问题

给一个长度为N的数列$\{a_i\}$,询问一个数字b是否存在。

方案一：
直接从头到尾遍历；
复杂度：$O(n)$

方案二：
先sort排序，然后二分查找；
复杂度：除去排序，复杂度为$O(logn)$

方案三：
用一个数字cnt$[x]$表示x在数列中存在多少次；
复杂度：$O(1)$

给一个长度为N的数列$\{a_i\}$,询问一个数字b是否存在。

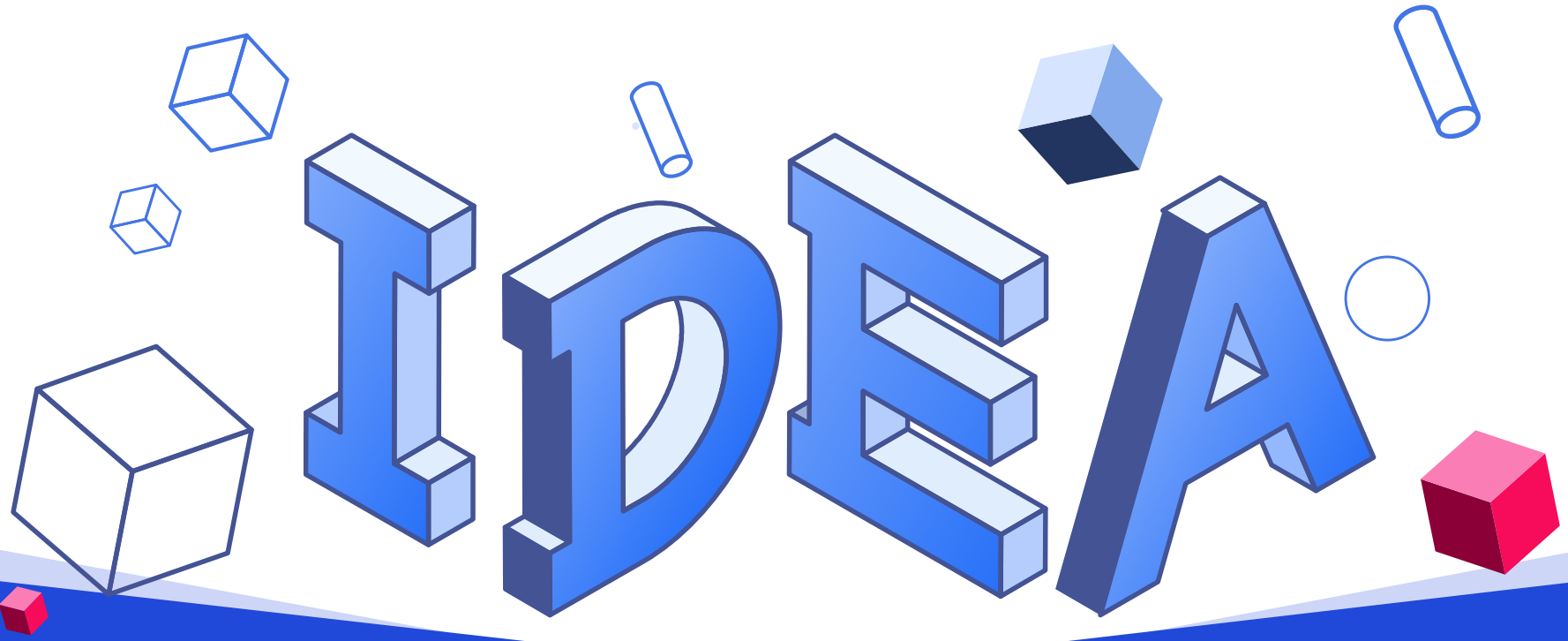
用一个数字cnt$[x]$表示x在数列中存在多少次;
复杂度：$O(1)$

如果a_i是double、string甚至是结构体

如果$a_i \leq 10^{15}$

函数$h()$ → 比较小的整数x

这种方法还可以使用吗?



