Algorithm final project Report

팀 : 15조

팀원 : 서유영, 안찬우, 유효민

1. Role of each member

	서 유 영	안 찬 우	유 효 민
프로그램 설계	O	o	O
모듈 구현	o	o	O
UI	o		
Report 작성	O	O	O

2. Introduction

이 프로그램은 Windows 환경에서 Visual studio를 이용해 개발하였다. 프로그램은 여행 일정 예약 프로그램으로 관광지 간 교통 편과 호텔 가격을 고려해 일정을 잡는다. 관광지 간 경로는 Graph로 구현하였고 각 관광지에 위치한 호텔은 가격에 따라 RB tree로 표현하였다. 경로 검색에는 Dijkstra 알고리즘을 이용했다.

< 기본 기능 >

- 여행 기간 계산
- 호텔 예약
- 경로 선택
- 관광지 선택
- ID에 따른 예약정보 저장
- 가격

< 추가 기능 >

- 각 관광지 관광 시간

< 제한되는 부분 >

- 날짜에 따른 일정 분류

3. User interface and modules description

1. User interface

1) 시작화면

메뉴를 선택할 수 있는 화면으로, 1번은 여행 계획을 짜는 메뉴, 2번은 여행 계획을 조회하는 메뉴, 3번은 여행 계획 예약을 취소하는 메뉴, 4번은 프로그램을 종료하는 메뉴이다.

2) 1번 메뉴

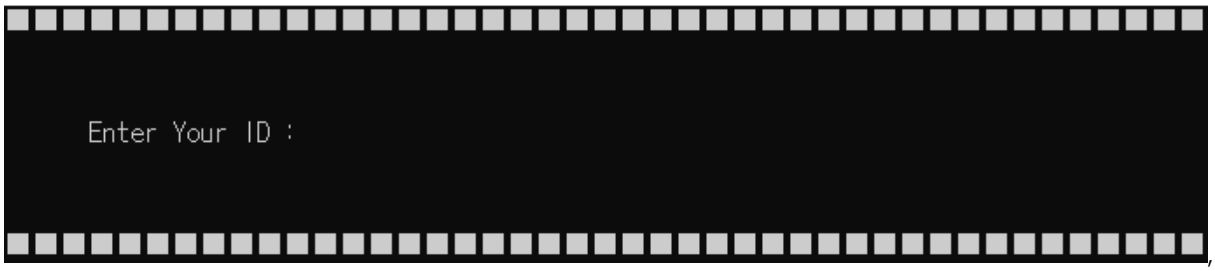
예약에 필요한 정보들을 입력하면 프로그램이 예산, 계획을 고려하여 여행 계획을 짜준다. 모든 정보를 입력하고 엔터를 누르면 계획 생성이 가능한지, 불가능한지 판단한다. 가능할 경우 사용자 정보를 RBtree에 저장하고 메뉴로 돌아간다. 불가능할 경우 불가능 메시지를 띄우고 메뉴로 돌아간다.

