Solutions for Programming Assignment 2

CSCI2100B

November 8, 2013

Exercise 2.17

Analysis: The first problem is kind of simple. You just need to simulate the process and compare the input and output sequentially. When there is a conflict, just output impossible. And the conflict exists only when you need to pop a element from the queue and find out the element is not at the head of the queue. The sample code is shown below.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#define N 100000

typedef struct {
    int *data;
    int head;
    int tail;
    int num;
    int size;
} Queue;

void makeEmpty(Queue *aqueue)
{
    aqueue->head=0;
    aqueue->tail=-1;
    aqueue->num=0;
}

int createQueue(Queue* aqueue, int size)
{
    aqueue->data=(int*)malloc(sizeof(int)*size);
    if (aqueue->data==NULL) return 0;
    aqueue->size = size;
    makeEmpty(aqueue);
    return 1;
}

int isEmpty(Queue* aqueue)
{
    if (!aqueue->num) return 1;
    else return 0;
}
```

```
int isFull(Queue* aqueue)
{
    if (aqueue->num==aqueue->size) return 1;
    else return 0;
}

int front(Queue* aqueue)
{
    return aqueue->data[aqueue->head];
}

int dequeue(Queue* aqueue)
{
    if (!isEmpty(aqueue)) {
        int adata=front(aqueue);
        aqueue->head=(aqueue->head+1)%aqueue->size;
        aqueue->num--;
        return adata;
    }
    else return 0;
}

int enqueue(Queue* aqueue, int adata)
{
    if (!isFull(aqueue)) {
        aqueue->tail=(aqueue->tail+1)%aqueue->size;
        aqueue->data[aqueue->tail]=adata;
        aqueue->num++;
        return 1;
    }
    else return 0;
}

int main()
{
    Queue *track;
    char *ans;
    int T, n, cart, i, j, k, flag, p, max;

    track = (Queue*)malloc(sizeof(Queue));
    if (!createQueue(track, N)) {
        printf("Out of Memory!\n");
        return 0;
    }
    ans = (char*)malloc(sizeof(char)*2*N);
    scanf("%d", &T);
    for (i=0;i<T;i++) {
        scanf("%d", &n);
        flag = 1;
        makeEmpty(track);
        p = 0;
```

```
        max = 0;
        for (j=0; j<n;j++) {
            scanf("%d", &cart);
            if (flag) {
                if (cart>max) {
                    for (k=max+1;k<cart;k++) {
                        enqueue(track, k);
                        ans[p++]='I';
                    }
                    ans[p++]='S';
                    max = cart;
                }
                else if (cart<max) {
                    if (front(track)==cart) {
                        dequeue(track);
                        ans[p++]='O';
                    }
                    else flag = 0;
                }
                else flag=0;
            }
        }
        ans[p]='\0';
        if (flag) printf("%s\n", ans);
        else printf("Impossible\n");
    }
    free(track);
    free(ans);
    return 0;
}
```

Exercise 3.35

Analysis: The idea is to use a recursive function (in the sample program is sumNode()) to compute the distance between nodes. The sample code is shown below. Here we thank Mr. LAU Cheuk Yin for providing the sample program since our program uses a more advanced algorithm which is beyond the scope of this course (originally n can be as large as 10^6, we have reduced it to 10^3 so that you can pass by using a brute-force algorithm). We think the annotation in this program is so clear that you can understand the recursive process easily.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
        int size;                // size of tree
        struct node {        // root and parent is not need in this structure
                int deg;        // degree of the node
                int *vtx;        // vertices that the node connects to
                int *wgt;        // corresponding weight of the edge
        } *nodes;                // array of struct node
} tree;

tree createTree(int n) {
```

```
        tree t;
        t.size = n;
        t.nodes = malloc(n*sizeof(struct node)); // n nodes in tree t
        int i;
        for (i=0; i<n; i++) {
                t.nodes[i].deg = 0;        // initial degree is 0
                t.nodes[i].vtx = malloc(n*sizeof(int));        // at most n vertex
                t.nodes[i].wgt = malloc(n*sizeof(int));        // at most n corresponding weight
        }
        return t;
}

void addEdge(tree *t, int va, int vb, int w) {
        int *a, *b, i;
        for (i=0; i<2; i++) {
                if (i%2) {a=&va; b=&vb;} else {a=&vb; b=&va;}
                int *deg = &(t->nodes[*a].deg);
                t->nodes[*a].vtx[*deg] = *b;
                t->nodes[*a].wgt[*deg] = w;
                (*deg)++;
        }
}

void sumNode(tree *t, const int intN, int preN, int link, long long *sum, long long curPW) {
        /*        intN        = initial node which initiates this function
        *        preN        = previous node;
        *        link        = the link used by the previous node to connect to this node
        *        sum         = accumulated sum
        *        curPW        = product of previous path weight
        *        The idea: sum only when current node larger than initial node        */

        int curN = t->nodes[preN].vtx[link];
        if (curN != intN) {         // ensure no self summing
                int i, deg = t->nodes[curN].deg;
                curPW *= t->nodes[preN].wgt[link];
                if (curN > intN) *sum += curPW;
                for (i=0; i<deg; i++) if (preN != t->nodes[curN].vtx[i])
                        sumNode(t, intN, curN, i, sum, curPW);
        }
}

long long sumWeight(tree *t) {
        int curN, next, n=t->size;
        long long sum=0;
        for (curN=0; curN<n-1; curN++) {
                int deg = t->nodes[curN].deg;
                for (next=0; next<deg; next++) sumNode(t, curN, curN, next, &sum, 1);
        }
        return sum;
}

int main() {
```

```
        int i, n; scanf("%d", &n);
        tree t = createTree(n);
        for (i=0; i<n-1; i++) {
                int a, b, w;
                scanf("%d%d%d", &a, &b, &w);
                addEdge(&t, a-1, b-1, w); // internal numbering starts from 0 instead of 1
        }
        printf("%lld\n", sumWeight(&t)%1000000007);
        free(t.nodes);
        return 0;
}
```

