

2014-5-10

吕淑宝

东北林业大学ACM集训队

**ACM-ICPC算法模板**

# 图论

# [网络流 Dinic算法求最大流模板](http://blog.csdn.net/lvshubao1314/article/details/30030439)

1. #include<cstdio>
2. #include<iostream>
3. **using** **namespace** std;
4. **const** **int** oo=1e9;
5. /\*\*oo 表示无穷大\*/
6. **const** **int** mm=111111;
7. /\*\*mm 表示边的最大数量，记住要是原图的两倍，在加边的时候都是双向的\*/
8. **const** **int** mn=999;
9. /\*\*mn 表示点的最大数量\*/
10. **int** node,src,dest,edge;
11. /\*\*node 表示节点数，src 表示源点，dest 表示汇点，edge 统计边数\*/
12. **int** ver[mm],flow[mm],next[mm];
13. /\*\*ver 边指向的节点，flow 边的容量，next 链表的下一条边\*/
14. **int** head[mn],work[mn],dis[mn],q[mn];
15. /\*\*head 节点的链表头，work 用于算法中的临时链表头，dis 计算距离\*/
17. /\*\*初始化链表及图的信息\*/
18. **void** prepare(**int** \_node,**int** \_src,**int** \_dest)
19. {
20. node=\_node,src=\_src,dest=\_dest;
21. **for**(**int** i=0; i<node; ++i)head[i]=-1;
22. edge=0;
23. }
24. /\*\*增加一条u 到v 容量为c 的边\*/
25. **void** addedge(**int** u,**int** v,**int** c)
26. {
27. ver[edge]=v,flow[edge]=c,next[edge]=head[u],head[u]=edge++;
28. ver[edge]=u,flow[edge]=0,next[edge]=head[v],head[v]=edge++;
29. }
30. /\*\*广搜计算出每个点与源点的最短距离，如果不能到达汇点说明算法结束\*/
31. **bool** Dinic\_bfs()
32. {
33. **int** i,u,v,l,r=0;
34. **for**(i=0; i<node; ++i)dis[i]=-1;
35. dis[q[r++]=src]=0;
36. **for**(l=0; l<r; ++l)
37. **for**(i=head[u=q[l]]; i>=0; i=next[i])
38. **if**(flow[i]&&dis[v=ver[i]]<0)
39. {
40. /\*\*这条边必须有剩余容量\*/
41. dis[q[r++]=v]=dis[u]+1;
42. **if**(v==dest)**return** 1;
43. }
44. **return** 0;
45. }
46. /\*\*寻找可行流的增广路算法，按节点的距离来找，加快速度\*/
47. **int** Dinic\_dfs(**int** u,**int** exp)
48. {
49. **if**(u==dest)**return** exp;
50. /\*\*work 是临时链表头，这里用i 引用它，这样寻找过的边不再寻找\*/
51. **for**(**int** &i=work[u],v,tmp; i>=0; i=next[i])
52. **if**(flow[i]&&dis[v=ver[i]]==dis[u]+1&&(tmp=Dinic\_dfs(v,min(exp,flow[i])))>0)
53. {
54. flow[i]-=tmp;
55. flow[i^1]+=tmp;
56. /\*\*正反向边容量改变\*/
57. **return** tmp;
58. }
59. **return** 0;
60. }
61. **int** Dinic\_flow()
62. {
63. **int** i,ret=0,delta;
64. **while**(Dinic\_bfs())
65. {
66. **for**(i=0; i<node; ++i)work[i]=head[i];
67. **while**(delta=Dinic\_dfs(src,oo))ret+=delta;
68. }
69. **return** ret;
70. }

# [费用流模板 （最大最小费用流）](http://blog.csdn.net/lvshubao1314/article/details/30233851)

1. #include <iostream>
2. #include <cstdio>
4. **using** **namespace** std;
6. **const** **int** oo=1e9;//无穷大
7. **const** **int** maxm=1111111;//边的最大数量，为原图的两倍
8. **const** **int** maxn=2222;//点的最大数量
10. **int** node,src,dest,edge;//node节点数，src源点，dest汇点，edge边数
11. **int** head[maxn],p[maxn],dis[maxn],q[maxn],vis[maxn];//head链表头，p记录可行流上节点对应的反向边，dis计算距离
13. **struct** edgenode
14. {
15. **int** to;//边的指向
16. **int** flow;//边的容量
17. **int** cost;//边的费用
18. **int** next;//链表的下一条边
19. } edges[maxm];
21. **void** prepare(**int** \_node,**int** \_src,**int** \_dest);
22. **void** addedge(**int** u,**int** v,**int** f,**int** c);
23. **bool** spfa();
25. **inline** **int** min(**int** a,**int** b)
26. {
27. **return** a<b?a:b;
28. }
30. **inline** **void** prepare(**int** \_node,**int** \_src,**int** \_dest)
31. {
32. node=\_node;
33. src=\_src;
34. dest=\_dest;
35. **for** (**int** i=0; i<node; i++)
36. {
37. head[i]=-1;
38. vis[i]=**false**;
39. }
40. edge=0;
41. }
43. **void** addedge(**int** u,**int** v,**int** f,**int** c)
44. {
45. edges[edge].flow=f;
46. edges[edge].cost=c;
47. edges[edge].to=v;
48. edges[edge].next=head[u];
49. head[u]=edge++;
50. edges[edge].flow=0;
51. edges[edge].cost=-c;
52. edges[edge].to=u;
53. edges[edge].next=head[v];
54. head[v]=edge++;
55. }
57. **bool** spfa()
58. {
59. **int** i,u,v,l,r=0,tmp;
60. **for** (i=0; i<node; i++) dis[i]=oo;
61. dis[q[r++]=src]=0;
62. p[src]=p[dest]=-1;
63. **for** (l=0; l!=r; ((++l>=maxn)?l=0:1))
64. {
65. **for** (i=head[u=q[l]],vis[u]=**false**; i!=-1; i=edges[i].next)
66. {
67. **if** (edges[i].flow&&dis[v=edges[i].to]>(tmp=dis[u]+edges[i].cost))
68. {
69. dis[v]=tmp;
70. p[v]=i^1;
71. **if** (vis[v]) **continue**;
72. vis[q[r++]=v]=**true**;
73. **if** (r>=maxn) r=0;
74. }
75. }
76. }
77. **return** p[dest]>=0;
78. }
80. **int** spfaflow()
81. {
82. **int** i,ret=0,delta;
83. **while** (spfa())
84. {
85. //按记录原路返回求流量
86. **for** (i=p[dest],delta=oo; i>=0; i=p[edges[i].to])
87. {
88. delta=min(delta,edges[i^1].flow);
89. }
90. **for** (**int** i=p[dest]; i>=0; i=p[edges[i].to])
91. {
92. edges[i].flow+=delta;
93. edges[i^1].flow-=delta;
94. }
95. ret+=delta\*dis[dest];
96. }
97. **return** ret;
98. }

# [强连通分量+缩点（记录所缩点的个数）](http://blog.csdn.net/lvshubao1314/article/details/35797121)

1. </pre><pre name="code" **class**="cpp">#include <stdio.h>
2. #include <string.h>
3. #include <iostream>
4. #include <stack>
5. **using** **namespace** std;
6. #define M 10005
7. #define N 105
8. **struct** note
9. {
10. **int** v,next;
11. }edge[M];
12. **int** head[N],dfn[N],low[N],belong[N],index,ip,cnt\_tar,count[N],instack[N\*2];
13. stack <**int**> q;
14. **int** n;
15. **void** addedge(**int** u,**int** v)//新增一条边的操作
16. {
17. edge[ip].v=v,edge[ip].next=head[u],head[u]=ip++;
18. }
19. **void** tarjan(**int** u)
20. {
21. dfn[u]=low[u]=++index;//为节点u设定次序编号和low初值
22. q.push(u);//将节点u压入栈中
23. instack[u]=1;//点u在栈内
24. **for**(**int** i=head[u];i!=-1;i=edge[i].next)//枚举每一条边
25. {
26. **int** v=edge[i].v;
27. **if**(!dfn[v])//如果节点v未被访问过
28. {
29. tarjan(v);//继续向下找
30. low[u]=min(low[u],low[v]);
31. }
32. **else** **if**(instack[v])//如果节点还在栈内
33. low[u]=min(low[u],dfn[v]);
34. }
35. **if**(dfn[u]==low[u])//如果节点u是强连通分量的根
36. {
37. cnt\_tar++;//计数加一
38. **int** j;
39. **do**
40. {
41. j=q.top();//将v退栈
42. q.pop();
43. belong[j]=cnt\_tar;//该强连通分量的最后一个出栈的点作为一个缩点出现在新的待建的图中作为该强连通分量的代表
44. instack[j]=0;//点u出栈
45. count[cnt\_tar]++;//记录这一强连通分量中所包含的点的个数；
46. }**while**(j!=u);
47. }
48. }
49. **void** solve()
50. {
51. index=0,cnt\_tar=0;
52. memset(dfn,0,**sizeof**(dfn));
53. memset(low,0,**sizeof**(low));
54. memset(instack,0,**sizeof**(instack));
55. memset(count,0,**sizeof**(count));
56. **for**(**int** i=1;i<=n;i++)
57. {
58. **if**(!dfn[i])
59. tarjan(i);
60. }
61. }
62. //主函数里面调用solve函数就可以了

**无向图的缩点（缩点后所有的边为桥）**

1. /\*\*
2. hdu4612  连通性，求树的直径，加一边求最少桥
3. 题目大意：给定一个无向连通图加上一条边后所得到的图所含的桥的数目最少
4. 解题思路：tarjan缩点边树，求出树的直径m，则原有桥数-m即为所求。
5. 本题要注意考虑重边和注意求树的直径的方法（见代码注释）
6. \*直径:；任意两个节点之间的最长距离
7. \*/
8. #pragma comment(linker, "/STACK:1024000000,1024000000")///申请空间
9. #include <stdio.h>
10. #include <string.h>
11. #include <algorithm>
12. #include <iostream>
13. #include <vector>
14. **using** **namespace** std;
15. **const** **int** maxn=200010;
16. **const** **int** maxm=2000014;
18. **struct** note
19. {
20. **int** v,next;
21. **bool** cut,chong\_bian;
22. } edge[maxm];
24. **int** head[maxn],ip;
25. **int** n,m;
27. **void** init()
28. {
29. memset(head,-1,**sizeof**(head));
30. ip=0;
31. }
32. **void** addedge(**int** u,**int** v,**bool** chong\_bian)
33. {
34. edge[ip].v=v,edge[ip].cut=**false**,edge[ip].chong\_bian=chong\_bian,edge[ip].next=head[u],head[u]=ip++;
35. }
37. **int** dfn[maxn],low[maxn],dex,inst[maxn],st[maxn],top,cnt,belong[maxn];
39. **void** tarjan(**int** u,**int** pre,**bool** ff)///ff代表重边
40. {
41. dfn[u]=low[u]=++dex;
42. st[top++]=u;
43. inst[u]=1;
44. **for**(**int** i=head[u]; i!=-1; i=edge[i].next)
45. {
46. **int** v=edge[i].v;
47. **if**(v==pre&&(!ff))**continue**;///与没有重边的区别
48. **if**(!dfn[v])
49. {
50. tarjan(v,u,edge[i].chong\_bian);
51. **if**(low[u]>low[v])low[u]=low[v];
52. **if**(low[v]>dfn[u])
53. {
54. ///bridge++;
55. edge[i].cut=**true**;
56. edge[i^1].cut=**true**;
57. }
58. }
59. **else** **if**(inst[v]&&dfn[v]<low[u])
60. {
61. low[u]=dfn[v];
62. }
63. }
64. **if**(dfn[u]==low[u])
65. {
66. cnt++;
67. **int** v;
68. **do**
69. {
70. v=st[--top];
71. inst[v]=0;
72. belong[v]=cnt;
73. }
74. **while**(v!=u);
75. }
76. }
78. vector <**int**> vec[maxn];
79. **int** dep[maxn];
81. **void** dfs(**int** u)///dfs求每个节点的深度
82. {
83. **for**(**int** i=0; i<vec[u].size(); i++)
84. {
85. **int** v=vec[u][i];
86. **if**(dep[v]!=-1)**continue**;
87. dep[v]=dep[u]+1;
88. dfs(v);
89. }
90. }
92. **void** solve()
93. {
94. memset(dfn,0,**sizeof**(dfn));
95. memset(inst,0,**sizeof**(inst));
96. cnt=dex=top=0;
97. tarjan(1,0,**false**);
98. **for**(**int** i=1;i<=cnt;i++)
99. vec[i].clear();
100. **for**(**int** u=1; u<=n; u++)
101. {
102. **for**(**int** i=head[u]; i!=-1; i=edge[i].next)
103. {
104. **if**(edge[i].cut)
105. {
106. **int** v=edge[i].v;
107. vec[belong[u]].push\_back(belong[v]);
108. }
109. }
110. }
111. ///=======求直径=========
112. memset(dep,-1,**sizeof**(dep));
113. dep[1]=0;
114. dfs(1);
115. **int** k=1;
116. **for**(**int** i=1; i<=cnt; i++)
117. {
118. **if**(dep[i]>dep[k])
119. k=i;
120. }
121. memset(dep,-1,**sizeof**(dep));
122. dep[k]=0;
123. dfs(k);
124. **int** ans=0;
125. **for**(**int** i=1; i<=cnt; i++)
126. {
127. ans=max(ans,dep[i]);
128. }
129. ///=======================
130. printf("%d\n",cnt-1-ans);
131. }
133. **struct** note\_
134. {
135. **int** x,y;
136. } node[maxm];
138. **bool** cmp(note\_ a,note\_ b)
139. {
140. **if**(a.x==b.x)
141. **return** a.y<b.y;
142. **return** a.x<b.x;
143. }
145. **int** main()
146. {
147. **while**(~scanf("%d%d",&n,&m))
148. {
149. **if**(n==0&&m==0)**break**;
150. **for**(**int** i=0; i<m; i++)
151. {
152. **int** u,v;
153. scanf("%d%d",&u,&v);
154. **if**(u==v)**continue**;
155. **if**(u>v)swap(u,v);
156. node[i].x=u;
157. node[i].y=v;
158. }
159. sort(node,node+m,cmp);
160. init();
161. **for**(**int** i=0; i<m; i++)
162. {
163. **if**(i==0||(node[i].x!=node[i-1].x||node[i].y!=node[i-1].y))
164. {
165. **if**(i<m-1&&node[i].x==node[i+1].x&&node[i].y==node[i+1].y)
166. {
167. addedge(node[i].x,node[i].y,**true**);
168. addedge(node[i].y,node[i].x,**true**);
169. }
170. **else**
171. {
172. addedge(node[i].x,node[i].y,**false**);
173. addedge(node[i].y,node[i].x,**false**);
174. }
175. }
176. }
177. solve();
178. }
179. **return** 0;
180. }

