西安交通大學

XI'AN JIAOTONG UNIVERSITY

算法设计与分析

第二次作业

课程名称：	算法设计与分析
姓名：	凌晨
学院：	软件学院
专业：	软件工程
学号：	**2214414320**

2023 年 10 月 14 日

一、 3-4

1. 题目描述

3-4 数字三角形问题。

问题描述：给定一个由 n 行数字组成的数字三角形，如图 3-7 所示。试设计一个算法，计算出从三角形的顶至底的一条路径，使该路径经过的数字总和最大。

4 5 2 6 5

图 3-7 数字三角形

算法设计：对于给定的由 n 行数字组成的数字三角形，计算从三角形的顶至底的路径经过的数字和的最大值。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行是数字三角形的行数 n，$1 \leqslant n \leqslant 100$。接下来 n 行是数字三角形各行中的数字。所有数字在 0～99 之间。

结果输出：将计算结果输出到文件 output.txt。文件第 1 行中的数是计算出的最大值。

输入文件示例	输出文件示例
input.txt	output.txt
5	30
7	
3 8	
8 1 0	
2 7 4 4	
4 5 2 6 5	

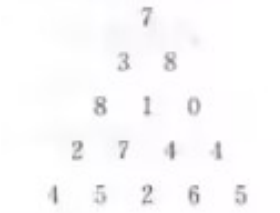


图 3-7 数字三角形

2. 算法设计

(1) 最优子结构

题目描述非常清楚，不再赘述路径限制以及算法任务。假设 x[i][j] 为第 i 行第 j 列的数字，dp[i][j] 为从底到第 i 行第 j 列的数字之和的最大值，那么应该有以下最优子结构。

而 dp[i+1][j],dp[i+1][j+1] 也应该为对应的最大值，下面使用反证法证明：

证明:
假设: dp[i+1][j],dp[i+1][j+1]不为对应的最大值，则一定存在一个var变量满足:
var>dp[i+1][j] and var>dp[i+1][j+1]
因此，根据公式可知:
dp[i][j] < var + x[i][j]
违背了定义，因此假设不成立。

```
即dp[i+1][j],dp[i+1][j+1]一定为对应的最大值
```

(2) 递归计算最优值

由最优子结构性质可以知道：

$$dp[i][j] = max(dp[i+1][j], dp[i+1][j+1]) + x[i][j] \tag{1}$$

下面可以编写算法：

```
def solution1(n, dp):
    for i in range(n - 2, 0):
        for j in range(0, i):
            dp[i][j] = max(dp[i + 1][j], dp[i + 1][j + 1]) + dp[i][j]
    return dp[0][0]
```

算法正确性不必赘述，需要注意的是，算法在边界条件和录入数据等地方存在忽略，老师需要注意!!

(3) 计算复杂度

- 空间复杂度

  由于算法直接在原有的数组上进行操作，空间复杂度为 O(1);

- 时间复杂度

  相当于遍历全部数据得到了最优解，时间复杂度为 $O(n^2)$;

二、 3-6

1. 题目描述

3-6 租用游艇问题。

问题描述：长江游艇俱乐部在长江上设置了 n 个游艇出租站 1, 2, …, n。游客可在这些游艇出租站租用游艇，并在下游的任何一个游艇出租站归还游艇。游艇出租站 i 到游艇出租站 j 之间的租金为 $r(i,j)$（$1 \leqslant i < j \leqslant n$）。试设计一个算法,计算出从游艇出租站 1 到游艇出租站 n 所需的最少租金。

算法设计：对于给定的游艇出租站 i 到游艇出租站 j 之间的租金为 $r(i,j)$（$1 \leqslant i < j \leqslant n$），计算从游艇出租站 1 到游艇出租站 n 所需的最少租金。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行中有 1 个正整数 n（$n \leqslant 200$），表示有 n 个游艇出租站。接下来的 $n-1$ 行是 $r(i,j)$（$1 \leqslant i < j \leqslant n$）。

结果输出：将算出的从游艇出租站 1 到游艇出租站 n 所需的最少租金输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
3	12
5 15	
7	

