Α' Εξάμηνο
ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Εργαστήριο – 10η σειρά ασκήσεων.

Κοζάνη, 10 Ιανουαρίου 2008.

Έχοντας γνωρίσει τις εντολές και μεθόδους που επιτρέπουν την ανάπτυξη "δομημένου" κώδικα, μπορούμε να προχωρήσουμε στην ανάπτυξη μιας πιο ολοκληρωμένης εφαρμογής για την επίλυση συγκεκριμένου πρακτικού προβλήματος. Το παράδειγμα αυτής της σειράς προέρχεται σχεδόν αυτούσιο[1] από την εξής πηγή: Herbert Schildt "Artificial Intelligence using C", Osborne Mc Graw-Hill, Berkeley, California, 1987. Ενώ ο όρος "τεχνητή νοημοσύνη" μπορεί να προκαλεί δέος, όπως θα δούμε, η γνώση της C που έχουμε αποκτήσει είναι αρκετή για προχωρημένες όσο και αποτελεσματικές εφαρμογές.

Πρόβλημα

Η αεροπορική εταιρεία XYZ εκτελεί πτήσεις σε διάφορες πόλεις σύμφωνα με τον εξής πίνακα:

Αναχώρηση	***Προορισμός***	***Απόσταση (μίλια)***
New York	Chicago	1000
Chicago	Denver	1000
New York	Toronto	800
New York	Denver	1900
Toronto	Calgary	1500
Toronto	Los Angeles	1800
Toronto	Chicago	500
Denver	Urbana	1000
Denver	Houston	1500
Houston	Los Angeles	1500
Denver	Los Angeles	1000

Ζητείται ένα πρόγραμμα που με δεδομένα το σημείο αναχώρησης και τον προορισμό να δίνει τις πτήσεις που χρειάζονται (με όλους τους ενδιάμεσους σταθμούς). Στη συνέχεια το πρόγραμμα θα βελτιωθεί ώστε να βρίσκει τις καλύτερες λύσεις με βάση κάποιο κριτήριο όπως ελάχιστη συνολική απόσταση ή ελάχιστος αριθμός ενδιάμεσων σταθμών.
Σαν παράδειγμα να χρησιμοποιηθεί η εύρεση πτήσεων με σημείο αναχώρησης τη Νέα Υόρκη και προορισμό Los Angeles.

Θεωρητικό πλαίσιο

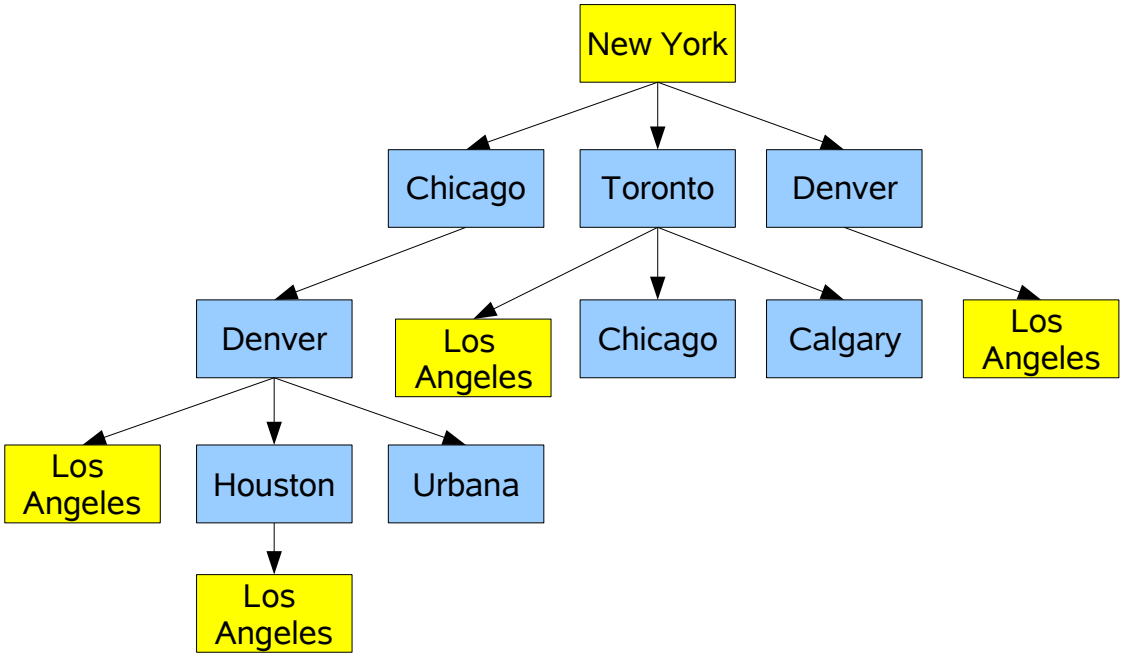
Τα προβλήματα αυτού του είδους μπορεί να είναι πιο δύσκολα απ' όσο φαίνεται με την πρώτη ματιά. Μια σίγουρη λύση είναι η εξαντλητική αναζήτηση δηλαδή η εύρεση όλων των δυνατών διαδρομών και η επιλογή της καλύτερης με κάποιο κριτήριο. Αυτό δεν είναι πρακτικό όταν έχουμε

