1.快排

1.1快排的思想

原理：以第一个数作为轴线，先保存这个值到tmp中，从右向左比较

如果后者小就将当前的数赋值到第一个位置，如果遇到赋值就变换方

向，从左向右比较，如果左侧值大于轴线值就赋值，并在此改变比较

方向。第一轮比较完之后把tmp赋值到中间的位置，第一轮比较的结

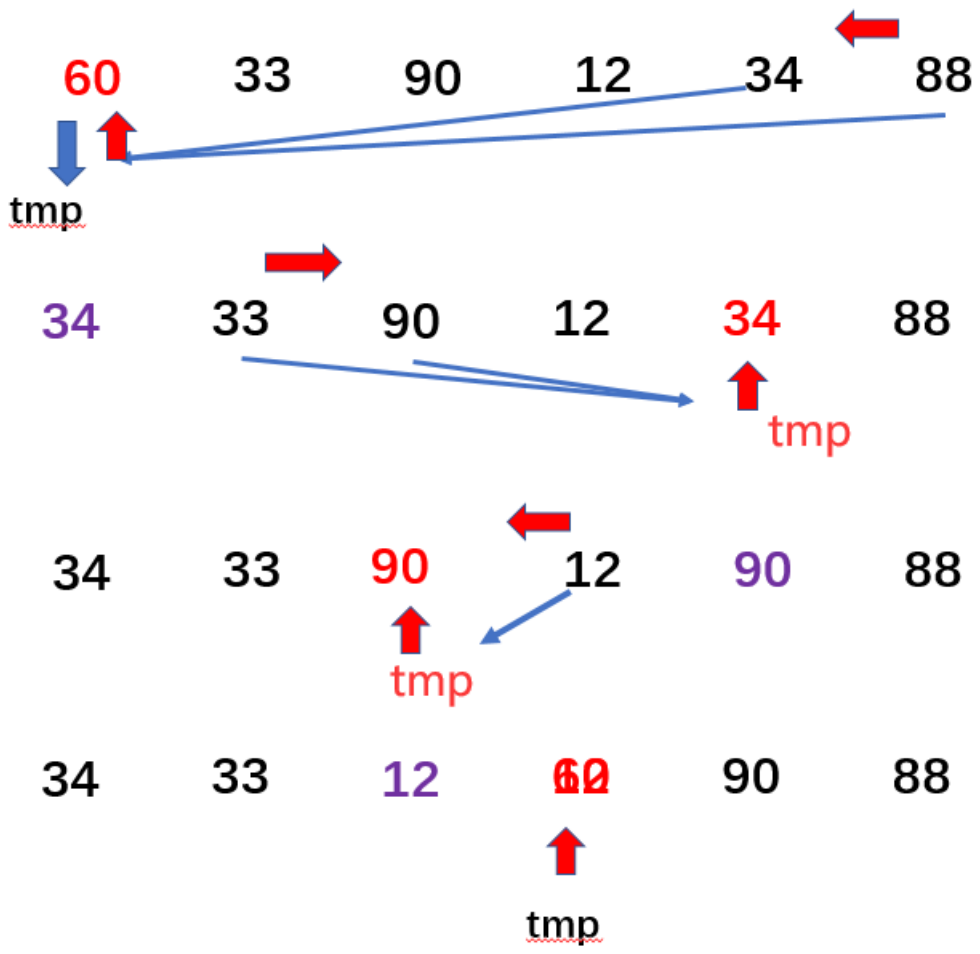
果为轴线左边的全部小于轴线值，轴线右边的全部大于轴线值。然后

递归再从左侧选出轴线，从右侧选出轴线，依次比较即可。

注：有一个快排的最坏情况就是如果待排序元素是正序或者逆序，

就会将除轴值以外的元素分到轴值的一边。

1.2快排的示意图



1.3快排的代码实现

```c
#include <stdio.h>

void Show(int* arr, int n)
{
    for (int i = 0; i < n; i++) {
        printf(" %d", arr[i]);
    }
    printf("\n");
}

void QuickSort(int* arr, int start, int end)
{
    int tmp = arr[start];
    int i = start, j = end;

```

```
    while (i < j) {
        while(i<j && tmp < arr[j]){
            j--;
        }
        arr[i] = arr[j];
        while(i<j && tmp >= arr[i]){
            i++;
        }
        arr[j] = arr[i];
    }
    arr[i] = tmp;

    if(start < i-1)
        QuickSort(arr,start,i-1);
    if(j+1 < end)
        QuickSort(arr,j+1,end);
}

int main(int argc, const char* argv[])
{
    int arr[] = { 60, 33, 90, 12, 34, 88 };

    Show(arr, 6);
    QuickSort(arr, 0, 5);
    Show(arr, 6);

    return 0;
}
```

