**컴퓨터 알고리즘과 실습**

1. 5회차 실습 -

2017.04.05

컴퓨터공학과

2013112149 박지용

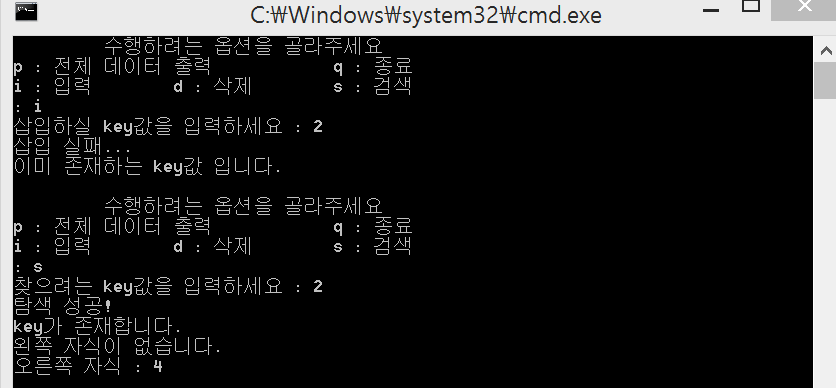
1. **실습문제 1번. 이진탐색트리의 탐색, 삽입, 삭제 연산**

<출력 결과>



트리를 만든 후에 이진탐색트리의 삽입 알고리즘(insert\_node 함수)을 거쳐 20, 6, 2, 4, 16, 10, 8, 12, 14, 9 를 연속으로 삽입해서 트리가 만들어진 모습을 출력(display 함수)하였습니다. 그 다음으론 search\_tree 함수를 이용해 10을 검색하여 10이 존재한다는 내용과 양쪽 자식 노드를 출력하였습니다. 여기선 8과 12가 출력되었습니다.

그 다음으론 delete\_node 함수를 이용해 6을 삭제한 후 다시 10을 탐색하여 양쪽 자식노드의 키값이 변한 것을 확인하였습니다. 실제로 이진탐색트리를 그려봤을 때랑 결과가 일치함을 알 수 있습니다. 여기선 10과 12가 출력되었습니다.



위의 출력화면처럼 이미 존재하는 값을 삽입하면 이미 존재하는 key값이란 메시지가 뜨고 탐색을시도했을 때 왼쪽이나 오른쪽 자식노드가 없을 경우엔(NULL일 경우엔) 이렇게 없다는 메시지를 출력합니다.

<소스코드>

// 2013112149 박지용

// 실습 1번. 완전이진트리의 삽입, 삭제, 탐색

#include <stdio.h>

#include <stdlib.h>

typedef int element;

typedef struct \_NODE

{

element key;

struct \_NODE \*left, \*right;

}NODE; //트리의 노드 구조체

//탐색 함수

NODE\* search\_tree(NODE\* root, element key)

{

NODE \*p = root;

while (p != NULL)

{

if (key == p->key) //내가 찾는 키와 동일할 때 -> 노드 반환

return p;

if(key < p->key) //내가 찾는 key가 더 작을 때 -> 왼쪽 서브트리

p = p->left;

else //내가 찾는 key가 더 클 때 -> 오른쪽 서브트리

p = p->right;

}

return p; //반복이 끝나면 노드 반환

}

//새로운 노드 삽입 함수 - 삽입 성공 시 1, 삽입 실패 시 0 반환

int insert\_node(NODE \*\*root, element key)

{

NODE \*p, \*temp, \*new\_node;

temp = \*root; //temp가 첫 노드를 가리키게 됨

p = NULL;

while (temp != NULL)

{

if (key == temp->key) //이미 해당 key가 트리에 존재할 때

return 0;

p = temp;

if (key < temp->key)

temp = temp->left;

else temp = temp->right;

}

new\_node = (NODE\*)malloc(sizeof(NODE));

if (new\_node == NULL)

return 0; //메모리 할당 오류

new\_node->key = key; //새로운 노드에 데이터 복사

new\_node->left = new\_node->right = NULL;

if (p != NULL)

{

if (key < p->key)

p->left = new\_node;

else p->right = new\_node;

}

else

\*root = new\_node;

return 1;

}

//노드 삭제함수 - 삭제 성공 시 삭제된 key값, 삭제 실패 시 0반환

int delete\_node(NODE \*\*root, element key)

