ACM中常用算法----字符串

原创 2015年07月26日 11:41:58 标签: acm / 字符串 4390

ACM中常用算法—-字符串

ACM中常用的字符串算法不多,主要有以下几种:

1. Hash
2. 字典树
3. KMP
4. AC自动机
5. manacher
6. 后缀数组
7. EX_KMP
8. SAM(后缀自动机)
9. 回文串自动机

下面来分别介绍一下:

0. Hash

字符串的hash是最简单也最常用的算法,通过某种hash函数将不同的字符串分别对应到不同的数字.进而配合其他数据结构或STL可以做到判重,统计,查询等操作.

- #### 字符串的hash函数:

一个很简单的hash函数代码如下:

```
1
2  ull xp[maxn],hash[maxn];
3
4  void init()
5  {
6      xp[0]=1;
7      for(int i=1;i<maxn;i++)
8          xp[i]=xp[i-1]*175;
9  }
10
11 ull get_hash(int i,int L)
12 {
13     return hash[i]-hash[i+L]*xp[L];
14 }
15
16 scanf("%s",str);
17 int n=strlen(str);
18 hash[n]=0;
19 for(int i=n-1;i>=0;i--)
20 {
21     hash[i]=hash[i+1]*175+(str[i]-'a'+1);
22 }
```

其中175是顺便选择的基数,对一个串通过init的预处理后,就用get_hash(i,L)可以得到从位置i开始的,长度为L的子串的hash值.

- **其他的一些hash函数介绍** 字符串hash函数
- **hash函数可能会遇到的问题**

码代码的猿猿的AC之
关注
原创 934 粉丝 135 喜欢 3
等级: 博客 访问量: 9
积分: 1万+ 排名: 627

一般情况下,这个简单的hash函数已经足够好了.但使用hash函数解题的时候还是有问题要注意:

1. hash函数的结果并不一定准确,hash的值可能会有冲突导致结果错误(但不常遇到可以换hash数即可).
2. 对于一般的字符串,这个hash函数准确性很高. 但是有的题目会刻意构造可以使hash函数失效的字符串,无论换什么样的hash数都过不了,这时就需要对hash函数进行修改,不能使用自然溢出的方式储存hash值,可以选取两个大质数,对用一个字符串记录它的hash值和这两个数的mod.用这种方法可以过掉几乎全部卡hash函数的题

例题

- HDOJ 4821 String
- HDOJ 4080 Stammering Aliens
- HDOJ 4622 Reincarnation
- CSU1647: SimplePalindromicTree

1. 字典树

字典树是储存着不同字符串的数据结构,是一个n叉树(n为字符集的大小),对于一棵储存26个字母的字典树来说,它的的每一个节点储存着26个指针可以分别代表这个节点的后面加上'a'~'z'后可以指向那个节点.

插入的时候从根节点开始,沿着对应的边走(如果某个指针后面指向的节点为空.可以新建一个节点),走到字符串结束的时候在当前停留的节点标记一下(是否出现过,出现了几次等).

查询的时候也是一样从根节点走,如果走到某个节点无路可走了,说明查不到.当一路走到字符串结束时,检查当前停留的节点是否被标记过.

一份代码参考:

```
/*字典树*/

const int CHAR=26,MAXN=100000;

struct Trie
{
    int tot,root,child[MAXN][CHAR];
    bool flag[MAXN];

    Trie()
    {
        memset(child[1],0,sizeof(child[1]));
        flag[1]=true;
        root=tot=1;
    }

    void Insert(const char *str)
    {
        int *cur=&root;
        for(const char*p=str;*p;p++)
        {
            cur=&child[*cur][*p-'a'];
            if(*cur==0)
            {
                *cur=++tot;
                memset(child[tot],0,sizeof(child[tot]));
                flag[tot]=false;
            }
        }

        flag[*cur]=true;
    }

    bool Query(const char *str)
    {
        int *cur=&root;
        for(const char *p=str;*p&&*cur;p++)
            cur=&child[*cur][*p-'a'];
        return (*cur)&&flag[*cur];
```

```
    }

}tree;
```

例题

- POJ 3630 Phone List
- HDOJ 4622 Reincarnation
- HDOJ 1251 统计难题

2. KMP

kmp是一种字符串匹配的算法,普通的字符串匹配需要时间O(n*m) *n:字符串长度 m:模版串长度*,kmp算法通过对模版串进行预处理来找到每个位置的后缀和第一个字母的前缀的最大公共长度,可以让复制度降低到O(n+m)

关于KMP算法白书有很详细的介绍,网上也有很多.

