```
================================================================================
            /local/submit/submit/comp10002/ass2/renjiem/src/myass2.c
================================================================================

/*****************************************************
 * This code is written by Renjie Meng               *
 *  2017/10/12                                       *
 * This is all of COMP10002                          *
 * Foundations of Algorithms                         *
 * 2017 S2 Assignment2                               *
 *                                                   *
 *It is a program about uber path finding.           *
 *                                                   *
 *                                                   *
 *   Algorithms are fum!                             *
 *                                                   *
 *****************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define DEBUG 1
#if DEBUG
#define DUMP_DBL(x) printf("line %d: %s = %.5f\n", __LINE__, #x, x)
#else
#define DUMP_DBL(x)
#endif

#define ONEWAY 999 /*One way streets have cost of 999*/
typedef struct {
    int street_num_x;
    int street_num_y;
} dimension_t; /*The size of the block*/
typedef struct {
    int street_x;
    char street_y;
}loc_t;/*The address of a location*/
typedef struct {
    dimension_t size;
    int nintersections;
    int possibilities;
    int useless;
    int total_cost;
    int nloc_supplied;
    loc_t first;
    loc_t last;
} data_stage1_t;/*data for stage 1*/
typedef struct {
    loc_t location;
    loc_t from;
    int east;
    int north;
    int west;
    int south;
    int total_cost;
    int gowest;
    int gosouth;
    int fromwest;
    int fromsouth;
}map_t;/*a map used to store infomation*/
typedef struct {
    loc_t from;
    int cost;
    int num;
}route_t;/*a route include from and cost*/

/*************************************************************/

/* function prototypes */

void read_size(dimension_t*);
```

```
75  map_t **create_a_new_map(dimension_t );
    loc_t* read_loc_supplied(int *);
    void count(int, int *, int *);
    void stage1_data(data_stage1_t*, dimension_t,
        map_t**, loc_t*, int );
80  void print_stage1(data_stage1_t);
    int compare_loc(loc_t current_from,
        loc_t stored_from);
    int update_location_info(int now_total, int now, int *total_cost,
        loc_t current_from, loc_t* stored_from);
85  int is_location_exist(int x, int y, dimension_t size);
    map_t** best_cost(loc_t start, map_t **map, dimension_t size);
    route_t* recursive_add(route_t* route, map_t **map, loc_t start,
        loc_t end, int *current_size, int *i);
    route_t* navigator_one_route(map_t **map, loc_t start,
90      loc_t end);
    void print_one_route(route_t *route);
    void print_gowest();
    void print_fromwest();
    void print_gosouth();
95  void print_fromsouth();
    void print_stage3(dimension_t size, map_t **map);
    int go_direction(loc_t location, loc_t from);
    void assign_go(int *go);
    map_t** assign_direction_go(map_t **map, dimension_t size);
100
    /*************************************************************/

    /* main program controls all the action
     */
105 int
    main(int argc, char* argv[]) {
        dimension_t size;
        map_t **map;
        loc_t *loc_supplied;
110     data_stage1_t data_stage1;
        route_t *route;
        int num_loc_supplied,
        i, j, y1;

115     /*stage 1*/
        read_size(&size); /*read and assign the size of the grid*/
        y1 = size.street_num_y;
        map = create_a_new_map(size);/*read and assign the map*/
        /*read and assign the location supplied*/
120     loc_supplied = read_loc_supplied(&num_loc_supplied);
        stage1_data(&data_stage1, size, map, loc_supplied,
            num_loc_supplied);
        print_stage1(data_stage1);

125     /*stage 2*/
        map = best_cost(loc_supplied[0], map, size);
        /*generate the best cost fo the map*/
        for (i=1; i<num_loc_supplied; i++){
            /*print out best route for each end location*/
130         route = navigator_one_route(map, loc_supplied[0],
                loc_supplied[i]);
            print_one_route(route);
        }
        printf("\n");
135
        /*stage 3*/
        for (i=0; i<num_loc_supplied; i++){
            /*generate the best cost for each start location*/
            map = best_cost(loc_supplied[i], map, size);
140     }
        /*assign action to each location*/
        map = assign_direction_go(map, size);
        print_stage3(size, map);

145     /*free pointers and set them to NULL*/
        for(j=0;j<y1;j++){
            free(map[j]);
            map[j] = NULL;
```

```
        }
        free(map);
        map = NULL;
        free(loc_supplied);
        loc_supplied = NULL;
        free(route);
        route = NULL;
        return 0;
    }

    /*****************************************************************/

    /* read and assign the size of the grid
     */
    void
    read_size(dimension_t* size) {
        int x, y;
        scanf("%d%d", &x, &y);
        size->street_num_x = x;
        size->street_num_y = y;
        return;
    }

    /*****************************************************************/