1 Έχουν γίνει μικροαλλαγές για να είναι σύμφωνο με τους κανόνες της ANSI C.

χιλιάδες δεδομένα. Γι' αυτό χρησιμοποιούνται διάφορες τεχνικές που βρίσκουν γρήγορα μία λύση ακόμη και αν δεν είναι καλύτερη. Οι κυριότερες είναι:

- αναζήτηση σε βάθος (depth-first)
- αναζήτηση κατά πλάτος (breadth-first)

Για να εξηγήσουμε αυτές τις έννοιες απεικονίζουμε γραφικά το συγκεκριμένο πρόβλημα και τις λύσεις του (που είναι εύκολο να βρεθούν λόγω του μικρού αριθμού δεδομένων), ως δέντρο:



Αναζήτηση σε βάθος σημαίνει να ακολουθήσουμε με τη σειρά κάθε δυνατή διαδρομή μέχρι το τέλος, μέχρι να βρούμε κάποια που μας κάνει. Αναζήτηση κατά πλάτος σημαίνει το αντίθετο: πρώτα ψάχνουμε όλους τους κόμβους ενός επιπέδου πριν προχωρήσουμε στο βαθύτερο μέχρι να βρούμε μία λύση που μας κάνει. Γενικά, αυτές οι μέθοδοι είναι "συμπληρωματικές" και κατά μέσο όρο έχουν παρόμοια απόδοση.

Αυτές οι τεχνικές μπορούν μετά να συνδυαστούν με "τεχνάσματα" ή "ευρετικές" μεθόδους (heuristics) για την εντοπισμό σχετικά καλύτερων λύσεων. Τα κυριότερα τεχνάσματα είναι:

- μέθοδος ανάβασης (hill climbing)
- μέθοδος ελάχιστου κόστους (least cost)

Κατά τη μέθοδο ανάβασης επιλέγουμε ως επόμενο βήμα τον κόμβο που πιστεύουμε (με βάση κάποιο κριτήριο) ότι είναι πλησιέστερα στο στόχο μας. Αντίθετα, κατά τη μέθοδο ελάχιστου κόστους επιλέγουμε ως επόμενο βήμα αυτό που ελαχιστοποιεί κάποιο ποσοτικό κριτήριο "προσπάθειας" που έχουμε ορίσει.

Στη συνέχεια αναπτύσσουμε το πρόγραμμα που υλοποιεί τη μέθοδο αναζήτησης σε βάθος. Μετά, με βάση αυτό, θα εισάγουμε τροποποιήσεις για να υλοποιήσουμε τις άλλες μεθόδους.

