

# C++程序设计实践

信息学院 | 在线编程实践

By Andy & lifeboat

2021年7月



# 数组模拟队列

# 用数组模拟队列

<http://www.xmuoj.com/problem/ACW829>

实现一个队列，队列初始为空，支持四种操作：

- (1) “push x” – 向队尾插入一个数x；
- (2) “pop” – 从队头弹出一个数；
- (3) “empty” – 判断队列是否为空；
- (4) “query” – 查询队头元素。

现在要对队列进行M个操作，其中的每个操作3和操作4都要输出相应的结果。

# 接口说明

## 输入样例 1

```
10
push 6
empty
query
pop
empty
push 3
push 4
pop
query
push 6
```

### 输入

第一行包含整数M，表示操作次数。

接下来M行，每行包含一个操作命令，操作命令为"push x"，"pop"，"empty"，"query"中的一种。

### 输出

对于每个"empty"和"query"操作都要输出一个查询结果，每个结果占一行。

其中，"empty"操作的查询结果为"YES"或"NO"，"query"操作的查询结果为一个整数，表示队头元素的值。

## 输出样例 1

```
NO
6
YES
4
```

# AcWing风格代码

// 在队尾插入元素，在队头弹出元素

```
int q[N], hh, tt = -1;
```

// 插入

```
q[ ++ tt] = x;
```

// 弹出

```
hh ++ ;
```

// 判断队列是否为空

```
if (hh <= tt) not empty
```

```
else empty
```

// 取出队头元素

```
q[hh]
```

# 参考代码

```
#include <iostream>
using namespace std;
const int N = 100010;
int q[N], qHead, qTail = -1; //qHead指向队首, qTail指向队尾

int main()
{
    //!cin\cout提速
    ios::sync_with_stdio(false); //来打消iostream的输入输出缓存
    cin.tie(0); //cin.tie(0)来解除cin与cout的绑定,0表示NULL
    //!根据题目输入输出
```

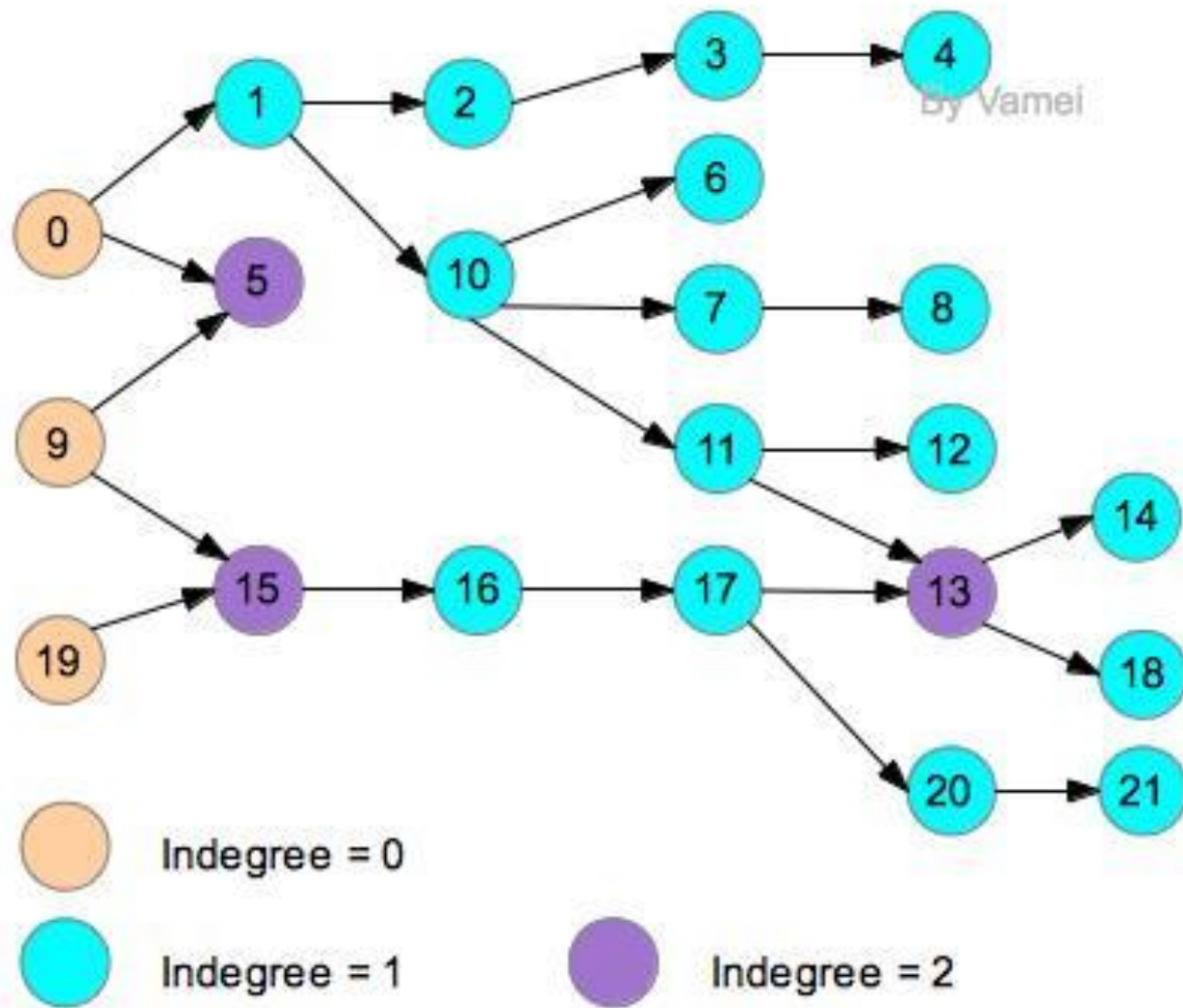
```
int m;  
cin >> m;  
while (m--)  
{  
    string op;  
    int x;  
    cin >> op;  
    if (op == "push") {  
        cin >> x;  
        q[++qTail] = x; //入队,从尾部入队  
    }  
    else if (op == "pop"){  
        qHead++; //出队  
    }  
    else if (op == "query"){  
        cout << q[qHead] << endl; //输出队首元素  
    }  
    else{  
        if (qHead <= qTail)  
            cout << "NO" << endl; //队判空  
        else  
            cout << "YES" << endl;  
    }  
}  
return 0;
```