信息学

哈希算法(Hash)

西南大学附属中学校

信息奥赛教练组

举例

a_i是一个坐标系(包含x,y值)的结构体，且横纵坐标都是1~99的范围，即是：

```
struct point{int x,y;};
```

函数h可以定义为

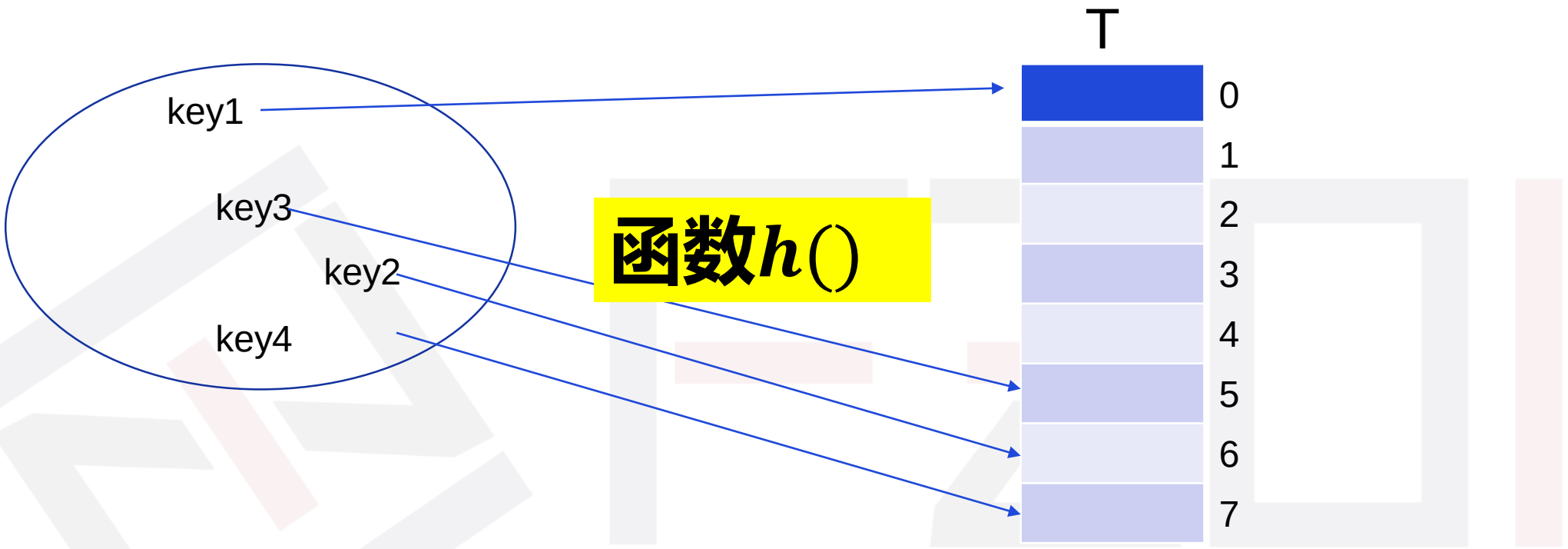
```
int h(point key){
        return key.x*100+key.y;
}
```