{

NODE \*p, \*child, \*succ, \*succ\_p, \*t; //p는 선행노드(부모), child는 자식노드, t는 삭제할 노드

int return\_value = 0;

p = NULL;

t = \*root;

while (t != NULL && key != t->key) //둘이 같지 않으면

{

p = t;

t = (key < t->key) ? t->left : t->right; //어느쪽 서브트리로 가야되는지 결정

}

if (t == NULL)

return 0;

if ((t->left == NULL) && (t->right == NULL)) //삭제할 노드가 단말노드일 때

{

if (p != NULL)

{

if (p->left == t)

p->left = NULL;

else p->right = NULL; //부모 노드에서 나오는 간선을 삭제해준다

}

else

\*root = NULL;

}

else if ((t->left == NULL || t->right == NULL)) //자식이 하나만 있을 때

{

child = (t->left != NULL) ? t->left : t->right;

if (p != NULL)

{

if (p->left == t)

p->left = child;

else p->right = child; //노드를 삭제한 뒤에 자식을 삭제된 부모 노드의 위치로 옮긴다

}

else

\*root = child;

}

else //두 자식 모두 있는 경우

{

//오른쪽 서브 트리에서 후속자 succ를 찾고 succ\_p의 선행노드(부모)는 t가 됨

succ\_p = t;

succ = t->right;

while (succ->left != NULL)

{

succ\_p = succ;

succ = succ->left; //가장 왼쪽 노드로 이동, 후속자의 부모도 따라 이동

}

if (succ\_p->left == succ) //succ이 부모의 왼쪽 자식일 때

succ\_p->left = succ->right;

else //t의 오른쪽 자식 노드가 왼쪽 자식을 갖고있지 않을 경우

succ\_p->right = succ->right; //succ을 삭제해주고

t->key = succ->key; //t에 succ의 데이터를 옮겨준다

t = succ; //succ이었던 노드를 free해주기 위함

}

return\_value = t->key;

free(t);

return return\_value;

}

//중위순회로 출력하는 함수

void display(NODE \*p)

{

if (p != NULL)

{

display(p->left);

printf("%d\t", p->key); // 키 값을 출력

display(p->right);

}

}

//메뉴 목록

void menu()

{

printf("\t 수행하려는 옵션을 골라주세요\n");

printf("p : 전체 데이터 출력\t\t");

printf("q : 종료\n");

printf("i : 입력\t");

printf("d : 삭제\t");

printf("s : 검색\n: ");

}

int main(void)

{

char select = 0;

int flag = 0;

element key;

NODE \*dest\_node, \*root = NULL; //탐색 대상이 될 노드

insert\_node(&root, 20); insert\_node(&root, 6); insert\_node(&root, 2);

insert\_node(&root, 4); insert\_node(&root, 16); insert\_node(&root, 10);

insert\_node(&root, 8); insert\_node(&root, 12); insert\_node(&root, 14); insert\_node(&root, 9); // 노드 삽입

while(1)

{

menu();

scanf(" %c", &select);

if (select == 'i')

{

printf("삽입하실 key값을 입력하세요 : ");

scanf("%d", &key);

flag = insert\_node(&root, key); // 새로운 노드의 key를 입력받아 삽입

if (flag != 0)

printf("삽입 성공!\n");

else

printf("삽입 실패...\n이미 존재하는 key값 입니다.\n");

}

else if (select == 'd')

{

if (root == NULL)

printf("데이터가 없습니다\n");

else

{

printf("삭제하실 key값을 입력하세요 : ");

scanf("%d", &key);

flag = delete\_node(&root, key); //삭제 진행

if (flag != 0)

printf("삭제 성공!\n");

else

printf("삭제 실패...\n존재하지 않는 key값 입니다.\n");

}

}

else if (select == 's')

{

if (root == NULL)

printf("탐색 실패...\n데이터가 없습니다\n");

else

{

printf("찾으려는 key값을 입력하세요 : ");

scanf("%d", &key);

dest\_node = search\_tree(root, key); //탐색 진행

if (dest\_node != NULL)