- 가능할 경우

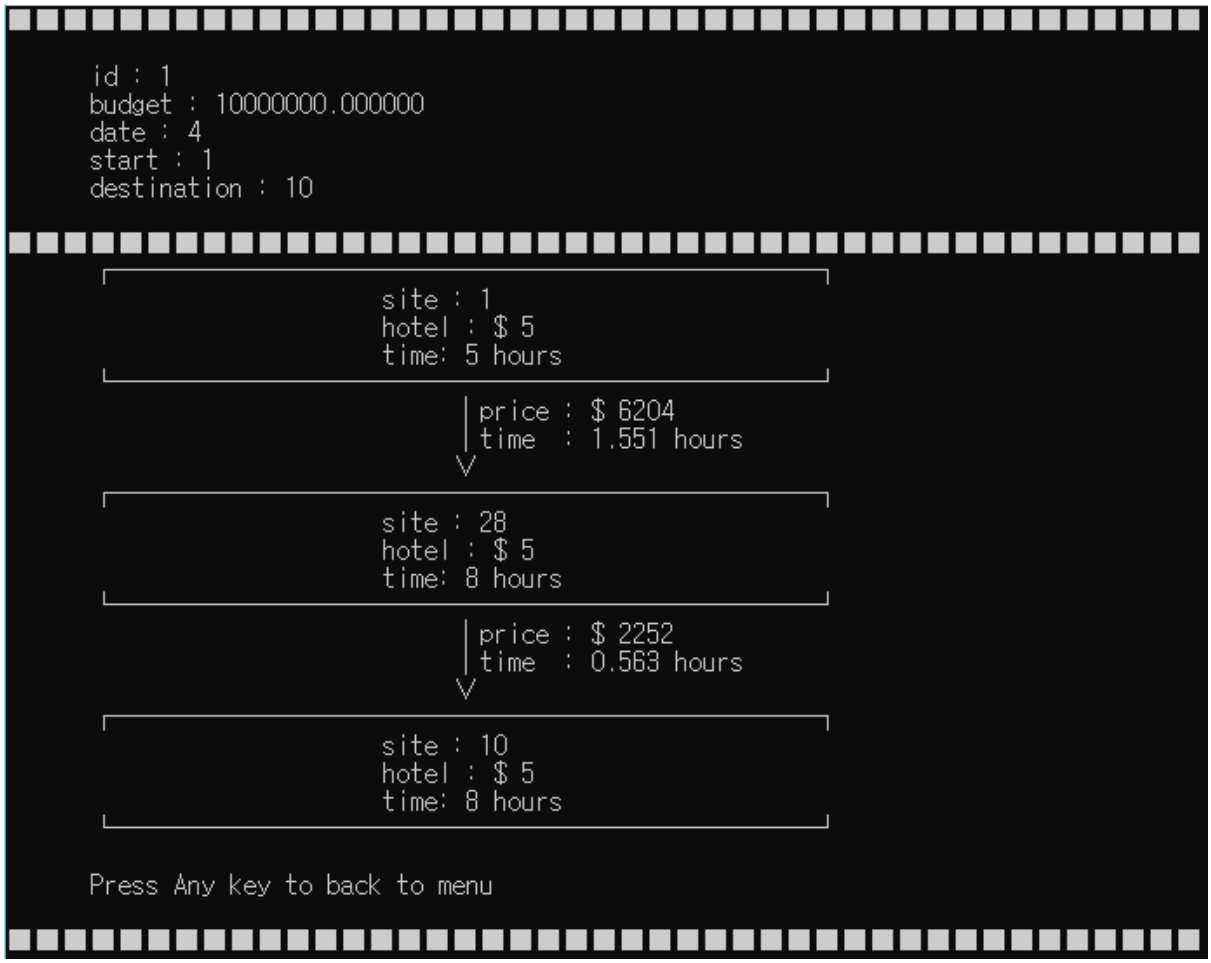
- 불가능할 경우

2) 2번 메뉴

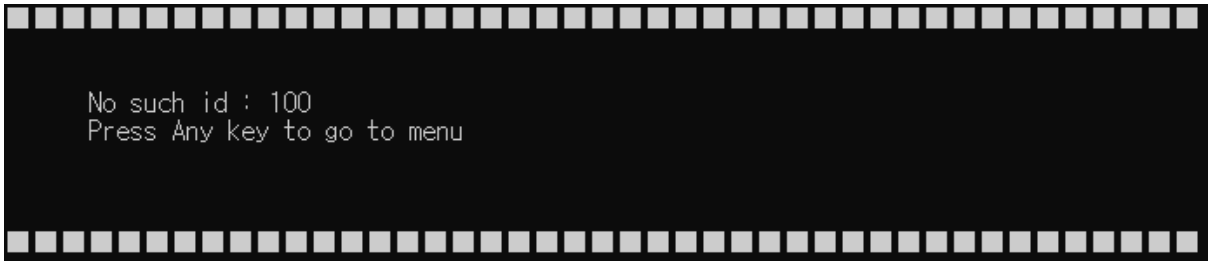
사용자의 ID를 입력하면 이것을 RBtree에서 검색한다. 검색결과가 존재할 경우 여행 내용을 출력하고, 없을 경우 없다는 메시지를 출력하고 메뉴로 돌아간다.



