Expriment 2

PB17111623 范睿

2019 年 11 月 17 日

1 数据库查询v2

1.1 题目

勤奋的小明为了预习下学期的数据库课程，决定亲自实现一个简单的数据库系统。该数据库系统需要处理用户的数据库插入和查询语句，并输出相应的输出。具体来说，用户的输入共包含若干条插入语句和查询语句。其中每条插入语句包含一个非负整数表示需要插入的数据。每条查询语句包含一个整数表示待查询的键值，若该键值存在则直接输出该键值，否则输出数据库中比该键值小的最大键值。

输入:
首先是若干行插入语句，每行的格式为如下的一种：
INSERT keykey
FIND keykey
最后单独的一行EXIT表示输入结束。

数据规模：
插入语句和查询语句一共不超过2,000,000条。
$0 \leq key \leq 10^9$

输出：
对每条查询语句输出一行，每行输出1个数字，表示查询的结果。
该键值存在则直接输出该键值，否则输出数据库中比该键值小的最大键值。

Sample Input:
INSERT 7
INSERT 11
INSERT 2
FIND 2
INSERT 5
INSERT 3
FIND 4
FIND 7

EXIT

Sample Output:

2

3

7

1.2 思路

利用红黑树的数据结构。

遇到INSERT: 插入红黑树，调整红黑树

遇到FIND: 利用红黑树的二叉搜索树的性质快速找到该键。若该键值不在树中，则先找到该插入位置的父节点，若该节点比其理论上的父节点键值大，输出父节点的键值，否则输出父节点的前驱的键值。

1.3 算法

Algorithm 1 Insert

输入: RBTree T, key

1: $//node \leftarrow malloc\ a\ memory\ for\ key$
2: $RBInsert(T, node)$
3: $RBAdjust(T, node)$

Algorithm 2 Find

输入: RBTree T, key

```
1: if T.data is key then
2:     print key return
3: end if
4: if T.data ≤ key then
5:     if T.right is NIL then
6:         print T.data
7:     else
8:         Find(T.right, key)
9:     end if
10: else
11:     if T.left is NIL then
12:         print Precessor(T).data
13:     else
14:         Find(T.left, key)
15:     end if
16: end if
```

1.4 复杂度分析

INSERT和红黑树的Insert时间复杂度一样，为$\mathcal{O}(\log n)$
FIND找到该插入的位置复杂度为$\mathcal{O}(\log n)$,找前驱复杂度为$\mathcal{O}(\log n)$，因此总复杂度为$\mathcal{O}(\log n)$

2 军训排队

2.1 题目

现有n个学生排成一个固定队伍进行军训，教官小明有一份所有n个人的名单（不同的人可能重名）。小明想要在整个队伍中找到一个连续的子队伍，并且满足该子队的所有人恰好有k个不重复的名字。请帮小明计算一下一共有多少种可能的子队伍。

输入：
一共有两行，第一行有两个数字n和k，用空格分隔第二行有n个单词$name_i$ ，用空格分隔
输入保证$len(name_i) \leq 5$（即输入的名字最多只含有5个字符）
数据规模：
$0 < k < n \ \leq \ 10,000,0000$
输出：
输出一共一个数字，即可能的子队伍数量。

Sample Input:
5 2
yyyyy iiiii yyyyy iiiii sssss

Sample Output:
7
/*
一共7种可能的子队伍为如下：
yyyyy iiiii
iiiii yyyyy
yyyyy iiiii
iiiii sssss
yyyyy iiiii yyyyy
iiiii yyyyy iiiii
yyyyy iiiii yyyyy iiiii
*/

2.2 思路

利用滑动窗口。low，mid，high标记窗口。low≤mid≤high。维持low和mid之间的名字种类有k个，low和high之间的名字种类有k+1个。每当找到这样的组合，可以计算出：

以low开始，不同名字的个数有k个的连续组合共有high-mid+1个，然后将low+1，继续寻找。
查找方法：
由于名字的位数最多有5位，将所有情况的个数计算出来，总数=26+26*26+26*26*26+26*26*26*26+26*26*26*26*26≤2000000，因此定义一个20000000的LUT，若改名字出现了，设置对应位置为1。

2.3 算法

Algorithm 3

输入: n, k, names
输出: sum

```
1: mid ← 1
2: NameNum ← 0
3: sum ← 0
4: for low = 1 to n do
5:     mid = max(mid, low)
6:     while mid ≤ n and NameNum ≤ k do
7:         //Add names[mid] into Queue
8:         mid ← mid+1
9:     end while
10:    if NameNum ≤ k then
11:        return sum
12:    end if
13:    high ← mid
14:    while high ≤ n and NameExists(names[high]) do
15:        high ← high+1
16:    end while
17:    sum += high-mid+1
18:    DecreaseNumberorDelete(names[low])
19: end for
```

2.4 复杂度分析

low从1走到n，为$\mathcal{O}(n)$,mid从1走到n，为$\mathcal{O}(n)$, 复杂度为$\mathcal{O}(n)$

3 内存分配

3.1 题目

C语言中需要申请一块连续的内存时需要使用malloc等函数。如果分配成功则返回指向被分配内存的指针(此存储区中的初始值不确定)，否则返回空指针NULL。现在小明决定实现一个类似malloc的内存分配系统，具体来说，他需要连续处理若干申请内存的请求，这个请求用一个闭区间$[a_i..b_i]$来表示。当这个区间和当前已被申请的内存产生重叠时，则返回内存分配失败的信息。否则返回内存分配成功，并将该区间标记为已被占用。假设初始状态下内存均未被占用，管理的内存地址范围为0-1GB（$0-2^{30}$）。

