100

计算机学院《算法设计与分析》第二次作业

计算机学院　20373673　于敬凯

October 16, 2022

1　小跳蛙问题

20

给定 n 块石头，依次编号为 1 到 n，第 i 块石头的高度是 h_i，青蛙最远跳跃距离为 k。

现有一只小跳蛙在第 1 块石头上，它重复一下操作，直到它到达第 n 块石头：

若它当前在第 i 块石头上，则可以跳到第 $j(j+1 \le min(i+k,n))$ 块石头上，耗费的体力为 $|h_i - h_j|$。

试设计算法求它最少耗费多少体力可以到达第 n 块石头。

1.1　状态设计

用 $f[i]$ 表示小跳蛙跳到第 i 块石头上时耗费的最少体力。

1.2　状态转移

$$f[i] = \min_{j=max\{i-k,1\}}^{j \le i-1} \{f[j] + |h_i - h_j|\}$$

上述状态转移方程的含义是，对于状态 $f[i]$，其可以从不越界且在步长范围内（$j = max\{i-k,1\}$）的状态转移而来，根据题目要求选择使得 $f[j] + |h_i - h_j|$ 最小的前置状态 $f[j]$。

1.3　边界条件

初始时，小跳蛙在第 1 块石头上，不需要耗费体力，因此 $f[1] = 0$。

1.4　目标状态

由题目知，目标状态即到达第 n 块石头的最小代价为 $f[n]$。

1.5　算法与伪代码

根据上述对状态转移方程的分析，可以得到算法如下：

首先为初始状态赋值，即 $f[1] = 0$

然后从 $i = 2$ 开始遍历，对每个 i，遍历 j 从 $max\{i-k,1\}$ 到 $i-1$，取 $min\{f[j] - |h_i = h_j\}$ 赋值给 $f[i]$，遍历结束即得到答案 $f[n]$

Algorithm 1: 小跳蛙问题——动态规划

```
Input: 正整数 n，为石头数量，长度为 n 的数组 h[] 表示石头高度
Output: 到达第 n 块石头耗费的最小体力
function main(n, h[]):
    // 小跳蛙初始在第1块石头上，不耗费体力
    f[1] = 0;
    for i ← 2 to n do
        // 初始化f[i]为极大值，方便后续比较
        f[i] = +∞;
    end
    for i ← 2 to n do
        for j ← max{i − k, 1} to i − 1 do
            // 状态转移
            f[i] = min{f[i], f[j] + |h_i − h_j|};
        end
    end
    return f[n];
end
```

1.6 时间复杂度分析

由上述状态转移方程和伪代码知，遍历石头从 1 到 n 的时间复杂度是 $O(n)$；每次状态转移的时间复杂度是 $O(k)$。综合上述讨论可以知道总时间复杂度 $T(k) = O(nk)$

2 二进制串变换问题 20

给定两个长度均为 n 的仅由 0 和 1 组成的字符串 a 和 b，你可以对串 a 进行如下操作：

1. 对任意 $i, j (1 \le i, j \le n)$，交换 a_i 和 a_j，操作代价为 $|i - j|$；

2. 对任意 $i (1 \le i \le n)$，取反 a_i，操作代价为 1；

请你设计算法计算将串 a 变为串 b 所需的最小代价（只能对串 a 进行操作），写出伪代码并分析算法的时间复杂度。

2.1 状态设计

用 $f[i]$ 表示将串 $a[1...i]$ 变为 $b[1...i]$ 的最小代价。

2.2 状态转移

$$f[i] = min\begin{cases} f[i-1] \text{ ,若} a[i] == b[i] \\ f[i-1]+1 \text{ ,若} a[i] \neq b[i] \\ f[i-2]+1 \text{ ,若} a[i] == b[i-1] \text{且} a[i-1] == b[i] \end{cases}$$

上述状态转移的含义是，当 $f[i]$ 从 $f[i-1]$ 直接转移而来时，只需根据操作 2，即当 $a[i]! = b[i]$ 时代价 $+1$；而当且仅当从 $f[i-2]$ 转移而来且 $a[i-1], a[i]$ 相邻两位需要取反操作时，操作 1 可能更优从而被选择。