- 있을 경우

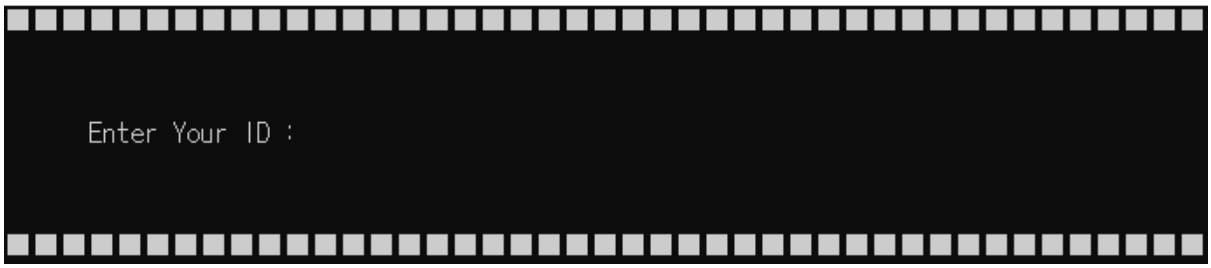


- 없을 경우

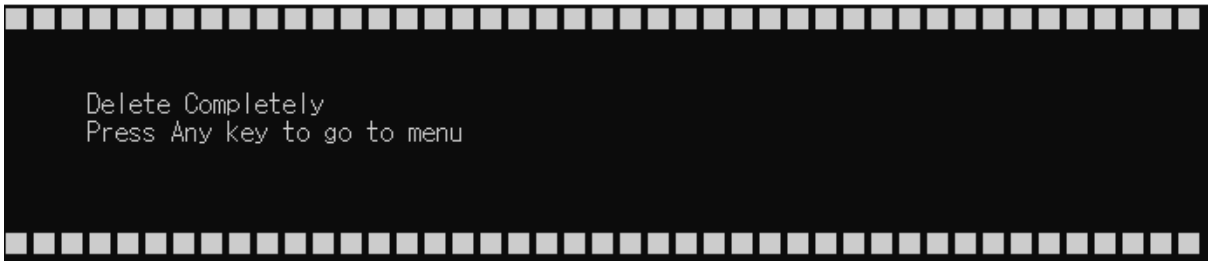


3) 3번 메뉴

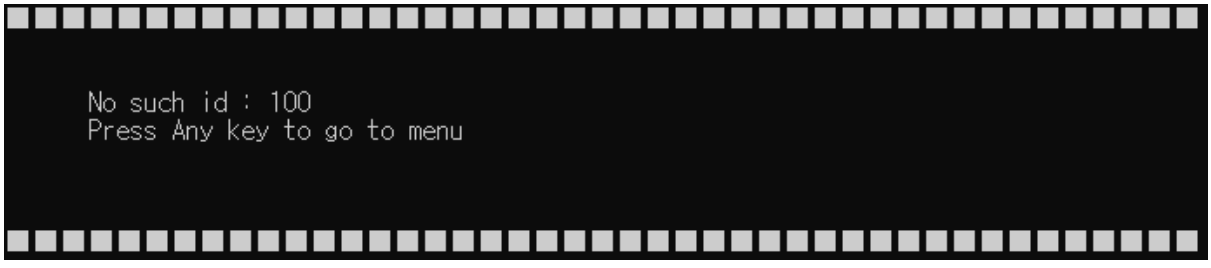
사용자 ID를 입력하면 그 ID를 RBtree에서 검색한다. 검색 결과가 존재할 경우, 해당하는 사용자 예약 정보를 RBtree에서 삭제한다. 없을 경우 없다는 메시지를 출력하고 메뉴로 돌아간다.



- 있을 경우



- 없을 경우



4) 4번 메뉴

입력 시 프로그램을 종료한다.

2. Modules

1) 관광지 정보

관광지 정보에는 해당 관광지를 여행하는데 걸리는 시간과 호텔 RB tree와 연결하기 위한 포인터가 들어간다.

```
typedef struct _sites {
        int tourtime; //관광지 여행에 걸리는 시간
        Hotel *H;
        }sites;
```

2) 사용자 정보

사용자 정보에는 id, 예산, 여행 기간, 목적지, 출발지, 예약정보를 저장하기 위한 배열이 들어간다. 사용자 정보는 RB tree를 이루기 때문에 RB 색상, 양쪽 자식 노드와 부모 노드를 가리키는 포인터도 들어간다.

```
typedef struct __Person
{
        int id;
        float budget;
        int date;
        int dst;
        int start;
        float data[4][101];

        enum Color color;
        struct __Person  *left;
        struct __Person  *right;
        struct __Person  *p;
        }Person;
```

3) 호텔 정보

호텔 구조체는 관광지마다 100개씩 들어가며 RB tree를 이룬다. 호텔 정보는 가격, id, RB 색상, 양쪽 자식 노드와 부모 노드를 가리키는 포인터가 들어간다.

```
typedef struct __Hotel
{

        float price;
        int id;
        enum Color color;
        struct __Hotel * left;
        struct __Hotel * right;
        struct __Hotel * parent;

        }Hotel;
```