Μέθοδος αναζήτησης σε βάθος: ανάλυση και σχεδιασμός

Θα χρειαστούμε ένα είδος βάσης δεδομένων για την πληροφορία κάθε πτήσης (αναχώρηση, προορισμός, απόσταση). Επίσης, θα προσθέσουμε μία "σημαία" για να σημειώνουμε κάθε εγγραφή της βάσης αυτής που έχει ήδη ερευνηθεί κατά την αναζήτηση. Η βάση αυτή μπορεί να υλοποιηθεί ως εξής:

```
#define MAX 100

struct FL {
    char from[20];
    char to[20];
```

```
    int distance;
    char skip;
};

struct FL flight[MAX];

int f_pos = 0;
int find_pos = 0;
```

Εδώ ορίσαμε μία δομή δεδομένων για τις εγγραφές των πτήσεων και με τη βοήθεια αυτής έναν πίνακα τέτοιων εγγραφών, μεγέθους MAX = 100. Επίσης, η καθολική μεταβλητή f_pos μας λέει πόσες εγγραφές έχουν καταχωρηθεί και η μεταβλητή find_pos χρησιμεύει κατά την αναζήτηση μέσα από τη βάση δεδομένων που ορίσαμε.

Το κύριο πρόγραμμα θα έχει γενικά μία δομή της εξής μορφής:
- Δημιουργία βάσης δεδομένων
- Εισαγωγή ερωτήματος (αναχώρηση, προορισμός)
- Εύρεση λύσης
- Εμφάνιση λύσης.

Κάθε ένα από τα παραπάνω σημεία μπορεί να είναι μία ξεχωριστή συνάρτηση ή ομάδα εντολών. Θα ορίσουμε τις εξής συναρτήσεις:
- setup, για τη δημιουργία της βάσης δεδομένων
- ask, για την εισαγωγή του ερωτήματος
- isflight, για την εύρεση λύσης
- route, για την εμφάνιση της λύσης

οι οποίες θα καλούνται μέσα από τη main. Στη συνέχεια αναλύουμε και υλοποιούμε αυτές τις συναρτήσεις υλοποιώντας σταδιακά το ίδιο το πρόγραμμα.

Μέθοδος αναζήτησης σε βάθος: περαιτέρω ανάλυση και υλοποίηση
setup
Αυτή η συνάρτηση γενικά μπορεί να διαβάζει τα δρομολόγια από ένα αρχείο, καθώς και να περιέχει και ένα διάλογο με το χρήστη για εισαγωγή εγγραφών. Εδώ θα περιοριστούμε στην εισαγωγή των δρομολογίων του πίνακα που δόθηκε στην αρχή μέσα από το ίδιο το πρόγραμμα. Η setup θα καλεί μία άλλη συνάρτηση με όνομα assert_flight για την εισαγωγή της κάθε εγγραφής. Αυτές οι δύο συναρτήσεις μπορούνε τότε να πάρουν την εξής μορφή:

```
void setup() {
    assert_flight("New York", "Chicago", 1000);
    assert_flight("Chicago", "Denver", 1000);
    assert_flight("New York", "Toronto", 800);
    assert_flight("New York", "Denver", 1900);
    assert_flight("Toronto", "Calgary", 1500);
    assert_flight("Toronto", "Los Angeles", 1800);
    assert_flight("Toronto", "Chicago", 500);
    assert_flight("Denver", "Urbana", 1000);
    assert_flight("Denver", "Houston", 1500);
    assert_flight("Houston", "Los Angeles", 1500);
```

```
    assert_flight("Denver", "Los Angeles", 1000);
}

void assert_flight(char *from, char *to, int dist) {
    if(f_pos < MAX) {
        strcpy(flight[f_pos].from, from);
        strcpy(flight[f_pos].to, to);
        flight[f_pos].distance = dist;
        flight[f_pos].skip = 0;
        f_pos++;
    } else
        printf("Flight database full\n");
}
```

ask

Αυτή διαβάζει δύο αλφαριθμητικά from (αναχώρηση) και to (προορισμός) που θα είναι τα ορίσματα της inflight. Μπορεί να έχει τη μορφή ενός απλού διαλόγου με τον χρήστη:

```
void ask(char *from, char *to) {
    printf("from? ");
    gets(from);
    printf("to? ");
    gets(to);
}
```

isflight

Αυτή η συνάρτηση υλοποιεί τη μέθοδο αναζήτησης. Η inflight μπορεί να παίρνει ως ορίσματα δύο αλφαριθμητικά from (αναχώρηση) και to (προορισμός) και να λειτουργεί με τη βοήθεια μιας στοίβας όπου αποθηκεύονται οι κόμβοι κάθε διαδρομής που ελέγχεται. Αυτό θα γίνεται ως εξής:

1. Πρώτα, ελέγχεται αν υπάρχει πτήση μεταξύ from και to.
2. Αν υπάρχει, τότε βρέθηκε η λύση, η συνάρτηση ωθεί τη σύνδεση στη στοίβα και επιστρέφει.
3. Αλλιώς, ελέγχεται κατά πόσον υπάρχει σύνδεση της from προς οπουδήποτε.
4. Αν υπάρχει σύνδεση, ωθείται στη στοίβα και η isflight καλείται αναδρομικά.
5. Αν δεν υπάρχει, τότε διώχνεται ο τελευταίος κόμβος από τη στοίβα και η isflight καλείται και πάλι αναδρομικά. Εδώ, η “σημαία” skip βοηθά ώστε να μη ξαναδοκιμάζονται οι ίδιες συνδέσεις πολλές φορές.
6. Αυτό συνεχίζεται μέχρι τη λύση ή την εξάντληση των διαδρομών.

Για να κάνουμε τα παραπάνω θα χρειαστούν: η καθολική μεταβλητή tos για την κορυφή της στοίβας, ο πίνακας bt_stack που θα υλοποιεί τη στοίβα και η δομή δεδομένων struct stack που περιγράφει τα στοιχεία της στοίβας. Όλα αυτά θα τοποθετηθούν στην αρχή του αρχείου και θα έχουν την εξής μορφή:

```
int tos = 0;
struct stack {
    char from[20];
    char to[20];
    int dist;
```

```
};
struct stack bt_stack[MAX];
```

Η ανάπτυξή της isflight θα διευκολυνθεί ακόμη περισσότερο αν χρησιμοποιηθούν ορισμένες βοηθητικές συναρτήσεις. Ορίζουμε και υλοποιούμε τις εξής συναρτήσεις:

- **match**. Αυτή προσδιορίζει αν υπάρχει πτήση μεταξύ δύο πόλεων και επιστρέφει την απόστασή τους ή 0 αν δεν υπάρχει πτήση. Η μορφή της είναι η παρακάτω:

```
int match(char *from, char *to) {
    register int t;
    int retval = 0;

    for (t = f_pos-1; t > -1; t--)
        if(!strcmp(flight[t].from, from)
        && !strcmp(flight[t].to, to)
            retval = flight[t].distance;

    return (retval);
}
```

- **find**. Αυτή παίρνει ως είσοδο μία πόλη και επιστρέφει το όνομα της πρώτης σύνδεσης με την πόλη αυτή και την απόσταση, αλλιώς επιστρέφει 0. Εδώ ελέγχονται μόνο οι εγγραφές οι οποίες έχουν το πεδίο skip ίσο με 0. Όταν ελεγχθούν, τότε το skip θα γίνεται 1 για να μην ξαναγίνει έλεγχος για αυτές. Η μορφή της find θα είναι:

```
int find(char *from, char *anywhere) {
    int retval = 0;

    find_pos = 0;
    while(find_pos < f_pos) {
        if(!strcmp(flight[find_pos].from, from)
        && !flight[find_pos].skip) {
            strcpy(anywhere, flight[find_pos].to);
            flight[find_pos].skip = 1;
            retval = flight[find_pos].distance;
        }
        find_pos++;
    }
    return (retval);
}
```

- **push** και **pop**. Αυτές παίζουν τον πιο κεντρικό ρόλο στη διαδικασία αναζήτησης γιατί "θέτουν σε λειτουργία" τη στοίβα. Η μορφή τους είναι η εξής:

```
void push(char *from, char *to, int dist) {
    if(tos < MAX) {
        strcpy(bt_stack[tos].from, from);
        strcpy(bt_stack[tos].to, to);
        bt_stack[tos].dist = dist;
        tos++;
    } else
        printf("Stack full.\n");
}

void pop(char *from, char *to, int *dist) {
    if(tos < 0) {
        tos--;
        strcpy(from, bt_stack[tos].from);
        strcpy(to, bt_stack[tos].to);
        *dist = bt_stack[tos].dist;
    } else
        printf("Stack underflow./n");
}
```