    /* read and assign the map
     */
    map_t **
    create_a_new_map(dimension_t size){
        int x, east, north, west, south;
        char y;
        int i, j, x1 = size.street_num_x ,
        y1 = size.street_num_y;
        map_t **map;
        /*allocate the memory*/
        map = (map_t **) malloc(y1*sizeof(**map));
        for(i=0;i<y1;i++){
            /*allocate the memory*/
            map[i] = (map_t *) malloc(x1*sizeof(*map[i]));
            for (j=0;j<x1;j++){
                scanf("%d%c%d%d%d%d", &x, &y, &east, &north,
                &west, &south);
                map[i][j].location.street_x = x;
                map[i][j].location.street_y = y;
                map[i][j].east  = east;
                map[i][j].north = north;
                map[i][j].west  = west;
                map[i][j].south = south;
                map[i][j].total_cost = ONEWAY;
            }
        }
        return map;
    }

    /*****************************************************************/

    /* read and assign the location supplied
     */
    loc_t*
    read_loc_supplied(int *num_supplied) {
        int x, current_size=1, i=0;
        char y;
        loc_t* A;
        A = (loc_t *) malloc(sizeof(*A));
        while (scanf("%d %c", &x, &y) == 2) {
            /*check if there is enough memory*/
            if (i == current_size) {
                current_size *= 2;
                /*no enought space? double it.*/
                A = realloc(A, current_size * sizeof(*A));
            }
            A[i].street_x = x;
            A[i].street_y = y;
            i++;
```

```
        }
        *num_supplied = i;
225     return A;
    }

    /*****************************************************************/

230 /* count the route can't be used
     */
    void
    count(int cost, int *numoneway, int *allcost){
        if (cost == ONEWAY) {
235         *numoneway += 1;
        } else {
            *allcost += cost;
        }
    }
240
    /*****************************************************************/

    /* assign the info of stage1
     */
245 void
    stage1_data(data_stage1_t* A, dimension_t size,
        map_t** B, loc_t* C, int num_loc_supplied){
        int numoneway = 0, allcost = 0, i, j, x, y,
            E, N, W, S;
250     x = size.street_num_x;
        y = size.street_num_y;
        for (i=0;i<y;i++){
            for (j=0;j<x;j++){
                /*assign the direction cost*/
255             E = B[i][j].east;
                N = B[i][j].north;
                W = B[i][j].west;
                S = B[i][j].south;
                count(E, &numoneway, &allcost);
260             count(N, &numoneway, &allcost);
                count(W, &numoneway, &allcost);
                count(S, &numoneway, &allcost);
            }
        }
265     A->size.street_num_x = size.street_num_x;
        A->size.street_num_y = size.street_num_y;
        A->nintersections = A->size.street_num_x
                          * A->size.street_num_y;
        A->possibilities = A->nintersections * 4;
270     A->useless = numoneway;
        A->total_cost = allcost;
        A->nloc_supplied = num_loc_supplied;
        if (C != NULL) {
            /*find the first and last location*/
275         A->first = C[0];
            A->last = C[num_loc_supplied-1];
        } else {
            exit(EXIT_FAILURE);
        }
280     return;
    }

    /*****************************************************************/

285 /* print out the stage 1
     */
    void
    print_stage1(data_stage1_t A) {
        printf("S1: grid is %d x %d, and has %d intersections\n",
290     A.size.street_num_x, A.size.street_num_y, A.nintersections);
        printf("S1: of %d possibilities, %d of them cannot be used\n",
        A.possibilities, A.useless);
        printf("S1: total cost of remaining possibilities is %d seconds\n",
        A.total_cost);
295     printf("S1: %d grid locations supplied, first one is %d%c, last one is %d%c\n",
        A.nloc_supplied, A.first.street_x, A.first.street_y,
```

```
        A.last.street_x, A.last.street_y);
        printf("\n");
    }

    /*************************************************************/

    /* compare 2 location address, 1 for first input bigger
     *0 for first one smaller
     */
    int
    compare_loc(loc_t current_from,
    loc_t stored_from){
        int x1 = current_from.street_x,
        x2 = stored_from.street_x;
        char y1 = current_from.street_y,
        y2 = stored_from.street_y;
        if (x1 < x2) {
            return 1;
        }else if (x1 == x2 && y1 < y2){
            return 1;
        }
        return 0;
    }

    /*************************************************************/

    /* update the total cost and from infomations
     */
    int
    update_location_info(int now_total, int now, int *total_cost,
        loc_t current_from, loc_t* stored_from){
            int n = now+now_total;
            loc_t loc = *stored_from;
            if (now == ONEWAY){
                return 0;
            }else if (n>*total_cost) {
                return 0;
            }else if (n == *total_cost &&
                !compare_loc(current_from, loc)){
                return 0;
            }else if (n == *total_cost &&