4) 함수

system function	void Person_init(Person * p , char * destination);
	Hotel * NEW_NODE(Hotel * parent, float z);
	void swapColors(Hotel *x1, Hotel *x2);
	void swapValues(Hotel *u, Hotel *v);
	Hotel *successor(Hotel *x);
	int isOnLeft(Hotel * self);

```
Hotel * uncle(Hotel * parent);
Hotel * find_replace(Hotel *x);
Hotel * RR_ROTATE(Hotel * par);
Hotel * LL_ROTATE(Hotel * par);
Hotel * LR_ROTATE(Hotel * par);
Hotel * RL_ROTATE(Hotel * par);
void FIX_FUNCTION(Hotel * node, Hotel * root);
Hotel * TREE_SEARCH(Hotel * root,float k);
Hotel * RB_INSERT(Hotel * root, float z);
void PRINT_BST(Hotel * root);
int height(Hotel * root);
void deleteNode(Hotel * v, Hotel * root);
void fixDoubleBlack(Hotel * x, Hotel * root);
void fixRedRed(Hotel * x, Hotel * root);
void rightRotate(Hotel * x, Hotel * root);
void leftRotate(Hotel *x, Hotel * root);
int hasRedChild(Hotel * v);
void moveDown(Hotel * nParent, Hotel * v);
Hotel * find_replace(Hotel *x);
Hotel * uncle(Hotel * parent);
int isOnLeft(Hotel * self);
void deleteByVal(float n, Hotel * root);

int isEmpty(Priority_Node ** head);
void push(Priority_Node ** head, int vertex, int priority);
void pop(Priority_Node ** head);
Priority_Node * peek(Priority_Node ** head);
Priority_Node * Priority_New_Node(int vertex, int priority);

void make_sites(sites * S);
void make_graph(int graph[][SIZE+1]);

void dijkstra(int graph[][SIZE + 1], int * index, int start_vertex);
int *  print_path(int start, int end, int* parent_vertex);
float find_cheapest_hotel(Hotel * root);
int check_route(Person* p, sites * S, int *path_vertex, int graph[][SIZE + 1]);
void Connect(sites *S);

Person* left_rotate(Person* tree, Person* base);
Person* right_rotate(Person* tree, Person* base);
Person* RB_insert_fixup(Person* tree, Person* input);
Person* RB_insert(Person* tree, Person* input);
Person* make_node(int i, float bud, int dat, int st, int ds);
Person* make_nil();
Person* tree_minimum(Person* root);
Person* tree_successor(Person* root);
Person* RB_delete_fixup(Person* tree, Person* put);
Person* RB_delete(Person* tree, Person* put);
Person* tree_search(Person* tree, int d);
void print_inorder(Person* tree, int level);
void delete_all(Person* tree);
```

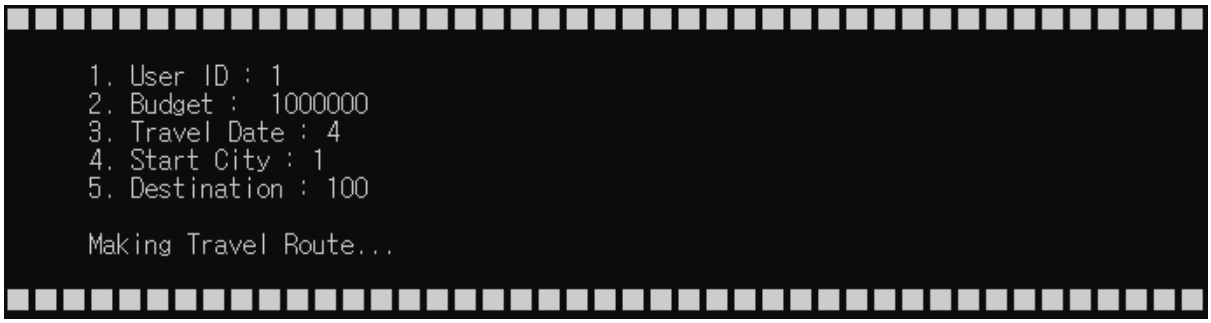
위의 함수들은 크게 Hotel 을 생성하고 저장하는 RB tree 함수, 다익스트라 알고리즘을 위한 큐 관련 함수, 사이트를 만들고 이것을 호텔과 연결하는 함수, 여행 루트를 만들고 주어진 가격과 기간 내에 여행이 가능한지 판별하는 함수, 사용자 ID 를 저장하는 RB tree 함수로 구분할 수 있다.

4. Extensive execution examples and analysis

프로그램을 실행하면 UI 화면이 뜨기 전 make_graph, make_sites, Connect 함수를 사용해 사이트 그래프와 각 사이트에 있는 호텔 정보들을 RBtree에 저장한다.

