# 计算几何

1. **const** **double** eps=1e-10;
2. **const** **double** Pi=acos(-1.);
3. **int** dcmp(**const** **double** &x) { **if**(fabs(x)<eps) **return** 0; **return** x<0?-1:1; }
4. **typedef** **struct** Poi Vec;
5. **struct** Poi
6. {
7. **double** x,y;
8. Poi(**const** **double** &a=0,**const** **double** &b=0):x(a),y(b) { }
9. Poi operator+(**const** Poi &P)**const** { **return** Poi(x+P.x,y+P.y); }
10. Poi operator-(**const** Poi &P)**const** { **return** Poi(x-P.x,y-P.y); }
11. Poi operator\*(**const** **double** &P)**const** { **return** Poi(x\*P,y\*P); }
12. Poi operator/(**const** **double** &P)**const** { **return** Poi(x/P,y/P); }
13. **bool** operator<(**const** Poi &P)**const** { **return** !dcmp(x-P.x)?dcmp(y-P.y)<=0:dcmp(x-P.x)<=0; }
14. };
15. **struct** Line
16. {
17. Poi Ps; Vec Dir;
18. **double** ang;
19. Line(**const** Poi &ps=Poi(),**const** Vec &dir=Vec()):Ps(ps),Dir(dir)
20. { ang=atan2(Dir.y,Dir.x); }
21. **bool** operator<(**const** Line &L1)**const** { **return** ang<L1.ang; }
22. };
23. **struct** Circle
24. {
25. Poi Pc;
26. **double** r;
27. Circle(**const** Poi &P=Poi(),**const** **double** &R=0):Pc(P),r(R) { }
28. Poi loc(**double** ang,**double** offset) { **return** Poi(Pc.x+cos(ang)\*(r+offset),Pc.y+sin(ang)\*(r+offset)); }
29. };
30. **double** dOt(**const** Vec &V1,**const** Vec &V2) { **return** V1.x\*V2.x+V1.y\*V2.y; }
31. **double** cRoss(**const** Vec &V1,**const** Vec &V2) { **return** V1.x\*V2.y-V1.y\*V2.x; }
32. **double** lEnth(**const** Vec V) { **return** sqrt(dOt(V,V)); }
33. Vec rOtate(**const** Vec &V,**const** **double** &P)
34. { **double** cOs=cos(P),sIn=sin(P); **return** Vec(V.x\*cOs-V.y\*sIn,V.x\*sIn+V.y\*cOs); }
35. **double** aNgle(**const** Vec &V1,**const** Vec &V2) { **return** acos(dOt(V1,V2)/lEnth(V1)/lEnth(V2)); }
36. Poi iNtersect(**const** Poi &P1,**const** Vec &V1,**const** Poi &P2,**const** Vec &V2)
37. { **return** P1+V1\*(cRoss(V2,P1-P2)/cRoss(V1,V2)); }
38. Poi iNtersect(**const** Line &L1,**const** Line &L2)
39. { **return** iNtersect(L1.Ps,L1.Dir,L2.Ps,L2.Dir); }
40. **int** HPI(Line \*L,**int** N,Poi \*Pol)