**[2-SAT 模板](http://blog.csdn.net/lvshubao1314/article/details/36008829)**

1. </pre><pre name="code" **class**="cpp">/\*==============================================\*\
2. 方法一：即为大白书上的描述
3. \\*==============================================\*/
4. **struct** TwoSAT
5. {
6. **int** head[maxn\*2];
7. EdgeNode edges[maxm\*2];
8. **int** edge,n;
9. **bool** mark[maxn\*2];
10. **int** S[maxn\*2],c;
11. **void** init(**int** n)
12. {
13. **this**->n=n;
14. memset(mark,0,**sizeof**(mark));
15. memset(head,-1,**sizeof**(head));
16. edge=0;
17. }
18. **void** addedge(**int** u,**int** v)
19. {
20. edges[edge].to=v,edges[edge].next=head[u],head[u]=edge++;
21. }
22. // x = xval or y = yval
23. **void** add\_clause(**int** x,**int** xval,**int** y,**int** yval)
24. {
25. x=x\*2+xval;
26. y=y\*2+yval;
27. addedge(x^1,y);  //x^1可以推出y就建立一条对应的边，下同
28. addedge(y^1,x);
29. }
30. // x = xval
31. **void** add\_con(**int** x,**int** xval)
32. {
33. x=x\*2+xval;
34. addedge(x^1,x);
35. }
36. **bool** dfs(**int** x)
37. {
38. **if** (mark[x^1]) **return** **false**; //节点已被访问，下同
39. **if** (mark[x]) **return** **true**;
40. mark[x]=**true**;  //标记xi
41. S[c++]=x;
42. **for** (**int** i=head[x]; i!=-1; i=edges[i].next) //邻接边搜索
43. **if** (!dfs(edges[i].to)) **return** **false**;
44. **return** **true**;
45. }
46. **bool** solve()
47. {
48. **for** (**int** i=0; i<n\*2; i+=2)
49. **if** (!mark[i]&&!mark[i+1])
50. {
51. c=0;
52. **if** (!dfs(i))//如果标记xi为假不成立，则从头标记xi为真
53. {
54. **while** (c>0) mark[S[--c]]=**false**;//返回假设的起点
55. **if** (!dfs(i+1)) **return** **false**;//判断是否成立
56. }
57. }
58. **return** **true**;
59. }
60. };
61. /\*==============================================\*\
62. 方法二：利用连通性，xi和xj不能再一个强连通分量里面
63. \\*==============================================\*/
64. //=============================================================
65. //2-sat模板
66. **const** **int** maxn=505;
67. **const** **int** maxm=100005;
68. **int** n,m,a[maxm][3];
69. **struct** note
70. {
71. **int** to;
72. **int** next;
73. }edge[maxn\*2\*maxn];
74. **int** head[maxn];
75. **int** ip;
76. **int** dfn[maxn],low[maxn],sccno[maxn],cnt,scc,instack[maxn];
77. stack<**int**>stk;
78. **void** init()
79. {
80. memset(head,-1,**sizeof**(head));
81. ip=0;
82. }
83. **void** addedge(**int** u,**int** v)
84. {
85. edge[ip].to=v,edge[ip].next=head[u],head[u]=ip++;
86. }
87. // x = xval or y = yval
88. //表示[x][xal]和[y][yal]不可同时出现
89. **void** add\_cluse(**int** x,**int** xval,**int** y,**int** yval)
90. {
91. x=x\*2+xval;
92. y=y\*2+yval;
93. addedge(x,y^1);
94. addedge(y,x^1);
95. }
96. **void** dfs(**int** u)
97. {
98. dfn[u]=low[u]=++scc;
99. stk.push(u);
100. instack[u]=1;
101. **for** (**int** i=head[u]; i!=-1; i=edge[i].next)
102. {
103. **int** v=edge[i].to;
104. **if** (!dfn[v])
105. {
106. dfs(v);
107. low[u]=min(low[u],low[v]);
108. }
109. **else** **if** (instack[v])
110. low[u]=min(low[u],dfn[v]);
111. }
112. **if** (low[u]==dfn[u])
113. {
114. cnt++;
115. **int** x;
116. **do**
117. {
118. x=stk.top();
119. stk.pop();
120. sccno[x]=cnt;
121. instack[x]=0;
122. }**while** (x!=u);
123. }
124. }
125. **bool** solve()
126. {
127. scc=cnt=0;
128. memset(sccno,0,**sizeof**(sccno));
129. memset(dfn,0,**sizeof**(dfn));
130. memset(low,0,**sizeof**(low));
131. memset(instack,0,**sizeof**(instack));
132. **while** (!stk.empty()) stk.pop();
133. **for** (**int** i=0; i<2\*n; i++) **if** (!dfn[i]) dfs(i);
134. **for** (**int** i=0; i<2\*n; i+=2)
135. {
136. **if** (sccno[i]==sccno[i^1]) **return** **false**;
137. }
138. **return** **true**;
139. }
140. //solve返回true证明2-sat有解，否则没解
141. //=============================================

**[最小树形图（有向图的最小生成树）朱刘算法](http://blog.csdn.net/lvshubao1314/article/details/37994367)**

1. //======================================================================
2. //最小树形图（有向图的最小生成树）朱刘算法模板
3. **const** **int** N=101,M=10001,inf=2147483647;
4. **struct** edge
5. {
6. **int** u,v;
7. **double** w;
8. } e[M];
9. **struct** node
10. {
11. **double** x,y;
12. } p[N];
13. **int** n,m,id[N],pre[N],v[N];
14. **double** inw[N];
15. **double** ans;
16. **void** zhu\_liu(**int** root)
17. {
18. **int** s,t,idx=n;
19. ans=0;
20. **while** (idx)
21. {
22. **for** (**int** i=1; i<=n; ++i) inw[i]=inf,id[i]=-1,v[i]=-1;
23. **for** (**int** i=1; i<=m; ++i)
24. {
25. s=e[i].u;
26. t=e[i].v;
27. **if** (e[i].w>inw[t] || s==t) **continue**;
28. pre[t]=s;
29. inw[t]=e[i].w;
30. }
31. inw[root]=0;
32. pre[root]=root;
33. **for** (**int** i=1; i<=n; ++i)
34. {
35. **if** (inw[i]==inf)
36. {
37. printf("poor snoopy\n");
38. **return**;
39. }
40. ans+=inw[i];
41. }
42. idx=0;
43. **for** (**int** i=1; i<=n; ++i)
44. **if** (v[i]==-1)
45. {
46. t=i;
47. **while** (v[t]==-1) v[t]=i,t=pre[t];
48. **if** (v[t]!=i || t==root) **continue**;
49. id[t]=++idx;
50. **for** (s=pre[t]; s!=t; s=pre[s]) id[s]=idx;
51. }
52. **if** (idx==0) **continue**;
53. **for** (**int** i=1; i<=n; ++i)
54. **if** (id[i]==-1) id[i]=++idx;
55. **for** (**int** i=1; i<=m; ++i)
56. {
57. e[i].w-=inw[e[i].v];
58. e[i].u=id[e[i].u];
59. e[i].v=id[e[i].v];
60. }
61. n=idx;
62. root=id[root];
63. }
64. printf("%.2lf\n",ans);
65. }
66. //==============================================================

**[二分图的最大匹配的匈牙利算法](http://blog.csdn.net/lvshubao1314/article/details/39829217)**

1. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. //二分图匹配（匈牙利算法的DFS实现）
3. //初始化：g[][]两边顶点的划分情况
4. //建立g[i][j]表示i->j的有向边就可以了，是左边向右边的匹配
5. //g没有边相连则初始化为0
6. //uN是匹配左边的顶点数，vN是匹配右边的顶点数
7. //调用：res=hungary();输出最大匹配数
8. //优点：适用于稠密图，DFS找增广路，实现简洁易于理解
9. //时间复杂度:O(VE)
10. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
11. //顶点编号从0开始的
12. **const** **int** MAXN=220;
13. **int** uN,vN;//u,v数目
14. **int** g[MAXN][MAXN];
15. **int** linker[MAXN];
16. **bool** used[MAXN];
17. **bool** dfs(**int** u)//从左边开始找增广路径
18. {
19. **int** v;
20. **for**(v=0;v<vN;v++)//这个顶点编号从0开始，若要从1开始需要修改
21. **if**(g[u][v]&&!used[v])
22. {
23. used[v]=**true**;
24. **if**(linker[v]==-1||dfs(linker[v]))
25. {//找增广路，反向
26. linker[v]=u;
27. **return** **true**;
28. }
29. }
30. **return** **false**;//这个不要忘了，经常忘记这句
31. }
32. **int** hungary()
33. {
34. **int** res=0;
35. **int** u;
36. memset(linker,-1,**sizeof**(linker));
37. **for**(u=0;u<uN;u++)
38. {
39. memset(used,0,**sizeof**(used));
40. **if**(dfs(u)) res++;
41. }
42. **return** res;
43. }
44. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**无向图的割点与桥**

1. /\*\*
2. \*  求 无向图的割点和桥
3. \*  可以找出割点和桥，求删掉每个点后增加的连通块。
4. \*  需要注意重边的处理，可以先用矩阵存，再转邻接表，或者进行判重
5. \*  调用solve输出割点数，全局变量bridge记录边的个数
6. \*/
7. #include <stdio.h>
8. #include <string.h>
9. #include <algorithm>
10. #include <iostream>
11. **using** **namespace** std;
12. **const** **int** maxn=10010;
13. **const** **int** maxm=100010;
15. **struct** note
16. {
17. **int** v,next;
18. **bool** cut;///是否为桥的标记
19. }edge[maxm];
21. **int** head[maxn],ip;
23. **void** init()
24. {
25. memset(head,-1,**sizeof**(head));
26. ip=0;
27. }
29. **int** low[maxn],dfn[maxn],st[maxn],dex,top;
30. **bool** in\_st[maxn],cut[maxn];
31. **int** add\_block[maxn];///删除一个点后增加的连通块
32. **int** bridge;
33. **int** n;
35. **void** addedge(**int** u,**int** v)
36. {
37. edge[ip].v=v,edge[ip].next=head[u],head[u]=ip++;
38. }
40. **void** tarjan(**int** u,**int** pre)
41. {
42. low[u]=dfn[u]=++dex;
43. st[top++]=u;
44. in\_st[u]=**true**;
45. **int** son=0;
46. **for**(**int** i=head[u];i!=-1;i=edge[i].next)
47. {
48. **int** v=edge[i].v;
49. **if**(v==pre)**continue**;
50. **if**(!dfn[v])
51. {
52. son++;
53. tarjan(v,u);
54. **if**(low[u]>low[v])low[u]=low[v];
55. ///桥
56. ///一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足DFS(u)<Low(v)。
57. **if**(low[v]>dfn[u])
58. {
59. bridge++;
60. edge[i].cut=**true**;
61. edge[i^1].cut=**true**;
62. }
63. ///割点
64. ///一个顶点u是割点，当且仅当满足(1)或(2) (1) u为树根，且u有多于一个子树。
65. ///(2) u不为树根，且满足存在(u,v)为树枝边(或称父子边，
66. ///即u为v在搜索树中的父亲)，使得DFS(u)<=Low(v)
67. **if**(u!=pre&&low[v]>=dfn[u])///不是树根
68. {
69. cut[u]=**true**;
70. add\_block[u]++;
71. }
72. }
73. **else** **if**(low[u]>dfn[v])
74. low[u]=dfn[v];
75. }
76. ///树根，需满足条件分支数大于1
77. **if**(u==pre&&son>1)cut[u]=**true**;
78. **if**(u==pre)add\_block[u]=son-1;
79. in\_st[u]=**false**;
80. top--;
81. }
83. **void** solve()
84. {
85. memset(dfn,0,**sizeof**(dfn));
86. memset(in\_st,**false**,**sizeof**(in\_st));
87. memset(add\_block,0,**sizeof**(add\_block));
88. memset(cut,**false**,**sizeof**(cut));
89. dex=top=0;
90. bridge=0;
91. **for**(**int** i=1;i<=n;i++)
92. {
93. **if**(!dfn[i])
94. tarjan(i,i);
95. }
96. **int** ans=0;
97. **for**(**int** i=1;i<=n;i++)
98. {
99. **if**(cut[i])
100. ans++;
101. }
102. printf("%d\n",ans);
103. }

