ICPC 代码模板-V2.0 字符串 + 计算几何

SDU(QD)　sdu1801

2019 年 11 月 14 日

目录

1 字符串

1.1 后缀数组

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <string>
#include <algorithm>

using namespace std;
int n;
const int N = 1000010;
int sa[N], x[N], c[N], y[N];
char s[N];
inline void SA()
{
    for (int i = 1; i <= n; ++i) x[i] = s[i];
    int m = 128;
    for (int i = 0; i <= m; ++i) c[i] = 0;
    for (int i = 1; i <= n; ++i) c[x[i]]++;
    for (int i = 1; i <= m; ++i) c[i] += c[i-1];
    for (int i = n; i; i--) sa[c[x[i]]--] = i;

    for (int k = 1, p; k <= n; k <<= 1){
        p = 0;
        for (int i = n; i > n - k; i--) y[++p] = i;
        for (int i = 1; i <= n; ++i)
            if(sa[i] > k) y[++p] = sa[i] - k;

        for (int i = 0; i <= m; ++i) c[i] = 0;
        for (int i = 1; i <= n; ++i) c[x[i]]++;
        for (int i = 1; i <= m; ++i) c[i] += c[i-1];
        for (int i = n; i; i--) sa[c[x[y[i]]]--] = y[i];

        p = y[sa[1]] = 1;
        for (int i = 2, a, b; i <= n; ++i){
            a = sa[i] + k > n? -1 : x[sa[i] + k];
            b = sa[i-1] + k > n?-1 : x[sa[i-1] + k];
            y[sa[i]] = (x[sa[i]] == x[sa[i-1]] && (a == b)? p : ++p);
        }
        swap(x, y);
        m = p;
        if (p == n)break;
    }
}
int height[N], rk[N], st[N][20];
inline void build_H(){
    int k = 0;
    for (int i = 1; i <= n; ++i) rk[sa[i]] = i;
    for (int i = 1; i <= n; ++i){
        if (rk[i] == 1) continue;
        if (k) --k;
        int j = sa[rk[i]-1];
```

```
51          while(j + k <= n && i + k <= n && s[i+k]==s[j+k]) ++k;
52          height[rk[i]] = k;
53      }
54  }
55  inline void build_lcp(){
56      for (int i = 1; i <= n; ++i)st[i][0] = height[i];
57      for (int i = 1; i <= 17; ++i){
58          for (int j = 1; j+(1<<i)-1 <= n; ++j)
59              st[j][i] = max(st[j][i-1], st[j+(1<<(i-1))][i-1]);
60      }
61  }
62  inline int query(int l, int r){
63      int k = log2(r - l + 1);
64      return max(st[l][k], st[r-(1<<k)+1][k]);
65  }
66  int main()
67  {
68      scanf("%s", s + 1);
69      n = strlen(s + 1);
70      SA();
71      build_H();
72      for (int i = 1; i <= n; ++i)printf("%d␣", sa[i]);
73  }
```

1.2 KMP

1.2.1 KMP

```
1  #include<cstdio>
2  #include<iostream>
3  #include<cstring>
4  #include<string>
5
6  using namespace std;
7  const int N = 10010;
8  char s[N];
9  int nxt[N];
10 //求 t 在 s 中是否出现，需要t的nxt数组
11 inline int check(char *s, int sl, char *t, int tl, int *nxt){
12     int l = 0;
13     for (int i = 1; i <= sl; ++i){
14         while(l && s[i] != t[l + 1]) l = nxt[l];
15         if (s[i] == t[l + 1]) l++;
16         if (l == tl) return 1;
17     }
18     return 0;
19 }
20 int main(){
21     scanf("%s", s + 1);
22     int p = 0, n = strlen(s + 1);
23     for (int i = 2; i <= n; ++i){
24         while(p && s[p + 1] != s[i]) p = nxt[p];
25         if (s[p + 1] == s[i]) p++;
26         nxt[i] = p;
```

```
27      }
28      for (int i = 1; i <= n; ++i)
29          printf("%d␣", nxt[i]);
30  }
```

1.2.2 exkmp

