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 as an example for specifying distances between the towns required in this training set. Later we may have a large file for roads data.

Before you come to the lab, you are requested to record for each town in the list, distances to the nearest 4 towns that are no more than 200 kms from it. If a town has no neighbouring town within 200 kms from it, then record distances to any 2 neighbouring towns. Use approximate distance between the neighbouring towns, if correct distances are difficult to find on the internet. We would assume that the distances between towns are same in both directions.

You will find some code at the 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 and may have 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.

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.

rajasthanGeography.h

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.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
      short distance; // Distance from the origin town
};

struct link {
      /* The link connects these towns */
      char lowIdx, highIdx;// Indices in array townships
      short 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
      Reassignement to be ignored */
int setFinalDist(struct town *destination, int distance);
```

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 = 0;
      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;
}

static char *getNamebyIdx(int idx){ /* Get name for 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) {
      /* Returns next road out of *HERE after *ROAD */
      struct link *nextRoad;
      assert(here != NULL);
      if (road == NULL)
            return NULL;
      if (road->highIdx == here->townIdx)
            nextRoad = road->nxtHighEdge;
      else nextRoad = road->nxtLowEdge;
}

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) {
      if (here == NULL)
            return NULL;
      else
            return here->edges;
}

/* Get link that connects HERE to the neighbour before
      the one connected by LINK.  */
struct link *getLink2earlierNeighbour(struct town *here,
                              struct link *link) {
      return nextRoadOutofHere(here, link);
}

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

/* 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
      Reassignement to be ignored */
int setFinalDist(struct town *destination, int distance);
```

main.c

```
#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;
}

int main(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);
	}
	return 0;
}
```