**[doop-ymc](http://yaowhat.com/" \o "doop-ymc)**

**网页版本对比系列二 —— Myers' Diff Algorithm 1**

2014-07-21 [#技术](http://yaowhat.com/tags.html#技术) [#算法](http://yaowhat.com/tags.html#算法)

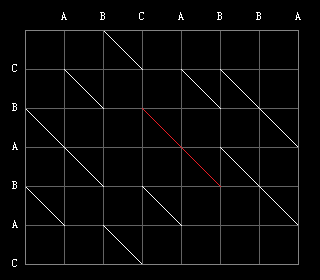
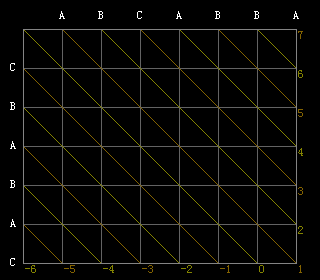
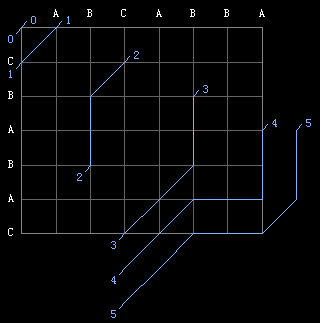
本节将详细的对 [Myer's diff algorithm](https://neil.fraser.name/software/diff_match_patch/myers.pdf) 的算法思想及过程进行详细的介绍。

**介绍**

Myer's diff algorithm 首次出是在1986年一篇论文中"An O(ND) Difference Algorithm and Its Variations", 在文中实现上介绍了两种此diff算法 的实现。两种实现的核心思想是一致的，只是在具体的实现过程中，为进一步提升算法的性能及空间利用率，采取了不一致的迭代方式。下面先介绍，论文中 描述的基础算法。

在进行具体的算法描述前，先给出一些文中用到的定义及前提。

**定义**

1. 最短编辑距离(SES)   
   最短编辑距离是指，在仅可进行插入，删除两种操作的情况下，将串A转换为串B所需要的操作次数。
2. 最长公共子序列(LCS) 最长公共子序列是指，对于两个串，在保持顺序不变的前提下，能够得到的两串间的最大长度。与最长公共子 串的不同处在于，最长公共子序列不要求是连续的。
3. 编辑图(Edit graph) 编辑图实现是由两个串A,B分别为x, y坐标构成一个矩阵，不过对于其中相等的点处会多画出一条斜边。  
   如对于串CBABAC, ABCABBA, 其编辑图可表示为下图:[](http://yaowhat.com/2014/07/21/entry-version-diff-1.html)
4. Snakes 对于Snakes的理解，存在一些歧义，我的理解是一条连续的由斜边组成的路径。如上图的红色路径示即为一条snake。
5. k斜线(k lines) k斜线是指对于由x-y具有相同的值的点组成一条直线，若k=x-y, 则称由这些点组成的线为k斜线。如下图示，为 编辑图上的k lines。[](http://yaowhat.com/2014/07/21/entry-version-diff-1.html)
6. d等距线(d contours)  
   d contours 是指到起始点处，具有相同的编辑距离的点构成的线。如下图示：[](http://yaowhat.com/2014/07/21/entry-version-diff-1.html)
7. d-path d path 是指从起始点开始, 假设为(0,0), 向前延伸的路径中包含d条非斜边的这样的一条路径。

8 d-path的递归定义 一条d-path必定由一条(d-1)-path接一条水平或者垂直边，再加上一条snake构成。

**算法描述**

基础算法是建立在以下两条定理的基础上的:

* 定理一：任何一条D-path都必定会终止于这些k lines, k ∈ { − D, − D + 2, . . . D − 2, D }.
* 定理二: 一条最远的在斜边k上结束的D-path，必定会由一条在k-1上结束的到达最远的(D-1)-path与一条水平边， 再与一条能够到达的最远的snake构成，或者由一条在k+1上结束的到达(D-1)-path与一条垂直边，再与一条能够 到达的最远的snake构成。