在后面的算法与伪代码中，将更进一步讨论和证明这一状态转移方程的正确性。

2.3 边界条件

初始时，串 a 和串 b 为空，无需任何操作，因此 $f[0]=0$

2.4 目标状态

由题目知，目标状态即为 $f[n]$，为将串 $a[1...n]$ 变为 $b[1...n]$ 的最小代价。

2.5 算法与伪代码

对于操作 1，当且仅当 $|i-j|\le 1$ 时，其比操作 2 更优，因此对于每个子情况只需要至多向前判断两位偏移量即可。

根据上述对状态设计、状态转移、边界条件与目标状态的分析，可以得到算法如下：

首先为初始状态赋值，即 $f[0]=0$

然后从 $i=2$ 开始遍历，对每个 i，进行上述状态转移部分介绍的操作，对三种可能的情况取最小值，遍历结束即得到答案 $f[n]$

Algorithm 2: 二进制串变换问题——动态规划

Input: 正整数 n，为字符串长度，长度为 n 的字符串 $a[]$ 和字符串 $b[]$，含义如题意所示

Output: 按照题目操作规则将 a 变换为 b 的最小代价

function $main(n, a[], b[])$**:**
    // 对边界条件赋初值
    $f[0]=0$;
    for $i \leftarrow 1$ **to** n **do**
        if $a[i]==b[i]$ **then**
            $f[i]=f[i-1]$;
        else
            $f[i]=f[i-1]+1$;
        end
        if $i>1$ **and** $a[i]==b[i-1]$ **and** $a[i-1]==b[i]$ **then**
            $f[i]=max\{f[i], f[i-2]+1\}$;
        end
    end
    return $f[n]$;
end

2.6 时间复杂度分析

由上述状态转移方程和伪代码知，遍历字符串 a 时间复杂度为 $O(n)$，状态转移的时间复杂度是 $O(1)$，因此总时间复杂度 $T(n)=O(n)$。

3 球队组建问题

20

有 $2n$ 个学生分为两派，每排有 n 个人，从左至右分别编号为 $1,2,\cdots,n$，如图所示。现在请你在这两排小学生中挑选出一些学生组成一支球队，挑选出的学生编号必须是严格递增的（编号相同的两名学生最多只能取出其中一个）。此外，为避免球队中的队员都来自同一排，不能同时选择同一排相邻的两名学生（例如，若选择第一排的 5 号同学，就不能再选择第一排的 4 号和 6 号同学）。组建队伍的总人数没有限制。

给出同学们的身高数据 $h_{i,j}$，$h_{1,k}(1\leq k\leq n)$ 表示第一排同学的身高，$h_{2,k}(1\leq k\leq n)$ 表示第二排同学的身高。请你设计算法使组建成的球队中队员的身高之和最大，写出伪代码并分析算法的时间复杂度。

3.1 状态设计

用 $f[i,j](j=1,2,3)$ 分别表示第 i 编号选择来自第 1 排的同学（针对 $j=1$）、第 i 编号选择来自第 2 排的同学（针对 $j=2$）、第 i 编号空选（针对 $j=3$）三种情况。

3.2 状态转移

由题意知，不能选择同一排相邻的两名学生，因此可以分如下情况讨论：

1. 编号 i 选择来自第 1 排的学生（$j=1$），则编号 $i-1$ 可以选择来自第 2 排的学生（$j=2$）或不选择学生（$j=3$）

2. 编号 i 选择来自第 2 排的学生（$j=2$），则编号 $i-1$ 可以选择来自第 1 排的学生（$j=1$）或不选择学生（$j=3$）

3. 编号 i 不选择学生（$j=3$），则编号 $i-1$ 可以选择来自第 1 排的学生（$j=1$）或选择来自第 2 排的学生（$j=2$）

根据上述分类讨论，可以写出状态转移方程如下：

$$f[i][j]=\max_{j'\in\{1,2,3\}\cap j'\neq j}\{f[i-1][j']\}+\begin{cases}h[1][i]\ ,\text{若}j=1\\h[2][i]\ ,\text{若}j=2\\0\ ,\text{若}j=3\end{cases}$$

