# KMP

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

# EX-KMP

1. **void** getextend()
2. {
3. next[1]=n;
4. **int** a=0,p=0;
5. fo(i,2,n)
6. {
7. **int** j=next[i-a+1];
8. **if**(i+j>p) **for**(j=max(0,p-i);i+j<=n && t[i+j]==t[j+1];j++);
9. next[i]=j;
10. **if**(i+j-1>p) a=i,p=i+j-1;
11. }
12. a=p=0;
13. fo(i,1,n)
14. {
15. **int** j=next[i-a+1];
16. **if**(i+j>p) **for**(j=max(0,p-i+1);i+j<=n && s[i+j]==t[j+1];j++);
17. extend[i]=j;
18. **if**(i+j-1>p) a=i,p=i+j-1;
19. }
20. }

# AC自动机

1. **int** ch[100050][26],val[100050],last[100050],pre[100050],sz=0;
2. **char** tex[100050];
3. **void** insert(**char** T[],**int** x)
4. {
5. **int** lenth=strlen(T),u=0;
6. **for**(**int** i=0;i<lenth;u=ch[u][T[i++]-97])
7. **if**(!ch[u][T[i]-97])
8. ch[u][T[i]-97]=++sz;
9. val[u]=x;
10. **return**;
11. }
12. **void** getfail()
13. {
14. **int** u,v,f;
15. queue<**int**>q;
16. **for**(**int** i=0;i<26;i++)
17. {
18. **if**(!ch[0][i])
19. **continue**;
20. pre[ch[0][i]]=last[ch[0][i]]=0;
21. q.push(ch[0][i]);
22. }
23. **while**(q.size())
24. {
25. u=q.front();
26. q.pop();
27. **for**(**int** i=0;i<26;i++)
28. {
29. **if**(!ch[u][i])
30. {
31. ch[u][i]=ch[pre[u]][i];
32. **continue**;
33. }
34. v=ch[u][i];
35. q.push(v);
36. f=pre[u];
37. **while**(f&&!ch[f][i])
38. f=pre[f];
39. pre[v]=ch[f][i];
40. last[v]=val[pre[v]]?pre[v]:last[pre[v]];
41. }
42. }
43. **return**;
44. }
45. **void** count(**int** u)
46. {
47. **if**(u)
48. {
49. // do some operators
50. **if**(last(u)) count(last(u));
51. }
52. **return**;
53. }
54. **void** find(**char** T[])
55. {
56. **int** lenth=strlen(T),u=0;
57. **for**(**int** i=0;i<lenth;i++)
58. {
59. /\*while(u&&!ch[u][T[i]-97])
60. u=pre[u];\*/
61. u=ch[u][T[i]-97];
62. **if**(val[u]) count(u);
63. **else** **if**(last[u]) count(last[u]);
64. }
65. **return**;
66. }

# 回文自动机

1. **const** **int** MAXL=100050;
2. **const** **int** MAXC=26;
3. **int** ch[MAXL][MAXC],len[MAXL],fail[MAXL],cnt[MAXL],last,sz;
4. **int** getfail(**char** T[],**int** x,**int** i)
5. {
6. **while**(T[i-len[x]-1]!=T[i]) x=fail[x];
7. **return** x;
8. }
9. **void** init()
10. {
11. memset(ch[0],last=0,**sizeof**(ch[0]));//last为当前的最长后缀回文
12. memset(ch[1],0,**sizeof**(ch[1]));
13. len[0]=0; len[1]=-1; fail[0]=1;
14. sz=1;
15. **return**;
16. }
17. **void** insert(**char** T[])
18. {
19. **int** lenth=strlen(T),cur;
20. **for**(**int** i=0;i<lenth;i++)
21. {
22. cur=getfail(T,last,i);
23. **if**(!ch[cur][T[i]-97])
24. {
25. fail[++sz]=ch[ getfail(T,fail[cur],i) ][T[i]-97];
26. memset(ch[sz],0,**sizeof**(ch[sz]));
27. len[sz]=len[u]+2;
28. ch[cur][T[i]-97]=sz;
29. }
30. last=ch[cur][T[i]-97];
31. cnt[last]++;
32. }
33. }