每个point类型，都可以通过$h()$函数得到一个可接受的自然数；

即可使用方案三进行存储；

即$cnt[\ h(a[i])\ ] = a[i]$

哈希表

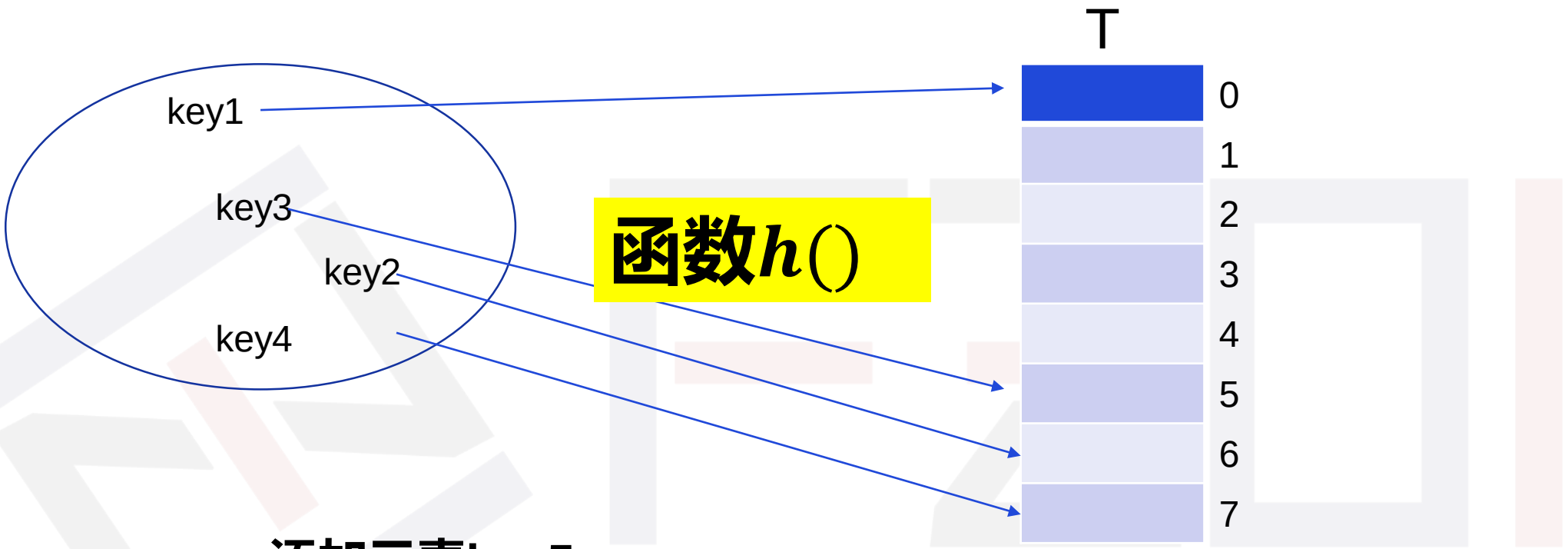


右边为一个长度较大的数组T， 每个元素储存位置的下标等于其通过哈希函数计算出的数值，

该数组即被称为**哈希表（Hash Table）**，也称作散列表

h函数被称为**哈希函数 (Hash Function)**,也称为散列函数

哈希表

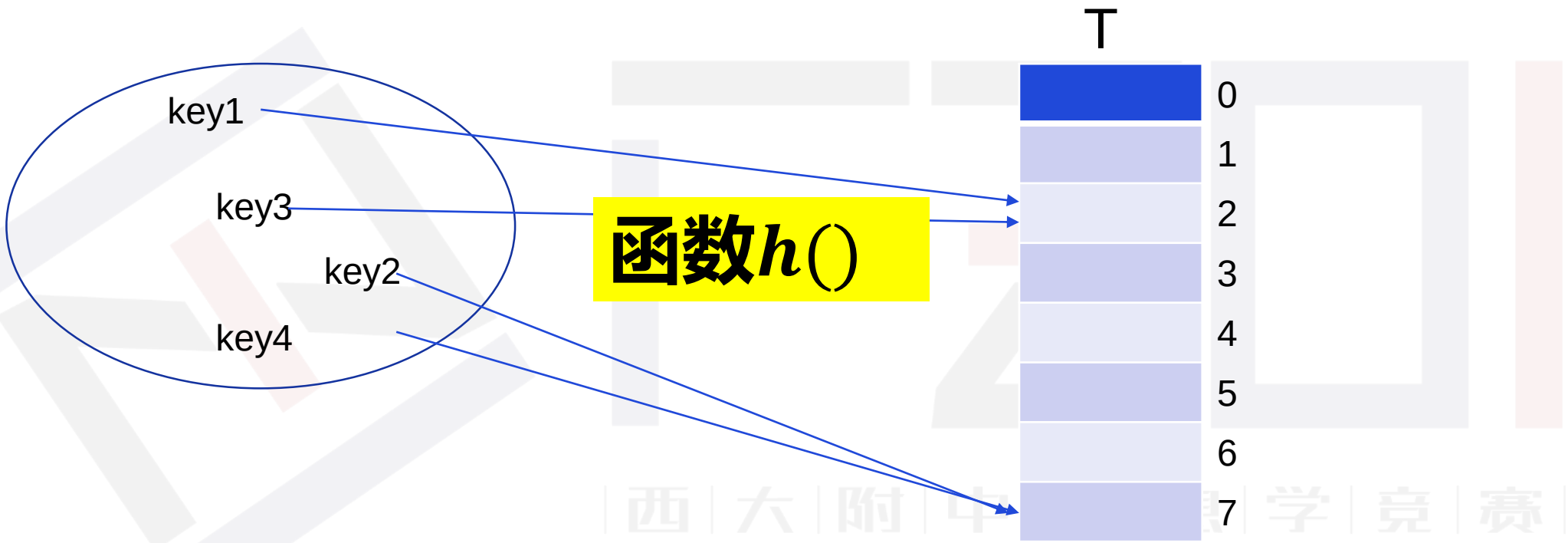


添加元素key5:

T[h(key5)]=key5

删除key4元素:

T[h(key4)] = null



是否可能存在这种情况?