**KM算法二分图的完美匹配**

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
3. KM算法模板C++
4. 作用：
5. 求二分图的最佳匹配
6. 注意：
7. (1)nx=ny=n，for (i：1~n)for (j：1~n)scanf (w[i][j]);
8. w[i][j],表示左边第i点匹配右边第j点的价值。i，j：从1开始。
9. 主函数调用:ans=KM(); ans的值即为所求。
10. (2)所求为最大完备匹配，若是求最小，则把边的权值取相反数，跑一遍模板，
11. 最后结果再取相反数即可。
12. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
13. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
14. #include <stdio.h>
15. #include <string.h>
16. #define M 310
17. #define inf 0x3f3f3f3f
19. **int** n,nx,ny;
20. **int** link[M],lx[M],ly[M],slack[M];///lx,ly为顶标，nx,ny分别为x点集y点集的个数
21. **int** visx[M],visy[M],w[M][M];
23. **int** DFS(**int** x)
24. {
25. visx[x] = 1;
26. **for** (**int** y = 1; y <= ny; y ++)
27. {
28. **if** (visy[y]) ontinue;
29. **int** t = lx[x] + ly[y] - w[x][y];
30. **if** (t == 0)
31. {
32. visy[y] = 1;
33. **if** (link[y] == -1||DFS(link[y]))
34. {
35. link[y] = x;
36. **return** 1;
37. }
38. }
39. **else** **if** (slack[y] > t)  ///不在相等子图中slack 取最小的
40. slack[y] = t;
41. }
42. **return** 0;
43. }
44. **int** KM()
45. {
46. **int** i,j;
47. memset (link,-1,**sizeof**(link));
48. memset (ly,0,**sizeof**(ly));
49. **for** (i = 1; i <= nx; i ++)          ///lx初始化为与它关联边中最大的
50. **for** (j = 1,lx[i] = -inf; j <= ny; j ++)
51. **if** (w[i][j] > lx[i])
52. lx[i] = w[i][j];
54. **for** (**int** x = 1; x <= nx; x ++)
55. {
56. **for** (i = 1; i <= ny; i ++)
57. slack[i] = inf;
58. **while** (1)
59. {
60. memset (visx,0,**sizeof**(visx));
61. memset (visy,0,**sizeof**(visy));
62. **if** (DFS(x))     ///若成功（找到了增广轨），则该点增广完成，进入下一个点的增广
63. **break**;  ///若失败（没有找到增广轨），则需要改变一些点的标号，使得图中可行边的数量增加。
64. ///方法为：将所有在增广轨中（就是在增广过程中遍历到）的X方点的标号全部减去一个常数d，
65. ///所有在增广轨中的Y方点的标号全部加上一个常数d
66. **int** d = inf;
67. **for** (i = 1; i <= ny; i ++)
68. **if** (!visy[i]&&d > slack[i])
69. d = slack[i];
70. **for** (i = 1; i <= nx; i ++)
71. **if** (visx[i])
72. lx[i] -= d;
73. **for** (i = 1; i <= ny; i ++) ///修改顶标后，要把所有不在交错树中的Y顶点的slack值都减去d
74. **if** (visy[i])
75. ly[i] += d;
76. **else**
77. slack[i] -= d;
78. }
79. }
80. **int** res = 0;
81. **for** (i = 1; i <= ny; i ++)
82. **if** (link[i] > -1)
83. res += w[link[i]][i];
84. **return** res;
85. }

**最小生成树**

并查集的方式，链式前向星存储

1. #include <string.h>
2. #include <stdio.h>
3. #include <iostream>
4. #include <algorithm>
5. **using** **namespace** std;
6. **struct** note
7. {
8. **int** start;
9. **int** end;
10. **int** w;
11. }edge[1005];
12. **int** pa[1005];
13. **int** m,n,k,p;
14. **bool** cmp(note a,note b)
15. {
16. **if**(a.w<b.w)
17. **return** **true**;
18. **return** **false**;
19. }
20. **void** make\_set()
21. {
22. **for**(**int** x=0;x<n;x++)
23. pa[x]=x;
24. }
25. **int** find(**int** x)
26. {
27. **if**(x!=pa[x])
28. **return** pa[x]=find(pa[x]);
29. **return** pa[x];
30. }
31. **int** kruskal()
32. {
33. **int** i,ans=0;
34. make\_set();
35. sort(edge,edge+p,cmp);
36. **for**(i=0;i<p;i++)
37. {
38. **int** x=find(edge[i].start);
39. **int** y=find(edge[i].end);
40. **if**(x!=y)
41. {
42. pa[y]=x;
43. ans+=edge[i].w;
44. }
45. }
46. **return** ans;
47. }
48. **int** main()
49. {
50. **char** x,y;
51. **while**(cin >> n)
52. {
53. p=0;
54. **if**(n==0)
55. **break**;
56. **for**(**int** i=0;i<n-1;i++)
57. {
58. cin>>x>> m;
59. **for**(**int** j=0;j<m;j++)
60. {
61. cin >> y >> k;
62. edge[p].start=x-'A';
63. edge[p].end=y-'A';
64. edge[p].w=k;
65. p++;
66. }
67. }
68. printf("%d\n",kruskal());
69. }
70. **return** 0;
71. }

**最短路径**

**Floyd 算法求多源最短路径**

1. #include <string.h>
2. #include <stdio.h>
3. #include <iostream>
4. **using** **namespace** std;
5. **const** **int** INF=1e9;
6. **int** map[1005][1005];
7. **int** m,n;
8. **void** chuzhi()//初始化数组，令其值为无穷大
9. {
10. **for**(**int** i=0;i<=n;i++)
11. **for**(**int** j=0;j<=m;j++)
12. map[i][j]=INF;
13. }
14. **int** main()
15. {
16. **int** x,y,a,b,w;
17. **while**(~scanf("%d%d",&n,&m))//n为节点数，m为路径数
18. {
19. chuzhi();//初始化数组
20. **for**(**int** i=0;i<m;i++)
21. {
22. cin >> a>> b>> w;
23. map[a][b]=map[b][a]=w;//采用邻接矩阵的方式存储数据
24. }
25. **for**(**int** k=1;k<=n;k++)//Floyd算法的核心部分，值得一提的是要注意三个循环的前后顺序
26. **for**(**int** i=1;i<=n;i++)
27. **for**(**int** j=1;j<=n;j++)
28. map[i][j]=min(map[i][j],map[i][k]+map[k][j]);
29. **for**(**int** i=1;i<=n;i++)//任意给出n组数据，输出其最短路径
30. {
31. cin >> x>>y;
32. printf("%d\n",map[x][y]);
33. }
34. }
35. **return** 0;
36. }

**[单源最短路spfa模板（stl更新版）](http://blog.csdn.net/lvshubao1314/article/details/39474411)**

1. #include <iostream>
2. #include <cstdio>
3. #include <cmath>
4. #include <queue>
5. #include <string.h>
6. **using** **namespace** std;
8. **const** **int** INF=0x3f3f3f3f;
9. **const** **int** maxm=511111;
10. **const** **int** maxn=111111;
12. **struct** EdgeNode
13. {
14. **int** to;
15. **int** w;
16. **int** next;
17. };
19. EdgeNode edges[maxm];
20. **int** N,M;
21. **int** head[maxn],edge;
22. **bool** vis[maxn];
23. queue <**int**> que;
24. **int** dis[maxn];
26. **void** addedge(**int** u,**int** v,**int** c)
27. {
28. edges[edge].w=c,edges[edge].to=v,edges[edge].next=head[u],head[u]=edge++;
29. }
31. **void** init()
32. {
33. memset(head,-1,**sizeof**(head));
34. edge=0;
35. }
37. **void** spfa(**int** s,**int** n)//单源最短路(s为起点，n为节点总数)
38. {
39. **int** u;
40. **for** (**int** i=0; i<=n; i++)
41. dis[i]=INF;
42. memset(vis,0,**sizeof**(vis));
43. **while** (!que.empty()) que.pop();
44. que.push(s);
45. vis[s]=**true**;
46. dis[s]=0;
47. **while** (!que.empty())
48. {
49. u=que.front();
50. que.pop();
51. vis[u]=**false**;
52. **for** (**int** i=head[u]; i!=-1; i=edges[i].next)
53. {
54. **int** v=edges[i].to;
55. **int** w=edges[i].w;
56. **if** (dis[v]>dis[u]+w)
57. {
58. dis[v]=dis[u]+w;
59. **if** (!vis[v])
60. {
61. vis[v]=**true**;
62. que.push(v);
63. }
64. }
65. }
66. }
67. }

**拓扑排序**

1. #include <stdio.h>
2. #include <string.h>
3. #include <iostream>
4. #include <algorithm>
5. #include <queue>
6. **using** **namespace** std;
7. **const** **int** maxn=30;
9. **int** head[maxn],ip,indegree[maxn];
10. **int** n,m,seq[maxn];
12. **struct** note
13. {
14. **int** v,next;
15. } edge[maxn\*maxn];
17. **void** init()
18. {
19. memset(head,-1,**sizeof**(head));
20. ip=0;
21. }
23. **void** addedge(**int** u,**int** v)
24. {
25. edge[ip].v=v,edge[ip].next=head[u],head[u]=ip++;
26. }
28. **int** topo()///拓扑，可做模板
29. {
30. queue<**int**>q;
31. **int** indeg[maxn];
32. **for**(**int** i=0; i<n; i++)
33. {
34. indeg[i]=indegree[i];
35. **if**(indeg[i]==0)
36. q.push(i);
37. }
38. **int** k=0;
39. **bool** res=**false**;
40. **while**(!q.empty())
41. {
42. **if**(q.size()!=1)res=**true**;
43. **int** u=q.front();
44. q.pop();
45. seq[k++]=u;
46. **for**(**int** i=head[u]; i!=-1; i=edge[i].next)
47. {
48. **int** v=edge[i].v;
49. indeg[v]--;
50. **if**(indeg[v]==0)
51. q.push(v);
52. }
53. }
54. **if**(k<n)**return** -1;///存在有向环不能进行拓扑排序
55. **if**(res)**return** 0;///可以进行拓扑排序，并且只有唯一一种方式，seq数组即是排序完好的序列
56. **return** 1;///可以进行拓扑排序，有多种情况，seq数组是其中一种序列
57. }

**数据结构**

**并查集**

1. #include <stdio.h>
2. #include <iostream>
3. #include <string.h>
4. **using** **namespace** std;
5. **int** n,m,k,fa[30010],num[30010],x,y;
6. **void** initset()//初始状态树的所有父亲节点都是其本身，该树的所含元素为1；
7. {
8. **for**(**int** i=0;i<n;i++)
9. {
10. fa[i]=i;
11. num[i]=1;
12. }
13. }
14. **int** find(**int** x)//寻找该点的根节点
15. {
16. **if**(x==fa[x])
17. **return** x;
18. **return** fa[x]=find(fa[x]);
19. }
20. **void** un(**int** x,**int** y)//若二者的根节点不同，则连到一起作为一个集合，该集合所含元素为先前二者之和
21. {
22. x=find(x);
23. y=find(y);
24. **if**(x!=y)
25. {
26. fa[y]=x;
27. num[x]+=num[y];
28. }
29. }
31. **int** main()
32. {
33. **while**(~scanf("%d%d",&n,&m))
34. {
35. **if**(n==0&&m==0)
36. **break**;
37. initset();
38. **for**(**int** i=0;i<m;i++)
39. {
40. scanf("%d",&k);
41. scanf("%d",&x);//输入一组数该组数一定在同一棵树上
42. **for**(**int** j=1;j<k;j++)
43. {
44. scanf("%d",&y);
45. un(x,y);
46. }
47. }
48. printf("%d\n",num[find(0)]);//树出和0有共同节点的树（即与0同一棵树）里的元素的个数//此处用num[a[0]]就错，我不知道为什么
49. }
50. **return** 0;
51. }

**优先队列**

STL中的优先队列的含义就是在队列出队时首先出队的是数值最大的那个元素。优先队列的头文件在<queue>中，其基本操作有：

1，定义：priority\_queue<类型> 名称，eg：priority\_queue<int >que;   -> 定义了一个名为que的元素类型为基本整型的优先队列