2. 算法设计

(1) 最优子结构

该问题具有最优子结构性质，具体描述如下：

r[i][j] 为第 i 行第 j 列的租金。dp[i][j] 为从 i 到 j 的最少租金，对于任意 k，k 满足 i<k<j，都有 dp[i][k] 和 dp[k][j] 为最少租金。

证明如下：

证明:
假设: `dp[i][j] = dp[i][k] + dp[k][j]`,但`dp[i][k],dp[k][j]`不为对应的最小值，则一定存在一个`var`变量满足:
`var < dp[i+1][j] or var < dp[i+1][j+1]`
因此，根据公式可知:
`dp[i][j] > var + dp[k][j] or dp[i][j] > dp[i][k] + var or dp[i][j] > 2*var`
违背了`dp[i][j]`定义，因此假设不成立。
即`dp[i+1][j],dp[i+1][j+1]`一定为对应的最小值

(2) 递归计算最优值

递归公式如下：

$$dp[i][j] = \min_{i<k<j}(dp[i][k] + dp[k][j], r[i][j]) \tag{2}$$

不再赘述公式的正确性，对于本问题而言，从底向上开始遍历，最后返回 dp[1][n] 的值即可。下面给出算法：

```
def solution2(n, r):
    dp = [[999] * (n - 1) for i in range(n)] # 999只是代表一个很大的数字
    for i in range(n):
        dp[i][i] = 0 # 给出发地和终止地一致的地方赋值为0
    for r in range(0, n):
        for i in range(n - r - 1):
            j = i + r - 1
            for k in range(i, j):
                dp[i][j] = min(dp[i][k] + dp[k][j] + r[i][j])
    return dp[0][n - 1]
```

(3) 计算复杂度

- 空间复杂度

  由于算法创建了 dp 数组，空间复杂度为 $O(n^2)$;

- 时间复杂度

  主要是三重循环，时间复杂度为 $O(n^3)$;

三、 3-11

1. 题目描述

3-11 正则表达式匹配问题

问题描述：许多操作系统采用正则表达式实现文件匹配功能。一种简单的正则表达式由英文字母、数字及通配符“*”和“?”组成。“?”代表任意一个字符，“*”则可以代表任意多个字符。现要用正则表达式对部分文件进行操作。

试设计一个算法，找出一个正则表达式，使其能匹配的待操作文件最多，但不能匹配任何不进行操作的文件。所找出的正则表达式的长度还应是最短的。

算法设计：对于给定的待操作文件，找出一个能匹配最多待操作文件的正则表达式。

数据输入：由文件 input.txt 提供输入数据。文件由 n（$1 \leqslant n \leqslant 250$）行组成。每行给出一个文件名。文件名由英文字母和数字组成。英文字符要区分大小写，文件名长度不超过 8 个字符。文件名后是一个空格符和一个字符“+”或“−”。“+”表示要对该行给出的文件进行操作，“−”表示不进行操作。

结果输出：将计算出的最多文件匹配数和最优正则表达式输出到文件 output.txt。文件第 1 行中的数是计算出的最多文件匹配数，第 2 行是最优正则表达式。

输入文件示例	输出文件示例
input.txt	output.txt
EXCHANGE +	3
EXTRA +	*A*
HARDWARE +	
MOUSE −	
NETWORK −	

2. 分析

设当前考察的正则表达式为 s，当前考察的文件为 f。用 match(i,j) 来表达 s[i...j] 和 f[i...j] 的匹配情况。当 s[i...j] 能匹配 f[i...j]，match(i,j)=1，反之，match(i,j)=0。存在下列递归式：

$$match(i,j)=\begin{cases}1 & match(i-1,j-1)=1 \quad \text{s[i]='?'}\\ 1 & match(i-1,j-1)=1 \quad \text{s[i]=f[i]}\\ 1 & match(i-1,k)=1 \quad \text{s[i]='*'}\\ 0 & else\end{cases} \tag{3}$$

