Atan2(y,x)

返回给定的 X 及 Y 坐标值的反正切值。反正切的角度值等于 X 轴与通过原点和给定坐标点 (Y坐标, X坐标) 的直线之间的夹角。结果以弧度表示并介于 -pi 到 pi 之间（不包括 -pi）。

　　语法

　　ATAN2（Y坐标,X坐标）

　　X坐标 点的 X 坐标。

　　Y坐标 点的 Y 坐标。

　　说明

? 结果为正表示从 X 轴逆时针旋转的角度，结果为负表示从 X 轴顺时针旋转的角度。

KMP

void preKmp(char \*src, int m, int \*rule){  
  int i=0,j=-1;  
  rule[0]=j;  
  while(i<m){  
    while(j!=-1 && src[j] != src[i])  
      j=rule[j];  
    i++,j++;  
    if(j>=m)  
      rule[i]=rule[j-1];  
    else  
      rule[i]=j;  
  }  
}  
void kmp(char \*src, int m, char \*dest, int n){  
  int rule[MAXN];  
  preKmp(src,m,rule);  
  int i=0, j=0;  
  while(i<n){  
    while(j!=-1 && src[j] != dest[i]){  
      j = rule[j];  
    }  
    i++,j++;  
    if(j==m){  
      //print ans  
      j=rule[j];  
    }  
  }  
}

半平面交

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<algorithm>

#include<cmath>

using namespace std;

#define MAXN 200

#define esp 1e-8

struct point{

double x,y;

point (double \_x=0,double \_y=0) : x(\_x) , y(\_y) {

}

void input(){

scanf("%lf %lf",&x,&y);

}

double len() const{

return sqrt(x\*x + y\*y);

}

};

struct line{

point s,e;

double ang;

};

int n;

line b[MAXN];

int sgn(double x){

return x > esp ? 1 : (x < -esp ? -1 : 0);

}

bool operator == (const point &p1,const point &p2){

return sgn(p1.x-p2.x) == 0 && sgn(p1.y-p2.y) == 0;

}

point operator - (const point &p1,const point &p2){

return point(p1.x-p2.x, p1.y-p2.y);

}

double operator \* (const point &p1,const point &p2){

return p1.x\*p2.y - p1.y\*p2.x;

}

bool para(line a,line b){

return sgn((a.e-a.s) \* (b.e-b.s)) == 0;

}

point JD(point a, point b, point c, point d){

double d1 = (b-a)\*(c-a),d2 = (b-a)\*(d-a),

d3 = (d-c)\*(a-c),d4 = (d-c)\*(b-c);

return point( (c.x\*d2 - d.x\*d1) / (d2-d1),

(c.y\*d2 - d.y\*d1) / (d2-d1));

}

point JD(line a, line b){

return JD(a.s, a.e, b.s, b.e);

}

void init()

{

int i,j,k;

point a[MAXN];

scanf("%d",&n);

for(i=0;i<n;i++)

a[i].input();

a[n] = a[0];

for(i=0;i<n;i++){

b[i].s = a[i+1];

b[i].e = a[i];

b[i].ang = atan2(a[i].x-a[i+1].x, a[i].y-a[i+1].y);

}

}

bool cmp(const line a, const line b){

if(sgn(a.ang - b.ang) != 0) return a.ang < b.ang;

else return sgn( (b.e - b.s) \* (a.e - b.s) ) > 0;

}

line d[MAXN];

//存的是逆时针的多边形

int solve()

{

int i,j,k,w;

int r,f,t;

t = 1;

sort(b,b+n,cmp);

for(i=1;i<n;i++)

if(sgn(b[i].ang - b[i-1].ang) != 0)

b[t++] = b[i];

if(t < 2) return 1;

r = 1; f = 0;

d[0] = b[0]; d[1] = b[1];

for(i=2;i<t;i++){

if( para(d[r],d[r-1]) || para(d[f+1],d[f]) ) return 0;

while(f < r && sgn( (b[i].e-b[i].s) \* (JD(d[r],d[r-1]) - b[i].s) )< 0 ) r--;

while(f < r && sgn( (b[i].e-b[i].s) \* (JD(d[f],d[f+1]) - b[i].s) )< 0 ) f++;

d[++r] = b[i];

