第六次算法作业

1. 初始时令 n 个时间槽为空，时间槽 i 为单位时间长度，结束于时刻 i。我们按惩罚值单调递减的顺序处理所有任务。当处理任务 a_j 时，如果存在不晚于 a_j 的截止时间 d_j 的空时间槽，则将 a_j 分配到其中最晚的那个，如果不存在这样的时间槽，将 a_j 分配到最晚的空时间槽。

a. 证明：此算法总能得到最优解。

（1） 贪心选择性质

假设 O={a_1,a_2,……,a_n}是最优解。

如果 a_j 调度的时间 t_j<d_j，假设有活动 a_k 的调度时间 t_k=d_j，那么调换 a_j 和 a_k 不会对惩罚值产生影响。

如果 a_j 的调度时间 t_j>d_j，但 d_j≤j，那么在前 j 个任务中必然存在任务 a_k 使得 a_k 的惩罚小于 a_j 的惩罚，这时可以替换 a_j 和 a_k 形成一个更优解，由于 O 是最优解，所以这种情况不可能发生。

如果 a_j 的调度时间 t_j>d_j，并且 d_j>j，那么 a_j 与其他任意必然延误的任务交换不会影响惩罚度的总和。

因此该问题满足贪心选择性质。

（2）最优子结构

假设最优解为 O={a_i,a_{i+1},……,a_j}，原问题在安排 a_i 之后，产生子问题的解 O'={a_{i+1},……,a_j}，假设存在 O''使得 O''比 O'更优，那么存在 O'''=O''∪{a_i}使得 O'''优于 O，这与 O 是最优解矛盾。因此该问题满足最优子结构性质。

由(1)(2)可知，此算法总能得到最优解。

b. 使用快速不相交集合实现此算法。

算法伪代码如下所示：

```
SCHEDULING_DISJOINT_SET_FOREST(A)
initialize an array D of size n
W=0
for i=1 to n
    D[i] = MAKE_SET(a[i])
for i=1 to n
    x = FIND_SET(D[A[i].deadline])
    if x!= null
        UNION(x, x-1)
    else
        W = W + A[i]. penalty
        x = FIND_SET(n)
        UNION(x-1,x)
return W
```

MAKE_SET, UNION, FIND_SET 操作的次数都是线性复杂度，所以算法的运行时间为 O(nα(n))

2. **假设当装载因子小于 1/3 时将表规模变为原来的 2/3，使用势函数**

$$\mathbf{\Phi(T) = |2 \cdot T \cdot num - T \cdot size|}$$

证明：使用此策略，TABLE-DELETE 操作的摊还代价的上界是一个常数。

分析第 i 个操作为 TABLE-DELETE 的情况。若 a_i≥1/2， 没有引起表收缩，

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot \mathrm{num}_i - \mathrm{size}_i) - (2 \cdot \mathrm{num}_{i-1} - \mathrm{size}_{i-1}) \\
&= \mathrm{num}_i + 2 - (\mathrm{num}_i + 1)) = 1
\end{aligned}
$$

若 $a_i \geqslant 1/3$，$a_i < 1/2$，$a_{i-1} \geqslant 1/2$，没有引起表收缩，

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot \mathrm{num}_i - \mathrm{size}_i) - (\mathrm{size}_{i-1} - 2 \cdot \mathrm{num}_{i-1}) \\
&= 3 + 4 \cdot \mathrm{num}_i - 2\mathrm{size}_i \\
&< 3
\end{aligned}
$$

若 $a_i \geqslant 1/3$，$a_{i-1} < 1/2$，$a_i < 1/2$，没有引起表收缩，则

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (\mathrm{size}_i - 2 \cdot \mathrm{num}_i) - (\mathrm{size}_{i-1} - 2 \cdot \mathrm{num}_{i-1}) \\
&= 1 + (\mathrm{size}_i - 2 \cdot \mathrm{num}_i) - \big(\mathrm{size}_i - 2 \cdot (\mathrm{num}_i + 1)\big) = 3
\end{aligned}
$$

若 $a_i < 1/3$，$a_{i-1} \geqslant 1/3$，则第 i 次操作引起表收缩，$\frac{\mathrm{size}_i}{2} = \frac{\mathrm{size}_{i-1}}{3} = \mathrm{num}_{i-1} = \mathrm{num}_i + 1$，$c_i = \mathrm{num}_i + 1$。