2，插入元素：que.push(要插入的元素)；

3，出队：que.top();

4，删除队头元素 que.pop();

5，que.empty();队列为空返回真

6，返回队列中元素的个数  que.size();

另外：优先队列的优先级默认的是大的数优先，有时候需要自定义优先级。

1. **struct** cmp//小的数优先
2. {
3. **bool** operator ()(**int** x,**int** y)
4. {
5. **return** x>y;
6. }
7. };
8. priority\_queue<**int**,vector<**int**>,cmp>que;
9. **struct** node//结构体的优先顺序自定义
10. {
11. **int** x, y;
12. **friend** **bool** operator < (node a, node b)
13. {
14. **return** a.x > b.x; //结构体中，x小的优先级高
15. }
16. };
17. priority\_queue<node>q;
18. #include <stdio.h>
19. #include <string.h>
20. #include <iostream>
21. #include <algorithm>
22. #include <queue>
23. **using** **namespace** std;
24. **struct** note
25. {
26. **int** a,b;
27. }aa[10005];
28. **bool** cmp(note a,note b)
29. {
30. **return** a.a<b.a;
31. }
32. **int** n,l,p;
33. **void** solve()
34. {
35. aa[n].a=l;
36. aa[n++].b=0;
37. priority\_queue<**int**> que;
38. **int** pos=0,tank=p,ans=0;
39. **for**(**int** i=0;i<n;i++)
40. {
41. **int** d=aa[i].a-pos;
42. **while**(tank-d<0)
43. {
44. **if**(que.empty())
45. {
46. puts("-1");
47. **return**;
48. }
49. tank+=que.top();
50. que.pop();
51. ans++;
52. }
53. tank-=d;
54. pos=aa[i].a;
55. que.push(aa[i].b);
56. }
57. printf("%d\n",ans);
58. }
59. **int** main()
60. {
61. **while**(~scanf("%d",&n))
62. {
63. **for**(**int** i=0;i<n;i++)
64. scanf("%d%d",&aa[i].a,&aa[i].b);
65. scanf("%d%d",&l,&p);
66. **for**(**int** i=0;i<n;i++)
67. aa[i].a=l-aa[i].a;
68. sort(aa,aa+n,cmp);
69. solve();
70. }
71. **return** 0;
72. }

**next\_permutation（全排列）**

**如果存在a之后的排列，就返回true。如果a是最后一个排列没有后继，返回false，每执行一次，a就变成它的后继**

1. **bool** solve()
2. {
3. sort(a,a+4);
4. **do**
5. {
6. **//各种操作**
7. }**while**(next\_permutation(a,a+4));//生成全排列的函数

**LCA算法-最近公共祖先**

**1，在线算法**

1. #include <cstdio>
2. #include <iostream>
3. #include <cstring>
4. #include <algorithm>
6. **using** **namespace** std;
7. **const** **int** INF=0x3f3f3f;
8. **const** **int** maxn=111111;
9. **const** **int** maxm=111111;
10. **int** n,m;
12. **struct** EDGENODE{
13. **int** to;
14. **int** w;
15. **int** next;
16. };
17. **struct** SGRAPH{
18. **int** head[maxn];
19. EDGENODE edges[maxm];
20. **int** edge;
21. **void** init(){
22. memset(head,-1,**sizeof**(head));
23. edge=0;
24. }
25. **void** addedge(**int** u,**int** v,**int** c){
26. edges[edge].w=c,edges[edge].to=v,edges[edge].next=head[u],head[u]=edge++;
27. }
28. //------------
29. **int** d[maxn][20];
30. //元素从1编号到n
31. **void** makeRmqIndex(**int** A[],**int** n){
32. **for**(**int** i=1;i<=n;i++) d[i][0]=i;
33. **for**(**int** j=1;(1<<j)<=n;j++)
34. **for**(**int** i=1;i+(1<<j)-1<=n;i++)
35. d[i][j] = A[d[i][j-1]] < A[d[i+(1<<(j-1))][j-1]]? d[i][j-1]:d[i+(1<<(j-1))][j-1];
36. }
37. **int** rmqIndex(**int** L,**int** R,**int** A[])
38. {
39. **int** k=0;
40. **while** ((1<<(k+1))<=R-L+1) k++;
41. **return** A[d[L][k]]<A[d[R-(1<<k)+1][k]]? d[L][k]:d[R-(1<<k)+1][k];
42. }
43. //---------------------
44. **int** E[maxn\*2],R[maxn],D[maxn\*2],mn;
45. **void** dfs(**int** u,**int** p,**int** d){
46. E[++mn]=u;
47. D[mn]=d;
48. R[u]=mn;
49. **for** (**int** i=head[u];i!=-1;i=edges[i].next){
50. **int** v=edges[i].to;
51. **if** (v==p) **continue**;
52. dfs(v,u,d+1);
53. E[++mn]=u;
54. D[mn]=d;
55. }
56. }
57. **void** LCA\_init(){
58. mn=0;
59. memset(R,0,**sizeof**(R));
60. dfs(1,-1,1);
61. makeRmqIndex(D,mn);
62. getd(1,-1,0);
63. }
64. **int** LCA(**int** u,**int** v){
65. **if** (R[u]>=R[v]) **return** E[rmqIndex(R[v],R[u],D)];
66. **else** **return** E[rmqIndex(R[u],R[v],D)];
68. }
69. //--------------------
70. **int** deep[maxn];
71. **void** getd(**int** u,**int** p,**int** w){
72. deep[u]=w;
73. **for** (**int** i=head[u];i!=-1;i=edges[i].next){
74. **int** v=edges[i].to;
75. **if** (v==p) **continue**;
76. getd(v,u,w+edges[i].w);
77. }
78. }
79. **int** getDis(**int** u,**int** v){
80. **int** lca=LCA(u,v);
81. **return** deep[u]+deep[v]-deep[lca]\*2;
82. }
83. **int** done(**int** x,**int** y,**int** z){
84. **int** ans=INF,res=0;
85. **int** lca1,lca2;
87. lca1=LCA(x,y);
88. res=deep[x]+deep[y]-deep[lca1]\*2;
89. lca2=LCA(lca1,z);
90. res+=deep[lca1]+deep[z]-deep[lca2]\*2;
91. ans=min(ans,res);
93. lca1=LCA(x,z);
94. res=deep[x]+deep[z]-deep[lca1]\*2;
95. lca2=LCA(lca1,y);
96. res+=deep[lca1]+deep[y]-deep[lca2]\*2;
97. ans=min(ans,res);
99. lca1=LCA(y,z);
100. res=deep[y]+deep[z]-deep[lca1]\*2;
101. lca2=LCA(lca1,x);
102. res+=deep[lca1]+deep[x]-deep[lca2]\*2;
103. ans=min(ans,res);
105. **return** ans;
106. }
107. }solver;

**2，离线算法**

1. /\*\*
2. LCA（离线算法）
3. 主函数除建边外还应调用
4. init();
5. dir[1]=0;
6. tarjan(1);
7. \*/
8. #include <stdio.h>
9. #include <string.h>
10. #include <iostream>
11. #include <algorithm>
12. **using** **namespace** std;
13. **const** **int** maxn=40010;
15. **struct** note
16. {
17. **int** u,v,w,lca,next;
18. }edge[maxn\*2],edge1[805];
20. **int** head[maxn],ip,head1[maxn],ip1;///需要建两次边。1，该树的边2，需要查询的两点
21. **int** m,n;
22. **int** father[maxn],vis[maxn],dir[maxn];
23. ///依次表示u点的祖先、标记是否访问过，到根节点的距离
25. **void** init()
26. {
27. memset(vis,0,**sizeof**(vis));
28. memset(dir,0,**sizeof**(dir));
29. memset(head,-1,**sizeof**(head));
30. memset(head1,-1,**sizeof**(head1));
31. ip=ip1=0;
32. }
34. **void** addedge(**int** u,**int** v,**int** w)
35. {
36. edge[ip].v=v,edge[ip].w=w,edge[ip].next=head[u],head[u]=ip++;
37. }
39. **void** addedge1(**int** u,**int** v)
40. {
41. edge1[ip1].u=u,edge1[ip1].v=v,edge1[ip1].lca=-1,edge1[ip1].next=head1[u],head1[u]=ip1++;
42. }
44. **int**  Find(**int** x)
45. {
46. **if**(father[x]==x)
47. **return** x;
48. **return** father[x]=Find(father[x]);
49. }
51. **void** Union(**int** x,**int** y)
52. {
53. x=Find(x);
54. y=Find(y);
55. **if**(x!=y)
56. father[y]=x;
57. }
59. **void** tarjan(**int** u)
60. {
61. vis[u]=1;
62. father[u]=u;
63. **for**(**int** i=head[u];i!=-1;i=edge[i].next)
64. {
65. **int** v=edge[i].v;
66. **int** w=edge[i].w;
67. **if**(!vis[v])
68. {
69. dir[v]=dir[u]+w;
70. tarjan(v);
71. Union(u,v);
72. }
73. }
74. **for**(**int** i=head1[u];i!=-1;i=edge1[i].next)
75. {
76. **int** v=edge1[i].v;
77. **if**(vis[v])
78. {
79. edge1[i].lca=edge1[i^1].lca=father[Find(v)];
80. }
81. }
82. }

**树链剖分**

1. /\*\*
2. hdu3966  树链剖分（区间更新和单点求值）
3. 题目大意：给定一棵树，对指定的连点之间的点的权进行更新，过程中询问指定点的当前值
4. 解题思路：树链剖分，区间更新维护的题目
5. \*/
6. #include <stdio.h>
7. #include <string.h>
8. #include <algorithm>
9. #include <iostream>
10. **using** **namespace** std;
11. **const** **int** maxn=50005;
12. **typedef** **long** **long** LL;
13. **int** n,m,q,z;
14. LL a[maxn],sum[maxn\*4],col[maxn\*4];
15. **int** fa[maxn],Hash[maxn],dep[maxn],son[maxn],siz[maxn];
16. **int** num[maxn],top[maxn];
18. **int** head[maxn],ip;
20. **void** init()
21. {
22. memset(head,-1,**sizeof**(head));
23. memset(col,0,**sizeof**(col));
24. memset(sum,0,**sizeof**(sum));
25. ip=0;
26. }
28. **struct** note
29. {
30. **int** v,next;
31. }edge[maxn\*4];
33. **void** addedge(**int** u,**int** v)
34. {
35. edge[ip].v=v,edge[ip].next=head[u],head[u]=ip++;
36. }
38. **void** dfs(**int** u,**int** pre)
39. {
40. son[u]=0,siz[u]=1,dep[u]=dep[pre]+1,fa[u]=pre;
41. **for**(**int** i=head[u];i!=-1;i=edge[i].next)
42. {
43. **int** v=edge[i].v;
44. **if**(v==pre)**continue**;
45. dfs(v,u);
46. **if**(siz[son[u]]<siz[v])
47. {
48. son[u]=v;
49. }
50. siz[u]+=siz[v];
51. }
52. ///printf("%d fa,dep,siz,son %d %d %d %d\n",u,fa[u],dep[u],siz[u],son[u]);
53. }
55. **void** set\_que(**int** u,**int** tp)
56. {
57. top[u]=tp,num[u]=++z,Hash[z]=u;
58. **if**(son[u])
59. {
60. set\_que(son[u],tp);
61. }
62. **for**(**int** i=head[u];i!=-1;i=edge[i].next)
63. {
64. **int** v=edge[i].v;
65. **if**(v==fa[u]||v==son[u])**continue**;
66. set\_que(v,v);
67. }
68. ///  printf("%d top num %d %d\n",u,top[u],num[u]);
69. }
71. **void** push\_up(**int** root)
72. {
73. sum[root]=sum[root<<1]+sum[root<<1|1];
74. }
75. **void** build(**int** root,**int** l,**int** r)
76. {
77. col[root]=0;
78. **if**(l==r)
79. {
80. sum[root]=a[Hash[l]];
81. **return**;
82. }
83. **int** mid=(l+r)>>1;
84. build(root<<1,l,mid);
85. build(root<<1|1,mid+1,r);
86. push\_up(root);
87. }
89. **void** push\_down(**int** root,**int** len)
90. {
91. **if**(col[root])
92. {
93. col[root<<1]+=col[root];
94. col[root<<1|1]+=col[root];
95. sum[root<<1]+=col[root]\*(len-(len>>1));
96. sum[root<<1|1]+=col[root]\*(len>>1);
97. col[root]=0;
98. }
99. }
100. **void** update(**int** root,**int** l,**int** r,**int** x,**int** y,**int** z)
101. {
102. **if**(l>y||r<x)**return**;
103. **if**(x<=l&&r<=y)
104. {
105. col[root]+=z;
106. sum[root]+=(r-l+1)\*z;
107. **return**;
108. }
109. push\_down(root,r-l+1);
110. **int** mid=(l+r)>>1;
111. update(root<<1,l,mid,x,y,z);
112. update(root<<1|1,mid+1,r,x,y,z);
113. push\_up(root);
114. }
115. LL query(**int** root,**int** l,**int** r,**int** loc)
116. {
117. **if**(l>loc||r<loc)**return** 0;
118. **if**(l==r)
119. {
120. **return** sum[root];
121. }
122. push\_down(root,r-l+1);
123. **int** mid=(l+r)>>1;
124. **if**(loc<=mid)
125. **return** query(root<<1,l,mid,loc);
126. **else**
127. **return** query(root<<1|1,mid+1,r,loc);
128. }
129. **int** main()
130. {
131. //freopen("data.txt","r",stdin);
132. **while**(~scanf("%d%d%d",&n,&m,&q))
133. {
134. init();
135. **for**(**int** i=1;i<=n;i++)
136. {
137. scanf("%I64d",&a[i]);
138. }
139. **for**(**int** i=0;i<m;i++)
140. {
141. **int** u,v;
142. scanf("%d%d",&u,&v);
143. addedge(u,v);
144. addedge(v,u);
145. }
146. **int** root=(n+1)>>1;
147. z=0,dep[0]=0,siz[0]=0;
148. dfs(root,0);
149. set\_que(root,root);
150. build(1,1,z);
151. **while**(q--)
152. {
153. **char** s[20];
154. **int** x,y,zz;
155. scanf("%s",s);
156. **if**(s[0]=='Q')
157. {
158. scanf("%d",&x);
159. printf("%I64d\n",query(1,1,z,num[x]));
160. }
161. **else**
162. {
163. scanf("%d%d%d",&x,&y,&zz);
164. **if**(s[0]=='D')zz=-zz;
165. **int** f1=top[x],f2=top[y];
166. **while**(f1!=f2)
167. {
168. **if**(dep[f1]<dep[f2])
169. {
170. swap(f1,f2);
171. swap(x,y);
172. }
173. update(1,1,z,num[f1],num[x],zz);
174. x=fa[f1],f1=top[x];
175. }
176. **if**(dep[x]>dep[y])
177. {
178. swap(x,y);
179. }
180. update(1,1,z,num[x],num[y],zz);
181. }
182. }
183. }
184. **return** 0;
185. }