```
1   //有两个字符串a,b,要求输出b与a的每一个后缀的最长公共前缀
2   //nxt里为b的next数组，extend里是b与a的每一个后缀的最长公共前缀
3   #include<cstdio>
4   #include<iostream>
5   #include<cstring>
6   #include<string>
7   #include<algorithm>
8
9
10  #define N 1000010
11
12  using namespace std;
13
14  int q,nxt[N],extend[N];
15  string s,t;
16
17  void getnxt()
18  {
19      nxt[0]=t.size();//nxt[0]一定是T的长度
20      int now=0;
21      while(t[now]==t[1+now]&&now+1<(int)t.size())now++;//这就是从1开始暴力
22      nxt[1]=now;
23      int p0=1;
24      for(int i=2;i<(int)t.size();i++)
25      {
26          if(i+nxt[i-p0]<nxt[p0]+p0)nxt[i]=nxt[i-p0];//第一种情况
27          else
28          {//第二种情况
29              int now=nxt[p0]+p0-i;
30              now=max(now,0);//这里是为了防止i>p的情况
31              while(t[now]==t[i+now]&&i+now<(int)t.size())now++;//暴力
32              nxt[i]=now;
33              p0=i;//更新p0
34          }
35      }
36  }
37
38  void exkmp()
39  {
40      getnxt();
41      int now=0;
42      while(s[now]==t[now]&&now<min((int)s.size(),(int)t.size()))now++;//暴力
43      extend[0]=now;
44      int p0=0;
45      for(int i=1;i<(int)s.size();i++)
46      {
47          if(i+nxt[i-p0]<extend[p0]+p0)extend[i]=nxt[i-p0];//第一种情况
```

```
48          else
49          {//第二种情况
50              int now=extend[p0]+p0-i;
51              now=max(now,0);//这里是为了防止i>p的情况
52              while(t[now]==s[i+now]&&now<(int)t.size()&&now+i<(int)s.size())now++;//暴
                    力
53              extend[i]=now;
54              p0=i;//更新p0
55          }
56      }
57  }
58
59  int main()
60  {
61      cin>>s>>t;
62      exkmp();
63      int len=t.size();
64      for(int i=0;i<len;i++)printf("%d␣",nxt[i]);//输出nxt
65      puts("");
66      len=s.size();
67      for(int i=0;i<len;i++)printf("%d␣",extend[i]);//输出extend
68      return 0;
69  }
```

1.3 manacher

```
1  void Manacher(char s[],int len) {//原字符串和串长
2      int l = 0;
3      String[l++] = '$'; // 0下标存储为其他字符,防止越界
4      String[l++] = '#';
5      for (int i = 0; i < len; i++) {
6          String[l++] = s[i];
7          String[l++] = '#';
8      }
9      String[l] = 0; // 空字符
10     int MaxR = 0;
11     int flag = 0;
12     for (int i = 0; i < l; i++) {
13         cnt[i] = MaxR > i ? min(cnt[2 * flag - i], MaxR - i) : 1;//2*flag-i是i点关于
               flag的对称点
14         while (String[i + cnt[i]] == String[i - cnt[i]])
15             cnt[i]++;
16         if (i + cnt[i] > MaxR) {
17             MaxR = i + cnt[i];
18             flag = i;
19         }
20     }
21 }
22 /*
23 * String: $ # a # b # a # a # b # a # 0
24 * cnt:    1 1 2 1 4 1 2 7 2 1 4 1 2 1
25 */
```

1.4 PAM

```
1  //本质不同回文串数量 cnt[i] - 2;
2
3  #include <cstdio>
4  #include <iostream>
5  #include <cstring>
6  #include <string>
7  #include <algorithm>
8
9  using namespace std;
10 const int N = 300010;
11 struct Pam{
12     int len;
13     int nxt[30];
14     int fail;
15 }st[N];
16 int n, tot, last;
17 int S[N], cnt[N], num[N];
18 inline int newnode(int x){
19     memset(st[tot].nxt, 0, sizeof(st[tot].nxt));
20     cnt[tot] = 0, st[tot].len = x, num[tot] = 0;
21     return tot++;
22 }
23 inline void pam_init(){
24     tot = n = last = 0, newnode(0), newnode(-1);
25     S[0] = -1, st[0].fail = 1;
26 }
27 inline int get_fail(int x){
28     while(S[n-st[x].len-1] != S[n]) x = st[x].fail;
29     return x;
30 }
31 inline int insert(int c){
32     S[++n] = c;
33     int cur = get_fail(last);
34     if (!st[cur].nxt[c]){
35         int now = newnode(st[cur].len + 2);
36         st[now].fail = st[get_fail(st[cur].fail)].nxt[c];
37         st[cur].nxt[c] = now;
38         num[now] = num[st[now].fail] + 1;
39     }
40     last = st[cur].nxt[c];
41     cnt[last]++;
42     return num[last];
43 }
44 inline void count(){
45     for (int i = tot - 1; i >= 0; --i) cnt[st[i].fail] += cnt[i];
46 }
47 char s[N];
48 int main()
49 {
50     pam_init();
51     scanf("%s", s+1);
52     int len = strlen(s + 1);
```