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= \mathrm{num}_i + 1 + (\mathrm{size}_i - 2 \cdot \mathrm{num}_i) - (\mathrm{size}_{i-1} - 2 \cdot \mathrm{num}_{i-1}) \\
&= \mathrm{num}_i + 1 + (2(\mathrm{num}_i + 1) - 2 \cdot \mathrm{num}_i) - \big(3 \cdot (\mathrm{num}_i + 1) - 2(\mathrm{num}_i + 1)\big) \\
&= 2
\end{aligned}
$$

因此，TABLE-DELETE 操作的摊还代价的上界是一个常数。

3. 有序数组上的二分查找花费对数时间，但插入一个元素的时间与数组规模呈线性关系。我们可以维护多个有序数组来提高插入性能。

a. 设计算法，实现这种数据结构上的 SEARCH 操作，分析最坏情况运行时间。

SEARCH 操作可以对每一个 Ai 执行二分查找操作，算法伪代码为：

```
DYNAMIC-SEARCH(A,num)
index=-1
for i = 0 to k-1
    if A[i].length != 0
        index = BINARY-SEARCH(A[i],num)
return index
```

在对 Ai 进行二分查找时，查找的算法复杂度为$\Theta(\lg 2^i)$，因此在最坏情况下，整个算法的算法复杂度为：

$$
\begin{aligned}
\mathrm{T(n)} &= \Theta(\lg 2^{k-1} + \lg 2^{k-2} + \cdots + \lg 2^1 + \lg 2^0) \\
&= \Theta\left(\frac{\mathrm{k(k-1)}}{2}\right) \\
&= \Theta\left(\frac{\lg(\mathrm{n}+1)(\lg(\mathrm{n}+1)-1)}{2}\right) \\
&= \Theta(lg^2 n)
\end{aligned}
$$

b. 设计 INSERT 算法，分析最坏情况运行时间和摊还时间。

将要插入的元素看做一个数组，如果 A0 为空，我们把 A0 替换为这个数组，否则将这个数组与 A0 归并排序为一个新的数组，检查 A1，如果 A1 为空就把 A1 替换为新的数组，否则对 A1 和新的数组归并排序为新的数组，检查 A2，以此类推。算法的伪代码描述为：

```
DYNAMIC-INSERT(A,num)
T={num}
for i = 0 to k-1
    if A[i].length = 0
        A[i]=T
    else
        T=MERGE(T, A[i])
```

讨论最坏情况下的算法复杂度，merge 两个 m 个元素的数组花费的时间为 2m，假设 A_{k-1}之前的所有数组都满了，此时的运行时间为：

$$\begin{aligned} \mathrm{T(n)} &= 2\cdot 2^0 + 2\cdot 2^1 + \cdots + 2\cdot 2^{k-2} \\ &= 2^k - 2 = 2n = \Theta(n) \end{aligned}$$

使用聚合法分析摊还时间。记 r 为 n 的最右边的 0 的位置，第 r 位为 0 的概率是$1/2^r$，讨论 n 次插入所需的时间，每个 r 对应的插入次数为$\left\lceil\frac{n}{2^r}\right\rceil$，总的复杂度为：

$$\mathrm{O}\left(\sum_{r=0}^{\lceil\lg(n+1)\rceil}\left\lceil\frac{n}{2^r}\right\rceil\cdot 2^r\right) = \mathrm{O(nlgn)}$$

因此每次插入的摊还时间为$\mathrm{O(lgn)}$

c. 讨论如何实现 DELETE

假设需要删除的数为 x，DELETE 算法过程为：

1. 找到最小的 j 满足 A_j是满数组，设 j 为 A_j最后一个元素
2. 使用 DYNAMIC-SEARCH 找到 x，使用 y 替换 x，并对这个数组重新排序
3. 切分 A_j数组，第 1 个元素进入 A_0，第 2、3 个元素进入 A_1，接下来 4 个元素进入 A_2，以此类推，设置 A_j为空数组。

在最坏情况下，找到 Aj 的算法复杂度为$\Theta(\mathrm{lgn})$，DYNAMIC-SEARCH 找到 x 的算法复杂度为$\Theta(lg^2n)$，用 y 替换 x 并排序的算法复杂度为$\Theta(\mathrm{n})$，切分 A_j数组的算法复杂度为$\Theta(2^j)$，总的算法复杂度为$\Theta(\mathrm{n})$