即$key1 != key2, h(key1) == h(key2)$

例如

大整数的哈希函数如下：

```
int h(int key){
    return key % 1000000;
}
```

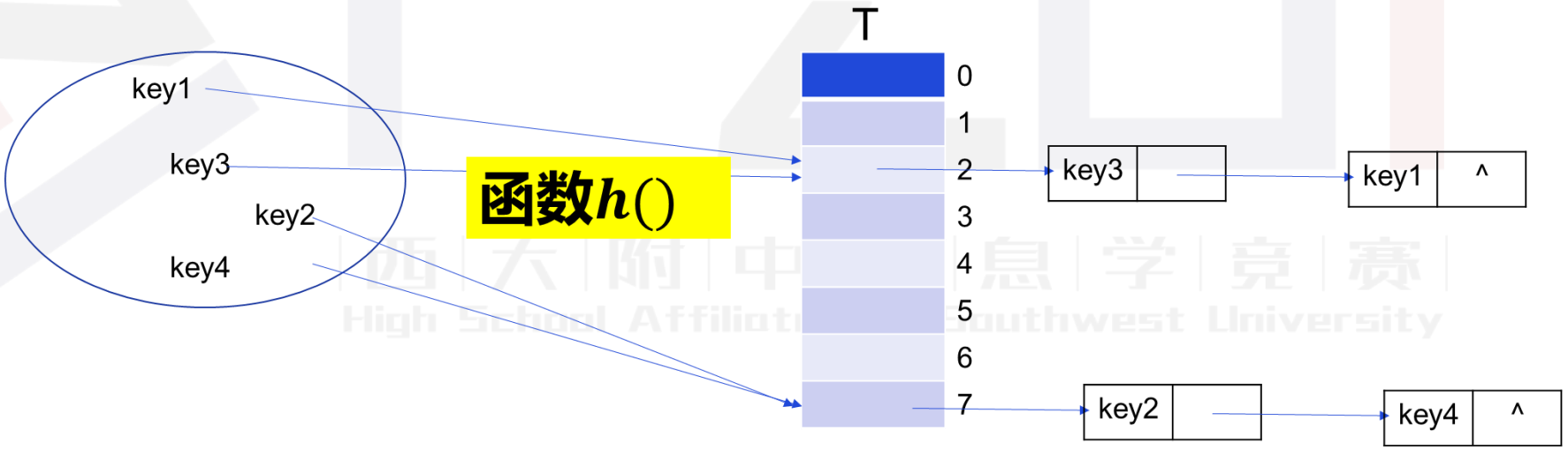
那么显然h(213123456)==h(4322123456)

哈希冲突

哈希冲突解决办法1：拉链法

哈希值冲突的元素存储在同一个链表上，
链表表头保存至哈希表对应下标位置

类似于**邻接表**



解决哈希冲突-拉链法


```
void insert(point key){
      1.新建元素 key;
      2.key.next=T[h(key)].next;
      3.T[h(key)].next=key的地址;
   }
point search(point key){
      直接在链表 T[h(key)] 中进行查找;
    }
void del(point key){
      在链表 T[h(key)] 中进行查找，并删除;
   }
```

拉链法


```
#include <bits/stdc++.h>
using namespace std;

const int N = 100003;
int h[N], e[N], ne[N], idx;

// 拉链法处理哈希冲突

void insert(int x) {
    int k = (x % N + N) % N;
    e[idx] = x; //单链表的头插法
    ne[idx] = h[k];
    h[k] = idx++;
}

//判断x是否已经在hash表中，如果在就返回true，否则返回false
bool search(int x) {
    int k = (x % N + N) % N;
    for (int i = h[k]; i != -1; i = ne[i])
        if (e[i] == x) return true; //在对应的单链表中找是否存在x
    return false;
}
```

```
int main () {
    int n;
    scanf("%d", &n);
    memset(h, -1, sizeof(h)); //单链表中空指针用-1
    while (n--) {
        char op[2];
        int x;
        scanf("%s%d", op, &x);
        if (op[0] == 'I') insert(x);
        else {
            if (search(x)) puts("Yes");
            else puts("No");
        }
    }
    return 0;
}
```