**高精度加法和乘法类**

1. /\*\*
2. 如何用：
3. 1、变量声明：可以给初值，如：BigInt ans=100;
4. 可以补给初值（默认为0），如BigInt ans；
5. 2、计算：可以连个BigInt对象相乘，相加；ans+ans\*ans;
6. 也可以和整数相乘相加，如：ans+78\*ans;
7. \*/
8. **struct** BigInt
9. {
10. **const** **static** **int** mod=10000;
11. **const** **static** **int** DLEN=4;
12. **int** a[600],len;
13. BigInt()
14. {
15. memset(a,0,**sizeof**(a));
16. len=1;
17. }
18. BigInt(**int** v)
19. {
20. memset(a,0,**sizeof**(a));
21. len=0;
22. **do**
23. {
24. a[len++]=v%mod;
25. v/=mod;
26. }
27. **while**(v);
28. }
29. BigInt(**const** **char** \*s)
30. {
31. memset(a,0,**sizeof**(a));
32. **int** L=strlen(s);
33. len=L/DLEN;
34. **if**(L%DLEN)len++;
35. **int** index=0;
36. **for**(**int** i=L-1; i>=0; i-=DLEN)
37. {
38. **int** t=0;
39. **int** k=i-DLEN+1;
40. **if**(k<0)k=0;
41. **for**(**int** j=k; j<=i; j++)
42. {
43. t=t\*10+s[j]-'0';
44. }
45. a[index++]=t;
46. }
47. }
48. BigInt operator +(**const** BigInt &b)**const**
49. {
50. BigInt res;
51. res.len=max(len,b.len);
52. **for**(**int** i=0; i<=res.len; i++)
53. {
54. res.a[i]=0;
55. }
56. **for**(**int** i=0; i<res.len; i++)
57. {
58. res.a[i]+=((i<len)?a[i]:0)+((i<b.len)?b.a[i]:0);
59. res.a[i+1]+=res.a[i]/mod;
60. res.a[i]%=mod;
61. }
62. **if**(res.a[res.len]>0)res.len++;
63. **return** res;
64. }
65. BigInt operator \*(**const** BigInt &b)**const**
66. {
67. BigInt res;
68. **for**(**int** i=0; i<len; i++)
69. {
70. **int** up=0;
71. **for**(**int** j=0; j<b.len; j++)
72. {
73. **int** temp=a[i]\*b.a[j]+res.a[i+j]+up;
74. res.a[i+j]=temp%mod;
75. up=temp/mod;
76. }
77. **if**(up!=0)
78. res.a[i+b.len]=up;
79. }
80. res.len=len+b.len;
81. **while**(res.a[res.len-1]==0&&res.len>1)res.len--;
82. **return** res;
83. }
84. **void** output()
85. {
86. printf("%d",a[len-1]);
87. **for**(**int** i=len-2; i>=0; i--)
88. printf("%04d",a[i]);
89. printf("\n");
90. }
91. };

**数论**

**素数筛选**

1. #include <string.h>
2. #include <stdio.h>
3. #include <iostream>
4. **using** **namespace** std;
5. **const** **int** N=165005;
6. **bool** prime[N];
7. **long** **long** p[N];
8. **int** main()
9. {
10. **long** **long** t,n,i,j,k=0;
11. memset(prime,**true**,**sizeof**(prime));
12. **for**(i=2; i<N; i++)
13. {
14. **if**(prime[i])
15. {
16. p[k++]=i;
17. **for**(j=i\*i; j<=N; j+=i)
18. prime[j]=**false**;
19. }
20. }
21. **while**(~scanf("%lld",&t))
22. {
23. **while**(t--)
24. {
25. scanf("%lld",&n);
26. printf("%lld\n",p[n-1]);
27. }
28. }
29. **return** 0;
30. }

**扩展欧几里得**

A\*X+B\*Y=GCD(A,B);函数的返回值为满足条件的X和Y.

A line on the plane is described by an equation Ax + By + C =0. You are to find any point on this line, whose coordinates are integer numbers from  - 5·1018 to 5·1018 inclusive, or to find out that such points do not exist.

**Input**

The first line contains three integers A, B and C ( - 2·109 ≤ A, B, C ≤ 2·109) — corresponding coefficients of the line equation. It is guaranteed that A2 + B2 > 0.

**Output**

If the required point exists, output its coordinates, otherwise output -1.

**Sample test(s)**

**input**

2 5 3

**output**

6 -3

思路：先将A,B,C除以gcd(a,b)，此时保证A,B已经互素，调用模板求出的特解x，y 只需乘上c即为答案。

1. #include <iostream>
2. #include <string.h>
3. #include <algorithm>
4. #include <stdio.h>
6. **using** **namespace** std;
7. **//-----------------------------------------------------------**
8. **//模板部分**
9. **typedef** **long** **long** LL;
10. LL gcd(LL a,LL b)
11. {
12. **return** b ? gcd(b,a%b):a;
13. }
14. **void** extend\_Euclid(LL a,LL b,LL &x,LL & y)
15. {
16. **if**(b == 0)
17. {
18. x = 1;
19. y = 0;
20. **return**;
21. }
22. extend\_Euclid(b,a%b,x,y);
23. LL tmp = x;
24. x = y;
25. y = tmp - (a / b) \* y;
26. }
27. //---------------------------------------------------------------
28. **int** main()
29. {
30. LL a,b,c,x,y;;
31. **while**(cin>>a>>b>>c)
32. {
33. c=-c;
34. LL g=gcd(a,b);
35. **if**(c%g)
36. {
37. printf("-1\n");
38. **continue**;
39. }
40. a/=g;
41. b/=g;
42. c/=g;
43. extend\_Euclid(a,b,x,y);
44. printf("%lld %lld\n",x\*c,y\*c);
45. }
46. **return** 0;
47. }

**素因子分解（n，n！）**

题目：输入两个整数，m和n求出最大的整数k使得m^k是n！的约数。

**[cpp]** [view plain](http://blog.csdn.net/lvshubao1314/article/details/25691701" \o "view plain)[copy](http://blog.csdn.net/lvshubao1314/article/details/25691701" \o "copy)



1. #include <stdio.h>
2. #include <string.h>
3. #include <math.h>
4. #include <iostream>
5. **using** **namespace** std;
6. **const** **int** N=10005;
7. **bool** prime[N];
8. **int** p[N];
9. **int** k,cnt;
10. **int** pri[N],num[N];
11. **void** isprime()//筛选出所需范围内所有的素数
12. {
13. k=0;
14. memset(prime,**true**,**sizeof**(prime));
15. **for**(**int** i=2;i<N;i++)
16. {
17. **if**(prime[i])
18. {
19. p[k++]=i;
20. **for**(**int** j=i+i;j<N;j+=i)
21. prime[j]=**false**;
22. }
23. }
24. }
25. **void** divide(**int** n)// 将整数n进行素因子分解素因子保存在pri[]素组中，而该素因子对应的幂指数保存在num[]素组中。
26. {
27. **int** t=(**int**)sqrt(1.0\*n);//若n不是素数那么最大的素因子只可能小于或等于根号n，优化省时
28. cnt=0;
29. **for**(**int** i=0;p[i]<=t;i++)
30. {
31. **if**(n%p[i]==0)
32. {
33. pri[cnt]=p[i];
34. **int** a=0;
35. **while**(n%p[i]==0)
36. {
37. a++;
38. n/=p[i];
39. }
40. num[cnt]=a;
41. cnt++;
42. }
43. }
44. **if**(n>1)//如果此时n还没被变为1，就说明n本身是一个素数
45. {
46. pri[cnt]=n;
47. num[cnt]=1;
48. cnt++;
49. }
50. }
51. **int** cal(**int** n,**int** p)//阶乘的素因子分解
52. {
53. **int** ans=0;
54. **while**(n)
55. {
56. ans+=n/p;
57. n/=p;
58. }
59. **return** ans;
60. }
61. **void** work(**int** m,**int** n)
62. {
63. divide(m);
64. **int** ans=99999999;
65. **for**(**int** i=0;i<cnt;i++)//若m的素因子在n！中都有则一定有n！可以整除m，反之就不行
66. {
67. **int** b=cal(n,pri[i]);
68. **int** tmp=b/num[i];
69. **if**(ans>tmp)//ans的最大取值总是等于n！和m对应素因子商的最小值
70. ans=tmp;
71. }
72. **if**(ans==0)
73. puts("Impossible to divide");
74. **else**
75. printf("%d\n",ans);
76. }
77. **int** main()
78. {
79. isprime();
80. **int** T,n,m,tt=1;
81. scanf("%d",&T);
82. **while**(T--)
83. {
84. scanf("%d%d",&m,&n);
85. printf("Case %d:\n",tt++);
86. work(m,n);
87. }
88. **return** 0;
89. }

**博弈**

斐波那契博弈

有一堆个数为n的石子，游戏双方轮流取石子，满足：

1)先手不能在第一次把所有的石子取完；

2)之后每次可以取的石子数介于1到对手刚取的石子数的2倍之间（包含1和对手刚取的石子数的2倍）。

1. #include<iostream>
2. #include<cstdio>
3. #include<cstring>
4. #include<algorithm>
5. #include<cmath>
6. #include<vector>
7. #include<string>
8. #include<map>
9. #define LL long long
10. #define N 1000000
11. #define inf 1<<20
12. **using** **namespace** std;
13. **int** fib[50];
14. **int** main(){
15. fib[0]=1;fib[1]=2;
16. **for**(**int** i=2;i<45;i++)
17. fib[i]=fib[i-1]+fib[i-2];
18. **int** n;
19. **while**(scanf("%d",&n)!=EOF&&n){
20. **int** i=0;
21. **for**(i=0;i<45;i++)
22. **if**(fib[i]==n)
23. **break**;
24. **if**(i<45)
25. puts("Second win");
26. **else**
27. puts("First win");
28. }
29. **return** 0;
30. }
31. #include<iostream>
32. #include<cstdio>
33. #include<cstring>
34. #include<algorithm>
35. #include<cmath>
36. #include<vector>
37. #include<string>
38. #include<map>
39. #define LL long long
40. #define N 1000000
41. #define inf 1<<20
42. **using** **namespace** std;
43. **int** fib[50];
44. **int** main(){
45. fib[0]=1;fib[1]=2;
46. **for**(**int** i=2;i<45;i++)
47. fib[i]=fib[i-1]+fib[i-2];
48. **int** n;
49. **while**(scanf("%d",&n)!=EOF&&n){
50. **int** i=0;
51. **for**(i=0;i<45;i++)
52. **if**(fib[i]==n)
53. **break**;
54. **if**(i<45)
55. puts("Second win");
56. **else**
57. puts("First win");
58. }
59. **return** 0;
60. }

**威佐夫博奕（Wythoff Game）**

有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. 威佐夫博弈
3. a(k)=k\*(1+sqrt(5))/2.0
4. b(k)=a(k)+k；
5. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
6. #include <iostream>
7. #include <cmath>
8. **using** **namespace** std;
10. **int** main()
11. {
12. **int** a,b;
13. **while**(cin>>a>>b)
14. {
15. **if**(a>b)
16. swap(a,b);
17. **int** k=b-a;
18. **if**(a==(**int**)(k\*(1+sqrt(5))/2.0))
19. cout<<0<<endl;
20. **else**
21. cout<<1<<endl;
22. }
23. **return** 0;
24. }

