目录

[1. 最小生成树 2](#_Toc419364109)

[2.最短路算法 7](#_Toc419364110)

[3.素数打表 11](#_Toc419364111)

[4.最大匹配 12](#_Toc419364112)

[5.线段树（敌兵布阵） 14](#_Toc419364113)

[6．线段树（逆序树） 16](#_Toc419364114)

[7.树形dp 18](#_Toc419364115)

[8.树状数组（段跟新） 20](#_Toc419364116)

[9.Kmp模板 22](#_Toc419364117)

[10.线段树（点跟新） 26](#_Toc419364118)

[11.强连通 28](#_Toc419364119)

[12.最小割 31](#_Toc419364120)

[13.单源最短路（spfa） 34](#_Toc419364121)

[14.三分查找 36](#_Toc419364122)

[15.字典树（统计难题） 38](#_Toc419364123)

[16.最大流入门题 1273 40](#_Toc419364124)

[17.状态压缩 43](#_Toc419364125)

[18.匈牙利（HDU 2063）（最大匹配） 45](#_Toc419364126)

[19.凸包 （HDU1348） 47](#_Toc419364127)

[20.树状数组 (HDU1166) 50](#_Toc419364128)

[21.强连通 52](#_Toc419364129)

[22. 前向星 55](#_Toc419364130)

[23.矩阵 58](#_Toc419364131)

[24.并查集 60](#_Toc419364132)

[25. SORT 61](#_Toc419364133)

[26. STL 63](#_Toc419364134)

[27. LCA (HDU 2874) 67](#_Toc419364135)

[28. 01背包 70](#_Toc419364136)

[29. 状态压缩代码： 72](#_Toc419364137)

[30. 快速幂 74](#_Toc419364138)

[31.矩阵快速幂 75](#_Toc419364139)

[32.GCD & LCM 77](#_Toc419364140)

[33.ACM小技巧： 78](#_Toc419364141)

[34. /\*\* 大数（高精度）求幂 \*\*/ 80](#_Toc419364142)

[35. /\*\* 大数除法与求余 \*\*/ 82](#_Toc419364143)

[36. /\*\* 大数阶乘 \*\*/ 84](#_Toc419364144)

[37. /\*\* 大数乘法 \*\*/ 85](#_Toc419364145)

[38. /\*\* 大数累加 \*\*/ 86](#_Toc419364146)

### 最小生成树

要连通n个城市需要n－1条边线路。可以把边上的权值解释为线路的造价。则最小生成树表示使其造价最小的生成树。

prim算法（矩阵形式）：  
  
#define inf 0x3f3f3f3f  
int prim(int n,int sta)//n表示有n个顶点，sta表从sta这个顶点出发生成最小生成树  
{  
    int mark[M],dis[M];  
    int i,sum = 0;     //sum是总的最小生成树边权值  
    for (i = 0;i < n;i ++) //初始化dis[i] 表从顶点sta到点i的权值  
    {  
        dis[i] = mat[sta][i];  
        mark[i] = 0;  
    }  
    mark[sta] = 1;         //sta 这个顶点加入最小生成树中  
    for (i = 1;i < n;i ++) //循环n-1次，每次找出一条最小权值的边 n个点的图  
    {        //只有n-1条边  
        int min = inf;       //inf 表无穷大  
        for (j = 0;j < n;j ++)//找出当前未在最小生成树中边权最小的顶点  
            if (!mark[j] && dis[j] < min)  
                min = dis[j],flag = j;  
        mark[flag] = 1;      //把该顶点加入最小生成树中  
        sum += dis[flag];       //sum加上其边权值  
        for (j = 0;j < n;j ++)  //以falg为起点更新到各点是最小权值  
            if (dis[j] > mat[flag][j])  
                dis[j] = mat[flag][j];  
    }  
    return sum;       //返回边权总和  
}  
  
prim算法（边表形式）：  
  
struct Edge//frm为起点,to为终点，w为边权，nxt指向下一个顶点  
{  
   // int frm;  
    int to,w,nxt;  
}edge[M];  
  
int vis[M],head[M],dis[M];  
void addedge (int cu,int cv,int cw)//生成边的函数  
{  
    //edge[e].frm = cu;  
    edge[e].to = cv;  
    edge[e].w = cw;  
    edge[e].nxt = head[cu];  
    head[cu] = e ++;  
    //edge[e].frm = cv;  
    edge[e].to = cu;  
    edge[e].w = cw;  
    edge[e].nxt = head[cv];  
    head[cv] = e ++;  
}  
  
int prim(int n,int sta) //n为顶点数量，sta为起点  
{  
    int sum = 0;  
    memset(dis,0x3f,sizeof(dis));  
    memset(vis,0,sizeof(vis));  
    for (i = head[sta];i != -1;i = edge[i].nxt)//遍历与sta点相连的所有顶点  
    {  
        int v = edge[i].to;  
        dis[v] = edge[i].w;  
    }  
    vis[sta] = 1; //加入到最小生成树中  
    int m = n - 1;  //只生成n-1条边，所以循环n-1次  
    while (m --)  
    {  
        int min = inf;  
        for (i = 0;i < n;i ++)//找出当前边权最小的边  
            if (!vis[i]&&dis[i] < min)  
                flag = i,min = dis[i];  
        sum += dis[flag];  
        vis[flag] = 1;//加入到最小生成树中  
        for (i = head[flag];i != -1;i = edge[i].nxt) //更新与flag顶点相连的点的dis  
        {              
            int v = edge[i].to;  
            if (edge[i].w < dis[v])  
                dis[v] = edge[i].w;  
        }  
    }  
    return sum; //返回边权总和  
}  
  
int main ()  
{  
    e = 0;                  //记得初始化  
    memset (head,-1,sizeof(head));  
    scanf ("%d %d %d",&a,&b,&w);  
    addedge(a,b,w);  
    .....  
    .....  
    prim(n,sta);  
    return 0;  
}

Kruskal算法：  
struct Edge  
{  
    int v1,v2,w;  
}edge[M],tree[M];           //w为v1顶点到v2顶点的边权  
/ \*  
int Find (int parent[],int u)//第1种写法  
{  
    int tmp = u;  
    while (paren[tmp] != -1)  
        tmp = parent[tmp];  
    return tmp;  
}  
\*/  
  
int Find (int u)  //第2种写法  
{  
    if (u != parent[u])  
        parent[u] = Find(paren[u]);  
    return parent[u];  
}  
bool cmp (Edge a,Edge b)  
{  
    return a.w < b.w;  
}  
int Kruskal()//parent[]表示集合  
{  
    int parent[M];  
    int i,j,sum,vf1,vf2;  
    sort(edge,edge+E,cmp);  
  
  //  memset (parent,-1,sizeof(parent));//对应第1种并查集的初始化  
  
    for (i = 0;i < n;i ++)    //对应第2种并查集的初始化  
        parent[i] = i;  
  
    sum = i = j = 0;  
    while (i < E && j < N - 1)//生成的边数为N-1  
    {  
        vf1 = Find(parent,edge[i].v1);  //找这两个点的祖先  
        vf2 = Find(parent,edge[i].v2);  
        if (vf1 != vf2) //若两个点的祖先不同，说明不在同一集合  
        {  
            parent[vf2] = vf1;//把vf2点加到vf1点的集合中  
            tree[j++] = edge[i];//把边加到tree[]数组中，这句题目没要求可忽略之  
            sum += edge[i].w;    //sum 加上其边权  
        }  
        i ++;  
    }  
    return sum;  
}

最小生成树 -- Kruskal算法：运用数组存点与边的权值

#include <stdio.h>

#include <stdlib.h>

#include <algorithm>

#define N 150

using namespace std;

int m,n,u[N],v[N],w[N],p[N],r[N];

int cmp(const int i,const int j) {return w[i]<w[j];}

int find(int x) {return p[x]==x?x:p[x]=find(p[x]);}

int kruskal()

{

int cou=0,x,y,i,ans=0;

for(i=0;i<n;i++) p[i]=i;

for(i=0;i<m;i++) r[i]=i;

sort(r,r+m,cmp);

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

{

int e=r[i];x=find(u[e]);y=find(v[e]);

if(x!=y) {ans += w[e];p[x]=y;cou++;}

}

if(cou<n-1) ans=0;

return ans;

}

int main()

{

int i,ans;

while(scanf("%d%d",&m,&n)!=EOF&&m)

{

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

{

scanf("%d%d%d",&u[i],&v[i],&w[i]);

}

ans=kruskal();

if(ans) printf("%d\n",ans);

else printf("?\n",ans);

}

return 0;

