****
Report: HW8
Author: E34071061 謝沅承 <andy420811@gmail.com>
Class: 化工 (甲班)
Description:
讀取 file 中的資料，轉換為固定資料並以 node 的形式創建新的資料結構，並進行存取，插入以及刪除等作業。
****
Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<stdint.h>

struct Data {
	uint32_t upper;
	uint32_t lower;
	uint16_t prefix;
};
typedef struct ipNode {
	struct Data data;
	struct ipNode *next;
}ipNode;
ipNode *mallocNode();
unsigned long long int begin, end, total = 0;
void dataInsert(ipNode *p,uint32_t upper,uint32_t
lower,uint16_t prefix);
void prt(uint32_t *p,int n);
void addList(ipNode **head,ipNode *p);
int searchList(ipNode *head,int a);
void nodeDelete(ipNode **head, ipNode *p);
void add(ipNode **a, ipNode **b, ipNode **c, ipNode *node);
int search(ipNode *a, ipNode *b, ipNode *c, int num);
void prtstate(int success, int fail);
static __inline__ unsigned long long rdtsc(void);
void insertfile(FILE *A, ipNode **a, ipNode **b, ipNode **c);
void statistic(uint64_t b,uint64_t c);
void searchfile(FILE *A, ipNode *a, ipNode *b, ipNode *c, int
*success, int *fail, int test);
```

```
void deletefile(FILE *A, ipNode **a, ipNode **b, ipNode **c);
void Delete(ipNode **a, ipNode **b, ipNode **c, ipNode *node);
void prtnode(ipNode *a);
int main(int argc, char *argv[]) {
	FILE *Prefix, *Trace, *Insert, *Delete;
	ipNode *a=NULL, *b=NULL, *c=NULL;

	if (argc != 5) { printf("lack file\n"); return -1; }
	if ((Prefix = fopen(argv[1], "r")) == NULL) {
		printf("%s can't be opened\n", argv[1]);
		return -1;
	}
	if ((Trace = fopen(argv[2], "r")) == NULL) {
		printf("%s can't be opened\n", argv[2]);
		exit(EXIT_FAILURE);
	}
	if ((Insert = fopen(argv[3], "r")) == NULL) {
		printf("%s can't be opened\n", argv[3]);
		exit(EXIT_FAILURE);
	}
	if ((Delete = fopen(argv[4], "r")) == NULL) {
		printf("%s can't be opened\n", argv[4]);
		exit(EXIT_FAILURE);
	}


	int success = 0, fail = 0;
	insertfile(Prefix, &a, &b, &c);
	fclose(Prefix);
//	prtnode(a);prtnode(b);prtnode(c);
	printf("After seg. table create\n");
	searchfile(Trace, a, b, c, &success, &fail,0);
	fclose(Trace);
	prtstate(success, fail);
	insertfile(Insert, &a, &b, &c);
//	prtnode(a);prtnode(b);prtnode(c);
	fclose(Insert);
	printf("After insertion\n");
```

```
	printf("avg. insertion time= %llu cycles\n", total);
	Trace = fopen(argv[2], "r");
	searchfile(Trace, a, b, c, &success, &fail, 0);
	fclose(Trace);
	prtstate(success, fail);
	printf("After deletion\n");
	deletefile(Delete, &a, &b, &c);
	 prtnode(a);prtnode(b);prtnode(c);
	fclose(Delete);
	printf("avg. deletion time= %llu cycles\n", total);
	Trace = fopen(argv[2], "r");
	searchfile(Trace, a, b, c, &success, &fail, 1);
	fclose(Trace);
	printf("avg. search times=%llu cycles\n", total);
	prtstate(success, fail);
	statistic(0, 1);



	return 0;
}
void deletefile(FILE *A, ipNode **a,ipNode **b,ipNode **c)
{
	char Char[15];
	uint8_t bit32, bit24, bit16, bit8, prefix;
	uint32_t num[4], count = 0;
	ipNode *node;
	total=0;
	while (fscanf(A, "%s", &Char) != EOF)
	{
		count++;
		begin = rdtsc();
		sscanf(Char, "%hhu.%hhu.%hhu.%hhu/%hhu",
&bit32, &bit24, &bit16, &bit8, &prefix);
		num[0] = (16777216)*(bit32)+(bit24)*(65536) +
(bit16)*(256) + (bit8);
		num[1] = prefix;
	//	prt(num, 0);
```