예약을 위해 1번을 입력하면 아래와 같은 UI가 뜨며, 안내에 따라 정보를 모두 기입한다.

모두 기입하면 아래와 같은 창이 뜨면서 프로그램이 여행 계획을 작성한다. 우선 다익스트라 함수를 통해 시작지부터 목적지까지의 direct 경로를 구한다. 이것을 path_arr이라는 배열에 저장한다.



check_route함수를 통해 사용자가 입력한 예산과 기간으로 여행이 가능한지 판단한다. 우선 각 sites를 지나면서 sites 정보, site에서 예약할 호텔의 가격정보, site의 tourtime, 한 site부터 다음 site까지의 거리정보를 Person 구조체 안의 data라는 2차원 배열에 저장한다.

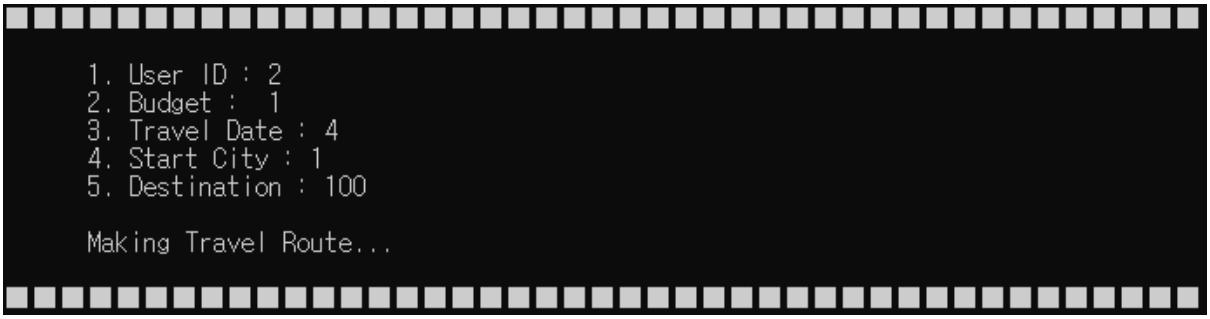
site 정보	1	84	…	100

호텔 가격	5	5	…	5
site tourtime	9	9	…	10
site간의 거리	784	1069	…	0

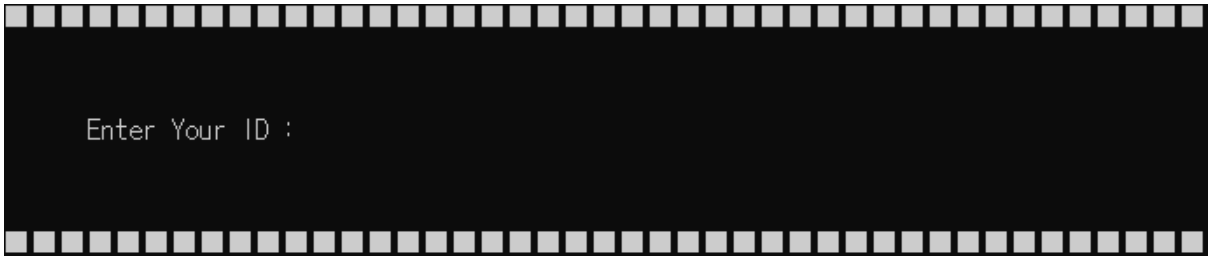
다음으로 이 배열을 탐색하면서 total_price와 total_date를 구한다. Total_price는 호텔 가격과 site를 이동할 때 필요한 돈의 합으로, 후자의 경우에는 거리에 비례하여 거리*4로 책정한다. Total_date는 site tourtime과 site를 이동할 때 필요한 시간의 합으로, 후자는 마찬가지로 거리에 비례하여 거리/1000으로 책정한다.

만약 total_price>p->budget이고, total_date>p->date일 경우 입력한 사용자 정보를 id를 기준으로 해서 RBtree에 삽입한다. 삽입이 완료된 후에는 아래와 같은 화면이 뜨며 아무키나 눌러 메인 메뉴로 돌아갈 수 있다.

