算法基础（八）：哈希法与字符串前缀哈希法

哈希

基本思想：

总的来说，哈希就是将一个很大值域的一堆数据，映射到值比较小的一些数据

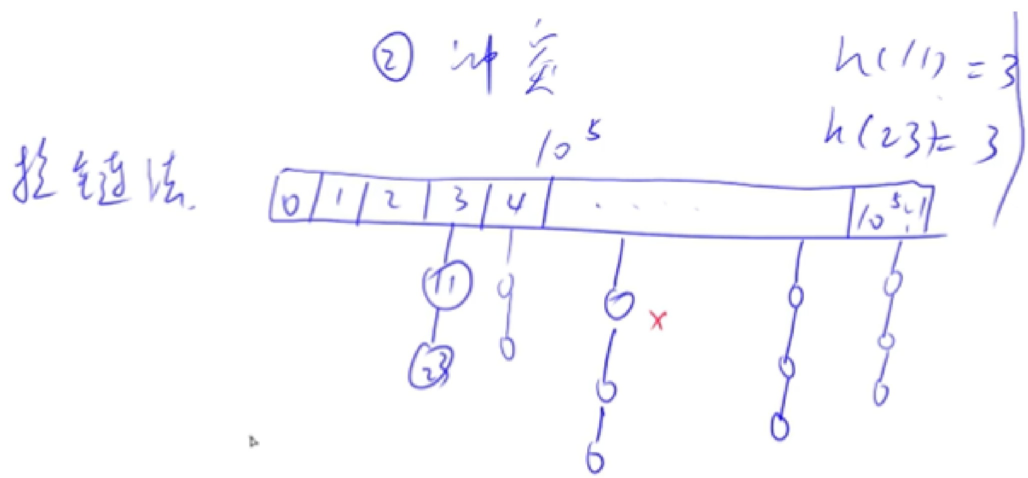
比如将值域为`1 ~10^9`的一组数（这组数的个数不会大于`10^5`），映射到`0 ~ 10^5`，比如这个数`x`通过一个函数`h(x)`映射到`0 ~ 10^5`，`h(x)`称为哈希函数。一般而言哈希函数是`mod i`即模某一个数，`i`一般是质数，并且离`2`的整次幂尽可能远

