[college name]《算法设计与分析》第 [homework id] 次作业

[college name]　[author id]　[author name]

September 14, 2022

1 [作业题目 1]

1.1 [作业题目 1.1]

$$T(n) = \begin{cases} 1\ ,\ n = 1 \\ T(n-2) + 3n\ ,\ n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-2) + 3n = T(n-4) + 3n + 3(n-2) = T(n-6) + 3n + 3(n-2) + 3(n-4) = ... \\ &= 3n + 3(n-2) + 3(n-4) + ... + 9 + 1 = \tfrac{1}{2}(3n+9)(\tfrac{3n-9}{6} + 1) + 1 = O(n^2) \end{aligned}$$

由此可知，原式渐进上界为 $O(n^2)$。

$n = 1$ 时，对 $c \geq 1$ 结论显然成立；

$n > 1$ 时，有：
$$\begin{aligned} T(n) &= T(n/2) + 2^n \\ &\leq c2^{\frac{n}{2}} + 2^n \\ &= c2^n - (c-1)2^n + c2^{\frac{n}{2}} \\ &= c2^n - ((c-1)2^{\frac{n}{2}} - c)2^{\frac{n}{2}} \\ &\leq c2^n - (2(c-1) - c)2^{\frac{n}{2}} \\ &= c2^n - (c-2)2^{\frac{n}{2}} \end{aligned}$$

显然，只要 $c \geq 2$，就有 $T(n) \leq c2^n$ 成立，从而原式渐进上界为 $O(2^n)$。

1.2 [作业题目 1.2]

$$T(n) = \begin{cases} 1\ ,\ n = 1 \\ 8T(n/4) + 2n\ ,\ n > 1 \end{cases}$$

使用主方法计算渐进上界，注意到 $2n = O(n)$，由上式可知 $a = 8, b = 4, d = 1$，由 $d < log_b a$，可得上式的渐进上界为 $O(n^{log_b a}) = O(n^{\frac{3}{2}})$。

2 [作业题目 2]

现有 k 个有序数组（从小到大排序），每个数组中包含 n 个元素。你的任务是将他们合并成 1 个包含 kn 个元素的有序数组。首先来回忆一下课上讲的归并排序算法，它提供了一种合并有序数组的算法 $Merge$。如果我们有两个有序数组的大小分别为 x 和 y，$Merge$ 算法可以用 $O(x + y)$ 的时间来合并这两个数组。

2.1 [作业题目 2 分析]

如果我们应用 $Merge$ 算法先合并第一个和第二个数组，然后由合并后的数组与第三个合并，再与第四个合并，直到合并完 k 个数组。请分析这种合并策略的时间复杂度（请用关于 k 和 n 的函数表示）。

显然，第一次合并的复杂度为 $O(n+n)$，第二次合并的复杂度为 $O(2n+n)$，第三次合并的复杂度为 $O(3n+n)$，以此类推，第 $k-1$ 次合并的复杂度为 $O((k-1)n+n)$。

从而总复杂度为 $n+2n+...+(k-1)n=\frac{1}{2}k(k-1)n=O(nk^2)$。

2.2 [作业题目 2 算法与伪代码]

针对本题的任务，请给出一个更高效的算法，并分析它的时间复杂度（此题若取得满分，所设计算法的时间复杂度应为 $O(nklogk)$）。

分析题目，只要使用优先队列维护 k 个有序数组未归并部分首个元素的集合即可，每次从优先队列中取出一个元素归并，再将提供被取出元素的数组的下一元素插入优先队列，这样共需要处理 kn 个元素，又因为优先队列的大小为 k，每个元素的处理时间为 $logk$，即得到 $O(nklogk)$ 的时间复杂度，伪代码如下。

Algorithm 1: k 路归并问题优化做法

Input: 正整数 n，为数组长度，正整数 k，为数组个数，k 个长度为 n 的数组 $A[]$

Output: 长度为 nk 的数组 Ans，为归并后的数组

```
global pointer[k]
function main(n, k, A):
    for i ← 0 to k − 1 do
        pointer[i] ← 0;
        push (A[i][pointer[i]], i) to priority_queue;
    end
    for i ← 0 to nk − 1 do
        (val, index) ← top of priority_queue;
        remove top from priority_queue;
        Ans[i] ← val;
        pointer[index] ← pointer[index] + 1;
        if pointer[index] < length of A[index] then
            push (A[index][pointer[index]], index) to priority_queue;
        end
    end
    return;
end
```