{

printf("탐색 성공!\nkey가 존재합니다.\n");

if (dest\_node->left != NULL)

printf("왼쪽 자식 : %d\n", dest\_node->left->key); //탐색이 성공할 경우 왼쪽과 오른쪽 자식 출력

else

printf("왼쪽 자식이 없습니다.\n");

if (dest\_node->right != NULL)

printf("오른쪽 자식 : %d\n", dest\_node->right->key);

else

printf("오른쪽 자식이 없습니다.\n");

}

else

printf("탐색 실패...\n존재하지 않는 key값 입니다.\n");

}

}

else if (select == 'p')

{

if (root == NULL)

printf("데이터가 없습니다\n");

else

{

display(root); // 출력 실행

printf("\n");

}

}

else break; // 입력으로 다른 것이 들어올 경우 반복 종료

printf("\n");

}

return 0;

}

문제 1번의 소스코드입니다. 이 소스코드는 사실 수업시간에 다룬 내용이므로 그 내용과 다른 부분만 설명드리겠습니다. 문제에서 트리 T1을 만들고 거기다가 삽입하라고 했는데 root노드를 모든 함수들의 매개변수로 전달함으로써 그것을 대신하였습니다. 각 노드의 key와 왼쪽 자식노드 left, 오른쪽 자식노드 right를 멤버로 가지고 있는 NODE 구조체를 통하여 연결리스트 형식으로 트리를 구현하였습니다. 삽입, 삭제, 탐색 알고리즘에서 다른 점이 있다면, 교재나 수업에서 했던 방식에서는 root의 자식을 가지고 탐색을 시직하는데, 저는 root를 가지고 수행하였다는 것만 다르고 나머지는 모두 동일한 과정입니다. 삽입함수와 삭제함수의 출력은 정상적으로 동작하는지를 확인하기 위해 int형으로 줬습니다. 삽입함수(insert\_node)는 정상적으로 동작하면 1을, 삭제함수(delete\_node)는 삭제된 키를, 탐색함수는 탐색에 성공한 노드를 반환합니다. 또, 출력화면을 콘솔형식으로 하였고 전체 트리를 중위순회하여 현재 가지고 있는 key값들을 모두 출력하는 함수인 display()를 추가하였습니다. 나머지 세부적인 내용들은 주석으로 달아두었습니다.

◎ **실습문제 2번. Dinamic Programming을 이용한 동전교환 문제**

<유사코드>

smallest(m, c, d)

If m >= 5

X ← smallest(m-5, c, d)

If m >= 3

Y ← smallest(m-3, c, d)

If m >= 1

Z ← smallest(m-1, c, d)