Με τις παραπάνω βοηθητικές συναρτήσεις, η isflight μπορεί να πάρει την εξής μορφή:

```
void isflight(char *from, char *to) {
    int d, dist;
    char anywhere[20];

    if(d=match(from, to))
        push(from, to, d);
    else if(dist = find(from, anywhere)) {
        push(from, to, dist);
        isflight(anywhere, to);
    } else if(tos > 0) {
        pop(from, to, &dist);
        isflight(from, to);
    }
}
```

Αν βρεθεί διαδρομή, τότε η στοίβα περιέχει τη λύση. Αν η στοίβα είναι άδεια, τότε δεν έχει βρεθεί διαδρομή.

route
Αυτή η συνάρτηση “αδειάζει” τη στοίβα και εμφανίζει τα περιεχόμενά της που δεν είναι παρά οι κόμβοι που απαρτίζουν τη ζητούμενη διαδρομή. Αυτό γίνεται με τον εξής τρόπο:

```
void route(char *to) {
    int dist, t;

    dist = 0;
    t = 0;
    while(t < tos) {
        printf("%s to ", bt_stack[t].from);
        dist += bt_stack[t].dist;
        t++;
    }
    printf("%s\n", to);
    printf("The distance is %d\n", dist);
}
```

main

Χρησιμοποιώντας τις παραπάνω συναρτήσεις, η main μπορεί να γραφεί στην παρακάτω απλή μορφή:

```
int main() {
    char from[20], to[20];

    setup();
    ask(from, to);
    isflight(from, to);
    route(to);
}
```

Πρόγραμμα p10-1 ***Μέθοδος Αναζήτησης Σε Βάθος***

Δημιουργήστε ένα project, ονομάστε το p10 και εισάγετε σε ένα αρχείο τις παραπάνω συναρτήσεις με την εξής σειρά (με γαλάζιο οι γραμμές που δίνονται από το περιβάλλον εκτέλεσης και με κόκκινο αυτές που προσθέτουμε εμείς):

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

#define MAX 100

struct FL {
    char from[20];
    char to[20];
    int distance;
    char skip;
};

struct FL flight[MAX];
```

```
int f_pos = 0;
int find_pos = 0;

int tos = 0;
struct stack {
    char from[20];
    char to[20];
    int dist;
};
struct stack bt_stack[MAX];

void assert_flight(char *from, char *to, int dist) {
    if(f_pos < MAX) {
        strcpy(flight[f_pos].from, from);
        strcpy(flight[f_pos].to, to);
        flight[f_pos].distance = dist;
        flight[f_pos].skip = 0;
        f_pos++;
    } else
        printf("Flight database full\n");
}

void setup() {
    assert_flight("New York", "Chicago", 1000);
    assert_flight("Chicago", "Denver", 1000);
    assert_flight("New York", "Toronto", 800);
    assert_flight("New York", "Denver", 1900);
    assert_flight("Toronto", "Calgary", 1500);
    assert_flight("Toronto", "Los Angeles", 1800);
    assert_flight("Toronto", "Chicago", 500);
    assert_flight("Denver", "Urbana", 1000);
    assert_flight("Denver", "Houston", 1500);
    assert_flight("Houston", "Los Angeles", 1500);
    assert_flight("Denver", "Los Angeles", 1000);
}

void ask(char *from, char *to) {
    printf("from? ");
    gets(from);
    printf("to? ");
    gets(to);
}
```

```
int match(char *from, char *to) {
    register int t;
    int retval = 0;

    for (t = f_pos-1; t > -1; t--)
        if(!strcmp(flight[t].from, from)
        && !strcmp(flight[t].to, to))
            retval = flight[t].distance;

    return (retval);
}

int findDepthFirst(char *from, char *anywhere) {
    int retval = 0;

    find_pos = 0;
    while(find_pos < f_pos) {
        if(!strcmp(flight[find_pos].from, from)
        && !flight[find_pos].skip) {
            strcpy(anywhere, flight[find_pos].to);
            flight[find_pos].skip = 1;
            retval = flight[find_pos].distance;
        }
        find_pos++;
    }
    return (retval);
}

