深 圳 大 学 实 验 报 告

课程名称：算法设计与分析

实验项目名称：实验四 动态规划—流水线问题

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：杨烜

报告人：沈晨珩 学号：2019092121 班级：19 计科国际

实验时间：2021.5.14

实验报告提交时间：2021.5.14

教务部制

实验四 动态规划—流水线问题

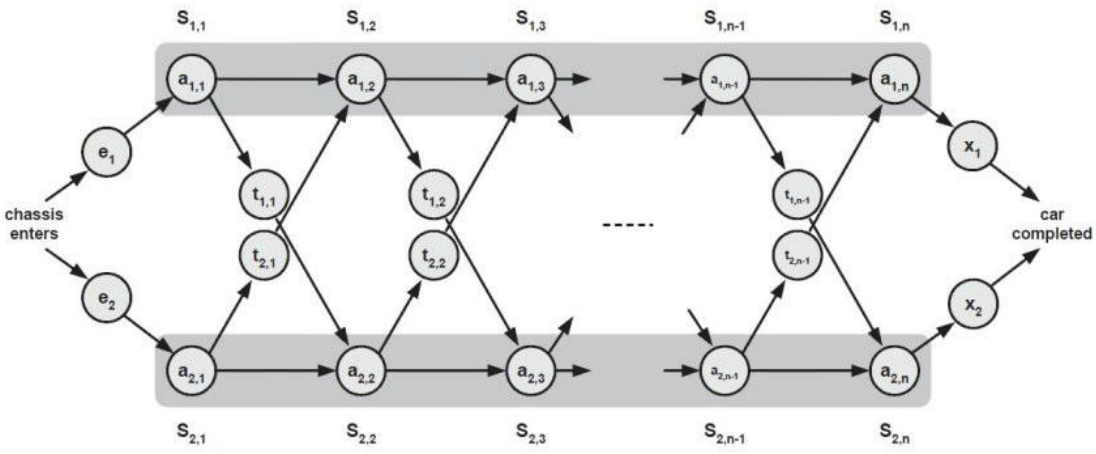
一、实验目的：

（1）掌握动态规划算法设计思想。
（2）掌握流水线问题的动态规划解法。

二、内容：

汽车厂有两条流水线，每条流水线有 n 个处理环节（station）: $S_{1,1}$，…，$S_{1,n}$ 和 $S_{2,1}$，…，$S_{2,n}$，其中下标的第一个字母表示流水线编号（流水线 1 和流水线 2）。其中 $S_{1,j}$ 和 $S_{2,j}$ 完成相同的功能，但是花费的时间不同，分别是 $a_{1,j}$, $a_{2,j}$ 。两条流水线的输入时间分别为 e_1 和 e_2, 输出时间是 x_1 和 x_2。

每个安装步骤完成后，有两个选择：
1）停在同一条安装线上，没有转移代价；
2）转到另一条安装线上，转移代价： $S_{i,j}$ 的代价是 $t_{i,j}$, j = 1，…，n - 1



问题: 如何选择安装线 1 和安装线 2 的节点组合，从而最小化安装一台车的总时间？

三、实验要求

1、给出解决问题的动态规划方程；

2、随机产生$S_{2,j}$ 、$t_{i,j}$的值，对小数据模型利用蛮力法测试算法的正确性；
3、随机产生$S_{2,j}$ 、$t_{i,j}$的值，对不同数据规模（n的值）测试算法效率，并与理论效率进行比对，请提供能处理的数据最大规模，注意要在有限时间内处理完；
4、该算法是否有效率提高的空间？包括空间效率和时间效率。

