9.1 进制哈希

9.1.1哈希函数BKDRHash

|  |  |
| --- | --- |
| 例9-1. 【模板】字符串哈希 洛谷 P3370 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include<bits/stdc++.h>  using namespace std;  #define ull unsigned long long  ull a[10010];  char s[10010];  ull BKDRHash(char \*s){ //哈希函数  ull P = 131, H = 0; //P是进制，H是哈希值  int n = strlen(s);  for(int i=0;i<n;i++)  H = H \* P + s[i]-'a'+1; //H = H \* P + s[i]; //两种方法  //上面三行可以简写为一行：  // while(\*s) H = H\*P + (\*s++);  return H; //隐含了取模操作，等价于 H % 264  }  int main(){  int n; scanf("%d",&n);  for(int i=0;i<n;i++) {  scanf("%s",s);  a[i] = BKDRHash(s); //把字符串s的hash值记录在a[i]中  }  int ans = 0;  sort(a,a+n);  for(int i=0;i<n;i++) //统计有多少个不同的hash值  if(a[i]!=a[i+1])  ans++;  printf("%d",ans);  } | |

9.1.2进制哈希的应用

|  |  |
| --- | --- |
| 例9-2. ANT-Antisymmetry 洛谷P3501 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | //改写自：https://www.luogu.com.cn/blog/xrx/solution-p3501  #include<bits/stdc++.h>  using namespace std;  #define ull unsigned long long  const int N = 5e5+10;  char s[N],t[N];  int n,PP =131;  long long ans;  ull P[N],f[N],g[N]; //P：计算PP的i次方，f：s的前缀哈希值，g：t的前缀哈希值  void bin\_search(int x){ //用二分法寻找以s[x]为中心的回文串  int L=0,R=min(x,n-x);  while(L<R){  int mid = (L+R+1)>>1;  if((ull)(f[x]-f[x-mid]\*P[mid])==(ull)(g[x+1]-g[x+1+mid]\*P[mid])) L = mid;  else R = mid-1;  }  ans += L; //最长回文串的长度是L，它内部有L个回文串  }  int main(){  scanf("%d",&n); scanf("%s",s+1);  P[0] = 1;  for(int i=1;i<=n;i++) s[i]=='1'? t[i]='0':t[i]='1'; //t是反串  for(int i=1;i<=n;i++) P[i] = P[i-1]\*PP; //P[i]=PP的i次方  for(int i=1;i<=n;i++) f[i] = f[i-1]\*PP + s[i]; //求s所有前缀的哈希值  for(int i=n;i>=1;i--) g[i] = g[i+1]\*PP + t[i); //求t所有前缀的哈希值  for(int i=1;i<n;i++) bin\_search(i);  printf("%lld\n",ans);  return 0;  } | |

**2.**

|  |  |
| --- | --- |
| 例9-3. 最短循环节问题 洛谷P4391 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include<bits/stdc++.h>  using namespace std;  #define ull unsigned long long  const int N = 1e6+100;  ull PP = 131;  char s[N];  ull P[N],H[N],n;  ull get\_hash(ull L,ull R){return H[R]-H[L-1]\*P[R-L+1];} //区间[L,R]的哈希值  int main(){  P[0]=1;  for(int i=1; i<=N-1; i++) P[i] = P[i-1]\*PP; //预处理PP的i次方  cin>>n; scanf("%s",s+1); //s[0]不用  for(ull i=1; i<=n; i++) H[i] = H[i-1]\*PP + s[i]; //预处理所有前缀的hash值  for(ull i=1; i<=n; i++) {  ull flag = 1;  ull last = get\_hash(1,i); //暴力验证区间[1,i]是否为循环节  for(int j=1; (j+1)\*i<=n; j++) { //一个区间一个区间地判断  if(get\_hash(j\*i+1,(j+1)\*i)!=last){ //这一区间是否与第一区间相同  flag=0;  break;  }  }  if(n\*I != 0){ //末位多了一小截，单独判断  ull len = n%i;  if(get\_hash(1,len) != get\_hash(n-len+1,n)) flag=0;  }  if(flag){ printf("%d\n",(int)i); break;} //找到了答案，输出然后退出  }  return 0;  } | |