哈希冲突解决办法2：开放地址法

所有的内容都储存在T中，插入时，当遇到冲突，需要连续地查找后面的元素，该操作称为**探查**，直到找到一个空的位置

但是检测的顺序并不一定是$0,1,2,3,\ldots,\mathrm{m}-1$(其中m为T的大小)，最佳的方案是要根据元素的特性来确定，因此哈希函数就需要第二个参数，即为探查的次数。

h(key,i)，key表示其元素，i表示第i+1次探查

那么其探查的顺序即为：$<h(\mathrm{key},0),\mathrm{h}(\mathrm{key},1),\ldots,\mathrm{h}(\mathrm{key},\mathrm{m}-1)>$

解决哈希冲突-开放地址法


```
 1  int insert(point key){
 2      int j;
 3      for(int i=0;i<m;i++){
 4              j=h(key,i);
 5              if(T[j]==null){
 6                      T[j]=key;
 7                      return j;
 8              }
 9      }
10      return -1;
11  }
12  int search(point key){
13      int j;
14      for(int i=0;i<m;i++){
15              j=h(key,i);
16              if(T[j]==key)
17                      return j;

18      }
19      return -1;
20  }
```

探查函数h该如何设计?

(1)线性探查。

线性探查的哈希函数形式为：

$$h(key, i) = (h'(key) + i) mod\ m$$

优点：

好写

想一想：存在什么缺点？

(1)线性探查。

线性探查的哈希函数形式为：

$$h(key, i) = (h'(key) + i) mod\ m$$

$$h'(key1) = 1, h'(key2) = 2, h'(key3) = 3, h'(key4) = 1$$

插入顺序：$< key1, key2, key3, key4 >$

将会怎么插入？

0	1	2	3	4	5	6	7	8	9

解决哈希冲突-开放地址法

(1)线性探查。

线性探查的哈希函数形式为：

$$h(key, i) = (h'(key) + i) mod\ m$$

$$h'(key1) = 1, h'(key2) = 2, h'(key3) = 3, h'(key4) = 1$$

插入顺序：$< key1, key2, key3, key4 >$

将会怎么插入？

0	1	2	3	4	5	6	7	8	9
	Key1	Key2	Key3	Key4					

只有一个元素存在冲突

解决哈希冲突-开放地址法

(1)线性探查。

线性探查的哈希函数形式为：

$$h(key, i) = (h'(key) + i) mod\ m$$

$$h'(key1) = 1, h'(key2) = 2, h'(key3) = 3, h'(key4) = 1$$

插入顺序： $< key1, key4, key2, key3 >$

将会怎么插入？

0	1	2	3	4	5	6	7	8	9

解决哈希冲突-开放地址法

(1)线性探查。

线性探查的哈希函数形式为：

$$h(key,i) = (h'(key) + i) mod\ m$$

$$h'(key1) = 1, h'(key2) = 2, h'(key3) = 3, h'(key4) = 1$$

插入顺序： $< key1, key4, key2, key3 >$

将会怎么插入？

0	1	2	3	4	5	6	7	8	9
	Key1	Key4	Key2	key3					

三个元素存在冲突

(2)二次探查

二次探查的函数形式为：

$$\mathrm{h(key,i)} = (\mathrm{h'(key)} + \mathrm{c}_1 i + c_2 i^2) mod\ m$$

h'和线性探查的内容相同，c_1, c_2是非负的辅助常量，这种函数比线性探查的效果要好得多，但是要注意c_1, c_2需要精心构造，要保证对于$\mathrm{h(key,i)}$能取到$[0, \mathrm{m})$的所有值。

(3)双重哈希

$$\mathrm{h(key,i)} = \big(h_1(key) + i \times h_2(key)\big) mod\ m$$

其中h_1和h_2是构造出来的两个不同的哈希函数，此处每次的偏移量都会因key值的不同而改变，所以能做到最好的均匀分布效果，也是开地址法**最推荐的写法**。

冲突解决方法	拉链法	开放地址法
途径	邻接表	设计哈希函数： 1. 线性探测 2. 二次探测 3. 双重哈希
空间	需要额外空间	不需要额外空间
时间	取决于哈希函数的构造	