41. {
42. **int** l,r,m;
43. Poi IP[N+50]; Line q[N+50];
44. sort(L,L+N);//按极角排序
45. q[l=r=0]=L[0];
46. **for**(**int** i=1;i<N;i++)
47. {
48. **while**(l<r&&dcmp(cRoss(L[i].Dir,IP[r-1]-L[i].Ps))<=0) r--; //新加入的直线可能是尾部的一些交点失效
49. **while**(l<r&&dcmp(cRoss(L[i].Dir,IP[l]-L[i].Ps))<=0) l++; //首部
50. q[++r]=L[i]; //加入
51. **if**(!dcmp(cRoss(q[r].Dir,q[r-1].Dir)))
52. { //对于平行直线要取靠左的
53. r--;
54. **if**(dcmp(cRoss(q[r+1].Dir,q[r].Ps-q[r+1].Ps))<0) q[r]=q[r+1];
55. }
56. **if**(l<r) IP[r-1]=iNtersect(q[r-1],q[r]); //如果队列中有至少两条线，则取交点
57. }
58. **while**(l<r&&dcmp(cRoss(q[l].Dir,IP[r-1]-q[l].Ps))<=0) r--; //后面一些交点可能实际上是无用的
59. **if**(r-l<2) **return** 0; //如果只有不到两条线，则失败了
60. IP[r]=iNtersect(q[l],q[r]); //将最后一条线和第一条线交起来
61. **for**(**int** i=l;i<=r;i++) Pol[m++]=IP[i];
62. **return** m;
63. }
64. **int** Andrew(Poi \*A,**int** N,Poi \*B)
65. {
66. **int** m=0;
67. sort(A,A+N);
68. **for**(**int** i=0;i<N;B[m++]=A[i++])
69. **while**(m>1&&cRoss(B[m-1]-B[m-2],A[i]-B[m-2])<=0) m--;
70. **for**(**int** i=N-2,k=m;i>=0;B[m++]=A[i--])
71. **while**(m>k&&cRoss(B[m-1]-B[m-2],A[i]-B[m-2])<=0) m--;
72. **if**(N>1) m--;
73. B[m]=B[0];
74. **return** m;
75. }
76. **int** ShamoS(**int** N,Poi \*P)
77. {
78. **int** r=1,ans=0,d;
79. **if**(N==1) **return** 0;
80. **else** **if**(N==2) **return** sQr(P[0]-P[1]);
81. **for**(**int** l=0;l<N;r=(r+1)%N)
82. {
83. d=dcmp(cRoss(P[l+1]-P[l],P[r+1]-P[r]));
84. **if**(d>0) **continue**;
85. ans=max(ans,(**int**)sQr(P[r--]-P[l++]));
86. **if**(!d) ans=max(ans,(**int**)sQr(P[r+1]-P[l]));
87. }
88. **return** ans;
89. }
90. **bool** onseg(**const** Poi &A,**const** Poi &B,**const** Poi &P)
91. {
92. Vec PA=A-P,PB=B-P;
93. **if**(!dcmp(cRoss(PA,PB))&&dcmp(dOt(PA,PB))<=0) **return** **true**;
94. **return** **false**;
95. }
96. **bool** segI(**const** Poi &A,**const** Poi &B,**const** Poi &C,**const** Poi &D)
97. {
98. **if**(onseg(A,B,C) ||onseg(A,B,D) ||onseg(C,D,A) ||onseg(C,D,B)) **return** **true**;
99. **if**((dcmp(cRoss(B-C,D-C))^dcmp(cRoss(A-C,D-C)))==-2
100. &&(dcmp(cRoss(D-B,A-B))^dcmp(cRoss(C-B,A-B)))==-2) **return** **true**;
101. **return** **false**;
102. }

