Aho-Corasick 多模式匹配算法、AC自动机详解

Aho-Corasick算法是多模式匹配中的经典算法，目前在实际应用中较多。

Aho-Corasick算法对应的数据结构是Aho-Corasick自动机，简称AC自动机。

搞编程的一般都应该知道自动机FA吧，具体细分为：确定性有限状态自动机(DFA)和非确定性有限状态自动机NFA。**普通的自动机不能进行多模式匹配，AC自动机增加了失败转移**，转移到已经输入成功的文本的后缀，来实现。

1.多模式匹配

多模式匹配就是有多个模式串P_1,P_2,P_3...，P_m，求出所有这些模式串在连续文本$T_{1....n}$中的所有可能出现的位置。

例如：求出模式集合{"nihao","hao","hs","hsr"}在给定文本"sdmfhsgnshejfgnihaofhsrnihao"中所有可能出现的位置。

2.Aho-Corasick算法

使用Aho-Corasick算法需要三步：

1.建立模式的Trie

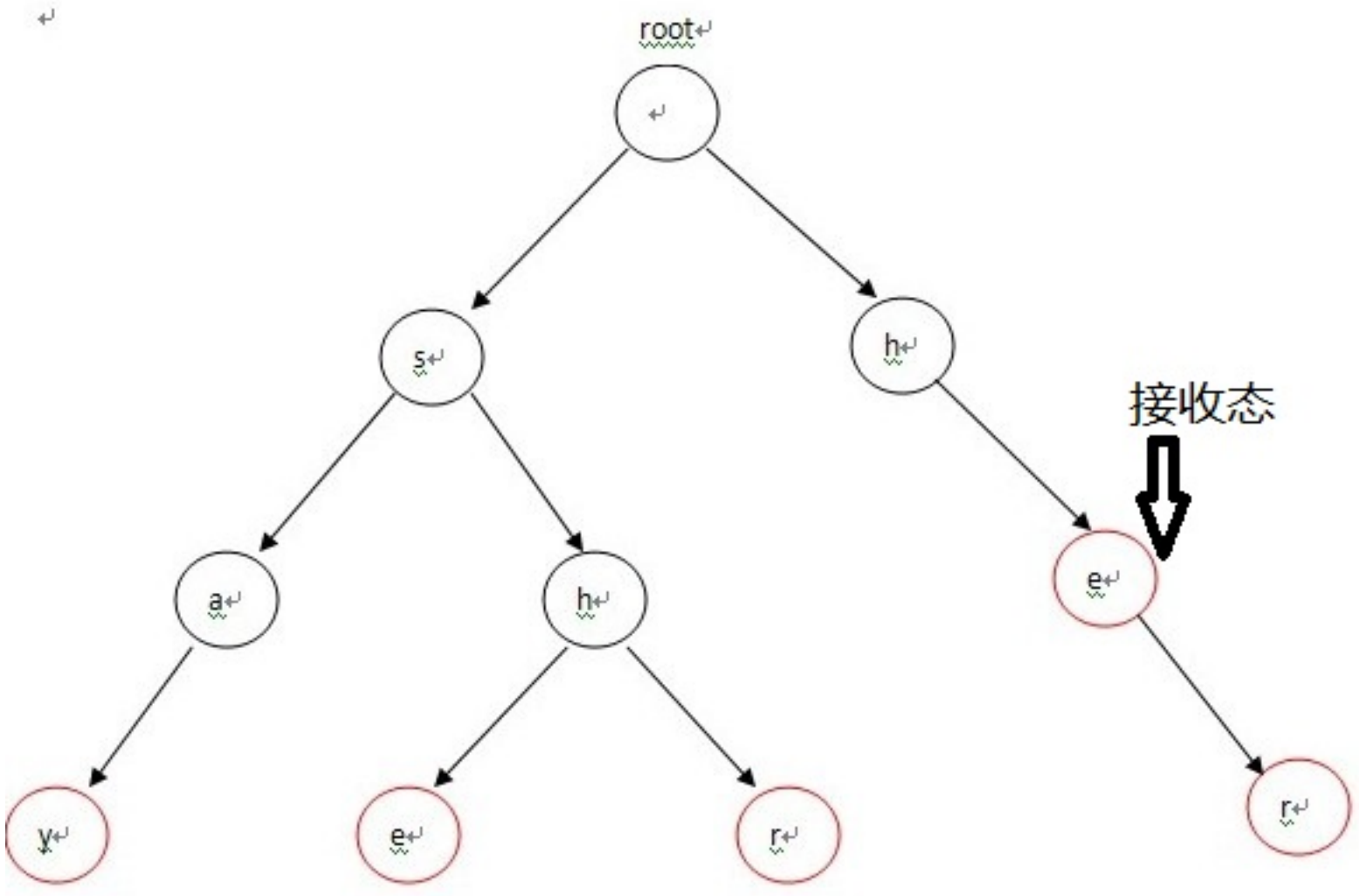
2.给Trie添加失败路径

3.根据AC自动机，搜索待处理的文本

下面说明这三步：

2.1建立多模式集合的Trie树

Trie树也是一种自动机。对于多模式集合{"say","she","shr","he","her"}，对应的Trie树如下，其中红色标记的圈是表示为接收态：

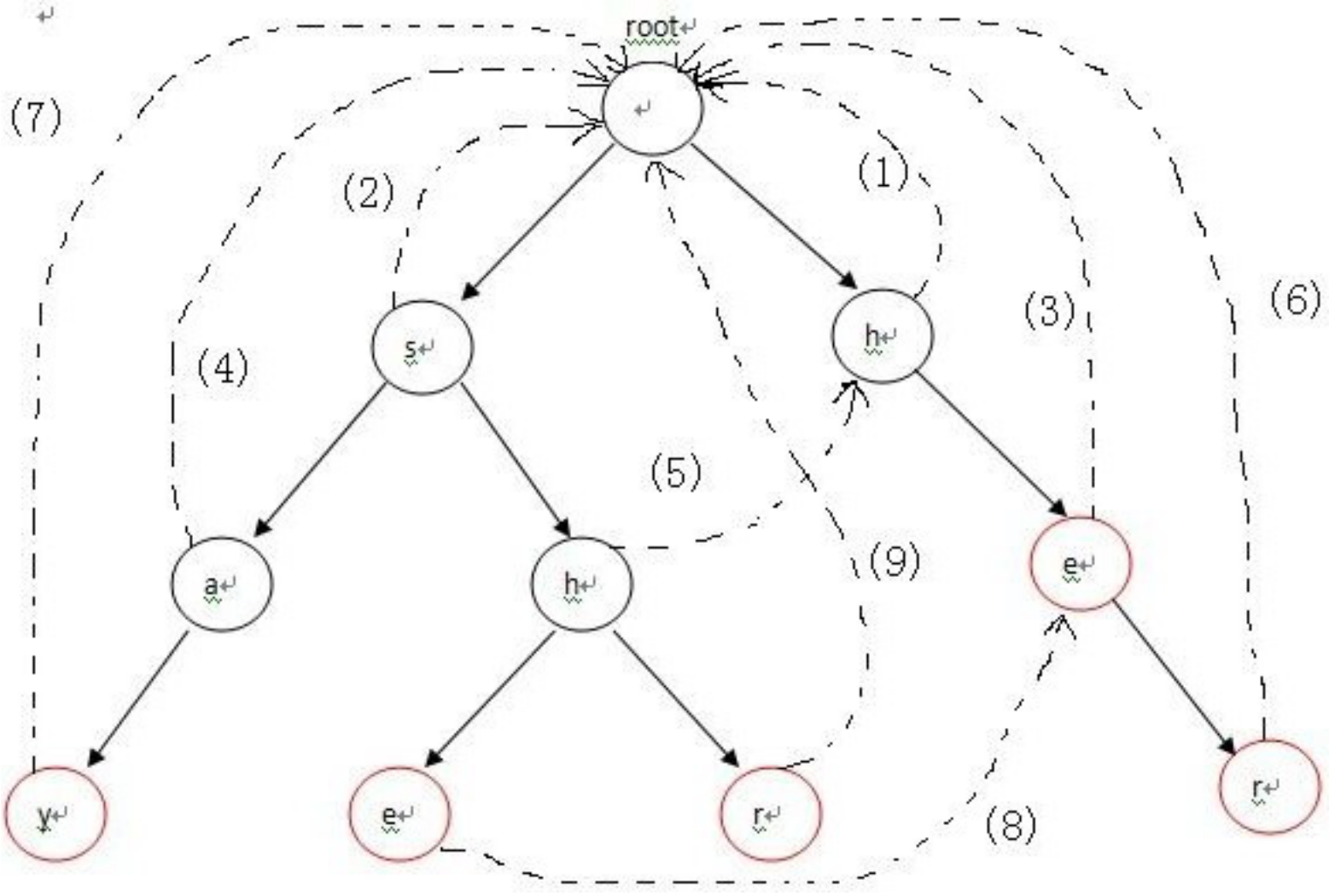


2.2为多模式集合的Trie树添加失败路径，建立AC自动机