四、实验过程及结果

一：实验结果正确性展示

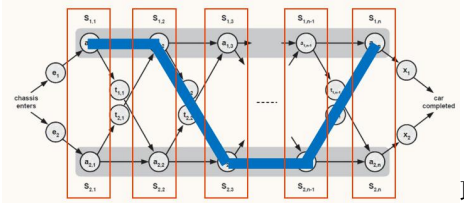
随机生成了多组 20 个结点，通过蛮力法与动态规划法分别求解，验证程序正确性，结果如下。

```
蛮力法解：7618
蛮力法路径  11111110000011100000
动态规划解：7618
动态规划路径11111110000011100000

请按任意键继续. . .
蛮力法解：8534
蛮力法路径  11111111111001110111
动态规划解：8534
动态规划路径11111111111001110111

请按任意键继续. . .
蛮力法解：8377
蛮力法路径  11111100000100011111
动态规划解：8377
动态规划路径11111100000100011111

请按任意键继续. . .
```

二：蛮力法求解

1. 算法原理

(1) 通过遍历所有可能的路径和并计算总时间的最小值。

(2) 

取任意一种可能的情况（如左图所示），可以将路径表示位 00110（0 表示上流水线，1 表示下流水线）

(3) 显然对于 n 个结点的流水线，所有可能的路线有 2^n 种可能性，可以用 0～ 2^n-1 的二进制表示所有情况。

2. 伪代码

```
for 0 to  2^n - 1:      // 遍历每一条路径
    sum += e +  结点 1
    for  二进制串的每一个位置:
        if  需要转移：
            sum +=  转移代价  +  结点 i
        else if  不需要转移
            sum +=  结点 i
    if sum < MIN:
        MIN = sum
```

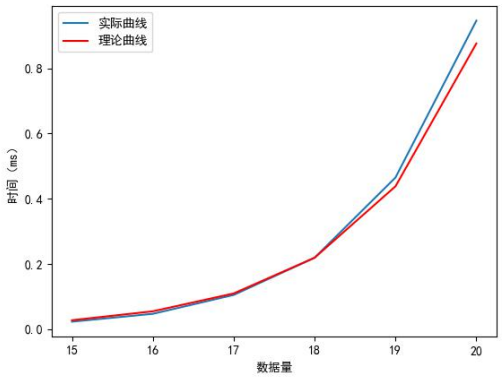
3. 复杂度分析

(1) 时间复杂度：$O(2^n)$

(2) 空间复杂度：$O(n)$

4. 数据分析

节点个数	15	16	17	18	19	20
运行时间(s)	0.023	0.047	0.105	0.219	0.465	0.946
理论运行时间(s)	0.026	0.525	0.105	0.21	0.42	0.84



三：动态规划法求解

①总时间计算

1. 算法原理

(1) 动态规划方程：

$$\begin{cases} dp1[1] = e1 + a[1][1] \\ dp2[1] = e2 + a[2][1] \\ dp1[i] = \min(dp1[i-1] + a[1][i], dp2 + t[2][i-1] + a[1][i]) & 2 \le i \le n \\ dp2[i] = \min(dp2[i-1] + a[2][i], dp1 + t[1][i-1] + a[2][i]) & 2 \le i \le n \\ res = \min(dp1 + x1, dp2 + x2) \end{cases}$$

其中参数含义：

e：输入时间　　t ：转移代价

a：花费时间　　x：输出时间

dp[i]：当前结点最短总时间　　res：最终总时间

2. 伪代码

```
dp1 =e1 + a[1][1], dp2 =e2 + a[2][1]
for i = 2 to n:
    dp1 = min(dp1 + a[1][i], dp2 + t[2][i - 1] + a[1][i])
    dp2 = min(dp2 + a[2][i], dp1 + t[1][i - 1] + a[2][i])
res = min(dp1 + x1, dp2 + x2)
```

