CSE312-L: DSA
The LNMIIT, Rupa-ki-Nangal, Jaipur 302031
Training Set 09

Thanks for working well and now you are about to complete the last practice program for DSA Lab. The data-structures we need are quite elaborate and challenging.

File `cities` has several towns in Rajasthan and in the neighbouring states. A few distances between these towns is listed in file `roads`. File `roads` is provided with huge amount of information in it. However, the data is not enough and it will be good if you add distances from a few towns in the central Rajasthan to many other towns in the state.

A lot of code has been written for you at end of this document to help you begin developing program for this training set. The primary problem solved by the program is to determine the shortest distances to towns around the state from Rupa ki Nangal.

The code provided by your instructor has not been tested adequately and may notice errors. Please be aware of these roadblocks when using the code provided to you.

Training Set 09: Task 01

First programming task you need to complete is to construct a graph with a node for each town in file cities. These nodes may be collected in an array to conveniently locate them through index `townIdx`.

To support your claim that the graph for Rajasthan towns is well constructed, you must demonstrate a breadth-first search over the towns starting from Rupa ki Nangal. List the towns in order you reach them during the search. To support your later tasks, it will be helpful to write a separate function to perform the search.

To complete this task, you are strongly advised to write code for the following functions and use them in code `main.c` to perform breadth-first search.

```
/* ADT interface functions for link related needs */
     /* To support iteration and looping */
/* get link that connects HERE to the last neighbour
     connected to it */
struct link *getLink2newestNeighbour(struct town *here);
/* Get link that connects HERE to the neighbour before
     the one connected by LINK.  */
struct link *getLink2earlierNeighbour(struct town *here,
                           struct link *link);
```

Training Set 09: Task 02

Augment breadth-first search function you wrote in Task 01 to assign distances from Rupa ki Nangal to towns that are its immediate neighbours. List all neighbouring towns of Rupa ki Nangal in increasing order of distances from Rupa ki Nangal.

Training Set 09: Task 03

The final task for DSA lab this semester is to write a program that uses Dijkstra Shortest-path algorithm to print shortest distances to all towns in and around Rajasthan from the town located on the banks of celebrated Kanuta dam. Every citizen of Rajasthan must visit Rupa ki Nangal once in their lifetime! We can tell them how far their home is from this famous town.

The following ADT interface functions are useful support for writing the algorithm:

```
/* Return distance from the origin node */
int getDistFromStart(struct town *destination);
/* Set distance to town from origin return new distance*/
int setDistFromStart(struct town *destination, int distance);
/* Set final distance to town from origin returns distance
     Further re-assignment of a value to distance is ignored */
int setFinalDist(struct town *destination, int distance);
/* Returns 1 if HERE has final distance set */
int isDistanceFinal(struct town *here);
```

rajasthanGeography.h

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

#define TOWNS 100
extern struct town *townships[];
extern int towns;

struct town {
      char *name;
      struct link *edges;
      char townIdx, finalisedDist, flag1, flag2; // As needed
      int distance; // Distance from the origin town
};

struct link {
      /* The link connects these towns */
      char lowIdx, highIdx;// Indices in array townships
      int length;             // Distance between towns
      struct link *nxtLowEdge, *nxtHighEdge;
};

/* Functions to make towns and link them */
struct town *makeTown(char *name);
void linkTowns(char *thisTown,
                        char *thatTown, int distance);

/* Get town from NAME */
struct town *getTown(char *name);
/* Get name for TOWN */
char *getName(struct town *town);

/* ADT interface fucntions for link related needs */
      /* To support iteration and looping */
/* get link that connects HERE to the last neighbour
      connected to it */
struct link *getLink2newestNeighbour(struct town *here);
/* Get link that connects HERE to the neighbour before
      the one connected by LINK.  */
struct link *getLink2earlierNeighbour(struct town *here,
                              struct link *link);
      /* Other utility functions */
/* Get pointer to other town on ROAD */
struct town *getOtherTown(struct town *here, struct link *road);
/* Get length of ROAD */
```

```
int length(struct link *road);
/* Get name of town connected to HERE by ROAD */
char *neighbour(struct town *here, struct link *road);