构造失败指针的过程概括起来就一句话：设这个节点上的字母为C，沿着他父亲的失败指针走，直到走到一个节点，他的儿子中也有字母为C的节点。然后把当前节点的失败指针指向那个字母也为C的儿子。如果一直走到了root都没找到，那就把失败指针指向root。

使用广度优先搜索BFS，层次遍历节点来处理，每一个节点的失败路径。

特殊处理：第二层要特殊处理，将这层中的节点的失败路径直接指向父节点(也就是根节点)。



2.3根据AC自动机，搜索待处理的文本

从root节点开始，每次根据读入的字符沿着自动机向下移动。

当读入的字符，在分支中不存在时，递归走失败路径。如果走失败路径走到了root节点，则跳过该字符，处理下一个字符。

因为AC自动机是沿着输入文本的最长后缀移动的，所以在读取完所有输入文本后，最后递归走失败路径，直到到达根节点，这样可以检测出所有的模式。

3.Aho-Corasick算法代码示例

模式串集合：{"nihao","hao","hs","hsr"}

待匹配文本："sdmfhsgnshejfgnihaofhsrnihao"

代码：

```
1 #include<iostream>
2 #include<string.h>
3 #include<malloc.h>
```

```
#include <queue>
using namespace std;

typedef struct node{
    struct node *next[26];  //接收的态
    struct node *par;   //父亲节点
    struct node *fail;  //失败节点
    char inputchar;
    int patterTag;    //是否为可接收态
    int patterNo;   //接收态对应的可接受模式
}*Tree,TreeNode;
char pattern[4][30]={"nihao","hao","hs","hsr"};

/**
申请新的节点，并进行初始化
*/
TreeNode *getNewNode()
{
    int i;
    TreeNode* tnode=(TreeNode*)malloc(sizeof(TreeNode));
    tnode->fail=NULL;
    tnode->par=NULL;
    tnode->patterTag=0;
    for(i=0;i<26;i++)
        tnode->next[i]=NULL;
    return tnode;
}

/**
将Trie树中，root节点的分支节点，放入队列
*/
int  nodeToQueue(Tree root,queue<Tree> &myqueue)
{
    int i;
    for (i = 0; i < 26; i++)
    {
        if (root->next[i]!=NULL)
            myqueue.push(root->next[i]);
    }
    return 0;
}

/**
建立trie树
*/
Tree buildingTree()
{
    int i,j;
```

