Data Structure HW2

20231515
컴퓨터공학과
김다은

<목차>

문제 1.

이 문제를 효율적으로 풀기 위해 int result 를 이진수 단위에서 활용하였다. result 를 이진수로 해석하여 해당 알파벳의 자리에 1 이 있는 경우 해당 알파벳을 출력한다. 예를 들어 입력된 전체 집합의 원소 수가 4 이고 result 를 이진수로 나타냈을 때, 0101 이라면, {a c}를 출력한다. 이 과정에서 shift 연산자와 %2 를 사용한다.

프로그램 실행 단계:

1. main 함수에서 전체 집합 S 의 원소 수(s)를 입력받는다.
2. PowerSet 의 첫번째 호출(flag = 1)에서 부분집합의 원소 수(num)을 0~s 까지 지정하여 PowerSet 함수를 호출한다.
3. PowerSet 의 이후 호출(flag = 0)에서
   A. 현재 계산중인 부분집합의 원소 수(cur)와 목표 원소 수(num)가 일치하면 printSet 함수를 호출한다.
   B. 일치하지 않다면 원소를 가능한 위치에 추가한 후 PowerSet 함수를 호출한다.
4. printSet 함수에서 부분집합의 원소를 이진수로 나타내는 result 를 argument 로 받아와 해석한 뒤 알파벳으로 출력한다.

```c
#include <stdio.h>

void printSet(int s, int num, int result) {
    printf("{");
    for(int i=0; i<s; i++) {
        if((result>>i)%2) { // 해당 원소가 존재한다면
            printf("%c", 'a'+i);
            if(--num>0) printf(" ");
        }
    }
    printf("} ");
}

// s: 집합의 원소 수
// num: 계산하고 싶은 부분집합의 원소 수
// cur: 현재 부분집합의 원소 수
// idx: 현재 계산중인 원소의 인덱스(a 가 1, b 가 2 ...)
// result: 부분집합
// flag: PowerSet 함수의 첫 호출을 구분하기 위한 수
void PowerSet(int s, int num, int cur, int idx, int result, int flag) {
    if(flag) { // PowerSet 의 첫 호출에서
        for(int i=0; i<=s; i++) {
```

```
            PowerSet(s, i, 0, 0, result, 0);
        }
    }
    else {
        if(cur==num) {
            printSet(s, num, result);
            return;
        }
        for(int i=idx; i<s; i++) {
            result += 1 << i; // c[i] = 1;
            PowerSet(s, num, cur+1, i+1, result, 0);
            result -= 1 << i; // c[i] = 0;
        }
    }

}

int main() {
    int s; // 집합 S 의 원소 수를 저장
    scanf("%d", &s);
    PowerSet(s, 0, 0, 0, 0, 1);
}
```

실제 입력 결과:

```
cse20231515@cspro2:~/cse3080/HW2$ gcc p1.c -o p1
cse20231515@cspro2:~/cse3080/HW2$ ./p1
3
{} {a} {b} {c} {a b} {a c} {b c} {a b c}
cse20231515@cspro2:~/cse3080/HW2$
```

```
cse20231515@cspro2:~/cse3080/HW2$ ./p1
5
{} {a} {b} {c} {d} {e} {a b} {a c} {a d} {a e} {b c} {b d} {b e} {c d} {c e} {d e} {a b c} {a b d} {a b e} {a c d} {a c e} {a d e} {b c d} {b c e} {b d e} {c d e} {a b c d} {a b c e} {a b d e} {a c d e} {b c d e} {a b c d e}
```

문제 2.

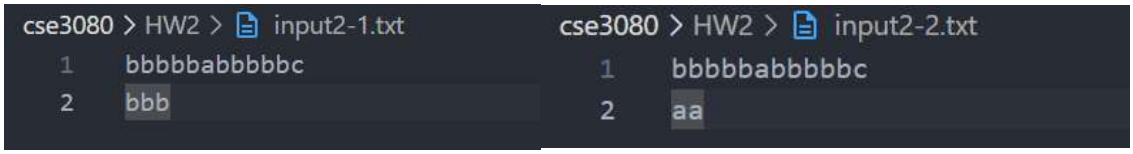
프로그램 실행 단계:

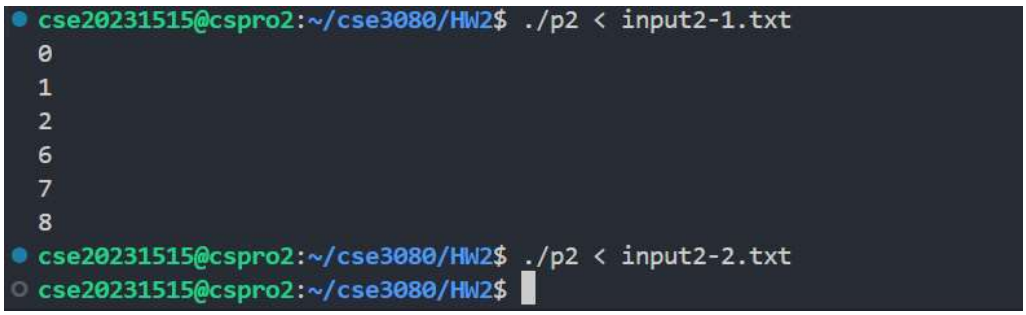
1. main 함수에서 string 과 pattern 을 입력 받는다.
2. failure table 를 만든다.
3. KMP algorithm 을 이용해 str 안에 pat 가 존재하는지 확인한다.
4. pat 가 존재하는 모든 위치의 index 를 출력한다.

```c
#include <stdio.h>
#include <string.h> // strlen 함수를 사용하기 위함

void pmatch_all(char str[], char pat[]) {
    int slen = strlen(str);
    int plen = strlen(pat);
    // failure table 만들기
    int failure[40] = {0,}; // 0 으로 초기화
    int j=0;
    for(int i=1; i<plen; i++) {
        while(j>0 && pat[i]!=pat[j]) {
            j = failure[j-1];
        }
        if(pat[i]==pat[j]) {
            failure[i] = ++j;
        }
    }

    // KMP 구현
    j = 0;
    for(int i=0; i<slen; i++) {
        while(j>0 && str[i]!=pat[j]) {
            j = failure[j-1];
        }
        if(str[i]==pat[j]) {
            if(j == plen-1) {
                printf("%d\n", i-plen+1);
                j = failure[j];
            }
            else {
                j++;
            }
        }
    }
```

```
}

int main() {
    char str[31]; // string 을 입력받아 저장
    char pat[31]; // pattern 을 입력받아 저장
    scanf("%s %s", str, pat);
    pmatch_all(str, pat);
}
```

실제 입력 결과





문제 3.

이 문제의 핵심은 O(n) 시간복잡도를 가져야 한다는 점이다. 배열을 정렬한 후 찾는다고 해도 O(nlogn)의 시간복잡도가 소요된다. 따라서 사용자의 입력을 저장하는 배열 array 이외에 추가적인 배열 table 를 생성하여 사용하였다.

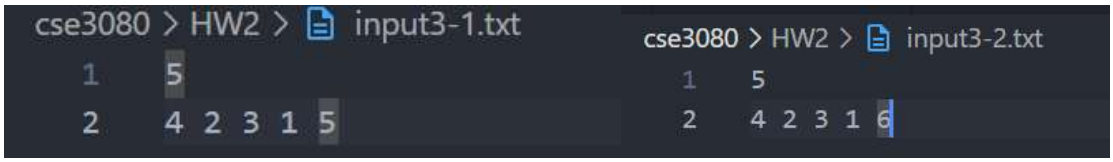
프로그램의 진행 단계는 다음과 같다.