return min(X, Y, Z) + 1

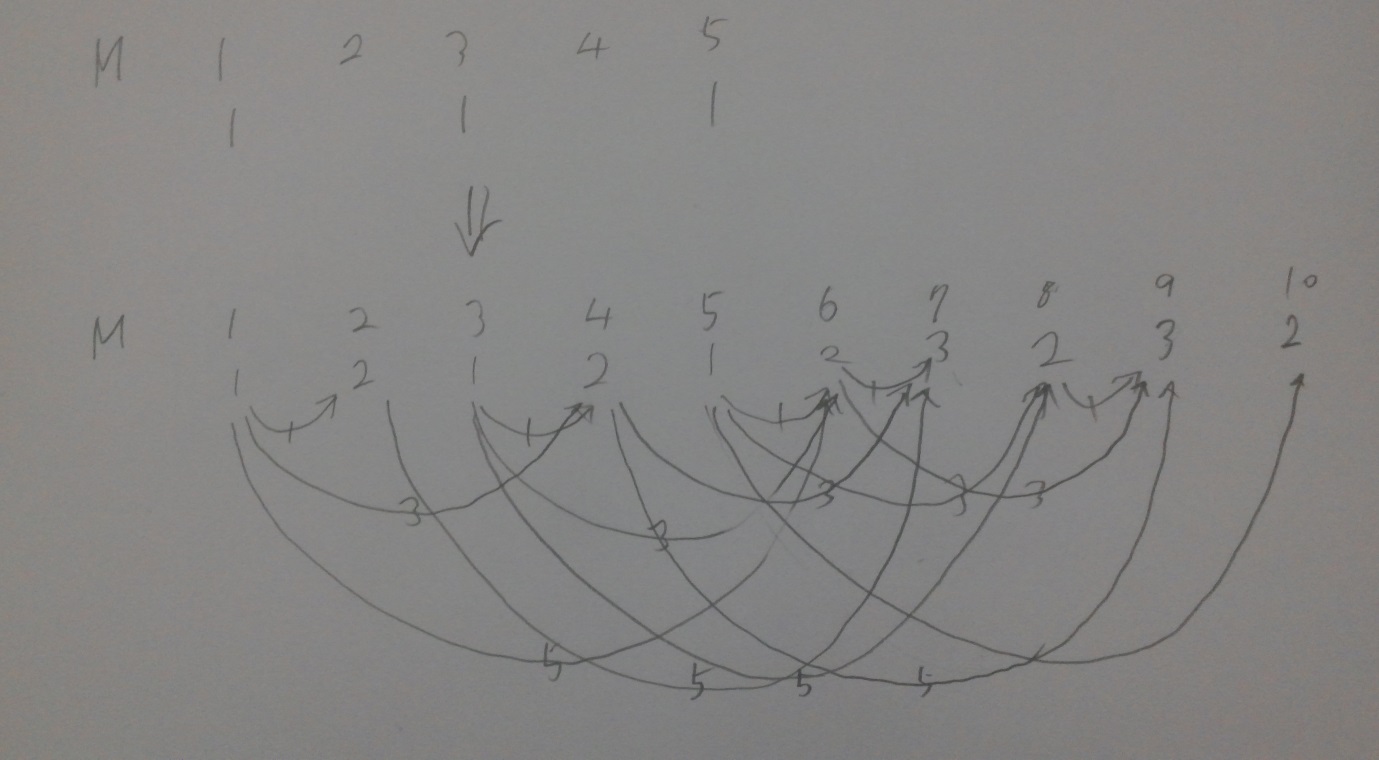
<알고리즘의 시간복잡도>

삽입, 삭제 알고리즘은 탐색에 의존하므로 시간복잡도는 같다고 볼 수 있다. 탐색의 시간복잡도는 다음과 같다.

최악 탐색 = O(n)

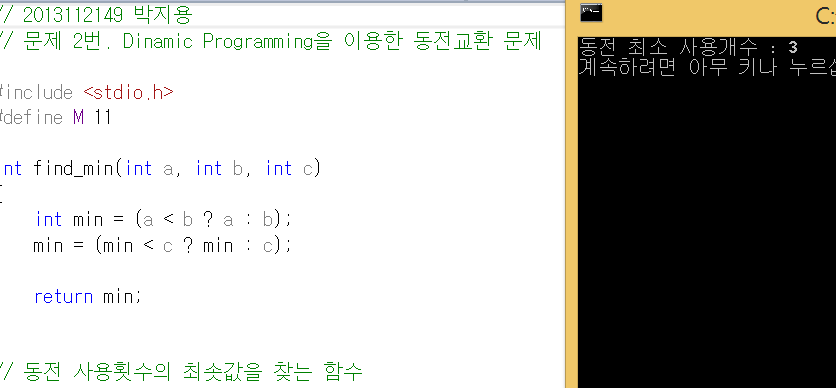
평균 탐색 = O(logn)

<진행 과정>



진행 과정을 그림으로 그려보았습니다. 그려놓고 보니 알아보기가 힘듭니다만... 동전 개수가 최소가 되려면 5개, 3개, 1개를 덜 썼을 때의 결과 중 작은 값에서 1개를 더해줘야 되는 것을 보여줍니다.

<출력 결과>



작성한 프로그램이 M=11, c = {1, 3, 5}, d = 3인 경우의 출력화면입니다. 이때의 최소로 쓴 동전의 개수는 이론적으로 계산해봤을 때 5를 2개쓰고 1을 1개 쓴 결과로 총 3개인데 그를 의미하는 3개, k = {1, 0, 2}가 정상적으로 출력되었습니다.(k는 c 배열의 각각이 몇 번 쓰였는지를 나타냅니다.)