KMP资料1 , KMP资料2

一种实现:

```

char t[1000],p[1000];
int f[1000];

void getfail(char* p,int* f)
{
    int m=strlen(p);
    f[0]=f[1]=0;
    for(int i=1;i<m;i++)
    {
        int j=f[i];
        while(j&&p[j]!=p[i]) j=f[j];
        f[i+1]=(p[i]==p[j])?j+1:0;
    }
}

void kmp(char* t,char* p,int* f)
{
    int n=strlen(t),m=strlen(p);
    getfail(p,f);
    int j=0;
    for(int i=0;i<n;i++)
    {
        while(j&&p[j]!=t[i]) j=f[j];
        if(p[j]==t[i]) j++;
        if(j==m)
        {
            ///i-m+1
            /// ans++;
            j=f[j];
        }
    }
}
```

例题

- HDOJ 1686 Oulipo
- Codeforces 346 B. Lucky Common Subsequence
- KMP+DP: Codeforces 494B. Obsessive String
- ZOJ 3587 Marlon's String
  kmp的应用不一定只在字符串中,只要是匹配问题都可以:
- CSU 1581 Clock Pictures

3. AC自动机

KMP是单字符串的匹配算法,如果有很多个模版串需要和文本串匹配,就需要用到AC自动机. AC自动机会预处理模版串,插入到一颗字典树中,并处理出fail指针.

具体实现可以看:
AC自动机1 , AC自动机2

我的一个模版:


```
/*
基于HDOJ 2222 的 AC自动机
文本串对多个模板串的查找
*/

const int maxn=610000;

int ch[maxn][26],fail[maxn],end[maxn];
int root,sz;
char str[1000100];

int newnode()
{
    memset(ch[sz],-1,sizeof(ch[sz]));
    end[sz++]=0;
    return sz-1;
}

void init()
{
    sz=0;
    root=newnode();
}

void insert(char str[])
{
    int len=strlen(str);
    int now=root;
    for(int i=0;i<len;i++)
    {
        int& temp=ch[now][str[i]-'a'];
        if(temp==-1) temp=newnode();
        now=temp;
    }
    end[now]++;
}

void build()
{
    queue<int> q;
    fail[root]=root;
    for(int i=0;i<26;i++)
    {
        int& temp=ch[root][i];
        if(temp==-1) temp=root;
        else
        {
            fail[temp]=root;
            q.push(temp);
        }
    }
    while(!q.empty())
    {
        int now=q.front(); q.pop();
        for(int i=0;i<26;i++)
        {
            if(ch[now][i]==-1)
                ch[now][i]=ch[fail[now]][i];
            else
            {
                fail[ch[now][i]]=ch[fail[now]][i];
                q.push(ch[now][i]);
```

```
            }
        }
    }
}

int query(char str[])
{
    int len=strlen(str);
    int now=root;
    int ret=0;
    for(int i=0;i<len;i++)
    {
        now=ch[now][str[i]-'a'];
        int temp=now;
        while(temp!=root&&~end[temp])
        {
            ret+=end[temp];
            end[temp]=-1;
            temp=fail[temp];
        }
    }
    return ret;
}
```

例题

- HDOJ 2222 Keywords Search
- UVA - 11468 Substring
- UvaLA 4670 Dominating Patterns
- HDOJ 2243 考研路茫茫
- POJ 1625 Censored!
- HDOJ 2896 病毒侵袭
- HDOJ 3065 病毒侵袭持续中

AC自动机+矩阵快速幂也是一种常见的类型:
* BZOJ 1009: [HNOI2008]GT考试
* POJ 2778 DNA Sequence

4. manacher

manacher是处理回文串问题的利器,manancher是一种dp方法和其他字符串关联不大,相对独立,manacher可以在O(1)的时间复杂度内处理出所有的位置的回文串的半径.