}

while(f < r && sgn( (d[f].e-d[f].s) \* (JD(d[r],d[r-1]) - d[f].s) ) < 0) r--;

while(f < r && sgn( (d[r].e-d[r].s) \* (JD(d[f],d[f+1]) - d[r].s) ) < 0) f++;

return r > f+1;

}

int main()

{

int t,i;

scanf("%d",&t);

for(i=1;i<=t;i++){

init();

if(solve() ) printf("YES\n");

else printf("NO\n");

}

return 0;

}

没看清题，没按字典序输出（lexicographical）！！

注意LONG LONG！！！

少打了const！！！

没考虑到边境情况！！！

改掉的程序提交时忘了改回来！！！

忘了删掉调试代码！！！！！！！

位运算优先级比比较运算低（ ('&'的优先级) < ('!='的优先级） ！！！！

冷静~！淡定~！!!!!!!!

这道题我觉得主要是在构造AC自动机上，不是纯粹地构造。

当然，首先要正常地构造建立一棵Tire树（以所有病毒字符串为词典）。

接下来就是构造AC自动机了（先学习完AC自动机）。

使用BFS的形式构造。

注意点一：为每个结点构造失败指针的同时，检查其失败指针所指向的节点是否为危险节点，如果是的话也需要把当前节点标记为危险节点。

所谓的危险节点，就是指以当前节点为结尾的字符串是某个病毒串，或者包含了某个病毒串。如果在bfs的同时进行这项操作的时候就非常方便，在构造完当前失败指针的同时检查其所指向的节点是否为危险节点。

注意点二：如果某个节点的子节点不存在，则需要把它指向当前节点失败指针所指节点的子节点。

或者说这其实是属于AC自动机的一部分，当查找到某个结尾时返回到某个可查找的地方。同样地，在bfs的同时进行，如果子节点为空则把它指向当前节点失败指针所指节点的对应子节点。

这样的话，AC自动机的部分就完成了。接下来是DP部分。

dp[i][j]，表示到达查找串第i个字符时，对应于AC自动机的j节点所需要的最小改变数。

状态转移为dp[i][j] = min( dp[i][j], dp[i-1][j]+code(s[i-1])!=k))

读到i个字符时，对应于j状态（DP的过程要两重循环i和j），要转移到son[j]（j的子节点状态，在这里用k在[0,3]一重循环遍历所有可以转字符），如果第i个字符跟所要转移到的字符相同，则代价为0，因为不需要改变；否则代价为1，因为需要改变。

注意点三：如果当前状态不可达，则不需要对其进行后续运算。

注意点四：如果子节点为危险节点，则不可以进行转移，即不可以使用上面的状态方程。

最后，循环dp[len][j]，即在读完最后一个字符后检查所有状态的最终值，取其最小。如果均不可达，则返回-1。

到此，这道题就完成了。

=====================================================================.题意是说给了N个带病毒的DNA串( DNA串只有AGCT几种单元组成)...再给一长串DNA..问这长串DNA最少改动几个(就是改..不是删除或者添加..)能保证没有包含病毒字串..输出这个最小改动的次数..若怎么修改都带病毒子串...输出-1...

    所谓dp就是要构造一个无后效性的状态...并能从前往后推出所要的最优解....

    我参考了<http://blog.csdn.net/human_ck/article/details/6577142>的思路...很清晰...代码也简洁...

    我也来说明几个问题..

    问题一：为什么当一个节点是病毒节点(某个病毒串的末位)..要逆向随着Fail.要往下传...

    如果前面有一段是病毒...指向末位的下面那个点和他几个Father连起来的必定也是这段病毒..

