assignment7

Xiaoma

2022 年 11 月 20 日

题目 1.

解答. 使用桶排序，基于元素的位数对元素进行分组，每个分组的排序方法为基数排序。

Algorithm 1: SORT

Input: The array : $A[0...A.length-1]$; The decimal of elements : r

Output: The array sorted : $C[0...C.length-1]$;

```
Initialized d[0...d.length − 1];
d_max = 0;
for i = 0; i < A.length; ++i do
    d[i] = -1;
    t = A[i];
    while t != 0 do
        ++d[i];
        t = t / r;
    if d[i] > d_max then
        d_max = d[i];
create a new array B[0...d_max];
for i = 0; i < d_max; ++i do
    make B[i] an empty list;
for i = 0; i < A.length; ++i do
    insert A[i] into list B[d[i]];
for i = 0; i < d_max; ++i do
    RADIX-SORT(B[i]);
concatenate lists B[0]...B[d_max − 1] together in C;
return C
```

第 i 个操作的代价为

$$d_i \begin{cases} i & if\ i = 2^k, k \in \mathbb{N} \\ 1 & else \end{cases}$$

n 个操作序列的总代价为

$$\sum_{i=1}^{n} a_i$$

1. 聚合分析：

$$\sum_{i=1}^{n} a_i \leq \sum_{i=1}^{\lceil \log n \rceil} 2^i + n \leq 5n \in O(n)$$

摊还代价为 $O(1)$

2. 核算法：

假设每个操作的摊还代价为 3

(a) 第一个操作的代价为 1，剩余信用为 2

(b) 假设在前 2^i 次操作后，信用不为负值，那么在进行后续 $2^{i+1}-1$ 次操作时，每次操作的代价为 1，在进行第 2^{i+1} 次操作时，信用值至少为 $2^{i+1}+1$，代价为 2^{i+1}，信用为 1。

摊还代价为 $O(1)$

3. 势能法：

设 k 为满足 $2^k \leq i$ 的最大整数，则势函数为

$$\Phi(D_i) = \begin{cases} k+3 \quad i = 2^k \\ \Phi(D_{2^k}) + 2(i-2^k) \quad else \end{cases}$$

则

$$\Phi(D_i) - \Phi(D_{i-1}) = \begin{cases} -2^k + 3 \quad i = 2^k \\ 2 \quad else \end{cases}$$

所以

$$\sum_{i=1}^{n} a_i = 3n = O(n)$$

摊还代价为 $O(1)$

题目 2.

解答. 当一个元素进入队列的时候，它前面的所有比它大的元素都不会对结果产生影响。

所以我们使用一个普通队列来存储队列中的元素，另外需要一个单调的双端队列存储其最小值的变化。

规定最小元素在双端队列的队头，在有新元素入队时，首先检查新元素是否小于当前队列最小值，若小于，则将当前双端队列清空，再将新元素插入双端队列，因为当前双端队列中元素相对于新元素在队列中较为靠前，故出队顺序优先，已知新元素为最小值，则原双端队列出队与否对整个队列的最小值不产生影响。若大于，则直接将元素插入双端队列。

Algorithm 2: My_Queue

Initialized : deque as d

Initialized : queue as q

Algorithm 3: ENQUEUE

Input: The number : *value*;

while *!d.empty() AND d.back()* $>$ *value* **do**

  d.pop_back();

d.push_back(value);

q.push(value);

Algorithm 4: DEQUEUE

Output: The number : *value*;

if *q.empty()* **then**
  return *ERROR*
value = q.front();

if *value == d.front()* **then**
  d.pop_front();
q.pop();

return *value*;

Algorithm 5: FIND_MIN

Output: The min number : *value*;

if *d.empty()* **then**
  return *ERROR*;
value = d.front();

return *value*;

聚合分析：

对于 n 个操作序列，ENQUEUE 与 DEQUEUE 的操作次数相同，而对于双端队列，其出队与入队次数也应该相同，则总的操作代价

$$\sum_{i=1}^{n} a_i \leq \sum_{i=1}^{x} 1 + \sum_{i=1}^{x} d_i + \sum_{i=1}^{x} 1 + (x - \sum_{i=1}^{x} d_i) + (n - 2x) \leq 3n$$

则每个操作的摊还复杂度为 $O(1)$

题目 3.

解答. 假设 u_i 为贪心算法第 i 次迭代还没有被覆盖的元素个数，k 为所需最少的集合个数，故最少需要 k 个最优集合可以覆盖 u_i 个元素，所以最优集合中至少一个集合包含至少 u_i/k 个元素。

$$u_{i+1} \leq (u_i - \frac{u_i}{k}) \leq (1 - \frac{1}{k})u_i \leq (1 - \frac{1}{k})(1 - \frac{1}{k})u_{i-1} \leq ... \leq (1 - \frac{1}{k})^{i+1}u_0 = (1 - \frac{1}{k})^{i+1}n$$

$$u_{i+1} \leq (1 - \frac{1}{k})^{i+1}n$$

$$(1 - \frac{1}{k})^i = ((1 - \frac{1}{k})^k)^{\frac{i}{k}} \leq e^{-\frac{i}{k}}$$

$$1 - (1 - \frac{1}{k})^k \geq 1 - \frac{1}{e}$$

则算法的近似比为 $1 - (1 - \frac{1}{k})^k \geq 1 - \frac{1}{e}$

当 $u_i \leq 1$ 时，终止迭代。

$$ne^{-\frac{i}{k}} \leq 1 \Leftrightarrow e^{-\frac{i}{k}} \leq \frac{1}{n} \Leftrightarrow -\frac{i}{k} \leq \ln n \Leftrightarrow i \geq k \ln n$$

则算法的时间复杂度为 $O(k \log n)$