```
53      long long ans = 0;
54      for (int i = 1; i <= len; ++i) insert(s[i] - 'a' + 1);
55      count();
56      for (int i = 1; i < tot; ++i) ans = max(ans, 1ll * st[i].len * cnt[i]);
57      printf("%lld\n", ans);
58  }
```

1.5 AC 自动机

1.5.1 求最大出现次数

```
1   //AC自动机求最大出现次数
2   //注意AC自动机上是可能出现自环的，所以在判环的时候要注意
3   //记得build
4   #include<cstdio>
5   #include<iostream>
6   #include<cstring>
7   #include<string>
8   #include<algorithm>
9   #include<queue>
10
11  using namespace std;
12  const int N = 156, L = 1e6 + 6;
13  namespace AC{
14      const int SZ = N * 80;
15      int tot, tr[SZ][26];
16      int fail[SZ], idx[SZ], val[SZ];
17      int cnt[N];
18      void init(){
19          memset(fail, 0, sizeof(fail));
20          memset(tr, 0, sizeof(tr));
21          memset(val, 0, sizeof(val));
22          memset(cnt, 0, sizeof(cnt));
23          memset(idx, 0, sizeof(idx));
24          tot = 0;
25      }
26      void insert(char *s, int id){
27          int u = 0;
28          for (int i = 1; s[i]; i++){
29              if (!tr[u][s[i] - 'a']) tr[u][s[i] - 'a'] = ++tot;
30              u = tr[u][s[i] - 'a'];
31          }
32          idx[u] = id;
33      }
34      queue<int> q;
35      void build(){
36          for (int i = 0; i < 26; ++i)
37              if (tr[0][i]) q.push(tr[0][i]);
38          while(!q.empty()){
39              int u = q.front();
40              q.pop();
41              for (int i = 0; i < 26; ++i){
42                  if (tr[u][i]) fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]);
43                  else tr[u][i] = tr[fail[u]][i];
```

```
44              }
45          }
46      }
47      inline int query(char *t){
48          int u = 0, res = 0;
49          for (int i = 1; t[i]; i++){
50              u = tr[u][t[i] - 'a'];
51              for (int j = u; j; j = fail[j]) val[j]++;
52          }
53          for (int i = 0; i <= tot; ++i)
54              if (idx[i]) res = max(res, val[i]), cnt[idx[i]] = val[i];
55          return res;
56      }
57  }
```

1.5.2 AC 自动机 (map) 版本

```
1   #include<cstdio>
2   #include<iostream>
3   #include<cstring>
4   #include<string>
5   #include<algorithm>
6   #include<queue>
7   #include<map>
8
9   using namespace std;
10  const int N = 50010, L = 1e6 + 6;
11  int pos[N];
12  namespace AC{
13      const int SZ = 100010;
14      int tot;
15      int vis[SZ], cnt[SZ], val[SZ];
16      int fail[SZ];
17      map<int, int> tr[SZ];
18      void insert(int *s, int l, int id){
19          int u = 0;
20          for (int i = 1; i <= l; i++){
21              if (!tr[u][s[i]]) tr[u][s[i]] = ++tot;
22              u = tr[u][s[i]];
23              // cout<<u<<' ';
24          }
25          pos[id] = u;
26          val[u]++;
27          // cout<<endl;
28      }
29      queue<int>q;
30      inline void build(){
31          for (auto i:tr[0]) q.push(i.second);
32          while(!q.empty()){
33              int u = q.front();
34              q.pop();
35              // cout<<u<<endl;
36              for (auto i:tr[u]){
37                  fail[i.second] = tr[fail[u]][i.first], q.push(i.second);
```

```
38              }
39          }
40      }
41      inline int query(int id, vector<int>t1){
42          int u = 0, res = 0;
43          int l = t1.size();
44          for (int i = 0; i < l; i++){
45              while(u && !tr[u][t1[i]]) u = fail[u];
46              u = tr[u][t1[i]];
47              for (int j = u; j; j = fail[j]){
48                  if (vis[j] != id){
49                      cnt[j]++, vis[j] = id;
50                      if (val[j]) res += val[j];
51                  }
52                  else break;
53              }
54          }
55          return res;
56      }
57  }
58  vector<int> q1[N], q2[N];
59  int t[N * 2], ans[N];
```

1.5.3 AC 自动机 + 矩阵快速幂