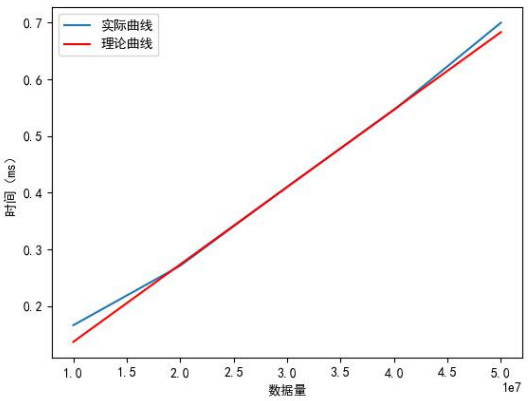
3. 复杂度分析

(1) 时间复杂度：$O(n)$

(2) 空间复杂度：$O(1)$　根据题意，不需要保存每一个结点的最优值，所以每次动态规划可以及时更新 dp 值。

4. 数据分析

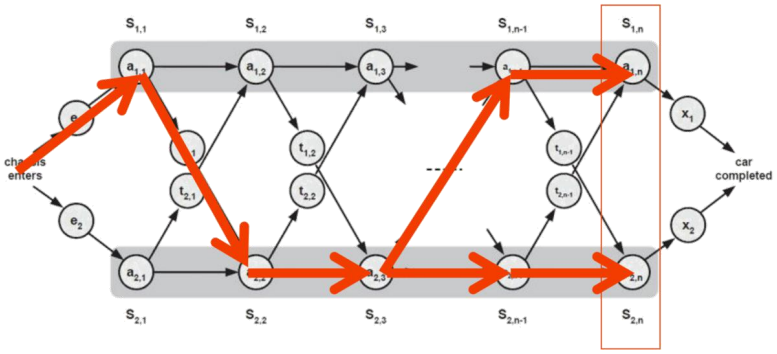
节点个数	1000w	2000w	3000w	4000w	5000w
运行时间(s)	0.166	0.271	0.41	0.546	0.7
理论运行时间(s)	0.1355	0.271	0.4065	0.542	0.6775



②路径计算

1. 路径存储原理（一）

(1) 算法原理



具体详细保存两条路径（具体生成过程见 PPT / 汇报）

如图所示，路径 1：ABBAA / 路径 2：BBBB

(2) 伪代码

dp1 =e1 + a[1][1], dp2 =e2 + a[2][1]

路线 1[1] = A, 路线 2[1] = B

for i = 2 to n:

    if dp1 + a[1][i] < dp2 + t[2][i - 1] + a[1][i]:

        dp1 = dp1 + a[1][i]

        路线 1 更新新节点 A

    else:

        dp1 = dp2 + t[2][i - 1] + a[1][i]

        路线 1 = 路线 2

        路线 1 更新新节点 A

    （对 dp2 做相同类似操作）

if dp1 + x1 < dp2 + x2:

    路线 1 为最优路径

else：

    路线 2 为最优路径

(3) 复杂度分析

① 时间复杂度：$O(n^2)$

具体生成路径过程见 PPT / 汇报，可以发现当需要转移流水线时，需要将路径进行切换（复制另一条路径），这会导致两条路径频繁的相互复制。

复制过程复杂度为$O(n)$，所以会导致算法总体复杂度$O(n^2)$。

② 空间复杂度：$O(2n)$

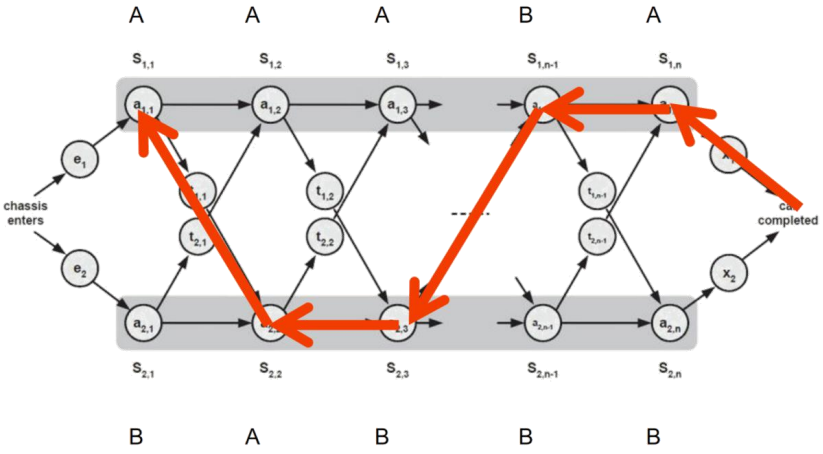
对于两条路径，直到最后加上输出代价才知道哪一条路径为更优解。所以需要保存两条路径，空间复杂度为$O(2n)$。