**字符串**

**[后缀数组模板](http://blog.csdn.net/lvshubao1314/article/details/38374359)**

1. #include <iostream>
2. #include <cstring>
3. #include <algorithm>
4. #include <cstdio>
5. **using** **namespace** std;
6. **const** **int** maxn=211111;
7. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
8. \*\*  后缀数组 Suffix Array
9. \*\*  INIT：solver.call\_fun(char\* s);
10. \*\*  CALL: solver.lcp(int i,int j); //后缀i与后缀j的最长公共前缀
11. \*\*  SP\_USE: solver.LCS(char \*s1,char\* s2); //最长公共字串
12. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
13. **struct** SuffixArray
14. {
15. **int** r[maxn];
16. **int** sa[maxn],rank[maxn],height[maxn];
17. **int** t[maxn],t2[maxn],c[maxn],n;
18. **int** m;//模板长度
19. **void** init(**char**\* s)
20. {
21. n=strlen(s);
22. **for** (**int** i=0; i<n; i++) r[i]=**int**(s[i]);
23. m=300;
24. }
25. **int** cmp(**int** \*r,**int** a,**int** b,**int** l)
26. {
27. **return** r[a]==r[b]&&r[a+l]==r[b+l];
28. }
29. /\*\*
30. 字符要先转化为正整数
31. 待排序的字符串放在r[]数组中，从r[0]到r[n-1]，长度为n，且最大值小于m。
32. 所有的r[i]都大于0,r[n]无意义算法中置0
33. 函数结束后，结果放在sa[]数组中(名次从1..n)，从sa[1]到sa[n]。s[0]无意义
34. \*\*/
35. **void** build\_sa()
36. {
37. **int** i,k,p,\*x=t,\*y=t2;
38. r[n++]=0;
39. **for** (i=0; i<m; i++) c[i]=0;
40. **for** (i=0; i<n; i++) c[x[i]=r[i]]++;
41. **for** (i=1; i<m; i++) c[i]+=c[i-1];
42. **for** (i=n-1; i>=0; i--) sa[--c[x[i]]]=i;
43. **for** (k=1,p=1; k<n; k\*=2,m=p)
44. {
45. **for** (p=0,i=n-k; i<n; i++) y[p++]=i;
46. **for** (i=0; i<n; i++) **if** (sa[i]>=k) y[p++]=sa[i]-k;
47. **for** (i=0; i<m; i++) c[i]=0;
48. **for** (i=0; i<n; i++) c[x[y[i]]]++;
49. **for** (i=1; i<m; i++) c[i]+=c[i-1];
50. **for** (i=n-1; i>=0; i--) sa[--c[x[y[i]]]]=y[i];
51. swap(x,y);
52. p=1;
53. x[sa[0]]=0;
54. **for** (i=1; i<n; i++) x[sa[i]]=cmp(y,sa[i-1],sa[i],k)?p-1:p++;
55. }
56. n--;
57. }
58. /\*\*
59. height[2..n]:height[i]保存的是lcp(sa[i],sa[i-1])
60. rank[0..n-1]:rank[i]保存的是原串中suffix[i]的名次
61. \*\*/
62. **void** getHeight()
63. {
64. **int** i,j,k=0;
65. **for** (i=1; i<=n; i++) rank[sa[i]]=i;
66. **for** (i=0; i<n; i++)
67. {
68. **if** (k) k--;
69. j=sa[rank[i]-1];
70. **while** (r[i+k]==r[j+k]) k++;
71. height[rank[i]]=k;
72. }
73. }
74. **int** d[maxn][20];
75. //元素从1编号到n
76. **void** RMQ\_init(**int** A[],**int** n)
77. {
78. **for** (**int** i=1; i<=n; i++) d[i][0]=A[i];
79. **for** (**int** j=1; (1<<j)<=n; j++)
80. **for** (**int** i=1; i+j-1<=n; i++)
81. d[i][j]=min(d[i][j-1],d[i+(1<<(j-1))][j-1]);
82. }
83. **int** RMQ(**int** L,**int** R)
84. {
85. **int** k=0;
86. L=rank[L];
87. R=rank[R];
88. **if**(L>R) swap(L,R);
89. L++;
90. **while** ((1<<(k+1))<=R-L+1) k++;
91. **return** min(d[L][k],d[R-(1<<k)+1][k]);
92. }
93. **void** LCP\_init()
94. {
95. RMQ\_init(height,n);
96. }
97. **int** lcp(**int** i,**int** j)
98. {
99. **if** (rank[i]>rank[j]) swap(i,j);
100. **return** RMQ(rank[i]+1,rank[j]);
101. }
102. **void** call\_fun(**char**\* s)
103. {
104. init(s);//初始化后缀数组
105. build\_sa();//构造后缀数组sa
106. getHeight();//计算height与rank
107. LCP\_init();//初始化RMQ
108. }
109. } solver;

**哈希表的构建与字符串哈希**

1. #include <stdio.h>
2. #include <iostream>
3. #include <string.h>
4. **using** **namespace** std;
5. **const** **int** maxn=2500\*2500+3;
6. **const** **int** maxd=15;
7. **const** **int** seed=2500\*2500+3;//是一个大素数
8. **typedef** **long** **long** LL;
9. **struct** HASH//哈希表
10. {
11. **int** head[seed];
12. **int** size;
13. **struct**  NODE
14. {
15. **int** key;
16. **int** value;
17. **int** next;
18. }edges[maxn];
19. **void** init()
20. {
21. memset(head,-1,**sizeof**(head));
22. size=0;
23. }
24. **void** insert(**int** key,**int** value)//插入值
25. {
26. **int** h=key%seed;
27. **for**(**int** i=head[h];i!=-1;i=edges[i].next)
28. {
29. **if**(edges[i].key==key)
30. {
31. edges[i].value=max(value,edges[i].value);
32. **return**;
33. }
34. }
35. edges[size].key=key;
36. edges[size].value=value;
37. edges[size].next=head[h];
38. head[h]=size++;
39. }
40. **int** search(**int** key)//寻找
41. {
42. **int** h=key%seed;
43. **for**(**int** i=head[h];i!=-1;i=edges[i].next)
44. {
45. **if**(edges[i].key==key)
46. {
47. **return** edges[i].value;
48. }
49. }
50. **return** -1;
51. }
52. }tr;
53. /\*\*另一种
54. \*/
55. **const** **int** MAXN=1<<20;
56. **const** **int** HASH = 1000007;
58. **struct** hashmap//建立哈希表
59. {
60. LL a[MAXN];
61. **int** head[HASH],next[MAXN],size;
62. **void** init() //初始化
63. {
64. memset(head,-1,**sizeof**(head));
65. size=0;
66. }
67. **bool** find(LL val) //查找一个元素是否在哈希表内
68. {
69. **int** tmp = (val%HASH + HASH)%HASH;
70. **for**(**int** i = head[tmp]; i!=-1; i=next[i])
71. **if**(val==a[i]) **return** **true**;
72. **return** **false**;
73. }
74. **void** add(LL val) //添加元素到哈希表中
75. {
76. **int** tmp =(val%HASH+HASH)%HASH;
77. **if**(find(val)) **return**;
78. a[size]=val;
79. next[size]=head[tmp];
80. head[tmp]=size++;
81. }
82. } h1;

**[ＳＴＬ中的find()函数，kmp的替代品](http://blog.csdn.net/lvshubao1314/article/details/29563951)**

1. #include <string>
2. #include <iostream>
3. **using** **namespace** std;
4. **int** main()
5. {
6. ////find函数返回类型 size\_type
7. string s("1a2b3c4d5e6f7g8h9i1a2b3c4d5e6f7g8ha9i");
8. string flag;
9. string::size\_type position;
11. //find 函数 返回jk 在s 中的下标位置
12. position = s.find("jk");
13. **if** (position != s.npos)  //如果没找到，返回一个特别的标志c++中用npos表示，我这里npos取值是4294967295，
14. {
15. cout << "position is : " << position << endl;
16. }
17. **else**
18. {
19. cout << "Not found the flag"<<endl;
20. }
21. //find 函数 返回flag 中任意字符 在s 中第一次出现的下标位置
22. flag = "c";
23. position = s.find\_first\_of(flag);
24. cout << "s.find\_first\_of(flag) is : " << position << endl;
25. //从字符串s 下标5开始，查找字符串b ,返回b 在s 中的下标
26. position=s.find("b",5);
27. cout<<"s.find(b,5) is : "<<position<<endl;

30. //查找s 中flag 出现的所有位置。
31. flag="a";
32. position=0;
33. **int** i=1;
34. **while**((position=s.find\_first\_of(flag,position))!=string::npos)
35. {
36. //position=s.find\_first\_of(flag,position);
37. cout<<"position  "<<i<<" : "<<position<<endl;
38. position++;
39. i++;
40. }
41. //查找flag 中与s 第一个不匹配的位置
42. flag="acb12389efgxyz789";
43. position=flag.find\_first\_not\_of (s);
44. cout<<"flag.find\_first\_not\_of (s) :"<<position<<endl;
45. //反向查找，flag 在s 中最后出现的位置
46. flag="3";
47. position=s.rfind (flag);
48. cout<<"s.rfind (flag) :"<<position<<endl;
49. **return** 0;
50. }

**KMP算法**

1. ///未经优化的kmp数组
2. **void** GetNext(**char**\* p,**int** next[])
3. {
4. **int** pLen = strlen(p);
5. next[0] = -1;
6. **int** k = -1;
7. **int** j = 0;
8. **while** (j < pLen)
9. {
10. //p[k]表示前缀，p[j]表示后缀
11. **if** (k == -1 || p[j] == p[k])
12. {
13. ++k;
14. ++j;
15. next[j] = k;
16. }
17. **else**
18. {
19. k = next[k];
20. }
21. }
22. }
23. ///优化过后的next 数组求法
24. **void** GetNextval(**char**\* p, **int** next[])
25. {
26. **int** pLen = strlen(p);
27. next[0] = -1;
28. **int** k = -1;
29. **int** j = 0;
30. **while** (j < pLen)
31. {
32. ///p[k]表示前缀，p[j]表示后缀
33. **if** (k == -1 || p[j] == p[k])
34. {
35. ++j;
36. ++k;
37. ///较之前next数组求法，改动在下面4行
38. **if** (p[j] != p[k])
39. next[j] = k;   ///之前只有这一行
40. **else**
41. ///因为不能出现p[j] = p[ next[j ]]，所以当出现时需要继续递归，k = next[k] = next[next[k]]
42. next[j] = next[k];
43. }
44. **else**
45. {
46. k = next[k];
47. }
48. }
49. }
50. **int** KmpSearch(**char**\* s, **char**\* p)
51. {
52. **int** i = 0;
53. **int** j = 0;
54. **int** sLen = strlen(s);
55. **int** pLen = strlen(p);
56. **while** (i < sLen && j < pLen)
57. {
58. ///①如果j = -1，或者当前字符匹配成功（即S[i] == P[j]），都令i++，j++
59. **if** (j == -1 || s[i] == p[j])
60. {
61. i++;
62. j++;
63. }
64. **else**
65. {
66. ///②如果j != -1，且当前字符匹配失败（即S[i] != P[j]），则令 i 不变，j = next[j]
67. ///next[j]即为j所对应的next值
68. j = next[j];
69. }
70. }
71. **if** (j == pLen)
72. **return** i - j;
73. **else**
74. **return** -1;
75. }

**AC自动机和矩阵**

1. #include <stdio.h>
2. #include <string.h>
3. #include <algorithm>
4. #include <stdio.h>
5. #include <map>
6. #include <queue>
7. **using** **namespace** std;
8. map<**char**,**int**>mp;
9. **int** N,M,P;
10. **struct** Matrix
11. {
12. **int** mat[110][110];
13. **int** n;
14. Matrix() {}
15. Matrix(**int** \_n)
16. {
17. n=\_n;
18. **for**(**int** i=0; i<n; i++)
19. {
20. **for**(**int** j=0; j<n; j++)
21. {
22. mat[i][j]=0;
23. }
24. }
25. }
26. };
28. **struct** Trie
29. {
30. **int** next[110][256],fail[110];
31. **bool** end[110];
32. **int** L,root;
33. **int** newnode()
34. {
35. **for**(**int** i=0; i<256; i++)
36. {
37. next[L][i]=-1;
38. }
39. end[L++]=**false**;
40. **return** L-1;
41. }
42. **void** init()
43. {
44. L=0;
45. root=newnode();
46. }
47. **void** insert(**char** \*buf)
48. {
49. **int** len=strlen(buf);
50. **int** now=root;
51. **for**(**int** i=0; i<len; i++)
52. {
53. **if**(next[now][mp[buf[i]]]==-1)
54. next[now][mp[buf[i]]]=newnode();
55. now=next[now][mp[buf[i]]];
56. }
57. end[now]=**true**;
58. }
59. **void** build()
60. {
61. queue<**int**>Q;
62. fail[root]=root;
63. **for**(**int** i=0; i<256; i++)
64. {
65. **if**(next[root][i]==-1)
66. next[root][i]=root;
67. **else**
68. {
69. fail[next[root][i]]=root;
70. Q.push(next[root][i]);
71. }
72. }
73. **while**(!Q.empty())
74. {
75. **int** now=Q.front();
76. Q.pop();
77. **if**(end[fail[now]]==**true**)
78. end[now]=**true**;
79. **for**(**int** i=0; i<256; i++)
80. {
81. **if**(next[now][i]==-1)
82. next[now][i]=next[fail[now]][i];
83. **else**
84. {
85. fail[next[now][i]]=next[fail[now]][i];
86. Q.push(next[now][i]);
87. }
88. }
89. }
90. }
91. Matrix getMatrix()
92. {
93. Matrix res=Matrix(L);
94. **for**(**int** i=0; i<L; i++)
95. {
96. **for**(**int** j=0; j<N; j++)
97. {
98. **if**(end[next[i][j]]==**false**)
99. res.mat[i][next[i][j]]++;
100. }
101. }
102. **return** res;
103. }
104. } ac;