对于定理一证明过程如下：

1. 当D=0时， 0-path就是对角线，满足条件

2. 设i-path，满足定理， 则i-path的终点必定在这些k-lines 上， k ∈ { − Di − Di+ 2, . . . Di− 2, Di }

则根据D-path的递归定义，(i+1)-path必定是由i-path后面跟一条垂直边或者一条水平边，再加一snake构成，这样一

条垂直边 会使(i+1)-path结束终止点的k-lines范围在{-i-1, i}, 一条水平边则将范围更改至{-i-1, i+1}, 而snake

是不会 更改k-lines的范围的，因此(i+1)-path的终点所在的k-lines必定在{-i-1, i+1},定理得证。

对于定理二，实际可由d-path的递归定义推导得，这儿就不证明了。 经以个两个定理，实际，已经可以迭代的求出能够到达串的编辑图的终止在对角点的D最小的一条D-path,这条D-path实际 就描绘出了最小编辑距离，实际上也就给出两中最长公共子序列（LCS）。

下面说说具体的算法思路：

通过上面的两条定理，就可以通过迭代，不断的求出各条k lines到达最远的D-path。 这样当有一条D-path到达了

编辑图的右下角， 就找到了两串间的LCS,而最大的D可能为M+N, M, N分别为两串的长度，因为最多两串完全不一致。

而由第一条定理知，D-path 必定 会终止于-D, D内的k lines上， 因此在计算D-path到达的最远距离时，只需要计

算-D，到D的k lines所到达的最远的距离，而每次 计算D-path在k lines到达的最远距离时，是可以根据 (D-1)-path

迭代来的，为此，通过迭代计算可以最终求出能最终到达编辑图 右下角的D-path,且拥有最小的D,即具有最小的编辑

距离。

下面是简单迭代过程：

Constant MAX ∈ [0,M+N]

Var V: Array [− MAX .. MAX] of Integer

V[1] ← 0

For D ← 0 to MAX Do

For k ← −D to D in steps of 2 Do

If k = −D **or** k ≠ D **and** V[k − 1] < V[k + 1] Then

**x** ← V[k + 1]

Else

**x** ← V[k − 1]+1

**y** ← **x** − k

While **x** < N **and** **y** < M **and** ax+1 = by+1 Do (**x**,**y**) ← (**x**+1,**y**+1)

V[k] ← **x**

If **x** ≥ N **and** **y** ≥ M Then

Length of an SES is D

Stop

Length of an SES is greater than MAX

# 下面是具体的迭代结果图：[edit](http://yaowhat.com/2014/07/21/entry-version-diff-1.html)上图中，蓝色线条为具体的迭代过程，红色为最终求得的D-path.根据这条D-path实现已可求出[doop-ymc](http://yaowhat.com/" \o "doop-ymc)

**网页版本对比系列三 —— Myers' Diff Algorithm 2**

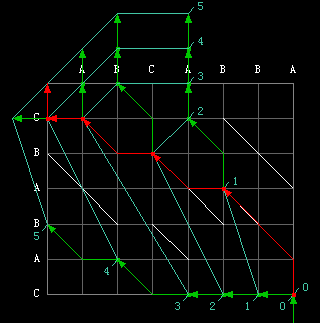
2014-07-23 [#技术](http://yaowhat.com/tags.html#技术) [#算法](http://yaowhat.com/tags.html#算法)

上节对 [Myer's diff algorithm](https://neil.fraser.name/software/diff_match_patch/myers.pdf) 的算法的基础算法的思想进行了详细的介绍，本节将着重对文中的的对基础算法的改进算法进行详细的 介绍。

**介绍**

在理解上一节的基础算法的基础上，对于本节算法的思想的理解其实较为容易。本节描述的算法是为进上步提升 求取LCS问题的时间和空间的复杂度的。

**算法描述**

上一节描述的基础算法，实际上是从编辑图的左上角开始，向右下角不断的查询能够延伸到的最远的D-path，直到到达右下角； 实际上这个过程也可以从右下角开始，直到到达左上角，这样就有如下图示的一个迭代过程：[](http://yaowhat.com/2014/07/23/entry-version-diff-2.html)根据上图的迭代过程，实际上也得到了串CBABAC与 串ABCABBA，的一条最长公共子序列（LCS),其与上一节得到的最长公共子 序列具有相同的子序列长度，却有着不一样的序列组成，上一节正向得到的LCS为CABA, 上图反向得到的串为BABA, 但两个序列 均是正确的，因为两个串完全可能存在多个均为最长的公共子序列。