}

### 2.最短路算法

①DIJK

C++代码

1. #define inf 0x3fffffff
2. #define M 105
4. int dist[M], map[M][M], n;
5. bool mark[M];
7. void init ()
8. {
9. int i, j;
10. for (i = 1; i <= n; i++)    //i==j的时候也可以初始化为0，只是有时候不合适
11. for (j = 1; j <= n; j++)
12. map[i][j] = inf;
13. }
15. void dijk (int u)
16. {
17. int i, j, mins, v;
18. for (i = 1; i <= n; i++)
19. {
20. dist[i] = map[u][i];
21. mark[i] = false;
22. }
23. mark[u] = true;
24. dist[u] = 0;    //既然上面的map当i==j时不是0，就要这句
25. while (1)
26. {
27. mins = inf;
28. for (j = 1; j <= n; j++)
29. if (!mark[j] && dist[j] < mins)
30. mins = dist[j], v = j;
31. if (mins == inf)
32. break;
33. mark[v] = true;
34. for (j = 1; j <= n; j++)
35. if (!mark[j] && dist[v] + map[v][j] < dist[j])
36. dist[j] = dist[v] + map[v][j];
37. }
38. }

②Floyd

1. #define inf 0x3fffffff  //注意，太大会溢出
2. #define M               //最大点数
3. int n, dist[M][M];           //n：实际点数
5. void init ()            //有时候需要初始化
6. {
7. int i, j;
8. for (i = 1; i <= n; i++)
9. for (j = i + 1; j <= n; j++)
10. dist[i][j] = dist[j][i] = inf;
11. }
13. void floyd ()
14. {
15. int i, j, k;
16. for (k = 1; k <= n; k++)
17. for (i = 1; i <= n; i++)
18. for (j = 1; j <= n; j++)    //有的题目会溢出就要自己变通了
19. if (dist[i][k] + dist[k][j] < dist[i][j])
20. dist[i][j] = dist[i][k] + dist[k][j];
21. }

③vector后插的SPFA

C++代码

1. #define inf 0x3fffffff
2. #define M 105    //最大点数
3. struct son{
4. int v, w;
5. };
6. vector<son> g[M];
7. bool inq[M];       //入队列标记
8. int dist[M], n;    //n：实际点数
10. void init ()
11. {
12. for (int i = 1; i <= n; i++)
13. g[i].clear();
14. }
16. void spfa (int u)
17. {
18. int i, v, w;
19. for (i = 1; i <= n; i++)
20. {
21. dist[i] = inf;
22. inq[i] = false;
23. }
24. queue<int> q;
25. q.push (u);
26. inq[u] = true;
27. dist[u] = 0;
28. while (!q.empty())
29. {
30. u = q.front();
31. q.pop();
32. inq[u] = false;
33. for (i = 0; i < g[u].size(); i++)
34. {
35. v = g[u][i].v;
36. w = g[u][i].w;
37. if (dist[u] + w < dist[v])
38. {
39. dist[v] = dist[u] + w;
40. if (!inq[v])
41. {
42. q.push (v);
43. inq[v] = true;
44. }
45. }
46. }
47. }
48. }

④模拟前插的SPFA(多数情况下比③快，数据较为复杂就会看出来)

C++代码

1. #define inf 0x3fffffff
2. #define M 1005  //最大点数
4. struct edge{
5. int v, w, next;
6. }e[10005];      //估计好有多少条边
8. int pre[M], cnt, dist[M], n;
9. bool inq[M];
10. //注意初始化
11. void init ()
12. {
13. cnt = 0;
14. memset (pre, -1, sizeof(pre));
15. }
16. //注意双向加边
17. void addedge (int u, int v, int w)    //加边函数，慢慢模拟就会明白的
18. {
19. e[cnt].v = v;
20. e[cnt].w = w;
21. e[cnt].next = pre[u];       //接替已有边
22. pre[u] = cnt++;             //自己前插成为u派生的第一条边
23. }
25. void spfa (int u)
26. {
27. int v, w, i;
28. for (i = 1; i <= n; i++) //对于从1到n的编号
29. dist[i] = inf, inq[i] = false;
30. dist[u] = 0;
31. queue<int> q;
32. q.push (u);
33. inq[u] = true;
34. while (!q.empty())
35. {
36. u = q.front();
37. q.pop();
38. inq[u] = false;
39. for (i = pre[u]; i != -1; i = e[i].next)
40. {
41. w = e[i].w;
42. v = e[i].v;
43. if (dist[u] + w < dist[v])
44. {
45. dist[v] = dist[u] + w;
46. if (!inq[v])
47. {
48. q.push (v);
49. inq[v] = true;
50. }
51. }
52. }
53. }
54. }

### 3.素数打表

#include<iostream>

#include<cstring>

#include<cmath>

#define MAXN 5000

using namespace std;

int prime[MAXN];

void print\_prime()

{

int n = (int) sqrt(MAXN);

for(int i = 2; i < n; i++)

{

if( !prime[i] )

{

for(int j = i\*i; j < MAXN; j += i)

prime[j] = 1;

prime[++prime[0]] = i;

}

}

for(int i = n; i < MAXN; i++)

if(!prime[i])

prime[++prime[0]] = i;

}

int main()

{

FILE \*out;

out = fopen("out.txt","w");

print\_prime();

for(int i = 1; i <= prime[0]; i++)

fprintf(out, "%d,", prime[i]);

return 0;

}

### 4.最大匹配

#include<stdio.h>

#include<string.h>

#define MAX 505

bool used[MAX];

bool match[MAX][MAX];

int boys[MAX],girls[MAX];

int k, n;

void init()

{

memset(girls, -1, sizeof(girls));

memset(boys, -1, sizeof(boys));

memset(match, false, sizeof(match));

}

bool can(int t)

{

int i;

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

{

if(!used[i] && match[t][i])

{

used[i] = true;

if(girls[i] = = -1 || can(girls[i]))

{

boys[t] = i;

girls[i] = t;

return true;

}

}

}

return false;

}

int main()

{

int tmpa, tmpb, i, res, t;

while(scanf("%d", &n) != EOF)

{

init();

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

{

scanf("%d: (%d)", &tmpa, &k);

while(k- -)

{

scanf("%d", &tmpb);

match[tmpa][tmpb] = true;

res = 0;

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

{

if(boys[i] == -1)

{

memset(used, false, sizeof(used));

if(can(i)) res ++;

}

}

}

}

printf("%d\n", n - res/2);

}

return 0;

}

### 5.线段树（敌兵布阵）

#include<stdio.h>

#include<string.h>

int a[50005],n;

void update(int x,int c){

int i;

for(i=x;i<=n;i+=(i&(-i)))

a[i]+=c;

}

int s(int x){

int i;

int sum=0;

for(i=x;i>0;i-=(i&-i))

{

sum+=a[i];}

return sum;

}

int main()

{

int t,i,c,b=1;

int z,y;

char sh[15];

scanf("%d",&t);

while(t--)

{

memset(a,0,sizeof(a));

scanf("%d",&n);

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

{

scanf("%d",&c);

update(i,c);

}

printf("Case %d:\n",b++);

while(~scanf("%s",sh))

{

if(sh[0]=='E')

break;

scanf("%d%d",&z,&y);

if(sh[0]=='A')

update(z,y);

else if(sh[0]=='S')

update(z,-y);

else printf("%d\n",s(y)-s(z-1));

}

}

return 0;

}

### 6．线段树（逆序树）

#include<stdio.h>

#include<string.h>

#include<algorithm>

using namespace std;

int a[500005],n;

int b[500005];

int d[500005];

void update(int x,int c){

int i;

for(i=x;i<=n;i+=(i&(-i)))

a[i]+=c;

}

int s(int x){

int i;

int sum=0;

for(i=x;i>0;i-=(i&-i))

{

sum+=a[i];}

return sum;

}

int main()

{

int i,j;

int ans;

while(scanf("%d",&n)!=EOF,n)

{

memset(d,0,sizeof(d));

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

{

scanf("%d",&b[i]);

b[i]=b[i]+1;

d[i]=b[i];

}

sort(b+1,b+n+1);

memset(a,0,sizeof(a));

ans=0;

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

{

update(d[i],1);

printf("%d ",s(d[i]));

ans+=i-s(d[i]);

printf("%d ",ans);

}

printf("%d\n",ans);

}

return 0;

}

### 7.树形dp

#include<stdio.h>

#include<iostream>

using namespace std;

struct Tree{

int father,child,brother,with\_max,without\_max;

int MAX()

{

return with\_max>=without\_max?with\_max:without\_max;

}

void init()

{

father=child=brother=without\_max=0;

}

}tree[6001];

void dfs(int id)

{

int child;

child=tree[id].child;

while(child)

{

dfs(child);

tree[id].with\_max+=tree[child].without\_max;

tree[id].without\_max+=tree[child].MAX();

child=tree[child].brother;

}

}

int main()

{

int n,i;

while(scanf("%d",&n)!=EOF)

{

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

{

scanf("%d",&tree[i].with\_max);

tree[i].init();

}

int a,b;

while(scanf("%d%d",&a,&b),a||b)

{

tree[a].father=b;

tree[a].brother=tree[b].child;

tree[b].child=a;