在映射过程中会出现冲突，就是说不同的`x1，x2`通过哈希函数会映射到同一个数，这时我们需要设计方法来解决冲突，按照解决冲突不同的方式哈希分为：

1. 开放定址法
2. 拉链法

拉链法：

拉链法就是开一个很大数组（槽），对数组中的某一个值`i`，若不同的数都映射到`i`出现冲突，则在后面拉一条链将这些数存起来，如下图：



数字`11`和`23`都映射为`3`，那么就在`3`后面拉一条链来存`11`和`23`，若想删除链上的某一个数，则直接标记一下，而不是真的删除这些数

哈希的基本操作也就是插入和查找两个

代码实现：

维护一个集合，支持如下几种操作：

1. `I x`，插入一个数 x；
2. `Q x`，询问数 x 是否在集合中出现过；

现在要进行 N 次操作，对于每个询问操作输出对应的结果。

输入格式

第一行包含整数 N，表示操作数量。

接下来 N 行，每行包含一个操作指令，操作指令为 `I x`，`Q x` 中的一种。

输出格式

对于每个询问指令 `Q x`，输出一个询问结果，如果 x 在集合中出现过，则输出 `Yes`，否则输出 `No`。

每个结果占一行。

数据范围

$1 \le N \le 10^5$
$-10^9 \le x \le 10^9$

输入样例：

```
5
I 1
I 2
I 3
Q 2
Q 5
```

输出样例：

```
Yes
No
```

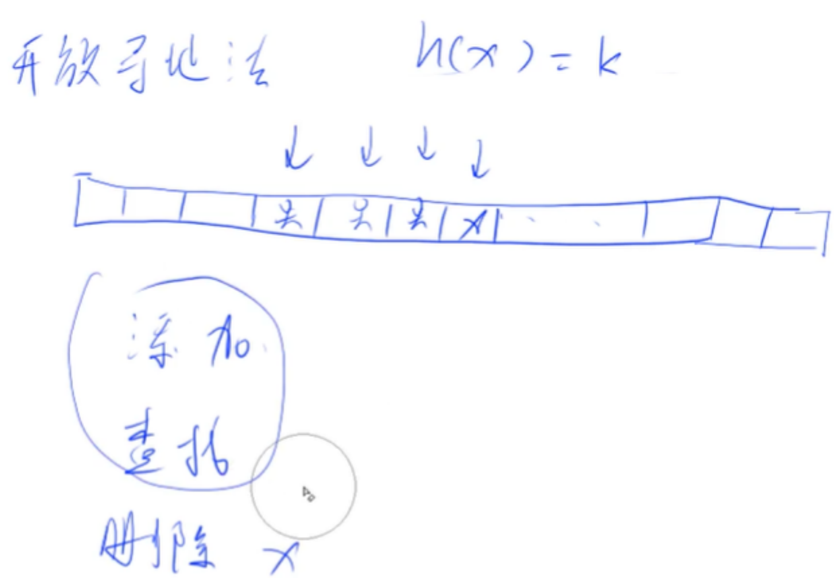
```
#include<iostream>
#include<cstring>
using namespace std;

const int N = 100003;//取大于100010的第一个质数

int h[N], e[N], ne[N], idx;

void insert(int x)
{
    //x可能是负数，这一步是为了确保取模之后一定映射到正数
    int k = (x % N + N) % N;


    //插入一个值到链的头部
    e[idx] = x;
    ne[idx] = h[k];
    h[k] = idx ;
    idx ++;
    //一个很巧妙的思想是，由于链的地址关系，数组e[N], ne[N]被拆成了很多的链
}

bool find(int x)
```

```
{
    int k = (x % N + N) % N;
    for(int i = h[k]; i != -1; i = ne[i])
    {
        if(e[i] == x) return true;
    }
    return false;
}

int main()
{
    int n;
    cin >> n;
    //h[i]的值是取模为i的所有数构成的链的头指针
    //一开始所有的指针指向-1，为空
    //图的存储结构也是这样的
    memset(h, -1, sizeof h);
    while(n --)
    {
        char op[2];
        int x;
        scanf("%s%d", op, &x);

        if(*op == 'I') insert(x);
        else
        {
            if(find(x)) puts("Yes");
            else puts("No");
        }
    }
    return 0;
}
```

开放寻址法：

开一个很大的数组，一般是题目中数的个数的`2 ~ 3`倍

仍然是先取模，如果有冲突的话，就往后存一次，直到找到一个没有被存的地方，查找这个数也是这样，从第`k`开始找，如果有数并且是`x`就找到，如果有数不是`x`，就继续往后找，如果没数，就表示`x`不存在



代码实现：

```cpp
#include<iostream>
#include<cstring>
using namespace std;
//大于2000000的最小质数
//这里数组的大小开到原来数的2~3倍可以减少很多冲突
//0x3f3f3f3f这个数是大于10^9的，这里用来表示无穷大，表示这个位置没有被存数
const int N = 200003, null = 0x3f3f3f3f;

int h[N];

int find(int x)
{
    int k = (x % N + N) % N;
    //当这个位置有数并且不是x的时候一直往后循环
    //由于N是所有数的个数的2~3被所以h[N]不可能被全部填完，所以while一定会退出循环
    while(h[k] != null && h[k] != x)
    {
        k++;
        //依次往后递增到末尾，继续从0开始然后向后递增
        if(k == N) k = 0;

    }
    //退出循环有两种可能
    //1. 找到了x，k是x的存储位置
    //2. 没找到x，也就是位置k为空，应该存x
    //我们统一返回k
    return k;
}


int main()
{
    int n;
    cin >> n;
    memset(h, 0x3f, sizeof h);

    while(n -- )
    {
        char op[2];
        int x;
        scanf("%s%d", op, &x);
        //返回位置k
        int k = find(x);
        //如果是插入，则之前必然不存在x，此时k是x应该插入的位置
        if(op[0] == 'I')
        {
            h[k] = x;
        }
        //如果是查找，则返回的可能是空位置，也可能是x的位置
        else
        {
            //空位置
            if(h[k] == null) puts("No");
            //不是空位置
            else puts("Yes");
        }
    }
    return 0;
}
```