```
1   int n;
2   char s[N][100], t[L];
3   int main()
4   {
5       while(~scanf("%d", &n)){
6           if (n == 0) break;
7           AC::init();
8           for (int i = 1; i <= n; ++i) scanf("%s", s[i] + 1), AC::insert(s[i], i);
9           AC::build();
10          scanf("%s", t + 1);
11          int x = AC::query(t);
12          printf("%d\n", x);
13          for (int i = 1; i <= n; ++i)
14              if (AC::cnt[i] == x) printf("%s\n", s[i] + 1);
15      }
16      return 0;
17  }
18  //有很多题需要判断是否是终点，在求fail结点的时候可以顺便求一下
19  if (tr[u][i]) fail[tr[u][i]] = tr[fail[u]][i], idx[tr[u][i]] |= idx[fail[tr[u][i]]], q
        .push(tr[u][i]);
20  //矩阵快速幂求出现过给定字符串的的所有长度的串
21  typedef unsigned long long ull;
22  struct rec{
23      int sz;
24      ull a[N][N];
25      rec(){}
26      void init(int _sz){
27          sz = _sz;
28          memset(a, 0, sizeof(a));
```

```
29      }
30      rec operator* (const rec &b) const {
31          rec res;
32          res.init(sz);
33          for (int i = 0; i <= sz; ++i){
34              for (int j = 0; j <= sz; ++j){
35                  for (int k = 0; k <= sz; ++k)
36                      res.a[i][j] += a[i][k] * b.a[k][j];
37              }
38          }
39          return res;
40      }
41  }e, ans, res;
42  inline rec qpow(rec a, long long q){
43      rec res = e;
44      for (;q; q >>= 1, a = a * a){
45          if (q & 1) res = res * a;
46      }
47      return res;
48  }
49  //省略了init和insert
50  namespace AC{
51      queue<int>q;
52      void build(){
53          for (int i = 0; i < 26; ++i)
54              if (tr[0][i]) q.push(tr[0][i]);
55          while(!q.empty()){
56              int u = q.front();
57              q.pop();
58              for (int i = 0; i < 26; ++i){
59                  if (tr[u][i]) fail[tr[u][i]] = tr[fail[u]][i], idx[tr[u][i]] |= idx[
                        fail[tr[u][i]]], q.push(tr[u][i]);
60                  else tr[u][i] = tr[fail[u]][i];
61              }
62          }
63      }
64      inline rec chengzi(){
65          res.init(tot + 1);
66          for (int i = 0; i <= tot; ++i){
67              if (idx[i]) res.a[i][tot + 1] = 1, res.a[i][i] = 26;
68              else for (int j = 0; j < 26; ++j){
69                      res.a[i][tr[i][j]]++;
70                  }
71          }
72          res.a[tot + 1][tot + 1] = 1;
73          return res;
74      }
75  };
```

1.6 SAM

1.6.1 SAM

```
1  const int N = 40010;
```

```
2  struct state{
3      int len, link;
4      int next[180];
5  }st[N];
6  int sz, last;
7  int s[N], a[N], n;
8  int l[N], r[N], c[N], q[N];
9  void sam_init(){
10     memset(st, 0, sizeof(st));
11     sz = last = 1;
12 }
13 void sam_extend(int c){
14     int cur = ++sz;
15     st[cur].len = st[last].len + 1;
16     int p = last;
17     while(p && !st[p].next[c]){
18         st[p].next[c] = cur;
19         p = st[p].link;
20     }
21     if (!p){
22         st[cur].link = 1;
23     } else {
24         int q = st[p].next[c];
25         if (st[p].len + 1 == st[q].len) {
26             st[cur].link = q;
27         } else {
28             int clone = ++sz;
29             st[clone].len = st[p].len + 1;
30             memcpy(st[clone].next, st[q].next, sizeof(st[q].next));
31             st[clone].link = st[q].link;
32             while(p && st[p].next[c] == q) {
33                 st[p].next[c] = clone;
34                 p = st[p].link;
35             }
36             st[q].link = st[cur].link = clone;
37         }
38     }
39     last = cur;
40 }
```

1.6.2 求子串出现次数 (树上 dp)

```
1  // 求字典序第k大子串，T为0则表示不同位置的相同子串算作一个。T=1则表示不同位置的相同子
       串算作多个
2  // T=0时，跑一个后缀自动机上的dp，求不同子串个数，相当于求路径数目，然后然后在树上一
       个一个点的找就可以了。
3  // T=1时，现求一个子串出现次数，做法是按照后缀链接跑树形dp，字符串上的点赋值为1，克隆
       出的点初值为0，跑完dp后在按照第一种方法做一下就可以
4  #include <cstdio>
5  #include <iostream>
6  #include <cstring>
7  #include <string>
8  #include <algorithm>
9
```