由此可以设计出最优匹配的回溯法：

```
def search(len):
    len = len + 1
    if len == 1 and s[len - 1] != '*':
        s[len] = '?'
        if check(len):
            search(len)
        s[len] = '*'
        if check(len):
            search(len)
    for i in range(p[len - 1]):
        s[len] = search_arr[len - 1][i]
        if check(len):
            search(len)
```

check() 函数是用来检查当前匹配情况，而 search_arr 是将出现的字符按照频率进行排序储存的数组。其余不再解释。

为了实现正则表达式匹配问题，我们需要讲解 DFA 算法。

3. DFA 算法

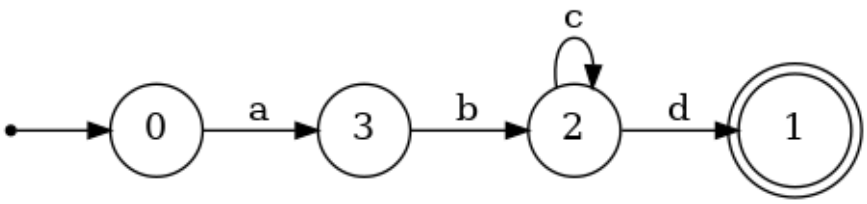
(1) 概念

DFA，全称 Deterministic Finite Automaton 即确定有穷自动机：从一个状态通过一系列的事件转换到另一个状态，即 state -> event -> state。

- 确定：状态以及引起状态转换的事件都是可确定的，不存在“意外”。
- 有穷：状态以及事件的数量都是可穷举的。

(2) 正则表达式实现

一般而言，根据输入的 pattern，构建不同的状态转移图，如 pattern 为“abc*d”图如下：



在实际编程中，可以通过字典和状态数组来进行实现，不再给出具体代码。

4. 计算复杂度

- 空间复杂度

  由于算法创建了查询数组，数组长度取决于输入，空间复杂度为 O(n);

- 时间复杂度

  递归时间公式如下

$$T(n) = T(n-1) + n$$

  所以时间复杂度为 O(n^2);

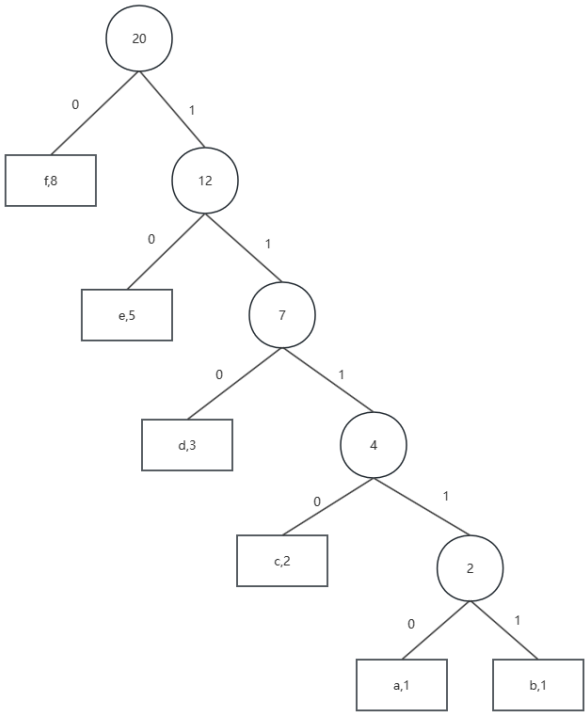
四、 4-3

1. 题目描述

4-3 字符 a～h 出现的频率恰好是前 8 个 Fibonacci 数，它们的哈夫曼编码是什么？将结果推广到 n 个字符的频率恰好是前 n 个 Fibonacci 数的情形。

