Ανάπτυξη Λογισμικού
για Πληροφοριακά Συστήματα
ΠΑΡΑΔΟΤΕΟ 1

Άννα Αλεξάνδρου: 1115201700202
Ελένη-Πολυξένη Παπαδάκη: 1115201700116
Μαρία Μαυράκη: 1115201700081

Μεταγλώττιση και εκτέλεση Προγράμματος:

**Η μεταγλώττιση και εκτέλεση του κώδικα πρέπει να γίνεται στον ίδιο φάκελο με τον φάκελο camera_specs (και με χρήση του large datasetW, επίσης στον ίδιο φάκελο).*

**Ο φάκελος camera_specs ΔΕΝ περιέχεται.*

Για μεταγλώττιση: επισυνάπτεται αρχείο Makefile

- make project1: για μεταγλώττιση του κώδικα
- make test_project1 : για μεταγλώττιση του κώδικα για το Software Testing

Για εκτέλεση:

- ./project1
- ./test_project1 ή ./test_project1 *όνομα test_συνάρτησης*

Εκκαθάριση:

make clean: διαγράφει τα εκτελέσιμα, τα object files και το αρχείο εξόδου

Υλοποίηση:

1) Γίνεται χρήση hashTable για την αποθήκευση και την εύρεση των specs.
2) Δημιουργούμε pointers από την κάθε κλίκα πίσω στο bucket (υλοποίηση της 1-1 αντιστοιχίας μεταξύ των κορυφών του γράφου και των specs των προϊόντων).
3) Για την υλοποίηση του προγράμματος χρησιμοποιούμε 2 λίστες, την απλή list και την list2. Κάθε κλίκα είναι και μία απλή list. Κάθε στοιχείο της list2 δείχνει σε μία κλίκα (δηλαδή στην αρχή μιας απλή list).
Η list2 έχει φτιαχτεί για να μπορούμε να έχουμε γρήγορη πρόσβαση (κατά την εκτύπωση του output) στις κλίκες χωρίς να χρειάζεται να διατρέχουμε το hashTable.

Παραδοχές:

1) Μέσα στα buckets του hashtable αποθηκεύουμε ολόκληρα τα specs ενώ μέσα στις κλίκες (δηλαδή τις απλές list) μόνο τα spec_ids και δείκτες πίσω στα specs.
2) Υποθέτουμε ότι για κάθε κατηγορία spec που διαβάζουμε οι χαρακτήρες είναι λιγότεροι από 51.000 και τα spec_ids περιέχουν λιγότερους από 50 χαρακτήρες.

Αρχεία:

readX.h:

Εδώ ορίζονται τα παρακάτω structs για την μεταφορά των περιεχομένων του κάθε αρχείου spec:

```
struct Spec {

  char *spec_id;

  int num_of_fields;
```

```
  struct Pair* fields;
};


struct Pair {
  char* category;
  char* details;
};
```

Κάθε αρχείο αποθηκεύεται στο σύστημα διατηρώντας το id του στη θέση *spec_id*, των αριθμό των κατηγοριών που περιλαμβάνει ως *num_of_fields*, καθώς και ένα πίνακα από τα ζευγάρια ‘τίτλος κατηγορίας’ και ‘περιεχόμενο κατηγορίας’ ως *category* και *details* αντίστοιχα.

readx.c:

Υλοποίηση των παρακάτω συναρτήσεων:

```
int read_datasetX(int HashtableNumOfEntries, struct Entry*
HashTable, int bucketSize);
```

Διασχίζει όλους τους φακέλους και τα αρχεία τους, δεσμεύει χώρο και μεταφέρει το περιεχόμενο των αρχείων σε struct Spec (τα οποία αποθηκεύονται σε ένα hashtable).

Καλεί την set_spec(…), η οποία αναλαμβάνει τη σωστή μεταφορά των περιεχομένων στο struct Spec και την insertInHashTable() για την τοποθέτηση αυτών στο hashtable.

```
void set_pair(struct Spec *spec,int pos, char *first_part, char
*second_part);
```

Δεσμεύει χώρο και αποθηκεύει τον τίτλο κατηγορίας και το περιεχόμενο, στην θέση *pos* του πίνακα του *Spec.*

```
void set_spec(struct Spec *spec, FILE *spec_file, char*
dir_name, char* file_name);
```

Διατρέχει το αρχείο στο οποίο δείχνει ο δείκτης `spec_file` και λαμβάνοντας υπόψη την μορφοποίησή του, εξαγάγει τις πληροφορίες που υπάρχουν για το spec.

