CSCI-570 2022年秋季
算法分析的最终项目

到期：2022年12月8日

I. 准则

该项目与第8周讲座中讨论的*序列排列*问题的两种不同解决方案的实施有关。你可以在不超过3人的小组内进行这个项目。

II. 项目描述

实现*序列排列*的基本动态编程解决方案
问题。运行所提供的测试集并展示你的结果。

A. 算法描述

假设我们得到两个字符串X和Y，其中X由符号序列$x_1, x_2, \ldots x_m$组成，Y由符号序列$y_1, y_2, \ldots y_n$组成。X由符号$x_1, x_2, \ldots x_m$的序列组成。Y由符号$y_1, y_2, \ldots y_n$的序列组成。

考虑集合$\{1, 2, \ldots, m\}$和$\{1, 2, \ldots n\}$代表字符串X和Y中的不同位置，并考虑这些集合的匹配；回顾一下，匹配是一个有序对的集合，其特性是每个项目最多出现在一对中。我们说这两个集合的匹配

M是一个排列组合，如果没有 "交叉 "对：如果(i, j)，$(i', j') \in M$并且$i < i'$，那么$j < j'$

。直观地说，对齐方式提供了一种将两个字符串排成一列的方法，告诉我们哪几对位置将相互排成一列。

我们对相似性的定义将基于根据以下标准找到X和Y之间的最佳排列。假设M是X和Y之间的给定排列。

1. 首先，有一个参数 $\delta_e > 0$ 定义了一个差距惩罚。对于每个X或Y的位置在M中不匹配--这是一个缺口--我们产生δ的成本。

2. 第二，对于我们的字母表中的每一对字母p, q，都有一个错位成本α_{pq} p与q排成一列。因此，对于每一个（i, j）ϵM，我们都要支付适当的错配成本$\alpha_{x_i y_j}$用于排队x_i y_j.一般来说，一个假设$\alpha_{pp} = 0$，每个字母p都没有错位成本，可以排队。一封信的另一份副本--尽管这在接下来的事情中没有必要。
3. M的成本是其差距和不匹配成本的总和，我们寻求最小成本的对齐。

B. 输入字符串生成器

程序的输入将是一个包含以下信息的文本文件。

1. 第一基数字符串(s_1)
2. 接下来的j行由索引组成，在这些索引之后，需要将前一个字符串的副本插入累积的字符串中。(例如下面给出的)
3. 二进制字符串(s_2)
4. 接下来的k行由索引组成，在这些索引之后，需要将前一个字符串的副本插入累积的字符串中。(例如下面给出的)

这些信息将有助于从最初的2个基础字符串生成2个字符串。这个文件可以作为你程序的输入，你的程序可以使用基础字符串和规则来生成实际的字符串。还要注意，数字j和k分别对应于第一个和第二个字符串。使确保你验证第一个和第二个字符串的长度为j。$2 * len_1(s)$和$2k*len(s_2)$。请注意，基数串不一定非得是相等的。长度，同样地，j不需要等于k。

```
ACTG
3
6
1
TACG
1
```

2
9

使用上述数字，生成的字符串将是ACACTGACTGACTGGTGACTGACTGG和TATTATACGCTATTATACGCGACGCGGACGCG。

以下是关于如何生成上述字符串的逐步过程。

ACTG

ACTGACTG

辽宁沈阳："我想说的是，我们都有一个共同点，那就是我们都有一个共同点，那就是我们都有一个共同点，那就是我们都有一个共同点，那就是我们都有一个共同点。

TACG

TATACGCG

tattatacgcgacgcg

tattatacgctatacgcgacgcgacgcgacgcgcgcg

C. Delta和Alphas的值

α的值如下。δ等于30。

e

	A	C	G	T
A	0	110	48	94
C	110	0	118	48
G	48	118	0	110
T	94	48	110	0

D. 编程/脚本语言

以下是可以使用的语言列表。

1. C
2. C++
3. 爪哇
4. 蟒蛇
5. Python3

E. 边界

1. 基本算法

$0 <= j, k <= 10$

$1 <= len(s_1), len(s_2) <= 2000$

$1 <= 2^{j} * len(s_1)^{k} * len(s_2) <= 2000$

2. 内存效率高的算法

$0 <= j, k <= 20$

$1 <= len(s_1), len(s_2) <= 20000$

$1 <= 2^{j} * len(s_1)^{k} * len(s_2) <= 20000$

III. 目标

以下是你的项目要实现的目标

A. 你的程序应该接受2个参数

1. 输入文件路径
2. 输出文件路径（如果路径有效且未找到文件，你的程序应创建它）

```
`python2 basic_2.py input.txt output.txt`。
java Basic input.txt output.txt。
python3 basic_3.py input.txt output.txt。
```