<소스코드>

// 2013112149 박지용

// 문제 2번. Dinamic Programming을 이용한 동전교환 문제

#include <stdio.h>

#define M 11

int find\_min(int a, int b, int c)

{

int min = (a < b ? a : b);

min = (min < c ? min : c);

return min;

}

// 동전 사용횟수의 최솟값을 찾는 함수

int smallest(int m, int c[], int d)

{

int result = 0, x = 100, y = 100, z = 100; // x는 5보다 작은 범위에서 존재하지 않으므로 비교 대상으로 치지않기 위해 처음부터 아주 큰 값을 넣어줌

if (m >= 0 && m <= 1)

return m; // m이 0일 때는 무조건 0, m이 1일 때는 무조건 1

if (m >= 5)

x = smallest(m - 5, c, d);

if (m >= 3)

y = smallest(m - 3, c, d);

if (m >= 1)

z = smallest(m - 1, c, d);

return find\_min(x, y, z) + 1; // m-5번째와 m-3번째, m-1번째에서 최소인 값에서 1개의 코인을 더 써야함

}

int main(void)

{

int c[] = { 1, 3, 5 };

int m = M, d = 3, i;

int k[3] = { 0, };

printf("동전 최소 사용개수 : %d\n", smallest(m, c, d));

return 0;

}

실습 2번의 소스코드입니다. 위에서 작성했던 유사코드와 매우 흡사합니다. 이 문제에서 c의 원소5, 3, 1의 3개이기에 어떤 m(바꿔야 되는 금액)에 대하여 5짜리 동전 하나를 덜 썼을 경우의 모든 동전 조합, 3짜리 동전을 덜 썼을 경우의 모든 동전 조합, 1짜리 동전을 덜 썼을 경우의 모든 동전 조합에 각각에 대하여 해당 동전을 1개씩만 더 쓰면 m을 바꾸는 모든 조합을 구할 수 있습니다. 이 아이디어에 근거하여 어떤 m을 바꾸는 데 필요한 동전 최소개수를 반환하는 smallest 함수를 recursive 형태로 작성하여 m-5, m-3, m-1의 순환 각각에 대해 최소 개수를 구하고 그 3개를 비교하여 그 중 최소인 것을 찾아(find\_min 함수 이용) 거기에 1을 더하는 방식으로 m일 때의 최소 개수를 구하였습니다. 단, m이 5보다 작을 경우에는 m-5의 순환이 불가능하므로(m-3은 3보다 작을 경우, m-1은 1보다 작을 경우) 처음에 각각의 순환에서 나오는 반환값을 저장하는 변수인 x, y, z에 매우 큰 초기값을 부여해서 값이 들어가지 않을 때는 비교대상에서 제외될 수 있도록 하였습니다. 나머지 내용은 주석으로 달아두었습니다.

◎ **실습문제 3번. Dinamic Programming을 이용한 돌 가져가기 게임**

3.1. 먼저하는 사람을 영희, 늦게하는 사람을 철수라고 할 때 양쪽 돌의 개수(m과 n)이 모두 짝수인 경우만 철수가 이기고 나머지 모든 게임에서 영희가 승리하게된다. 따라서 이 전략을 모를 경우 선수를 잡는 것이 극도로 유리한 게임이라고 할 수 있다.

3.2. 영희(선수)가 이길 때 : W, 철수(후수)가 이길 때 : L

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 1 | W | W | W | W | W |
| 2 | W | L | W | L | W |
| 3 | W | W | W | W | W |
| 4 | W | L | W | L | W |
| 5 | W | W | W | W | W |

3.3. <유사코드>

Dinamic Programming 방법을 고려하여 위의 표를 볼 때 다음 두 가지 관점에서 볼 수 있다.

1. 영희가 패배하는 경우에서 돌이 어느 쪽이든 상관없이 한쪽에서 1개 적거나 양쪽에서 1개씩 적은 경우는 영희가 반드시 승리한다. 즉

W W

W L

의 패턴으로 4개씩 반복된다. 이는 한번에 한쪽에서 1개나 양쪽에서 1개씩 돌을 가져올 수 있으므로 지는 돌의 개수에서 1턴이 추가되는 경우엔 승리할 수 있다는 것이다.

1. 돌의 개수가 양쪽 다 짝수인 경우 철수가 승리하고 그 경우를 제외하고는 영희가 모두 승리한다. 위에서 4개씩 반복되는 성질에 의하여 이런 결론을 도출할 수 있었다. 소스코드는 각 판마다 가져가는 돌의 내용을 출력하기 위해 (1)의 방법으로 recursive 형태로 작성하지 않고 이 내용으로 작성하였다.

따라서 유사코드도 이 관점에서 기술하였다. 유사코드는 매우 단순하다. 유사코드에서 game(m, n)은 선수를 잡은 사람이 승리 시 1, 패배 시 0을 return한다.

