

首先我先把LinkedList寫出來，然後在鍊表的基礎上添加題目所需的函式，只要有關修改鍊表的函式，我都是以傳址為主，但有些回傳布林值或長度的我就會寫return值。比較特別的是我的search、delete、insert的函式沒有包含讀取檔案與建立鍊表的步驟，我將更細部的步驟像是insert中所需的sort拆分出來以利於日後擴充與除錯需求。

以下是我的函式宣告，實作的代碼在P.3~P.12

#ifndef HW11Func\_H

#define HW11Func\_H

struct prefix{

    unsigned ip;

    unsigned char len;

    struct prefix \*next;

};

struct prefix \*sort(struct prefix \*head);

void Insert(struct prefix\*\* head,unsigned newIP,unsigned char newLen);

// void prefix\_insert(struct prefix\*\* targetHead,const char\* PATH);

void prefix\_insert(struct prefix\*\* targetHead,struct prefix \*insertNode);

void prefix\_delete(struct prefix\*\* targetHead,struct prefix \*deleteNode);

int Len(struct prefix \*head);

void PrintLinkedList(struct prefix\* head);

void input(struct prefix\*\* head,const char\* PATH);

void lenth\_distribution(struct prefix \*head);

struct prefix \*\*segment(struct prefix \*head,int d);

int search(struct prefix \*head,struct prefix \*node);

#endif

#include<stdio.h>

#include<stdlib.h>

#include"hw11Func.h"

#ifndef HW11Func\_H

#define HW11Func\_H

struct prefix{

    unsigned ip;

    unsigned char len;

    struct prefix \*next;

};

struct prefix \*sort(struct prefix \*head);

void Insert(struct prefix\*\* head,unsigned newIP,unsigned char newLen);

// void prefix\_insert(struct prefix\*\* targetHead,const char\* PATH);

void prefix\_insert(struct prefix\*\* targetHead,struct prefix \*insertNode);

void prefix\_delete(struct prefix\*\* targetHead,struct prefix \*deleteNode);

int Len(struct prefix \*head);

void PrintLinkedList(struct prefix\* head);

void input(struct prefix\*\* head,const char\* PATH);

void lenth\_distribution(struct prefix \*head);

struct prefix \*\*segment(struct prefix \*head,int d);

int search(struct prefix \*head,struct prefix \*node);

/\*\*

 \* @brief Using Selection Sort to sort the linkedList, return a sorted linkedList

 \*

 \* @param head the linkedList

 \* @return struct prefix\* , a sorted linkedList

 \*/

struct prefix \*sort(struct prefix \*head){

    struct prefix \*orginP = head;

    while(head->next){

        struct prefix \*minPre = head,\*minP,\*minPNext;

        struct prefix \*pPre = head,\*p;

        // find minPre

        while(pPre->next){

            if(minPre->next->ip > pPre->next->ip){

                minPre = pPre;

            }

            pPre = pPre->next;

        }

        pPre = head;

        if(pPre == orginP && pPre->ip > minPre->next->ip){

            minP = minPre->next;

            minPNext = minP->next;

            p = pPre->next;

            unsigned temip = pPre->ip;

            unsigned char temlen = pPre->len;

            pPre->ip = minP->ip;

            pPre->len = minP ->len;

            minP->ip = temip;

            minP->len = temlen;

            // PrintLinkedList(orginP);

            continue;

        }

        if(minPre != head){

            minP = minPre->next;

            minPNext = minP->next;

            p = pPre->next;

            // connect minPre to minNext

            minPre->next = minPNext;

            // Insert minP to p;

            pPre->next = minP;

            minP->next = p;

            // next

            head = head->next;

            // PrintLinkedList(orginP);

        } else {

            head = head->next;

            // PrintLinkedList(orginP);

        }

    }

    return orginP;

}

/\*\*

 \* @brief this is an Insert function only for type struct prefix

 \*

 \* @param head the linkedList's pointer

 \* @param newIP the newIP data

 \* @param newLen the newLen data

 \*/

void Insert(struct prefix\*\* head,unsigned newIP,unsigned char newLen){

    struct prefix \*newNode = (struct prefix\*)malloc(sizeof(struct prefix));

    newNode->ip = newIP;

    newNode->len = newLen;

    newNode->next = \*head;

    \*head = newNode;

}

// /\*\*

//  \* @brief  insert a prefix in a one-by-one fashion in the increasing order of the unsigned numbers of the prefixes.