2.查找

2.1查找的概念

查找(或检索)是在给定信息集上寻找特定信息元素的过程。

待查找的数据单位(或数据元素)称为记录。记录由若干数据项(或属性)组成，如学生记录:

其中，“学号”、“姓名”、“性别”、“年龄”等都是记录的数据项。若某个数据项的值能标识(或识别)一个或一组记录，称其为关键字(key)。若一个key能唯一标识一个记录，称此key为主key。如“学号”的值给定就唯一对应一个学生，不可能多个学生的学号相同，故“学号”在学生记录里可作为主key。若一个key能标识一组记录，称此key为次key。如“年龄”值为20时，可能有若干同学的年龄为20岁，故“年龄”可作次key。下面主要讨论对主key的查找。

2.2查找的种类

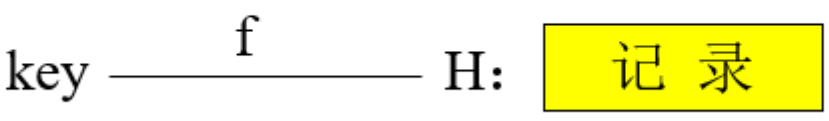
查找方法有顺序查找、折半查找、分块查找、Hash表查找等等

2.3哈希的定义

Hash表的含义

Hash表，又称散列表。在前面讨论的顺序、折半、分块查找和树表的查找中，其ASL的量级在O(n)~O(log2n)之间。不论ASL在哪个量级，都与记录长度n有关。随着n的扩大，算法的效率会越来越低。ASL与n有关是因为记录在存储器中的存放是随机的，或者说记录的key与记录的存放地址无关，因而查找只能建立在key的“比较”基础上。

理想的查找方法是：对给定的k，不经任何比较便能获取所需的记录，其查找的时间复杂度为常数级**O(C)**。这就要求在建立记录表的时候，确定记录的key与其存储地址之间的关系f，即使key与记录的存放地址H相对应：



或者说，记录按key存放。

之后，当要查找key=k的记录时，通过关系f就可得到相应记录的地址而获取记录，从而免去了key的比较过程。这个关系f就是所谓的Hash函数（或称散列函

数、杂凑函数），记为H(key)。它实际上是一个地址映象函数，其自变量为记

录的key，函数值为记录的存储地址（或称Hash地址）。

另外，不同的key可能得到同一个Hash地址，即当keyl≠key2时，可能有H(key1)=H(key2)，

此时称key1和key2为**同义词。**这种现象称为"**冲突**"或"碰撞"，因为一个数据单位

只可存放一条记录。一般，选取Hash函数只能做到使冲突尽可能少，却不能完

全避免。这就要求在出现冲突之后，寻求适当的方法来解决冲突记录的存放问题。

2.4hash中数组大小获取方式

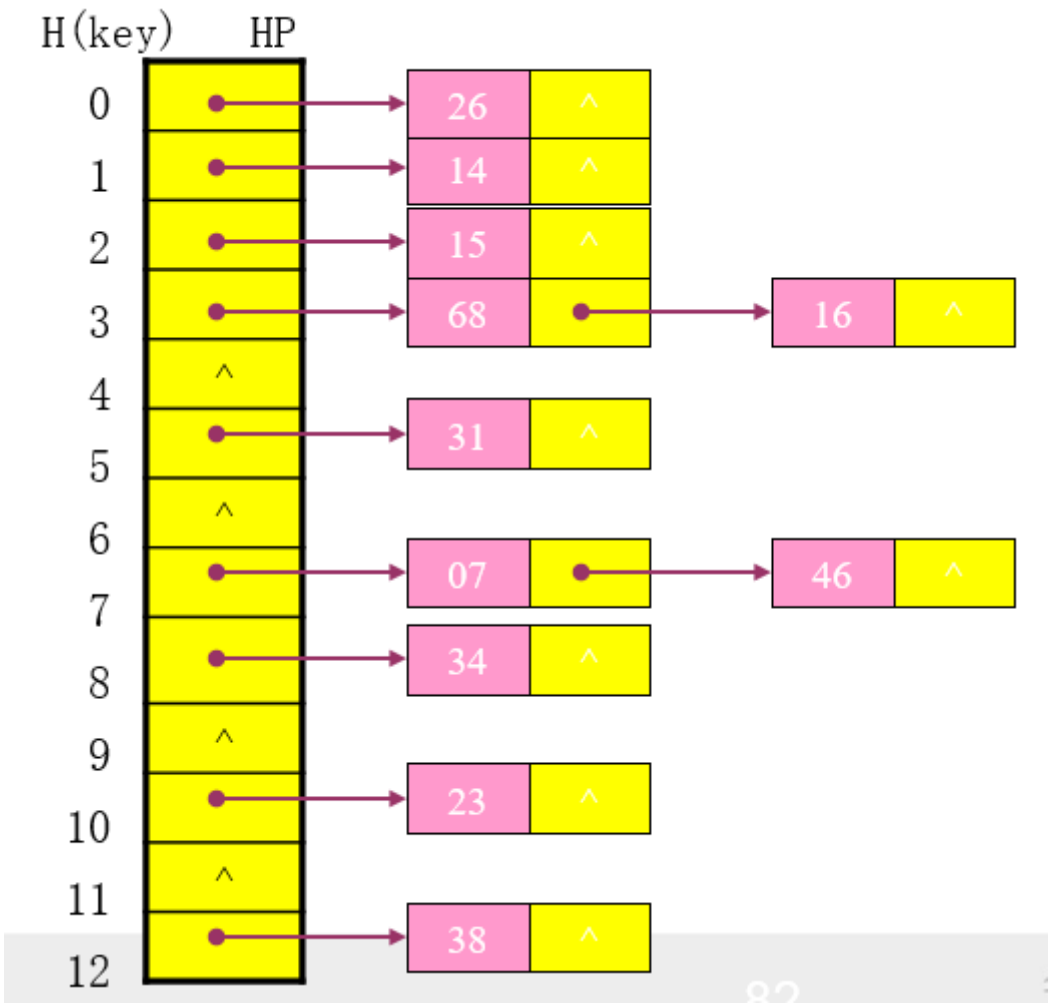
例如在一个数组中有这些成员key集合k={23，34，14，38，46，16，68，15，07，31，26}，