**数位dp的模式（例1）**

1. /\*\*
2. spoj10606 数位dp （求出现的数字，所有偶数出现奇数次，所有奇数出现偶数次）
3. 解题思路：3进制表示数字0~9的出现情况，0表示没有出现，1表示奇数次，2表示偶数次
4. \*/
5. #include <string.h>
6. #include <stdio.h>
7. #include <iostream>
8. #include <algorithm>
9. **using** **namespace** std;
10. **typedef** **long** **long** LL;
11. LL dp[20][60000];
12. **int** bit[20];
13. **bool** check(**int** s)
14. {
15. **int** num[10];
16. **for**(**int** i=0;i<10;i++)
17. {
18. num[i]=s%3;
19. s/=3;
20. }
21. **for**(**int** i=0;i<10;i++)
22. {
23. **if**(num[i]!=0)
24. {
25. **if**(i%2==0&&num[i]==2)**return** **false**;
26. **if**(i%2==1&&num[i]==1)**return** **false**;
27. }
28. }
29. **return** **true**;
30. }
31. **int** getnews(**int** x,**int** s)
32. {
33. **int** num[10];
34. **for**(**int** i=0;i<10;i++)
35. {
36. num[i]=s%3;
37. s/=3;
38. }
39. **if**(num[x]==0)
40. num[x]=1;
41. **else**
42. num[x]=3-num[x];
43. **int** news=0;
44. **for**(**int** i=9;i>=0;i--)
45. {
46. news\*=3;
47. news+=num[i];
48. }
49. **return** news;
50. }
52. LL dfs(**int** pos,**int** s,**int** flag,**int** z)
53. {
54. **if**(pos==-1)**return** check(s);
55. **if**(!flag&&dp[pos][s]!=-1)
56. **return** dp[pos][s];
57. LL ans=0;
58. **int** end=flag?bit[pos]:9;
59. **for**(**int** i=0;i<=end;i++)
60. {
61. ans+=dfs(pos-1,(z&&i==0)?0:getnews(i,s),flag&&i==end,z&&i==0);
62. }
63. **if**(!flag)dp[pos][s]=ans;
64. **return** ans;
65. }
66. LL solve(LL n)
67. {
68. **int** len=0;
69. **while**(n)
70. {
71. bit[len++]=n%10;
72. n/=10;
73. }
74. **return** dfs(len-1,0,1,1);
75. }
77. **int** main()
78. {
79. **int** T;
80. memset(dp,-1,**sizeof**(dp));
81. LL a,b;
82. scanf("%d",&T);
83. **while**(T--)
84. {
85. scanf("%lld%lld",&a,&b);
86. printf("%lld\n",solve(b)-solve(a-1));
87. }
88. **return** 0;
89. }

**数位dp的模式（例2）**

1. /\*\*
2. hdu 3709   数位dp（自身平衡的数字）
3. 题目大意：求给定区间内满足自身平衡的数的个数，所谓平衡，
4. 比如：4139 is a balanced number with pivot fixed at 3. The torqueses are 4\*2 + 1\*1 = 9 and 9\*1 = 9, for left part and right part, respectively.
5. 解题思路：枚举支点。dp[i][j][k] i表示处理到的数位，j是支点，k是力矩和。但是要要把全是0的数排除
6. \*/
7. #include <stdio.h>
8. #include <string.h>
9. #include <algorithm>
10. #include <iostream>
11. **using** **namespace** std;
12. **typedef** **long** **long** LL;
14. LL dp[20][20][2000];
15. **int** bit[25];
17. LL dfs(**int** pos,**int** level,**int** presum,**int** flag)
18. {
19. **if**(pos==-1)**return** presum==0;
20. **if**(presum<0)**return** 0;
21. **if**(!flag&&dp[pos][level][presum]!=-1)
22. **return** dp[pos][level][presum];
23. **int** end=flag?bit[pos]:9;
24. LL ans=0;
25. **for**(**int** i=0; i<=end; i++)
26. {
27. ans+=dfs(pos-1,level,presum+(pos-level)\*i,flag&&(i==end));
28. }
29. **if**(!flag)dp[pos][level][presum]=ans;
30. **return** ans;
31. }
33. LL solve(LL n)
34. {
35. **int** len=0;
36. **while**(n)
37. {
38. bit[len++]=n%10;
39. n/=10;
40. }
41. LL ans=0;
42. **for**(**int** i=0; i<len; i++)
43. {
44. ans+=dfs(len-1,i,0,1);
45. }
46. **return** ans-(len-1);///去掉全部为0的情况
47. }
48. **int** main()
49. {
50. **int** T;
51. LL l,r;
52. memset(dp,-1,**sizeof**(dp));
53. scanf("%d",&T);
54. **while**(T--)
55. {
56. scanf("%I64d%I64d",&l,&r);
57. printf("%I64d\n",solve(r)-solve(l-1));
58. }
59. **return** 0;
60. }

**LIS**

[LIS 的 n\*logn 写法并标记输出其中任意一个](http://blog.csdn.net/lvshubao1314/article/details/38561889)

1. #include <cstdio>
2. #include <cstring>
3. #include <algorithm>
4. **using** **namespace** std;
5. **const** **int** maxn = 3010;
6. **int** q[maxn],path[maxn],c[maxn],st[maxn],n;
7. //q:最长上升子序列的下标，path：该位置在上升子序列中前一个位置的下表 c：原序列,st:最后保存一个最长上升子序列
9. **int** bin\_search(**int** x,**int** l,**int** r)//返回已知的最长上升子序列中最后一个小于x的位置+1
10. {
11. **while**(l<=r)
12. {
13. **int** mid=(l+r)>>1;
14. **if**(c[q[mid]]<x)
15. l=mid+1;
16. **else**
17. r=mid-1;
18. }
19. **return** l;
20. }
22. **int** main()
23. {
24. //freopen("lis.in","r",stdin);
25. //freopen("lis.out","w",stdout);
26. **while**(~scanf("%d",&n))
27. {
28. **for**(**int** i=0; i<n; i++)
29. scanf("%d",&c[i]);
30. **int** len=0;
31. q[len++]=0;
32. path[0]=0;
33. **for**(**int** i=0; i<n; i++)
34. {
35. **if**(c[i]>c[q[len-1]])//已求的序列中没有比这个数大的，最长上升子序列加1
36. {
37. path[i]=q[len-1];
38. q[len++]=i;
39. }
40. **else**
41. {
42. **int** idex = bin\_search(c[i],0,len-1);//寻找已知子序列中最后一个比它小的位置，该数放在其后
43. path[i] = idex ? q[idex-1] : 0;
44. q[idex] = i;
45. }
46. }
47. **int** k=0;
48. printf("%d\n",len);
49. st[k++]=c[q[len-1]];
50. **int** tmp=q[len-1];
51. **while**(--len)
52. {
53. tmp = path[tmp];
54. st[k++]=c[tmp];
55. }
56. **for**(**int** i=k-1;i>=0;i--)
57. printf(i==0?"%d\n":"%d ",st[i]);
58. }
59. **return** 0;
60. }
61. /\*\*
63. 7
64. 2 3 1 4 0 3 6
66. \*/

**计算几何**

**(一)求多边形的面积（用叉积计算）**

1. //叉积,可以用来判断方向和求面积
2. **double** cross(Point a,Point b,Point c){
3. **return** (c.x-a.x)\*(b.y-a.y) - (b.x-a.x)\*(c.y-a.y);
4. }

7. //求多边形的面积
8. **double** S(Point p[],**int** n){
9. **double** ans = 0;
10. p[n] = p[0];
11. **for**(**int** i=1;i<n;i++)
12. ans += cross(p[0],p[i],p[i+1]);
13. **if**(ans < 0) ans = -ans;
14. **return** ans / 2.0;
15. }

**(二)求多边形的重心**

1. //求多边形的重心
2. Point grabity(Point p[],**int** n){
3. Point G;
4. **double** sum\_area=0;
5. **for**(**int** i=2;i<n;i++){
6. **double** area = cross(p[0],p[i-1],p[i]);
7. sum\_area+=area;
8. G.x+=(p[0].x+p[i-1].x+p[i].x)\*area;
9. G.y+=(p[0].y+p[i-1].y+p[i].y)\*area;
10. }
11. G.x=G.x/3/sum\_area,G.y=G.y/3/sum\_area;
12. **return** G;
13. }

**(三)andrew算法求凸包**

1. /\*\*
2. 求二维凸包Andrew算法，将所有的点按x小到大(x相等，y小到大)排序
3. 删去重复的点，得到一个序列p1,p2...,然后把p1,p2放入凸包中，从p3
4. 开始当新点再前进方向左边时(可以用叉积判断方向)继续，否则，依次
5. 删除最近加入凸包的点，直到新点再左边。
6. \*\*/
8. **int** ConvexHull(Point \*p,**int** n,Point \*stack){
9. sort(p,p+n);
10. n=unique(p,p+n)-p;
11. **int** m=0;
12. **for**(**int** i=0;i<n;i++){//如果不希望凸包的边上有输入的点则把两个等号去掉
13. **while**(m>1&&cross(stack[m-2],p[i],stack[m-1])<=0) m--;
14. stack[m++]=p[i];
15. }
16. **int** k=m;
17. **for**(**int** i=n-2;i>=0;i--){
18. **while**(m>k&&cross(stack[m-2],p[i],stack[m-1])<=0)m--;
19. stack[m++]=p[i];
20. }
21. **if**(n>1) m--;
22. **return** m;
23. }

**(四)比较函数提高精度：**

1. //判断符号，提高精度
2. **int** dcmp(**double** x){
3. **if**(fabs(x)<eps) **return** 0;
4. **else** **return** x < 0 ? -1 : 1;
5. }

**(五)向量/以及常见运算重载**

1. **struct** Point{
2. **double** x,y;
3. Point():x(0),y(0){}
4. Point(**double** \_x,**double** \_y):x(\_x),y(\_y){}
5. **bool** operator <(**const** **struct** Point &tmp) **const**{
6. **if**(x==tmp.x) **return** y<tmp.y;
7. **return** x<tmp.x;
8. }
9. };
11. **typedef** Point Vector;
12. Vector operator + (Vector A, Vector B){
13. **return** Vector(A.x+B.x, A.y+B.y);
14. }
15. Vector operator - (Point A, Point B){
16. **return** Vector(A.x-B.x, A.y-B.y);
17. }
18. Vector operator \* (Vector A, **double** p){
19. **return** Vector(A.x\*p, A.y\*p);
20. }
21. Vector operator / (Vector A, **double** p){
22. **return** Vector(A.x/p, A.y/p);
23. }
24. **bool** operator == (Vector A,Vector B){
25. **return** dcmp(A.x-B.x)==0&&dcmp(A.y-B.y)==0;
26. }
27. **double** Dot(Vector A, Vector B){//向量相乘
28. **return** A.x\*B.x + A.y\*B.y;  //a\*b\*cos(a,b)
29. }
30. **double** Length(Vector A){
31. **return** sqrt(Dot(A, A));    //向量的长度
32. }
33. **double** Angle(Vector A, Vector B){
34. **return** acos(Dot(A, B) / Length(A) / Length(B));    //向量的角度
35. }
36. **double** Cross(Vector A, Vector B){//叉积
37. **return** A.x\*B.y - A.y\*B.x;
38. }
39. /\*\*
40. 向量(x,y) 绕起点逆时针旋转a度。
41. x' = x\*cosa - y\*sina
42. y' = x\*sina + y\*cosa
43. \*\*/
44. Vector Rotate(Vector A,**double** a){
45. **return** Vector (A.x\*cos(a)-A.y\*cos(a),A.x\*sin(a)+A.y\*cos(a));
46. }
48. **double** trans(**double** ang){
49. **return** ang/180\*acos(-1.0);
50. }

**(六)旋转卡壳求凸包的直径，平面最远的点对**

1. //旋转卡壳求凸包的直径，平面距离最远的点对的距离
2. **double** rotatint\_calipers(Point \*p,**int** n){
3. **int** k=1;
4. **int** ans = 0;
5. p[n]=p[0];
6. **for**(**int** i=0;i<n;i++){
7. **while**(fabs(Cross(p[i+1],p[k],p[i]))<fabs(Cross(p[i+1],p[k+1],p[i])))
8. k=(k+1)%n;
9. ans = max(ans,max(dis(p[i],p[k]),dis(p[i+1],p[k])));
10. }
11. **return** ans;
12. }

