

Algorithm Code Book

Tanvir Hasan Anick

July 8, 2015

Contents

1	Data Structure	2
2	Graph Theory	3
3	Flow networks/ matching	4
4	Dynamic programming	5
5	Strings	6
5.1	KMP	6
5.2	Trie	7
5.2.1	Static Trie	7
5.3	Aho Corasick	8
5.3.1	Aho Corasick with Dynamic Trie	8
5.3.2	Aho Corasick with Static Trie	10
5.4	Manacher's Algorithm	13
6	Computational geometry	14
7	Math	15
8	Number Theory	16

Chapter 1

Data Structure

Chapter 2

Graph Theory

Chapter 3

Flow networks/ matching

Chapter 4

Dynamic programming

Chapter 5

Strings

5.1 KMP

Tutorial

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 char TXT[10000000], ptr[10000000];
4 vector<int> compute_prefix(const char *p)
5 {
6     int m=strlen(p+1);
7     vector<int> prefix(m+1);
8     prefix[1]=0;
9     int k=0;
10    for(int i=2; i<=m; i++)
11    {
12        while(k>0 and p[k+1]!=p[i]) k=prefix[k];
13        if(p[k+1]==p[i]) k=k+1;
14        prefix[i]=k;
15    }
16    return prefix;
17 }
18 vector<int> KMP_match(const char *txt, const char *ptrn)
19 {
20     int n=strlen(txt+1);
21     int m=strlen(ptrn+1);
22     vector<int> Prefix=compute_prefix(ptrn);
23     vector<int> Match_position;
24     int q=0;
25     for(int i=1; i<=n; i++)
26     {
27         while(q>0 and ptrn[q+1]!=txt[i]) q=Prefix[q];
28         if(ptrn[q+1]==txt[i]) q=q+1;
29         if(q==m)
30         {
31             Match_position.push_back(i-m);
32             q=Prefix[q];
33         }
34     }
```

```

35     return Match_position;
36 }
37 int main()
38 {
39     scanf("%s %s",TXT+1,ptr+1);
40     vector<int> Match_position=KMP.match(TXT,ptr);
41     for(int i=0; i<Match_position.size(); i++)
42     {
43         if(!i) printf("%d",Match_position[i]);
44         else printf(" %d",Match_position[i]);
45     }
46     return 0;
47 }

```

5.2 Trie

5.2.1 Static Trie

```

1 #define Max 10005
2 int getId(char c)
3 {
4     return c>='a'?c-'a':c-'A'+26;
5 }
6 struct Trie
7 {
8     struct Tree
9     {
10         int Next[52];
11         bool word;
12         void clear()
13         {
14             word=false;
15             memset(Next,-1,sizeof(Next));
16         }
17     }T[Max];
18     int ptr;
19     void clear()
20     {
21         ptr=1;
22         T[0].clear();
23         memset(T[0].Next,0,sizeof(T[0].Next));
24     }
25     void Insert(const char *str)
26     {
27         int p=0;
28         for(int i=0;str[i];i++)
29         {
30             int id=getId(str[i]);
31             if(T[p].Next[id]<=0)
32             {
33                 T[p].Next[id]=ptr;
34                 T[ptr++].clear();
35             }
36             p=T[p].Next[id];
37         }

```



```

38     T[p].word=true;
39 }
40 bool Search(const char *str)
41 {
42     int p=0;
43     for(int i=0;str[i];i++)
44     {
45         int id=getId(str[i]);
46         if(T[p].Next[id]>0)
47         {
48             p=T[p].Next[id];
49         }
50         else return false;
51     }
52     return T[p].word;
53 }
54 };
55 Trie A;

```

5.3 Aho Corasick

5.3.1 Aho Corasick with Dynamic Trie

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define Max 26
4 int getID(char c)
5 {
6     return c>='a'?c-'a':c-'A';
7 }
8 char inp[1000005];
9 char text[1000005];
10 int ans[5000];
11 map<string,int>Map;
12 vector<int>v;
13 struct Trie
14 {
15     Trie *next[26],*fail;
16     int stringMap;
17     Trie()
18     {
19         stringMap=0;
20         for(int i=0;i<Max;i++)next[i]=NULL;
21         fail=NULL;
22     }
23 };
24 Trie *root;
25 void Insert(const char *str,int M)
26 {
27     Trie *p=root;
28     for(int i=0;str[i];i++)
29     {
30         int id=getID(str[i]);
31         if(p->next[id]==NULL)p->next[id]=new Trie();
32         p=p->next[id];
33     }