9.2 Manacher

9.2.3 模板代码

|  |
| --- |
| 例9-4. manacher算法 洛谷P3805 |

下面是代码。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | #include<bits/stdc++.h>  using namespace std;  const int N=11000002;  int n,P[N<<1]; //P[i]: 以s[i]为中心的回文半径  char a[N],s[N<<1];  void change(){ //变换  n = strlen(a);  int k = 0; s[k++]='$'; s[k++]='#';  for(int i=0;i<n;i++){s[k++]=a[i]; s[k++]='#';} //在每个字符后面插一个#  s[k++]='&'; //首尾不一样，保证第18行的while循环不越界  n = k;  }  void manacher(){  int R = 0, C;  for(int i=1;i<n;i++){  if(i < R) P[i] = min(P[(C<<1)-i],P[C]+C-i); //合并处理两种情况  else P[i] = 1;  while(s[i+P[i]] == s[i-P[i]]) P[i]++; //暴力：中心扩展法  if(P[i]+i > R){  R = P[i]+i; //更新最大R  C = i;  }  }  }  int main(){  scanf("%s",a); change();  manacher();  int ans=1;  for(int i=0;i<n;i++) ans=max(ans,P[i]);  printf("%d",ans-1);  return 0;  } |

9.3.2 模板代码

用下面的例题给出字典树的代码。

|  |
| --- |
| 例9-5. 于是他错误的点名开始了 洛谷 P2580 |

这一题有两种方法：STL map、字典树。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | //洛谷 P2580的map实现  #include <bits/stdc++.h>  using namespace std;  int main(){  map<string,int>student; string name;  int n; cin>>n;  while(n--){ cin>>name; student[name]=1; } //直接把名字当成下标处理  int m; cin>>m;  while(m--){  cin>>name;  if(student[name]==1){ puts("OK"); student[name]=2;}  else if(student[name]==2) puts("REPEAT");  else puts("WRONG");  }  return 0;  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | #include <bits/stdc++.h>  using namespace std;  const int N = 800000;  struct node{  bool repeat; //这个前缀是否重复  int son[26]; //26个字母  int num; //这个前缀出现的次数  }t[N]; //trie  int cnt = 1; //当前新分配的存储位置。把cnt=0留给根结点  void Insert(char \*s){  int now = 0;  for(int i=0;s[i];i++){  int ch=s[i]-'a';  if(t[now].son[ch]==0) //如果这个字符还没有存过  t[now].son[ch] = cnt++; //把cnt位置分配给这个字符  now = t[now].son[ch]; //沿着字典树往下走  t[now].num++; //统计这个前缀出现过多少次  }  }  int Find(char \*s){  int now = 0;  for(int i=0;s[i];i++){  int ch = s[i]-'a';  if(t[now].son[ch]==0) return 3; //第一个字符就找不到  now = t[now].son[ch];  }  if(t[now].num == 0) return 3; //这个前缀没有出现过  if(t[now].repeat == false){ //第一次被点名  t[now].repeat = true;  return 1;  }  return 2;  // return t[p].num; //若有需要，返回以s为前缀的单词的数量  }  int main(){  char s[51];  int n;cin>>n;  while(n--){ scanf("%s",s); Insert(s); }  int m; scanf("%d",&m);  while(m--) {  scanf("%s",s);  int r = Find(s);  if(r == 1) puts("OK");  if(r == 2) puts("REPEAT");  if(r == 3) puts("WRONG");  }  return 0;  } |