字符串前缀哈希法

基本思想：

全称字符串前缀哈希法，把字符串变成一个`p`进制数字（哈希值），实现不同的字符串映射到不同的数字。
对形如$X_1X_2X_3\cdots X_{n-1}X_n$的字符串,采用字符的`ascii`码乘上`P`的次方来计算哈希值。

映射公式$(X_1 \times P^{n-1} + X_2 \times P^{n-2} + \cdots + X^{n-1} \times P^1 + X_n \times P^0) mod Q$
注意点：

1. 任意字符不可以映射成`0`，否则会出现不同的字符串都映射成`0`的情况，比如`A`,`AA`,`AAA`皆为`0`
2. 冲突问题：通过巧妙设置`P (131 或 13331)`,`Q (2^64)`的值，一般可以理解为不产生冲突。
3. 我们用一个数组`h[]`来存放字符的前缀哈希值，比如`h[3]`表示$X_1X_2X_3$的哈希值
   1. 我们将`h[]`数组声明为`unsigned long long`，这样当字符串相加后的值大于`Q`时就相当于自动取模了
   2. 我们用`s[]`表示单个字符的哈希值
   3. **证明**：前缀和公式：`h[i+1] = h[i] * P + s[i+1] i∈[0,n-1]`
      1. $h[i] = (X_1 \times P^{i-1} + X_2 \times P^{i-2} + \cdots + X_i \times P^0) mod Q$
      2. $s[i+1] = X_{i+1} mod Q$
      3. $h[i+1] = (X_1 \times P^i + X_2 \times P^{i-1} + \cdots + X_i \times P^1 + X_{i+1}) mod Q$
      4. $h[i] * P + s[i+1] = ((X_1 \times P^{i-1} + X_2 \times P^{i-2} + \cdots + X_i \times P^0) mod Q * P + X_{i+1} mod Q)$
      5. 由于数据设置为`unsigned long long`，所以`h[i] * P + s[i]`的中间结果总是`mod Q`的，故有
      6. $$h[i] * P + s[i+1] = (((X_1 \times P^{i-1} + X_2 \times P^{i-2} + \cdots + X_i \times P^0) mod Q * P mod Q) mod Q + X_{i+1} mod Q) mod Q$$
      7. 由于取模公式：$(a * b) mod = (a mod * b mod) mod, (a + b) mod = (a mod + b mod) mod$
      8. 所以`h[i+1] = h[i] * P + s[i+1]`

比如，$X_1X_2X_3\cdots X_{n-1}X_n$中$X_1X_2$的前缀和为`h[2]`，当要求`h[3]`的时候字符串是$X_1X_2X_3$很显然是先让`h[2]`左移一位然后加上$X_3$的哈希值，就这样递推求，一直到`h[n]`，就是$X_1X_2X_3\cdots X_{n-1}X_n$的哈希值

问题是比较不同区间的子串是否相同，就转化为对应的哈希值是否相同。求一个字符串的哈希值就相当于求前缀和，求一个字符串的子串哈希值就相当于求部分和。

那么求部分和的时候比如求$X_1X_2X_3\cdots X_l \ldots Xr \ldots X_{n-1}X_n$的子串$X_l \ldots Xr$的哈希值，注意子串的哈希值指的是字串单独作为一个字符串的时候的哈希值，其中X_r的权值是`0`，而不是在原来的完整串中的权值