```

```

34     p->stringMap=M;
35 }
36 void computeFailure()
37 {
38     Trie *u,*prefix;
39     queue<Trie*>Q;
40     Q.push(root);
41     while(!Q.empty())
42     {
43         u=Q.front(); ///Take a new node
44         Q.pop();
45         for(int i=0;i<Max;i++)
46         {
47             if(u->next[i]!=NULL) ///select fail position of ith
node of parent u
48             {
49                 prefix=u->fail; /// Going to u node fail position/
prefix position
50                 while(prefix!=NULL)
51                 {
52                     if(prefix->next[i]!=NULL) ///if match found
53                     {
54                         u->next[i]->fail=prefix->next[i];
55                         break;
56                     }
57                     prefix=prefix->fail; /// match not found, going
to upper child prefix position
58                 }
59                 if(prefix==NULL)u->next[i]->fail=root;
60                 Q.push(u->next[i]);
61             }
62         }
63     }
64 }
65 void AhoCorasick(const char *str)
66 {
67     Trie *p=root;
68     int cnt=0;
69     for(int i=0;str[i];i++)
70     {
71         int id=getID(str[i]);
72         while(p->next[id]==NULL&&p!=root)p=p->fail, cnt++;
73         if(p->next[id]!=NULL)p=p->next[id];
74         Trie *tp=p;
75         while(tp!=root)
76         {
77             cnt++;
78             if(tp->stringMap>0)ans[tp->stringMap]++;
79             tp=tp->fail;
80         }
81     }
82 }
83 void Delete(Trie *u)
84 {
85     if(u==NULL) return;
86     for(int i=0;i<Max;i++)Delete(u->next[i]);
87     delete u;

```

```

88 }
89
90 int main()
91 {
92     int test;
93     scanf("%d",&test);
94     for(int t=1;t<=test;t++)
95     {
96         Map.clear();
97         v.clear();
98         memset(ans,0,sizeof(ans));
99         root=new Trie();
100         int N;
101         scanf("%d",&N);
102         scanf("%s",text);
103         int cnt=1;
104         for(int i=0;i<N;i++)
105         {
106             scanf("%s",inp);
107             if(Map.find(inp)==Map.end())Map[inp]=cnt++;
108             Insert(inp,Map[inp]);
109             v.push_back(Map[inp]);
110         }
111         computeFailure();
112         AhoCorasick(text);
113         printf("Case %d:\n",t);
114         for(int i=0;i<N;i++)
115         {
116             printf("%d\n",ans[v[i]]);
117         }
118         Delete(root);
119     }
120     return 0;
121 }

```

5.3.2 Aho Corasick with Static Trie

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define root 0
4 #define NuLL -1
5 #define Max 248878
6 #define MC 26
7 int ans[10000];
8 char text[1000005];
9 char inp[100000];
10 map<string,int>Map;
11 vector<int> v;
12 int getID(const char c)
13 {
14     return c>='a'?c-'a':c-'A';
15 }
16 struct Trie
17 {
18     struct node
19     {
20         int Next[26],fail;

```