# 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. }

# 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. }

# Treap

1. **struct** jp \*null;
2. **struct** jp
3. {
4. **int** data,size;
5. jp \*son[2];
6. jp(**const** **int** a=0,**const** **int** b=0):data(a),size(b)
7. {
8. son[0]=son[1]=null;
9. }
10. }\*root;
11. **void** adjust(jp\* &node,**int** d)
12. {
13. jp \*temp=node->son[d];
14. **int** t=node->size-temp->size;
15. node->size-=(temp->size-temp->son[d^1]->size);
16. temp->size+=t;
17. node->son[d]=temp->son[d^1];
18. temp->son[d^1]=node;
19. node=temp;
20. **return**;
21. }
22. **void** insert(jp\* &r,**int** key)
23. {
24. **if**(r==null)
25. {
26. r=**new** jp(key,1);
27. **return**;
28. }
29. r->size++;
30. **if**(r->data!=key)
31. insert(r->son[r->data<key],key);
32. **return**;
33. }
34. **void** del(jp\* &r,**int** key)
35. {
36. **if**(r==null)
37. **return**;
38. **if**(r->data==key)
39. {
40. **if**(r->size-(r->son[0]->size)-(r->son[1]->size)==1)
41. {
42. **for**(**int** d;r->son[0]!=null&&r->son[1]!=null;r=r->son[d^1])
43. adjust(r,d=(r->son[0]->size<r->son[1]->size));
44. **if**(r->son[0]!=null)
45. r=r->son[0];
46. **else** **if**(r->son[1]!=null)
47. r=r->son[1];
48. **else**
49. r=null;
50. }
51. **else**
52. r->size--;
53. **return**;
54. }
55. **else**
56. del(r->son[r->data<key],key);
57. r->size--;
58. **return**;
59. }
60. **int** query(**const** jp \*r,**int** key)
61. {
62. **if**(r==null)
63. **return** 1;
64. **if**(r->data==key)
65. **return** r->son[0]->size+1;
66. **if**(key<r->data)
67. **return** query(r->son[0],key);
68. **return** r->size-r->son[1]->size+query(r->son[1],key);
69. }
70. **int** k\_th(**const** jp \*r,**int** rank)
71. {
72. **if**(r==null)
73. **return** 0;
74. **if**(rank<=r->son[0]->size)
75. **return** k\_th(r->son[0],rank);
76. rank-=r->son[0]->size;
77. **return** (rank<=r->size-r->son[1]->size-r->son[0]->size)?r->data:k\_th(r->son[1],rank-(r->size-r->son[1]->size-r->son[0]->size));
78. }
79. **int** find(**int** key,**int** d)
80. {
81. **int** d2,ans;
82. d2=root->data==key?-1:root->data<key;
83. **if**(d2==-1||d2==d)
84. ans=0;
85. **else**
86. ans=root->data;
87. jp \*r=root;
88. **while**(r!=null)
89. {
90. d2=r->data==key?-1:r->data<key;
91. **if**(d2==(d^1))
92. ans=r->data;
93. r=r->son[d2==-1?d:d2];
94. }
95. **return** ans;
96. }
97. null=**new** jp();
98. root=null->son[0]=null->son[1]=null;