```
10 using namespace std;
11 const int N = 1000010;
12 struct state{
13     int len, link;
14     int nxt[27];
15 }st[N];
16 int n, sz, T, last;
17 long long k;
18 char s[N];
19 int c[N], q[N];
20 long long dp[N], cnt[N];
21 inline void sam_init(){
22     sz = last = 1;
23 }
24 inline void sam_extend(int c){
25     int cur = ++sz;
26     st[cur].len = st[last].len + 1;
27     int p = last;
28     while(p && !st[p].nxt[c]){
29         st[p].nxt[c] = cur;
30         p = st[p].link;
31     }
32     if (!p) st[cur].link = 1;
33     else {
34         int q = st[p].nxt[c];
35         if (st[p].len + 1 == st[q].len){
36             st[cur].link = q;
37         } else {
38             int clone = ++sz;
39             st[clone].len = st[p].len + 1;
40             memcpy(st[clone].nxt, st[q].nxt, sizeof(st[q].nxt));
41             st[clone].link = st[q].link;
42             while(p && st[p].nxt[c] == q){
43                 st[p].nxt[c] = clone;
44                 p = st[p].link;
45             }
46             st[q].link = st[cur].link = clone;
47         }
48     }
49     last = cur;
50 }
51 inline void sam(){
52     long long now = 0;
53     int p = 1;
54     while(1){
55         for (int i = 1; i <= 26; ++i){
56             if (now + dp[st[p].nxt[i]] < k)
57                 now += dp[st[p].nxt[i]];
58             else {
59                 putchar('a' + i - 1);
60                 now += cnt[st[p].nxt[i]];
61                 if (now >= k) return;
62                 p = st[p].nxt[i];
63                 break;
```

```
64              }
65          }
66      }
67  }
68  int main()
69  {
70      scanf("%s", s + 1);
71      n = strlen(s + 1);
72      sam_init();
73      for (int i = 1; i <= n; ++i)
74          sam_extend(s[i] - 'a' + 1);
75      for (int i = 1; i <= sz; ++i) c[st[i].len]++;
76      for (int i = 1; i <= n; ++i) c[i] += c[i-1];
77      for (int i = sz; i; i--) q[c[st[i].len]--] = i;
78      scanf("%d%lld", &T, &k);
79      if (!T){
80          for (int i = 1; i <= sz; ++i) cnt[i] = 1;
81      } else {
82          int p = 1;
83          for (int i = 1; i <= n; ++i){
84              int x = s[i] - 'a' + 1;
85              p = st[p].nxt[x];
86              cnt[p] = 1;
87          }
88          for (int i = sz; i >= 1; --i){
89              p = q[i];
90              cnt[st[p].link] += cnt[p];
91          }
92      }
93      for (int i = sz; i >= 1; --i){
94          int p = q[i];
95          if (p != 1) dp[p] = cnt[p];
96          for (int j = 1; j <= 26; ++j)
97              dp[p] += dp[st[p].nxt[j]];
98      }
99      if (k > dp[1]) puts("-1");
100     else sam();
101 }
```

1.6.3 求最长不重叠子串，使用 lastpos 的性质

```
1  //求最长不重叠子串
2  const int N = 40010;
3  struct state{
4      int len, link;
5      int next[180];
6  }st[N];
7  int sz, last;
8  int s[N], a[N], n;
9  int l[N], r[N], c[N], q[N];
10 void sam_init(){
11     memset(l, 63, sizeof(l));
12     memset(r, 0, sizeof(r));
13     memset(c, 0, sizeof(c));
```

```
14      memset(q, 0, sizeof(q));
15      memset(st, 0, sizeof(st));
16      sz = last = 1;
17  }
18  void sam_extend(int c){
19      int cur = ++sz;
20      st[cur].len = st[last].len + 1;
21      l[cur] = r[cur] = st[cur].len;
22      int p = last;
23      while(p && !st[p].next[c]){
24          st[p].next[c] = cur;
25          p = st[p].link;
26      }
27      if (!p){
28          st[cur].link = 1;
29      } else {
30          int q = st[p].next[c];
31          if (st[p].len + 1 == st[q].len) {
32              st[cur].link = q;
33          } else {
34              int clone = ++sz;
35              st[clone].len = st[p].len + 1;
36              memcpy(st[clone].next, st[q].next, sizeof(st[q].next));
37              st[clone].link = st[q].link;
38              while(p && st[p].next[c] == q) {
39                  st[p].next[c] = clone;
40                  p = st[p].link;
41              }
42              st[q].link = st[cur].link = clone;
43          }
44      }
45      last = cur;
46  }
47  int main()
48  {
49      while(scanf("%d", &n)){
50          if (!n)break;
51          int ans = 0;
52          for (int i = 1; i <= n; ++i){
53              scanf("%d", &s[i]);
54              a[i] = s[i] - s[i - 1];
55          }
56          sam_init();
57          for (int i = 2; i <= n; ++i){
58              sam_extend(a[i] + 88);
59          }
60          for (int i = 1; i <= sz; ++i)c[st[i].len]++;
61          for (int i = 1; i < n; ++i) c[i] += c[i-1];
62          for (int i = sz; i; i--)q[c[st[i].len]--] = i;
63          for (int i = sz; i; i--){
64              int p = q[i];
65              l[st[p].link] = min(l[st[p].link], l[p]);
66              r[st[p].link] = max(r[st[p].link], r[p]);
67          }
```