```
                num[0] >>= (32 - num[1]);
                num[2] = num[0] <<= (32 - num[1]);//lower
                num[3] = num[0] + (1 << (32 - num[1]));//higher
        //      prt(num, 3);
                node = mallocNode();
                dataInsert(node, num[3], num[2], num[1]);
                Delete(a, b, c, node);
                end = rdtsc();
                total += (end - begin);
                //
        printf("%u.%u.%u.%u/%u\n",bit32,bit24,bit16,bit8,pr
efix);
                //
        printf("%llu,%u\t",num[0],num[1]);
        }
        count?total = total / count:0;
}
void Delete(ipNode **a, ipNode **b, ipNode **c, ipNode *node)
{
        int i = node->data.prefix;
        if (i < 16) {
                nodeDelete(a, node);
        }
        if (i > 15 && i < 24) {
                nodeDelete(b, node);
        }
        if (i > 23) {
                nodeDelete(c, node);
        }
}
void insertfile(FILE *A, ipNode **a, ipNode **b, ipNode **c)
{
        char Char[15];
        uint8_t bit32, bit24, bit16, bit8, prefix;
        uint32_t num[4],count=0;
        ipNode *node;
        total = 0;
        while (fscanf(A, "%s", &Char) != EOF)
```

```
	{
		count++;
		begin = rdtsc();
		sscanf(Char, "%hhu.%hhu.%hhu.%hhu/%hhu",
&bit32, &bit24, &bit16, &bit8, &prefix);
		num[0] = (16777216)*(bit32)+(bit24)*(65536) +
(bit16)*(256) + (bit8);
		num[1] = prefix;
	//	prt(num, 0);
		num[0] >>= (32 - num[1]);
		num[2] = num[0] <<= (32 - num[1]);//lower
		num[3] = num[0] + (1 << (32 - num[1]));//higher
	//	prt(num, 3);
		node=mallocNode();
		dataInsert(node, num[3], num[2], num[1]);
		add(a, b, c, node);
		end = rdtsc();
		total += (end - begin);
		//
	printf("%u.%u.%u.%u/%u\n",bit32,bit24,bit16,bit8,pr
efix);
		//
	printf("%llu,%u\t",num[0],num[1]);
	}
	count?total /= count:0;
}
void statistic(uint64_t b,uint64_t c) {
	static uint64_t a[100] = { 0 };
	if (c) {
		for (int i = 0; i < 100; i++) {
			printf("%llu:%4llu\n", 5000 * (i + 1),
a[i]);
		}
	}
	else
	{
		a[b / 5000]++;
	}
```

```
}
void prtstate(int success, int fail) {
	printf("success search times = %d\n", success);
	printf("fail search times = %d\n", fail);
}
static __inline__ unsigned long long rdtsc(void)
{
	unsigned hi, lo;
	__asm__ __volatile__("rdtsc" : "=a"(lo), "=d"(hi));
	return ((unsigned long long)lo) | (((unsigned long
long)hi) << 32);
}


void searchfile(FILE *A,ipNode *a,ipNode *b,ipNode *c,int
*success,int *fail,int test) {
	uint32_t num;
	uint32_t count=0;
	total = 0;
	*success = *fail = 0;
	while (fscanf(A,"%u",&num)!=EOF)
	{
		begin = rdtsc();
		search(a, b, c, num) ? (*success)++ : (*fail)++;
		count++;
		end = rdtsc();
		if(test)statistic(end - begin, 0);
		total += (end - begin);
	}
	count?total = total / count:0;
}
int search(ipNode *a, ipNode *b, ipNode *c, int num) {
	if (searchList(c, num))return 1;
	if (searchList(b, num))return 1;
	if (searchList(a, num))return 1;
	return 0;
}
```