# Splay

1. **struct** jp \*null;
2. **struct** jp
3. {
4. **int** key,size;
5. **bool** flag;
6. jp \*son[2];
7. jp(**const** **int** &x=0):key(x) { size=1; flag=**false**; son[0]=son[1]=null; }
8. **int** cmp(**int** k) { **return** k==(son[0]->size+1)?-1:(k<=son[0]->size?0:1); }
9. **void** updata() { size=son[0]->size+son[1]->size+1; **return**; }
10. **void** down()
11. {
12. **if**(!flag) **return**;
13. flag=**false**;
14. swap(son[0],son[1]);
15. son[0]->flag^=**true**;
16. son[1]->flag^=**true**;
17. **return**;
18. }
19. }\*root;
20. **int** cnt=0,n,m;
21. **void** adjust(jp\* &r,**int** d)
22. {
23. jp\* u=r->son[d]; r->son[d]=u->son[d^1]; u->son[d^1]=r;
24. r->updata(); (r=u)->updata(); **return**;
25. }
26. **void** build(jp\* &u,**int** l,**int** r)
27. {
28. **if**(l>r) { u=null; **return**; }
29. **int** mid=(l+r)>>1;
30. u=**new** jp();
31. build(u->son[0],l,mid-1);
32. u->key=cnt++;
33. build(u->son[1],mid+1,r);
34. u->updata();//勿忘
35. **return**;
36. }
37. **void** m41441(jp\* &r,**int** k)
38. {
39. r->down();
40. **int** d=r->cmp(k);
41. **if**(d==1) k=k-r->son[0]->size-1;
42. **if**(d!=-1)
43. {
44. jp\* u=r->son[d];
45. u->down();
46. **int** d2=u->cmp(k);
47. **int** k2=d2==1?k-u->son[0]->size-1:k;
48. **if**(d2!=-1)
49. {
50. m41441(u->son[d2],k2);
51. **if**(d2==d) adjust(r,d);
52. **else** adjust(r->son[d],d2);
53. }
54. adjust(r,d);
55. }
56. **return**;
57. }
58. **void** split(jp\* all,**int** k,jp\* &left,jp\* &right)
59. {
60. m41441(all,k);
61. left=all;
62. right=all->son[1];
63. left->son[1]=null;
64. left->updata();
65. **return**;
66. }
67. jp \*merge(jp\* left,jp\* right)
68. {
69. m41441(left,left->size);
70. left->son[1]=right;
71. left->updata();
72. **return** left;
73. }
74. **void** rEverse(**int** l,**int** r)
75. {
76. jp \*temp,\*left,\*mid,\*right;
77. split(root,l,left,temp);
78. split(temp,r-l+1,mid,right);
79. mid->flag^=**true**;
80. root=merge(merge(left,mid),right);
81. **return**;
82. }
83. **int** tot=0;
84. **void** print(jp\* &r)
85. {
86. **if**(r==null) **return**;
87. r->down();
88. print(r->son[0]);
89. **if**(r->key)
90. {
91. printf("%d",r->key);
92. **if**(++tot!=n) putchar(' ');
93. }
94. print(r->son[1]);
95. **return**;
96. }
97. root=null=**new** jp();
98. null->son[0]=null->son[1]=null;
99. null->size=0;
100. build(root,0,n);

# LCT

1. **const** **int** MAXN=10050;
2. **int** fa[MAXN],son[MAXN][2],st[MAXN];
3. **bool** rev[MAXN];
4. **bool** isroot(**int** x)
5. {// Check whether x is the root of the Auxiliary tree it belongs to
6. **return** x!=son[fa[x]][0]&&x!=son[fa[x]][1];
7. }
8. **void** Down(**int** x)
9. {
10. **if**(!rev[x]) **return**;
11. swap(son[son[x][0]][0],son[son[x][0]][1]);
12. swap(son[son[x][1]][0],son[son[x][1]][1]);
13. rev[son[x][0]]^=**true**;
14. rev[son[x][1]]^=**true**;
15. rev[x]=**false**;
16. **return**;
17. }
18. **void** adjust(**int** x)
19. {
20. **int** fa1=fa[x],fa2=fa[fa[x]],d;
21. d=son[fa1][1]==x;
22. **if**(!isroot(fa1))
23. son[fa2][son[fa2][1]==fa1]=x;
24. fa[x]=fa2;
25. fa[fa1]=x;
26. fa[son[x][d^1]]=fa1;
27. son[fa1][d]=son[x][d^1];
28. son[x][d^1]=fa1;
29. **return**;
30. }
31. **void** Splay(**int** x)
32. {
33. st[0]=0;
34. st[++st[0]]=x;
35. **for**(**int** i=x;!isroot(i);i=fa[i])
36. st[++st[0]]=fa[i];
37. **while**(st[0]) Down(st[st[0]--]);
38. **int** fa1,fa2;
39. **while**(!isroot(x))
40. {
41. fa1=fa[x];
42. fa2=fa[fa[x]];
43. **if**(!isroot(fa1))
44. {
45. **if**((son[fa1][1]==x)==(son[fa2][1]==fa1))
46. adjust(fa1);
47. **else** adjust(x);
48. }
49. adjust(x);
50. }
51. **return**;
52. }
53. **void** Access(**int** x)
54. {
55. **int** t=0;
56. **while**(x)
57. {
58. Splay(x);
59. son[x][1]=t;
60. t=x;
61. x=fa[x];
62. }
63. **return**;
64. }
65. **void** Change\_to\_root(**int** x)
66. {
67. Access(x); Splay(x);
68. swap(son[x][0],son[x][1]);
69. rev[x]^=**true**;
70. **return**;
71. }
72. **void** Link(**int** x,**int** y)
73. {
74. Change\_to\_root(y);
75. fa[y]=x;
76. //Splay(y);
77. **return**;
78. }
79. **void** Cut(**int** x,**int** y)
80. {
81. Change\_to\_root(x);
82. Access(y);//
83. Splay(y);
84. son[y][0]=fa[x]=0;
85. **return**;
86. }
87. **int** Find\_root(**int** x)
88. {
89. Access(x); Splay(x);
90. **while**(son[x][0]) x=son[x][0];
91. **return** x;
92. }

