近似串匹配实验报告

杜金鸿,15338039

2017 年 4 月 16 日

目录

1 实验目的

1.1 问题描述

设有两个字符串，样本 $P=p_1p_2\cdots p_m$, 文本 $T=t_1t_2\cdots t_n$。假设样本是正确的，样本 P 在文本 T 中的 $K-$ 近似匹配 $(K-approximate\ Match)$ 是指 P 在 T 中包含至多 K 个差别的匹配。其中差别有如下三种类型：

1. 修改　$T:abc \to P:adc$
2. 删去　$T:abc \to P:ac$
3. 插入　$T:abc \to P:adbc$

1.2 实验要求

1. 设计算法判断样本 P 是否是文本 T 的 $K-$ 近似匹配；
2. 设计程序实现你设计的算法，并设计有代表性的测试数据；
3. 分析算法的时间复杂度。

2 实验内容

2.1 设计思想

不同对应方式计算出的错误操作数可能不同, 因此我们应明确 $K-$ 近似匹配的含义：

1. 样本 P 与文本 T 的差别数 k 至多为 K;
2. 差别数 k 是指样本 P 与文本 T 在所有匹配对应方式下的最小编辑错误总数。

寻找 $K-$ 近似匹配，即寻找文本 T 的某一个（或多个）子串，它（们）与样本 P 的差别数不超过 K。

2.2 测试数据

一般选取英语句子中某个单词加以修改，或对简单的字符组合进行修改、删去、插入操作来作为测试数据。

$$
\begin{array}{ll}
P: happy & T: Have\ a\ hsppy\ day! \\
P: ab & T: aubua \\
P: word & T: You\ keep\ your\ wword\ in\ the\ world. \\
\vdots &
\end{array}
$$

2.3 近似串匹配算法

近似串匹配算法 ($Approximate\ String\ Matching, ASM$) 利用动态规划实现。首先定义代价函数 $D(i,j)(0 \leqslant i \leqslant m, 0 \leqslant j \leqslant n)$ 表示样本 P 的前缀子串 $p_1 p_2 \cdots p_i$ 与文本 T 前缀子串 $t_1 t_2 \cdots t_j$ 的差别数 k，i 或 j 为 0 时表示对应子串为空串。

代价函数的初始值如下:

1. $D(0,j)=0$　　表示样本空子串与文本子串 $t_1 t_2 \cdots t_j$ 差别数为 0;
2. $D(i,0)=i$　　表示样本子串 $p_1 p_2 \cdots p_i$ 与文本空子串差别数为 i.

在 $i-1, j-1$ $(i,j>0)$ 时的状态已经确定的情况下, 下一状态之间的关系如下:

1. 若 t_j 与 p_i 相对应且 $p_i = t_j$, 则 $D(i,j) = D(i-1,j-1)$;
2. 若 t_j 与 p_i 相对应且 $p_i \neq t_j$, 则 $D(i,j) = D(i-1,j-1)+1$;
3. 若 t_j 与 t_{j-1} 之间有缺失, 则 $D(i,j) = D(i-1,j)+1$;
4. 若 t_j 多余, 则 $D(i,j) = D(i,j-1)+1$.

在这里，我们将样本 P 作为原字符串，删去、插入、修改的操作均为对 P 进行。

因此我们可以得到如下递推式:

$$D(i,j) = \begin{cases} 0, & i=0 \\ i, & j=0 \\ \min\{D(i-1,j-1), D(i-1,j)+1, D(i,j-1)+1\}, & i>0, j>0, p_i=t_j \\ \min\{D(i-1,j-1)+1, D(i-1,j)+1, D(i,j-1)+1\}, & i>0, j>0, p_i \neq t_j \end{cases}$$

3 设计与编码

3.1 第一近似串匹配算法

1. 符号假设如下:
   - P: 样本字符串
   - T: 文本字符串
   - m: P 的长度
   - n: T 的长度
   - D: 代价矩阵（$(m+1) \times (n+1)$）
2. 算法流程