    问题二：为什么在构造Fail指针时若其某个孩子没有要指向其Father->Fail的相同孩子...

    其实这相当于是往上传~~一直往上Fail传直到找到上面某个的孩子中有这个...若这样到根都没有..那么这个点就直接指向根了...指向根代表当前这 课Trie中没有符合的..这样的好处就是虽然构造的Trie中没有这个点...也没有给这个点分配空间...但是...我们给了这个点一个标记的类 似...使得后面的DP中找不到时能直接向上返回...如果不这么搞也行..就要在DP中多加些东西.. 例如

                                         1

                                          AAA

                                         GGG

这组数据可以说在后面通过遍历Trie做DP时一个点都进不去...如果前面加了关于Trie中没得该点的预处理...那么实际上就利用起来了那个没带字符信息的超级结点.....

1. //POJ 3691 - AC自动机DP初步
2. #include<iostream>
3. #include<queue>
4. using namespace std;
5. struct node
6. {
7. int s[4],fail;
8. bool word;
9. }a[1001];
10. int t,n,len,g,dp[1011][1011];
11. char s[1011];
12. queue<int> myqueue;
13. int turn(char c)
14. {
15. if (c=='A') return 0;
16. if (c=='G') return 1;
17. if (c=='C') return 2;
18. if (c=='T') return 3;
19. }
20. void Built\_Trie(int h,int k)
21. {
22. int p;
23. if (k==len)
24. {
25. a[h].word=true;
26. return;
27. }
28. p=turn(s[k]);
29. if (!a[h].s[p])
30. {
31. g++;
32. a[h].s[p]=g;
33. }
34. Built\_Trie(a[h].s[p],k+1);
35. }
36. int UpdataFail(int h,int g)
37. {
38. if (a[h].s[g]) return a[h].s[g];
39. else
40. if (!h) return h;
41. else return UpdataFail(a[h].fail,g);
42. }
43. void Built\_AC\_Automation()
44. {
45. int h,i;
46. while (!myqueue.empty()) myqueue.pop();
47. myqueue.push(0);
48. while (!myqueue.empty())
49. {
50. h=myqueue.front();
51. myqueue.pop();
52. for (i=0;i<4;i++)
53. if (a[h].s[i])
54. {
55. myqueue.push(a[h].s[i]);
56. if (h) a[a[h].s[i]].fail=UpdataFail(a[h].fail,i);
57. if (a[a[a[h].s[i]].fail].word) a[a[h].s[i]].word=true;  //!!!
58. }else
59. if (h) a[h].s[i]=a[a[h].fail].s[i];
60. }
61. return;
62. }
63. int GetAnswer()
64. {
65. int i,j,w,k,x,ans;
66. memset(dp,0x7F,sizeof(dp));
67. dp[0][0]=0;
68. for (int p=0;p<len;p++)
69. {
70. w=turn(s[p]);
71. for (j=0;j<=g;j++)
72. if (dp[j][p]<=1000)
73. for (i=0;i<4;i++)
74. if (!a[a[j].s[i]].word)
75. {
76. k=a[j].s[i];
77. if (i==w) x=dp[j][p]; else x=dp[j][p]+1;
78. if (dp[k][p+1]>x) dp[k][p+1]=x;
79. }
80. }
81. ans=1001;
82. for (i=0;i<=g;i++)
83. if (dp[i][len]<ans) ans=dp[i][len];
84. if (ans>1000) return -1;
85. else return ans;
86. }
87. int main()
88. {
89. t=0;
90. while (~scanf("%d",&n))
91. {
92. if (!n) break;
93. getchar();
94. memset(a,0,sizeof(a));
95. g=0;
96. while (n--)
97. {
98. scanf("%s",s);
99. len=strlen(s);
100. Built\_Trie(0,0);
101. }
102. Built\_AC\_Automation();
103. scanf("%s",s);
104. len=strlen(s);
105. printf("Case %d: %d\n",++t,GetAnswer());
106. }
107. return 0;
108. }