Algorithm 2: 填数字问题朴素做法

Input: 正整数 n，为数组长度

Output: 长度为 n 的数组 A，为填充后的数组

```
function main(n):
    push (0, n − 1) to priority_queue;
    for i ← 1 to n do
        (l, r) ← top of priority_queue;
        remove top from priority_queue;
        mid ← (l + r)/2;
        A[mid] ← i;
        if mid − 1 ≥ l then
            push (l, mid − 1) to priority_queue;
        end
        if r ≥ mid + 1 then
            push (mid + 1, r) to priority_queue;
        end
    end
    return A;
end
```

Algorithm 3: 填数字问题优化做法

Input: 正整数 n，为数组长度

Output: 长度为 n 的数组 A，为填充后的数组

```
global list[n];
function search(l, r):
    mid ← (l + r)/2;
    insert mid to head of list[r − l + 1];
    if r ≥ mid + 1 then
        call search(mid + 1, r);
    end
    if mid − 1 ≥ l then
        call search(l, mid − 1);
    end
end
function main(n):
    call search(0, n − 1);
    count ← 1;
    for i ← n to 1 do
        for j ← head of list[i] to tail of list[i] do
            A[value of j] ← count;
            count ← count + 1;
        end
    end
    return A;
end
```

3 [附录暨部分算法的实现]

3.1 填数字问题朴素做法与测试数据生成

```
#include <queue>
#include <cstdio>
using namespace std;

struct Node
{
  int l,r,len;

  inline Node(int l,int r)
  {
    this->l=l,this->r=r;
    len=r-l+1; return;
  }

  inline bool operator<(const Node& rhs) const
  {
    if (this->len<rhs.len) return true;
    if (this->len>rhs.len) return false;
    return this->l>rhs.l;
  }

  inline bool operator>(const Node& rhs) const
  {
    if (this->len>rhs.len) return true;
    if (this->len<rhs.len) return false;
    return this->l<rhs.l;
  }
};

int a[10005];

int main()
{
  int n=10000;
  freopen("testcase.txt","w",stdout);
  printf("%d\n",n);
  priority_queue<Node> pq;
  Node node(0,n-1); pq.push(node);
  for (int i=1,l,r;i<=n;i++)
  {
    Node node=pq.top(); pq.pop();
    l=node.l,r=node.r;
    // printf("%d %d\n",l,r);
    int mid=(l+r)>>1;
    a[mid]=i;
    if (mid-1>=l) {Node newNode(l,mid-1); pq.push(newNode);}
    if (r>=mid+1) {Node newNode(mid+1,r); pq.push(newNode);}
  }
  for (int i=0;i<n;i++)
  printf("%d\n",a[i]);
  return 0;
}
```

3.2 填数字问题优化做法与自动对拍

```
#include <cstdio>
#include <vector>
using namespace std;
```

```

int a[10005];
int tot,fst[10005],nxt[10005],val[10005];

void build(int l,int r)
{
  nxt[++tot]=fst[r-l+1],fst[r-l+1]=tot;
  int mid=(l+r)>>1; val[tot]=mid;
  if (r>=mid+1) build(mid+1,r);
  if (mid-1>=l) build(l,mid-1);
  return;
}

int main()
{
  int n,cnt=0;
  freopen("testcase.txt","r",stdin);
  scanf("%d",&n);
  build(0,n-1);
  for (int i=n;i>0;i--)
  for (int j=fst[i];j;j=nxt[j])
  a[val[j]]=++cnt;
  for (int i=0;i<n;i++) printf("%d\n",a[i]);
  bool isRight=true;
  for (int i=0,std;i<n;i++)
  {
    scanf("%d",&std);
    if (a[i]!=std) isRight=false;
  }
  printf(isRight?"AC\n":"WA\n");
  return 0;
}
```