}

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

if(!tree[i].father)

{

dfs(i);

printf("%d\n",tree[i].MAX());

break;

}

}

return 0;

}

### 8.树状数组（段跟新）

1. #include <stdio.h>
2. #include <string.h>
3. const int MAXN=110000;
4. int n,c[MAXN];
5. int lowbit(int x)
6. //计算2^k
7. {
8. x=x&-x;
9. return x;
10. }
11. void update(int num,int val)
12. //向下查询，num是要更新的子节点，val是要修改的值
13. {
14. while(num>0)
15. {
16. c[num]+=val;
17. num-=lowbit(num);
18. }
19. }
20. int getSum(int num)
21. //向上统计每个区间被染色的次数
22. {
23. int sum=0;
24. while(num<=n)
25. {
26. sum+=c[num];
27. num+=lowbit(num);
28. }
29. return sum;
30. }
31. int main()
32. {
33. int a,b;
34. while(scanf("%d",&n),n)
35. {
36. memset(c,0,sizeof(c));
37. for(int i=0;i<n;i++)
38. {
39. scanf("%d%d",&a,&b);
40. //将b以下区间+1
41. update(b,1);
42. //将a以下区间-1
43. update(a-1,-1);
44. }
45. for(int j=1;j<n;j++)
46. {
47. printf("%d ",getSum(j));
48. }
49. printf("%d\n",getSum(n));
50. }
51. return 0;
52. }

### 9.Kmp模板

#include<stdio.h>

#include<string.h>

#include<iostream>

using namespace std;

#define N 1000010

int len,len1,next[N];

int str[N],str1[N];

void get\_next()

{

int i=0,j=-1;

next[0]=-1;

while(i<=len1)

{

if(j==-1||str1[i]==str1[j])

{

i++;j++;

next[i]=j;

}

else

j=next[j];

}

}

int kmp()

{

int i=0,j=0;

{

if(j==-1||str[i]==str1[j])

{

i++;j++;

}

else

{

j=next[j];

}

}

if(j>=len1)

{

return i-len1+1;

}

return 0;

}

int main()

{

int i,t,flag;

scanf("%d",&t);

while(t--)

{

memset(str,0,sizeof(str));

memset(str1,0,sizeof(str1));

memset(next,0,sizeof(next));

scanf("%d%d",&len,&len1);

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

scanf("%d",&str[i]);

for(i=0;i<len1;i++)

scanf("%d",&str1[i]);

get\_next();

flag=kmp();

// printf("%d\n",flag);

if(flag==0)

printf("-1\n");

else

printf("%d\n",flag);

}

return 0;

}

计算出字符串 str2 中含有的 str1 的个数

#include<stdio.h>

#include<string.h>

int next[10005];

char str1[10005], str2[1000005];

int n, m, ans;

void getNext(char \*p, int \*next)

{

int j, k;

next[0] = -1;

j = 0;

k = -1;

while(j < m)

{

if(k == -1 || p[j] == p[k]) //匹配的情况下,p[j]==p[k]

{

j++;

k++;

next[j] = k;//printf("%d \*\*%d \*\*%d\n",j, k, next[j]);

}

else //p[j]!=p[k]

k = next[k];

}

}

int KMPMatch(char \*s, char \*p)

{

int i = 0, j = 0;

getNext(p, next);

/\*for(i = 1; i < m; i++) printf("%d ", next[i]);

i = 0; printf("\n");\*/

while(i < n)

{

if(j == -1 || s[i] == p[j])

{

i++;

j++;

}

else

{

j = next[j]; //消除了指针i的回溯printf("i = %d, j = %d\n", i, j),

}

if(j == m )

ans++;//return i - m + 1;

}

return ans;

}

int main()

{

int T;

scanf("%d",&T);

while(T--)

{

memset(str1, '\0', sizeof(str1));

memset(str2, '\0', sizeof(str2));

memset(next, 0, sizeof(next));

ans = 0;

scanf("%s", str1);

scanf("%s", str2);

n = strlen(str2);

m = strlen(str1);

printf("%d\n",KMPMatch(str2, str1));

}

return 0;

}

### 10.线段树（点跟新）

#include<stdio.h>

#include<string.h>

#define N 200010

struct Node{

int l,r,max;

}node[3\*N];

int score[N];

int max(int a,int b){

return a>b?a:b;

}

void Build(int left,int right,int index)

{

node[index].l=left;

node[index].r=right;

if(left==right)

node[index].max=score[left];

else

{

int mid;

mid=(left+right)/2;

Build(left,mid,index\*2);

Build(mid+1,right,index\*2+1);

node[index].max=max(node[2\*index].max,node[2\*index+1].max);

}

}

void Update(int stu,int c,int index)

{

node[index].max=max(c,node[index].max);

if(node[index].l==node[index].r)

{

return ;

}

if(stu<=node[2\*index].r)

{

Update(stu,c,index\*2);

}

else

{

Update(stu,c,index\*2+1);

}

}

int query(int left,int right,int index)

{

if(left==node[index].l&&right==node[index].r)

{

return node[index].max;

}

if(right<=node[2\*index].r)

{

return query(left,right,index\*2);

}

if(left>=node[2\*index+1].l)

{

return query(left,right,index\*2+1);

}

int mid=(node[index].l+node[index].r)/2;

return max(query(left,mid,2\*index),query(mid+1,right,index\*2+1));

}

int main()

{

int n,m;

while(scanf("%d%d",&n,&m)!=EOF)

{

int i;

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

scanf("%d",&score[i]);

getchar();

char c;

int s,e;

Build(1,n,1);

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

{

scanf("%c%d%d",&c,&s,&e);

getchar();

if(c=='U')

{

Update(s,e,1);

}

if(c=='Q')

{

printf("%d\n",query(s,e,1));

}

}

}

return 0;

### 11.强连通

# include<stdio.h>

# include<string.h>

# define N 1005

# define M 2005

struct node{

int from,to,next;

}edge1[M],edge2[M];

int visit1[N],visit2[N],head1[N],head2[N],T[N];

int in\_degree[N],out\_degree[N],tol1,tol2,Bcnt,Tcnt;

int val[N],v1[N],min,Belong[N];

void add(int a,int b)

{

edge1[tol1].from=a;edge1[tol1].to=b;edge1[tol1].next=head1[a];head1[a]=tol1++;

edge2[tol2].from=b;edge2[tol2].to=a;edge2[tol2].next=head2[b];head2[b]=tol2++;

}

void dfs1(int x)

{

int i;

visit1[x]=1;

for(i=head1[x];i!=-1;i=edge1[i].next)

if(visit1[edge1[i].to]==0) dfs1(edge1[i].to);

T[Tcnt++]=x;

}

void dfs2(int x)

{

int i;

Belong[x]=Bcnt;

if(val[x]<min) min=val[x];

visit2[x]=1;

for(i=head2[x];i!=-1;i=edge2[i].next)

if(visit2[edge2[i].to]==0) dfs2(edge2[i].to);

}

int main()

{

int i,n,m,sum,x,y,a,b,count;

while(scanf("%d%d",&n,&m)!=EOF)

{

tol1=tol2=0;

Bcnt=Tcnt=0;

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

{

head1[i]=-1;

head2[i]=-1;

visit1[i]=0;

visit2[i]=0;

scanf("%d",&val[i]);

}

while(m--)

{

scanf("%d%d",&a,&b);

add(a,b);

}

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

if(visit1[i]==0) dfs1(i);

for(i=Tcnt-1;i>=0;i--)

{

if(visit2[T[i]]==0)

{

min=0xfffffff;

dfs2(T[i]);

v1[Bcnt]=min;

Bcnt++;

}

}

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

{

in\_degree[i]=0;

out\_degree[i]=0;

}

for(i=0;i<tol1;i++)

{

x=Belong[edge1[i].from];

y=Belong[edge1[i].to];

if(x!=y)

{

in\_degree[y]=1;

out\_degree[x]=1;

}

}

sum=0;

count=0;

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

{

if(in\_degree[i]==0) {sum+=v1[i];count++;}

}

printf("%d %d\n",count,sum);

}

return 0;

}

### 12.最小割

#include<stdio.h>

#include<string.h>

#include<algorithm>

#include<iostream>

#include<math.h>

#define oo 10000000

#define M 1000000

#define N 100000

using namespace std;

int head[N],mark[N],work[N],Q[M];

int map[205][205],dist[N];

int tot,src,dest,n,m;

struct node{

int to,cap,next;

}edge[M];

void add(int a,int b,int c,int d){

edge[tot].to=b,edge[tot].cap=c,edge[tot].next=head[a],head[a]=tot++;

edge[tot].to=a,edge[tot].cap=d,edge[tot].next=head[b],head[b]=tot++;

}

int min(int a,int b){

return a<b?a:b;

}

bool BFS(){

int i,j,k,l,h,y;

for(i=src;i<=dest;i++)

dist[i]=-1;

dist[src]=0;

l=h=0;