//  \*

//  \* @param targetHead the linkedList

//  \* @param PATH Inserted File Path

//  \*/

// void prefix\_insert(struct prefix\*\* targetHead,const char\* PATH){

//     struct prefix \*insert\_table = NULL,\*sorted\_insert\_table = NULL,\*head;

//     input(&insert\_table,PATH);

//     sorted\_insert\_table = sort(insert\_table);

//     head = sorted\_insert\_table;

//     while(head){

//         struct prefix \*newNode = (struct prefix\*)malloc(sizeof(struct prefix));

//         newNode->ip = head->ip;

//         newNode->len = head->len;

//         newNode->next = \*targetHead;

//         \*targetHead = newNode;

//         head = head->next;

//     }

//     // free the memory

// }

/\*\*

 \* @brief insert a prefix into targetLinkedList

 \*

 \* @param targetHead the targetLinkedList

 \* @param insertNode the insertedList

 \*/

void prefix\_insert(struct prefix\*\* targetHead,struct prefix \*insertNode){

    struct prefix \*newNode = (struct prefix\*)malloc(sizeof(struct prefix));

    newNode->ip = insertNode->ip;

    newNode->len = insertNode->len;

    newNode->next = \*targetHead;

    \*targetHead = newNode;

}

/\*\*

 \* @brief Delete a node from targetLinkedList

 \*

 \* @param targetHead the targetLinkedList

 \* @param deleteNode the deleteNode

 \*/

void prefix\_delete(struct prefix\*\* targetHead,struct prefix \*deleteNode){

    if(!(\*targetHead)) return;

    struct prefix \*preHead = \*targetHead;

    while(preHead->next){

        // if deleteNode == the head

        if((preHead->ip == deleteNode->ip) && (preHead->len == deleteNode->len) && (preHead == \*targetHead)){

            struct prefix \*p = preHead;

            preHead = preHead->next;

            \*targetHead = preHead;

            free(p);

            return;

        } else if(preHead->next->ip == deleteNode->ip && preHead->next->len == deleteNode->len){

            struct prefix \*head = preHead->next;

            preHead->next = head->next;

            free(head);

            return;

        }

        preHead = preHead->next;

    }

    // if only one node in targetLinkedList

    if(!(preHead->next) && (preHead->ip == deleteNode->ip) && (preHead->len == deleteNode->len) && (preHead == \*targetHead)){

    // struct prefix \*p = preHead;

    \*targetHead = NULL;

    // free(p);

    }

}

/\*\*

 \* @brief Return the total number of prefixes in the linkedList.

 \*

 \* @param head The linkedList

 \* @return int

 \*/

int Len(struct prefix \*head){

    struct prefix \*p;

    int count = 0;

    for(p = head;p != NULL;p = p->next,count++);

    return count;

}

/\*\*

 \* @brief Print out the whole linkList

 \*

 \* @param head the linkList

 \*/

void PrintLinkedList(struct prefix\* head){

    while(head != NULL){

        unsigned a,b,c,d;

        a = ((head->ip) >> 24) & 0xFF;

        b = ((head->ip) >> 16) & 0xFF;

        c = ((head->ip) >> 8) & 0xFF;

        d = ((head->ip) >> 0) & 0xFF;

        // printf("%d/%d\n", head->ip,head->len);

        printf("%u.%u.%u.%u/%hhu\n", a,b,c,d,head->len);

        head = head->next;

    }

    printf("NULL\n");

}

/\*\*

 \* @brief Read all the prefixes from the input file.

 \*

 \* @param head The linkedList's pointer

 \* @param PATH File path

 \*/

void input(struct prefix\*\* head,const char\* PATH){

    FILE \*file = fopen(PATH, "r");

    unsigned flag,a,b,c,d,ip;

    unsigned char len;

    while ((flag = fscanf(file, "%u.%u.%u.%u/%hhu", &a, &b, &c, &d, &len))!= EOF) {

        if(flag >= 5){

            // printf("%u.%u.%u.%u/%hhu ",a,b,c,d,len);

            ip = (a << 24)|(b << 16)|(c << 8)|d;

            // printf("IP: %d\n",ip);

            Insert(head,ip,len);

        } else {

            len = (d)?32:(c)?24:(b)?16:(a)?8:8;

            // printf("%u.%u.%u.%u/%hhu ",a,b,c,d,len);

            ip = (a << 24)|(b << 16)|(c << 8)|d;

            // printf("IP: %d\n",ip);

            Insert(head,ip,len);