Αποθηκεύει τις πληροφορίες αυτές στο struct Spec. Καλεί την extract_id για την καταχώρηση του spec_id.

```
char *extract_id(char * dir_name, char *file_name);
```

Δέχεται το όνομα του αρχείου και το όνομα του φακέλου στο οποίο βρίσκεται. Χρησιμοποιεί τις δύο αυτές πληροφορίες για τον υπολογισμό της τιμής του spec_id και επιστρέφει την τιμή αυτή.

```
void print_pair(struct Pair pair);

void print_spec(struct Spec spec);
```

Βοηθητικές Συναρτήσεις για εκτύπωση των περιεχομένων των struct Spec.

result.c & result.h

Υλοποίηση της εκτύπωσης του αποτελέσματος

```
void print_results( struct FirstId *start )
```

Δέχεται ως όρισμα μια λίστα που διατηρεί δείκτες στην αρχή της κάθε κλίκας.

Εκτυπώνει τα ζεύγη των specs που αναφέρονται στο ίδιο αντικείμενο/προϊόν.

Αρχικά,

Δημιουργεί ένα νέο αρχείο output.csv.

Διαπερνά την λίστα start και για κάθε κλίκα που βρίσκει, εκτυπώνει στο νέο αρχείο, τα περιεχόμενά της σε ζεύγη.

readW.c & readW.h

Υλοποίηση της ανάγνωσης του dataset W (αρχείο csv)

```
int readDatasetW( FILE* datasetW, char* left, char* right)
```

Δέχεται ως όρισμα ένα csv αρχείο και δύο δεσμευμένα strings μέσω των οποίων επιστρέφει κάθε ζεύγος από spec_ids που διαβάζει στο dataset.
Αρχικά ελέγχει αν γίνεται σωστά η ανάγνωση του αρχείου, διαφορετικά επιστρέφει κωδικό λάθους (ENOENT).
Έπειτα, διαβάζει το αρχείο γραμμή-γραμμή έως ότου φτάσει στο τέλος του, οπότε ενημερώνει επιστρέφοντας κωδικό EOF.
Κατά την ανάγνωση, ελέγχει για κάθε γραμμή που διαβάζει την τιμή του πεδίου label. Αν label=0 αγνοεί το ζεύγος που μόλις διάβασε και προχωράει στην επόμενη γραμμή του αρχείου. Αν label=1 σταματάει την ανάγνωση του αρχείου και επιστρέφει στη συνάρτηση που την κάλεσε το ζεύγος που μόλις διάβασε. Όταν η συνάρτηση ξανακληθεί για να δώσει το επόμενο ζεύγος, συνεχίζει την ανάγνωση του αρχείου από εκεί που σταμάτησε την τελευταία φορά.

```
void split_specid( char* specid_parts, char* left, char*
specid)
```

Δέχεται ως όρισμα το id ενός spec (specid) και επιστρέφει μέσω του πρώτου ορίσματος έναν πίνακα που περιέχει τα τμήματα στα οποία σπάει το specid σύμφωνα με τον εξής χωρισμό:

«σπάσε το specid στα σημεία που υπάρχουν //» (π.χ. για το specid «www.ebay.com//100» θα επιστραφεί ένας πίνακας δύο στοιχείων που έχει ως πρώτο στοιχείο το «www.ebay.com» και ως δεύτερο το «100»).

```
int greaterSpecId( char* left_spec_id, char* right_spec_id)
```