```
68          for (int i = 1; i <= sz; ++i)
69              ans = max(ans, min(st[i].len, r[i] - l[i]));
70          if (ans < 4)puts("0");
71          else printf("%d\n", ans + 1);
72      }
73  }
```

2 计算几何

2.1 计算几何基础

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <string>
5  #include <algorithm>
6  #include <cmath>
7
8  using namespace std;
9  const double inf = 1e18;
10 const double eps = 1e-9;
11 inline int dcmp(double x){
12     if (fabs(x) <= eps) return 0;
13     return x < 0 ? -1 : 1;
14 }
15 struct point{
16     double x, y;
17     point(double x = 0, double y = 0) : x(x), y(y) {}
18     bool operator == (const point &t) const {return x == t.x && y == t.y;}
19     point operator + (const point &t) const {return point(x + t.x, y + t.y);}
20     point operator - (const point &t) const {return point(x - t.x, y - t.y);}
21     double operator * (const point &t) const {return x * t.x + y * t.y;}
22     point operator * (const double &k) {return point(k*x, k*y);}
23     double operator ^ (const point &t) const {return x * t.y - y * t.x;}
24     double dis() {return sqrt(x * x + y * y);}
25     double dis2() {return x * x + y * y;}
26 };
27 typedef point Vector;
28 //两向量夹角
29 double angle(Vector a, Vector b){
30     return acos(a * b / a.dis() / b.dis());
31 }
32 //向量旋转（逆时针 弧度）
33 Vector rotate(Vector A, double rad){
34     return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
35 }
36 //逆时针九十度的法向量
37 Vector Normal(Vector A){
38     double L = A.dis();
39     return Vector(-A.y/L, A.x/L);
40 }
41 struct Line{
42     point v, p;
```

```
43      Line(point v, point p):v(v), p(p) {}
44      //返回点P = V + (P - V) * T;
45      point Point(double t){
46          return v + (p - v) * t;
47      }
48  };
49  //两直线交点，需保证两直线不相交（叉积不为0）P to AB
50  // v ^ w != 0
51  point Inter(point p, Vector v, point q, Vector w){
52      Vector u = p - q;
53      double t = (w ^ u) / (v ^ w);
54      return p + v * t;
55  }
56  //点到直线距离，绝对值为有向距离
57  double DisToLine(point P, point A, point B){
58      Vector v1 = B - A, v2 = P - A;
59      return fabs((v1 ^ v2) / v1.dis());
60  }
61  //点到线段距离 P to AB
62  double DisToSeg(point P, point A, point B){
63      if (A == B) return (P - A).dis();
64      Vector v1 = B - A, v2 = P - A, v3 = P - B;
65      if (dcmp(v1 * v2) < 0)
66          return v2.dis();
67      if (dcmp(v1 * v3) > 0)
68          return v3.dis();
69      return DisToLine(P, A, B);
70  }
71  //P在AB上的投影点
72  point GetPro(point P, point A, point B){
73      Vector v = B - A;
74      return A + v * (v * (P - A) / (v * v));
75  }
76  //p是否在线段a1a2上
77  bool OnSeg(point p, point a1, point a2){
78      return dcmp((a1 - p) ^ (a2 - p)) == 0 && dcmp((a1 - p) * (a2 - p)) < 0;
79  }
80  //判断两线段相交
81  bool SegInter(point a1, point a2, point b1, point b2){
82      double c1 = (a2 - a1) ^ (b1 - a1), c2 = (a2 - a1) ^ (b2 - a1);
83      double c3 = (b2 - b1) ^ (a1 - b1), c4 = (b2 - b1) ^ (a2 - b1);
84      if (!dcmp(c1) || !dcmp(c2) || !dcmp(c3) || !dcmp(c4)){
85          bool f1 = OnSeg(b1, a1, a2);
86          bool f2 = OnSeg(b2, a1, a2);
87          bool f3 = OnSeg(a1, b1, b2);
88          bool f4 = OnSeg(a2, b1, b2);
89          bool f = (f1 | f2 | f3 | f4);
90          return f;
91      }
92      return (dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0);
93  }
94
95  // pick定理 2S = 2A + B - 2， A为多边形内部的格点数， B为边上的格点数， 边上的格点数
        : GCD(X, Y).
```