那么我们需要用`h[r]`与`h[l-1]`来做，`h[l - 1]`是$X_1X_2X_3\cdots X_{l-1}$的哈希值，我们需要将其权值提升到跟`h[r]`中一样，即变成$X_1X_2X_3\cdots X_{l-1}0000\ldots$，后面添加`r-(l-1)`个零使得位数与`h[r]`一样，然后用$X_1X_2X_3\cdots X_l \ldots Xr$一减即得

所以可以得到部分和公式为`h[l...r] = h[r] - h[l-1] * P^(r - l + 1)`

证明：

1. $h[r] = (X_1 \times P^{r-1} + X_2 \times P^{r-2} + \cdots + X_r \times P^0) mod Q$
2. $h[l-1] * P^{r-l+1} = (X_1 \times P^{l-1-1} + X_2 \times P^{r-l+1-2} + \cdots + X_{l-1} \times P^0) mod Q * P^{r-l+1} mod Q$
3. $h[l \ldots r] = (X_l \times P^{r-l} + X_{l+1} \times P^{r-l-1} + \cdots + X_r \times P^0) mod Q$
4. 同上面证明，后续证明略

总而言之，当我们需要快速判断两个字符串是否相等的时候就可以用这个做法

代码实现：

给定一个长度为 n 的字符串，再给定 m 个询问，每个询问包含四个整数 l_1, r_1, l_2, r_2，请你判断 $[l_1, r_1]$ 和 $[l_2, r_2]$ 这两个区间所包含的字符串子串是否完全相同。

字符串中只包含大小写英文字母和数字。

输入格式

第一行包含整数 n 和 m，表示字符串长度和询问次数。

第二行包含一个长度为 n 的字符串，字符串中只包含大小写英文字母和数字。

接下来 m 行，每行包含四个整数 l_1, r_1, l_2, r_2，表示一次询问所涉及的两个区间。

注意，字符串的位置从 1 开始编号。

输出格式

对于每个询问输出一个结果，如果两个字符串子串完全相同则输出 `Yes`，否则输出 `No`。

每个结果占一行。

数据范围

$1 \le n, m \le 10^5$

输入样例：

```
8 3
aabbaabb
1 3 5 7
1 3 6 8
1 2 1 2
```

输出样例：

```
Yes
No
Yes
```

```
#include<iostream>
#include<algorithm>
using namespace std;

typedef unsigned long long ULL;
const int N = 100010, P = 131;//P用来表示进制

int n, m;
char str[N];//str用来存放字符串，从1开始，便于公式计算，道理见前缀和
//h[N]用来存放前i个字符串的哈希值
//p[N]用来存放对应的权值，p[i]就是P^i，便于后面的计算
ULL h[N], p[N];//在上面的证明中权值是自动modULL的，所以必须声明为UNLL的类型

//p[]从0开始时因为p[0]代表的是P的0次方，是有意义的
//h[]从1开始时因为递推公式h[i] = h[i - 1] * P + str[i]
//若从0开始的话会出现h[-1]这种情况得单独特判处理，所以从1开始所有的数的递推形式一致
//p[]和h[]两个数组代表不同的意义，其实没有比较性
int get(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}

```

```
int main()
{
    scanf("%d%d%s", &n, &m, str+1);

    p[0] = 1;

    for(int i = 1; i <= n; i ++)
    {
        p[i] = p[i - 1] * P;
        h[i] = h[i - 1] * P + str[i];//利用公式递推进行哈希值的计算
    }

    while(m --)
    {
        int l1, r1, l2, r2;
        scanf("%d%d%d%d", &l1, &r1, &l2, &r2);

        if(get(l1, r1) == get(l2, r2)) puts("Yes");
        else puts("No");
    }

    return 0;
}
```

为什么说这个算法逆天呢， 因为它可以在`O(1)`的时间内进行两个字符串的比较，比`KMP`还`nb`