# 表达式树

1. **int** nc,lch[100050],rch[100050],op[100050];
2. **int** build\_tree(**char** \*s,**int** x,**int** y)
3. {
4. **int** c1=-1,c2=-1,p=0,u;
5. **if**(y-x==1)
6. {
7. u=++nc;
8. lch[u]=rch[u]=0;
9. op[u]=s[x];
10. **return** u;
11. }
12. **for**(**int** i=x;i<y;i++)
13. {
14. **switch**(s[i])
15. {
16. **case** '(':p++;**break**;
17. **case** ')':p--;**break**;
18. **case** '+':**case** '-':**if**(!p)c1=i;**break**;
19. **case** '\*':**case** '/':**if**(!p)c2=i;**break**;
20. }
21. }
22. **if**(c1<0)c1=c2;
23. **if**(c1<0)**return** build\_tree(s,x+1,y-1);
24. u=++nc;
25. lch[u]=build\_tree(s,x,c1);
26. rch[u]=build\_tree(s,c1+1,y);
27. op[u]=s[c1];
28. **return** u;
29. }

# Dinic

1. **const** **int** MAXN=3050;
2. **const** **int** MAXM=4000050;
3. **const** **int** INF=1000000050;
4. **int** fir[MAXN],cur[MAXN],lev[MAXN],end[MAXM],next[MAXM],f[MAXM],ed,S,T;
5. **void** addedge(**int** u,**int** v,**int** cap)
6. {
7. end[++ed]=v;
8. next[ed]=fir[u];
9. fir[u]=ed;
10. f[ed]=cap;
11. end[++ed]=u;
12. next[ed]=fir[v];
13. fir[v]=ed;
14. f[ed]=0;
15. **return**;
16. }
17. **bool** BFS()
18. {
19. **int** u;
20. memset(lev,-1,**sizeof**(lev));
21. queue<**int**>q;
22. lev[S]=0;
23. q.push(S);
24. **while**(q.size())
25. {
26. u=q.front(); q.pop();
27. **for**(**int** i=fir[u];i;i=next[i]) **if**(f[i]&&lev[end[i]]==-1)
28. {
29. lev[end[i]]=lev[u]+1;
30. q.push(end[i]);
31. }
32. }
33. memcpy(cur,fir,**sizeof** fir);
34. **return** lev[T]!=-1;
35. }
36. **int** DFS(**int** u,**int** maxf)
37. {
38. **if**(u==T||!maxf) **return** maxf;
39. **int** cnt=0;
40. **for**(**int** &i=cur[u],tem;i;i=next[i]) **if**(f[i]&&lev[end[i]]==lev[u]+1)
41. {
42. tem=DFS(end[i],min(maxf,f[i]));
43. maxf-=tem;
44. f[i]-=tem;
45. f[i^1]+=tem;
46. cnt+=tem;
47. **if**(!maxf) **break**;
48. }
49. **if**(!cnt) lev[u]=-1;
50. **return** cnt;
51. }
52. **int** Dinic()
53. {
54. **int** ans=0;
55. **while**(BFS())
56. ans+=DFS(S,2147483647);
57. **return** ans;
58. }
59. **void** init(**int** SS,**int** TT)
60. {
61. memset(fir,0,**sizeof**(fir));
62. ed=1;
63. S=SS; T=TT;
64. **return**;
65. }