注意：如第一部分B所述，输入文件将有数据来生成字符串。由于差距惩罚（δ）和不匹配惩罚（α_{pq}）是**固定的**，你必须**硬编码**他们在你的程序中。

你**不**允许使用任何图书馆。

B. 实现动态编程算法。你的程序应该在输出文件的相应行中打印以下信息。

1. 对齐的成本（整数）。
2. 第一个字符串对齐（由A、C、T、G、_（间隙）字符组成）。
3. 第二个字符串对齐（由A、C、T、G、_（间隙）字符组成）。
4. 时间，以毫秒为单位（浮点数）。
5. 以千字节为单位的内存（浮点数）

注意：可以有多条成本相同的排列组合。你可以打印任何由你的程序生成的排列组合。唯一的条件是它应该有一个最小的成本。
例如，对于字符串s_1：A和s_2：C，对齐方式A_，_C和_A，C_都有对齐的成本是60，这是最低的。你可以打印其中的任何一个。

C. 实现该解决方案的内存效率版本，并重复B部分的测试。

D. 用折线图画出B部分和C部分的结果
请使用*'datapoints'*文件夹中提供的输入文件来生成数据点以绘制图表。

1. 两种解决方案的*CPU时间与问题大小的*单图。
2. 两种解决方案的*内存使用量与问题大小的*单图。单位。CPU时间-毫秒，内存（KB），问题大小m+n

IV. 提交

A. 你应该提交包含以下文件的ZIP文件。

a. 基本算法文件
程序文件的名称应该是 '*basic.c*' / '*basic.cpp*' / '*Basic.java*' / '*basic_2.py*' (Python 2.7) / '*basic_3.py*' (Python 3) 。

b. 内存效率高的算法文件
程序文件的名称应该是*'efficient.c' / 'efficient.cpp' / 'Efficient.java' / 'efficient_2.py'* (Python 2.7) / '*efficient_3.py*' (Python 3) 。

c. 摘要.pdf
它必须包含以下细节

1. 数据点输出表（由提供的输入文件生成）。
2. 两张图和洞察力
3. 每个小组成员的贡献，如编码、测试、报告准备等，如果每个人的贡献不一样的话。

(请使用提供的Summary.docx文件，填写详细信息并以PDF格式上传)

d. 2 外壳文件 "*basic.sh* "和 "*efficient.sh* "中包含编译和运行基本版和高效版的命令。需要这些文件来为你提供灵活性，以传递你的程序可能需要的任何附加编译器/运行参数。参见更多提示（更多细节见第七部分E）。

basic.sh

```
javac Basic.java
java Basic "1" "2"
```

执行：./basic.sh input.txt output.txt

./efficient.sh input.txt output.txt

B. 你的压缩文件的名称应该有你的小组中每个人的USC ID（不是电子邮件ID），用下划线分开。

- 1234567890_1234567890_1234567890.zip
  - 1234567890_1234567890_1234567890
    - basic_2.py
    - efficient_2.py
    - 摘要.pdf
    - basic.sh
    - 高效.sh

V. 分级

请阅读以下说明，以了解如何评估您的投稿。

A. 算法的正确性 - **70**分

1. 两个程序（基本程序/高效程序）都能正确地输出文件，且所有5行的顺序都是正确的。**15**分
2. 基本算法。**25**分
3. 内存效率高的算法。**30**分

注意：评分员将在`Linux`操作系统上执行你的程序。A部分的目标是检查正确性。该程序应该以最小的代价输出有效的排列。内存和时间将在B部分进行评估。

B. 绘图，分析结果，洞察力和观察力。**30**分

1. 你的程序将在输入文件（由我们在 "数据点 "文件夹中提供）上运行，生成输出文件。输出文件中的内存和时间应该与你在Summary.pdf中提供的内容 "*接近*"。
2. 图形的正确性
3. 分析的正确性/见解的正确性

注意：与A部分不同，B部分的评估是主观的，所以它将被手动完成。因此，如果你的图表/数据点有 "*一些* "离群值，那也没关系。

VI. 压缩文件中向你提供了什么？

A. 包含样本输入和输出文件的SampleTestCases文件夹

B. 包含输入文件的数据点文件夹，用于生成图形数据点。

C. 摘要.docx文件供参考

VII. 提示、注意事项和常见问题

A. 关于输入和字符串的生成

1. 我们永远不会给你的程序提供一个无效的输入。输入字符串将始终只包含A、C、G、T。
2. 最终输入字符串的长度应等于 $2^{number\ of\ times}(base\ string)$，如文件中提到。
3. 无论算法的基本版本还是高效版本，字符串的生成机制都是一样的。
4. 整个程序（字符串生成、解决方案、写输出）应该写在一个文件中。你可以把这些功能分解到不同的类中，使代码模块化，但应该只有一个文件。你的程序不会根据它的模块化程度而被评估。

