Homework9

刘喆骐 2020013163 探微化01

16-4

a.

假设L是一个最佳的解。如果a_j在ddl前就被规划执行，我们就可以把它和安排在其ddl上执行的任务交换而不受惩罚。若a_j在ddl后被安排执行，并且$a_j.ddl \leq j$，又由于L是最优解，那么就肯定不存在有一个a_j前安排的任务，它的惩罚比a_j小（若有，则交换此任务和a_j，将得到更优解）。若a_j在ddl后被安排执行，并且$a_j.ddl > j$，那么a_j和任意迟到的元素交换都不会增加惩罚。这就满足了题中贪心算法的条件，故贪心算法得到最优。

b.

让MAKE-SET(x)返回一个指向元素x的指针，而x元素是一个只含自己的集合。不相交集合将是在连续时间内安排调度的元素的集合。使用此结构快速找到下一个可用时间来安排任务。在每个不相交集合的代表x处储存此集合中最早开始的任务的时间x.low和最晚开始的任务的时间x.high。利用UNION(x,y)维护这两个值，而这可以在常数的时间内完成。只在两个集合的时间连续的情况下合并它们。假设任务a1具有最大的惩罚，任务a2具有第二大的惩罚，以此类推，存放在数组A中，其中$A[i] = a_i$。维护数组D，使得D[i]为截止日期是i的任务。D的大小最多为n，因为截止日期晚于n的任务不可能按时安排。故总共最多有n个MAKE-SET、n个UNION和n个FIND-SET操作，因此根据定理21.14，运行时间为$O(n\alpha(n))$
伪代码如下:

```
schedule(a)
    let D[1..n] be a new array
    for i = 1 to n
        a[i].time = a[i].deadline
        if D[a[i].deadline] != NULL
            y = FIND-SET(D[a[i].deadline])
            a[i].time = y.low - 1
        x = MAKE-SET(a[i])
        D[a[i].time] = x
        x.low = x.high = a[i].time
        if D[a[i].time - 1] != NULL
            UNION(D[a[i].time - 1], D[a[i].time])
        if D[a[i].time + 1] != NULL
            UNION(D[a[i].time], D[a[i].time + 1])
```

17-2

a.

算法伪代码如下

```
bisearch(a,key)
    if a.length==0
        return -1
    int mid=a.length/2
    if key==mid
        return a+mid
    else if key>mid
        bisearch(a[mid+1,a.end],target)
    else
        bisearch(a[0,mid],target)
search(A,key)
    //A是k个有序数组的集合
    for a in A:
        e=bisearch(a,key)
            if e!=-1
                return (e,a)
```

最坏时间：假设每个子数组都是满的，并且需要遍历整个子数组集合，每次搜索耗时为$log(2^i)$，那么耗时为$\sum_0^{k-1} log(2^i) = \Theta(k^2) = \Theta(log^2 n)$

b.

将插入的元素放入A_0中，然后逐次合并数组$A_0, A_1, ..., A_{m-1}$为A_m，直至每个数组或满或空。最坏情况下,m=k,合并耗时为$O(2^k) = O(n)$.

均摊分析：使用accounting method。每次插入分配log(n)的代价。对于每个元素来说，其最多需要合并log(n)次，因为最多只有log(n)个数组，故分配的的代价多余实际的代价，故时间复杂度为$O(logn)$。

c.

取出最小的m，使得$n_m > 0$.如果要删除的值在A_m中，则直接删除。如果要删除的值不在A_m中，假设在A_s中，则删除此值，并将A_m中取任意一个元素插入A_s。随后将A_m按照数组的下标拆分成$A_0, A_1, ..., A_{m-1}$，这一步的耗时为$O(log(n))$。只考虑删除而不考虑搜索此待删除元素的时间，时间复杂度为$O(logn)$。