同时，为了记录组队方案，设计 $pre[i][j]$ 表示编号 i 选择来自第 j 排的学生时（$j=3$ 意味着该编号空选）编号 $i-1$ 选择的同学。

$$pre[i][j]=\begin{cases}2\ ,\text{若}j=1\text{且}f[i-1][2]>f[i-1][3]\\3\ ,\text{若}j=1\text{且}f[i-1][3]>f[i-1][2]\\1\ ,\text{若}j=2\text{且}f[i-1][1]>f[i-1][3]\\3\ ,\text{若}j=2\text{且}f[i-1][3]>f[i-1][1]\\1\ ,\text{若}j=3\text{且}f[i-1][1]>f[i-1][2]\\2\ ,\text{若}j=3\text{且}f[i-1][2]>f[i-1][1]\end{cases}$$

3.3 边界条件

初始时，$f[1][1]=h[1][1],f[1][2]=h[2][1],f[1][3]=0$

3.4 目标状态

有题目知，目标状态为 $max\{f[n][1],f[n][2],f[n][3]\}$

3.5 算法与伪代码

根据上述对状态设计、状态转移、边界条件与目标状态的分析，可以得到算法如下：

首先为初始状态赋值，即 $f[1][1]=h[1][1],f[1][2]=h[2][1],f[1][3]=0$

然后从 $i=2$ 开始遍历，对每个 i，按照上述状态转移方程更新 $f[i][1],f[i][2],f[i][3]$，分别代表第 i 编号选择来自第 1 排编号 i 的同学、选择来自第 2 排编号 i 的同学和不选择任何同学。

Algorithm 3: 球队组建问题——动态规划

Input: 正整数 n，为数组长度，长度为 n 的数组 $h[1][1\dots n]$ 和数组 $h[2][1\dots n]$，含义如题意所示

Output: 最大球队队员身高和

```
function main(n, h[1][1...n], h[2][1...n]):
    // 初始边界状态赋值
    f[1][1] = h[1][1], f[1][2] = h[2][1], f[1][3] = 0;
    // 选择方案初始状态赋值，表示这些方案前一个都是
    pre[1][1] = pre[1][2] = pre[1][3] = 0;
    for i ← 2 to n do
        // 状态转移
        if f[i−1][2] > f[i−1][3] then
            f[i][1] = f[i−1][2] + h[1][i];
            pre[i][1] = 2;
        else
            f[i][1] = f[i−1][3] + h[1][i];
            pre[i][1] = 3;
        end
        if f[i−1][1] > f[i−1][3] then
            f[i][2] = f[i−1][1] + h[2][i];
            pre[i][2] = 1;
        else
            f[i][2] = f[i−1][3] + h[2][i];
            pre[i][2] = 3;
        end
        if f[i−1][1] > f[i−1][2] then
            f[i][3] = f[i−1][1];
            pre[i][3] = 1;
        else
            f[1][3] = f[i−1][2];
            pre[i][3] = 2;
        end
    end
    max_val, max_i = max{f[n][1], f[n][2], f[n][3]}, 最大f对应的第二维参数;
    // 初始化答案数组、下标指针和编号指针，其中答案数组中的元素是(编号，学生来源)元组
    case[], cur_i, ptr = [], max_i, n;
    while ptr >= 1 do
        case.add((ptr, cur_i));
        cur_i = pre[ptr][cur_i];
        ptr = ptr − 1;
    end
    return max_val, case;
end
```

3.6 时间复杂度分析

遍历 i 从 1 到 n 的时间复杂度是 $O(n)$，每次状态转移的时间复杂度是 $O(1)$，总时间复杂度 $T(n) = O(n)$。

4 括号匹配问题

20

定义合法的括号串如下：

1. 空串是合法的括号串；

2. 若串 s 是合法的，则 (s) 和 $[s]$ 也是合法的；

3. 若串 a,b 均是合法的，则 ab 也是合法的。

现在给定由 $'[','\,]'$ 和 $'(','\,)'$ 构成的字符串，请你设计算法计算该串中合法的子序列的最大长度，写出伪代码并分析算法的时间复杂度。例如字符串 ”([(])])”，最长的合法子序列 ”([()])” 长度为 6。

4.1 状态设计

用 $f[i][j]$ 表示串 $s[i][j]$ 中合法子序列的最大长度。

4.2 状态转移

考虑合法子序列的生成方式，第一种是利用规则 2 将一个合法子序列用 () 或 [] 包裹起来；第二种是利用规则 3 将两个合法的子序列链接起来。针对这两种生成方式，可以得到如下状态转移方程。

$$f[i][j] = max\begin{cases} f[i+1][j-1]+2, \text{若s[i]==’(’ } \textbf{and} \text{ s[j]==’)’或s[i]==’[’ } \textbf{and} \text{ s[j]==’]’} \\ f[i][k]+f[k+1][j], \text{遍历} i<k<j \end{cases}$$

