 'Q');

         return 1;
}

int initialize(int** a)
{/*main 함수 내에 선언되어 있는  array포인터 변수를 직접 변경하기 위한 2중포인터 매개변수.*/
         int i;
    int *temp = NULL;

    /* array가 NULL인 경우 메모리 할당 */
    if(*a == NULL)
        {
        temp = (int*)malloc(sizeof(int) * MAX_ARRAY_SIZE);
        *a = temp;  /* 할당된 메모리의 주소를 복사 --> main에서 배열을 control 할수 있도록 함*/
    }
         else /*기존에 할당된 배열이 있으면 단순히 주소를 복사.*/
        temp = *a;

        /* 랜덤값을 배열의 값으로 저장 */
    for(i = 0; i < MAX_ARRAY_SIZE; i++)
        temp[i] = rand() % MAX_ARRAY_SIZE;

    return 0;
}

int freeArray(int *a)
{/*단순한 동적할당 배열이므로 반복문을 돌 필요없이 바로 할당해제.*/
    if(a != NULL)
        free(a);
         return 0;
}

void printArray(int *a)
{
         int i;
    if (a == NULL)/*예외처리 부분.*/
        {
        printf("nothing to print.\n");
        return;
    }
    /*배열의 index를 print*/
    for(i = 0; i < MAX_ARRAY_SIZE; i++)
            printf("a[%02d] ", i);
        printf("\n");
    /*실제 배열의 값을 print*/
    for(i = 0; i < MAX_ARRAY_SIZE; i++)
            printf("%5d ", a[i]);
        printf("\n");
}
```

```
int selectionSort(int *a)
{
    int min;
    int minindex;
    int i, j;

    printf("Selection Sort: \n");
    printf("-----------------------------------------------------------------\n");

    printArray(a);
         /*정렬 전 배열출력(다음 정렬함수부터 설명 생략).*/
    for (i = 0; i < MAX_ARRAY_SIZE; i++)
    {
        minindex = i;/*배열의 첫번째 원소를 최소값의 위치로 가정하고 하나씩 늘려가며 정렬.*/
         min = a[i]; /*최소값을 임시로 저장해놓음.*/
            for(j = i+1; j < MAX_ARRAY_SIZE; j++)
            {
                if (min > a[j])
                {/*반복문을 돌면서 더 작은값을 발견하면 index 와 value를 최신화.*/
                    min = a[j];
                    minindex = j;
                }
            }
        a[minindex] = a[i]; /*i번째 원소를 최소값이 있는 위치에 덮어씀.*/
        a[i] = min; /*최소값을 i번째 위치에 정렬.*/
    }

    printf("-----------------------------------------------------------------\n");
    printArray(a); /*정렬된 배열을 출력하고 return(다음 정렬함수부터 설명 생략).*/
    return 0;
}

int insertionSort(int *a)
{
    int i, j, t;

    printf("Insertion Sort: \n");
    printf("-----------------------------------------------------------------\n");

    printArray(a);

    for(i = 1; i < MAX_ARRAY_SIZE; i++)
    {/*삽입정렬할 영역을 하나씩 늘려가며 진행, index와 값을 임시로 저장.*/
        t = a[i];
        j = i;
        while (a[j-1] > t && j > 0)
        {/*while 문을 더 작은값을 찾았거나, index가 0번까지 갈 때까지 반복*/
            a[j] = a[j-1];
            j--;
        }
        a[j] = t; /*반복문이 빠져나왔을때 위의 for문에서 새로 들어온 i의 값이 0번쨰 이거나,
                    자신보다 작은 값 바로 뒤에 위치할 것임.(J의 위치에 따라서)*/
    }

    printf("-----------------------------------------------------------------\n");
    printArray(a);

    return 0;
}

int bubbleSort(int *a)
{
    int i, j, t;

    printf("Bubble Sort: \n");
    printf("-----------------------------------------------------------------\n");

    printArray(a);

    for(i = 0; i < MAX_ARRAY_SIZE; i++)
    {
        for (j = 1; j < MAX_ARRAY_SIZE; j++)
        {/*기존에 J가 0부터 시작했을 때 if에서 J-1이 참조가 되어 배열의 끝값이
                        쓰레기 값으로 정렬이 되는 현상을 for문에서 1번부터 시작하는것으로 수정.*/
            if (a[j-1] > a[j])
            {/*한칸씩 진행해 나가면서 큰값을 발견했으면 swap하며 최대값을 끝으로 보냄.*/
                t = a[j-1];
                a[j-1] = a[j];
                a[j] = t;
            }
        }
    }
         /*해당 반복문이 모두 완료된다면 정렬이 끝난것임.*/
```

```
    printf("----------------------------------------------------------------\n");
    printArray(a);

    return 0;
}

int shellSort(int *a)
{
    int i, j, k, h, v;

    printf("Shell Sort: \n");
    printf("----------------------------------------------------------------\n");

    printArray(a);

    for (h = MAX_ARRAY_SIZE/2; h > 0; h /= 2)
    {/*h를 최초 배열의 절반의 size 로 시작해 반으로 줄여나가며 h거리만큼 있는 원소들을 정렬함.*/
        for (i = 0; i < h; i++)
        {/*해당 부분은 h거리를 기준으로 작은index에 위치한  원소*/
            for(j = i + h; j < MAX_ARRAY_SIZE; j += h)
            { /*해당 부분은 h거리를 기준으로 큰index에 위치한  원소*/
                v = a[j];
                k = j;
                    while (k > h-1 && a[k-h] > v)
                    {
                        a[k] = a[k-h];
                        k -= h;
                    }/*해당 h거리만큼 비교해서 바꾸어 나가면서 최종적으로h==1 이 되었을 때 삽입정렬의
                                            형태로 셸 정렬에서 의도한 대충정렬 후 삽입정렬의 개념을 적용할 수 있다.*/
                a[k] = v;
            }
        }
    }
    printf("----------------------------------------------------------------\n");
    printArray(a);