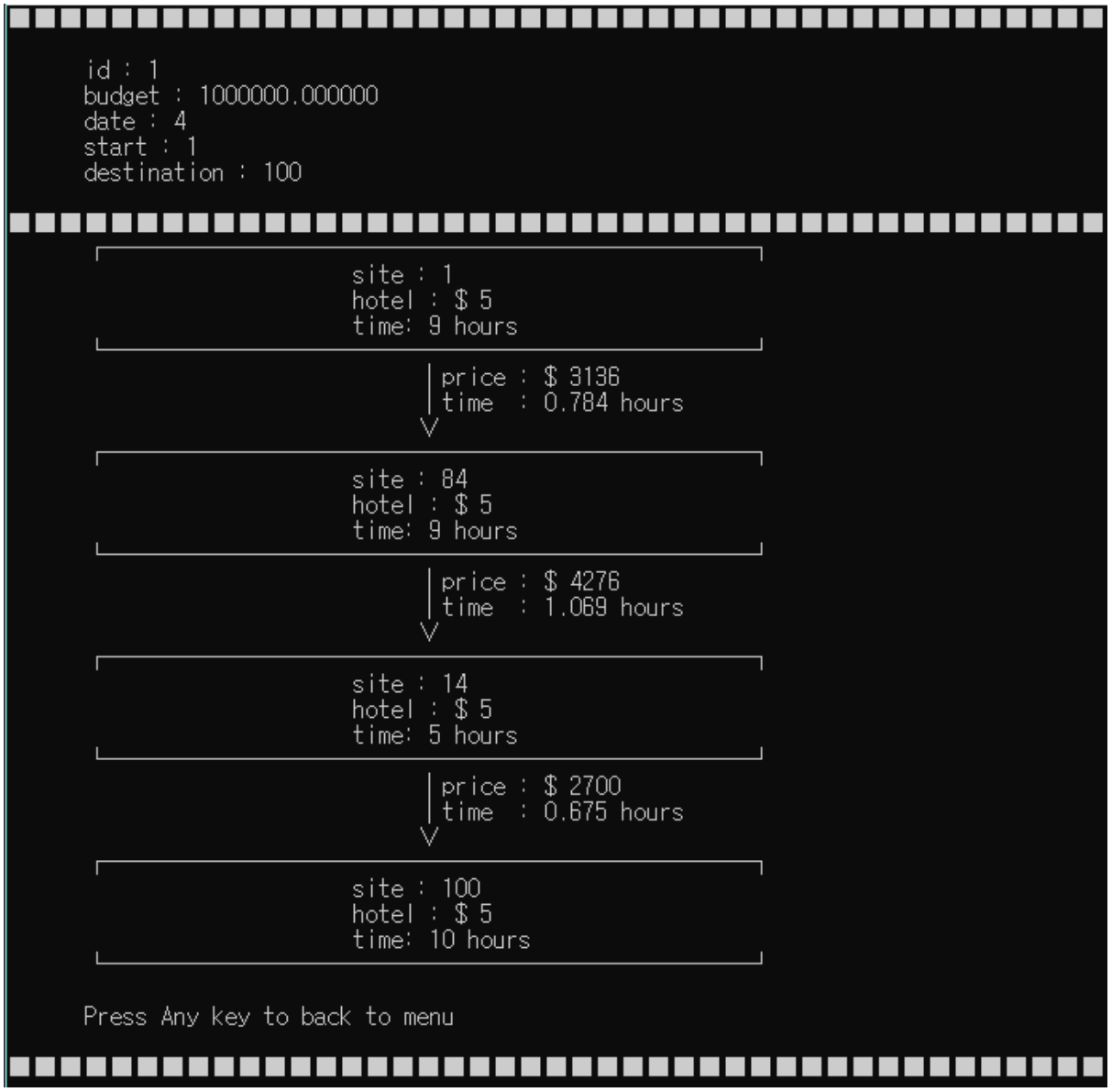
반대로 total_price<p->budget이거나, total_date<p->date일 경우 사용자 정보를 저장하지 않고 아래와 같은 화면을 띄운 후 메인메뉴로 돌아갈 수 있다. 아래는 예시이다. 아래에는 budget이 1로, (당연하게도) 예산이 여행을 하기에는 부족하므로 여행 계획이 성립되지 않아 저장되지 않는다.