输入：
输入数据共n+1行。第一行一个整数n表示共需要处理n次内存分配。然后是n行数据，每行的格式为a_ib_i，表示申请区间为$[a_i, b_i]$

数据规模：
$n \leq 1,000,000$
$0 < a_i \leq b_i \leq 2^{30}$

输出:
输出共n行。对于每行内存分配的申请，若申请成功则输出一行0，若申请失败则输出一行-1。

Sample Input:

```
5
0 1
8 9
5 7
1 2
2 4
```

Sample Output:

```
0
0
0
-1
0
```

3.2 思路

利用区间树。每个RBNode存一个区间，键值为区间的low值。每次到来一个新区间先查找有没有重叠，若没有，加入区间树，返回1，否则返回0。
查看重叠的方法：从根结点查找，若区间的low值大于根结点的max值，说明没哟overlap，

输出1，将它加入区间树，否则观察该区间的low和根结点的键值的大小，若前者大，在根结点的右子树中再次判断，否则在根节点的左子树中再次判断。

3.3 算法

Algorithm 4 FindInterval

输入: RBTree T, Interval data

输出: 1 for no overlap, 0 for overlap

```
1: if T is NIL then
2:     return 1
3: end if
4: if data.low ≥ T.max then
5:     return 1
6: else
7:     if Overlap(T.interval, data) then
8:         return 0
9:     else
10:        if data.low ≥ T.data.low then
11:            return FindInterval(T.right, data)
12:        else
13:            return FindInterval(T.left, data)
14:        end if
15:    end if
16: end if
```

3.4 复杂度分析

插入区间和红黑树插入+调整的复杂度相同，为$\mathcal{O}(\log n)$
判断重叠的复杂度为$\mathcal{O}(\log n)$

4 危险品放置

4.1 题目

现有若干危险品需要放置在A,B两个仓库。当两种特定的危险品放置在相同地点时即可能产生危险。我们用危险系数$\alpha_{i,j}$表示危险品i,j放置在一起的危险程度。一些危险品即使放置在一起也不会产生任何危险，此时$\alpha_{i,j}=0$，还有一些危险品即使单独放置也会产生危险，此时$\alpha_{i,i}>0$。定义两个仓库整体的危险系数为$\max(\max_{i,j\in A}\alpha_{i,j},\max_{i,j\in B}\alpha_{i,j})$，即放置在一起的所有危险品两两组合的危险系数的最大值。现在对于一组给定的危险系数，需要设计方案使得整体危险系数最小。

输入:
输入共m+1行。第一行两个整数n和m表示共有n种危险品，危险品之间的危险组合（危险系数非零的物品组合）共m种。接下来的m行，每行三个整数$i, j, \alpha_{i,j}$，表示(i,j)为危险组合（i,j可能相等），其危险系数为$\alpha_{i,j} \geq 0$。
数据规模：

$0 < n \leq 100,000$
$0 < m \leq 1,000,000$

输出:
输出共一行，包含一个整数，表示整体危险系数的最小值。

Sample Input:
3 3
1 2 4
2 3 3
1 3 2

Sample Output:
2
/*
将1,3放在A仓库，2放在B仓库
*/

4.2 思路

利用不相交集合的树形结构。先将所有危险程度大小从大到小排序，然后从大到小一次访问。每次访问拿到i，j和$\alpha_{i,j}$,先判断i，j有无冲突。若冲突，输出$\alpha_{i,j}$，返回，否则为i,j,i',j'分别建立四个集合，然后将i和j'合并，将j和i'合并，访问下一个。
判断i,j是否冲突的方法是：顺着i的parent向上找到根，顺着j的parent向上找到根，若根相同，说明i和j处在同一集合，不能分开，说明冲突，否则不冲突。

4.3 算法

4.4 复杂度分析

i从1到n，为$\mathcal{O}(n)$，MakeSet时间为$\mathcal{O}(1)$，Union的操作是不相交集合的Union复杂度相同，为$\mathcal{O}(\alpha(n)), \alpha(n) < 4$，所以总复杂度为$\mathcal{O}(n)$

Algorithm 5 dangerous goods

输入: goods' number n, sorted dangerous degree Ns[m]

输出: minimun dangerous degress degree

```
1: a[n] stores goood_i's pointer
2: b[n] stores goood_i''s pointer
3: for i = 1 to n do
4:     if a[Ns[i].i] is NULL then MakeSet(a, Ns[i].i)
5:     end if
6:     if a[Ns[i].j] is NULL then MakeSet(a, Ns[i].j)
7:     end if
8:     if b[Ns[i].i] is NULL then MakeSet(b, Ns[i].i)
9:     end if
10:    if b[Ns[i].j] is NULL then MakeSet(b, Ns[i].j)
11:    end if
12:    if FindSet(a, Ns[i].i) == FindSet(a, Ns[i].j) then
13:        return Ns[i].alpha
14:    end if
15:    Union(a[Ns[i].i], b[Ns[i].j])
16:    Union(a[Ns[i].j], b[Ns[i].i])
17: end for
```