2. 路径存储原理（二）

(1) 算法原理

① 每一结点都保存其来自于第一/第二流水线。

② 解码过程通过回溯法即可获取路径。具体解码过程见 PPT / 汇报。



(2) 伪代码

路径存储：

```
dp1 =e1 + a[1][1], dp2 =e2 + a[2][1]
routeA[1] = A, routeB[1] = B
for i = 2 to n:
        if dp1 + a[1][i] < dp2 + t[2][i - 1] + a[1][i]:
                dp1 = dp1 + a[1][i]
                routeA  更新新节点 A
        else:
                dp1 = dp2 + t[2][i - 1] + a[1][i]
                routeA  更新新节点 B
        （对 dp2  做相同类似操作）
```

解码过程：

```
if dp1 + x1 < dp2 + x2:
        起始点为 S[1][n], t = 1
else:
        起始点为 S[2][n], t = 2
for i = n to 2
        output route[t][i]
        t = route[t][i]
```

(3) 复杂度分析

① 时间复杂度：$O(n)$

② 空间复杂度：$O(2n)$

(4) 数据分析

利用 char 数组存储路径。（用 bit 存储路径，可以更加节省空间）
Windows 下，最多开到 8-9 亿，存储 1 亿的时间大约 12s。
Linux 下，存储 1 亿的时间大约 17s，极限 510 亿 2.5 小时。
符合 O(n)的时间复杂度

由上一部分可知，路径存储即对于二进制串的存储。以下部分为对于二进制串存储的压缩算法分析。

3. 压缩算法（一）

(1) 算法原理

① n 位连续数字可以保存为 n。

② 解码过程中，AB 交替出现。所以当知道末尾结点来自 A 或 B，可以自尾向前解码。

③ 例子：

路径：A BB AAAAAA BBBB A B AA BBB AA
压缩编码：1 2 6 4 1 1 2 3 2
路径：AAAAAAAAAAA BBBBBBBBBB
压缩编码：11 11

(2) 数据离散度测试

显然，数据的离散程度对于压缩效率有着较大的影响。
测试数据：随机生成一条长度为 10 亿的路径。

转移代价：花费代价	平均连续位数	最大连续位数
1：20	2.1	28
1：10	2.2	30
1：5	2.4	40
1：1	3.7	97
2：1	4.3	147
7：1	9.1	400
8：1	10	420
9：1	10.8	500

由表中数据可以得知，当转移代价越大时，二进制连续性越大。
具体影响程序见后续数据分析。

(3) 伪代码

```
end_s = A or B
BEGIN
        s = next input
        if s == route[end]:
                code[end] += 1
        else:
                code.push_back(1)
                end_s = s
END
```

(4) 数据分析

① 测试结果：1 亿数据存储时间约为 21 秒，最大测试结果为 1550 亿 9 小时（理论上可以更大，堆内存大小即可）

② 压缩效率理论测试：

1) 二进制串长度：N

2) 最大连续位数：max

3) 平均连续位数：avg

4) 压缩效率：$\dfrac{N}{\mathrm{N}\div\mathrm{avg}\times\lceil\log_2\max\rceil}=\dfrac{avg}{\lceil\log_2\max\rceil}$

转移代价：花费代价	平均连续位数	最大连续位数	压缩效率
1：20	2.1	28	0.4366
1：10	2.2	30	0.4484
1：5	2.4	40	0.4524
1：1	3.7	97	0.5617
7：1	9.1	400	1.0638
9：1	10.8	500	1.2048