一篇很好的介绍: manacher

我的模版

```
//URAL 1297
//
//
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

char str[1100],ans[3300];
int p[3300],pos,how;

void pre()
{
    int tot=1;
    memset(ans,0,sizeof(ans));
    ans[0]='$';
    int len=strlen(str);
    for(int i=0;i<len;i++)
```

```
    {
        ans[tot]='#';tot++;
        ans[tot]=str[i];tot++;
    }
    ans[tot]='#';
}

void manacher()
{
    pos=-1;how=0;
    memset(p,0,sizeof(p));
    int len=strlen(ans);
    int mid=-1,mx=-1;
    for(int i=0;i<len;i++)
    {
        int j=-1;
        if(i<mx)
        {
            j=2*mid-i;
            p[i]=min(p[j],mx-i);
        }
        else p[i]=1;

        while(i+p[i]<len&&ans[i+p[i]]==ans[i-p[i]])
        {
            p[i]++;
        }

        if(p[i]+i>mx)
        {
            mx=p[i]+i; mid=i;
        }
        if(p[i]>how)
        {
            how=p[i]; pos=i;
        }
    }
}

int main()
{
    while(scanf("%s",str)!=EOF)
    {
        pre();
        manacher();
        how--;
        for(int i=pos-how;i<=pos+how;i++)
        {
            if(ans[i]!='#') putchar(ans[i]);
        }
        putchar(10);
    }
    return 0;
}
```

manacher在回文串问题中应用还是很多的,回文串自动机也可以处理回文串问题,但是略复杂.

在不用manacher的情况下也可以用 枚举+hash 也可以解决回文串问题. 具体做法可以枚举回文串中心点,二分出这个中心点的最大半径(一个大的半径的回文串肯定包含了小半径的回文串).

这是我曾经出过的一题,用的就是这种想法:
CSU1647: SimplePalindromicTree

例题

- HDOJ 3613 Best Reward
- URAL 1297 Palindrome
- USACO Calf Flac

5. 后缀数组

后缀数组的主要思想就是将某个字符串的后缀排序,这样取后缀的某一段前缀就是这个字符串的子串.

但是字符串的排序并不是O(1)的,所以后缀数组的代码中主要的一个部分就是为了加字符串的排序快排序速度.
常用的一种排序方法为*倍增法*

关于后缀数组排序,大白书中有详细的介绍.

罗穗骞后缀数组——处理字符串的有力工具

例题

- HDOJ 3948 The Number of Palindromes
- HDOJ 4691 Front compression
- POJ 3693 Maximum repetition substring
- POJ 2046 Power Strings
- URAL 1517 Freedom of Choice
- HDOJ 5008 Boring String Problem
- SPOJ 694 Distinct Substrings
- POJ 2774 Long Long Message
- HDOJ 4416 Good Article Good sentence
- HDOJ 4080 Stammering Aliens

~~*神奇的分割线*~~

> 以上的方法是非常常见的字符串处理方法,需要很好的理解和运用
> 下面介绍一些复杂一些的,但是在解决某些问题非常有用的方法

6. EXKMP

exkmp可以处理出模版串中每个位置i开始和模版开头的最大匹配长度,exkmp可以实现普通kmp的所有功能.