# 主席树

# 斜率优化DP

# 点分治

# Dinic

# 欧拉筛（欧拉函数、莫比乌斯函数）

1. **const** **int** MAXN=100050;
2. **int** pri[MAXN],phi[MAXN],u[MAXN];
3. **bool** vis[MAXN];
4. **void** Euler()
5. {
6. memset(vis,**true**,**sizeof**(vis));
7. pri[0]=0;
8. u[1]=1; phi[1]=1;
9. **for**(**int** i=2;i<MAXN;i++)
10. {
11. **if**(vis[i])
12. {
13. pri[++pri[0]]=i;
14. phi[i]=i-1;
15. u[i]=-1;
16. }
17. **for**(**int** j=1;j<=pri[0]&&i\*pri[j]<MAXN;j++)
18. {
19. vis[i\*pri[j]]=**false**;
20. **if**(i%pri[j])
21. {
22. phi[i\*pri[j]]=phi[i]\*(pri[j]-1);
23. u[i\*pri[j]]=-u[i];
24. }
25. **else**
26. {
27. phi[i\*pri[j]]=phi[i]\*pri[j];
28. **break**;
29. }
30. }
31. }
32. }

# FNT(内含超级IO)

1. **const** **int** P=(479<<21)+1,G=3,D=10000000;
2. **int** A[405000],B[405000],rev[400050],n,m,lenth,inver;
3. **char** in[D],\*I=in,out[D/10],\*O=out;
4. **template**<**typename** Ty>**void** read1n(Ty &x)
5. {
6. **for**(x=0;\*I<'0'||'9'<\*I;I++);
7. **while**('0'<=\*I&&\*I<='9')x=(x<<1)+(x<<3)+\*(I++)-'0';
8. **return**;
9. }
10. **template**<**typename** Ty>**void** write1n(Ty x)
11. {
12. **static** **int** s[70];
13. **if**(!x)\*(O++)='0';
14. **for**(s[0]=0;x;x/=10)s[++s[0]]=(**int**)x%10;
15. **while**(s[0])\*(O++)=s[s[0]--]+'0';
16. **return**;
17. }
18. **int** pOw(**int** a,**int** n)
19. {
20. **int** ans;
21. **for**(ans=1;n;n>>=1,a=(**int**)((**long** **long**)a\*a%P))
22. **if**(n&1)ans=(**int**)((**long** **long**)ans\*a%P);
23. **return** ans;
24. }
25. **void** FFT\_pre(**int** &n)
26. {
27. **int** tem;**for**(tem=0;(1<<tem)<n;tem++);n=1<<tem;
28. **for**(**int** i=0;i<n;i++)rev[i]=rev[i>>1]>>1|((i&1)<<(tem-1));
29. inver=pOw(n,P-2);
30. **return**;
31. }
32. **void** FFT(**int** X[],**int** n,**bool** inv)
33. {
34. **for**(**int** i=0;i<n;i++)**if**(rev[i]>i)swap(X[i],X[rev[i]]);
35. **int** temk,w,cur,temp;
36. **for**(**int** k=2;k<=n;k<<=1)
37. {
38. temk=k>>1;
39. cur=1;w=pOw(G,(P-1)/k);
40. **if**(inv)w=pOw(w,k-1);
41. **for**(**int** i=0;i<temk;i++)
42. {
43. **for**(**int** j=0;j<n;j+=k)
44. {
45. temp=(**int**)((**long** **long**)cur\*X[i+j+temk]%P);
46. X[i+j+temk]=(**int**)((**long** **long**)(X[i+j]-temp+P)%P);
47. X[i+j]=(**int**)((**long** **long**)(X[i+j]+temp)%P);
48. }
49. cur=(**int**)((**long** **long**)cur\*w%P);
50. }
51. }
52. **if**(inv)**for**(**int** i=0;i<n;i++)X[i]=(**int**)((**long** **long**)X[i]\*inver%P);
53. **return**;
54. }
55. **int** main()
56. {
57. in[fread(in,1,D,stdin)]=-1;
58. read1n(n);read1n(m);n++;m++;
59. lenth=n+m-1;
60. **for**(**int** i=0;i<n;i++)read1n(A[i]);
61. **for**(**int** i=0;i<m;i++)read1n(B[i]);
62. n=max(n,m);
63. n<<=1;
64. FFT\_pre(n);
65. FFT(A,n,**false**);
66. FFT(B,n,**false**);
67. //write1n(10),\*(O++)='\n';
68. **for**(**int** i=0;i<n;i++)A[i]=(**int**)((**long** **long**)A[i]\*B[i]%P);
69. FFT(A,n,**true**);
70. **for**(**int** i=0;i<lenth;i++)write1n(A[i]),\*(O++)=' ';
71. fwrite(out,1,O-out,stdout);
72. **return** 0;
73. }