哈希函数设计原则

哈希函数设计原则：

1、均匀分布；

2、尽量将元素所有信息都用到；

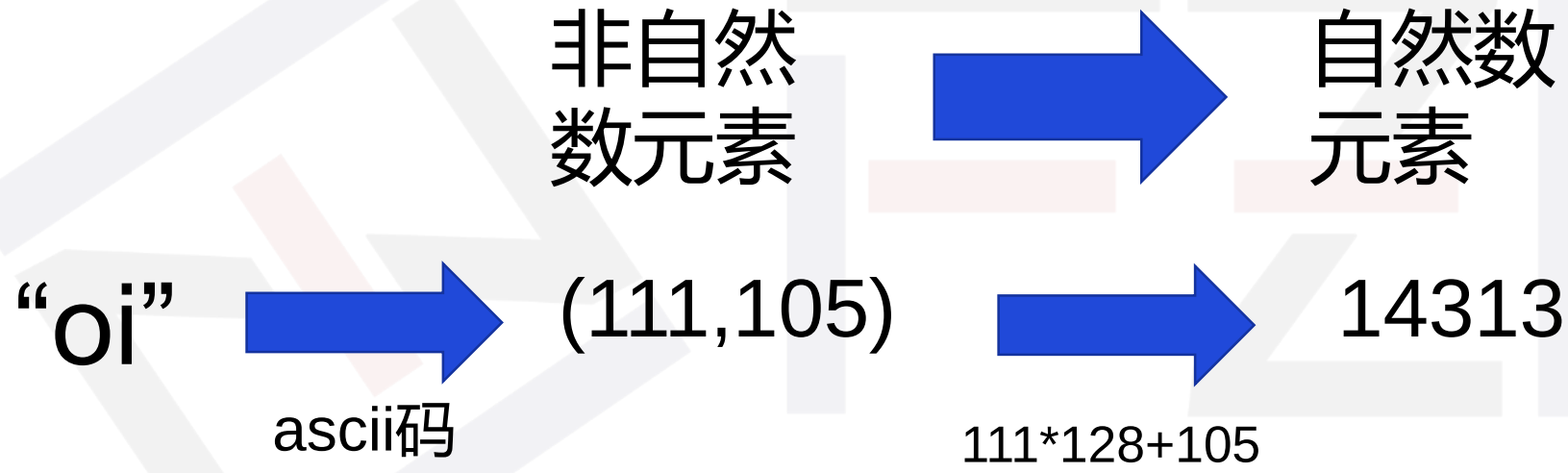
非自然数元素 → 自然数元素 → 可接受自然数元素

取余法：

h(key) = key mod m

建议m为不接近2的幂的素数

哈希函数设计原则



Sumsets

给出一个整数集合s，找到集合中最大的d，让等式a+b+c=d成立，

其中，a,b,c,d是集合S中不同的元素。

输入
有多组数据，每组数据第一行为n(1<=n<=1000),表示S中有n个数据，接下来n行，
每行一个整数（-536870912~+536870911），最后n=0时结束