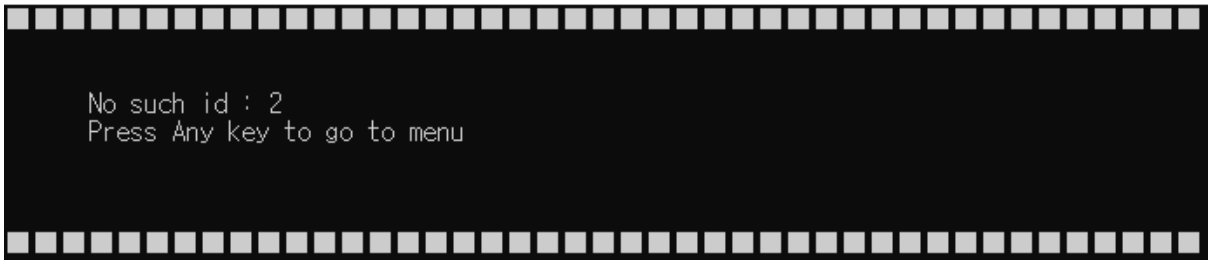
다음으로 예약 정보를 확인하기 위해 메인메뉴에서 2를 입력한다. 입력하면 아래와 같이 ID를 입력하는 창이 뜬다. 여기에 ID를 입력하면 ID 정보가 저장된 RBtree에서 해당 ID를 검색한다.



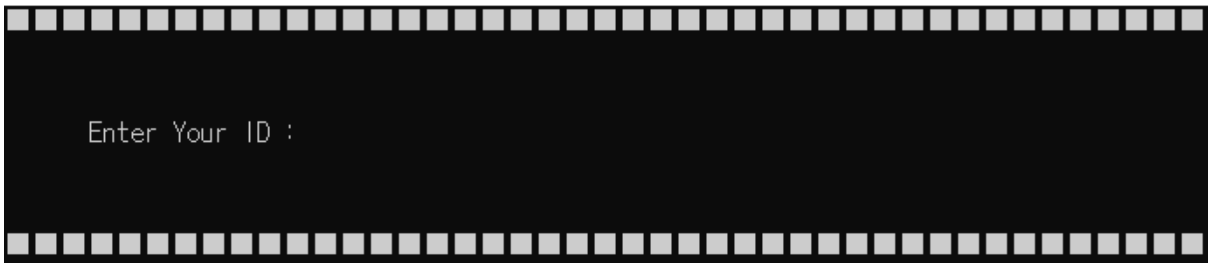
검색결과가 존재할 경우 사용자의 ID, 예산, 여행 기간, 시작지, 도착지의 정보가 출력되며, 그 아래에 세부적인 여행 계획이 출력된다. 출력은 각 site마다의 정보와 site간 이동에 필요한 가격과 시간이 함께 출력된다. 아무 키나 누르면 다시 메인메뉴로 돌아갈 수 있다.



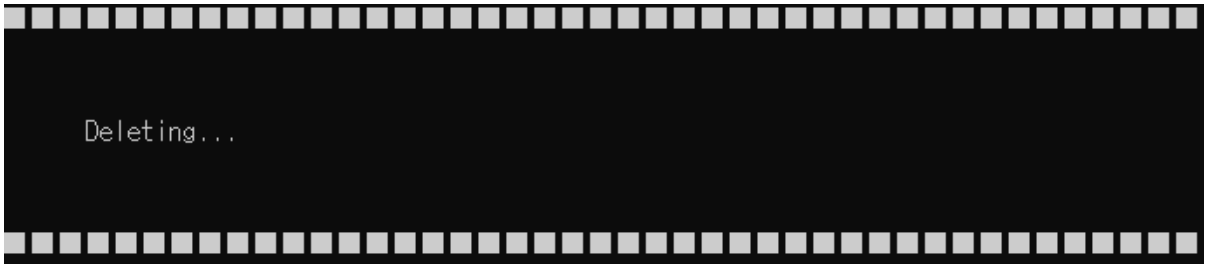
만약 탐색 결과, 해당 ID가 존재하지 않을 경우에는 아래의 화면이 뜨면서 메인메뉴로 돌아간다.

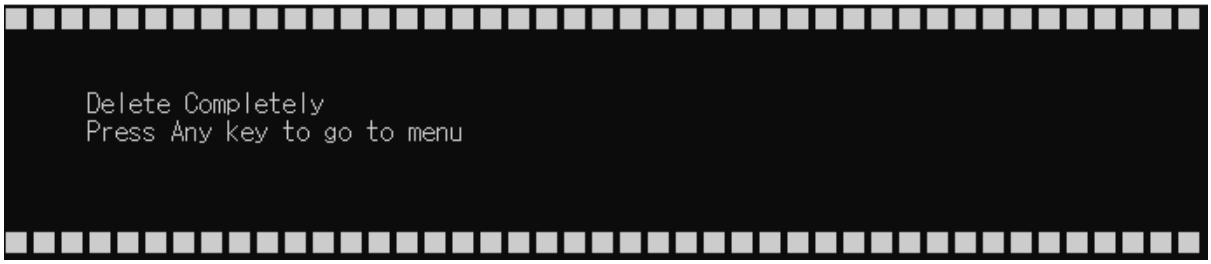


예약 정보를 지우기 위해 메인메뉴에서 3을 입력하면 2와 같이 ID를 입력하는 화면이 뜬다. ID를 입력하면 마찬가지로 RBtree에서 ID를 탐색하고 탐색 결과에 따라 다른 화면이 출력된다.