9.4.2 模板代码

|  |  |
| --- | --- |
| 例9-6. Victor and String hdu 5421  输出：对操作3和操作4，输出结果。 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | //改写自 https://blog.csdn.net/qq\_40858062/article/details/103956999  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  const int N = 3e5+8;  int s[N];  struct node{  int len,fail,son[26],siz;  void init(int \_len){  memset(son,0,sizeof(son));  fail = siz = 0;  len = \_len;  }  }tree[N];  ll num,last[2],ans,L,R; //L:在S的头部加字符；R:在S的尾部加字符  void init() { //初始化一个结点  last[0]=last[1]=0; //从0号点开始  ans=0; num=1;  L=1e5+8, R=1e5+7;  tree[0].init(0); //小技巧，置0号点len = 0  memset(s,-1,sizeof(s));  tree[1].init(-1); //小技巧，置1号点len = -1  tree[0].fail=1; //0指向1，1不必指向0  }  int getfail(int p,int d){ //后缀链跳跃。复杂度可以看成O(1)  if(d) //新字符在尾部  while(s[R-tree[p].len-1] != s[R])  p = tree[p].fail;  else //新字符在头部  while(s[L+tree[p].len+1] != s[L])  p = tree[p].fail;  return p; //返回结点p  }  void Insert(int x,int d){ //往回文树上插入新结点，这个结点表示一个新回文串  if(d) s[++R] = x; //新字符x插到S的尾部  else s[--L] = x; //新字符x插到S的头部  int father = getfail(last[d],d); //插到一个后缀的子结点上  int now = tree[father].son[x];  if(!now){ //字典树上还没有这个字符，新建一个  now = ++num;  tree[now].init(tree[father].len+2);  tree[now].fail = tree[getfail(tree[father].fail,d)].son[x];  tree[now].siz = tree[tree[now].fail].siz+1;  tree[father].son[x] = now;  }  last[d] = now;  if(R-L+1 == tree[now].len) last[d^1]=now;  ans += tree[now].siz;  //char ch = x +'a'; //在这里打印回文树，帮助理解  //cout<<" fa="<<father<<",me="<<now<<",char="<<ch;  //cout<<",fail="<<tree[now].fail<<",len="<<tree[now].len<<endl;  }  int main(){  int op,n;  while(scanf("%d",&n)!=EOF){  init();  while(n--) {  char c; scanf("%d",&op);  if(op==1) scanf(" %c",&c), Insert(c-'a',0);  if(op==2) scanf(" %c",&c), Insert(c-'a',1);  if(op==3) printf("%lld\n",num-1);  if(op==4) printf("%lld\n",ans);  }  }  return 0;  } | |

9.5.3 模板代码和例题

|  |  |
| --- | --- |
| 例9-7. **剪花布条** hdu 2087 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | #include<bits/stdc++.h>  using namespace std;  const int N = 1005;  char str[N], pattern[N];  int Next[N];  int cnt;  void getNext(char \*p, int plen){ //计算Next[1]~Next[plen]  Next[0]=0; Next[1]=0;  for(int i=1; i < plen; i++){ //把i的增加看成后缀的逐步扩展  int j = Next[i]; //j的后移：j指向前缀阴影w的后一个字符  while(j && p[i] != p[j]) //阴影的后一个字符不相同  j = Next[j]; //更新j  if(p[i]==p[j]) Next[i+1] = j+1;  else Next[i+1] = 0;  }  }  void kmp(char \*s, char \*p) { //在s中找p  int last = -1;  int slen=strlen(s), plen=strlen(p);  getNext(p, plen); //预计算Next[]数组  int j=0;  for(int i=0; i<slen; i++) { //匹配S和P的每个字符  while(j && s[i]!=p[j]) //失配了。注意j==0是情况(1)  j=Next[j]; //j滑动到Next[j]位置  if(s[i]==p[j]) j++; //当前位置的字符匹配，继续  if(j == plen) { //j到了P的末尾，找到了一个匹配  //这个匹配，在S中的起点是i+1-plen，末尾是i。如有需要可以打印：  // printf("at location=%d, %s\n", i+1-plen,&s[i+1-plen]);  //-------------------30--33行是本题相关  if( i-last >= plen) { //判断新的匹配和上一个匹配是否能分开  cnt++;  last=i; //last指向上一次匹配的末尾位置  }  //-------------------  }  }  }  int main(){  while(~scanf("%s", str)){ //读串  if(str[0] == '#') break;  scanf("%s", pattern); //读模式串  cnt = 0;  kmp(str, pattern);  printf("%d\n", cnt);  }  return 0;  } |