    return 0;
}

int quickSort(int *a, int n)
{
    int v, t;
    int i, j;

    if (n > 1)
    {
        v = a[n-1];/*해당 함수에서는  pivot 값을 가장 마지막 원소로 지정.*/
        i = -1;
        j = n - 1;/*마지막 index번호를 저장*/

        while(1)
         {
         while(a[++i] < v);/*마지막원소보다 큰값이 나올때까지 i를 증가*/
         while(a[--j] > v);/*마지막 원소보다 작은값이 나올때까지 J를 감소*/

         if (i >= j) break; /*i가 j를 넘어가버린다면 반복문을 break */
                  t = a[i];
         a[i] = a[j];
         a[j] = t; /*위의  if문이 실행되지 않는다면 pivot값을 기준으로 큰값을 앞으로 보냄*/
         }
          /*해당 명령어가 실행된다면 if문에서 i가 j를 넘어가 버린 것이므로 i를 기존의pivot위치와
                    바꿈으로써 i번째 index기준으로 앞쪽에는 큰값이, 뒤쪽에는 작은값이 정렬되게 됨.*/
         t = a[i];
         a[i] = a[n-1];
        a[n-1] = t;

        quickSort(a, i);/*i를 기준으로 앞쪽을 다시 재귀형식으로 퀵정렬*/
        quickSort(a+i+1, n-i-1);/*i의 다음 인덱스와, a~i까지의 인덱스의크기만큼 빠진 size를 재귀적으로 호출*/
    }
        return 0;
}

int hashCode(int key)
{/*0~MAX_HASH_TABLE_SIZE-1 의 값을 retrun*/
  return key % MAX_HASH_TABLE_SIZE;
}

int hashing(int *a, int **ht)
{
         int i;
    int *hashtable = NULL;

        /* hash table이 NULL인 경우 메모리 할당 */
    if(*ht == NULL)
```

```
        {
        hashtable = (int*)malloc(sizeof(int) * MAX_ARRAY_SIZE);
        *ht = hashtable;  /* 할당된 메모리의 주소를 복사 --> main에서 배열을 control 할수 있도록 함*/
    }
         else
         {
        hashtable = *ht;          /* hash table이 NULL이 아닌경우, table 재활용, reset to -1 */
    }

    for(i = 0; i < MAX_HASH_TABLE_SIZE; i++)
         hashtable[i] = -1;

        /*
        for(int i = 0; i < MAX_HASH_TABLE_SIZE; i++)
                printf("hashtable[%d] = %d\n", i, hashtable[i]);
        */
         /*사용할 변수들의 초기화*/
    int key = -1;
    int hashcode = -1;
    int index = -1;

    for (i = 0; i < MAX_ARRAY_SIZE; i++)
    {
        key = a[i];
        hashcode = hashCode(key);
                /*
                printf("key = %d, hashcode = %d, hashtable[%d]=%d\n", key, hashcode, hashcode,
hashtable[hashcode]);
                */
        if (hashtable[hashcode] == -1)/*해당 테이블의 slot은 하나임을 알 수 있다.*/
            {/*해당 테이블이 비어있다면.*/
                hashtable[hashcode] = key; /*정상적으로 할당함.*/
            }
                        else/*해당 해시테이블은 개방 주소법을 사용함. cahining으로 overflow를 처리하지 않음.*/
                        {
                index = hashcode;/*찾으려는 key의 code값을 index로*/

                while(hashtable[index] != -1)
                    {/*해당 함수에서는 특정 primenumber(13)를 사용해서 index 값을 더한 후 제산하는 double hashing 을
이용해
                                                overflow발생시 처리함.*/
                        index = (++index) % MAX_HASH_TABLE_SIZE;
                                /*
                                printf("index = %d\n", index);
                                */
                    }
                hashtable[index] = key;
            }     /*비어있는 공간을 찾았을 경우 대입하고 return*/
    }
    return 0;
}

int search(int *ht, int key)
{/*인자로 받은 key에 해당하는 값을 담을 변수 */
    int index = hashCode(key);
         /*실제 해당index의 맞는 키가있다면 정상적으로 반환 */
    if(ht[index] == key)
        return index;

    while(ht[++index] != key)
    {/*위에서의 조건문이 실행이 되지않는다면 충돌 및 오버플로우가 발생해 다른 index 에 삽입한 것이므로
          Double Hashing을 통해 key가 같을때까지 탐색*/
        index = index % MAX_HASH_TABLE_SIZE;
    }
         return index;
}/*해당 return이 실행된다면 값을 찾은것임. 해당 함수에서 값을 찾지 못했을 경우의 예외처리가 없음*/
```

코드의 양이 많은 관계로 줄간격 및 글자 크기를 줄였습니다.
이번 과제를 통해 각종 정렬의 동작원리, 해싱에서의 overflow가 발생했을 경우의 처리 등에 대해서 개략적으로 알 수 있어서 매우 뜻깊은 과제였던 것 같습니다.

Github repository Address : https://github.com/dydtjr1515/Data-Structure-HW.git