/* Return distance from the origin node */
int getDistFromStart(struct town *destination);
/* Set distance to town from origin return new distance*/
int setDistFromStart(struct town *destination, int distance);
/* Set final distance to town from origin returns distance
      Further reassignement to distances  are ignored */
int setFinalDist(struct town *destination, int distance);
/* Returns 1 if HERE has final distance set */
int isDistanceFinal(struct town *here);
```

rajasthanGeography.c

```
#include "rajasthanGeography.h"

struct town *townships[TOWNS];
int towns = 0;

struct town *makeTown(char *name) {
	struct town *tn = malloc(sizeof(struct town));
	if (tn == NULL || towns == TOWNS) {
		return NULL;
	}
	townships[towns] = tn;
	tn->name = malloc(strlen(name)+1);
	strcpy(tn->name, name);
	tn->edges = NULL;
	tn->townIdx = towns++;
	tn->finalisedDist = 0;
	tn->distance = INT_MAX/TOWNS;
	return tn;
}

/* Get town from NAME */
struct town *getTown(char *name) {
	int i;
	for (i= 0; i < towns; i++)
		if (strcmpi(townships[i]->name, name) == 0)
			return townships[i];
	return NULL;
}
/* Get name for TOWN */
char *getName(struct town *town) {
	return town->name;
}

/* Get name for IDX */
static char *getNamebyIdx(int idx){
	return townships[idx]->name;
}

/* Get town connected to HERE by ROAD */
struct town *getOtherTown(struct town *here, struct link *road) {
	/* the road goes from *HERE to the other town */
	if (road == NULL || here == NULL)
		return NULL;
	if (townships[road->highIdx] == here)
		return townships[road->lowIdx];
	else if (townships[road->lowIdx] == here)
		return townships[road->highIdx];
```

```
      else
            return NULL;
}

/* Get name of town connected to HERE by ROAD */
char *neighbour(struct town *here, struct link *road) {
      struct town *t = getOtherTown(here, road);
      if (t != NULL)
            return t->name;
      return "Ghost town";
}

static struct link *nextRoadOutofHere(struct town *here, struct link *road)
{
      /* Return next road out of *HERE after *ROAD */
      struct link *nextRoad;
      assert(here != NULL);
      < 5 lines of C code removed - to be done by the student >
}

static struct link *insertLink(struct town *town1,
                                                          struct town *town2) {
      struct link *link = malloc(sizeof(struct link));
      assert(link != NULL);

      if (town1->townIdx>town2->townIdx) {
            link->highIdx = town1->townIdx;
            link->lowIdx = town2->townIdx;
            link->nxtHighEdge = town1->edges;
            town1->edges = link;
            link->nxtLowEdge = town2->edges;
            town2->edges = link;
      } else {
            link->highIdx = town2->townIdx;
            link->lowIdx = town1->townIdx;
            link->nxtHighEdge = town2->edges;
            town2->edges = link;
            link->nxtLowEdge = town1->edges;
            town1->edges = link;
      }
      return link;
}

void linkTowns(char *thisTown,
                        char *thatTown, int distance) {
      /* Links two towns by road */
      struct town *here, *there;
      struct link *here2there, *exists;
      here = getTown(thisTown);
```

```
	there = getTown(thatTown);
	if (here == there)
		return; // No self loops
	assert(here != NULL);
	assert(there != NULL);
	/* Does a road exist previously */
	exists = here->edges;
	here2there = NULL;
	while (here2there == NULL && exists != NULL) {
	/* Search if the towns are already connected */
		if (getOtherTown(here, exists) == there)
			here2there = exists;
		else
			exists = nextRoadOutofHere(here, exists);
	}
	if (here2there != NULL) {
		if (here2there->length > distance)
			here2there->length = distance;
		return;
	}
	here2there = insertLink(here, there);
	here2there->length = distance;
}

/* get link that connects HERE to the last neighbour
	connected to it */
struct link *getLink2newestNeighbour(struct town *here) {
	< 5 or less lines of C code removed. To be done by the student>
}

/* Get link that connects HERE to the neighbour before
	the one connected by LINK.  */
struct link *getLink2earlierNeighbour(struct town *here,
                              struct link *link) {
	< A single line of C code removed. To be done by the student >
}

/* Get length of ROAD */
int length(struct link *road) {
	return road->length;
}
```

main.c