# 点分治

1. **const** **int** MAXN=100050;
2. **const** **int** MAXL=1050;
3. **int** fir[MAXN],end[MAXN<<1],next[MAXN<<1],len[MAXN<<1],ed=0;
4. **int** size[MAXN],hson[MAXN],dis[MAXN],q[MAXN],root;
5. **bool** vis[MAXN];
6. **void** addedge(**int** u,**int** v,**int** lenth)
7. {
8. end[++ed]=v;
9. next[ed]=fir[u];
10. fir[u]=ed;
11. len[ed]=lenth;
12. end[++ed]=u;
13. next[ed]=fir[v];
14. fir[v]=ed;
15. len[ed]=lenth;
16. **return**;
17. }
18. **void** init()
19. {
20. memset(fir,ed=0,**sizeof**(fir));
21. memset(vis,**false**,**sizeof**(vis));
22. **return**;
23. }
24. **void** read1n(**int** &x)
25. {
26. **char** ch=getchar();
27. **while**(ch<'0'||'9'<ch) ch=getchar();
28. **for**(x=0;'0'<=ch&&ch<='9';ch=getchar())
29. x=(x<<1)+(x<<3)+ch-'0';
30. **return**;
31. }
32. **void** get\_root(**int** u,**int** fa,**int** all)
33. {
34. size[u]=1; hson[u]=0;
35. **for**(**int** i=fir[u];i;i=next[i]) **if**(end[i]!=fa&&!vis[end[i]])
36. {
37. get\_root(end[i],u,all);
38. size[u]+=size[end[i]];
39. hson[u]=max(hson[u],size[end[i]]);
40. }
41. hson[u]=max(hson[u],all-size[u]);
42. **if**(root==-1||hson[u]<hson[root])
43. root=u;
44. **return**;
45. }
46. **void** DFS(**int** u,**int** fa)
47. {
48. size[u]=1;
49. **if**(dis[u]<=K) q[++q[0]]=dis[u];
50. **for**(**int** i=fir[u];i;i=next[i]) **if**(end[i]!=fa&&!vis[end[i]])
51. {
52. dis[end[i]]=dis[u]+len[i];
53. DFS(end[i],u);
54. size[u]+=size[end[i]];
55. }
56. **return**;
57. }
58. **long** **long** Cnt(**int** u,**int** st\_dis)
59. {
60. **long** **long** sum=q[0]=0;
61. dis[u]=st\_dis;
62. DFS(u,u);
63. sort(q+1,q+q[0]+1);
64. **for**(**int** l=1,r=1;l<=q[0];l++)
65. {
66. **while**(r<q[0]&&q[r+1]+q[l]<=K) r++;
67. **while**(l<r&&K<q[r]+q[l]) r--;
68. **if**(l<r) sum=sum+r-l;
69. }
70. **return** sum;
71. }
72. **long** **long** get\_ans(**int** u,**int** all)
73. {
74. **long** **long** ans=0;
75. root=-1;
76. get\_root(u,u,all);
77. u=root;
78. vis[u]=**true**;
79. ans+=Cnt(u,0);
80. **for**(**int** i=fir[u];i;i=next[i]) **if**(!vis[end[i]])
81. ans-=Cnt(end[i],len[i]);
82. **for**(**int** i=fir[u];i;i=next[i]) **if**(!vis[end[i]])
83. ans+=get\_ans(end[i],size[end[i]]);
84. **return** ans;
85. }

# AC自动机

1. **int** ch[100050][26],val[100050],last[100050],pre[100050],sz=0;
2. **void** insert(**char** T[],**int** x)
3. {
4. **int** lenth=strlen(T),u=0;
5. **for**(**int** i=0;i<lenth;u=ch[u][T[i++]-97])
6. **if**(!ch[u][T[i]-97])
7. ch[u][T[i]-97]=++sz;
8. val[u]=x;
9. **return**;
10. }
11. **void** getfail()
12. {
13. **int** u,v,f;
14. queue<**int**>q;
15. **for**(**int** i=0;i<26;i++)
16. {
17. **if**(!ch[0][i])
18. **continue**;
19. pre[ch[0][i]]=last[ch[0][i]]=0;
20. q.push(ch[0][i]);
21. }
22. **while**(q.size())
23. {
24. u=q.front();
25. q.pop();
26. **for**(**int** i=0;i<26;i++)
27. {
28. **if**(!ch[u][i])
29. {
30. ch[u][i]=ch[pre[u]][i];
31. **continue**;
32. }
33. v=ch[u][i];
34. q.push(v);
35. f=pre[u];
36. **while**(f&&!ch[f][i])
37. f=pre[f];
38. pre[v]=ch[f][i];
39. last[v]=val[pre[v]]?pre[v]:last[pre[v]];
40. }
41. }
42. **return**;
43. }
44. **void** count(**int** u)
45. {
46. **if**(u)
47. {
48. // do some operators
49. **if**(last(u)) count(last(u));
50. }
51. **return**;
52. }
53. **void** find(**char** T[])
54. {
55. **int** lenth=strlen(T),u=0;
56. **for**(**int** i=0;i<lenth;i++)
57. {
58. /\*while(u&&!ch[u][T[i]-97])
59. u=pre[u];\*/
60. u=ch[u][T[i]-97];
61. **if**(val[u]) count(u);
62. **else** **if**(last[u]) count(last[u]);
63. }
64. **return**;
65. }

