哈希表/字符串哈希

2023年11月26日　　16:16

一、存储结构

将复杂的数据结构映射到从0-N的数字

例如将0~10^9　　->　　0~10^5

维护一个集合，支持如下几种操作：

`I x`，插入一个整数 `x`；

`Q x`，询问整数 `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

$1 \le N \le 10^5$

$-10^9 \le x \le 10^9$

思路：如果使用STL中的set，查找的时间复杂度为O(nlogn)（logn是因为set底层使用红黑树优化），即便如此时间复杂度还是很高。如果将输入的x映射到10^5以内，此时映射值作为数组下标可以直接存储，那么查询的时间复杂度为O(1)　　**(离散化是一种特殊的哈希，需要保序)**

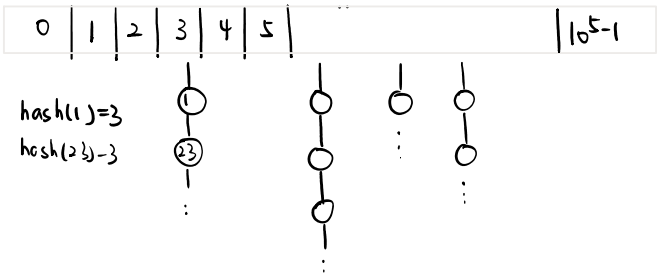
常见的写法：取模、解决冲突

根据解决冲突的方式分为两种：

1、开放寻址法

2、拉链法

二、拉链法实现



1、思路：

插入 ：**将计算模之后再对应的数字上拉一条链**

查找：求模，在对应的模的链上查找有无该值

删除：按照查找的思路找x，打上标记（不会真的删除）

期望算法：时间复杂度为O(1)

2、实现

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

const int N = 100003;                  // 求模操作的底数应为质数，且距离2的幂次越远越好

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

void insert(int x)
{
    int k = (x%N + N)%N;               // 考虑到负数求模的结果仍是负数
    e[idx] = x;
    ne[idx] = h[k];
    h[k] = idx++;
}

bool find(int x)
{
    int k = (x%N + N)%N;
    for(int i = h[k];i!=-1;i = ne[i])    // 链表的结束标志为ne[i] = -1
    {
        if(e[i] == x) return true;
    }
    return false;
}

int main()
{
   scanf("%d",&n);
   memset(h,-1,sizeof(h));                 // 对应链表的结束标志，需要进行初始化
   for(int i = 0;i < n;i++)
   {
       char opt[2];
       int x;
       scanf("%s%d",opt,&x);
       if(opt[0] == 'I')
       {
           insert(x);
       }
       else
       {
           if(find(x)) puts("Yes");
           else puts("No");
       }
   }
```

```
    return 0;
}
```

三、开放寻址法

1、思路：

只开一维数组，但是数组的长度一般开到题目要求的2、3倍

插入：取模后查找对应位置上有无元素，如果有的话依次向后，直到找到空位置

查找：从取模后的位置开始找，如果有值且≠x，查找成功；

如果有值且≠x，继续向后查找；

如果无值，查找失败，没有该元素。

删除：按照查找的思路找到x，打上标记

2、实现

核心在于find，如果存在返回当前的位置，如果不存在返回应该插入的地方

同时为了判断当前位置是否为空，需要设置一个在数据范围之外的数辅助判断

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

const int N = 200003,un = 0x3f3f3f3f;    // 取模的底数为素数，un为数据范围之外的数，用于判空
int h[N],n;

int find(int x)
{
    int t = (x%N + N) % N;               // 相似的取模操作
    while(h[t]!=un && h[t]!=x)
    {
        t++;
        if(t == N) t = 0;                // 如果超出边界，从0开始继续
    }
    return t;
}
int main()
{
    scanf("%d",&n);
    memset(h,0x3f,sizeof(h));            // 赋初值为数据范围外的数，0x3f经常使用，int的最大值
    for(int i = 0;i < n;i++)
    {
        char opt[2];
        int x;
        scanf("%s%d",opt,&x);
        int k = find(x);
        if(opt[0] == 'I')   h[k] = x;
        else
        {
```

```
            if(h[k] == un) puts("No");
            else puts("Yes");
        }
    }
    return 0;
}
```

字符串哈希/字符串前缀哈希法

最核心的：把一个字符串通过k进制，转化为一个数值

一、定义

引言：给定一个字符串str = ″ABCABCDEYXCAcwing″，希望映射为数值

问题1：如何让将子串映射为哈希值

希望将子串映射为数值，例如

```
h[0] = 0
h[1] = "A"的哈希值
h[2] = "AB"的哈希值
h[3] = "ABC"的哈希值
h[4] = "ABCA"的哈希值…
```

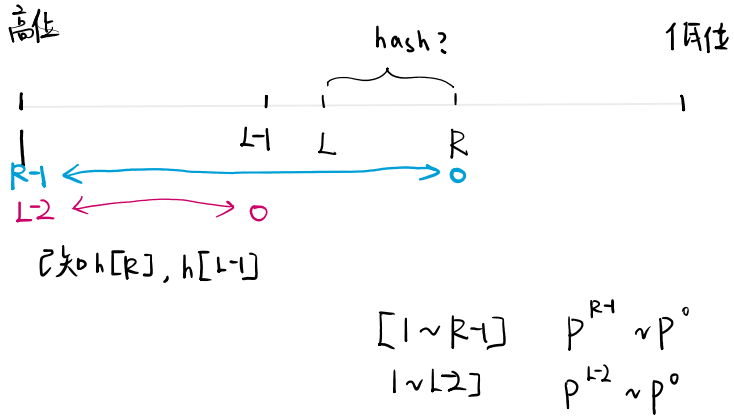
思路：当作P进制数来计算

″ABCD″ -> $(1\ 2\ 3\ 4)_p$ -> $(1\ 2\ 3\ 4)_p \ mod\ Q$ 从而将原字符串映射到数值Q之内

注意：①不能将字符映射为0，否则产生无法区分的问题

②p的选择要足够好，**假定**存在冲突（常用的是经验值p = 131,Q = 2^{64}

问题2：如何根据前缀哈希求任意一个子串的哈希



Step1：将L-1段左移，直到和R段对齐为止 即 $\times p^{R-L+1}$

Step2：h[R]-h[L-1]' = h[R]-h[L-1] $\times p^{R-L+1}$

突出的优势：快速判断两个字符串是否相等 O(1)的时间复杂度

二、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 l1,r1,l2,r2，请你判断 [l1,r1] 和 [l2,r2] 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 l1,r1,l2,r2，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≤n,m≤10^5

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long ULL;

const int N = 100010,P = 131;
ULL p[N],h[N];                // h[i]表示前i个子串的哈希值
char str[N];                  // 使用数组，以便初始下标从1开始

int n,m;

ULL get_res(int l,int r)
{
    return h[r] - h[l-1]*p[r-l+1];   // 虽然有可能超出范围，但ULL相当于与2^64取模
}

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

```
    }

    while(m--)
    {
        int l1,r1,l2,r2;
        scanf("%d%d%d%d",&l1,&r1,&l2,&r2);
        if(get_res(l1,r1) == get_res(l2,r2)) puts("Yes");
        else puts("No");

    }

    return 0;
}
```

三、扩展

可以解决快速判断两个字符串是否相等的问题

（感觉今天看到的csp第二题就是这个问题，下象棋，判断棋盘是否相等，我只能想到对行进行计算hash，然后依次比较每一行，没想到矩阵怎么写？直接当成行的和？）