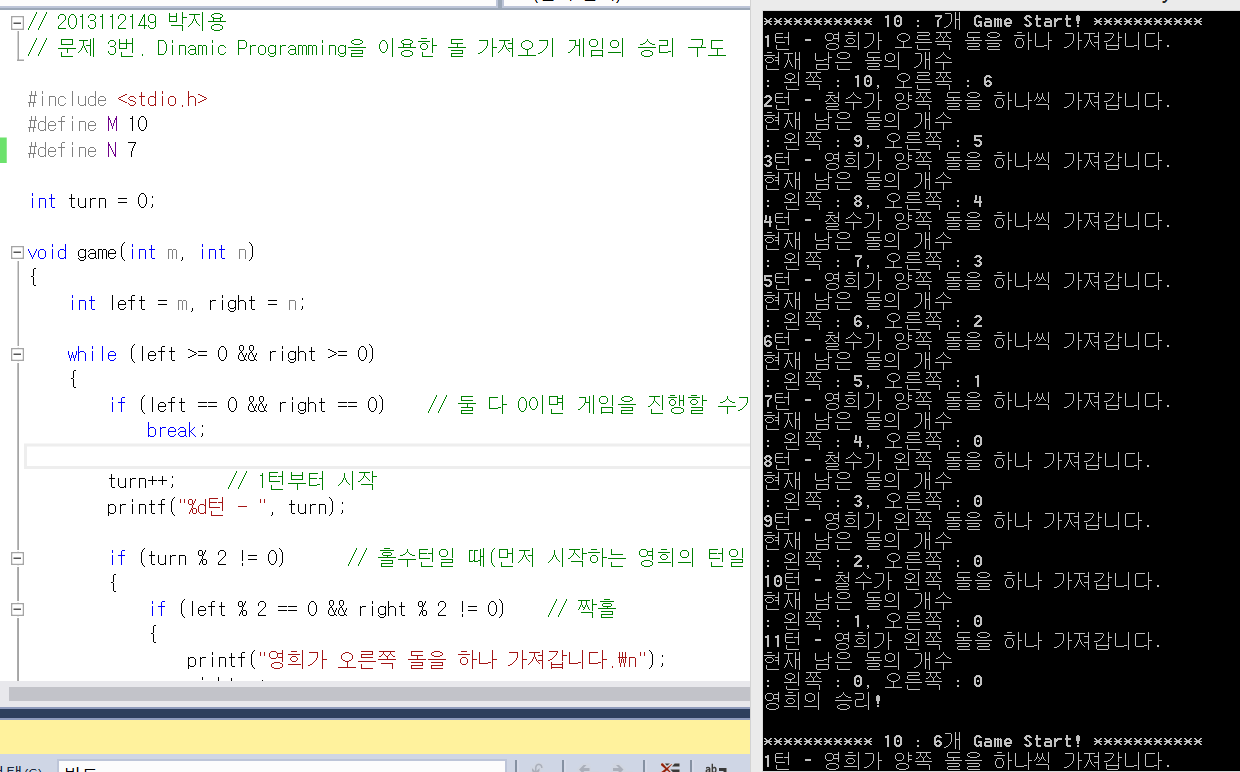
game(m , n)