Δέχεται ως ορίσματα δύο spec_ids. Επιστρέφει είτε την τιμή LEFT είτε την τιμή RIGHT προκειμένου να υποδείξει ποιο spec_id από τα δύο είναι μεγαλύτερο ως προς την αξία του.
Για να γίνει η σύγκριση, σπάει το κάθε spec_id καλώντας την split_specid(). Συγκρίνει αρχικά τα πρώτα τμήματα υλοποιώντας αλφαριθμητική σύγκριση με την strcmp(). Αν τα δύο strings είναι διαφορετικά μεταξύ τους, βάσει του αποτελέσματος της strcmp() επιστρέφει την κατάλληλη ένδειξη για το μεγαλύτερο από τα δύο. Αλλιώς, (αν τα δύο strings είναι ίδια) η σύγκριση προχωράει στο επόμενο πεδίο. Εκεί τα strings μετατρέπονται σε ακεραίους ώστε να γίνει αριθμητική σύγκριση των τιμών τους και να υποδειχθεί έτσι το μεγαλύτερο string. Σημείωση: σε περίπτωση ισότητας γίνεται η παραδοχή να επιστραφεί RIGHT υποδεικνύοντας το δεξί string ως μεγαλύτερο.

```
void init_specid_parts ( char** s)
void free_specid_parts ( char** s)
```

Πρόκειται για δύο βοηθητικές συναρτήσεις. Η πρώτη καλείται από την split_specid() και παίρνει σαν όρισμα τον πίνακα με τα τμήματα του specid μόλις αυτός δημιουργηθεί ώστε να αρχικοποιήσει τα πεδία του σε NULL. Η δεύτερη καλείται από την greaterSpecId(), παίρνει σαν όρισμα τον ίδιο πίνακα (όταν αυτός περιέχει πλέον strings) και είναι υπεύθυνη για τη σωστή αποδέσμευση της μνήμης.

ids_list.h

Εδώ ορίζονται τα παρακάτω structs που αναπαριστούν τους κόμβους για τα δύο είδη λιστών που υλοποιούνται στο πρόγραμμα:

```
struct ListId {
char *id;
struct ListId *next_list_id;
struct BucketSpec *mat_spec;
};
```

```
struct FirstId {
struct ListId *first_id;
struct FirstId *next_id;
};
```

Η πρώτη λίστα, που περιέχει κόμβους τύπου ListId, αναπαριστά μία κλίκα από specs που ταιριάζουν μεταξύ τους. Κάθε κόμβος περιέχει το id του spec που ανήκει στην κλίκα, ένα δείκτη στο επόμενο στοιχείο της κλίκας και ένα δείκτη στη θέση που βρίσκεται το συγκεκριμένο spec μέσα στη δομή του bucket.

Η δεύτερη λίστα, που περιέχει κόμβους τύπου FirstId, κρατάει ουσιαστικά δείκτες στο πρώτο στοιχείο κάθε κλίκας, ώστε κατά την εκτύπωση του output να μπορούμε να έχουμε πρόσβαση στην αρχή οποιασδήποτε κλίκας χωρίς να διατρέξουμε το hashtable. Κάθε κόμβος περιέχει τον δείκτη στον πρώτο κόμβο (τύπου ListId) της κλίκας και έναν δείκτη στο επόμενό του κόμβο.

ids_list.c

```
void addIdInList(struct ListId* origin_matching_list,struct
ListId* dest_matching_list)
```

Με αυτή την συνάρτηση ενώνουμε 2 κλίκες/απλές λίστες, έτσι ώστε στο τέλος της destination list να συνδέσουμε την origin list.

```
void addIdInList2(struct ListId* new_list_id,struct FirstId**
first_ids_list)
```

Με αυτή την συνάρτηση προσθέτουμε στη list2 έναν καινούριο pointer σε κλίκα. Η εισαγωγή γίνεται στην αρχή της λίστας.

```
void deleteIdFromList2(struct ListId* removing_id,struct
FirstId** firstId)
```

Η συγκεκριμένη συνάρτηση λαμβάνει σαν όρισμα το ListId στο οποίο δείχνει το FirstId που θα πρέπει να σβήσει και τον pointer που δείχνει

στην αρχή της λίστας 2. Διατρέχει τη λίστα 2 και σβήνει το συγκεκριμένο FirstId.

hashTable.h :

Δομές που χρησιμοποιούνται για το hash table:

```
struct BucketSpec{
	struct Spec* spec;  //pointer to a spec
	struct ListId* matching_ids;  //list of matching specs
= clique of spec
};

struct Bucket{
	struct BucketSpec* bucket_specs;  //content of bucket
	struct Bucket* nextBucket;  //pointer to next bucket
(in case of overflow)
	int isFull;   //current number of BucketSpecs inside
bucket => place in bucket to add a new spec
};

struct Entry{
	int key;
	struct Bucket* bucket;
};
```

hashTable.c :

Δημιουργία hash table:

Ορίζουμε ως σταθερές τον αριθμό των entries του hash table και το μέγεθος των buckets.

```
#define NUM_OF_ENTRIES 20
#define BUCKET_SIZE 100
```

Χρησιμοποιούνται οι συναρτήσεις initializeHashTable και initialiseBucket για την αρχικοποίηση του hash table σύμφωνα με τις παραπάνω σταθερές.