# KMP

1. **char** tex[1050],T[100050];
2. **int** KMP()
3. {
4. **int** n=strlen(tex),m=strlen(T),pre[1050],i,k;
5. pre[0]=-1;
6. **for**(i=1,k=-1;i<n;pre[i++]=k)
7. {
8. **while**(k>=0&&tex[i]!=tex[k+1])
9. k=pre[k];
10. **if**(tex[i]==tex[k+1])
11. k++;
12. }
13. **for**(i=0,k=-1;i<m;i++)
14. {
15. **while**(k>=0&&T[i]!=tex[k+1])
16. k=pre[k];
17. **if**(T[i]==tex[k+1])
18. {
19. k++;
20. **if**(k==n-1)
21. **return** i-k;
22. }
23. }
24. **return** -1;
25. }

# 字符串Hash

1. **char** tex[1050],T[100050];
2. unsigned **long** **long** H[100050][3],h[3],p[3];
3. **int** Hash()
4. {
5. n=strlen(tex);
6. m=strlen(T);
7. p[0]=p[1]=p[2]=1;
8. **for**(**int** i=1;i<=n;i++)
9. {
10. p[0]\*=13;
11. p[1]\*=17;
12. p[2]\*=19;
13. }
14. **for**(**int** i=n-1;i>=0;i--)
15. {
16. h[0]=h[0]\*13+tex[i]-97;
17. h[1]=h[1]\*17+tex[i]-97;
18. h[2]=h[2]\*19+tex[i]-97;
19. }
20. **for**(**int** i=m-1;i>=0;i--)
21. {
22. H[i][0]=13\*H[i+1][0]+T[i]-97;
23. H[i][1]=17\*H[i+1][1]+T[i]-97;
24. H[i][2]=19\*H[i+1][2]+T[i]-97;
25. }
26. **for**(**int** i=0;i<=m-n;i++)
27. **if**(H[i][0]-H[i+n][0]\*p[0]==h[0]
28. &&H[i][1]-H[i+n][1]\*p[1]==h[1]
29. &&H[i][2]-H[i+n][2]\*p[2]==h[2])
30. **return** i;
31. **return** -1;
32. }

# 回文树

1. **int** ch[100050][26],len[100050],fail[100050],cnt[100050],last,sz;
2. **int** getfail(**char** T[],**int** x,**int** i)
3. {
4. **while**(T[i-len[x]-1]!=T[i]) x=fail[x];
5. **return** x;
6. }
7. **void** init()
8. {
9. memset(ch[0],last=0,**sizeof**(ch[0]));//lastÎªµ±Ç°µÄ×î³¤ºó×º»ØÎÄ
10. memset(ch[1],0,**sizeof**(ch[1]));
11. len[0]=0; len[1]=-1; fail[0]=1;
12. sz=1;
13. **return**;
14. }
15. **void** insert(**char** T[])
16. {
17. **int** lenth=strlen(T),cur;
18. **for**(**int** i=0;i<lenth;i++)
19. {
20. cur=getfail(T,last,i);
21. **if**(!ch[cur][T[i]-97])
22. {
23. ch[cur][T[i]-97]=++sz;
24. memset(ch[sz],0,**sizeof**(ch[sz])); len[sz]=0;
25. fail[sz]=ch[ getfail(T,fail[cur],i) ][T[i]-97];
26. }
27. last=ch[cur][T[i]-97];
28. cnt[last]++;
29. }
30. }