**(七)旋转卡壳求凸包的宽度，即找一组距离最近的平行线似的凸包的点在两根线的内侧**

1. **double** rotating\_calipers(Point \*p,**int** n){
2. **int** k = 1;
3. **double** ans = 0x7FFFFFFF;
4. p[n] = p[0];
5. **for**(**int** i=0;i<n;i++){
6. **while**(fabs(cross(p[i],p[i+1],p[k])) < fabs(cross(p[i],p[i+1],p[k+1])))
7. k = (k+1) % n;
8. **double** tmp = fabs(cross(p[i],p[i+1],p[k]));
9. **double** d   = dist(p[i],p[i+1]);
10. ans = min(ans,tmp/d);
11. }
12. **return** ans;
13. }

**(八)求线段的中垂线**

1. //求线段的中垂线
2. **inline** Line getMidLine(**const** Point &a, **const** Point &b) {
3. Point mid = (a + b);
4. mid.x/=2.0;
5. mid.y/=2.0;
6. Point tp = b-a;
7. **return** Line(mid, mid+Point(-tp.y, tp.x));
8. }

**（九）直线相关**

1. **struct** Line
2. {
3. Point s,e;
4. Line(){}
5. Line(Point \_s,Point \_e)
6. {
7. s = \_s;
8. e = \_e;
9. }
10. **bool** operator ==(Line v)
11. {
12. **return** (s == v.s)&&(e == v.e);
13. }
14. **void** input()
15. {
16. s.input();
17. e.input();
18. }
19. //两线段相交判断
20. //2 规范相交
21. //1 非规范相交
22. //0 不相交
23. **int** segcrossseg(Line v)
24. {
25. **int** d1 = sgn((e-s)^(v.s-s));
26. **int** d2 = sgn((e-s)^(v.e-s));
27. **int** d3 = sgn((v.e-v.s)^(s-v.s));
28. **int** d4 = sgn((v.e-v.s)^(e-v.s));
29. **if**( (d1^d2)==-2 && (d3^d4)==-2 )**return** 2;
30. **return** (d1==0 && sgn((v.s-s)\*(v.s-e))<=0) ||
31. (d2==0 && sgn((v.e-s)\*(v.e-e))<=0) ||
32. (d3==0 && sgn((s-v.s)\*(s-v.e))<=0) ||
33. (d4==0 && sgn((e-v.s)\*(e-v.e))<=0);
34. }
35. //直线和线段相交判断
36. //-\*this line   -v seg
37. //2 规范相交
38. //1 非规范相交
39. //0 不相交
40. **int** linecrossseg(Line v)
41. {
42. **int** d1 = sgn((e-s)^(v.s-s));
43. **int** d2 = sgn((e-s)^(v.e-s));
44. **if**((d1^d2)==-2) **return** 2;
45. **return** (d1==0||d2==0);
46. }
47. };

(十)最小矩形覆盖

1. #include<stdio.h>
2. #include<string.h>
3. #include<algorithm>
4. #include<cmath>
5. #include<iostream>
6. **using** **namespace** std;
8. **typedef** **long** **long** LL;
10. **const** **double** eps = 1e-10;
12. **const** **int** maxn = 4e3+10;
14. **const** **double** inf = 1e17;
16. **int** dcmp(**double** x){
17. **if**(fabs(x)<eps) **return** 0;
18. **if**(x>0) **return** 1;
19. **return** -1;
20. }
22. **struct** point{
23. **double** x,y;
24. **bool** operator <(**const** **struct** point &tmp)**const**{
25. **if**(dcmp(x-tmp.x)==0) **return** dcmp(y-tmp.y)<0;
26. **return** dcmp(x-tmp.x)<0;
27. }
28. **bool** operator == (**const** **struct** point &tmp)**const**{
29. **return** dcmp(x-tmp.x)==0&&dcmp(y-tmp.y)==0;
30. }
31. }p[maxn],st[maxn];
33. **double** Cross(point a,point b,point c){
34. **return** (c.x-a.x)\*(b.y-a.y) - (b.x-a.x)\*(c.y-a.y);
35. }
37. **double** dot(point a,point b,point c){//点积
38. **double** s1=b.x-a.x;
39. **double** t1=b.y-a.y;
40. **double** s2=c.x-a.x;
41. **double** t2=c.y-a.y;
42. **return** s1\*s2+t1\*t2;
43. }
45. **int** ConvexHull(point \*p,**int** n,point \*st){
46. sort(p,p+n);
47. n=unique(p,p+n)-p;
48. **int** m=0;
49. **for**(**int** i=0; i<n; i++) {
50. **while**(m>1&&Cross(st[m-2],p[i],st[m-1])<=0) m--;
51. st[m++]=p[i];
52. }
53. **int** k=m;
54. **for**(**int** i=n-2; i>=0; i--){
55. **while**(m>k&&Cross(st[m-2],p[i],st[m-1])<=0)m--;
56. st[m++]=p[i];
57. }
58. **if**(n>1) m--;
59. **return** m;
60. }
62. **double** dist(point a,point b){
63. **return** sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));
64. }
66. **double** rotating\_calipers(point \*p,**int** n){
67. **int** R=1,U=1,L;
68. **double** ans = 10000000000000;
69. p[n]=p[0];
70. **for**(**int** i=0;i<n;i++){
71. **while**(dcmp(Cross(p[i],p[i+1],p[U+1])-Cross(p[i],p[i+1],p[U]))<=0) U=(U+1)%n;//最上面一点
72. **while**(dcmp(dot(p[i],p[i+1],p[R+1])-dot(p[i],p[i+1],p[R]))>0) R=(R+1)%n;//最右一点
73. **if**(i==0) L=R;
74. **while**(dcmp(dot(p[i],p[i+1],p[L+1])-dot(p[i],p[i+1],p[L]))<=0) L=(L+1)%n;//最左一点
75. **double** d=dist(p[i],p[i+1])\*dist(p[i],p[i+1]);
76. **double** area=fabs(Cross(p[i],p[i+1],p[U]))\*//求面积
77. fabs(dot(p[i],p[i+1],p[R])-dot(p[i],p[i+1],p[L]))/d;
78. **if**(area<ans) ans = area;
79. }
80. **return** ans;
81. }
83. **int** main(){
84. **int** n,t,cas=1;
85. scanf("%d",&t);
86. **while**(t--){
87. scanf("%d",&n);
88. n\*=4;
89. **for**(**int** i=0;i<n;i++){
90. scanf("%lf%lf",&p[i].x,&p[i].y);
91. }
92. **int** m = ConvexHull(p,n,st);
93. **double** ans;
94. **if**(m<3) ans=0;
95. **else** ans = rotating\_calipers(st,m);
96. LL tmp = ans+0.5;
97. printf("Case #%d:\n%I64d\n",cas++,tmp);
98. }
99. **return** 0;
100. }

**(十一)多个圆的面积交**

1. **typedef** **long** **long** LL;
2. **typedef** unsigned **long** **long** ULL;
3. **typedef** vector <**int**> VI;
4. **const** **int** INF = 0x3f3f3f3f;
5. **const** **double** eps = 1e-10;
6. **const** **int** MOD = 100000007;
7. **const** **int** MAXN = 1000010;
8. **const** **double** PI = acos(-1.0);
9. #define sqr(x) ((x)\*(x))
10. **const** **int** N = 1010;
11. **double** area[N];
12. **int** n;
14. **int** dcmp(**double** x)
15. {
16. **if** (x < -eps) **return** -1;
17. **else** **return** x > eps;
18. }
20. **struct** cp
21. {
22. **double** x, y, r, angle;
23. **int** d;
24. cp() {}
25. cp(**double** xx, **double** yy, **double** ang = 0, **int** t = 0)
26. {
27. x = xx;
28. y = yy;
29. angle = ang;
30. d = t;
31. }
32. **void** get()
33. {
34. scanf("%lf%lf%lf", &x, &y, &r);
35. d = 1;
36. }
37. } cir[N], tp[N \* 2];
39. **double** dis(cp a, cp b)
40. {
41. **return** sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
42. }
44. **double** cross(cp p0, cp p1, cp p2)
45. {
46. **return** (p1.x - p0.x) \* (p2.y - p0.y) - (p1.y - p0.y) \* (p2.x - p0.x);
47. }
49. **int** CirCrossCir(cp p1, **double** r1, cp p2, **double** r2, cp &cp1, cp &cp2)
50. {
51. **double** mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx \* mx;
52. **double** my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my \* my;
53. **double** sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) \* (sq - sqr(r1 + r2));
54. **if** (d + eps < 0) **return** 0;
55. **if** (d < eps) d = 0;
56. **else** d = sqrt(d);
57. **double** x = mx \* ((r1 + r2) \* (r1 - r2) + mx \* sx) + sx \* my2;
58. **double** y = my \* ((r1 + r2) \* (r1 - r2) + my \* sy) + sy \* mx2;
59. **double** dx = mx \* d, dy = my \* d;
60. sq \*= 2;
61. cp1.x = (x - dy) / sq;
62. cp1.y = (y + dx) / sq;
63. cp2.x = (x + dy) / sq;
64. cp2.y = (y - dx) / sq;
65. **if** (d > eps) **return** 2;
66. **else** **return** 1;
67. }
69. **bool** circmp(**const** cp& u, **const** cp& v)
70. {
71. **return** dcmp(u.r - v.r) < 0;
72. }
74. **bool** cmp(**const** cp& u, **const** cp& v)
75. {
76. **if** (dcmp(u.angle - v.angle)) **return** u.angle < v.angle;
77. **return** u.d > v.d;
78. }
80. **double** calc(cp cir, cp cp1, cp cp2)
81. {
82. **double** ans = (cp2.angle - cp1.angle) \* sqr(cir.r)
83. - cross(cir, cp1, cp2) + cross(cp(0, 0), cp1, cp2);
84. **return** ans / 2;
85. }
87. **void** CirUnion(cp cir[], **int** n)
88. {
89. cp cp1, cp2;
90. sort(cir, cir + n, circmp);
91. **for** (**int** i = 0; i < n; ++i)
92. **for** (**int** j = i + 1; j < n; ++j)
93. **if** (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)
94. cir[i].d++;
95. **for** (**int** i = 0; i < n; ++i)
96. {
97. **int** tn = 0, cnt = 0;
98. **for** (**int** j = 0; j < n; ++j)
99. {
100. **if** (i == j) **continue**;
101. **if** (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
102. cp2, cp1) < 2) **continue**;
103. cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
104. cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
105. cp1.d = 1;
106. tp[tn++] = cp1;
107. cp2.d = -1;
108. tp[tn++] = cp2;
109. **if** (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
110. }
111. tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, PI, -cnt);
112. tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -PI, cnt);
113. sort(tp, tp + tn, cmp);
114. **int** p, s = cir[i].d + tp[0].d;
115. **for** (**int** j = 1; j < tn; ++j)
116. {
117. p = s;
118. s += tp[j].d;
119. area[p] += calc(cir[i], tp[j - 1], tp[j]);
120. }
121. }
122. }
124. **void** solve()
125. {
126. scanf("%d", &n);
127. **for** (**int** i = 0; i < n; ++i)
128. cir[i].get();
129. memset(area, 0, **sizeof**(area));
130. CirUnion(cir, n);
131. //去掉重复计算的
132. **for** (**int** i = 1; i <= n; ++i)
133. {
134. area[i] -= area[i + 1];
135. }
136. //area[i]为重叠了i次的面积
137. //tot 为总面积
138. **double** tot = 0;
139. **for**(**int** i=1; i<=n; i++) tot += area[i];
140. printf("%f\n", tot);
141. }
143. **int** main()
144. {
145. //freopen("input.txt", "r", stdin);
146. **return** 0;
147. }

技巧：

康拓逆展开：

#include<cstdio>

#include<cstring>

const int fac[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320};///阶乘

bool vis[10];

///n为ans大小，k为全排列的编码

void invKT(int ans[], int n, int k)

{

int i, j, t;

memset(vis, 0, sizeof(vis));

--k;

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

{

t = k / fac[n - i - 1];

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

if (!vis[j])

{

if (t == 0) break;

--t;

}

ans[i] = j, vis[j] = true;

k %= fac[n - i - 1];///余数

}

}

int main()

{

int a[10];

invKT(a, 5, 16);

for (int i = 0; i < 5; ++i)

printf("%d ", a[i]);///1 4 3 5 2

}

康拓展开

#include<cstdio>

const int fac[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320};///阶乘

int KT(int s[], int n)

{

int i, j, cnt, sum;

sum = 0;

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

{

cnt = 0;

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

if (s[j] < s[i]) ++cnt;

sum += cnt \* fac[n - i - 1];

}

return sum;

}

int main()

{

int a[] = {3, 5, 7, 4, 1, 2, 9, 6, 8};

printf("%d\n", 1 + KT(a, sizeof(a) / sizeof(\*a))); ///1+98884

}