Q[l++]=src;

while(h<l){

k=Q[h++];

for(i=head[k];i!=-1;i=edge[i].next)

{

y=edge[i].to;

if(edge[i].cap>0&&dist[y]==-1)

{

dist[y]=dist[k]+1;

Q[l++]=y;

}

}

}

return (dist[dest]>=0);

}

int DFS(int x,int exp){

mark[x]=-1;

if(x==dest)

return exp;

for(int temp,y,i=work[x];i!=-1;i=edge[i].next,work[x]=i)

{

y=edge[i].to;

if(edge[i].cap>0&&dist[y]==dist[x]+1&&!mark[y])

{

if((temp=DFS(y,min(edge[i].cap,exp)))>0)

{

edge[i].cap-=temp;

edge[i^1].cap+=temp;

return temp;

}

}

}

return 0;

}

int Dinic\_flow(){

int ans=0,flow,i;

while(BFS()){

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

work[i]=head[i];

while(1){

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

mark[i]=0;

flow=DFS(src,oo);

if(flow==0)

break;

ans+=flow;

}

}

return ans;

}

int main()

{

int i,j,x,y,k,v=1;

while(scanf("%d%d",&n,&m)!=EOF)

{

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

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

scanf("%d",&map[i][j]);

memset(head,-1,sizeof(head));

tot=0;

src=0;dest=n\*m+1;

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

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

{

if(j+1<=m)

add((i-1)\*m+j,(i-1)\*m+j+1,1,1);

if(i+1<=n)

add((i-1)\*m+j,i\*m+j,1,1);

if(map[i][j]==1)

add(src,(i-1)\*m+j,oo,0);

else if(map[i][j]==2)

add((i-1)\*m+j,dest,oo,0);

}

printf("Case %d:\n%d\n",v++,Dinic\_flow());

}

return 0;

}

### 13.单源最短路（spfa）

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<queue>

#define false 0

#define true 1

using namespace std;

const int maxn=100000;

const int inf=0x7ffffff;

struct edge

{

int from,to,w,p,next;

}e[100001];

int head[maxn];

int vis[maxn];

int dist[maxn];

int cost[maxn];

int n,m,t,sum;

void add(int i,int j,int w,int p){

e[t].from=i;

e[t].to=j;

e[t].w=w;

e[t].p=p;

e[t].next=head[i];

head[i]=t++;

}

void spfa(int s)

{

queue<int> q;

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

{

dist[i]=inf;

cost[i]=inf;}

memset(vis,false,sizeof(vis));

q.push(s);

dist[s]=0;

cost[s]=0;

while(!q.empty())

{

int u=q.front();

q.pop();

vis[u]=false;

for(int i=head[u];i!=-1;i=e[i].next)

{

int v=e[i].to;

if(dist[v]>dist[u]+e[i].w)

{

dist[v]=dist[u]+e[i].w;

cost[v]=cost[u]+e[i].p;

if(!vis[v])

{

vis[v]=true;

q.push(v);

}

}

if(dist[v]==dist[u]+e[i].w&&cost[v]>cost[u]+e[i].p)

cost[v]=cost[u]+e[i].p;

}

}

}

int main()

{

int a,b,c,s,e,p;

while(scanf("%d%d",&n,&m)!=EOF,n,m)

{

t=0;

memset(head,-1,sizeof(head));

while(m--)

{

scanf("%d%d%d%d",&a,&b,&c,&p);

add(a,b,c,p);

add(b,a,c,p);

}

scanf("%d%d",&s,&e);

spfa(s);

printf("%d %d\n",dist[e],cost[e]);

}

return 0;

}

### 14.三分查找

static bool Find(int[] sortedArray, int number)  
        {  
            if (sortedArray.Length == 0)  
                return false;

            int start = 0;  
            int end = sortedArray.Length - 1;

            while (end >= start)  
            {  
                int firstMiddle = (end - start) / 3 + start;  
                int secondMiddle = end - (end - start) / 3;  
                if (sortedArray[firstMiddle] > number)  
                    end = firstMiddle - 1;  
                else if (sortedArray[secondMiddle] < number)  
                    start = secondMiddle + 1;  
                else if (sortedArray[firstMiddle] != number && sortedArray[secondMiddle] != number)  
                {  
                    end = secondMiddle - 1;  
                    start = firstMiddle + 1;  
                }  
                else  
                    return true;  
            }  
            return false;  
        }

二分查找

int binary\_search(int\* a, int len, int goal)

{

int low = 0;

int high = len - 1;

while(low <= high)

{

int middle = (low + high)/2;

if(a[middle] == goal)

return middle;

//在左半边

else if(a[middle] > goal)

high = middle - 1;

//在右半边

else

low = middle + 1;

}

//没找到

return -1;

}

### 15.字典树（统计难题）

#include<stdio.h>

#include<string.h>

#include <stdlib.h>

//typedef struct Node;

struct Node

{

int num;

struct Node \*next[26];

};

struct Node \*root;

void Insert(char \*s)

{

struct Node \*p, \*q;

int i = 0, len = strlen(s);

p = root;

while( i < len)

{

if( p->next[s[i] - 'a'] != NULL)

{

p = p ->next[s[i] - 'a'];

p -> num++;

}

else

{

q = (Node \*)malloc( sizeof( Node));

memset( q, NULL, sizeof(Node));

p->next[s[i] - 'a'] = q;

p = q;

p -> num++;

}

i++;

}

}

int Select(char \*s)

{

Node \*p;

int i = 0, len = strlen(s);

p = root;

while( (p!=NULL) && i < len)

{

if( p->next[s[i] - 'a'] != NULL)

{

p = p->next[s[i] - 'a'];

}

else break;

i++;

}

if(p != NULL && i == len) return p->num;

else return 0;

}

int main()

{

int n, m, i;

char s1[15],s2[15];

root = (struct Node\*)malloc(sizeof(struct Node));

memset(root,NULL,sizeof(struct Node));

while(~scanf("%d", &n))

{

getchar();

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

{

gets(s1);

if(s1[0] == '\0') break;

Insert(s1);

}

scanf("%d",&m);getchar();

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

{

gets(s2);

printf("%d\n",Select(s2));

}

}

return 0;

}

### 16.最大流入门题 1273

可增广路径

所谓可增广路径，是指这条路径上的流可以修改，通过修改，使得整个网络的流值增大。设f是一个可行流，P是从源点s到汇点t的一条路，若p满足下列条件：

(1)在p上的所有前向弧(vi→vj)都是非饱和弧，即0≤fij<cij

(2)在p上的所有后向弧(vi←vj)都是非零弧，即0<fij≤cij则称p为(关于可行流f的)一条可增广路径。

#include<stdio.h>

#include<string.h>

#include<queue>

#define N\_node 550

#define N\_edge 30000

#define INF 1000000000

using namespace std;

typedef struct

{

int from ,to ,next ,cost;

}STAR;

typedef struct

{

int x ,t;

}DEP;

STAR E[N\_edge];

DEP xin ,tou;

int deep[N\_node];

int list[N\_node] ,listt[N\_node] ,tot;

void add(int a ,int b ,int c)

{

E[++tot].from = a;

E[tot].to = b;

E[tot].cost = c;

E[tot].next = list[a];

list[a] = tot;

E[++tot].from = b;

E[tot].to = a;

E[tot].cost = 0;

E[tot].next = list[b];

list[b] = tot;

}

int minn(int x ,int y)

{

return x < y ? x : y;

}

bool bfs\_deep(int s ,int t ,int n)

{

xin.x = s;

xin.t = 0;

queue<DEP>q;

q.push(xin);

memset(deep ,255 ,sizeof(deep));

deep[s] = 0;

while(!q.empty())

{

tou = q.front();

q.pop();

for(int k = list[tou.x] ;k ;k = E[k].next)

{

xin.x = E[k].to;

xin.t = tou.t + 1;

if(deep[xin.x] != -1 || !E[k].cost)

continue;

deep[xin.x] = xin.t;

q.push(xin);

}

}

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

listt[i] = list[i];

return deep[t] != -1;

}

int DFS(int s ,int t ,int flow)

{

if(s == t) return flow;

int nowflow = 0;

for(int k = listt[s] ;k ;k = E[k].next)

{

listt[s] = k;

int c = E[k].cost;

int to = E[k].to;

if(!c || deep[to] != deep[s] + 1)

continue;

int tmp = DFS(to ,t ,minn(c ,flow - nowflow));

nowflow += tmp;

E[k].cost -= tmp;

E[k^1].cost += tmp;

if(nowflow == flow) break;

}

if(!nowflow) deep[s] = 0;

return nowflow;

}

int DINIC(int s ,int t ,int n)

{

int ans = 0;

while(bfs\_deep(s ,t ,n))

{

ans += DFS(s ,t ,INF);

}

return ans;

}

int main ()