数据个数是n=11。拿n/0.75取余 = 14.6从这个14.6中取出最大的质数，这个质数就是hash数

组定义的成员的个数，结果就是P=13。在向hash表中存数据的时候，使用存放的数组的下

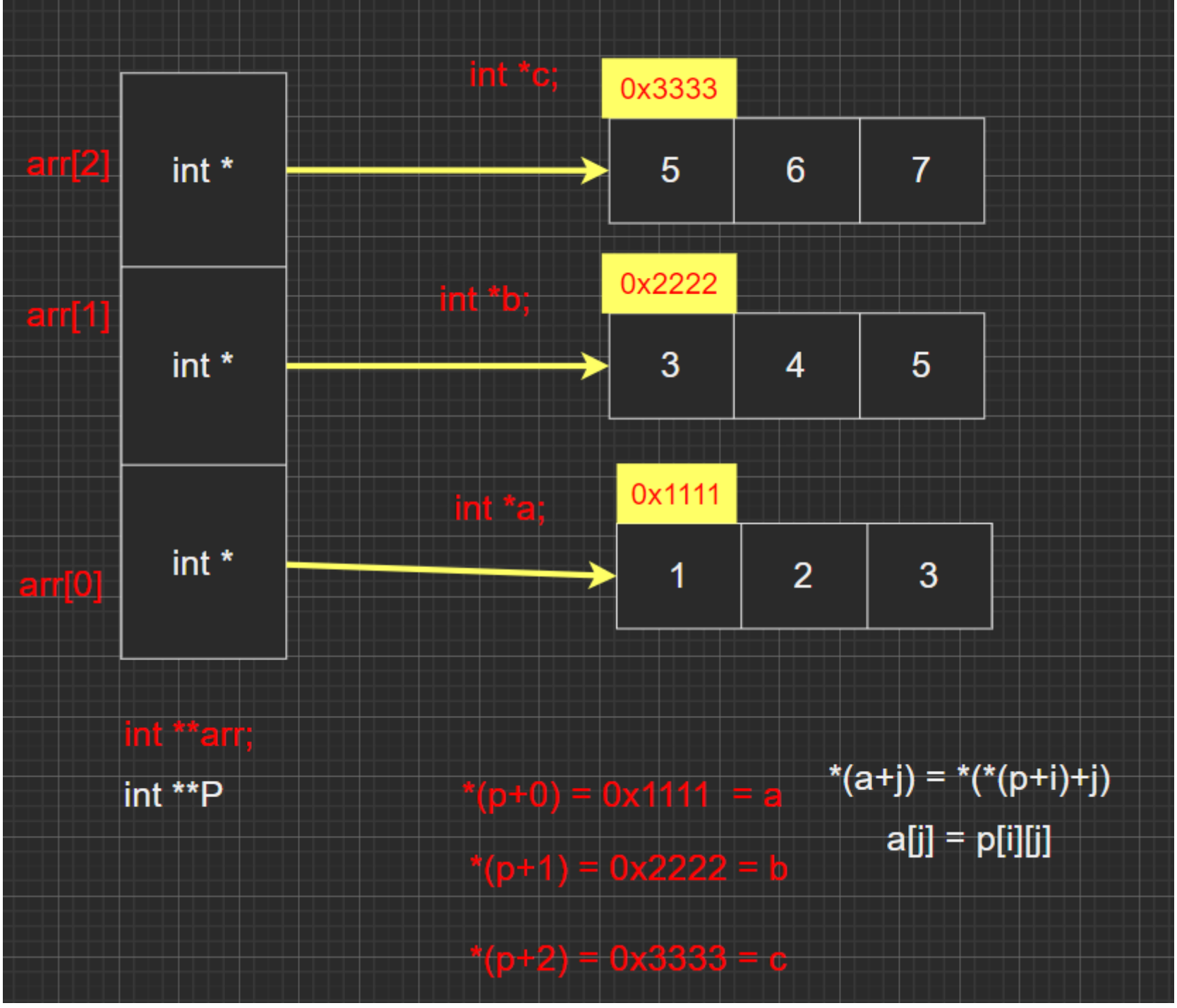
标=key%13

。哈希表的存储结构如下：



##

2.5指针数组和二级指针的关系




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

int** func(void)
{
    int* a = (int*)malloc(3 * sizeof(int));
    for (int i = 0; i < 3; i++) {
        a[i] = i + 1;
    }

    int* b = (int*)malloc(3 * sizeof(int));
    for (int i = 0; i < 3; i++) {
        b[i] = i + 3;
    }

    int* c = (int*)malloc(3 * sizeof(int));

    for (int i = 0; i < 3; i++) {
        c[i] = i + 5;
    }

    int** arr;
    arr = (int **)malloc(3 * sizeof(int*));
    arr[0] = a;
    arr[1] = b;
    arr[2] = c;

    return arr;
}
int main(int argc, const char* argv[])
{
    int **p;
    p = func();

    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            printf("%d\t",*(*(p+i)+j));
        }
        printf("\n");
    }
    return 0;
}
```

2.6哈希查找代码

hash.h

```h
#ifndef __HASH_H__
#define __HASH_H__

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

#define N 13 //  11/0.75 = 14.6取最大质数
#define datatype int
typedef struct node {
    datatype data;
    struct node* next;
} hash_t;

hash_t** HashTableCreate(void);
int HashTableInsertData(hash_t** t, datatype data);
void HashTableShow(hash_t** t);
int HashTableSearchData(hash_t** t, datatype data);
#endif
```

hash.c

```c
#include "hash.h"

hash_t** HashTableCreate(void)
{
    hash_t** t;

    t = (hash_t**)malloc(N * sizeof(hash_t*));
    if (t == NULL) {
        printf("%s malloc memory error\n", __func__);
        return NULL;
    }

    for (int i = 0; i < N; i++) {
        t[i] = (hash_t*)malloc(sizeof(hash_t));
        if (t[i] == NULL) {