                compare_loc(current_from, loc)){
                    *total_cost = n;
                    *stored_from = current_from;
                    return 1;
            }
            *total_cost = n;
            *stored_from = current_from;
            return 1;
    }

    /*************************************************************/

    /* determine if the locaiton exist, 1 for existing, 0 for not
     */
    int
    is_location_exist(int x, int y, dimension_t size){
        if(x < size.street_num_x&& 0 <= x
        && y < size.street_num_y&& 0 <= y){
            return 1;
        }
        return 0;
    }

    /*************************************************************/

    /* generate the best cost with respect to a start location
     */
    map_t**
    best_cost(loc_t start, map_t **map, dimension_t size){
        int x = start.street_x,
        y = start.street_y - 97,
        x1 = size.street_num_x,
        y1 = size.street_num_y,
```

```
        update = 1,
        all,
        is_update = 1;
        /*set the total cost of the start locaiton to 0
375      *and from to its location*/
        map[y][x].total_cost = 0;
        map[y][x].from.street_x = start.street_x;
        map[y][x].from.street_y = start.street_y;
        /*keep doing the loop until there is no uodate*/
380     while(is_update){
        all = 0;
        for (y=0;y<y1;y++){
            for (x=0;x<x1;x++){
                /*for each location check 4 directions*/
385             if (is_location_exist(x+1, y, size)){
                    update =
                    update_location_info( map[y][x].total_cost,
                        map[y][x].east,
                        &map[y][x+1].total_cost,
390                     map[y][x].location,
                        &map[y][x+1].from);
                    all += update;
                    }
                if (is_location_exist(x, y-1, size)){
395                 update =
                    update_location_info( map[y][x].total_cost,
                        map[y][x].north,
                        &map[y-1][x].total_cost,
                        map[y][x].location,
400                     &map[y-1][x].from);
                    all += update;
                    }
                if (is_location_exist(x-1, y, size)){
                    update =
405                 update_location_info( map[y][x].total_cost,
                        map[y][x].west,
                        &map[y][x-1].total_cost,
                        map[y][x].location,
                        &map[y][x-1].from);
410                 all += update;
                    }
                if (is_location_exist(x, y+1, size)){
                    update =
                    update_location_info( map[y][x].total_cost,
415                     map[y][x].south,
                        &map[y+1][x].total_cost,
                        map[y][x].location,
                        &map[y+1][x].from);
                    all += update;
420                 }
                }
            }
            is_update = all;
        }
425     return map;
    }

    /*************************************************************/

430 /* add the route into an array recursively
     */
    route_t*
    recursive_add(route_t* route, map_t **map,
        loc_t start, loc_t end, int *current_size, int *i){
435     int x = end.street_x,
        y = end.street_y - 97;
        char y1 = end.street_y;
        if (!(x == start.street_x&&y1 == start.street_y)){
            if (*i == *current_size){
440             *current_size *= 2;
                route = (route_t *)realloc(route, *current_size* sizeof(*route));
            }
            /*add each route into an array one by one*/
            route[*i].from.street_x = x;
```

```
445             route[*i].from.street_y = y1;
                route[*i].cost = map[y][x].total_cost;
                *i += 1;
                return recursive_add(route, map, start, map[y][x].from,current_size,i);
            }
450         route[*i].from.street_x = x;
            route[*i].from.street_y = y1;
            route[*i].cost = map[y][x].total_cost;
            /*assign the num of routes in the array*/
            route->num = *i+1;
455         return route;
        }

        /*************************************************************/

460     /* find the route from the end to the start
         */
        route_t*
        navigator_one_route(map_t **map, loc_t start, loc_t end){
            route_t* route;
465         int i=0;
            int current_size = 1;
            route = malloc(sizeof(*route));
            route = recursive_add(route, map, start, end, &current_size,&i);

470         return route;
        }

        /*************************************************************/

475     /* print out one best route
         */
        void
        print_one_route(route_t *route){
            int num = route->num, i;
480         printf("S2: start at grid %d%c, cost of %d\n",
            route[num-1].from.street_x,
            route[num-1].from.street_y, route[num-1].cost);
            for (i = route->num -2;i>=0; i--){
                /*print out the route from the end of the array*/
485             printf("S2:     then to %d%c, cost of %d\n",
                route[i].from.street_x,