在给出具体的算法前，还需要明白下面的一个定理, 接上节，这里称定理三：

如果存在一条从(0,0)到(M,N)的D-path，当且仅当存在一条从(0,0)到(x,y)的D/2-path, 及一条比(u,v)到(M,N)的

D/2-path, (x,y),(u,v)需满足以下条件：

(可行性) u+v ≥ D/2 **and** x+y ≤ N+M-D/2 **and**

(重叠性) x-y = u -v **and** x ≥ u

并且以上的两条D/2-path均需要是D-path的一部分。

以上的定理实际上是很好推导的，在上一节描述的第二条定理中实际上就有部分体现，通过第二条定理知，任一条D-path, 实际上是 可以由一条E-path, snake,及一条(D-E)-path构成(E>=0 && E <= D)。特别的，对于E=D/2时，任一条D-path，就可以由一条D/2-path， 一条snake, 及其后的另一条D/2-path构成, 假设得到的中间的snake为middle snake。  
到这儿，根据上一条定理，新算法的思想差不多就出来了，如下：

由于从正向及反向查找D-path实际上在理论上是等价的，这样的在查找M,N编辑图上的D-path，就可以转化为同时

从正向及反 向同时计算D/2-path的过程，不过这两个D/2-path需要最后同时终止于一条相同的k-line, 在这边k-lines

上会根据判定重叠条件得到一条middle snake。接下来就是将问题二分化，以middle snake为切分，将问题分解为继续

求两个D/2-path的middle snake 的过程, 最终所有得到的middle snake即最后的M,N编辑图中D-path的track。

上面的算法思想，说得比较清楚了，关键是重叠的判定 。由于正向的k-lines是以0-line为中心的，而反向的k-lines是以∆ =N-M为中心的。 另由定理1知，D-path的D奇偶性与∆ 是一致的，这样，当∆ 为奇数时，只需要在前向延伸时，进行重叠性的判定，而当∆ 为偶数时，则只需要在 反射延伸时进行重叠判定。具体的重叠判定，由于在算法中都是采用相同的v向量，则仅需要判定v向量对应的k-line是否存在值及x是否重叠即可。

看过上面的算法思想的描述，通常情况下会存在以下几个疑问:

1. 为什么还要继续以middle snake为分割，将问题分割成两个子问题继续迭代，而不是将得到的两个D/2-path, middle snake直接构成 D-path, 用以生成最终的LCS?

那是因为，最终重叠在k line上的两条snake并不一定是相等的，middle snake只是从一个方向迭代时得到的snake。如果 两条snake是不一至的,那么, 以middle snake及两端的D/2-path为基础是无法构建出最终的D-path的，上面的算法思想中 说的可以从正反两个方向分别迭代，以终止于相同的k-line,实际是为了查找出最后重叠的k line上属于D-path的snake, 再以此为分割进行反复迭代，实际是就可以求解出最后的所有属于D-path的snake,即track, 当track确定时实际D-path 就确定了，D-path确定时同样，LCS就确定了。

1. 为什么从两个方向迭代时，最后重叠于k-line 中的middle snake 一定是最终的D-path中的一段snake呢？

这个问题实际可以从定理3中看出，结合编辑图与D-path的定义，实际上，若我们假设求得的middle snake是正向求解的 D-path中的一条snake, 那么middle snake必定会属于正向D-path中的一条snake，再将问题分解成子问题，只需要保证 每次得到的middle snake分为正向的snake中一条即可。对于前半部分，由于判断重叠的条件一致，最后得到的snake， 必定会是正向snake中的一条，而对于后半部分，由于是以正向snake的终点为起点的，在相同的重叠判断条件下，实际 上也会得到的middle snake也会是正向D-path的中一条snake,反之，亦然。

这样，以上算法可以用下面的计算过程描述:

∆ ← N−M

For D ← 0 to ✷( M + N ) / 2 ✸ Do

For k ← −D to D **in** steps of 2 Do

Find the end of the furthest reaching forward D-path **in** diagonal k

If ∆ **is** odd **and** k ∈ [ ∆ − ( D − 1 ) , ∆ + ( D − 1 ) ] Then

If the path overlaps the furthest reaching reverse ( D − 1 )-path **in** diagonal k Then

Length of an SES **is** 2D−1

The last snake of the forward path **is** the middle snake

For k ← −D to D **in** steps of 2 Do

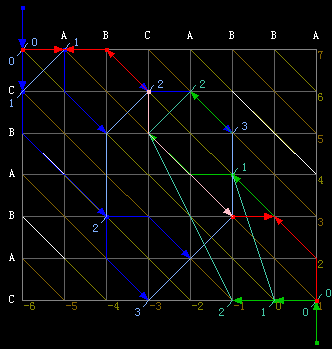
Find the end of the furthest reaching reverse D-path **in** diagonal k+∆