Exercise 4.17

Analysis: For each sum, first write a for-loop to enumerate all the keys, namely key_1. After subtracting the value of key from sum, try to find whether the remaining value, namely key_2, is one of the keys (pay attention that the key_1 should be different from key_2). To reduce the time complexity, you need to build a hash table so that we can find key_2 in $O(1)$ time. The hash table size should be large enough and is better to be a prime. The sample code is shown below.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#define HASHTABLESIZE 9999979

typedef struct
{
    int key[HASHTABLESIZE];
    char state[HASHTABLESIZE];
   /* 0=empty, 1=occupied */
} hashtable;

/* The hash function */
int hash(int input)
{
    int value;
    value = input % HASHTABLESIZE;
    if (value < 0) value = HASHTABLESIZE + value;
    return value;
}

/* The h function */
int h(int k, int input)
{
    int value;
    value = (hash(input) + k)% HASHTABLESIZE;
    if (value < 0) value = HASHTABLESIZE + value;
    return value;
}

void insert(int item, hashtable * ht )
{
```

```
    int hash_value, i, k;

    hash_value = hash(item);
    i = hash_value;
        k = 1;
        while (ht->state[i]!= 0) {
                i = h(k++,item);
        }
        ht->key[i] = item;
        ht->state[i] = 1;
}

int find(int item, hashtable * ht)
{
    int hash_value, i, k, flag;

    hash_value = hash(item);
    i = hash_value;
        k = 1;
        flag = 0;
        while (ht->state[i]!= 0) {
                if (ht->key[i] == item) {
                        flag = 1;
                        break;
                }
                i = h(k++,item);
        }
        return flag;
}

int main()
{
    int n, m, i, j, num, want, flag;
    int array[100000];
    hashtable *ht;

    scanf("%d%d", &n, &m);
    ht = (hashtable*)malloc(sizeof(hashtable));
    memset(ht->key,0,sizeof(int)*HASHTABLESIZE);
    memset(ht->state,0,sizeof(char)*HASHTABLESIZE);
    for (i=0; i<n; i++) {
        scanf("%d", &num);
        array[i] = num;
        insert(num, ht);
    }
    for (i=0; i<m; i++) {
        scanf("%d", &num);
        flag = 0;
        for (j=0;j<n;j++) {
            want = num-array[j];
            if (find(want, ht) && (want != array[j])) {
                printf("Yes\n");
```

```
                flag = 1;
                break;
            }
        }
        if (!flag) printf("No\n");
    }
    return 0;
}
```