```
void add(ipNode **a, ipNode **b, ipNode **c, ipNode *node)
{
	int i = node->data.prefix;
	if (i < 16) {
		addList(a, node);
	}
	if (i > 15 && i < 24) {
		addList(b, node);
	}
	if (i > 23) {
		addList(c, node);
	}
}
void prtnode(ipNode *a){
	int count=0;
	for(;a;a=a->next){
		count++;
	}printf("num=%d\n",count);
}

ipNode *mallocNode(){
	ipNode *p;
	if((p=malloc(sizeof(ipNode)))==NULL){printf("malloc
error");exit(EXIT_FAILURE);}
	p->next=NULL;
	return p;
}
void dataInsert(ipNode *p,uint32_t upper,uint32_t
lower,uint16_t prefix){
	p->data.upper=upper;
	p->data.lower=lower;
	p->data.prefix=prefix;
}
void addList(ipNode **head,ipNode *p){
	if(*head==NULL){
		*head=p;
		return;
	}
```

```
        p->next=(*head)->next;
        (*head)->next=p;
}
int searchList(ipNode *head,int a){
        int prefix=0;
        for(;head;head=head->next){
                if(a<head->data.upper&&a>head->data.lower){
                                prefix=head->data.prefix;
                                break;
                }
        }
        return prefix==0?0:1;
}
void nodeDelete(ipNode **head,ipNode *p){
        ipNode *i=*head;ipNode *pre=NULL;
        if(p==NULL)return;
        if((*head)->data.upper==p->data.upper&&
           (*head)->data.lower==p->data.lower&&
           (*head)->data.prefix==p->data.prefix){
                (*head)=(*head)->next;
                return;
        }
        for(;i;pre=i,i=i->next){
                if(i->data.upper==p->data.upper&&
                   i->data.lower==p->data.lower&&
                   i->data.prefix==p->data.prefix){
                        pre->next=i->next;
                        free(i);
                }
        }
}