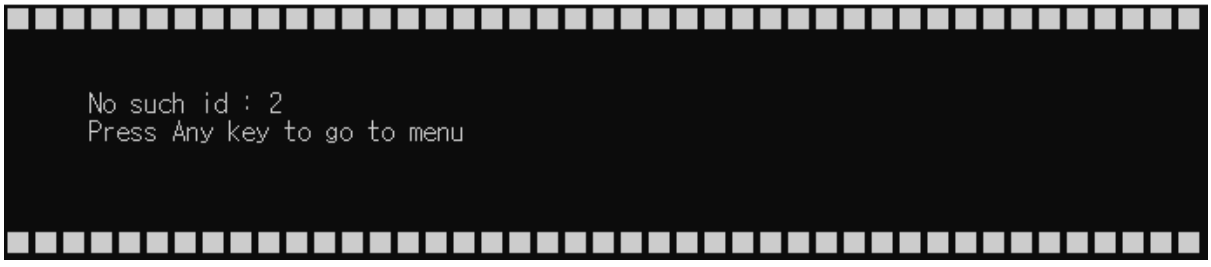


취소하고자 하는 ID가 존재할 경우 아래와 같은 화면이 뜨면서 RBtree에서 해당 노드를 delete한다. 삭제가 완료된 후에는 아무키나 눌러 메인메뉴로 돌아갈 수 있다.





존재하지 않을 경우에는 2번 메뉴에서와 같은 화면이 출력된다.



마지막으로 메인메뉴에서 4를 입력할 경우 프로그램이 즉시 종료된다.

5. Discussion and conclusion

여행 일정 예약 프로그램을 작성하면서 어려웠던 점은 다음과 같다.

1) 전체적인 틀 잡기

site와 hotel 정보를 연결하는 것에 어려움이 있었다. 과제 내용 이해에 있어 어려움이 있던 것으로, 조원들끼리 기초 설계에 대해 자신의 의견을 나누면서 알맞은 구현 방법을 결정하였다.

2) 다익스트라 알고리즘 경로 저장

다익스트라 알고리즘 실행 후 거리 계산은 정상적으로 작동했지만 경로 저장이 힘들었다. 다음 노드로 이동 시 부모 노드 정보를 저장하여 노드끼리 이어지게 만들어 경로 출력이 가능하도록 했다.

3) 레드 블랙 트리에 노드가 저장

노드를 삽입을 한 후 트리를 업데이트해야했지만 업데이트하는 코드가 빠져 호텔 정보가 각 사이트에 저장이 되지 않았다. 이후 코드 수정을 통해 이 문제를 해결했다.

4) UI

폰트의 차이로 visual studio의 작업 영역와 cmd에서 보여지는 UI가 차이가 나서 cmd 우선으로 UI를 수정하는 것이 힘들었다. 또한 cmd창이 바로 꺼져버리는 문제가 있고, system("pause")를 추가하면 프로그램이 바로 종료되어버리는 문제가 있었다. 이것은 한 메뉴의 기능이 종료될 시에 _getch()함수를 추가하여 키 입력을 하면 메뉴로 돌아가는 방식으로 바꾸어 해결했다.

추가 및 보완해야 할 내용은 다음과 같다.

1) direct 경로가 아닌 indirect 경로 구현

현재는 direct 경로로만 여행 일정 예약을 했기에 여행 기간이 남을 경우 아무 것도 할 수 없다. Indirect 경로를 추가하여 여행 기간과 예산이 남을 경우 추가 site를 방문할 수 있게 한다.

2) 날짜 별로 일정 계획

1일차, 2일차와 같은 방식으로 24시간을 기준으로 하여 여행 계획을 작성한다. 특히 호텔의 경우 현재는 한 site를 방문할 때마다 호텔을 예약하도록 되어있는데, 이것을 하루의 마지막 site에서 호텔 예약이 가능하도록 설정한다.

3) 교통수단의 종류

현재는 교통수단이 하나라는 가정 하에, 거리에 비례하여 시간과 가격이 나오도록 설정했다. 교통수단의 종류를 늘려서 이용하는 교통수단에 따라 시간과 가격이 다르게 나오게 하여 여행 일정의 현실성을 증가시킨다.