{

int n ,m ,i ,j;

int a ,b ,c;

while(~scanf("%d %d" ,&n ,&m))

{

memset(list ,0 ,sizeof(list));

tot = 1;

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

{

scanf("%d %d %d" ,&a ,&b ,&c);

add(a ,b ,c);

}

printf("%d\n" ,DINIC(1 ,n ,1));

}

return 0;}

### 17.状态压缩

//状态压缩DP POJ3254

#include<stdio.h>

#include<string.h>

#define mod 100000000

#define M 15

int map[M],s[1<<M],dp[M][1<<M];

int main()

{

int m,n,i,j,k,t,cnt,sum;

while(~scanf("%d%d",&m,&n))

{

cnt=0;

sum=0;

memset(dp,0,sizeof(dp));

memset(map,0,sizeof(map));

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

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

{

scanf("%d",&t);

if(t)

map[i]+=t<<(n-j);

}

for(i=0;i<1<<n;i++)

if((i&(i<<1))==0)

s[cnt++]=i;

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

if((map[0]|s[i])==map[0])

dp[0][i]++;

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

for(j=0;j<cnt;j++)

if((map[i]|s[j])==map[i])

{

for(k=0;k<cnt;k++)

if((map[i-1]|s[k])==map[i-1])

if((s[k]&s[j])==0)

dp[i][j]+=dp[i-1][k];

if(dp[i][j]>mod)

dp[i][j]%=mod;

}

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

{

sum+=dp[m-1][i];

if(sum>mod)

sum%=mod;

}

printf("%d\n",sum);

}

return 0;

}

### 18.匈牙利（HDU 2063）（最大匹配）

//二分匹配

#include<stdio.h>

#include<string.h>

#define MAXK 1005

#define MAXN 505

struct node

{

int to;

int next;

}edge[MAXK<<1];

//一个标记是否走过，一个标记是否已经匹配

int head[MAXN],markd[MAXN],markp[MAXK];

int n,m,cnt;

void add(int u,int v)

{

cnt++;

edge[cnt].next=head[u];

edge[cnt].to=v;

head[u]=cnt;

}

int dfs(int x) //匈牙利

{

int i;

for(i=head[x];i;i=edge[i].next)

if(markd[edge[i].to]==0)

{

markd[edge[i].to]=1;

if(markp[edge[i].to]==-1||dfs(markp[edge[i].to]))

{

markp[edge[i].to]=x;

return 1;

}

}

return 0;

}

int main()

{

int k,i,j,u,v,s;

while(~scanf("%d",&k)&&k)

{

cnt=0;

s=0;

memset(head,0,sizeof(head));

memset(markp,-1,sizeof(markp));

scanf("%d%d",&n,&m);

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

{

scanf("%d%d",&u,&v);

add(u,v);

}

for(i=1;i<=n;i++) //n个女生

{

memset(markd,0,sizeof(markd)); //需要每次都更新

if(dfs(i))

s++;

}

printf("%d\n",s);

}

return 0;

}

### 19.凸包 （HDU1348）

#include<stdio.h>

#include<math.h>

#include<algorithm>

using namespace std;

#define pi 3.1415926

#define MAX 999999999

struct node

{

int x,y;

}point[1010],stack[1010];

int n;

//计算距离；

double dis(struct node a,struct node b)

{

return sqrt(double((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y)));

}

//向量叉乘函数；

int multi(struct node a,struct node b,struct node c)

{

return (a.x-c.x)\*(b.y-c.y)-(b.x-c.x)\*(a.y-c.y);

}

//斜率排序比较函数；

int cmp(struct node a,struct node b)

{

if(multi(a,b,point[0])>0)

return 1;

if(multi(a,b,point[0])==0&&dis(a,point[0])<dis(b,point[0]))

return 1;

return 0;

}

//Graham\_scan算法；

int Graham\_scan()

{

int i,x,k,top;

struct node t;

x=MAX;

k=0;

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

if(point[i].x<x||point[i].x==x&&point[i].y<point[k].y)

{

k=i;

x=point[i].x;

}

t=point[k];

point[k]=point[0];

point[0]=t;

sort(point+1,point+n,cmp);

stack[0]=point[0];

stack[1]=point[1];

top=1;

for(i=2;i<n;i++)

{

while(top>1&&multi(point[i],stack[top],stack[top-1])>=0)//注意

top--;

stack[++top]=point[i];

}

return top;

}

int main()

{

int i,t,T,L,m;

double len;

t=0;

scanf("%d",&T);

while(T--)

{

scanf("%d%d",&n,&L);

len=2\*pi\*L;

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

scanf("%d%d",&point[i].x,&point[i].y);

m=Graham\_scan();

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

len+=dis(stack[i],stack[i+1]);//

len+=dis(stack[m],stack[0]);//注意是stack数组；

if(t)

printf("\n");

printf("%.0lf\n",len);

t++;

}

return 0;

}

int rotating\_calipers() //旋转卡壳

{

int q=1,ans=0,p;

stack[++top]=stack[0];

for(p=0; p<top; p++)

{ //叉乘表示面积，也表示一定距离的点，相互比较；

while(xmult(stack[p],stack[p+1],stack[q+1])>xmult(stack[p],stack[p+1],stack[q]))

q=(q+1)%top;

ans=max(ans,max(Distance(stack[p],stack[q]),Distance(stack[p+1],stack[q+1])));

}

return ans;

}

### 20.树状数组 (HDU1166)

//敌兵布阵

#include<stdio.h>

#include<string.h>

int a[50010],n;

void up(int x,int c)

{

int i;

for(i=x;i<=n;i+=i&(-i))

a[i]+=c;

}

int sum(int x)

{

int s=0;

while(x>0)

{

s+=a[x];

x-=x&(-x);

}

return s;

}

int main()

{

int T=1,ca,i,j,x,p,q,ss;

char s[10];

scanf("%d",&ca);

while(ca--)

{

scanf("%d",&n);

printf("Case %d:\n",T);

T++;

memset(a,0,sizeof(a));

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

{

scanf("%d",&x);

up(i,x);

}

while(~scanf("%s",s))

{

if(s[0]=='E')

break;

else

{

scanf("%d%d",&p,&q);

if(s[0]=='Q')

{

ss=sum(q)-sum(p-1);

printf("%d\n",ss);

}

else

{

if(s[0]=='S')

up(p,-q);

else if(s[0]=='A')

up(p,q);

}

}

}

}

return 0;

}

### 21.强连通

//HDU 1269

题意：任意两个房间都是相互连通的，即：对于任意的i和j，至少存在一条路径可以从房间i到房间j，也存在一条路径可以从房间j到房间i。

#include<stdio.h>

#include<iostream>

#include<stack>

#define M 200005

#define N 20005

using namespace std;

struct node

{

int next;

int to;

}edge1[M],edge2[M];

int head1[N],head2[N],mark[N],cnt;

stack<int>s;

//两个add，存正反两条边；

void add(int u,int v)

{

cnt++;

edge1[cnt].next=head1[u];

edge1[cnt].to=v;

head1[u]=cnt;

edge2[cnt].next=head2[v];

edge2[cnt].to=u;

head2[v]=cnt;

}

//两次深搜，这是第一次深搜，对原图深搜，对访问结束

//进行入栈；

void dfs1(int u)

{

int i,v;

mark[u]=1;

for(i=head1[u];i;i=edge1[i].next)

{

v=edge1[i].to;

if(mark[v]==0)

dfs1(v);

}

s.push(u);

}

//第二次深搜，得到强连通分量；

void dfs2(int u)

{

int i,v;

mark[u]=1;

for(i=head2[u];i;i=edge2[i].next)

{

v=edge2[i].to;

if(mark[v]==0)

dfs2(v);

}

}

int main()

{

int n,m,i,a,b,t,sum;

while(~scanf("%d%d",&n,&m))

{

if(n==0&&m==0)

break;

cnt=0;

while(!s.empty())

{s.pop();}

memset(head1,0,sizeof(head1));

memset(head2,0,sizeof(head2));

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

{

scanf("%d%d",&a,&b);

add(a,b);

}

sum=0;

memset(mark,0,sizeof(mark));

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

if(mark[i]==0)

{

sum++;

if(sum>1)

{

printf("No\n");

break;

}

dfs1(i);

}

if(sum>1)

continue;

sum=0;

memset(mark,0,sizeof(mark));

while(!s.empty())

{

t=s.top();

s.pop();

if(mark[t]==0)

{

sum++;

if(sum>1)

{

printf("No\n");

break;

}

dfs2(t);

}

}

if(sum>1)

continue;

printf("Yes\n");

}

return 0;

}

### 前向星

链式前向星

#include <cstdio>

#include <cstring>

#define MAXM 1000005

#define MAXN 10005

/\*其中edge[i].to表示第i条边的终点,edge[i].next表示与第i条边同

起点的下一条边的存储位置,edge[i].w为边权值.另外还有一个数组head[i],

它是用来表示以i为起点的第一条边存储的位置,实际上你会发现这里的

第一条边存储的位置其实在以i为起点的所有边的最后输入的那个编号.\*/

struct node

{

int next;

int to;

int w;