4.3 边界条件

初始时，$f[i][i] = 0, 1 \le i \le n$，表示长度为 1 的子序列的最长合法子序列的长度都为 0。

4.4 目标状态

由题目知，目标状态即为整个 s 的最大合法子序列长度 $f[1][n]$。

4.5 算法与伪代码

根据上述对状态设计、状态转移、边界条件与目标状态的分析，可以得到算法如下：

首先为初始状态赋值，即 $f[i][i] = 0, 1 \le i \le n$

然后从 $i = 1, j = 1$ 开始遍历，进行如下状态转移：

$$f[i][j] = max\begin{cases} f[i+1][j-1]+2, \text{若s[i]==’(’ } \textbf{and} \text{ s[j]==’)’或s[i]==’[’ } \textbf{and} \text{ s[j]==’]’} \\ f[i][k]+f[k+1][j], \text{遍历} i<k<j \end{cases}$$

其中，对于第一种状态转移，其自己构成了递归子问题结构；对于第二种状态转移，可以通过枚举 k 即第一个合法子序列结束的位置来遍历找到最长的合法子序列。

Algorithm 4: 括号匹配问题——动态规划

Input: 正整数 n，为串 s 长度，长度为 n 的串 $s[]$

Output: 最长合法子序列长度

```
function main(n, s[]):
    for i ← 1 to n do
        // 为边界初始状态赋值
        f[i][i] = 0;
    end
    for l ← 2 to n do
        for i ← 1 to  n − l + 1 do
            j = i + l − 1;
            if s[i] ==′ (′  and  s[j] ==′)′ or s[i] ==′ [′ and s[j] ==′]′ then
                f[i][j] = f[i + 1][j − 1] + 2;
            end
            for k ← i to j − 1 do
                f[i][j] = max{f[i][j], f[i][k] + f[k + 1][j]};
            end
        end
    end
    return f[1][n];
end
```

4.6 时间复杂度分析

在本算法中共有 n^2 种状态，该部分遍历时间复杂度是 $O(n^2)$，状态转移时间复杂度是 $O(n)$，因此总时间复杂度是 $O(n^3)$。

5 箱子问题 20

给定 n 种箱子 $a_1, \cdots a_n$，第 i 种箱子 a_i 可表示为 $h_i \times w_i \times d_i$ 的长方体。请用这些箱子搭建一个尽可能高的塔：如果一个箱子 A 要水平的放在另一个箱子 B 上，那么要求箱子 A 底面的长和宽都严格小于箱子 B。可以任意旋转箱子，每种箱子可以用任意次。

设计一个算法求出一个建塔方案使得该塔的高度最高，写出伪代码并分析算法的时间复杂度。

例如给定 $n = 1$ 种箱子，其可表示 $3 \times 4 \times 5$ 的长方体，建塔方案如下：

1. 最底层，放置一个以 4×5 为底面的箱子，该箱子高度为 3；

2. 第二层，放置一个以 3×4 为底面的箱子，该箱子高度为 5。

此时该塔高度最高，为 $3 + 5 = 8$。

5.1 题目分析与预处理

首先分析题目中的箱子可能的使用情况，可以发现，对于每个箱子有 3 种形态（对高的选择进行 $\binom{3}{1}$ 种方案）。为了使得后续计算有序，我们首先使得每个箱子的长度大于等于宽度（若长度小于宽度，交换长度和宽度的值即可）。对于每种形态，其在使用一次后，不能被再次使用（其长度和宽度恰等于自身）。因此，我们获得了 $3n$ 个不同的且只能使用一次的箱子。

考虑一个子情况，当分析第 i 个箱子时，我们依赖于前 $i-1$ 个子情况转移而来，这要求前 $i-1$ 个子情况已经计算出正确结果，因此，我们首先对 $3n$ 个箱子按照长度进行升序排序，且当长度相等时按照宽度升序排序。这样排序处理后，可以保证对于编号为 $i-1$ 的任意箱子，编号大于 $i-1$ 的箱子都不能