# FFT

1. **struct** jp
2. {
3. **double** r,i;
4. jp(**double** A=0,**double** B=0):r(A),i(B){}
5. jp operator+(**const** jp X){**return** jp(r+X.r,i+X.i);}
6. jp operator-(**const** jp X){**return** jp(r-X.r,i-X.i);}
7. jp operator\*(**const** jp X){**return** jp(r\*X.r-i\*X.i,r\*X.i+i\*X.r);}
8. }A[405000],B[405000];
9. **const** **int** D=10000000;
10. **char** in[D],\*I=in,out[D/10],\*O=out;
11. **int** rev[405000],n,m,lenth;
12. **template**<**typename** Ty>
13. **void** read1n(Ty &x)
14. {
15. **for**(x=0;\*I<'0'||'9'<\*I;I++);
16. **while**('0'<=\*I&&\*I<='9')x=x\*10+\*(I++)-'0';
17. **return**;
18. }
19. **template**<**typename** Ty>
20. **void** write1n(Ty x)
21. {
22. **static** **int** s[70];
23. **if**(!x)\*(O++)='0';
24. **for**(s[0]=0;x;x/=10)s[++s[0]]=x%10;
25. **while**(s[0])\*(O++)=s[s[0]--]+'0';
26. **return**;
27. }
28. **void** FFT\_pre(**int** &n)
29. {
30. **int** tem;**for**(tem=0;(1<<tem)<n;tem++);n=1<<tem;
31. **for**(**int** i=1;i<n;i++)rev[i]=rev[i>>1]>>1|((i&1)<<(tem-1));
32. **return**;
33. }
34. **const** **double** pI=2\*acos(-1.);
35. **void** FFT(jp X[],**int** n,**bool** inv)
36. {
37. **for**(**int** i=1;i<n;i++)**if**(rev[i]>i)swap(X[i],X[rev[i]]);
38. jp w,cur,temp;
39. **int** temk;
40. **for**(**int** k=2;k<=n;k<<=1)
41. {
42. temk=k>>1;
43. cur=jp(1.,0.);
44. w=!inv?jp(cos(pI/k),sin(pI/k)):jp(cos(-1.\*pI/k),sin(-1.\*pI/k));
45. **for**(**int** i=0;i<temk;i++)
46. {
47. **for**(**int** j=0;j<n;j+=k)
48. {
49. temp=cur\*X[i+j+temk];
50. X[i+j+temk]=X[i+j]-temp;
51. X[i+j]=X[i+j]+temp;
52. }
53. cur=cur\*w;
54. }
55. }
56. **if**(inv)**for**(**int** i=0;i<n;i++)X[i].r/=n;
57. **return**;
58. }
59. **int** main()
60. {
61. in[fread(in,1,D,stdin)];
62. read1n(n);read1n(m);n++;m++;lenth=n+m-1;
63. **for**(**int** i=0;i<n;i++)read1n(A[i].r);
64. **for**(**int** i=0;i<m;i++)read1n(B[i].r);
65. n=max(n,m)<<1;
66. FFT\_pre(n);FFT(A,n,**false**);FFT(B,n,**false**);
67. **for**(**int** i=0;i<n;i++)A[i]=A[i]\*B[i];
68. FFT(A,n,**true**);
69. **for**(**int** i=0;i<lenth;i++)write1n(**int**(A[i].r+0.5)),\*O++=' ';
70. fwrite(out,1,O-out,stdout);
71. **return** 0;
72. }