        }

    }

}

/\*\*

 \* @brief compute the number of prefixes with prefix length i, for i = 0 to 32

 \*

 \* @param head the linkedList

 \* @return int

 \*/

void lenth\_distribution(struct prefix \*head){

    struct prefix \*p;

    for(int i = 0;i<=32;i++){

        int count = 0;

        for(p = head;p != NULL;p = p->next){

            if(p->len == i) count++;

        }

        if(count != 0) printf("the number of prefixes with prefix length %d = %d\n",i,count);

    }

}

/\*\*

 \* @brief Return an struct prefix array contains grouped prefixes, the prefixes in each group are linked by a linkedList

 \*

 \* @param head the reference linkedList

 \* @param d d

 \* @return struct prefix\*\* ,an array stored (struct prefix \*) pointers in groups with linkedList

 \*/

struct prefix \*\*segment(struct prefix \*head,int d){

    struct prefix \*\*newSegment = (struct prefix\*\*)malloc(sizeof(struct prefix\*)\*(1<<d));

    // initialize the newSegment array with NULL pointer

    for(int i = 0;i< (1<<d) ;i++){

        \*(newSegment+i) = NULL;

    }

    struct prefix \*p = head;

    for(int i = 0;i<(1<<d);i++){

        for(p = head;p != NULL;p = p->next){

            if( (p->ip)>>(32-d) == i && p->len >= 8){

                struct prefix \*newNode = (struct prefix\*)malloc(sizeof(struct prefix));

                newNode->ip = p->ip;

                newNode->len = p->len;

                newNode->next = \*(newSegment+i);

                \*(newSegment+i) = newNode;

                // printf("now is in %d :\n",i);

                // PrintLinkedList(\*(newSegment+i));

            }

        }

    }

    return newSegment;

}

/\*\*

 \* @brief search the node from the LinkedList, if true return 1, otherwise return 0

 \*

 \* @param head the LinkedList

 \* @param node the node

 \* @return int ,0 for false, 1 for true

 \*/

int search(struct prefix \*head,struct prefix \*node){

    while(head){

        if(head->ip == node->ip) return 1;

        head = head->next;

    }

    return 0;

};

#endif

/\*\*

 \* @param argv[1] "routing\_table.txt"

 \* @param argv[2] "inserted\_prefixes.txt"

 \* @param argv[3] "deleted\_prefixes.txt"

 \* @param argv[4] "trace\_file.txt"

 \* @param argv[5] "d" use atoi to transform to integer

\*/

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

    struct prefix \*routing\_table = NULL;

    input(&routing\_table,argv[1]);

    // PrintLinkedList(routing\_table);

    printf("The total number of prefixes in the input file is : %d.\n",Len(routing\_table));

    // newSegment is an array stored (struct prefix \*) pointers in groups with linkedList

    // struct prefix \*\*newSegment = segment(routing\_table,atoi(argv[5]));

    // for(int i = 0;i < (1<<(atoi(argv[5]))) ;i++){

    //     printf("The number of prefixes in group %d = %d\n",i,Len(\*(newSegment+i)));

    // }

    // PrintLinkedList(routing\_table);

    // DO INSERT

    printf("Do insert:\n");

    struct prefix \*insert\_table = NULL,\*sorted\_insert\_table = NULL,\*head;

    input(&insert\_table,argv[2]);

    sorted\_insert\_table = sort(insert\_table);

    head = sorted\_insert\_table;

    while(head){

        prefix\_insert(&routing\_table,head);

        head = head->next;

    }

    printf("The total number of prefixes in the input file is : %d.\n",Len(routing\_table));

    // PrintLinkedList(routing\_table);

    //DO DELETE

    printf("Do delete:\n");

    struct prefix \*delete\_table = NULL;

    input(&delete\_table,argv[3]);

    head = delete\_table;

    while(head){

        prefix\_delete(&routing\_table,head);

        head = head->next;

    }

    printf("The total number of prefixes in the input file is : %d.\n",Len(routing\_table));

    // PrintLinkedList(routing\_table);

    //DO SEARCH

    printf("Do search:\n");

    struct prefix \*search\_table = NULL;

    input(&search\_table,argv[4]);

    head = search\_table;

    int result;

    while(head){

        result = search(routing\_table,head);

        head = head->next;

        // printf("%s\n",result?"True":"False");

    }

    // printf("The total number of prefixes in the input file is : %d.\n",Len(routing\_table));

    // PrintLinkedList(routing\_table);

}