由表中数据可以得知，当转移代价越大时，压缩效率越高。当代价比例大于 7 时，压缩算法才有效。

但显然不符合实际逻辑。所以并不是一个好的压缩算法。

4. 压缩算法（二）

(1) 算法原理

① 利用哈希表存储一个字典，在生成动态二进制串时，利用 LZW 算法，对串进行压缩。

② 具体生成过程见 PPT / 汇报。

③ 举例：

路径：ABABBABABABBA

压缩编码：1 2 3 4 6 5 1

字典如下所示

字符串	key
A	1
B	2
AB	3
BA	4
ABB	5
BAB	6
BABA	7
ABBA	8

(2) 伪代码

```
BEGIN
  s = next input character;
  while not EOF
      { c = next input character;

        if s + c exists in the dictionary
          s = s + c;
        else
          { output the code for s;
            add string s + c to the dictionary with a new code;
            s = c;
          }
       }
  output the code for s;
END
```

(3) 数据分析

① 压缩效率理论测试：

1) 二进制串长度：N　　字典长度：T　　压缩后位数：M

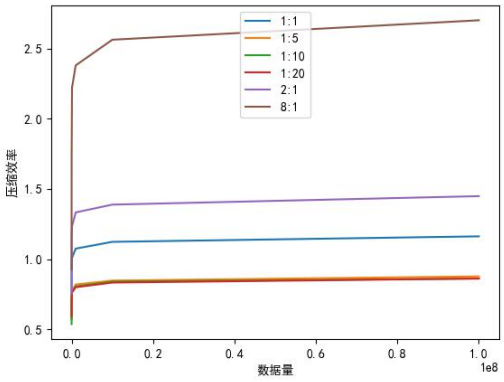
2) 压缩效率：$\frac{N}{\mathrm{M}\times\lceil\log_2 T\rceil}$

② 当转移代价：花费代价 ＝1 ：1，测试数据如下。

二进制串长度：N	字典长度：T	压缩后位数：M	压缩效率
100	56	29	0.5747
1000	292	145	0.7662
10000	1936	973	0.9343
100000	14127	7134	1.001
1000000	109871	55180	1.0660
10000000	894639	449095	1.1133
100000000	7520622	3776648	1.1512

③ 当转移代价：花费代价 ＝1 ：1，测试数据如下。

转移代价： 花费代价	2：1	8：1	1：1	1：5	1：10	1：20
100	0.7576	0.9259	0.5747	0.5747	0.5376	0.5952
1000	0.8696	1.3889	0.7692	0.6536	0.6329	0.6410
10000	1.1494	1.7241	0.9346	0.7246	0.7092	0.7092
100000	1.2346	2.2222	1.0101	0.7752	0.7634	0.7634
1000000	1.3333	2.3810	1.0753	0.8197	0.8065	0.8000
10000000	1.3889	2.5641	1.1236	0.8475	0.8403	0.8333
100000000	1.4493	2.7027	1.1628	0.8772	0.8621	0.8621



由表中数据可以得知，当转移代价越大时，压缩效率越高。当代价比例大于 1 时，压缩算法有效。

在小规模时，压缩效率提升巨大，大规模时，压缩效率有提升，但是较慢。

但是字典压缩编码还需要存储一个相对较大的字典，由第一个表格可知，字典实际上时比较庞大的，所以仍然需要耗费相当一部分空间。将压缩编码与字典空间相加后，不一定相较于 bit 存储会有提升。（或许对压缩编码再次进行多次字典压缩，会有一定提升）

相较于连续位压缩，是一个相对较好的压缩算法，但是也不是完美的压缩方案。

五、经验总结

经过本次实验，感受到了动态规划法对于整体效率有着极大的影响。本次实验的动态规划方程相对简单，更多的工作在于对于路径的压缩工作。但是尝试多种压缩算法后，发现结果并不是十分显著，可能还是直接通过 bit 存储会比较节约空间。

还有就是最大规模的测试，有时算法的不足可以利用硬件去弥补，例如增加内存大小，可以使程序存储更大量的数据集。

指导教师批阅意见：   成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。