# 斜堆

1. **template**<**typename** Ty>
2. **struct** heap
3. {
4. **static** **const** **int** MAXN=100050;
5. **int** tOp;
6. **int** L[MAXN],R[MAXN],dist[MAXN],p[MAXN];
7. Ty data[MAXN];
8. heap()
9. {
10. tOp=0;
11. memset(L,0,**sizeof**(L));
12. memset(R,0,**sizeof**(R));
13. memset(dist,0,**sizeof**(dist));
14. **for**(**int** i=1;i<MAXN;i++)
15. p[i]=i;
16. p[0]=MAXN-1;
17. }
18. **void** clear()
19. {
20. tOp=0;
21. memset(L,0,**sizeof**(L));
22. memset(R,0,**sizeof**(R));
23. memset(dist,0,**sizeof**(dist));
24. **for**(**int** i=1;i<MAXN;i++)
25. p[i]=i;
26. p[0]=MAXN-1;
27. **return**;
28. }
29. **bool** empty() { **return** (**bool**)tOp; }
30. **int** merge(**int** A,**int** B)
31. {
32. **if**(!A) **return** B;
33. **if**(!B) **return** A;
34. **if**(data[A]<data[B]) swap(A,B);
35. R[A]=merge(R[A],B);
36. **if**(dist[L[A]]<dist[R[A]])
37. swap(L[A],R[A]);
38. dist[A]=R[A]?dist[R[A]]+1:0;
39. **return** A;
40. }
41. **void** push(Ty X)
42. {
43. data[p[p[0]]]=X;
44. tOp=merge(tOp,p[p[0]--]);
45. **return**;
46. }
47. **void** pop()
48. {
49. p[++p[0]]=tOp;
50. tOp=merge(L[tOp],R[tOp]);
51. L[p[p[0]]]=R[p[p[0]]]=0;
52. **return**;
53. }
54. Ty top() { **return** data[tOp]; }
55. };

# 表达式树

1. **const** **int** MAXN=100050;
2. **int** nc,lch[MAXN],rch[MAXN],op[MAXN];
3. **int** build\_tree(**char** \*s,**int** x,**int** y)
4. {
5. **int** c1=-1,c2=-1,p=0,u;
6. **if**(y-x==1)
7. {
8. u=++nc;
9. lch[u]=rch[u]=0;
10. op[u]=s[x];
11. **return** u;
12. }
13. **for**(**int** i=x;i<y;i++)
14. {
15. **switch**(s[i])
16. {
17. **case** '(':p++;**break**;
18. **case** ')':p--;**break**;
19. **case** '+':**case** '-':**if**(!p)c1=i;**break**;
20. **case** '\*':**case** '/':**if**(!p)c2=i;**break**;
21. }
22. }
23. **if**(c1<0)c1=c2;
24. **if**(c1<0)**return** build\_tree(s,x+1,y-1);
25. u=++nc;
26. lch[u]=build\_tree(s,x,c1);
27. rch[u]=build\_tree(s,c1+1,y);
28. op[u]=s[c1];
29. **return** u;
30. }