Algorithm 1 第一近似串匹配算法 *ASM*

$D(0,j)=0 \qquad (j=1,2,\cdots,m);$

$D(i,0)=i \qquad (i=1,2,\cdots,n);$

$k=l=1;$

for $l<=n$ do

  for $k<=m$ do

    if $P(k)==T(l)$ then

      $D(k,l)=\min\{D(k-1,l-1),D(k-1,l)+1,D(k,l-1)+1\};$

    else

      $D(k,l)=\min\{D(k-1,l-1)+1,D(k-1,l)+1,D(k,l-1)+1\};$

    end if

  end for

  if $D(m,l)<=K$ then

    return l;

  end if

end for

3. 代码示例

```
int ASM(char P[],char T[],int m, int n,int K){
    int D[m+1][n+1];
    // 代价矩阵初始化
    for (int j=0; j<=n; j++) {
        D[0][j]=0;
    }
    for (int j=0; j<=m; j++) {
        D[j][0]=j;
    }
    // 递推计算
    for (int j=1; j<=n; j++) {
        for (int i=1; i<=m; i++) {
            if(P[i-1]==T[j-1]){
                D[i][j]=min(D[i-1][j-1],D[i-1][j]+1);
                D[i][j]=min(D[i][j],D[i][j-1]+1);
            }
            else{
                D[i][j]=min(D[i-1][j-1]+1,D[i-1][j]+1);
                D[i][j]=min(D[i][j],D[i][j-1]+1);
            }
        }
        // 找到第一个即返回
        if(D[m][j]<=K)
            return j;
    }
    // 查找失败
    return -1;
}
```

3.2 所有近似串匹配算法

1. 符号假设如下：

   - P：样本字符串
   - T：文本字符串
   - m：P 的长度
   - n：T 的长度
   - D：代价矩阵（$(m+1)\times(n+1)$）

2. 算法流程

Algorithm 2 所有近似串匹配算法 *ASM*

$D(0,j)=0 \qquad (j=1,2,\cdots,m);$

$D(i,0)=i \qquad (i=1,2,\cdots,n);$

$k=l=1;$

for $l<=n$ do

  for $k<=m$ do

    if $P(k)==T(l)$ then

      $D(k,l)=\min\{D(k-1,l-1),D(k-1,l)+1,D(k,l-1)+1\};$

    else

      $D(k,l)=\min\{D(k-1,l-1)+1,D(k-1,l)+1,D(k,l-1)+1\};$

    end if

  end for

end for

print D;

---

3. 代码示例

```
void ASM(char P[],char T[],int m, int n){
    int D[m+1][n+1];
    // 代价矩阵初始化
    for (int j=0; j<=n; j++) {
        D[0][j]=0;
    }
    for (int j=0; j<=m; j++) {
        D[j][0]=j;
    }
    // 递推计算
    for (int j=1; j<=n; j++) {
        for (int i=1; i<=m; i++) {
            if(P[i-1]==T[j-1]){
                D[i][j]=min(D[i-1][j-1],D[i-1][j]+1);
                D[i][j]=min(D[i][j],D[i][j-1]+1);
            }
```

```
            else{
                D[i][j]=min(D[i-1][j-1]+1,D[i-1][j]+1);
                D[i][j]=min(D[i][j],D[i][j-1]+1);
            }
        }
    }
    // 输出代价矩阵
    for (int i=0; i<=m; i++) {
        if (i>0) {
            cout<<P[i-1]<<'\t';
        }
        else
            cout<<'\t';
        for (int j=0; j<=n; j++){
            cout<<D[i][j]<<'\t';

        }
        cout<<endl;
    }
}
```

4 运行与测试

4.1 运行结果

以所有近似串匹配算法为例，主程序如下：

```
#include <iostream>
using namespace std;
void ASM(char P[],char T[],int m, int n);

int main(int argc, const char * argv[]) {
    char T[30],P[30];
    cout<<"Please input P(<30 chars):"<<endl;
    cin.getline(P,30,'\n');
    cout<<"Please input T(<30 chars):"<<endl;
    cin.getline(T,30,'\n');
    int m,n;
    cout<<"Please input m:"<<endl;
    cin>>m;
    cout<<"Please input n:"<<endl;
    cin>>n;
    cout<<"The cost matrix is:"<<endl;
    cout<<'\t'<<'\t';
    for (int i=0; i<n; i++) {
        cout<<T[i]<<'\t';
    }
    ASM(P,T,m,n);
    return 0;
}
```