2.2 求凸包

```
typedef vector<point> polygon;
inline int cmp(const point &a, const point &b){
    return a.x == b.x ? a.y < b.y : a.x < b.x;
}
int st[N], ct;
polygon ConvexHull(point *P, int sz){
    polygon res;
    sort(P, P + sz, cmp), ct = 0;
    st[++ct] = 0, st[++ct] = 1;
    for (int i = 2; i < sz; ++i){
        while(ct > 1 && dcmp((P[st[ct]] - P[st[ct-1]]) ^ (P[i] - P[st[ct]])) <= 0)
            ct--;
        st[++ct] = i;
    }
    for (int i = 1; i <= ct; ++i) res.push_back(P[st[i]]);
    ct = 0, st[++ct] = sz - 1, st[++ct] = sz - 2;
    for (int i = sz - 3; i >= 0; --i){
        while(ct > 1 && dcmp((P[st[ct]] - P[st[ct-1]]) ^ (P[i] - P[st[ct]])) <= 0)
            ct--;
        st[++ct] = i;
    }
    for (int i = 2; i < ct; ++i) res.push_back(P[st[i]]);
    return res;
}
```

2.3 旋转卡壳

```
//旋转卡壳，平面最远点对
inline int caliper(polygon P) {
    int res = 0;
    if (n == 2){
        res = (p[0] - p[1]).dis2();
    } else if (n == 3) {
        res = max((p[0] - p[1]).dis2(), (p[0] - p[2]).dis2());
        res = max(res, (p[1] - p[2]).dis2());
    } else {
        P.push_back(point(0, 0));
        int m = P.size() - 1;
        int j = 2;
        for (int i = 0; i < m; ++i) {
            //万分注意这里一定是小于不能是小于等于，否则有可能被卡
            while (abs((P[i] - P[i + 1]) ^ (P[j] - P[i + 1])) < abs((P[i] - P[i + 1])
                ^ (P[j + 1] - P[i + 1])))
                j = (j + 1) % m;
            res = max(res, (P[i] - P[j]).dis2());
        }
    }
    return res;
}
```

2.4 最小圆覆盖

```cpp
#include <cstdio>
#include <iostream>
#include <string>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <vector>

using namespace std;
const double eps = 1e-9;
const double inf = 1e18;
const int N = 101000;
inline int dcmp(double x){
    if (fabs(x) <= eps) return 0;
    return x < 0 ? -1 : 1;
}
int n;
double r;

struct point{
    ...
}a[N], O;

//最小圆覆盖
bool in_circle(point a){return (a - O).dis() <= r? 1:0;}

inline int calc1(double a, double b, double c, double d, double e, double f){
    O.x = (b*f - d*e) / (b*c - a*d);
    O.y = (c*e - a*f) / (b*c - a*d);
}
inline void min_cover_circle(){
    random_shuffle(a+1, a+n+1);
    for (int i = 1; i <= n; ++i)
        if (!in_circle(a[i])){
            O = a[i], r = 0;
            for (int j = 1; j < i; ++j)
                if (!in_circle(a[j])){
                    O = (a[i] + a[j]) / 2.0;
                    r = (a[i] - O).dis();
                    for (int k = 1; k < j; ++k)
                        if (!in_circle(a[k])) {
                            calc1(a[i].x - a[j].x, a[i].y - a[j].y,a[i].x - a[k].x, a[
                                i].y - a[k].y,
                                (a[i].dis2() - a[j].dis2())/2.0,(a[i].dis2() - a[k].
                                    dis2())/2.0);
                            r = (a[i] - O).dis();
                        }
                }
        }
    printf("%.3f\n", r);
};
```