void push(char *from, char *to, int dist) {
    if(tos < MAX) {
        strcpy(bt_stack[tos].from, from);
        strcpy(bt_stack[tos].to, to);
        bt_stack[tos].dist = dist;
        tos++;
    } else
        printf("Stack full.\n");
}

void pop(char *from, char *to, int *dist) {
    if(tos < 0) {
        tos--;
```

```
        strcpy(from, bt_stack[tos].from);
        strcpy(to, bt_stack[tos].to);
        *dist = bt_stack[tos].dist;
    } else
        printf("Stack underflow./n");
}

int (*find)() = findDepthFirst;

void isflightDepthFirst(char *from, char *to) {
    int d, dist;
    char anywhere[20];

    if(d=match(from, to))
        push(from, to, d);
    else if(dist = find(from, anywhere)) {
        push(from, to, dist);
        isflightDepthFirst(anywhere, to);
    } else if(tos > 0) {
        pop(from, to, &dist);
        isflightDepthFirst(from, to);
    }
}

void route(char *to) {
    int dist, t;

    dist = 0;
    t = 0;
    while(t < tos) {
        printf("%s to ", bt_stack[t].from);
        dist += bt_stack[t].dist;
        t++;
    }
    printf("%s\n", to);
    printf("The distance is %d\n", dist);
}

void  (*isflight)() = isflightDepthFirst;

int main(char argc, char *argv[])
{
```

```
    char from[20], to[20];

    setup();
    ask(from, to);
    isflight(from, to);
    route(to);
    system("PAUSE");
    return 0;
}
```

Στο παραπάνω αρχείο υπάρχει μία μικρή προσθήκη σε σχέση με όσα παρουσιάσαμε προηγουμένως, σε σχέση με την ανάλυση και υλοποίηση της αναζήτησης σε βάθος. Συγκεκριμένα:

- μετονομάσαμε τη συνάρτηση find σε findDepthFirst
- μετά από τον ορισμό της pop προσθέσαμε τη γραμμή
  `int (*find)() = findDepthFirst;`
  δηλαδή, το όνομα find το μετατρέψαμε σε δείκτη στη συνάρτηση findDepthFirst,
- μετονομάσαμε τη συνάρτηση isflight σε isflightDepthFirst,
- πριν από τη main προσθέσαμε τη γραμμή
  `void (*isflight)() = isflightDepthFirst;`
  δηλαδή, το όνομα isflight το μετατρέψαμε σε δείκτη στη συνάρτηση isflightDepthFirst.

Οι λόγοι αυτών των αλλαγών θα γίνουν φανεροί στη συνέχεια.
Μεταγλωττίστε και εκτελέστε.

Μέθοδος αναζήτησης κατά πλάτος: ανάλυση, σχεδιασμός και υλοποίηση.
Τα περισσότερα από όσα είπαμε πριν ισχύουν και για τη μέθοδο αναζήτησης κατά πλάτος. Η μόνη αλλαγή που πρέπει να γίνει είναι στη συνάρτησης isflight που θα πάρει την παρακάτω μορφή:

```
void isflight(char *from, char *to) {
    int d, dist;
    char anywhere[20];

    while(dist=find(from, anywhere)) {
        if(d=match(anywhere, to)) {
            push(from, anywhere, dist);
            push(anywhere, to, d);
            return;
        }
    }
    if(dist = find(from, anywhere)) {
        push(from, to, dist);
        isflight(anywhere, to);
    } else if(tos > 0) {
        pop(from, to, &dist);
        isflight(from, to);
    }
```

```
}
```

Πρόγραμμα p10-2 ***Μέθοδος Αναζήτησης Σε Βάθος***
Στο προηγούμενο πρόγραμμα κάνουμε τις εξής προσθήκες και αλλαγές:
Μεταξύ της inflightDepthFirst και της route προσθέτουμε την παραπάνω τροποποιημένη inflight με το όνομα inflightBreadthFirst:

```
void isflightBreadthFirst(char *from, char *to) {
    int d, dist;
    char anywhere[20];

    while(dist=find(from, anywhere)) {
        if(d=match(anywhere, to)) {
            push(from, anywhere, dist);
            push(anywhere, to, d);
            return;
        }
    }
    if(dist = find(from, anywhere)) {
        push(from, to, dist);
        isflightBreadthFirst(anywhere, to);
    } else if(tos > 0) {
        pop(from, to, &dist);
        isflightBreadthFirst(from, to);
    }
}
```

Τροποποιούμε τη γραμμή με το δείκτη συνάρτησης inflight ακριβώς πριν από τη main ώστε να δείχνει στη συνάρτηση που προσθέσαμε:

```
void  (*isflight)() = isflightBreadthFirst;
```

Μεταγλωττίστε και εκτελέστε.

Ευρετικές Τεχνικές
Και εδώ επίσης, η γενικότερη λογική παραμένει ίδια. Οι μόνες αλλαγές που χρειάζονται αφορούν τη συνάρτηση find.
Στην περίπτωση της **μεθόδου της ανάβασης**, ψάχνουμε ολόκληρη τη βάση δεδομένων για να βρούμε τη σύνδεση που είναι πιο μακριά από την πόλη αναχώρησης. Αυτό το κάνουμε επειδή θεωρούμε ότι έτσι έχουμε περισσότερες πιθανότητες να φτάσουμε με τις λιγότερες προσπάθειες πιο κοντά στον προορισμό μας.
Αντίθετα, στην περίπτωση της **μεθόδου ελάχιστου κόστους**, σε κάθε βήμα ψάχνουμε να βρούμε τη σύνδεση με τη μικρότερη απόσταση. Αυτό το κάνουμε επειδή θεωρούμε ότι έτσι αυξάνονται οι πιθανότητες η τελική λύση να έχει μικρότερο μήκος, δηλαδή το ταξίδι να είναι πιο σύντομο.

Πρόγραμμα p10-3 ***Μέθοδος Ανάβασης***
Στο προηγούμενο πρόγραμμα κάνουμε τις εξής προσθήκες και αλλαγές:
Επαναφέρουμε τη χρήση της αρχικής inflight γράφοντας και πάλι πριν από τη main:

```
void  (*isflight)() = isflightDepthFirst;
```

Αμέσως μετά από τον ορισμό της findDepthFirst προσθέτουμε τον παρακάτω ορισμό:

```
int findHillClimb(char *from, char *anywhere) {
    int pos, dist, retval = 0;

    pos = dist = 0;
    find_pos = 0;

    while(find_pos < f_pos) {
        if(!strcmp(flight[find_pos].from, from)
        && !flight[find_pos].skip) {
            if(flight[find_pos].distance > dist) {
                pos = find_pos;
                dist = flight[find_pos].distance;
            }
       }
       find_pos++;
    }
    if(pos) {
        strcpy(anywhere, flight[pos].to);
        flight[pos].skip = 1;
        retval = flight[pos].distance;
    }
    return (retval);
}
```

Πηγαίνουμε εκεί που ορίζεται ο δείκτης find και αλλάζουμε:

```
int (*find)() = findHillClimb;
```

Μεταγλωττίστε και εκτελέστε.

Πρόγραμμα p10-4 *Μέθοδος Ελάχιστου Κόστους*

Αμέσως μετά από τον ορισμό της findClimbHill προσθέτουμε τον παρακάτω ορισμό, όπου με κόκκινο χρώμα φαίνονται τα σημεία στα οποία η νέα συνάρτηση διαφέρει από την findClimbHill:

```
int findLeastCost(char *from, char *anywhere) {
    int pos, dist, retval = 0;

    pos = 0;
    dist = 32000; /* larger than the longest route */
    find_pos = 0;
```

```
    while(find_pos < f_pos) {
        if(!strcmp(flight[find_pos].from, from)
        && !flight[find_pos].skip) {
            if(flight[find_pos].distance < dist) {
                pos = find_pos;
                dist = flight[find_pos].distance;
            }
        }
        find_pos++;
    }
    if(pos) {
        strcpy(anywhere, flight[pos].to);
        flight[pos].skip = 1;
        retval = flight[pos].distance;
    }
    return (retval);
}
```

Πηγαίνουμε εκεί που ορίζεται ο δείκτης find και αλλάζουμε:

```
int (*find)() = findLeastCost;
```

Μεταγλωττίστε και εκτελέστε.