输出结果:

```
Please input P(<30 chars):
happy
Please input T(<30 chars):
Have a hsppy day.
Please input m:
5
Please input n:
17
The cost matrix is:
        H   a   v   e       a       h   s   p   p   y       d   a   y   .
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
h   1   1   1   1   1   1   1   1   0   1   1   1   1   1   1   1   1   1
a   2   2   1   2   2   2   1   2   1   1   2   2   2   2   2   1   2   2
p   3   3   2   2   3   3   2   2   2   2   1   2   3   3   3   2   2   3
p   4   4   3   3   3   4   3   3   3   3   2   1   2   3   4   3   3   3
y   5   5   4   4   4   4   4   4   4   4   3   2   1   2   3   4   3   4
Program ended with exit code: 0
```

可以看到，输出的代价矩阵最后一行的元素 $D(m,j)$ 即代表了 T 以 $T(j)$ 元素结尾的子串的最小近似匹配。例如:

$D(m,12)=1$, 匹配如下

$$hsppy \quad \longleftrightarrow \quad happy$$

即：将 s 修改为 a;

$D(m,13)=2$ 匹配如下:

$$hsppy\_ \quad \longleftrightarrow \quad happy$$

即：将 s 修改为 a, 将空格 _ 删去;

$D(m,14)=3$ 匹配如下:

$$hsppy\_d \quad \longleftrightarrow \quad happy$$

即：将 s 修改为 a, 将空格 _ 删去，将 d 删去……

4.2 算法分析

考虑算法复杂度，在给定问题规模的情况下，即已知样本 P 的长度 m 和文本 T 的长度 n, 由于算法基本操作数为

$$n+m+n\times m$$

因此算法时间复杂度为 $O(n\times m)$。

若采用暴力搜索近似匹配，不同的选择方法总数为样本长度 m 的指数函数，计算量巨大。因此采用动态规划的近似串匹配算法是有巨大改进的。

4.3 算法的改进

1. 显式的对应关系

   $D(m,j)$ 仅代表 T 以 $T(j)$ 元素结尾的子串的最小近似匹配，但我们并不知道该子串的长度，也不知道对应的操作。

   可以通过设定一个矩阵 $B(m,n)$ 来存储操作结果:

   当两字符对应且相等记录为 $B(i-1,j-1)=$'_';

当不等且 $D(i,j)=D(i-1,j-1)+1$ 时，记录为 $B(i-1,j-1)=$’r’(revise);

当 $D(i,j)=D(i-1,j)$ 时，记录为 $B(i-1,j-1)=$’d’(delete);

当 $D(i,j)=D(i,j-1)$ 时，记录为 $B(i-1,j-1)=$’i’(insert)。

以上述测试数据为例，得到 B 矩阵如下（注意 D 是 $(m+1)\times(n+1)$ 维的，而 B 是 $m\times n$ 维的。）

	H	a	v	e		a		h	s	p	p	y		d	a	y
h	d	d	d	d	d	d	d	-	d	d	d	d	d	d	d	d
a	d	-	d	d	d	-	d	r	i	d	d	d	d	d	-	d
p	d	r	i	d	d	r	i	r	d	-	-	d	d	d	r	i
p	d	r	d	i	d	r	d	d	d	-	-	r	r	d	r	d
y	d	r	d	d	i	r	d	d	d	r	r	-	r	r	r	-

易见，$hsppy$ 与 $happy$ 的对应关系即为’_i_____’。当然，这种方法仍需判断 $P(i)$ 与 $T(j)$ 的关系以及 $D(i,j)$ 的取值情况，并且对应关系不唯一。

一方面，通过递归回溯可以实现对应关系的显示表达，但其计算复杂度随 m 呈指数上升。

另一方面，若在近似匹配过程中，增加两个矩阵 $X(m,n),Y(m,n)$ 来记录 $D(i,j)$ 取值与 $D(i-1,j-1),D(i-1,j),D(i,j-1)(i,j>1)$ 的关系，即