2. 解答

根据哈夫曼算法，可以得到对于 aħ 的哈夫曼树。



	编码
a	11110
b	11111
c	1110
d	110
e	10
f	0

3. n 个字符

通过对 Fibonacci 数的定义我们可以得知：

$$x_n = x_{n-1} + x_{n-2}$$

当使用哈夫曼编码时，最小的两个数合并后的数字总是处于去除该两个数的序列的倒数第二小。

因此，字符 a 为 n 个 1 组成的编码，而后第 i 个字符为长度为 n-i+1，第一位为 0，其余位为 1 的编码。

前缀码如下：

	编码
a	1...11
b	1...10
c	1...0
...	...
$char_{n-1}$	10
$char_n$	0

五、 4-4

1. 题目描述

4-4 磁盘文件最优存储问题。

问题描述：设磁盘上有 n 个文件 $f_1, f_2, \cdots, f_n$，每个文件占用磁盘上的 1 个磁道。这 n 个文件的检索概率分别是 $p_1, p_2, \cdots, p_n$ 且 $\sum_{i=1}^{n} p_i = 1$。磁头从当前磁道移到被检信息磁道所需的时间可用这两个磁道之间的径向距离来度量。如果文件 f_i 存放在第 i（$1 \leqslant i \leqslant n$）道上，则检索这 n 个文件的期望时间是 $\sum_{1 \leqslant i \leqslant j \leqslant n} p_i p_j d(i,j)$。式中，$d(i,j)$是第 i 道与第 j 道之间的径向距离 $|i-j|$。

磁盘文件的最优存储问题要求确定这 n 个文件在磁盘上的存储位置，使期望检索时间达到最小。试设计一个解此问题的算法，并分析算法的正确性与计算复杂性。

算法设计：对于给定的文件检索概率，计算磁盘文件的最优存储方案。

数据输入：由文件 input.txt 给出输入数据。第 1 行是正整数 n，表示文件个数。第 2 行有 n 个正整数 a_i，表示文件的检索概率。实际上第 k 个文件的检索概率应为 $a_k \Big/ \sum_{i=1}^{n} a_i$。

结果输出：将计算的最小期望检索时间输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
5	0.547396
33 55 22 11 9	

2. 分析

算法采用贪心选择策略，每次将最大的数取出，放置最中间的位置，关于最中间的位置有如下规则：

(1) 靠近中间磁道

(2) 尽量离大数近点

比如：有 _33 55 22 _

当插入 11 时，有最左边和最右边选择，这时候，选择最左边的位置，因为 33>22, 离大数近

下面进行证明，通过该贪心算法可以得到最小值：假设我们现在通过上述规则在磁道上排好了相应的队列 s，假设 f1,f2,...,fi,...,fn。

如果存在更好的队列 s'，f1',f2',...,fi',...,fn',s' 一定能通过 s 中的值两两交换而得到，不失一般性，每次交换都可以是处于“中间”的大数 i 与处于“边缘”的小数 j 进行交换。

我们仅考虑一次交换过程，假设变换前的用时 t，变换后用时 t'。

$$t - t' = \sum p_i * p_j * d(i,j) - \sum p'_i * p'_j * d(i,j)$$

将上式子化简计算可得：

$$t - t' < 0$$

所以，假设存在更好的队列不成立。证明完毕！

3. 算法实现

```
def solution4(n, arr):
    p_arr = [-1 for i in range(n)]
    sort(arr) // 按照从大到小进行排序arr
    for i in range(n):
        insert(p_arr,arr[i]) // 按照刚刚制定的规则插入新建的p_arr
    t_sum = 0 // 表示最小期望检索时间
    for i in range(n):
        for j in range(i,n):
            t_sum = t_sum + p_arr[i]*p_arr[j]*(j-i)/(sum(p_arr)*sum(p_arr))
    return t_sum
```

4. 计算复杂度

- 空间复杂度

  由于算法创建了数组，数组长度取决于输入，空间复杂度为 $O(n)$;

- 时间复杂度排序花费 $O(nlogn)$

  在计算最小期望检索时间，时间复杂度为 $O(n^2)$

  所以时间复杂度为 $O(n^2)$;

六、 4-6

1. 题目描述

4-6 最优服务次序问题。

问题描述：设有 n 个顾客同时等待一项服务，顾客 i 需要的服务时间为 t_i（$1 \leqslant i \leqslant n$）。应如何安排 n 个顾客的服务次序才能使平均等待时间达到最小？平均等待时间是 n 个顾客等待服务时间的总和除以 n。

算法设计：对于给定的 n 个顾客需要的服务时间，计算最优服务次序。

数据输入：由文件 input.txt 给出输入数据。第 1 行是正整数 n，表示有 n 个顾客。接下来的 1 行中，有 n 个正整数，表示 n 个顾客需要的服务时间。