输出
每组数据输出一行，输出d
如果无解则输出 no solution。

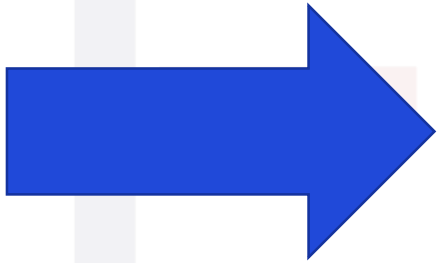
第一想法：

枚举a,b,c,查找d是否存在；

时间复杂度$O(n^4)$

发现：

只支持枚举两个数字



a+b+c=d

a+b=d-c

分别枚举左右两边的数字，然后查找是否存在

利用哈希

Sumsets

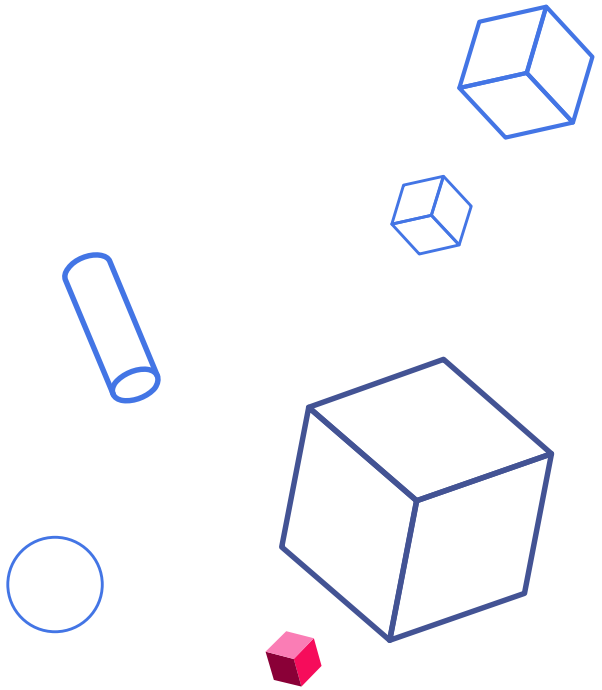

```
memset(head , -1 , sizeof(head));
k=0;
for(int i=0 ; i<n ; i++) scanf("%d" , &a[i]);
sort(a , a+n , cmp);
for(int i=0 ; i<n ; i++){
    for(int j=i+1 ; j<n ; j++){
        int v = a[i]+a[j];
        insertHash(v , i , j);
    }
}
int ret = INT_MIN;
for(int i=0 ; i<n ; i++){
    for(int j=0 ; j<n ; j++){
        if(i==j) continue;
        int v = a[i]-a[j];
        if(searchHash(v , i , j)){
            ret = max(ret , a[i]);
            break;
        }
    }
    if(ret!=INT_MIN) break;
}
if(ret==INT_MIN) printf("no solution\n");
else printf("%d\n" , ret);
```

```
void insertHash(int key , int x , int y)
{
    int pos = ((key<0)?-key:key)%MOD;
    _hash[k].val = key , _hash[k].next = head[pos] , _hash[k].x=x , _hash[k].y=y;
    head[pos] = k++;
}

bool searchHash(int key , int a , int b)
{
    int pos = ((key<0)?-key:key)%MOD;
    for(int i=head[pos] ; ~i ; i=_hash[i].next){
        if(key == _hash[i].val){
            if(a==_hash[i].x||a==_hash[i].y||b==_hash[i].x||b==_hash[i].y) continue;
            return true;
        }
    }
    return false;
}

bool cmp(int a , int b)
{
    return a>b;
}
```

字符串哈希



图书馆管理

【问题描述】

图书管理是一件十分繁杂的工作，图书馆每天都会有许多新书缴入，为了更方便管理图书（以便于帮助客人快速查找是否有他们所需要的书），我们需要设计一个图书管理系统，该系统需要支持两种操作：

1）add(s)，表示新加入一本书名为s的图书；

2）find(s)，表示查询是否存在一本书名为s的图书；

【输入格式】

第一行包括一个正整数n（n≤30000），表示操作数。

以下n行，每行所给出两个操作中的一种，指令格式为：

add s

find s

在书名s与指令间有一个空格，保证书名长度都不超过200，可以加上读入数据是准确无误的，注意s的生成为随机。

字符串hash

BKDRHash：

$$\mathrm{h(s)} = \sum_{i=0}^{|s|-1} s_i \mathrm{p}^{|s|-i-1} \ mod\ M$$

$$H(S) = (S_1 \cdot P^{n-1} + S_2 \cdot P^{n-2} + \ldots + S_{n-1} \cdot P^1 + S_n \cdot P^0) \mod M$$

S为字符串，M为一个大质数，P为一个大于字符集（为s中所有会出现的字符组成的集合)大小的正整数。该函数本质为将字符串看做p进制数下的整数在模M意义下的值。

例如：

设p= 131, $M = 10^9 + 7$,当$s =$ oi

$h(s) = (' o' \times p^1 + ' i' \times p^0) mod\ M = (111 \times 131 + 105) mod\ M = 14646$

字符串hash

BKDRHash：

$$\mathrm{h(s)} = \sum_{i=0}^{|s|-1} s_i \mathrm{p}^{|s|-i-1} \; mod \; M$$

若我们提前将每个字符串前缀的哈希值求出，可以$O(1)$算出子串的哈希值，公式如下：

$$\mathrm{h(s[l,r])} = \left(h(s[0,r]) \; - h(s[0,l]) \times p^{r-l}\right) mod \; M$$

BKDR Hash


```
//预处理p^n
pn[0] = 1;
for (int i = 1; i < N; i++)
   pn[i] = pn[i - 1] * p;
```

```
//求解主串hash值,前缀和hash
for (int i = 1; i <= strlen(str); i++)
            Hash[i] = (Hash[i - 1] * p + (LL)(str[i] - 'A' + 1)) %Q;
```

```
//起点为i，长度为len的子串hash值
Hash[i+len-1] - Hash[i-1] * pn[len]
```