刘雅琼 的《扩展的KMP算法》介绍很好

```
/*
扩展KMP
next[i]: P[i..m-1] 与 P[0..m-1]的最长公共前缀
ex[i]: T[i..n-1] 与 P[0..m-1]的最长公共前缀
*/

char T[maxn],P[maxn];
int next[maxn],ex[maxn];

void pre_exkmp(char P[])
{
    int m=strlen(P);
    next[0]=m;
    int j=0,k=1;
    while(j+1<m&&P[j]==P[j+1]) j++;
    next[1]=j;
    for(int i=2;i<m;i++)
    {
        int p=next[k]+k-1;
        int L=next[i-k];
        if(i+L<p+1) next[i]=L;
        else
        {
            j=max(0,p-i+1);
            while(i+j<m&&P[i+j]==P[j]) j++;
            next[i]=j; k=i;
        }
    }
```

```
}

void exkmp(char P[],char T[])
{
    int m=strlen(P),n=strlen(T);
    pre_exkmp(P);
    int j=0,k=0;
    while(j<n&&j<m&&P[j]==T[j]) j++;
    ex[0]=j;
    for(int i=1;i<n;i++)
    {
        int p=ex[k]+k-1;
        int L=next[i-k];
        if(i+L<p+1) ex[i]=L;
        else
        {
            j=max(0,p-i+1);
            while(i+j<n&&j<m&&T[i+j]==P[j]) j++;
            ex[i]=j; k=i;
        }
    }
}
```

我的一些扩展kmp的总结

例题

参考上面的连接

- HDOJ 4333 Revolving Digits
- HDOJ 4300 Clairewd’s message
- HDOJ 4763 Theme Section
- UOJ #5. 【NOI2014】动物园
- Codeforces 432 D. Prefixes and Suffixes
- Codeforces 149 E. Martian Strings

7. SAM后缀自动机

后缀自动机的基本思想是:
将一个串的所有后缀加到一颗”字典树”里,由于一个字符串的所有后缀的空间复杂度是O(n^2)的.所以后缀自动机对这棵”字典树”进行了特殊的压缩.

参考资料:
陈立杰营员交流资料

后缀自动机很难理解,要注意掌握几SAM的几个性质.
后缀自动机与线性构造后缀树

SAM的一点性质:

1. 代码中 p->len 变量，它表示该状态能够接受的最长的字符串长度。
   该状态能够接受的最短的字符串长度。实际上等于该状态的 fa 指针指向的结点的 len + 1
   (p->len)-(p->fa->len)：表示该状态能够接受的不同的字符串数,不同的字符串之间是连续的,
   既:p 和 p->fa 之间 有最长的公共后缀长度 p->fa->len
2. num 表示这个状态在字符串中出现了多少次，该状态能够表示的所有字符串均出现过 num 次
3. 序列中第i个状态的子结点必定在它之后，父结点必定在它之前。
   既然p出现过，那么p->fa肯定出现过。因此对一个点+1就代表对整条fa链+1.
4. 从root到每一个接收态表示一个后缀,到每一个普通节点表示一个子串

我的实现:

```
const int CHAR=26,maxn=251000;

struct SAM_Node
```

```
{
    SAM_Node *fa,*next[CHAR];
    int len,id,pos;
    SAM_Node(){}
    SAM_Node(int _len)
    {
        fa=0; len=_len;
        memset(next,0,sizeof(next));
    }
};

SAM_Node SAM_node[maxn*2],*SAM_root,*SAM_last;
int SAM_size;

SAM_Node *newSAM_Node(int len)
{
    SAM_node[SAM_size]=SAM_Node(len);
    SAM_node[SAM_size].id=SAM_size;
    return &SAM_node[SAM_size++];
}

SAM_Node *newSAM_Node(SAM_Node *p)
{
    SAM_node[SAM_size]=*p;
    SAM_node[SAM_size].id=SAM_size;
    return &SAM_node[SAM_size++];
}

void SAM_init()
{
    SAM_size=0;
    SAM_root=SAM_last=newSAM_Node(0);
    SAM_node[0].pos=0;
}

void SAM_add(int x,int len)
{
    SAM_Node *p=SAM_last,*np=newSAM_Node(p->len+1);
    np->pos=len;SAM_last=np;
    for(;p&&!p->next[x];p=p->fa)
        p->next[x]=np;
    if(!p)
    {
        np->fa=SAM_root;
        return ;
    }
    SAM_Node *q=p->next[x];
    if(q->len==p->len+1)
    {
        np->fa=q;
        return ;
    }
    SAM_Node *nq=newSAM_Node(q);
    nq->len=p->len+1;
    q->fa=nq; np->fa=nq;
    for(;p&&p->next[x]==q;p=p->fa)
        p->next[x]=nq;
}

void SAM_build(char *s)
{
    SAM_init();
    int len=strlen(s);
    for(int i=0;i<len;i++)
        SAM_add(s[i]-'a',i+1);
}


/// !!!!!!!!!!!! 统计每个节点出现的次数

int c[maxn],num[maxn];
```

```
SAM_Node* top[maxn];

void Count(char str[],int len)
{
    for(int i=0;i<SAM_size;i++) c[SAM_node[i].len]++;
    for(int i=1;i<=len;i++) c[i]+=c[i-1];
    for(int i=0;i<SAM_size;i++) top[--c[SAM_node[i].len]]=&SAM_node[i];

    SAM_Node *p=SAM_root;
    for(;p->len!=len;p=p->next[str[p->len]-'a']) num[p->id]=1; num[p->id]=1;

    for(int i=SAM_size-1;i>=0;i--)
    {
        p=top[i];
        if(p->fa)
        {
            SAM_Node *q=p->fa; num[q->id]+=num[p->id];
        }
    }
}
```