被放在编号为 $i-1$ 的箱子上面，因此分析编号为 i 的箱子时可以保证其依赖的前 $i-1$ 个箱子都已经正确计算完毕。

以上完成了预处理部分，对于动态规划设计部分，后续将详细给出。

5.2 状态设计

用 $f[i]$ 表示以第 i 个箱子为底部箱子时的最大塔高。

用 $pre[i]$ 表示以第 i 个箱子上方紧挨着的箱子的编号，如果其上方没有箱子，则为 0。

5.3 状态转移

$$f[i] = max \begin{cases} h[i] \\ f[j] + h[i] \text{ ,若} d[j] < d[i] \text{且} w[j] < w[i] \end{cases}$$

上述状态转移方程的含义是，$f[i]$ 从满足 $1 \leq j < i$ 且满足 $w_j < w_i \ and \ d_j < d_i$ 的 j 中的最大 $f[j] + h_i$ 转移而来。

5.4 边界条件

初始对有序的 $3n$ 个箱子中的第 1 个箱子状态赋值 $f[1] = h[1]$。

初始时对每个箱子赋值 $pre[i] = 0$，表示其上面目前没有别的箱子。

5.5 目标状态

由题意知，目标状态高度即为 $max\{f[i]\}, 1 \leq i \leq 3n$；而目标建塔方案则通过找到使得 $f[i]$ 最大的 i，通过向前递归得到的箱子编号串。

5.6 算法与伪代码

在题目分析与预处理阶段，我们已经得到了有序的 $3n$ 个箱子和其对应的长宽高，并在边界条件阶段赋了初值。

接下来，我们考虑任意一般状态，不妨假设此时正在计算 $f[i]$，对于 $f[j], 1 \leq j \leq i-1$，如果其满足长度和宽度分别小于第 i 个箱子的长度和宽度，那么说明这是一种可能的状态转移路径，遍历满足这样条件的 j，取最大的 $f[j]$，再加上第 i 个箱子的高度 $h[i]$ 即得到 $f[i]$；如果没有满足条件的 j，则使 $f[i] = h[i]$，表明其是该方案下的顶层箱子。

对于建塔方案记录，如果上述计算中存在满足条件的 j，则将 $pre[i]$ 赋值为 j，否则不进行赋值，又由于在边界条件阶段为 $pre[i]$ 赋初值 0，满足状态设计，即表明其上方没有箱子。

Algorithm 5: 箱子问题——动态规划

Input: 正整数 n，为箱子种类数，长度为 n 的数组 $h[], w[], d[]$ 分别表示 n 种箱子的高度 $height$，宽度 $width$，长度（深度）$dipth$

Output: 最大塔高和对应的建塔方案

```
function main(n, h[], w[], d[]):
    将每种箱子按照高度进行旋转得到三种不同形态，分别按顺序赋值给h, w, d，并按照长度 d
     进行升序排序，若长度d相同则按照宽度w进行升序排序，得到新的h, w, d数组，长度为3n
    // 为临界条件设置初始值
    for i ← 1 to 3n do
        pre[i] = 0;
        f[i] = 0;
    end
    f[1] = h[1];
    for i ← 2 to 3n do
        for j ← 1 to i − 1 do
            // 满足条件的j如果可以使状态更优，则更新f, pre
            if w[j] < w[i] and d[j] < d[i] and f[j] + h[i] > f[i] then
                f[i] = f[j] + h[i];
                pre[i] = j;
            end
        end
    end
    max_i = 0;
    max_f = 0;
    for i ← 1 to 3n do
        if max_f < f[i] then
            // 获取答案f[i]与对应的i
            max_i, max_f = i, f[i];
        end
    end
    // 获取叠塔方案，用case列表保存最大叠塔方案
    case[] = [];
    cur_i = max_i;
    // pre数组保存该箱子之上的箱子编号，遍历将所有该方案下箱子加入答案
    while cur_i ≠ 0 do
        if cur_i! = 3 then
            case.add(ptr, cur_i);
        end
        cur_i = pre[cur_i];
    end
    return f[n], case;
end
```

5.7 时间复杂度分析

使用快速排序对长宽高数组进行预排序，时间复杂度为 $O(nlogn)$，每次状态转移要考虑其之前的所有状态，时间复杂度是 $\sum_{i=1}^{3n} O(i) = O(n^2)$，综合以上，总时间复杂度为 $T(n) = O(n^2)$。