9.6.2 模板代码

|  |
| --- |
| 例9-11. Keywords Search hdu 2222 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | //改写自：https://blog.csdn.net/u011815404/article/details/88245190  #include <bits/stdc++.h>  using namespace std;  const int N=1000005;  struct node{  int son[26]; //26个字母  int end; //字符串结尾标记  int fail; //失配指针  }t[N]; //trie[]，字典树  int cnt; //当前新分配的存储位置  void Insert(char \*s){ //在字典树上插入单词s  int now = 0; //字典树上当前匹配到的结点，从root=0开始  for(int i=0;s[i];i++){ //逐一在字典树上查找s[]的每个字符  int ch=s[i]-'a';  if(t[now].son[ch]==0) //如果这个字符还没有存过  t[now].son[ch]=cnt++; //把cnt位置分配给这个字符  now = t[now].son[ch]; //沿着字典树往下走  }  t[now].end++; //end>0，它是字符串的结尾。end=0不是结尾  }  void getFail(){ //用BFS构建每个结点的fail指针  queue<int>q;  for(int i=0;i<26;i++) //把第一层入队，即root的子结点  if(t[0].son[i]) //这个位置有字符  q.push(t[0].son[i]);  while(!q.empty()){  int now = q.front(); //队首的fail指针已求得，下面求它孩子的fail指针  q.pop();  for(int i=0;i<26;i++){ //遍历now的所有孩子  if(t[now].son[i]){ //若这个位置有字符  t[t[now].son[i]].fail=t[t[now].fail].son[i];  //这个孩子的Fail=“父结点的Fail指针所指向的结点的与x同字符的子结点”  q.push(t[now].son[i]); //这个孩子入队，后面再处理它的孩子  }  else //若这个位置无字符  t[now].son[i]=t[t[now].fail].son[i];//**虚拟结点**，用于底层的Fail计算  }  }  }  int query(char \*s){ //在文本串s中找有多少个模式串P  int ans=0;  int now=0; //从root=0开始找  for(int i=0;s[i];i++){ //对文本串进行遍历  int ch = s[i]-'a';  now = t[now].son[ch];  int tmp = now;  while(tmp && t[tmp].end!=-1){ //利用fail指针找出所有匹配的模式串  ans+=t[tmp].end; //累加到答案中。若这不是模式串的结尾，end=0  t[tmp].end = -1; //以这个字符为结尾的模式串已经统计，后面不再统计  tmp = t[tmp].fail; //fail指针跳转  cout << "tmp="<<tmp<<" "<<t[tmp].son;  }  }  return ans;  }  char str[N];  int main(){  int k;  scanf("%d",&k);  while(k--){  memset(t,0,sizeof(t)); //清空，准备一个测试  cnt = 1; //把cnt=0留给root  int n; scanf("%d",&n);  while(n--){scanf("%s",str);Insert(str);} //输入模式串, 插入字典树中  getFail(); //计算字典树上每个结点的失配指针  scanf("%s",str); //输入文本串  printf("%d\n",query(str));  }  return 0;  } |

9.7.1 后缀树和后缀数组的概念

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include<bits/stdc++.h>  using namespace std;  int find(string S, string T, int \*sa){ //在S中查找子串T；sa是S的后缀数组  int i=0, j=S.length();  while(j-i > 1) {  int k = (i+j)/2; //二分法，操作O(logn)次  if(S.compare(sa[k], T.length(), T)<0) //匹配一次，复杂度是O(m)  i = k;  else j = k;  }  if(S.compare(sa[j], T.length(), T) == 0) //找到了，返回t在s中的位置  return sa[j];  if(S.compare(sa[i], T.length(), T) == 0)  return sa[i];  return -1; //没找到  }  int main(){  string s="vamamadn", t="ad"; //母串和子串  int sa[]={5, 3, 1, 6, 4, 2, 7, 0}; //sa是s的后缀数组，假设已经得到了  int location = find(s, t, sa);  cout << location <<":"<<&s[location]<< endl<<endl; //打印t在s中的位置  } |