}edge[MAXM<<1]; //边的两倍；

int head[MAXN], cnt;

int n, m;

void add(int u, int to)

{

cnt++;

edge[cnt].next = head[u];

edge[cnt].to = to;

head[u] = cnt;

}

int main()

{

int i,j,u,to;

while (~scanf("%d%d", &n, &m))

{

memset(head, 0, sizeof(head));

cnt = 0;

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

{

scanf("%d%d", &u, &to);

add(u, to);

// add(to, u); 双方向

}

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

{

printf("%d:",i);

for (j=head[i]; j; j=edge[j].next)

printf("%d ", edge[j].to);

printf("\n");

}

}

return 0;

}

普通前向星

#define N\_node 1000 //点的个数

#define N\_edge 5000 //边的条数

typedef struct //节点信息

{

int to ,next; //to 点，next E中下标信息

int cost; //当前边的权值

}STAR;

STAR E[N\_edge]; //节点信息数组

int list[N\_node]; //记录该点所连接的一系列点中的一个的E数组下标

int tot; //记录当前E数组已经用到下标几了

void add(int a ,int b ,int c) //添加一条a 到 b 的权值为c的边

{

E[++tot].to = b; //创建出当前节点b

E[tot].cost = c; //到达当前节点的权值

E[tot].next = list[a]; //把当前节点插入到与a相连的所有点的最前方

list[a] = tot; //节点a指向新加的，也就是插入到最前方的这个点

}

//存边之前记得把list数组清空，还有把list数组当前所有下标赋值成1

memset(list ,0 ,sizeof(list));

tot = 1;

//遍历，输出点a所连接的所有点b

/\*for(int i = 1; i <= n; i++)

{

for(int k = list[i] ;k ;k = E[k].next) //从头开始一个一个的遍历与点a相连的所有点

{

int b = E[k].to;

printf("%d %d\n", i, b);

}

}\*/

for(int k = list[a] ;k ;k = E[k].next) //从头开始一个一个的遍历与点a相连的所有点

{

int b = E[k].to;

printf("%d\n" ,b);

}

### 23.矩阵

//矩阵 只有偶数与（|）1才是加1；

struct Matrix

{

int a[15][15];

}per;

//矩阵加法

Matrix add(Matrix a,Matrix b)

{

int i,j;

Matrix c;

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

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

c.a[i][j]=(a.a[i][j]+b.a[i][j])%mod;

return c;

}

//矩阵乘法

Matrix multi(Matrix a,Matrix b)

{

int i,j,k;

Matrix c;

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

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

{

c.a[i][j]=0;

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

c.a[i][j]+=(a.a[i][k]\*b.a[k][j])%M;

c.a[i][j]=c.a[i][j]%M;

}

return c;

}

//矩阵快速幂

Matrix pm(Matrix a)

{

Matrix s=per;//初始化

while(m)

{

if(m&1)

{

s=multi(s,a);

m--;

}

m>>=1;

a=multi(a,a);

}

return s;

}

//单位矩阵的初始化

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

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

per.a[i][j]=(i==j);

### 24.并查集

#include<iostream>

#include<cstring>

#include<cstdio>

#include<cstdlib>

using namespace std;

int father[50002], a, b, m, n, p;

int find(int x)

{

if (father[x] != x) father[x] = find(father[x]);

return father[x];

}

int main()

{

scanf("%d%d%d",&n, &m, &p);

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

father[i]=i;

for (int i=1;i<=m;i++)

{

scanf("%d%d",&a,&b);

a=find(a),b=find(b);

father[a]=b;

}

for(int i = 1; i <= p; i++)

{

scanf("%d%d", &a, &b);

a = find(a);

b = find(b);

if(a == b) printf("Yes");

else printf("No");

}

return 0;

}

### 25. SORT

sort与stable\_sort

这两个函数的原理都是快速排序，时间复杂度在所有排序中最低，为O(nlog2n) ;

sort的应用；

1、可以传入两个参数；

sort(a,a+N) ,其中a是数组，a+N表示对a[0]至a[N-1]的N个数进行排序(默认从小到大排序)；

2、传入三个参数；

sort(a,a+N,cmp),第三个参数是一个函数 ；

如果让函数从大到小排序，可以用如下算法实现；

bool cmp(int a,int b){return a>b};

sort(A,A+N,cmp);

而stable\_sort的用法与sort一致，区别是stable\_sort函数遇到两个数相等时，不对其交换顺序；这个应用在数组里面不受影响，当函数参数传入的是结构体时，会发现两者之间的明显区别；

#include<stdio.h>

#include<string.h>

#include<iostream>

#include<algorithm>

using namespace std;

struct Q

{

char a[100],b[100];

}node[1000];

int cmp(const Q a,const Q b)

{

if(strcmp(a.a,b.a)<0)return 1;

else

{

if(strcmp(a.a,b.a)==0)

{

if(strcmp(a.b,b.b)>0)return 1;

return 0;

}

else

return 0;

}

}

int main()

{

int i;

for(i=1;i<=4;i++)

{

scanf("%s%s",node[i].a,node[i].b);

}

sort(node+1,node+1+4,cmp);

for(i=1;i<=4;i++)

printf("%s %s\n",node[i].a,node[i].b);

}

### 26. STL

STL大致可以分为三大类：算法(algorithm)、容器(container)、迭代器(iterator)。

STL容器是一些模板类，提供了多种组织数据的常用方法，例如vector(向量，类似于数组)、list(列表，类似于链表)、deque(双向队 列)、set(集合)、map(映象)、stack(栈)、 queue(队列)、priority\_queue(优先队列)等，通过模板的参数我们可以指定容器中的元素类型。

STL算法是一些模板函数，提供了相当多的有用算法和操作，从简单如for\_each（遍历）到复杂如stable\_sort（稳定排序）。

STL迭代器是对C中的指针的一般化，用来将算法和容器联系起来。几乎所有的STL算法都是通过迭代器来存取元素序列进行工作的，而STL中的每一个容器也都定义了其本身所专有的迭代器，用以存取容器中的元素。有趣的是，普通的指针也可以像迭代器一样工作。

熟悉了STL后，你会发现，很多功能只需要用短短的几行就可以实现了。通过STL，我们可以构造出优雅而且高效的代码，甚至比你自己手工实现的代码效果还要好。

STL的另外一个特点是，它是以源码方式免费提供的，程序员不仅可以自由地使用这些代码，也可以学习其源码，甚至按照自己的需要去修改它。

在ACM竞赛中，熟练掌握和运用STL对快速编写实现代码会有极大的帮助。

使用STL

在C++标准中，STL被组织为以下的一组头文件（注意，是没有.h后缀的！）：

algorithm / deque / functional / iterator / list / map

memory / numeric / queue / set / stack / utility / vector

当我们需要使用STL的某个功能时，需要嵌入相应的头文件。

#### 26.1． MAP

HDU 1004 统计一堆字符串中出现次数最多的字符串

#include<stdio.h>

#include<string>

#include<map>

using namespace std;

int main()

{

int i,n,m;

char s[20],a[20];

while(~scanf("%d",&n)&&n)

{

m=0;

map<string,int>mp;

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

{

scanf("%s",s);

mp[s]++;

if(mp[s]>m)

{

m=mp[s];

strcpy(a,s);

}

}

printf("%s\n",a);

}

return 0;

}

#include<iostream>

#include<string>

#include<map>

using namespace std;

map<string,int>map1;

int main()

{

int n,countt,i;

scanf("%d",&n);

char s[100][10];

countt=1;

map1.clear();

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

{

scanf("%s",s[i]);

if(map1[s[i]]==0)

{

map1[s[i]]=countt;

countt+=1;

}

}

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

{

printf("%s %d\n",s[i],map1[s[i]]);

}

}

#### 26.2. SET

//HDU1412 题意：输入2个集合，按大小顺序输出合并后的集合（注：集合元素的唯一性）

//set容器，集合排序的

#include<stdio.h>

#include<set>

using namespace std;

int main()

{

int n,m,i,x;

while(~scanf("%d%d",&n,&m))

{

set<int>s;

set<int>::iterator t1,t2;

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

{

scanf("%d",&x);

s.insert(x);

}

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

{

scanf("%d",&x);

s.insert(x);

}

t2=s.end();

t1=s.begin();

printf("%d",\*t1);

for(t1++;t1!=t2;t1++)

printf(" %d",\*t1);

printf("\n");

}

return 0;

}

### 27. LCA (HDU 2874)

#include<stdio.h>

#include<string.h>

#define MAXN 10005

#define MAXM 10005

struct node1 //链式前向星存图

{

int next;

int to;

int w;

}edge[MAXM<<1];

struct node2

{

int root;

int sw;

int deep;