2.5 半平面交

```cpp
#include<cmath>
#include<cstring>
#include<vector>
#include<cstdio>
#include<iostream>
#include<algorithm>
using namespace std;
const double eps=1e-6;
const int maxn=2e5+10;
const double Pi=acos(-1.00);
inline int dcmp(double x)
{
    if(x>eps)return 1;
    return x<-eps?-1:0;
}
struct Vector
{
    double x,y;
    Vector(double X=0,double Y=0)
    {
        x=X,y=Y;
    }
    bool operator == (const Vector &b)const
    {
        return dcmp(x-b.x)==0&&dcmp(y-b.y)==0;
    }
    double angle()
    {
        return atan2(y,x);//求出极角
    }
};
typedef Vector Point;
Vector operator + (Vector a,Vector b){return Vector(a.x+b.x,a.y+b.y);}
Vector operator - (Vector a,Vector b){return Vector(a.x-b.x,a.y-b.y);}
Vector operator * (Vector a,double b){return Vector(a.x*b,a.y*b);}
Vector operator / (Vector a,double b){return Vector(a.x/b,a.y/b);}
struct Line
{
    Point s,t;
    double ang;
    Line(Point X=Vector(),Point Y=Vector())
    {
        s=X,t=Y,ang=(Y-X).angle();
    }
};
typedef Line Segment;
double dot(Vector a,Vector b)
{
    return a.x*b.x+a.y*b.y;
}
double cross(Vector a,Vector b)
{
    return a.x*b.y-a.y*b.x;
}
```

```
55  bool is_parallel(Line a,Line b)//判断a,b直线是否平行
56  {
57      return dcmp(cross(a.t-a.s,b.t-b.s))==0;
58  }
59  Point intersection(Line a,Line b)//求出a,b的交点
60  {
61      return a.s+(a.t-a.s)*(cross(b.t-b.s,a.s-b.s)/cross(a.t-a.s,b.t-b.s));
62  }
63  double area(Point *p,int n)//求出多边形的面积
64  {
65      double res=0;
66      p[n+1]=p[1];
67      for(int i=1;i<=n;i++)res+=cross(p[i],p[i+1]);
68      return fabs(res/2);
69  }
70  bool operator < (const Line &a,const Line &b)//极角排序，如果极角相同则，选择最靠左的
        直线
71  {
72      double r=a.ang-b.ang;
73      if(dcmp(r)!=0)return dcmp(r)==-1;
74      return dcmp(cross(a.t-a.s,b.t-a.s))==-1;
75  }
76  bool OnRight(Line a,Point b)//检查b是否在a直线的右边
77  {
78      return dcmp(cross(a.t-a.s,b-a.s))<0;
79  }
80  bool SI(Line *l,int n,Point *s,int &m)//增量法求半平面交
81  {
82      static Line que[maxn];
83      static Point que2[maxn];//两个双端队列
84      int head=0,tail=0;
85      sort(l+1,l+1+n);
86      que[0]=l[1];
87      for(int i=2;i<=n;i++)
88          if(dcmp(l[i].ang-l[i-1].ang)!=0)//极角相等的直线，取一个
89          {
90              if(head<tail&&(is_parallel(que[head],que[head+1])||is_parallel(que[tail],
                    que[tail-1])))return false;//如果两个直线共线，但是极角不同，则没有半平
                    面交
91              while(head<tail&&OnRight(l[i],que2[tail-1]))tail--;//如果在直线右边，删除
                    点
92              while(head<tail&&OnRight(l[i],que2[head]))head++;
93              que[++tail]=l[i];
94              if(head<tail)que2[tail-1]=intersection(que[tail],que[tail-1]);//加入新点
95          }
96      while(head<tail&&OnRight(que[head],que2[tail-1]))tail--;//删去多余点
97      while(head<tail&&OnRight(que[tail],que2[head]))head++;
98      if(tail-head<=1)return false;//只有一个点或零个点，没有半平面交
99      que2[tail]=intersection(que[head],que[tail]);//加入最后一条边，和第一条边的交点
100     m=0;
101     for(int i=head;i<=tail;i++)s[++m]=que2[i];
102     return true;
103 }
104 const double lim=10000;
```

```
105 int n,m;
106 Point p[maxn];
107 Line l[maxn];
108 double solve()
109 {
110     Point a=Point(0,0);//加入最大限制，防止半平面交无限大
111     Point b=Point(lim,0);
112     Point c=Point(lim,lim);
113     Point d=Point(0,lim);
114     l[++n]=Line(a,b);
115     l[++n]=Line(b,c);
116     l[++n]=Line(c,d);
117     l[++n]=Line(d,a);
118     if(!SI(l,n,p,m))return 0;
119     return area(p,m);
120 }
121 int main()
122 {
123     scanf("%d",&n);
124     for(int i=1;i<=n;i++)
125     {
126         Point a,b;
127         scanf("%lf%lf%lf%lf",&a.x,&a.y,&b.x,&b.y);
128         l[i]=Line(a,b);
129     }
130     printf("%.1f\n",solve());
131 }
```