# Berlekamp-Massey Algorithm

1. #include <cstdio>
2. #include <vector>
4. **using** **namespace** std;
6. **namespace** BerlekampMassey {
7. **const** **int** mod = 1e9 + 7;
8. **int** L, m, b, n;
9. vector<**int**> s, C, B;
10. **void** init() {
11. s.clear();
12. C.clear();
13. B.clear();
14. C.push\_back(1);
15. B.push\_back(1);
16. L = n = 0;
17. m = b = 1;
18. }
19. **int** pow\_mod(**int** a, **int** k) {
20. **int** s = 1;
21. **while** (k) {
22. **if** (k & 1)
23. s = 1ll \* s \* a % mod;
24. a = 1ll \* a \* a % mod;
25. k >>= 1;
26. }
27. **return** s;
28. }
29. **void** update(**int** d) {
30. s.push\_back(d);
31. **for** (**int** i = 1; i <= L; ++i)
32. d = (d + 1ll \* C[i] \* s[n - i] % mod) % mod;
33. **if** (d == 0)
34. ++m;
35. **else** **if** (2 \* L <= n) {
36. vector<**int**> T = C;
37. C.resize(n + 1 - L + 1);
38. **for** (**int** i = L + 1; i <= n + 1 - L; ++i)
39. C[i] = 0;
40. **for** (**int** i = 0; i < B.size(); ++i)
41. C[i + m] = (C[i + m] + mod - 1ll \* d \* pow\_mod(b, mod - 2) % mod \* B[i] % mod) % mod;
42. L = n + 1 - L;
43. B = T;
44. b = d;
45. m = 1;
46. } **else** {
47. **for** (**int** i = 0; i < B.size(); ++i)
48. C[i + m] = (C[i + m] + mod - 1ll \* d \* pow\_mod(b, mod - 2) % mod \* B[i] % mod) % mod;
49. ++m;
50. }
51. ++n;
52. }
53. **void** output() {
54. printf("F(n)=");
55. **for** (**int** i = 1; i < C.size(); ++i) {
56. **int** output = (mod - C[i]) % mod;
57. **if** (output > mod / 2)
58. output -= mod;
59. printf("%s%d\*F(n-%d)", (output < 0 || i == 1) ? "" : "+", output, i);
60. }
61. puts("");
62. }
63. **void** output\_code\_for() {
64. **static** **const** **char** \*name = "dp";
65. **static** **const** **char** \*index = "i";
66. **static** **const** **char** \*upperbound = "maxn";
67. puts("// Generated by Berlekamp-Massey algorithm");
68. **for** (**int** i = 1; i < C.size(); ++i) {
69. printf("%s[%d]=%d;\n", name, i - 1, s[i - 1]);
70. }
71. printf("for(int i=%d;i<%s;++i)\n", (**int**)C.size() - 1, upperbound);
72. printf("  %s[%s]=((", name, index);
73. **for** (**int** i = 1; i < C.size(); ++i) {
74. **int** output = (mod - C[i]) % mod;
75. **if** (output > mod / 2)
76. output -= mod;
77. printf("%s%d\*%s[%s-%d]%%mod", (output < 0 || i == 1) ? "" : "+", output, name, index, i);
78. }
79. puts(")%mod+mod)%mod;");
80. }
81. **void** output\_code\_matrix() {
82. // TODO
83. }
84. };
85. #include "BerlekampMassey.cpp"
87. **int** main() {
88. **int** arr[12] = {2, 24, 96, 416, 1536, 5504, 18944, 64000, 212992, 702464, 2301952, 7512064};
89. BerlekampMassey::init();
90. **for** (**int** i = 0; i < 12; ++i) {
91. BerlekampMassey::update(arr[i]);
92. }
93. printf("Formule: ");
94. BerlekampMassey::output();
95. printf("Code: \n");
96. BerlekampMassey::output\_code\_for();
97. **return** 0;
98. }

最小表示法

1. **int** minP(**char** s[])
2. {
3. **int** l=strlen(s);
4. **int** i=0,j=1,k=0;
5. **while**(**true**)
6. {
7. **if**(i+k>=l||j+k>=l) **break**;
8. **if**(s[i+k]==s[j+k])
9. {
10. k++;
11. **continue**;
12. }
13. **else**
14. {
15. **if**(s[j+k]>s[i+k]) j+=k+1;
16. **else** i+=k+1;
17. k=0;
18. **if**(i==j) j++;
19. }
20. }
21. **return** min(i,j);
22. }