例题

- Codeforces 235C. Cyclical Quest
- HDOJ 4416 Good Article Good sentence
- SPOJ 1811. Longest Common Substring LCS
- SPOJ 8222 NSUBSTR Substrings
- HDOJ 3518 Boring counting
- SPOJ LCS2 1812. Longest Common Substring II

7. 回文串自动机

去年(2014)新在比赛中出现的数据结构,资料不是很多

用一种类似AC自动机的方法构造出一个字符串的回文串树

Palindromic Tree——回文树【处理一类回文串问题的强力工具】

我的模版:

```
const int maxn=330000;
const int C=30;

int next[maxn][C];
int fail[maxn];
int cnt[maxn]; // 本质不同的回文串出现的次数(count后)
int num[maxn]; // 表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
int len[maxn]; // 节点i表示的回文串的长度
int s[maxn]; // 节点i存的字符
int last; // 新加一个字母后所形成的最长回文串表示的节点
int p; // 添加节点的个数 p-2为本质不同的回文串个数
int n; // 添加字符的个数

int newnode(int x)
{
    for(int i=0;i<C;i++) next[p][i]=0;
    cnt[p]=0; num[p]=0; len[p]=x;
    return p++;
}

void init()
{
    p=0;
    newnode(0); newnode(-1);
    last=0; n=0;
    s[0]=-1; fail[0]=1;
}

```

```
int get_fail(int x)
{
    while(s[n-len[x]-1]!=s[n]) x=fail[x];
    return x;
}

void add(int c)
{
    c-='a';
    s[++n]=c;
    int cur=get_fail(last);
    if(!next[cur][c])
    {
        int now=newnode(len[cur]+2);
        fail[now]=next[get_fail(fail[cur])][c];
        next[cur][c]=now;
        num[now]=num[fail[now]]+1;
    }
    last=next[cur][c];
    cnt[last]++;
}

void count()
{
    for(int i=p-1;i>=0;i--) cnt[fail[i]]+=cnt[i];
}
```


例题

- BZOJ 3676 Apio2014 回文串
- 2014 Xi'an Regional G The Problem to Slow Down You
  (回文串自动机+hash ~~有卡自然溢出hash的数据~~)

模版代码来自于我的ACM模版:我的ACM模版

大部分例题可以在我的博客:我的博客中找到题解.

a330616862 2016-07-23 12:20 #1楼 回复

```
void kmp(char* t,char* p,int* f)
{
int n=strlen(t),m=strlen(p);
getfail(p,f);
int j=0;
for(int i=0;i<n;i++)
{
while(j&&p[j]!=t[i]) j=f[j];
if(p[j]==t[i]) j++;
if(j==m)
```

```
{
///i-m+1
/// ans++;
j=f[j];
}
}
}
```

这个代码if(p[j]==t[i]) j++; 应该是if(p[i]==t[j]) j++;吧...

4