void prt(uint32_t *p,int n){
                for(uint32_t j=0x80000000;j;j>>=1)
             printf("%u",(p[n]&j)?1:0);
        printf("\n");
}
```

Compilation:

gcc -o hw8 hw8.c -g

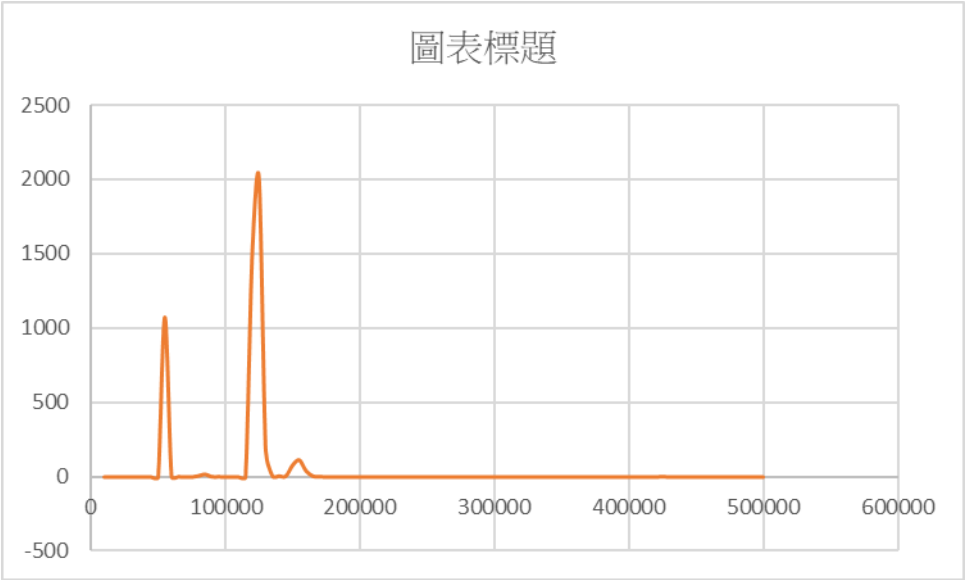
Execution:

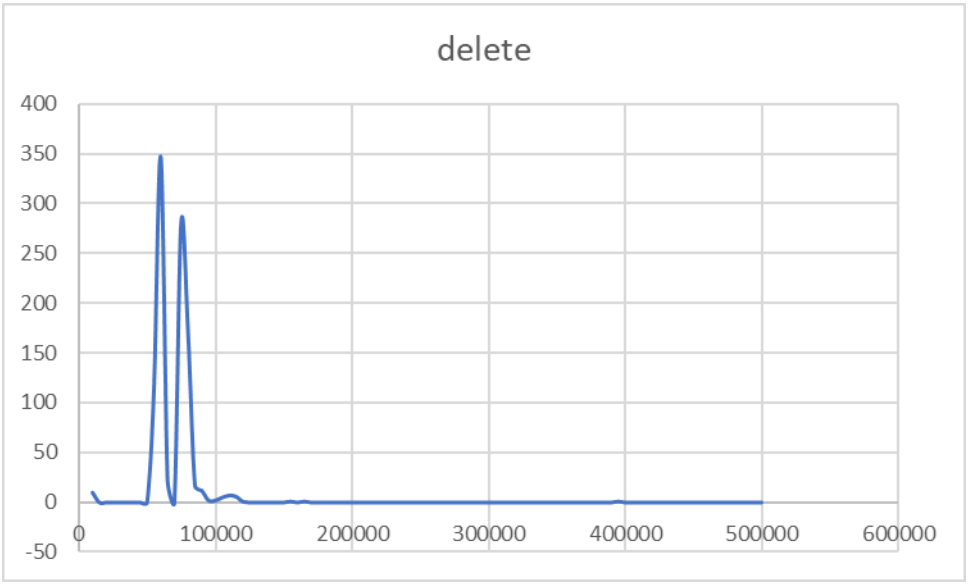
./hw8 prefix_10K.txt trace_IPaddress_100K.txt insert_1K.txt delete_1K.txt

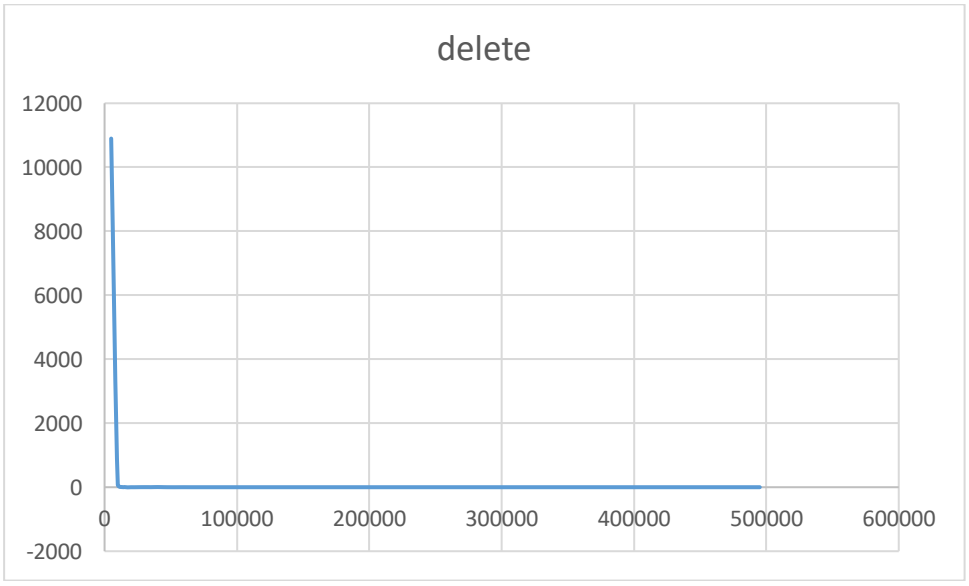
Output:

```
After seg. table create
success search times = 5080
fail search times = 0
After insertion
avg. insertion time= 2106 cycles
success search times = 5080
fail search times = 0
After deletion
avg. deletion time= 65730 cycles
avg. search times=109900 cycles
success search times = 3930
fail search times = 1150
```

search:




delete
400
350
300
250
200
150
100
50
0
-50
0
100000
200000
300000
400000
500000
600000


delete
12000
10000
8000
6000
4000
2000
0
-2000
0
100000
200000
300000
400000
500000
600000