If m % 2 = 0 and n % 2 = 0

return 0

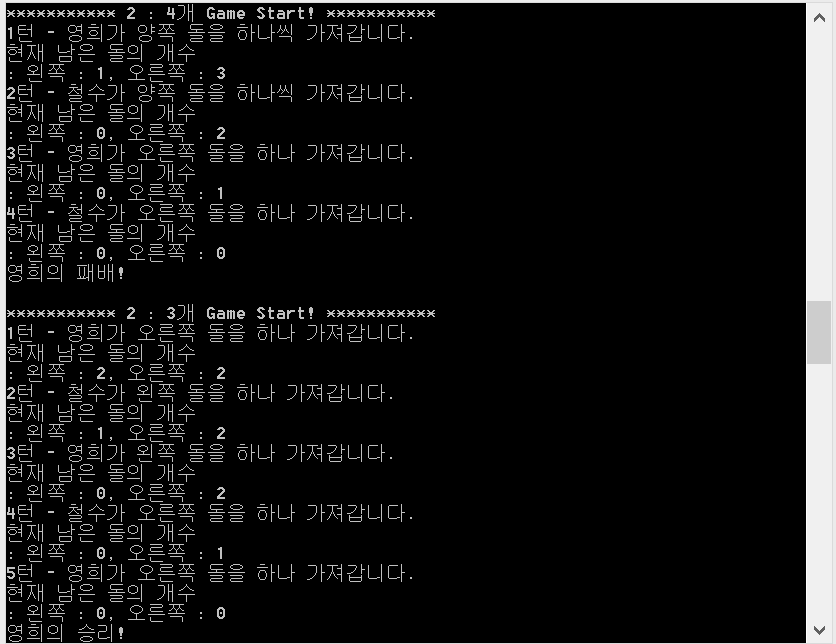
else return 1

3.4. <출력 결과>



m = 10, n = 7 일 때의 출력화면입니다.(마찬가지로 영희가 선수를 잡았습니다.)

게임이 시작할 때 돌이 양쪽에 몇 개씩 있는지를 출력한 후에 게임이 진행되는 동안 돌의 흐름이 어떻게 되는지를 모두 보여주고 결과적으로 마지막에 돌을 가져간 사람이 누구인지, 그에 따라 승자가 누구인지를 보여주도록 설계하였습니다. 아래의 화면은 영희가 질 경우(양쪽 돌의 개수가 모두 짝수일 경우)를 출력하였습니다.



<소스코드>

// 2013112149 박지용

// 문제 3번. Dinamic Programming을 이용한 돌 가져오기 게임의 승리 구도

#include <stdio.h>

#define M 10

#define N 7

int turn = 0;

void game(int m, int n)

{

int left = m, right = n;

while (left >= 0 && right >= 0)

{

if (left == 0 && right == 0) // 둘 다 0이면 게임을 진행할 수가 없음

break;

turn++; // 1턴부터 시작

printf("%d턴 - ", turn);

if (turn % 2 != 0) // 홀수턴일 때(먼저 시작하는 영희의 턴일 때)

{

if (left % 2 == 0 && right % 2 != 0) // 짝홀

{

printf("영희가 오른쪽 돌을 하나 가져갑니다.\n");

right--;

}

else if (left % 2 != 0 && right % 2 == 0) // 홀짝

{

printf("영희가 왼쪽 돌을 하나 가져갑니다.\n");

left--;

}

else if (left % 2 != 0 && right % 2 != 0) // 홀홀

{

printf("영희가 양쪽 돌을 하나씩 가져갑니다.\n");

left--; right--;

}

else // 짝홀

{

if ((left > 2 || right > 2) && left != 0 && right != 0) /\* 왼쪽2, 오른쪽2 일 경우 양쪽에서 1개씩 가져가면 안되기 때문

그리고 left나 right가 0이면 양쪽에서 가져올 수가 없다 \*/

{

printf("영희가 양쪽 돌을 하나씩 가져갑니다.\n");

left--; right--;

}

else if (left == 0) // left가 0이면 왼쪽에서 가져올 수가 없다

{

printf("영희가 오른쪽 돌을 하나 가져갑니다.\n");

right--;

}

else

{

printf("영희가 왼쪽 돌을 하나 가져갑니다.\n"); // 왼쪽에서 가져올 수 있는 경우 왼쪽에서 우선적으로 가져옴(3:2나 2:3은 동일하기 때문)

left--;

}

}

}

else // 짝수턴일 때(늦게 시작하는 철수의 턴일 때)

{

if (left % 2 == 0 && right % 2 != 0) // 짝홀

{

printf("철수가 오른쪽 돌을 하나 가져갑니다.\n");

right--;

}

else if (left % 2 != 0 && right % 2 == 0) // 홀짝

{

printf("철수가 왼쪽 돌을 하나 가져갑니다.\n");

left--;

}

else if (left % 2 != 0 && right % 2 != 0) // 홀홀

{

printf("철수가 양쪽 돌을 하나씩 가져갑니다.\n");

left--; right--;

}

else // 짝홀

{

if ((left > 2 || right > 2) && left != 0 && right != 0) /\* 왼쪽2, 오른쪽2 일 경우 양쪽에서 1개씩 가져가면 안되기 때문