9.7.2 倍增法求后缀数组

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | //计算后缀数组sa[]模板。参考《挑战程序设计竞赛》秋叶拓哉，379页，“4.7.3后缀数组”  #include<bits/stdc++.h>  using namespace std;  const int N = 200005; // 字符串的长度。  char s[N]; //输入字符串  int sa[N], rk[N], tmp[N+1];  int n, k;  bool comp\_sa(int i, int j){ //组合数有两部分，高位是rk[i]，低位是rk[i+k]。  if(rk[i] != rk[j]) //先比较高位：rk[i]和rk[j]  return rk[i] < rk[j];  else{ //高位相等，再比较低位的rk[i+k]和rk[j+k]  int ri = i+k <= n? rk[i+k] : -1;  int rj = j+k <= n? rk[j+k] : -1;  return ri < rj;  }  }  void calc\_sa( ) { //计算字符串S的后缀数组  for(int i = 0; i <= n; i++) {  rk[i] = s[i]; //字符串的原始数值  sa[i] = i; //后缀数组，在每一步记录当前排序后的结果  }  for( k =1; k <= n; k = k\*2){ //开始一步步操作，每一步递增2倍进行组合  sort(sa, sa+n, comp\_sa); //排序，结果记录在sa[]中  tmp[sa[0]] = 0;  for(int i = 0; i < n; i++) //用sa[]倒推组合数，并记录在tmp[]中  tmp[sa[i+1]] = tmp[sa[i]] + (comp\_sa(sa[i],sa[i+1]) ? 1: 0);  for(int i = 0; i< n; i++) //把tmp[]拷贝给rk[]，用于下一步操作  rk[i] = tmp[i];  }  }  int main(){  while(scanf("%s",s)!=EOF){ //读字符串  n=strlen(s);  calc\_sa(); //求后缀数组sa[]  for(int i=0;i<n;i++) //打印后缀数组  cout<<sa[i]<<" ";  }  return 0;  } |

**5. 用基数排序求后缀数组**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | //改编自：《算法竞赛入门经典训练指南》，刘汝佳，陈锋，清华大学出版社，3.4.1节  //main()部分和上面用sort()的版本一样。  char s[N];  int sa[N],cnt[N],t1[N],t2[N],rk[N],height[N];  int n;  void calc\_sa() {  int m = 127;  int i,\*x=t1,\*y=t2;  for(i=0;i<m;i++) cnt[i]=0;  for(i=0;i<n;i++) cnt[x[i]=s[i]]++;  for(i=1;i<m;i++) cnt[i]+=cnt[i-1];  for(i=n-1;i>=0;i--) sa[--cnt[x[i]]]=i;  //sa[]: 从0到n-1  for(int k=1;k<=n;k=k\*2){ //利用对长度为k的排序的结果对长度为2\*k的排序  int p=0;  //2nd  for(i=n-k;i<n;i++) y[p++]=i;  for(i=0;i<n;i++) if(sa[i]>=k) y[p++]=sa[i]-k;  //1st  for(i=0;i<m;i++) cnt[i]=0;  for(i=0;i<n;i++) cnt[x[y[i]]]++;  for(i=1;i<m;i++) cnt[i]+=cnt[i-1];  for(i=n-1;i>=0;i--) sa[--cnt[x[y[i]]]]=y[i];  swap(x,y);  p=1; x[sa[0]]=0;  for(i=1;i<n;i++)  x[sa[i]] =  y[sa[i-1]]==y[sa[i]]&&y[sa[i-1]+k]==y[sa[i]+k]?p-1:p++;  if(p>=n) break;  m=p;  }  } |