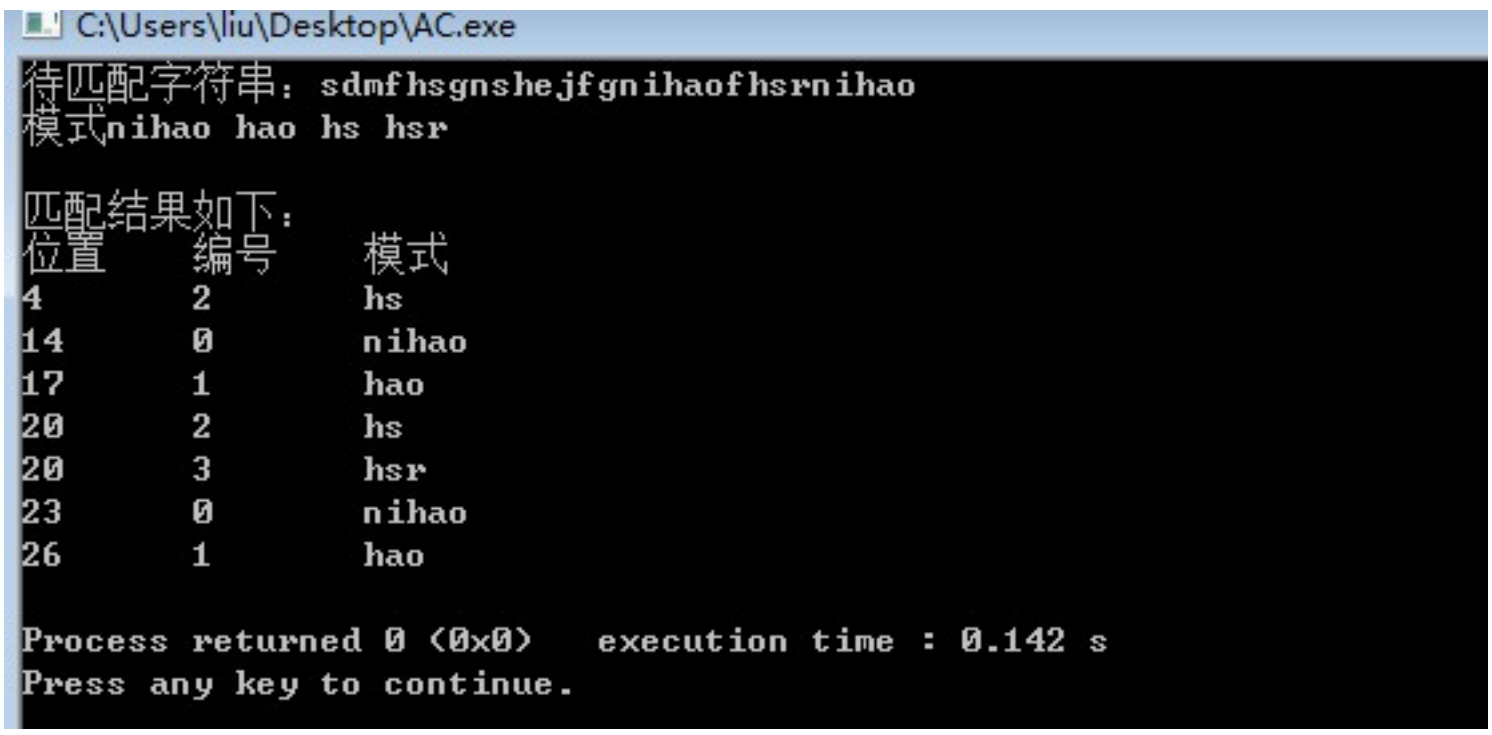
```
52      Tree root=getNewNode();
53      Tree tmp1=NULL,tmp2=NULL;
54      for(i=0;i<4;i++)
55      {
56          tmp1=root;
57          for(j=0;j<strlen(pattern[i]);j++)   ///对每个模式进行处理
58          {
59              if(tmp1->next[pattern[i][j]-'a']==NULL) ///是否已经有分支,
Trie共用节点
60              {
61                  tmp2=getNewNode();
62                  tmp2->inputchar=pattern[i][j];
63                  tmp2->par=tmp1;
64                  tmp1->next[pattern[i][j]-'a']=tmp2;
65                  tmp1=tmp2;
66              }
67              else
68                  tmp1=tmp1->next[pattern[i][j]-'a'];
69          }
70          tmp1->patterTag=1;
71          tmp1->patterNo=i;
72      }
73      return root;
74  }
75
76  /**
77  建立失败指针
78  */
79  int buildingFailPath(Tree root)
80  {
81      int i;
82      char inputchar;
83      queue<Tree> myqueue;
84      root->fail=root;
85      for(i=0;i<26;i++)   ///对root下面的第二层进行特殊处理
86      {
87          if (root->next[i]!=NULL)
88          {
89              nodeToQueue(root->next[i],myqueue);
90              root->next[i]->fail=root;
91          }
92      }
93
94      Tree tmp=NULL,par=NULL;
95      while(!myqueue.empty())
96      {
97          tmp=myqueue.front();
98          myqueue.pop();
```

```
 99         nodeToQueue(tmp,myqueue);
100
101         inputchar=tmp->inputchar;
102         par=tmp->par;
103
104         while(true)
105         {
106             if(par->fail->next[inputchar-'a']!=NULL)
107             {
108                 tmp->fail=par->fail->next[inputchar-'a'];
109                 break;
110             }
111             else
112             {
113                 if(par->fail==root)
114                 {
115                     tmp->fail=root;
116                     break;
117                 }
118                 else
119                     par=par->fail->par;
120             }
121         }
122     }
123     return 0;
124 }
125
126 /**
127 进行多模式搜索，即搜寻AC自动机
128 */
129 int searchAC(Tree root,char* str,int len)
130 {
131     TreeNode *tmp=root;
132     int i=0;
133     while(i < len)