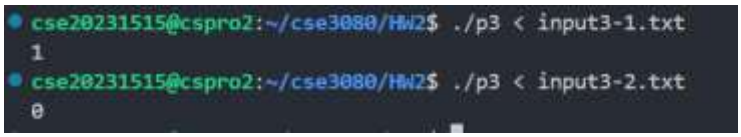
1. main 함수에서 입력 받은 값을 array 에 저장한다.
2. check 함수에서
   A. 입력 값 중에 중복이 존재하는지 빠르게 확인하기 위해 table 배열을 생성한다.
   B. array 의 값을 반복문을 통해 index 1 부터 순회하면서, array 의 최솟값과 최댓값을 각각 minidx 와 maxidx 에 저장한다.
   C. 이 과정에서 중복되는 수가 발견된 경우 table 에 할당된 메모리를 해제한 뒤, 0 을 리턴한다.
   D. maxidx-minidx == n-1 인 경우 maxidx 와 minidx 사이에 모든 수가 연속적으로 존재한다는 의미이므로 table 에 할당된 메모리를 해제한 뒤, 1 을 리턴한다.
   E. maxidx-minidx != n-1 인 경우 maxidx 와 minidx 사이에 존재하지 않는 수가 있다는 의미이므로 table 에 할당된 메모리를 해제한 뒤, 0 을 리턴한다.
3. check 함수의 리턴값을 출력한다.
4. array 에 할당된 메모리를 해제한다.

```c
#include <stdio.h>
#include <stdlib.h> // 동적 할당을 위해 include

// 수가 연속적인지 확인
int check(int n, int *array) {
    // table: 중복되는 값을 판단하기 위한 배열
    int *table = (int*)malloc(100*sizeof(int));
    int minidx = array[0];
    int maxidx = array[0];
    table[array[0]]++;
    for(int i=1; i<n; i++) {
        if(table[array[i]]) {
            free(table);
            return 0; // 중복되는 수 존재
        }
        if(array[i]<minidx) minidx = array[i];
        else if(array[i]>maxidx) maxidx = array[i];
    }
```

```
    free(table);
    if(maxidx-minidx==n-1) return 1; // 연속적
    return 0; // minidx 와 maxidx 사이에 공백 수 존재
}

int main() {
    int n, *array;
    scanf("%d", &n); // 원소의 개수를 입력받아 n 에 저장
    array = (int*)malloc(n*sizeof(int));

    for(int i=0; i<n; i++) {
        scanf("%d", &array[i]);
    }
    printf("%d\n", check(n, array));
    free(array);
}
```

실제 입력 결과





문제 4.

lastname 과 firstname 정보를 효과적으로 저장하기 위해 다음과 같은 array structure 를 사용한다. 또한 array 를 element 로 하는 linked list 를 만들고 각 삽입 과정에서 동시에 sorting 을 진행하도록 코딩하였다.

```
typedef struct array {
    char* lastname;
    char* firstname;
    struct array* link;
} array;
```

프로그램의 진행 단계는 다음과 같다.

1. File input 을 통해 "students.txt"를 읽기모드로 연다.
2. 이후 n 개의 줄에서 학생의 lastname 과 firstname 를 입력받는다.
3. sort_and_insert 함수
   A. createarray 함수를 호출하여 lastname 과 firstname 를 argument 로 넘긴다. createarray 함수에서는 해당 학생명을 갖는 array 를 생성하고 메모리 동적 할당을 진행한 후 생성한 array 를 반환한다.
   B. compare 함수를 호출하여 tmp 가 가리키는 array 와 새로 만든 newarray 의 학생명을 비교한다.
   C. compare 함수를 통해 얻은 이름순을 정렬된 linked list 의 위치에 newarray 를 삽입한다. 이때 linked list 의 맨 앞에 삽입하는 경우, 중간에 삽입하는 경우, 맨 마지막에 삽입하는 경우를 고려하였다.
4. printarray 함수에서 정렬된 linked list 의 학생명을 순서대로 출력한다.
5. freearray 함수에서 linked list 에 동적 할당된 메모리를 모두 해제한다.
6. fclose 를 통해 읽어온 txt 파일을 닫는다.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct array {
    char* lastname;
    char* firstname;
    struct array* link;
} array;

// 문자열 a 와 b 의 정렬 우선순위를 리턴한다.
```

```
// lastname 를 먼저 비교한 후 firstname 를 비교한다.
// a 가 b 보다 우선순위가 높은 경우 return 1
// a 와 b 의 우선순위가 같은 경우 return 0
// b 가 a 보다 우선순위가 높은 경우 return -1
int compare(array *a, array *b) {
    // lastname 비교
    char* alast = a->lastname;
    char* blast = b->lastname;
    for(int i=0; ; i++) {
        if(alast[i]=='\0' && blast[i]=='\0') break;
        else if(alast[i]=='\0') return 1; // a 가 더 우선순위
        else if(blast[i]=='\0') return -1; // b 가 더 우선순위

        if(alast[i]<blast[i]) return 1;
        else if(alast[i]>blast[i]) return -1;
    }

    // firstname 비교(lastname 이 동일한 경우)
    char* afirst = a->firstname;
    char* bfirst = b->firstname;
    for(int i=0; ; i++) {
        if(afirst[i]=='\0' && bfirst[i]=='\0') break;
        else if(afirst[i]=='\0') return 1; // a 가 더 우선순위
        else return -1; // b 가 더 우선순위

        if(afirst[i]>bfirst[i]) return 1;
        else if(afirst[i]<bfirst[i]) return -1;
    }
    return 0; // 완전 동일
}