Το hash table αποτελείται από NUM_OF_ENTRIES entries, η κάθε μία από τις οποίες δείχνει σε ένα bucket (struct Bucket).

Το κάθε bucket αποτελείται από έναν πίνακα από struct BucketSpec δομές, έναν pointer σε επόμενο bucket και μία μεταβλητή που δείχνει πόσο γεμάτο είναι.

Κάθε BucketSpec δομή αποτελείται από έναν pointer σε ένα spec (struct Spec) και έναν pointer σε μία λίστα (struct ListId), στην οποία αποθηκεύονται τα παρόμοια ids με το συγκεκριμένο spec.

hash function:

```
int hashFunction(int HashtableNumOfEntries,char* hashName)
```

Η συνάρτηση αυτή υλοποιεί το hashing. Δέχεται ως όρισμα ένα spec_id και το πλήθος των entries που υποστηρίζονται από το πρόγραμμα. Επιστρέφει έναν ακέραιο στο διάστημα [0, HashtableNumOfEntries-1] ο οποίος εκφράζει μία θέση στων πίνακα των Entries. Για να υπολογίσει αυτό τον ακέραιο, η συνάρτηση σπάει αρχικά το spec_id ώστε να απομονώσει το νούμερο στο τέλος, καλεί τη συνάρτηση sumDigits() που υπολογίζει το άθροισμα των ψηφίων του και, τέλος, υπολογίζει το υπόλοιπο της διαίρεσης του αθροίσματος με το HashtableNumOfEntries

Εισαγωγή στο hashtable:

```
void insertInHashTable(struct Spec* spec, int
HashtableNumOfEntries, struct Entry* HashTable, int bucketSize)
```

Λαμβάνουμε ένα-ένα τα specs (τύπου struct Spec) από το datasetX και καλούμε τη συνάρτηση insertInHashTable. Καλούμε τη συνάρτηση hash function έτσι ώστε να βρεθεί το σωστό entry του hash table για το συγκεκριμένο spec, βρίσκουμε σε αυτό το entry την πρώτη ελεύθερη θέση σε bucket και αποθηκεύουμε εκεί το spec.

Στη συνέχεια, βάζουμε την matching_ids του συγκεκριμένου spec να αποτελείται από το id του (αντιστοιχία κάθε node με τον εαυτό του).

Αναζήτηση στο hash table:

```
void searchHashTable(struct Entry* hashTable, int
HashtableNumOfEntries, int bucketSize, char* id,char* dest_id)
```

Αφού εισάγουμε όλα τα specs στο hash table διαβάζουμε ένα-ένα τα ζεύγη παρόμοιων ids από το datasetW.

Για κάθε ένα ζεύγος που διαβάζουμε (έστω a,b, όπου a<b) ψάχνουμε αρχικά να βρούμε την matching_ids list (κλίκα) του μικρότερου, δηλαδή του a, την οποία και αποθηκεύουμε προσωρινά ως destination list. Ύστερα, ψάχνουμε τη λίστα matching_ids του b, την οποία αποθηκεύουμε ως origin list και πρέπει να την ενώσουμε στο τέλος της λίστας του a.
Για να επιτευχθεί αυτό καλείται η βοηθητική συνάρτηση fixing_pointers, η οποία αλλάζει κάθε pointer από bucket_spec που δείχνει στην origin list, να δείχνει στη destination list.

Software testing:

Χρησιμοποιήθηκε η βιβλιοθήκη “acutest.h”.

Ελέγχονται οι συναρτήσεις στο αρχείο test.c:

- addinlist2
- addinlist
- insertInHashTable
- searchHashTable
- extract_id
- deleteIdFromList2
- hashFunction
- greaterSpecId
- set_spec

*Επισυνάπτεται αρχείο 1111.json, βοηθητικό για τον έλεγχο ορθότητας της συνάρτησης set_spec.