习题5

202328015926048-丁力

1. 最大子段和问题: 给定整数序列 $a_1, a_2, \cdots, a_n$, 求该序列形如 $\sum_{k=i}^{j} a_k$ 的子段和的最大值:

$$\max\left\{0, \max_{1\le i\le j\le n}\sum_{k=i}^{j} a_k\right\}$$

- 已知一个简单算法如下:

```
int Maxsum(int n, int a[], int& besti, int& bestj) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        int suma = 0;
        for (int j = i; j <= n; j++) {
            suma += a[j];
            if (suma > sum) {
                sum = suma;
                besti = i;
                bestj = j;
            }
        }
    }
    return sum;
}
```

试分析该算法的时间复杂性。

> 显然，该算法的时间复杂度为$O(n^2)$。
>
> 下面我们通过分析证明之:
>
> 设计算次数为$T(n)$:
>
> $$\begin{aligned} T(n) =& n + (n-1) + (n-2) + \ldots + 0 \\ &= \frac{n(n+1)}{2} \\ &= O(n^2) \end{aligned}$$
>
> Q.E.D

- 试用分治算法解最大子段和问题, 并分析算法的时间复杂性。

> 从中间开始划分数组, 分为一下几块:
>
> - $a_1, a_2, a_3, \ldots, a_{\lfloor n/2 \rfloor}$
> - $a_{\lfloor n/2 \rfloor + 1}, \cdots, a_n$
>
> 那么最大子段和要么存在于左半边，或者右半边，或者位于两种中间的拼接处。

解法：

> 对于左边或者是右边的子段，都可以通过递归的调用求解最大子段来得到。对于位于两种中间拼接形成的类型，从中间开始向左右两边搜索。得到最大值之和。
>
> 下面给出这个解法的 `Python` 代码实现：

```python
def find_max_subarray(array):
 def find_max_sub(array, left, right):
     # Base case: single element
     if left == right:
         return array[left]
     mid = (left + right) // 2
     left_max_sum = find_max_sub(array, left, mid)
     right_max_sum = find_max_sub(array, mid+1, right)
     cross_sum_max = find_cross_sum_max(array, left, right, mid)
     return max(left_max_sum, right_max_sum, cross_sum_max)

 def find_cross_sum_max(array, left, right, mide):
     if left == right:
         return array[left]
     left_sum_max = -float('inf')
     right_sum_max = -float('inf')
     left_sum = 0
     right_sum = 0
     for i in range(mide, left-1, -1):
         left_sum += array[i]
         if left_sum > left_sum_max:
             left_sum_max = left_sum
     for j in range(mide+1, right+1):
         right_sum += array[j]
         if right_sum > right_sum_max:
             right_sum_max = right_sum
     return left_sum_max + right_sum_max

 return find_max_sub(array, 0, len(array)-1)
```

计算复杂度分析：

从上面的计算表达式可以看出其地推关系式为：

$$T(n) = 2T(n/2) + O(n) \qquad \text{计算交叉最大和的部分为} O(n)$$

通过主定理容易得到，其时间计算复杂度为：

$$O(nlog(n))$$

- 试说明最大子段和问题具有最优子结构性质，并设计一个动态规划算法解最大子段和问题。分析算法的时间复杂度。(提示：令 $b(j) = \max_{1 \le i \le j \le n} \sum_{k=i}^{j} a_k, j = 1, 2, \cdots, n$)。

如果要使用动态规划来解决这个问题，我们首先需要定义一下这个最优解的递推关系式，设其为$b(j)$，那么我们有：

$$b(j) = \max_{1 \le i \le j \le n} \sum_{k=i}^{j} a_k, j = 1, 2, \cdots, n$$

那么，我们有：

$$b(j+1) = max(a_j, a_j + b(j))$$

给出其 `Python` 代码实现：

```python
def find_max_subarray_by_dp(array):
 max_sum = - float('inf')
 sum = 0
 size = len(array)
 for i in range(size):
     sum += array[i]
     if sum > max_sum:
         max_sum = sum
     if sum < 0:
         sum = 0
 return max_sum
```

容易得知，其只有一个for循环构成，所以其时间复杂度为$O(n)$

2.(双机调度问题) 用两台处理机 A 和 B 处理 n 个作业。设第 i 个作业交给机器 A 处理时所需要的时间是 a_i, 若由机器 B 来处理, 则所需要的时间是 b_i 。现在要求每个作业只能由一台机器处理, 每台机器都不能同时处理两个作业。设计一个动态规划算法, 使得这两台机器处理完这 n 个作业的时间最短 (从任何一台机器开工到最后一台机器停工的总的时间)。以下面的例子说明你的算法:

$$n=6,(a_1,a_2,a_3,a_4,a_5,a_6)=(2,5,7,10,5,2),$$
$$(b_1,b_2,b_3,b_4,b_5,b_6)=(3,8,4,11,3,4)$$

