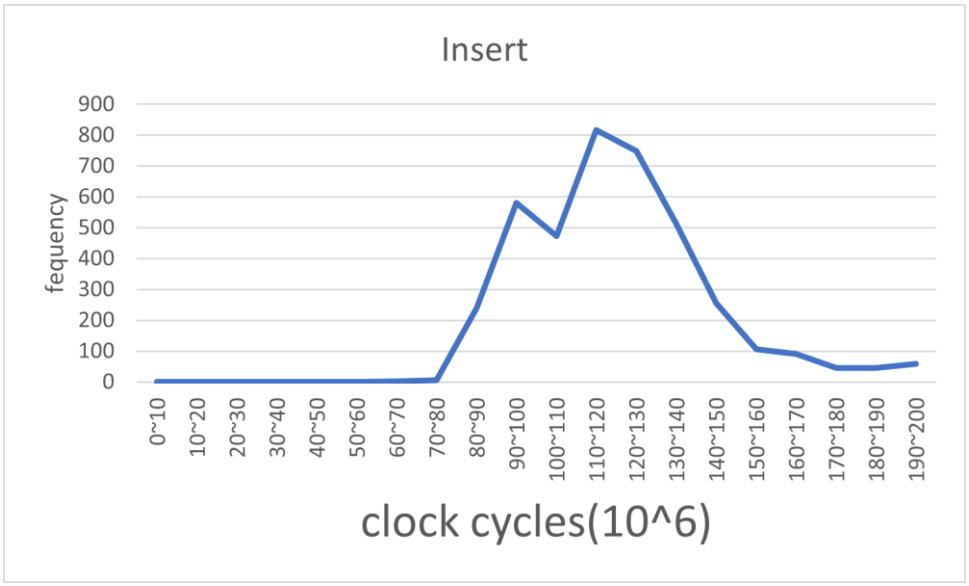
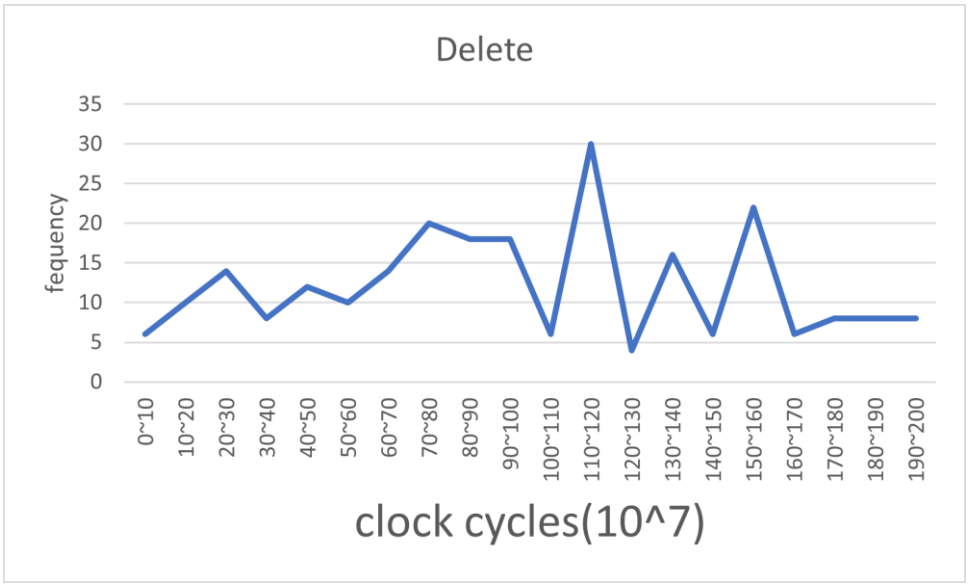
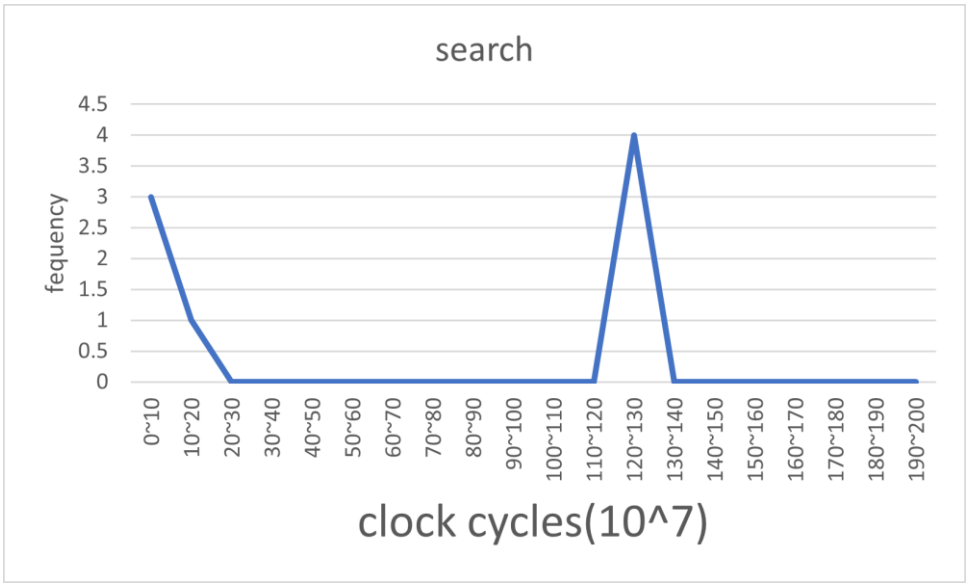

Insert
fequency
900
800
700
600
500
400
300
200
100
0
0~10
10~20
20~30
30~40
40~50
50~60
60~70
70~80
80~90
90~100
100~110
110~120
120~130
130~140
140~150
150~160
160~170
170~180
180~190
190~200
clock cycles(10^6)


Delete
fequency
35
30
25
20
15
10
5
0
0~10
10~20
20~30
30~40
40~50
50~60
60~70
70~80
80~90
90~100
100~110
110~120
120~130
130~140
140~150
150~160
160~170
170~180
180~190
190~200
clock cycles(10^7)


search
fequency
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0~10
10~20
20~30
30~40
40~50
50~60
60~70
70~80
80~90
90~100
100~110
110~120
120~130
130~140
140~150
150~160
160~170
170~180
180~190
190~200
clock cycles(10^7)

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
```

```c
#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);


}
```