134     {
135         int pos=str[i]-'a';
136         if (tmp->next[pos]!=NULL)
137         {
138             tmp=tmp->next[pos];
139             if(tmp->patterTag==1)    ///如果为接收态
140             {
141                 cout<<i-strlen(pattern[tmp->patterNo])+1<<'\t'<<tmp-
>patterNo<<'\t'<<pattern[tmp->patterNo]<<endl;
142             }
143             i++;
144         }
145         else
```

```
146         {
147             if(tmp==root)
148                 i++;
149             else
150             {
151                 tmp=tmp->fail;
152                 if(tmp->patterTag==1)    //如果为接收态
153                     cout<<i-strlen(pattern[tmp-
>patterNo])+1<<'\t'<<tmp->patterNo<<'\t'<<pattern[tmp->patterNo]<<endl;
154             }
155         }
156     }
157     while(tmp!=root)
158     {
159         tmp=tmp->fail;
160         if(tmp->patterTag==1)
161             cout<<i-strlen(pattern[tmp->patterNo])+1<<'\t'<<tmp-
>patterNo<<'\t'<<pattern[tmp->patterNo]<<endl;
162     }
163     return 0;
164 }
165
166 /**
167 释放内存，DFS
168 */
169 int destory(Tree tree)
170 {
171     if(tree==NULL)
172         return 0;
173     queue<Tree> myqueue;
174     TreeNode *tmp=NULL;
175
176     myqueue.push(tree);
177     tree=NULL;
178     while(!myqueue.empty())
179     {
180         tmp=myqueue.front();
181         myqueue.pop();
182
183         for (int i = 0; i < 26; i++)
184         {
185             if(tmp->next[i]!=NULL)
186                 myqueue.push(tmp->next[i]);
187         }
188         free(tmp);
189     }
190     return 0;
191 }
```

```
192
193 int main()
194 {
195     char a[]="sdmfhsgnshejfgnihaofhsrnihao";
196     Tree root=buildingTree();   ///建立Trie树
197     buildingFailPath(root); ///添加失败转移
198     cout<<"待匹配字符串：<<a<<endl;
199     cout<<"模式"<<pattern[0]<<" "<<pattern[1]<<" "<<pattern[2]<<" "
<<pattern[3]<<" "<<endl<<endl;
200     cout<<"匹配结果如下：<<endl<<"位置\t"<<"编号\t"<<"模式"<<endl;
201     searchAC(root,a,strlen(a)); ///搜索
202     destory(root);  ///释放动态申请内存
203     return 0;
204 }
```

输出：



（上面的两个图，参考网页：http://www.cppblog.com/mythit/archive/2009/04/21/80633.html）