            printf("%s malloc memory error\n", __func__);
            return NULL;
        }
        t[i]->data = (datatype)0;
        t[i]->next = NULL;
    }

    return t;
}
int HashMembInsert(hash_t* h, datatype data)
{
    hash_t* tmp;

    tmp = (hash_t*)malloc(sizeof(hash_t));
    if (tmp == NULL) {
        printf("%s malloc memory error\n", __func__);
        return -1;
    }
    tmp->data = data;

    tmp->next = h->next;
    h->next = tmp;
    return 0;
}
int HashTableInsertData(hash_t** t, datatype data)
{
    return HashMembInsert(t[data % N], data);
}
void HashMembShow(hash_t* h)
{
    while (h->next) {
        printf("-%d", h->next->data);
        h = h->next;
    }
}
void HashTableShow(hash_t** t)
{
    for (int i = 0; i < N; i++) {
        printf("t[%d]", i);
        HashMembShow(t[i]);
        printf("-\n");
    }
}
int HashMembSearchData(hash_t* h, datatype data)
{
```

```c
    int pos = 0;
    while (h->next) {
        if (h->next->data == data)
            return pos;
        pos++;
        h = h->next;
    }

    return -1;
}
int HashTableSearchData(hash_t** t, datatype data)
{
    return HashMembSearchData(t[data % N], data);
}
```

main.c

```c
#include "hash.h"

#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof(arr[0]))

int main(int argc, const char* argv[])
{
    hash_t** t;
    int k[] = { 23, 34, 14, 38, 46, 16, 68, 15, 07, 31, 26 };

    t = HashTableCreate();
    if (t == NULL)
        return -1;

    for (int i = 0; i < ARRAY_SIZE(k); i++) {
        HashTableInsertData(t, k[i]);
    }

    HashTableShow(t);

    int num = 46;
    printf("HashTable search t[%d]:data=%d : pos=%d\n",num%N,num,HashTableSearchData(t,num));
    return 0;
}
```