Matrix-Tree

1. #include<iostream>
2. #include<algorithm>
3. #include<cstring>
4. #include<vector>
5. #include<cstdio>
6. #define p 31011
7. #define N 1003
8. **using** **namespace** std;
9. **int** a[12][12],c[N][N],n,m,vis[N],fa[N],U[N];
10. vector<**int**> V[N];
11. **struct** data{
12. **int** x,y,c;
13. **bool** operator<(**const** data &a)**const**{
14. **return** c<a.c;
15. }
16. }e[N];
17. **int** gauss(**int** n)
18. {
19. **for** (**int** i=1;i<=n;i++)
20. **for** (**int** j=1;j<=n;j++) a[i][j]%=p;
21. //for (int i=1;i<=n;i++,cout<<endl)
22. // for (int j=1;j<=n;j++) cout<<a[i][j]<<" ";
23. **int** ret=1;
24. **for** (**int** i=1;i<=n;i++) {
25. **int** num=i;
26. **for** (**int** j=i+1;j<=n;j++)
27. **if** (abs(a[j][i])) num=j;
28. **for** (**int** j=1;j<=n;j++) swap(a[num][j],a[i][j]);
29. **if** (num!=i) ret=-ret;
30. **for** (**int** j=i+1;j<=n;j++)
31. **while** (a[j][i]) {
32. **int** t=a[j][i]/a[i][i];
33. **for** (**int** k=1;k<=n;k++)
34. a[j][k]=(a[j][k]-t\*a[i][k])%p;
35. **if** (!a[j][i]) **break**;
36. ret=-ret;
37. **for** (**int** k=1;k<=n;k++) swap(a[i][k],a[j][k]);
38. }
39. ret=(ret\*a[i][i])%p;
40. }
41. //cout<<ret<<endl;
42. **return** (ret%p+p)%p;
43. }
44. **int** find(**int** x,**int** f[N])
45. {
46. **if** (x==f[x]) **return** x;
47. **else** **return** find(f[x],f);
48. }
49. **int** main()
50. {
51. freopen("bzoj\_1016.in","r",stdin);
52. freopen("bzoj\_1016.out","w",stdout);
53. scanf("%d%d",&n,&m);
54. **for** (**int** i=1;i<=m;i++) scanf("%d%d%d",&e[i].x,&e[i].y,&e[i].c);
55. sort(e+1,e+m+1);
56. **for** (**int** i=1;i<=n;i++) vis[i]=0,fa[i]=i;
57. **int** ans=1; **int** last=-1;
58. **for** (**int** i=1;i<=m+1;i++) {
59. **if** (e[i].c!=last||i==m+1) {
60. **for** (**int** j=1;j<=n;j++)
61. **if** (vis[j]) {
62. **int** r1=find(j,U);
63. V[r1].push\_back(j);
64. vis[j]=0;
65. }
66. **for** (**int** j=1;j<=n;j++)
67. **if** (V[j].size()>1) {
68. memset(a,0,**sizeof**(a));
69. **int** len=V[j].size();
70. **for** (**int** k=0;k<len;k++)
71. **for** (**int** l=k+1;l<len;l++) {
72. **int** x=V[j][k]; **int** y=V[j][l]; **int** t=c[x][y];
73. a[k+1][l+1]-=t; a[l+1][k+1]-=t;
74. a[k+1][k+1]+=t; a[l+1][l+1]+=t;
75. }
76. ans=ans\*gauss(len-1)%p;
77. **for** (**int** k=0;k<len;k++) fa[V[j][k]]=j;
78. }
79. **for** (**int** j=1;j<=n;j++) {
80. U[j]=fa[j]=find(j,fa);
81. V[j].clear();
82. }
83. last=e[i].c;
84. **if**(i==m+1) **break**;
85. }
86. **int** x=e[i].x; **int** y=e[i].y;
87. **int** r1=find(x,fa); **int** r2=find(y,fa);
88. **if** (r1==r2) **continue**;
89. U[find(r1,U)]=find(r2,U); vis[r1]=1; vis[r2]=1;
90. c[r1][r2]++; c[r2][r1]++;
91. }
92. **int** flag=1;
93. **for** (**int** i=2;i<=n;i++)
94. **if** (find(i,U)!=find(i-1,U)) flag=0;
95. ans=(ans\*flag%p+p)%p;
96. printf("%d\n",ans);
97. }

高斯消元

1. **void** gauss()
2. {
3. **int** limit=equ.size(),tot=0;
4. **int** j;
5. **for**(**int** i=0;i<limit;i++)
6. {
7. **for**(j=tot;j<limit&&!equ[j][i];j++);
8. **if**(j==limit) Free++;
9. **else**
10. {
11. swap(equ[tot++],equ[j]);
12. **for**(j=tot;j<limit;j++) **if**(equ[j][i])
13. equ[j]^=equ[tot-1];
14. }
15. }
16. **return**;
17. }