结果输出：将计算的最小平均等待时间输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
10	532.00
56 12 1 99 1000 234 33 55 99 812	

2. 解答

算法采用贪心选择策略，每次先服务未服务序列中的所需最小的服务时间的顾客。
证明同 4-4 证明，不再给出证明过程。
下面是伪代码实现：

```
def solution5(n, arr):
    sort(arr) // 按照从小到大进行排序arr
    t_sum = 0 // 总等待时间
    for i in range(n):
        t_sum = t_sum + arr[i] * (n-i-1)
    return t/n
```

3. 计算复杂度

- 空间复杂度

  算法仅设置了若干变量，空间复杂度为 O(1);

- 时间复杂度排序花费 O($nlogn$)

  在计算平均服务时间，时间复杂度为 O(n)

  所以时间复杂度为 O(n);

七、 4-9

1. 题目描述

4-9 虚拟汽车加油问题。

问题描述：一辆虚拟汽车加满油后可行驶 n km。旅途中有若干加油站。设计一个有效算法，指出应在哪些加油站停靠加油，使沿途加油次数最少。并证明算法能产生一个最优解。

算法设计：对于给定的 n 和 k 个加油站位置，计算最少加油次数。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 k，表示汽车加满油后可行驶 $n$km，且旅途中有 k 个加油站。接下来的 1 行中有 k+1 个整数，表示第 k 个加油站与第 k–1 个加油站之间的距离。第 0 个加油站表示出发地，汽车已加满油。第 k+1 个加油站表示目的地。

结果输出：将计算的最少加油次数输出到文件 output.txt。如果无法到达目的地，则输出“No Solution”。

输入文件示例	输出文件示例
input.txt	output.txt
7　7	4
1 2 3 4 5 1 6 6	

2. 最优子结构

根据描述可以得知，该问题具有最优子结构，即：
假设给定 n，最少加油次数为 f(n)，满足以下不等式：

$$f(n)=\begin{cases} f(n-1)+1 & \text{当剩余油量不够从 n-1 抵达 n} \\ f(n-1) & \text{当剩余油量够从 n-1 抵达 n} \end{cases} \tag{4}$$

下面进行证明：
假设，不存在最优子结构，则说明有以下两种情况：

(1) 当车辆可以不在 n-1 停留，直接到 n 时，若 f(n)!=f(n-1)，同时现实可以得知 f(n) 不可能小于 f(n+1)。因此，f(n)>f(n-1)，这不符合最少加油次数，因为可以不进行加油而达到 n 点。

(2) 当车辆必须在 n-1 停留，再到 n 时，若 f(n)!=f(n-1)+1，同时现实可以得知 f(n) 不可能小于 f(n+1)。因此，要么 f(n)=f(n-1) 要么 f(n)>f(n+1)+1，前者办不到，因为油量不够抵达 n 点，后者不符合最少加油次数，明明有最佳选择。

3. 贪心选择策略

策略如下：
每次加完油，行驶到能开到最远的加油站。
证明如下：
如果不在行驶在最远的加油站停车加油，说明有两种情况：

(1) 在达到最远加油站前，就停车加油，这样多了几次加油次数，不符合题设。

(2) 在达到最远加油站后，不停车加油，直接开走，这样无法达到下一个加油站，不符合题设。

4. 算法实现

实现伪代码如下：

```
def solution6(n, k, arr):
    park = 0 // 停车次数
    oil = n // 初始油量
    for i in range(k):
        if( n < arr[k] ):
            return "No Solution" // 所有油耗尽也到不了下一个加油站
    for i in range(k):
        if (oil < arr[k]): // 到达不了下一个点
            park = park + 1 // 停车
            oil = n // 加油
        else:
            oil = oil - arr[k] // 直接驶过该加油站
    return park
```

5. 计算复杂度

- 空间复杂度

  算法仅设置了若干变量，空间复杂度为 $\mathrm{O}(1)$;

- 时间复杂度遍历即可，时间复杂度为 $\mathrm{O}(n)$

  所以时间复杂度为 $\mathrm{O}(n)$;