(a) 若 $P(i)=T(j)$, 且 $D(i-1,j-1)=\min\{D(i-1,j-1),D(i-1,j)+1,D(i,j-1)+1\}$, 则

$$\begin{cases} X(i-1,j-1)=i-2 \\ Y(i-1,j-1)=j-2 \end{cases}$$

(b) 若 $P(i)=T(j)$, 且 $D(i-1,j)+1=\min\{D(i-1,j-1),D(i-1,j)+1,D(i,j-1)+1\}$, 则

$$\begin{cases} X(i-1,j-1)=i-2 \\ Y(i-1,j-1)=j-1 \end{cases}$$

(c) 若 $P(i)=T(j)$, 且 $D(i,j-1)+1=\min\{D(i-1,j-1),D(i-1,j)+1,D(i,j-1)+1\}$, 则

$$\begin{cases} X(i-1,j-1)=i-1 \\ Y(i-1,j-1)=j-2 \end{cases}$$

(d) 若 $P(i)\neq T(j)$, 且 $D(i-1,j-1)+1=\min\{D(i-1,j-1),D(i-1,j)+1,D(i,j-1)+1\}$, 则

$$\begin{cases} X(i-1,j-1)=i-2 \\ Y(i-1,j-1)=j-2 \end{cases}$$

(e) 若 $P(i)\neq T(j)$, 且 $D(i-1,j)+1=\min\{D(i-1,j-1),D(i-1,j)+1,D(i,j-1)+1\}$, 则

$$\begin{cases} X(i-1,j-1)=i-2 \\ Y(i-1,j-1)=j-1 \end{cases}$$

(f) 若 $P(i)\neq T(j)$, 且 $D(i,j-1)+1=\min\{D(i-1,j-1),D(i-1,j)+1,D(i,j-1)+1\}$, 则

$$\begin{cases} X(i-1,j-1)=i-1 \\ Y(i-1,j-1)=j-2 \end{cases}$$

事实上 X 与 Y 的作用是记录该位置的最小代价前驱，函数代码如下：

```
void ASM(char P[],char T[],int m, int n){
    int D[m+1][n+1];
    char B[m][n];
    int X[m][n];
    int Y[m][n];
    for (int j=0; j<=n; j++) {
        D[0][j]=0;
    }
    for (int j=0; j<=m; j++) {
        D[j][0]=j;
    }
    for (int j=1; j<=n; j++) {
        for (int i=1; i<=m; i++) {
            if(P[i-1]==T[j-1]){
                D[i][j]=min(D[i-1][j-1],D[i-1][j]+1);
                D[i][j]=min(D[i][j],D[i][j-1]+1);
                if (D[i][j]==D[i-1][j-1]) {
                    B[i-1][j-1]='_';
                    X[i-1][j-1]=i-2;
                    Y[i-1][j-1]=j-2;
                }
                else if (D[i][j]==D[i-1][j]+1) {
                    B[i-1][j-1]='d';
                    X[i-1][j-1]=i-2;
                    Y[i-1][j-1]=j-1;
                }
                else{
                    B[i-1][j-1]='i';
                    X[i-1][j-1]=i-1;
                    Y[i-1][j-1]=j-2;
                }
            }
            else{
                D[i][j]=min(D[i-1][j-1]+1,D[i-1][j]+1);
                D[i][j]=min(D[i][j],D[i][j-1]+1);
                if (D[i][j]==D[i-1][j-1]+1) {
                    B[i-1][j-1]='r';
                    X[i-1][j-1]=i-2;
                    Y[i-1][j-1]=j-2;
                }
                else if (D[i][j]==D[i-1][j]+1){
                    B[i-1][j-1]='d';
                    X[i-1][j-1]=i-2;
                    Y[i-1][j-1]=j-1;
                }
                else{
                    B[i-1][j-1]='i';
                    X[i-1][j-1]=i-1;
                    Y[i-1][j-1]=j-2;
                }
            }
        }
    }
    for (int i=0; i<=m; i++) {
```

```
        if (i>0) {
            cout<<P[i-1]<<'\t';
        }
        else
            cout<<'\t';
        for (int j=0; j<=n; j++){
            cout<<D[i][j]<<'\t';

        }
        cout<<endl;
    }
    cout<<"----------------------------------------------------------------------"<<endl;
    cout<<"Matrix␣X:"<<endl;
    for (int i=0; i<m; i++) {
        cout<<P[i]<<'\t'<<'\t';
        for (int j=0; j<n; j++){
            cout<<X[i][j]<<'\t';

        }
        cout<<endl;
    }
    cout<<"----------------------------------------------------------------------"<<endl;
    cout<<"Matrix␣Y:"<<endl;
    for (int i=0; i<m; i++) {
        cout<<P[i]<<'\t'<<'\t';
        for (int j=0; j<n; j++){
            cout<<Y[i][j]<<'\t';

        }
        cout<<endl;
    }
    cout<<"----------------------------------------------------------------------"<<endl;
    cout<<"Matrix␣B:"<<endl;
    for (int i=0; i<m; i++) {
            cout<<P[i]<<'\t'<<'\t';
        for (int j=0; j<n; j++){
            cout<<B[i][j]<<'\t';

        }
        cout<<endl;
    }
    cout<<"----------------------------------------------------------------------"<<endl;
    for (int k=0; k<n; k++) {
        char c[m+n];
        int l=0;
        int i2,j2;
        for (int i=m-1,j=k;(i>=0 && j>=0);) {
            c[l]=B[i][j];
            l++;
            i2=X[i][j];
            j2=Y[i][j];
            i=i2;j=j2;

        }
        //若子串长度不足$m$，则前面部分的对应关系必为删除delete
```

```
        for (int i=l; i<m; i++) {
            cout<<'d';
        }
        // 逆序输出对应关系
        for (int i=l-1; i>=0; i--) {
            cout<<c[i];
        }
        cout<<endl;
    }
}
```