// 문자열 s 의 길이를 리턴한다.
int length(char* s) {
    int i;
    for(i=0; s[i]!='\0'; i++) {}
    return i;
}

// b 의 문자열을 a 에 복사한다.
void copy(char* a, char* b) {
    for(int i=0; i<length(b)+1; i++) {
```

```
        a[i] = b[i];
    }
}

// 학생의 이름을 argument 로 받아 array 를 생성하고 메모리 동적 할당을 진행한다.
array* createarray(char *lastname, char *firstname) {
    array *newarray = (array*)malloc(sizeof(array));
    char* newlast = (char*)malloc(sizeof(char)*(length(lastname)+1));
    char* newfirst = (char*)malloc(sizeof(char)*(length(firstname)+1));
    copy(newlast, lastname);
    copy(newfirst, firstname);
    newarray->lastname = newlast;
    newarray->firstname = newfirst;
    newarray->link = NULL;
    return newarray;
}

// 학생명을 입력받아 array 를 생성하고 이름순으로 정렬된 위치에 linked list 에 추가한다.
void sort_and_insert(array** base, char *lastname, char *firstname) {
    array* newarray = createarray(lastname, firstname);

    if(*base == NULL) { // 아예 0
        *base = newarray;
        return;
    }
    array* tmp = *base;
    array* pre = NULL;
    for(; tmp; pre = tmp, tmp=tmp->link) {
        if(compare(newarray, tmp)>=0) {
            if(pre==NULL) { // 맨 앞 삽입
                *base = newarray;
                newarray->link = tmp;
            }
            else { // 중간에 삽입
                pre->link = newarray;
                newarray->link = tmp;
            }

            return;
        }
    }
```

```
    // 맨 마지막 삽입
    pre->link = newarray;
}

void printarray(array* base) {
    array* tmp = base;
    for(; tmp; tmp=tmp->link) {
        printf("%s %s\n", tmp->lastname, tmp->firstname);
    }
}

void freearray(array* base) {
    array* tmp = NULL;
    while(!base) {
        tmp = base->link;
        free(base->firstname);
        free(base->lastname);
        free(base);
        base = tmp;
    }
}

int main() {
    int n;
    array* base = NULL;
    char c;
    char *str;
    int strlen = 0;
    char *lastname;
    char *firstname;
    FILE *fp = fopen("student.txt", "r");

    char test[1000];
    fscanf(fp, "%d", &n);
    for(int i=0; i<n; i++) {
        str = (char*)malloc(sizeof(char));
        strlen = 0;

        fscanf(fp, " %c", &c);
        while(c!=EOF) {
            // printf("%c", c);
```

```
            if(c==' ') {
                break;
            }
            else {
                //printf("%c", c);
                str[strlen] = c;
                str = (char*)realloc(str, sizeof(char)*(++strlen));
                fscanf(fp, "%c", &c);
            }
        }
        str[strlen] = '\0';
        lastname = (char*)malloc(sizeof(char)*(strlen+1));
        copy(lastname, str);
        free(str);

        str = (char*)malloc(sizeof(char));
        strlen = 0;
        while(fscanf(fp, "%c", &c)!=EOF) {
            //printf("%c", c);
            if(c=='\n') {
                break;
            }
            else {
                //printf("%c", c);
                str[strlen] = c;
                str = (char*)realloc(str, sizeof(char)*(++strlen));
            }
        }
        str[strlen] = '\0';
        firstname = (char*)malloc(sizeof(char)*(strlen+1));
        copy(firstname, str);
        free(str);

        sort_and_insert(&base, lastname, firstname);
        free(lastname); free(firstname);
    }

    printarray(base);
    freearray(base);
    fclose(fp);
}
```

실제 입력 결과:


cse3080 > HW2 > student.txt
1 5
2 Kim Minsu
3 Kim Minju
4 Choi Hojeong
5 Cho Yujin
6 Lee Minsu
7 Choi Minjeong
cse20231515@cspro2:~/cse3080/HW2$ gcc p4.c -o p4
cse20231515@cspro2:~/cse3080/HW2$ ./p4
Cho Yujin
Choi Hojeong
Choi Minjeong
Kim Minsu
Kim Minju
Lee Minsu