各种hash函数测评：https://www.cnblogs.com/uvsjoh/archive/2012/03/27/2420120.html

字符串hash

BKDRHash：

$$h(s) = \sum_{i=0}^{|s|-1} s_i p^{|s|-i-1} \ mod \ M$$

非自然数元素 → 自然数元素 → 可接受自然数元素

当字符串转换成自然数后，可以帮助我们进行快速字符串匹配

该自然数可以称作字符串的**特征值**（判断字符串是否相等）

如果需要将其作为下标，数字仍然太大，还需要进一步转换

字符串hash

BKDRHash：

$$h(s) = \sum_{i=0}^{|s|-1} s_i b^{|s|-i-1} \ mod \ p$$

非自然数元素 → 自然数元素

当字符串转换成自然数后，可以帮助我们进行快速字符串匹配（判断字符串是否相等）

两个字符串的哈希值相等，两个字符串并不一定相等；
但是如果两个字符串的哈希值不相等，两个字符串一定不相等；

字符串hash

BKDRHash：

$$h(s) = \sum_{i=0}^{|s|-1} s_i p^{|s|-i-1} \ mod \ M$$

双哈希：

即选取两组整数P1M1、P2M2，构造出两个哈希函数，让每个字符串对应两个特征值，

注意：

1. P1与P2必须得互质
   常见的取值为13,131,13331等
2. M1和M2一般选取大质数
   例如10^9+7,10^9+9两个质数是很常用的

优势：

1、匹配计算快；

2、借助之前公式，可快速计算子串；

劣势：

并不一定能百分之百正确；

图书馆管理


```
int main()
{
    scanf("%d", &n);
    while (n--) {
        scanf("%s%s", a, s + 1);
        ll sum1 = 0, sum2 = 0;

        sum1 = hash1();
        sum2 = hash2();
        if (a[0] == 'a') {
            clil1[sum1] = 1;
            clil2[sum2] = 1;
        } else if (a[0] == 'f') {
            if (clil1[sum1] == 0 || clil2[sum2] == 0)
                printf("no\n");
            else
                printf("yes\n");
        }
    }
    return 0;
}
```

图书馆管理


```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll unsigned long long
const int p1 = 131, p2 = 113, mod1 = 19999983, mod2 = 20000093;
int n;
char a[20], s[521];
bool clil1[mod1 + 17], clil2[mod2 + 17];

int hash1(){
    int m = strlen(s + 1);
    ll sum = 0;
    for (int i = 1; i <= m; i++) {
        sum = sum * p1 + (ll)(s[i] - 'A' + 1) % mod1;
    }
    return sum % mod1;
}
int hash2(){
    int m = strlen(s + 1);
    ll sum = 0;
    for (int i = 1; i <= m; i++) {
            sum = sum * p2 + (ll)(s[i] - 'A' + 1) % mod2;
    }
    return sum % mod2;
}
```

例：Oulipo

给定两个串S1，S2，只有大写字母，求S1在S2中出现了几次。

输入
输入T组数据，每组数据两个串S1，S2.

strlen(S1)<=10^4，strlen(S2)<=10^6，

输出
对于每组数据，输出答案。

样例
样例输入1
3
BAPC
BAPC
AZA
AZAZAZA
VERDI
AVERDXIVYERDIAN
样例输出1
1
3
0

Oulipo核心代码


```
int len1, len2;
len1 = strlen(s1 + 1), len2 = strlen(s2 + 1);

//主串的hash,s2
for (int i = 1; i <= len2; i++)
  Hash[i] = Hash[i - 1] * p + (LL)(s2[i] - 'A' + 1);

//子串的hash,s1
  LL subhash = 0;
  for (int i = 1; i <= len1; i++)
     subhash = subhash * p + (LL)(s1[i] - 'A' + 1);

//匹配hash值
  int ans = 0;
  for (int i = 0; i <= len2 - len1; i++)
      if (subhash == Hash[i + len1] - Hash[i] * pn[len1])
            ans++;
```

小结

哈希算法需要合理的设计哈希函数，并能够恰当的处理冲突，推荐拉链法

字符串的哈希值，准确来说是特征值，用于判断是否匹配

字符串常用哈希算法是BKDR

$$\mathrm{h(s)} = \sum_{i=0}^{|s|-1} s_i b^{|s|-i-1} \ mod \ p$$

Thanks

For Your Watching

FZOI
西大附中信息学竞赛
High School Affilited to Southwest University