(提示: 类似于 0/1 背包问题的点对表示, 可以采用三元组 (T_A,T_B,δ) 表示调度状态, 其中, T_A、T_B 分别表示机器 A, B 已占用的加工时间, δ 是一个整数, 用以表示当前调度方案中安排给机器 A 的作业情况, 例如, 将作业 i 安排给机器 A, 则 δ 增加 2^{i-1} 。剩下来的工作就是计算调度的状态集并进行化简:

$$S_0=\{(0,0,0)\},S_i=\left((a_i,0,2^{i-1})+S_{i-1}\right)\cup((0,b_i0)+S_{i-1}),$$
$$i=1,2,\cdots,n$$

期间的状态集化简可根据某个设定的间值进行)。

解: 在完成前 k 个作业时, 设机器 A 工作了 x 时间, 则机器 B 此时最小的工作时间是 x 的一个函数。
设 $\mathrm{F[k][x]}$ 表示完成前 k 个作业时, 机器 B 最小的工作时间, 则

$$\mathrm{F[k](x)=\min\{F[k-1](x)+b_k,F[k-1]\,(x-a_k)\}}$$

其中 $\mathrm{F[k-1](x)+b_k}$ 对应第 k 个作业由机器 B 来处理（完成 $\mathrm{k-1}$ 个作业时机器 A则完成前 k 个作业所需的时间为 $\mathrm{max\{x,F[k](x)\}}$

- 当处理第一个作业时, $a[1]=2,b[1]=3;$
  机器 A 所花费时间的所有可能值范围: $\mathrm{0\le x\le a[0]}$.
  $\mathrm{x<0}$ 时, 设 $\mathrm{F[0][x]=\infty}$, 则 $\mathrm{max(x,\infty)=\infty}$;
  $0\le x<2$ 时, $F[1][x]=3$, 则 $\mathrm{Max}(0,3)=3$,
  $x\ge 2$ 时, $F[1][x]=0$, 则 $\mathrm{Max}(2,0)=2$;
- 处理第二个作业时: x 的取值范围是: $\mathrm{0<=x<=(a[0]+a[1])}$,当 $\mathrm{x<0}$ 时, 记 $\mathrm{F[2][x]=\infty}$;
  以此类推下去
- 实例分析:

1. **初始状态**：没有作业时，两台机器的工作时间都是0。

2. **第1个作业**:
   - 当分配给机器A时（工作时间为2），机器B还没有开始工作，所以 `F[1][2] = 0`。
   - 当不分配给机器A时，机器B需要3个时间单位来完成这个作业，因此 `F[1][0] = 3`。
3. **第2个作业**:
   - 接着，我们考虑到目前为止机器A的工作时间最多为7（前两个作业的时间总和），我们更新 `F[2][x]`。
   - 对于 `x = 7`，即机器A已经工作了7个时间单位，如果第2个作业分配给机器B，那么机器B的工作时间为0，因此 `F[2][7] = 0`。
4. **第3个作业**:
   - 类似地，我们更新 `F[3][x]`。对于 `x = 14`（前三个作业的时间总和），如果第3个作业分配给机器B，那么 `F[3][14] = 0`。
5. **第4个作业**:
   - 继续更新 `F[4][x]`。对于 `x = 24`（前四个作业的时间总和），如果第4个作业分配给机器B，那么 `F[4][24] = 0`。
6. **第5个作业**:
   - 更新 `F[5][x]`。对于 `x = 29`（前五个作业的时间总和），如果第5个作业分配给机器B，那么 `F[5][29] = 0`。
7. **第6个作业**:
   - 最后，我们更新 `F[6][x]`。对于 `x = 31`（所有作业的时间总和），如果第6个作业分配给机器B，那么 `F[6][31] = 0`。

在计算的过程中，我们一直在寻找使机器B工作时间最小化的工作时间 `x`。最终，我们找到在机器A工作14个时间单位的情况下，机器B可以在15个时间单位内完成其作业，这是所有作业完成所需时间最短的情况。

答案:

- 机器A的最优工作时间是14个时间单位。
- 机器B的最优工作时间是15个时间单位。
- 完成所有作业所需的最短时间是15个时间单位。

3.考虑下面特殊的整数线性规划问题

$$\max \sum_{i=1}^{n} c_i x_i$$

$$\sum_{i=1}^{n} a_i x_i \le b, \quad x_i \in \{0,1,2\}, 1 \le i \le n$$

试设计一个解此问题的动态规划算法，并分析算法的时间复杂度。

这里我们考虑这样的一种情况，由于x_i的取值范围为$\{0,1,2\}$，所以把其拆解为两个变量，然后转换成一个0/1背包问题。也就是:

$$x_i = y_i + y_{i+n}, \quad i \in [1,n]$$

其中,$y_i, y_{i+n} \in \{0,1\}$。

那么上书问题转换成：

$$\max \sum_{i=1}^{2n} c_i x_i$$

$$\sum_{i=1}^{2n} a_i y_i \leq b, \quad y_i \in \{0,1\}, 1 \leq i \leq 2n$$

把c_i看做价值，a_i看做重量，b看做背包容量，此时可以转换成0/1背包问题。所以，此时的计算复杂度为：

- 每个物品都会被考虑一次，总共有$2n$个物品。
- 对于每个物品，算法都需要遍历从0到b的所有可能的容量值。
- 因此，总的时间复杂度是$O(2nb)$。

4.4. 可靠性设计: 一个系统由 n 级设备串联而成, 为了增强可靠性, 每级都可能并联了不止一台同样的设备。假设第 i 级设备 $\mathrm{D_i}$ 用了 $\mathrm{m_i}$ 台, 该级设备的可靠性是 $\mathrm{g_i}\left(\mathrm{m_i}\right)$, 则这个系统的可靠性是 $\Pi \mathrm{g_i}\left(\mathrm{m_i}\right)$ 。一般来说 $\mathrm{g_i}\left(\mathrm{m_i}\right)$ 都是递增函数, 所以每级用的设备越多系统的可靠性越高。但是设备都是有成本的, 假定设备 D_i 的成本是 $\mathrm{c_i}$, 设计该系统允许的投资不超过 c,那么, 该如何设计该系统 (即各级采用多少设备) 使得这个系统的可靠性最高。试设计一个动态规划算法求解可靠性设计问题。

显然，这个问题是一个数学规划的问题， 给出问题的定义：

$$\max(\Pi \mathrm{g_i}\left(\mathrm{m_i}\right))$$

$$\left\{\sum_{i=1}^{n} c_i \times g_i(m_i) \leq c \rightarrow 1 \leq \mathrm{m_1} \leq 1+\left\lfloor\frac{\mathrm{c}-\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{c_i}}{\mathrm{c_n}}\right\rfloor\right.$$

记 $G[k](x)$ 为第 k 级设备在可用投资为 x 时的系统可靠性最大值则有如下关系式:

$$\mathrm{G}[\mathrm{k}](\mathrm{x})=\max_{1 \leq \mathrm{m_k} \leq\left\lfloor\frac{\mathrm{x}}{c_{\mathrm{k}}}\right\rfloor}\left\{\mathrm{g_k}\left(\mathrm{m_k}\right) \cdot \mathrm{G}[\mathrm{k}-1]\left(\mathrm{x}-\mathrm{c_k} \mathrm{m_k}\right)\right\}$$

定义下列函数

$$G[k](x)=\begin{cases}-\infty, & x<\sum_{i=k}^{n} c_i \\ \max_{1 \leq m_3 \leq \min \left(\frac{x}{a}\right)\left\lfloor\frac{c}{c_i}\right\rfloor)}\left\{g_k\left(m_k\right) \cdot G[k-1]\left(x-c_k m_k\right)\right\}, & x \geq \sum_{i=k}^{n} c_i\end{cases}$$

初始计算 $G[0](c)$, 依次求解, 即可得出策略集。
令 $\mathrm{c}^{\prime}=\mathrm{c}-\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{c_i}, \mathrm{m}_1^{\prime}=\mathrm{m}_1-1$, 则
转化问题描述:

$$\max \prod_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{g_i}\left(\mathrm{m}_1^{\prime}\right)$$

约束条件:

$$\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{m}_1^{\prime} \mathrm{c_i} \leq \mathrm{c}^{\prime}, \quad 0 \leq \mathrm{m}_1^{\prime} \leq\left\lfloor\frac{\mathrm{c}-\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{c_i}}{\mathrm{c_n}}\right\rfloor$$

$$G[0](x)=\begin{cases}\prod_{i=1}^{n} g_i(1), & 0 \leq x \leq c-\sum_{i=1}^{n} c_i \\ -\infty, & \text{else}\end{cases}$$

$$G[1](x)=\begin{cases}-\infty, & x<c_1 \\ \max_{1 \leq m_1 \leq \min}\left\{\left[\frac{x}{c_1}\right] \mid\left\lfloor\left[\frac{c^{\prime}}{c_1}\right\}\right. & \left\{g_1\left(m_1\right) \cdot G[0]\left(x-c_1 m_1\right)\right\}, \quad x \geq c_1\end{cases}$$

$$G[k](x) = \begin{cases} -\infty, & x < c_k \\ \max_{1 \leq m_{\mathrm{y}} \leq \min\left(\lfloor \frac{x}{c_1} \rfloor \middle\lfloor \left[\frac{c'}{c_3} \right\}\right.} \{g_k\left(m_k\right) \cdot G[k-1]\left(x - c_k m_k\right)\}, & x \geq c_k \end{cases}$$