그리고 left나 right가 0이면 양쪽에서 가져올 수가 없다 \*/

{

printf("철수가 양쪽 돌을 하나씩 가져갑니다.\n");

left--; right--;

}

else if (left == 0) // left가 0이면 왼쪽에서 가져올 수가 없다

{

printf("철수가 오른쪽 돌을 하나 가져갑니다.\n");

right--;

}

else

{

printf("철수가 왼쪽 돌을 하나 가져갑니다.\n"); // 왼쪽에서 가져올 수 있는 경우 왼쪽에서 우선적으로 가져옴(3:2나 2:3은 동일하기 때문)

left--;

}

}

}

printf("현재 남은 돌의 개수\n: 왼쪽 : %d, 오른쪽 : %d\n", left, right);

}

}

int main(void)

{

int m, n;

for (m = M; m >= 8; m--)

{

for (n = N; n >= 6; n--)

{

printf("\n\*\*\*\*\*\*\*\*\*\*\* %d : %d개 Game Start! \*\*\*\*\*\*\*\*\*\*\*\n", m, n);

game(m, n);

if (turn % 2 == 0)

{

printf("영희의 패배!\n");

turn = 0; // 다음 판을 하려면 turn을 0으로 복귀시켜줌

}

else

{

printf("영희의 승리!\n");

turn = 0; // 다음 판을 하려면 turn을 0으로 복귀시켜줌

}

}

}

return 0;

}

실습 3번의 소스코드입니다. 위에서 이야기했듯이 돌을 가져가는 game형태로 소스코드를 구현하기 위해 recursive형태로 하지 않고 game()이란 함수를 써서 m과 n이 모두 짝수이면 철수가 이기는 성질을 이용하여 작성하였습니다. 처음 인자로 받은 m을 left에 저장하고 n을 right에 저장한 후에 왼쪽에서 하나를 가져갈 때마다 left를 1 낮춰주고 오른쪽에서 하나를 가져갈 때마다 right를 낮춰주는 식으로 작성하였고 main함수에서 이중 for문을 통해 m과 n값을 조절해가면서 각 경우에 대한 출력을 할 수 있습니다. 현재 붙인 코드에서 m을 8부터, n을 6부터로 설정한 이유는 범위가 클 경우에 출력화면의 내용이 너무 많아 잘리기 때문입니다. 0부터 시작해도 정상적으로 동작합니다. 다만 0, 0일 경우는 애초에 게임 자체가 성립되지 않으므로 처리하지 않습니다. 그외의 부수적인 조건은 if문으로 모두 해결하였습니다. 자세한 소스코드 설명은 주석으로 달아두었습니다.

◎ **실습문제 4번. 증식하는 박테리아와 잡아먹는 바이러스 문제**

i) n = 1 일 경우

|  |  |  |
| --- | --- | --- |
|  | 처음 | 1분 후 |
| 박테리아 | 1 | 0 |
| 바이러스 | 1 | 2 |

1분 후에 모두 죽습니다.

ii) n = 2 일 경우

|  |  |  |  |
| --- | --- | --- | --- |
|  | 처음 | 1분 후 | 2분 후 |
| 박테리아 | 2 | 2 | 0 |
| 바이러스 | 1 | 2 | 4 |

2분 후에 모두 죽습니다.

iii) n = 3 일 경우

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 처음 | 1분 후 | 2분 후 | 3분 후 |
| 박테리아 | 3 | 4 | 4 | 0 |
| 바이러스 | 1 | 2 | 4 | 8 |

3분 후에 모두 죽습니다.

iv) n = 4 일 경우

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 처음 | 1분 후 | 2분 후 | 3분 후 | 4분 후 |
| 박테리아 | 4 | 6 | 8 | 8 | 0 |
| 바이러스 | 1 | 2 | 4 | 8 | 16 |

4분 후에 모두 죽습니다.

이런 식으로 n분 후에 모두 죽습니다.

바이러스는 2의 거듭제곱(0제곱부터)꼴로 커져가고 x분 후에는 2의 x승이 됩니다. x분째에 박테리아는 2\*{(x-1분 째의 박테리아 수)-} 마리입니다.

이를 식으로 나타내면 다음과 같습니다.

즉, 이 0이 될 때의 x가 우리가 찾는 답인데, 가 n이라는 것을 근거로 이 점화식을 정리하면 x = n임을 알 수 있습니다.