```

21     int stringMap;
22     void clear()
23     {
24         memset(Next, -1, sizeof(Next));
25         fail=-1;
26         stringMap=0;
27     }
28 }T[Max];
29 int ptr;
30 void clear()
31 {
32     ptr=1;
33     T[0].clear();
34 }
35 void Insert(char *str, int M)
36 {
37     int p=0;
38     for(int i=0; str[i]; i++)
39     {
40         int id=getID(str[i]);
41         if(T[p].Next[id]==-1)
42         {
43             T[p].Next[id]=ptr;
44             T[ptr++].clear();
45         }
46         int q=p;
47         p=T[p].Next[id];
48         if(p<0)
49         {
50             while(1);
51         }
52     }
53     T[p].stringMap=M;
54 }
55 void ComputeFailure()
56 {
57     queue<int>Q;
58     Q.push(root);
59     int u, prefix;
60     int cnt=0, cnt2=0;
61     while(!Q.empty())
62     {
63         u=Q.front();
64         Q.pop();
65         for(int i=0; i<MC; i++)
66         {
67             if(T[u].Next[i]!=NULL)
68             {
69                 int now=T[u].Next[i];
70                 prefix=T[u].fail;
71                 while(prefix!=NULL)
72                 {
73                     cnt2++;
74                     if(T[prefix].Next[i]!=NULL)
75                     {
76                         T[now].fail=T[prefix].Next[i];
77                         break;

```

```

78         }
79         prefix=T[prefix].fail;
80     }
81     if (prefix==NULL)T[now].fail=root;
82     Q.push(now);
83 }
84 }
85 }
86 }
87 };
88 void AhoCorasick(const Trie &A, const char *str)
89 {
90     int p=root;
91     int cnt1=0,cnt2=0;
92     for (int i=0;str[i];i++)
93     {
94         int id=getID(str[i]);
95         while (A.T[p].Next[id]==NULL&&p!=root)p=A.T[p].fail;
96         if (p!=NULL&&A.T[p].Next[id]!=NULL)p=A.T[p].Next[id];
97         int tp=p;
98         while (tp!=root)
99         {
100             if (A.T[tp].stringMap>0)ans[A.T[tp].stringMap]++;
101             tp=A.T[tp].fail;
102         }
103     }
104 }
105 Trie A;
106 int main()
107 {
108     #ifdef _ANICK_
109         freopen("input.txt","r",stdin);
110     #endif // _ANICK_
111     int test;
112     scanf("%d",&test);
113     for (int t=1;t<=test;t++)
114     {
115         Map.clear();
116         v.clear();
117         memset(ans,0,sizeof(ans));
118         A.clear();
119         int N;
120         scanf("%d",&N);
121         scanf("%s",text);
122         int cnt=1;
123         for (int i=0;i<N;i++)
124         {
125             scanf("%s",inp);
126             if (Map.find(inp)==Map.end())Map[inp]=cnt++;
127             A.Insert(inp,Map[inp]);
128             v.push_back(Map[inp]);
129         }
130         A.ComputeFailure();
131         AhoCorasick(A,text);
132         printf("Case %d:\n",t);
133         for (int i=0;i<N;i++)
134         {

```

```

135         printf("%d\n", ans[v[i]]);
136     }
137 }
138 return 0;
139 }

```

5.4 Manacher's Algorithm

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 string s, t;
4 char str[1000005];
5 void prepare_string()
6 {
7     int i;
8     t = "^#";
9     for(i = 0; i < s.size(); i++)
10         t += s[i], t += "#";
11     t += "$";
12 }
13
14 int manacher()
15 {
16     prepare_string();
17
18     int P[t.size()], c = 0, r = 0, i, i_mirror, n = t.size() - 1;
19
20     for(i = 1; i < n; i++)
21     {
22         i_mirror = (2 * c) - i;
23
24         P[i] = r > i ? min(r - i, P[i_mirror]) : 0;
25
26         while(t[i + 1 + P[i]] == t[i - 1 - P[i]])
27             P[i]++;
28
29         if(i + P[i] > r)
30         {
31             c = i;
32             r = i + P[i];
33         }
34     }
35     return *max_element(P + 1, P + n);
36 }
37
38 int main()
39 {
40     int kase = 1;
41     while(scanf("%s", str) && str[0] != 'E')
42     {
43         s = str;
44         printf("Case %d: %d\n", kase++, manacher());
45     }
46     return 0;
47 }

```

Chapter 6

Computational geometry

Chapter 7

Math

Chapter 8

Number Theory