If ∆ **is** even **and** k + ∆ ∈ [ − D, D ] Then

If the path overlaps the furthest reaching forward D-path **in** diagonal k+∆ Then

Length of an SES **is** 2D

The last snake of the reverse path **is** the middle snake

下面是串CBABAC与 串ABCABBA的迭代图：[](http://yaowhat.com/2014/07/23/entry-version-diff-2.html)

下面是具体的算法的实现代码:

void php\_qvdiff\_bisect(zval \*qvdiff, qvdiff\_wchar\_t \*left, qvdiff\_wchar\_t \*right, int deadline)

{

zval \*changes, \*diff;

int text1\_len = left->len;

int text2\_len = right->len;

wchar\_t \*text1 = MB\_STR\_HEAD\_P(left);

wchar\_t \*text2 = MB\_STR\_HEAD\_P(right);

int maxD = (int) ((text1\_len + text2\_len + 1) / 2);

int voffset = maxD;

int vlength = 2 \* maxD;

int \*v1 = (int\*)ecalloc(vlength + 1, sizeof(int));

int \*v2 = (int\*)ecalloc(vlength + 1, sizeof(int));

int i;

for (i = 0; i <= vlength; i++) {

v1[i] = -1;

v2[i] = -1;

}

v1[voffset + 1] = 0;

v2[voffset + 1] = 0;

int delta = text1\_len - text2\_len;

int front = delta % 2 != 0;

int k1start = 0, k1end = 0, k2start = 0, k2end = 0;

int d, k1, k2, x1, x2, y1, y2, k1offset, k2offset;

for (d = 0; d < maxD; d++) {

for (k1 = -d + k1start; k1 < d + 1 -k1end; k1 += 2) {

k1offset = voffset + k1;

**if** (k1 == -d || (k1 != d && v1[k1offset - 1] < v1[k1offset + 1])) {

x1 = v1[k1offset + 1];

} else {

x1 = v1[k1offset - 1] + 1;

}

y1 = x1 - k1;

while (x1 < text1\_len && y1 < text2\_len && text1[x1] == text2[y1]) {

x1++;

y1++;

}

v1[k1offset] = x1;

**if** (x1 > text1\_len) {

k1end += 2;

} else if (y1 > text2\_len) {

k1start += 2;

} else if (front) {

k2offset = voffset + delta - k1;

**if** (k2offset >= 0 && k2offset < vlength && v2[k2offset] != -1) {

x2 = text1\_len - v2[k2offset];

**if** (x1 >= x2) {

php\_qvdiff\_bisect\_split(qvdiff, left, right, x1, y1, deadline);

goto end;

}

}

}

}

for (k2 = -d + k2start; k2 < d + 2 -k2end; k2 += 2) {

k2offset = voffset + k2;

**if** (k2 == -d || (k2 != d && v2[k2offset - 2] < v2[k2offset + 2])) {

x2 = v2[k2offset + 1];

} else {

x2 = v2[k2offset - 1] + 1;

}

y2 = x2 - k2;

while (x2 < text1\_len && y2 < text2\_len && text1[text1\_len - x2 - 1] == text2[text2\_len - y2 - 1]) {

x2++;

y2++;

}

v2[k2offset] = x2;

**if** (x2 > text1\_len) {

k2end += 2;

} else if (y2 > text2\_len) {

k2start += 2;

} else if (!front) {

k1offset = voffset + delta - k2;

**if** (k1offset >= 0 && k1offset < vlength && v1[k1offset] != -1) {

x1 = v1[k1offset];

y1 = voffset + x1 - k1offset;

x2 = text1\_len - x2;

**if** (x1 >= x2) {

php\_qvdiff\_bisect\_split(qvdiff, left, right, x1, y1, deadline);

goto end;

}

}

}

}

}

//意外或者超时情况的处理

changes = zend\_read\_property(qv\_diff\_ce, qvdiff, ZEND\_STRL("\_changes"), 0 TSRMLS\_CC);

diff = php\_qvdiff\_set\_diff\_code(QV\_DIFF\_DELETE, left);

add\_next\_index\_zval(changes, diff);

diff = php\_qvdiff\_set\_diff\_code(QV\_DIFF\_INSERT, right);

add\_next\_index\_zval(changes, diff);

zend\_update\_property(qv\_diff\_ce, qvdiff, ZEND\_STRL("\_changes"), changes TSRMLS\_CC);

end:

efree(v1);

efree(v2);

}

上面的代码是一php扩展的部分，具体使用时需要适当修改。