Epsm算法分为两个阶段。

首先定义查询表，它是记录采样串在模式串的出现位置，其中epsm采用固定长度为8bytes的采样串。程序中由一指针数组Node\* shift[2048]来实现。其中结点Node，数据结构如下：

typedef struct node

{

struct node \*next;

int pos;

} NODE;

它存储某一采样串p1在模式串p中的位置，其中p1通过计算hash值h去定位到它的结点shift[h]。

那么在预处理阶段，对于模式串P，从串位置i起取8bytes算一个hash值h，那么对于采样串p1，算得hash值h后，将p1在p中出现的位置i存储在shift[h]中，对于hash冲突造成的重值，存储在shift[h]对应的下一链表结点中。

在搜索阶段，首先指针指向匹配串T的m-8位置，并取8bytes子串，算hash值h，若shift[h]有值i，说明该子串可能是串P在i位置的一部分，然后根据i值将串P的i位置和该子串对齐，并调用memcmp函数进行比较，若shift[h]有多个链表结点，则需根据i值进行多次对齐和比较。完成后指针再继续前进m-8的长度，并重复刚才的搜索步骤。其具体算法描述如图1。

根据鸽笼原理，m-8是epsm不漏扫的最大距离。

~~而我们的优化点在于用BM的思想，使指针的移动距离尽量达到m，即初始化阶段根据模式串P计算出一偏移表shift，该表数据结构为hash表，传入任意8bytes字符串即可得到指针下一步的偏移长度。~~

所以容易看出，如果采样串长度α较小，即能获取较大的跳跃距离m-α，但与此同时数组结点对应的链表会很长，这样每次比较都会去遍历这个链表，另外α越小，采样串的匹配概率越大，导致mencmp全匹配函数的调用次数加大，抵消改进；而根据经验，shift数组越大，程序会因cpu缓存原因查询越慢，所以合适的大小是2048。但对于基因串来说，4的8次方为65536，会造成较大的hash冲突，同样降低性能。

所以我们使用两个方法去提高算法在基因串的查询性能。

一、我们可以通过实验去找到合适的采样串长度，使得它既能获取较大的跳跃距离，同时又不造成很高的hash冲突；

二、因为不论α选多大，我们实际上都是读取8bytes的采样串p0，但通过掩码来获取变长的采样串p1， 所以我们通过在调用memcmp前比较p0对应的64bit数值与对应模式串位置的64bit数值，这样就能大大减少全比较函数调用次数。具体实现是在Node定义中加一字段存储原采样串的64bit数值，同时在预处理阶段根据模式串P将其初始化。如下所示：

typedef struct node

{

struct node \*next;

int pos;

unsigned long long val;

} NODE;

通过实验，我们获悉将α选为6并加上long值比较能在基因数据匹配上获取比epsm更好的性能。