                route[i].from.street_y, route[i].cost);
            }
        }
490
        /*************************************************************/

        /* print out the symbol of go west
         */
495     void
        print_gowest(){
            printf(" <<<<");
        }

500     /*************************************************************/

        /* print out the symbol of from west
         */
        void
505     print_fromwest(){
            printf(" >>>>");
        }

        /*************************************************************/
510
        /* print out the symbol of go south
         */
        void
        print_gosouth(){
515         printf("    v");/*eight spaces*/
        }

        /*************************************************************/
```

```
/* print out the symbol of from south
 */
void
print_fromsouth(){
    printf("    ^");/*eight spaces*/
}

/*****************************************************************/

/* print out the stage3
 */
void
print_stage3(dimension_t size, map_t **map){
    int x = size.street_num_x,
    y = size.street_num_y, i, j, k;
    map_t n;
    /*first line of the map*/
    printf("S3:");
    for (i=0;i<x;i++){
        printf("%9d",i);
    }
    /*second line of the table*/
    printf("\nS3:  +----+");
    for (i=1;i<x;i++){
        printf("--------+");
    }
    printf("\n");
    /*middle part of the map*/
    for (i=0;i<y-1;i++){
        /*the horizontal direction*/
        printf("S3: %c |%5d", i+97, map[i][0].total_cost);
        for (j=1;j<x;j++){
            n = map[i][j];
            if(n.gowest == 1){
                print_gowest();
            }else if(n.fromwest == 1){
                print_fromwest();
            }else{
                printf("   ");/*five spaces*/
            }
            printf("%4d",n.total_cost);
        }
        printf("\n");
        /*the vertical direction*/
        for (k=0;k<2;k++){
            printf("S3:  |");
            if (map[i][0].fromsouth == 1){
                printf("  ^");
            }else if (map[i][0].gosouth == 1){
                printf("  v");
            }
            for (j=1;j<x;j++){
                n = map[i][j];
                if(n.gosouth == 1){
                    print_gosouth();
                }else if(n.fromsouth == 1){
                    print_fromsouth();
                }else{
                    printf("    ");/*nine spaces*/
                }
            }
            printf("\n");
        }
    }
    /*last line of the map*/
        printf("S3: %c |%5d", i+97, map[y-1][0].total_cost);
        for (j=1;j<x;j++){
            n = map[y-1][j];
            if(n.gowest == 1){
                print_gowest();
            }else if(n.fromwest == 1){
                print_fromwest();
            }else{
```

```
                    printf("   ");/*five spaces*/
                }
595             printf("%4d",n.total_cost);
            }
            printf("\n");
    }

600 /*****************************************************************/

    /* generate the direction with current location and from location
     */
    int
605 go_direction(loc_t location, loc_t from){
        int x = location.street_x,
        y = location.street_y - 97,
        x1 = from.street_x,
        y1 = from.street_y - 97;
610     if (y == y1 && x == x1 - 1){
            return 1; /*from esat*/
        }else if (y == y1+1 && x == x1){
            return 2; /*from north*/
        }else if (y == y1 && x == x1 + 1){
615         return 3; /*from west*/
        }else if (y == y1-1 && x == x1){
            return 4; /*from south*/
        }
        return 0;
620 }

    /*****************************************************************/

    /* assign the go to 1
625  */
    void
    assign_go(int *go){
        *go = 1;
    }
630
    /*****************************************************************/

    /* assign the go and from direction in map
     */
635 map_t**
    assign_direction_go(map_t **map, dimension_t size){
        int x = size.street_num_x,
        y = size.street_num_y, i, j,k;
        for (i=0;i<y;i++){
640         for (j=0;j<x;j++){
                k=go_direction(map[i][j].location, map[i][j].from);
                if (k == 1){
                    /*if the go direction is from east
                     *assign the go west to 1 of the location is
645                  *on the west of current location*/
                    assign_go(&(map[i][j+1].gowest));
                }else if (k == 2){
                    /*if the go direction is from north
                     *assign the go south to 1 of the location is
650                  *on the north of current location*/
                    assign_go(&(map[i-1][j].gosouth));
                }else if (k == 3){
                    /*if the go direction is from west
                     *assign the from west to 1 of current location*/
655                 assign_go(&(map[i][j].fromwest));
                }else if (k == 4){
                    /*if the go direction is from south
                     *assign the from south to 1 of current location*/
                    assign_go(&(map[i][j].fromsouth));
660             }
            }
        }
        return map;
    }
665
```

```
670 /*Algorithms are fun!*/
```