**6. 高度数组height[]**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void getheight(int n){ //n是字符串长度。  int i, j, k=0;  for(i=0 ;i<n; i++) rk[sa[i]]=i; //用sa[]推导rk[]  for(i=0; i<n; i++) {  if(k) k--;  int j = sa[rk[i]-1];  while(s[i+k]==s[j+k]) k++;  height[rk[i]] = k;  }  } |

9.7.3 后缀数组的经典应用

|  |  |
| --- | --- |
| 例9-12. **Longest Common Substring** hdu 1403 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // 省略了calc\_sa( )、getheight()函数，已在上一节给出。  int main(){  int len1, ans;  while(scanf("%s", s)!=EOF) { //读第1个字符串  n = strlen(s);  len1 = n;  s[n] = '$'; //用'$'分隔2个字符串。  scanf("%s", s+n+1); //读第2个字符串，与第1个合并。  n = strlen(s);  calc\_sa(); //求后缀数组sa[]  getheight(n); //求height[]数组  ans = 0;  for(int i = 1; i < n; i++)  //找最大的height[i]，并且它对应的sa[i-1]和sa[i]分别属于前后2个字符串。  if(height[i]>ans &&  ((sa[i-1]<len1 &&sa[i]>=len1) || (sa[i-1]>=len1&&sa[i]<len1)))  ans = height[i];  printf("%d\n",ans);  }  return 0;  } |

9.8.4 模板代码

|  |
| --- |
| 例9-13. Reincarnation hdu 4622 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73 | #include <bits/stdc++.h>  using namespace std;  const int N = 2007;  int sz,last; //sz: 结点（状态）的编号；last：指向最后被添加的结点  struct node{ //用字典树存结点  int son[26]; //26个字母  int father;  int len; //这个等价类的最大子串长度  }t[N<<1]; //后缀自动机的状态数不超过2n个  void newNode(int length){ //建一个新结点。sz=0是根  t[++sz].len = length; //这个结点所表示的子串的长度  t[sz].father = -1; //它的父结点还未知  memset(t[sz].son,0,sizeof(t[sz].son));  }  void init(){  sz = -1; last = 0; //根是0，根指向-1，表示结束  newNode(0);  }  void Insert(int c){  newNode(t[last].len+1);  int p = last, cur = sz; //p:上个结点的位置，cur:新结点的位置  while(p!=-1 && !t[p].son[c])  t[p].son[c] = cur, p = t[p].father;  if(p==-1)  t[cur].father = 0;  else{  int q = t[p].son[c];  if(t[q].len == t[p].len + 1)  t[cur].father = q;  else{  newNode(t[p].len+1);  int nq = sz; //复制结点  memcpy(t[nq].son,t[q].son,sizeof(t[q].son));  t[nq].father = t[q].father;  t[cur].father = t[q].father = nq;  while(p>=0 && t[p].son[c] == q)  t[p].son[c] = nq, p = t[p].father;  }  }  last = cur;  /\* 打印后缀自动机的所有结点和边  for(int i=0;i<=sz;i++)  for(int j=0;j<26;j++)  if(t[i].son[j]) { //起点-(边上的字符)-终点  int start=i,end=t[i].son[j];  printf("%d-(%c)-%d ",start,j+'a',end);  printf(" father=%d len=%d\n",t[end].father,t[end].len);  }  cout<<endl;  \*/  }  char S[N];  int ans[N][N];  int main(){  int T; scanf("%d",&T);  while(T--){  scanf("%s",S);  int n = strlen(S);  for(int i = 0;i < n;i++){  init(); //每次重新求S[i,j]的后缀自动机  for(int j = i;j < n;j++){  Insert(S[j]- 'a');  ans[i][j] = ans[i][j-1] + t[last].len - t[t[last].father].len;  }  }  int Q, L, R; scanf("%d",&Q);  while(Q--){  scanf("%d%d",&L,&R);  printf("%d\n",ans[--L][--R]);  }  }  return 0;  } |