```
#include <limits.h>
#include "rajasthanGeography.h"

char *squeeze(char *begin, char *end) {
	/* Squeeze out superflous white spaces */
	char *last, *start = begin;
	int first = 0;
	/* Set start to teh first non space */
	while (isspace(*start)) start++;
	/* locate last non space */
	while (isspace(*end) || *end == '\0')
		*(end--) = '\0';
	last = end;
	while (last >= start) {
		while (first && isspace(*last))
			last--;
		first = isspace(*last);
		*(end--) = *(last--);
	}
	return end+1;
}

void setUpGraph(void) {
		int i, distance;
	FILE *cities = fopen("cities", "r");
	FILE *roads = fopen("roads", "r");
	assert(cities != NULL && roads != NULL);
	char name[100];
	char *roadStart, *roadEnd, *kms;
	struct link *ll;

	while (fgets(name, 99, cities)) {
		if (strlen(name) < 2) break;
		name[strlen(name)-1] = '\0';
		if (makeTown(squeeze(name, name+strlen(name)))
		                                                == NULL)
			break; // Data structure full
	}

	while (fgets(name, 99, roads)) {
		if (strlen(name) < 4) break;
		name[strlen(name)-1] = '\0';
		roadStart = kms = name;
		while (!isdigit(*kms)) kms++;
		*(kms-1) = '\0';
		sscanf(kms, "%d", &distance);
		roadEnd = kms;
		while (!isspace(*roadEnd)) roadEnd++;
		linkTowns(squeeze(roadStart,roadStart+strlen(roadStart)),
			squeeze(roadEnd, roadEnd+strlen(roadEnd)), distance);
	}
}

void findShortestDistances(struct town *here) {
	/* Distances to other towns from HERE */
	int townsIdx, update = 0;
	if (here == NULL)
		return;
	int nextDist;
	struct town *newFound, *there;

	< 20 Lines of C code removed. Student to add code here for Task 03 >
```

```
      /* recursive call from town closest to the starting town
            Whose distance is not yet final */
      findShortestDistances(there);
}

int main(void) {
      int townsIdx;
      struct town *rupaNangal;
      setUpGraph();
      rupaNangal = getTown("Rupa ki Nangal");
      assert(rupaNangal != NULL);
      setFinalDist(rupaNangal, 0);
      findShortestDistances(rupaNangal);
      /* Print distances. */
      for (townsIdx = 0; townsIdx < towns; townsIdx++) {
            printf("Town %s is ", getName(townships[townsIdx]));
            if (isDistanceFinal(townships[townsIdx]))
                  printf("%d kms from LNMIIT\n",
                                    getDistFromStart(townships[townsIdx]));
            else
                  printf("unconnected to LNMIIT in this graph\n");
      }
      return 0;
}
```

Program output for the instructor is as follows:

```
Town Rupa ki Nangal is 0 kms from LNMIIT
Town Agra is 326 kms from LNMIIT
Town Delhi is 317 kms from LNMIIT
Town Jaisalmer is 634 kms from LNMIIT
Town Ajmer is 144 kms from LNMIIT
Town Jaipur is 13 kms from LNMIIT
Town Sikar is 113 kms from LNMIIT
Town Neem ka Thana is 133 kms from LNMIIT
Town Barmer is 557 kms from LNMIIT
Town Alwar is 154 kms from LNMIIT
Town Neemrana is 143 kms from LNMIIT
Town Sri Ganganagar is 735 kms from LNMIIT
Town Pilani is 263 kms from LNMIIT
Town Jhunjhunu is 322 kms from LNMIIT
Town Churu is 572 kms from LNMIIT
Town Banswara is 523 kms from LNMIIT
Town Udaipur is 418 kms from LNMIIT
Town Bikaner is 377 kms from LNMIIT
Town Nagaur is 758 kms from LNMIIT
Town Sawai Madhopur is 190 kms from LNMIIT
Town Kota is 344 kms from LNMIIT
Town Abu is 519 kms from LNMIIT
Town Kishangarh is 116 kms from LNMIIT
Town Pali is 314 kms from LNMIIT
Town Hanumangarh is 686 kms from LNMIIT
Town Jodhpur is 349 kms from LNMIIT
Town Chirawa is 278 kms from LNMIIT
Town Bharatpur is 189 kms from LNMIIT
Town Bundi is 307 kms from LNMIIT
Town Chittaurgarh is 335 kms from LNMIIT
```