B. 关于算法和输出

1. 请不要参考Kleinberg和Tardos中提供的伪代码。
2. 不要使用任何库来编写你的算法。
3. 我们提供了时间和内存计算的样本。为了保持一致性，请使用它们。
4. 你对常规算法和内存效率算法的解决方案应该在两个不同的程序中。
5. 可以有多个具有相同最佳成本的有效序列，你可以输出其中任何一个。所有这些都是有效的。
6. 你应该同时编码基本版本和内存效率算法。即使内存效率版本会通过简单版本的所有界限，你也不能在两个都使用内存效率版本。
   子问题。
7. 你的程序在运行时不应该打印任何东西。它应该只写到输出文件中。
8. 对时间和内存浮动值的精度没有具体要求。
9. 时间和内存取决于许多因素，如CPU、操作系统等。因此，在输出方面可能会有差异。因此，它将被主观地评估。在时间/内存复杂度为O(n)与$O(n^2)$与O(logn)的程序之间，必须有明确的行为区分。

C. 关于情节

1. 这两张图都是线形图。X轴表示问题大小为m+n，其中m和n是生成字符串的长度，内存图的Y轴表示内存，单位为KB，时间图的Y轴表示时间，单位为毫秒。图中的两条线将代表基本算法和内存效率算法的统计。
2. 你可以使用任何语言的任何库/包来绘制图表。
3. 你不需要提供生成图表的代码。只需在Summary.pdf中添加图片

D. 关于提交

1. 小组中只有1个人需要提交项目。我们会从文件名中得到所有其他团队成员的USC ID。
2. 为了能在合理的时间内给全班同学打分，如果你的程序在一个输入文件上运行超过一分钟，我们就会杀掉它。

E. 关于贝壳文件

为了使我们的评估工作顺利进行，请确保你也有一个名为 "*basic.sh* "和 "*efficient.sh* "的shell脚本，其中包含运行你的程序所需的命令。例如，这个文件的内容可以是以下之一。

C

basic.sh	`gcc basic.c` `./a.out "1" "2"`
高效.sh	`gcc efficient.c` `./a.out "1" "2"`

C++

basic.sh	`g++ basic.cpp` `./a.out "1" "2"`
高效.sh	`g++ efficient.cpp` `./a.out "1" "2"`

爪哇

basic.sh	`javac Basic.java` `java Basic "1" "2"`
高效.sh	`javac Efficient.java` `java Efficient "1" "2"`

python 2.7

basic.sh	`python2 basic_2.py "1" "2"`
高效.sh	`python2 efficient_2.py "1" "2"`

蟒蛇3

basic.sh	`python3 basic_3.py "1" "2"`
高效.sh	`python3 efficient_3.py "1" "2"`

请注意，上述内容只是例子。你可以根据你的方便来修改它们。我们的目标是要有一个独立于语言的机制来运行你的脚本以获得你的输出。

还要注意的是，对于python2或python3的用户来说，在最后有2或3个后缀是很重要的。

F. 内存和时间计算的示例代码 Python

```
输入sys
从资源中导入*导入时间
输入psutil
```

```
def process_memory():
    process = psutil.Process()
    memory_info = process.memory_info()
    memory_consumed = int(memory_info.rss/1024)
    return memory_consumed
```

```
def time_wrapper() :
    start_time =
    time.time()
    call_algorithm()
    end_time = time.time()
    time_taken = (end_time - start_time)*1000
    return time_taken
```

爪哇

```
私有的静态双数getMemoryInKB() {
   double total = Runtime.getRuntime().totalMemory();
   return (total-Runtime.getRuntime().freeMemory())
   /10e3;
}
```

```
private static double getTimeInMilliseconds() {
   return System.nanoTime()/10e6;
}
```

```
double beforeUsedMem=getMemoryInKB();
double startTime = getTimeInMilliseconds();

alignment = basicSolution(firstString, secondString, delta, alpha);

double afterUsedMem = getMemoryInKB();
double endTime = getTimeInMilliseconds()。

double totalUsage = afterUsedMem-beforeUsedMem;
double totalTime = endTime - startTime;
```

C/C++

```
#include <sys/resource.h>
#include <errno.h>
#include <stdio.h>

外来int errno。

// getrusage()在linux中可用。你的代码将在Linux操作系统中被评估。
long getTotalMemory() {
   struct rusage usage;
   int returnCode = getrusage(RUSAGE_SELF, &usage);
   if(returnCode == 0) {
       返回usage.ru_maxrss。
   } else {
       //它不应该出现。请查看 man getrusage 以了解更多信息
调试。
       // printf("error %d", errno);
       return -1;
   }
}
```

```
int main() {

    struct timeval begin, end;
    gettimeofday(&begin, 0);
    //在这里写下你的解决方案
```

```
    //请在调用你的解决方案函数后才调用getTotalMemory()。它可以计算出程序使用的最大内存。
    double totalmemory = getTotalMemory();
    gettimeofday(&end, 0);
    long seconds = end.tv_sec - begin.tv_sec。
    long microseconds = end.tv_usec - begin.tv_usec;
    double totaltime = seconds*1000 + microseconds*1e-3;
    printf("%f\n", totaltime);
    printf("%f\n", totalmemory)。
}
```