# 后缀数组

1. **const** **int** MAXL=100050;
2. **char** T[MAXL];
3. **int** sa[MAXL],height[MAXL],c[MAXL],t1[MAXL],t2[MAXL],rank[MAXL],lenth;
4. **void** build\_sa()
5. {
6. **int** \*x=t1,\*y=t2,m=26,p;
7. memset(c,0,**sizeof**(c));
8. **for**(**int** i=0;i<lenth;i++) c[x[i]=T[i]-97]++;
9. **for**(**int** i=1;i<m;i++) c[i]+=c[i-1];
10. **for**(**int** i=lenth-1;i>=0;i--) sa[--c[x[i]]]=i;
11. **for**(**int** k=1;k<=lenth;k<<=1)
12. {
13. p=0;
14. **for**(**int** i=lenth-k;i<lenth;i++) y[p++]=i;
15. **for**(**int** i=0;i<lenth;i++) **if**(sa[i]>=k) y[p++]=sa[i]-k;
16. memset(c,0,**sizeof**(c));
17. **for**(**int** i=0;i<lenth;i++) c[x[y[i]]]++;
18. **for**(**int** i=1;i<m;i++) c[i]+=c[i-1];
19. **for**(**int** i=lenth-1;i>=0;i--) sa[--c[x[y[i]]]]=y[i];
20. swap(x,y);
21. x[sa[0]]=0;
22. **for**(**int** i=p=1;i<lenth;i++)
23. x[sa[i]]=y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k]?p-1:p++;
24. **if**(p>=lenth)**break**;
25. m=p;
26. }
27. **return**;
28. }
29. **void** bulid\_height()
30. {
31. **for**(**int** i=0;i<lenth;i++) rank[sa[i]]=i;
32. height[rank[0]]=0;
33. **for**(**int** i=1,k=0,j;i<lenth;i++)
34. {
35. **if**(k) k--;
36. **if**(!rank[i])**continue**;
37. j=sa[rank[i]-1];
38. **while**(T[i+k]==T[j+k])k++;
39. height[rank[i]]=k;
40. }
41. **return**;
42. }
43. **int** solve1(**int** a,**int** b)
44. {//最长公共前缀
45. //转换为排行为a的后缀和排行为b的后缀的最长公共前缀缀b的最长公共前缀
46. **int** ans=0;
47. a=rank[a]; b=rank[b];
48. **if**(a>b) swap(a,b);
49. **for**(**int** i=a+1;i<=b;i++) ans=max(ans,height[i]);//这是暴力写法
50. //因为是RMQ问题，所以可以有更优的做法
51. **return** ans;
52. }
53. **int** solve2()
54. {//可重叠最长重复子串
55. //因为可重叠重复子串都是某些后缀的公共前缀，所以问题转换为求后缀间最长的公共前缀
56. **int** ans=0;
57. **for**(**int** i=0;i<lenth;i++) ans=max(ans,height[i]);
58. //扫一遍，O(n)
59. **return** ans;
60. }
61. **bool** solve3\_check(**int** x,**int** k)
62. {
63. **int** cnt=1;
64. **for**(**int** i=0;i<=lenth;i++)
65. {
66. **if**(i==lenth||height[i]<x)
67. {
68. **if**(cnt>=k)**return** **true**;
69. cnt=1;
70. }
71. **else** cnt++;
72. }
73. **return** **false**;
74. }
75. **int** solve3(**int** k)
76. {//可重叠的k次最长重复子串
77. //问题转换为求一个最长的至少为k个后缀的公共前缀的子串，可二分答案
78. **int** l,r,mid;
79. **if**(!solve3\_check(1,k))**return** 0;
80. l=1; r=lenth+1;
81. **while**(l<r-1)
82. {
83. mid=(l+r)>>1;
84. **if**(solve3\_check(mid,k)) l=mid;
85. **else** r=mid;
86. }
87. **return** l;
88. }

# 莫比乌斯函数

1. **int** u[100050],p[100050];
2. **bool** exist[100050];
3. **void** Mobius()
4. {
5. memset(exist,**true**,**sizeof**(exist));
6. u[1]=1;
7. **for**(**int** i=2;i<100050;i++)
8. {
9. **if**(exist[i]) p[++p[0]]=i,u[i]=-1;
10. **for**(**int** j=1;i\*p[j]<100050&&j<=p[0];j++)
11. {
12. exist[i\*p[j]]=**false**;
13. **if**(i%p[j]) u[i\*p[j]]=-u[i];
14. **else**
15. {
16. u[i\*p[j]]=0;
17. **break**;
18. }
19. }
20. }
21. **return**;
22. }