输出结果如下：

```
          H   a   v   e       a       h   s   p   p   y       d   a   y
     0    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
h    1    1   1   1   1   1   1   1   0   1   1   1   1   1   1   1   1
a    2    2   1   2   2   2   1   2   1   1   2   2   2   2   2   1   2
p    3    3   2   2   3   3   2   2   2   2   1   2   3   3   3   2   2
p    4    4   3   3   3   4   3   3   3   3   2   1   2   3   4   3   3
y    5    5   4   4   4   4   4   4   4   4   3   2   1   2   3   4   3
------------------------------------------------------------------------
Matrix X:
h        -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
a         0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
p         1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
p         2   2   2   2   2   2   2   2   2   2   2   3   3   2   2   2
y         3   3   3   3   3   3   3   3   3   3   3   3   4   4   3   3
------------------------------------------------------------------------
Matrix Y:
h        -1   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
a        -1   0   1   2   3   4   5   7   7   8   9  10  11  12  13  14
p        -1   1   1   2   3   5   5   7   7   8   9  10  11  12  14  14
p        -1   1   1   2   3   5   5   6   7   8   9  10  11  12  14  14
y        -1   1   1   2   3   5   5   6   7   9  10  10  11  12  14  14
------------------------------------------------------------------------
Matrix B:
h         r   r   r   r   r   r   r   _   r   r   r   r   r   r   r   r
a         r   _   r   r   r   _   r   d   r   r   r   r   r   r   _   r
p         r   d   r   r   r   d   r   d   r   _   _   r   r   r   d   r
p         r   d   r   r   r   d   r   r   r   _   _   i   i   r   d   r
y         r   d   r   r   r   d   r   r   r   d   d   _   i   i   d   _
------------------------------------------------------------------------
ddddr
r_ddd
r_ddr
r_drr
r_rrr
r_ddd
r_ddr
r_drr
r_rrr
_dr_d
_r__d
_r___
_r___i
_r___ii
r_ddd
r_dd_
Program ended with exit code: 0
```

如图，以任意一个字符 $T(j)$ 结束的最近似匹配子串的长度和对应关系都可以确定下来。例如，由 _r___i

可知，以 $T(12) = \_$ 结尾的近似匹配子串长度为 6，即为 $hsppy\_$, 对应的操作是将 $happy$ 的第二位进行修改操作、最后一位之后进行插入操作。

2. 多文本最小近似配对
只需遍历每个文本，计算出其与样本的代价矩阵，计算出其中最小的 $K-$ 近似匹配值，找出该值所对应的文本即可。

5 总结

在近似串匹配算法中，若直接采用穷举法，由于允许有三种不同操作，两个字符串的对应关系有很多选择，并且文本的子串长度也有多种选择，计算量极大。但是我们采用了动态规划方法，计算复杂度仅有 $O(m \times n)$，极大地减少了算法复杂度，十分便捷。