}tree[MAXN];

int n,m,cnt,head[MAXN];

int visit[MAXN];

void add(int u,int v,int w)

{

cnt++;

edge[cnt].next=head[u];

edge[cnt].to=v;

edge[cnt].w=w;

head[u]=cnt;

}

void buildtree(int root,int sw,int deep) //用递归来建树；

{

int i,k;

for(i=head[root];i;i=edge[i].next)

{

k=edge[i].to;

if(visit[k])

continue;

visit[k]=1;

tree[k].root=root;

tree[k].deep=deep+1;

tree[k].sw=sw+edge[i].w;

buildtree(k,tree[k].sw,tree[k].deep);

}

}

int LCA(int a,int b) //并查集求最近父节点；

{

if(tree[a].deep>tree[b].deep)

while(tree[a].deep>tree[b].deep)

{

a=tree[a].root;

}

else if(tree[b].deep>tree[a].deep)

while(tree[b].deep>tree[a].deep)

{

b=tree[b].root;

}

while(a!=b)

{

a=tree[a].root;

b=tree[b].root;

}

return a;

}

int main()

{

int i,a,b,c,w,x,y,dis,s;

while(~scanf("%d%d%d",&n,&m,&c))

{

cnt=0;

memset(visit,0,sizeof(visit));

memset(head,0,sizeof(head));

memset(tree,0,sizeof(tree));

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

{

scanf("%d%d%d",&a,&b,&w);

add(a,b,w);

add(b,a,w);

}

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

if(visit[i]==0)

{

visit[i]=1;

buildtree(i,0,0);

}

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

{

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

s=LCA(x,y);

if(s==0)

printf("Not connected\n");

else

{

dis=tree[x].sw+tree[y].sw-2\*tree[s].sw;

printf("%d\n",dis);

}

}

}

return 0;

}

### 28. 01背包

首先01背包题目的雏形是

有N件物品和一个容量为V的背包。第i件物品的费用是c[i]，价值是w[i]。求解将哪些物品装入背包可使价值总和最大。

从这个题目中可以看出，01背包的特点就是：每种物品仅有一件，可以选择放或不放。

其状态转移方程是：

f[i][v]=max{f[i-1][v],f[i-1][v-c[i]]+w[i]}

对于这方方程其实并不难理解，方程之中，现在需要放置的是第i件物品，这件物品的体积是c[i],价值是w[i],因此f[i-1][v]代表的就是不将这件物品放入背包，而f[i-1][v-c[i]]+w[i]则是代表将第i件放入背包之后的总价值，比较两者的价值，得出最大的价值存入现在的背包之中。

理解了这个方程后，将方程代入实际题目的应用之中，可得

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

{

for(j = v; j>=c[i]; j--)//在这里，背包放入物品后，容量不断的减少，直到再也放不进了

{

f[i][v]=max(f[i-1][v],f[i-1][v-c[i]]+w[i]);

}

}

很经典的一道01背包题，要注意的是这里只要剩余的钱不低于5元，就可以购买任何一件物品，所以5在这道题中是很特许的，再使用01背包之前，我们首先要在现在所拥有的余额中保留5元，用这五元去购买最贵的物品，而剩下的钱就是背包的总容量，可以随意使用，因此可得代码

#include <stdio.h>

#include <algorithm>

using namespace std;

int cmp(int a,int b)

{

return a<b;

}

int main()

{

int n;

while(~scanf("%d",&n),n)

{

int i,price[2013]= {0},dp[2013] = {0};

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

scanf("%d",&price[i]);

sort(price+1,price+1+n,cmp);

int MAX=price[n];

int j,m;

scanf("%d",&m);

if(m<5)//低于5元不能购买

{

printf("%d\n",m);

continue;

}

m-=5;//取出5元用于购买最贵的物品

for(i = 1; i<n; i++)//01背包

{

for(j = m;j>=price[i];j--)

{

dp[j] = max(dp[j],dp[j-price[i]]+price[i]);

}

}

printf("%d\n",m+5-dp[m]-MAX);

}

return 0;

}

### 29. 状态压缩代码：

#include<stdio.h>

#include<string.h>

int DP[13][5000];

int main()

{

int n,m,i,j,cnt,a,k,count,map[15],s[5000];

while(~scanf("%d%d",&n,&m))

{

memset(DP,0,sizeof(DP));

memset(map,0,sizeof(map));

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

for(j=m-1;j>=0;j--)

{

scanf("%d",&a);

map[i]+=a\*(1<<j);

}

cnt=0;

//printf("%d\n",1<<m);

for(i=0;i<1<<m;i++)

{

//printf("%d\n",i&(i>>1));

if((i&(i>>1))==0)

s[cnt++]=i;

}

//printf("cnt=%d\n",cnt);

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

for(j=0;j<cnt;j++)

{

if(i==0)

{

if((s[j]|map[i])==map[i]) DP[i][j]=1;

}

else

{

if((s[j]|map[i])==map[i])

{

for(k=0;k<cnt;k++)

if(DP[i-1][k]&&((s[j]&s[k])==0))

{

DP[i][j]+=DP[i-1][k];

if(DP[i][j]>100000000)

DP[i][j]%=100000000;

}

}

}

}

/\*for(i=0;i<n;i++)

{

for(j=0;j<cnt;j++)

printf("%d ",DP[i][j]);

printf("\n");

}\*/

count=0;

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

{

//printf("%d ",DP[n-1][i]);

count+=DP[n-1][i];

if(count>100000000)

count%=100000000;

}

printf("%d\n",count);

}

return 0;

}

### 30. 快速幂

#include <stdio.h>

int main()

{

\_\_int64 T,a,b,c;

scanf("%d",&T);

while(T--)

{

scanf("%I64d%I64d%I64d",&a,&b,&c);

\_\_int64 s=1;

while(b)

{

if(b%2==1)

{

s=(s\*a)%c;

b--;

}

b/=2;

a=(a\*a)%c;

}

printf("%I64d\n",s);

}

return 0;

}

### 31.矩阵快速幂

#include<stdio.h>

#include<string.h>

struct Matrix

{

int a[15][15];

}per;

int m,n;

Matrix multi(Matrix a,Matrix b)

{

int i,j,k;

Matrix c;

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

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

{

c.a[i][j]=0;

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

c.a[i][j]+=(a.a[i][k]\*b.a[k][j])%9973;

c.a[i][j]=c.a[i][j]%9973;

}

return c;

}

//矩阵快速幂

Matrix pm(Matrix a)

{

Matrix s=per;//初始化

while(m)

{

if(m&1)

{

s=multi(s,a);

m--;

}

m>>=1;

a=multi(a,a);

}

return s;

}

int main()

{

int N,i,j;

Matrix q,p;

scanf("%d",&N);

while(N--)

{

\_\_int64 sum=0;

scanf("%d%d",&n,&m);

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

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

per.a[i][j]=(i==j);

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

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

scanf("%d",&q.a[i][j]);

p=pm(q);

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

{

sum+=p.a[i][i];

sum%=9973;

}

printf("%d\n",sum);

}

}

### 32.GCD & LCM

辗转相除法求最大公约数（LCM）和 最小公倍数（GCD）

int GCD(int m, int n)

{

int p;

if(n > m)

{

p = m;

m = n;

n = p;

}

while(m % n != 0)

{

p = m % n;

m = n;

n = p;

}

return n;

}

int LCM(int m, int n)

{

return n \* m / GCD(m, n);

}

最简ＧＣＤ

int gcd(int m,　int n)

{　return 　n　?　gcd(n,　m%n)　:　m;　}

### 33.ACM小技巧：

1.一般用C语言节约空间，要用C++库函数或STL时才用C++;  
cout、cin和printf、scanf最好不要混用。

大数据输入输出时最好不要用cin、cout，防止超时。

2.有时候int型不够用，可以用long long或\_\_int64型(两个下划线\_\_)。

值类型表示值介于 -2^63 ( -9,223,372,036,854,775,808) 到2^63-1(+9,223,372,036,854,775,807 )之间的整数。

printf("%I64d",a); //\_\_int64 一般VC编译器使用

printf("%lld",a); //long long 一般g++编译器使用

3.OJ判断是只看输出结果的。

所以大部分题处理一组数据后可以直接输出，就不需要用数组保存每一个Case的数据。

while(case--)

{scanf(...);

......

printf(...);

}

4.纯字符串用puts()输出。

数据大时最好用scanf()、printf()减少时间。

先用scanf()，再用gets()会读入回车。所以在中间加一个getchar();

scanf("%c%c",&c1,&c2)会读入空格；建议用%s读取字符串，取第一个字符。

5.读到文件的结尾，程序自动结束

while( ( scanf(“%d”,&a) ) != -1 )

while( ( scanf(“%d”,&a) ) != EOF)

while( ( scanf(“%d”,&a) ) == 1 )

while( ~( scanf(“%d”,&a) )  )

读到一个0时，程序结束

while( scanf(“%d”,&a) ,a)

读到多个0时，程序结束

while( scanf(“%d%d%d”,&a,&b,&c),a+b+c ) //a,b,c非负

while( scanf(“%d%d%d”,&a,&b,&c),a|b|c )

6.数组定义int a[10]={0};可以对其全部元素赋值为0；

数组太大不要这样，防止CE。

全局变量，静态变量自动初始化为0；

7.有很多数学题是有规律的，直接推公式或用递归、循环。

8.圆周率=acos(-1.0)  
自然对数=exp(1.0)

9.如果要乘或除2^n,用位移运算速度快。a>>n;a<<n;

10.定义数组时，数组大小最好比告诉的最大范围大一点。

字符数组大小必须比字符串最大长度大1。

处理字符数组时不要忘了在最后加'/0'或者0。

 11.擅用三目运算符

int max(int a,int b)

{return a>b?a:b;

}

int gcd(int m,int n)

{return n?gcd(n,m%n):m;

}

 int abs(int a)

{return a<0?-a:a;

}

12. 将乘法转换成加法减少时间 log(a\*b)=log(a)+log(b)

将乘法转换成除法防止溢出 a/(b\*c)=a/b/c

 13.排序要求不高时可以用C++的STL模板函数sort(),stable\_sort()

int a[n]={...};

sort(a,a+n);

bool cmp(int m,int n)

{return m>n;

}

sort(a,a+n,cmp);

 14.有的题数据范围小但是计算量大可以用打表法，

先把结果算出来保存在数组里，要用时直接取出来。

 15.浮点数比较时最好控制精度

#define eps 1e-6

fabs(a-b)<eps

16.有些字符串与整型的转换函数是非标准的

可以使用sscanf()和sprintf()代替 sscanf(s,"%d",&n);//从字符串s中读入整数n

sprintf(s,"%d",n);//将n转换为字符串s

### 34. /\*\* 大数（高精度）求幂 \*\*/

//输入两个数字 m（可以带小数点），n，求m的n次方

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char multiply[1000];

void multiplyer(char \*a,char \*b) //实质上就是俩个整型大数的乘法

{

int i,j,num,tem;

int multy[1000]= {0};

for(i=strlen(a)-1; i>=0; i--)

for(j=strlen(b)-1,tem=strlen(a)-1-i; j>=0; j--,tem++)

{

num=(a[i]-'0'+0)\*(b[j]-'0'+0)+multy[tem];

multy[tem]=num%10;

multy[tem+1]+=num/10;

}

for(i=999; i>=0; i--)

if(multy[i])

break;

for(j=0; i>=0; i--)

multiply[j++]=multy[i]+'0';

}

int main()

{

char a[10];

int b,i,fcount,num,flag;

while(scanf("%s %d",a,&b)!=EOF)

{

memset(multiply,0,sizeof(multiply));

for(i=0,num=0; i<=strlen(a)-1; i++)

if(a[i]=='.') //需记录浮点数所在的位数

{

fcount=strlen(a)-1-i;

for(; i<=strlen(a)-1; i++)

a[i]=a[i+1];

num=fcount\*b; //记录小数点后数字的位数

break;

}

for(i=0; i<=strlen(a)-1; i++)

if(a[i]!='0')

break;

for(fcount=0; i<=strlen(a); i++)

a[fcount++]=a[i];

strcpy(multiply,a);

b=b-1;

while(b--)

multiplyer(multiply,a);

if(num<=strlen(multiply)) //根据补小数点的位置，需分情况讨论

{

for(i=strlen(multiply)-1; i>=0; i--)

if(multiply[i]!='0')

{

flag=i;

break;

}

for(i=0; i<=strlen(multiply)-1; i++)

{

printf("%c",multiply[i]);

if(i==strlen(multiply)-num-1) //浮点数求幂在相应位置上补小数点

{

if(i==flag) //判断若是整数的幂次方，则不需要输出小数点而直接跳出

break;

printf(".");

}

if(flag==i)

break;

}

}

else //若num>multily的位数，需要在小数点后补足相应位数的零

{

for(i=strlen(multiply)-1; i>=0; i--)

if(multiply[i]!='0')

{

multiply[i+1]='\0';

break;

}

printf(".");

for(i=strlen(multiply); i<=num-1; i++)

printf("0");

printf("%s",multiply);

} puts(""); } return 0;}

### 35. /\*\* 大数除法与求余 \*\*/

//这个是uva上的一个题，本身写的那个比较繁琐，借鉴了一些其他人的方法 ...

/\*\*\*

这个题大意为输入两个数和一个符号（ '/' or ''% ），求除法或是求余（其中除数是n满足 0<n<231），例：

输入为：

110 / 100

99 % 10

输出为：

1

9

\*\*\*/

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

char s[1000],result[1000];

int main()

{

long long mod,divis;

int n,i,k,flag,len;

char c;

while(cin>>s>>c>>n) //输入被除数 s，符号 c （ '/' or ''% ），以及除数n

{

len=strlen(s);

if(c=='%')

{

mod=0;

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

{

mod=mod\*10+s[i]-'0';

mod=mod%n; //利用除法性质，其实质是每次都是一个最多比n多一位的mod对n进行求余

}

cout<<mod<<endl;

}

else

{

divis=flag=0;

for(i=k=0; i<len; i++)

{

divis=divis\*10+s[i]-'0';

if(divis>=n&&!flag) //利用除法性质，当divs大于除数n时，开始进行整除

{

result[k++]=divis/n+'0';

divis=divis%n; //除法性质，余数\*10加下一位的数字便是新的被除数

flag=1;

}

else if(flag)

{

result[k++]=divis/n+'0';

divis=divis%n;

}

}

if(!k) result[k++]='0';

result[k]='\0';

cout<<result<<endl;

}

}

return 0;

}

### 36. /\*\* 大数阶乘 \*\*/

// 最大可以计算到10000！

#include <stdio.h>

#include <string.h>

#define N 36000 //10000的阶乘大约要36000位

int f[N];

int main()

{

int i,j,n;

int c,s;

while(scanf("%d",&n)!=EOF)

{

memset(f,0,sizeof(f));

f[0]=1;

for(i=2; i<=n; i++)

{

c=0;

for (j=0; j<N; j++) // 将每位阶乘数导入数组

{

s=f[j]\*i+c; // s为f[j]\*i加上上一位进位的数字

f[j]=s%10; // s%10为本位数字

c=s/10; // s/10为进位的数字

}

}

for (j=N-1; j>=0; j--)

if (f[j]) break;

for (i=j; i>=0; i--)

printf("%d",f[i]);

printf("\n");

}

return 0;

}

### 37. /\*\* 大数乘法 \*\*/

#include<stdio.h>

#include<string.h>

#define N 500

int multy[1000]; //一千位以下的大数

void multiplyer(char \*a,char \*b)

{

int i,j,num,tem,h;

char k;

memset(multy,0,sizeof(multy));

for(i=strlen(a)-1; i>=0; i--)

for(j=strlen(b)-1,tem=strlen(a)-1-i; j>=0; j--,tem++)

{

num=(a[i]-'0')\*(b[j]-'0')+multy[tem]; // 转化成两个数字相乘

multy[tem]=num%10; // num%10得到的数为muity本位的数字

multy[tem+1]+=num/10; // num/10得到的是向前进位的数字

}

for(i=999; i>=0; i--) //从multy数组999位开始遍历，找到第一个不为0的数字，即：两数相乘得到的数的的最高为数字

if(multy[i])

break;

if(i==-1) //注意，当输入的数字为0时，i 为-1，但数字与零相乘需输出 0，因此此处需特殊处理

i=0;

for(; i>=0; i--)

printf("%d",multy[i]);

puts("");

}

int main()

{

char a[N+1],b[N+1];

while(scanf("%s%s",a,b)==2)

multiplyer(a,b);

return 0;

}

### 38. /\*\* 大数累加 \*\*/

#include<stdio.h>

#include<string.h>

#define N 1000

int sum[N+1]= {0}; // 对于大数的累加来说，开一个int型的数组存数字比开字符型的数组操作起来要更加简单易用

void SumAdd(char \*a)

{

int i,j;

for(i=N,j=strlen(a)-1; j>=0; j--,i--)

sum[i]+=a[j]-'0'; //将a[j]转化成数字加到sun[j]的相应位上

for(i=N; i>=0; i--)

if(sum[i]>9)

{

sum[i]-=10;

sum[i-1]+=1; //进位

}

}

int main()

{

char num[1000];

int i;

while(scanf("%s",num),strcmp(num,"0")) //不断输入数字，直到输入0时结束。

SumAdd(num);

for(i=0; i<=N; i++) //从sum[0]遍历到第一个不为0的数字，即：sum的最高位数字

if(sum[i]) break;

for(; i<=N; i++)

printf("%d",sum[i]); //从最高位开始输出一直到个位

puts("");

return 0;

}

39.