# 拓扑排序



# 拓扑排序 (入度、出度) 有向无环图



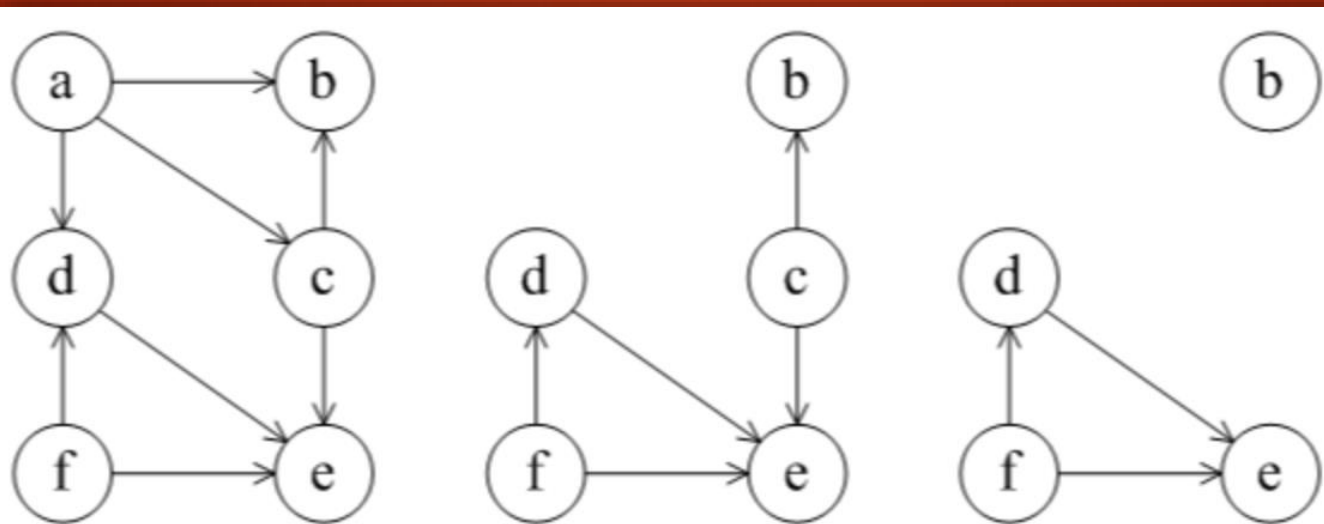
# 拓扑排序 (Topological Sorting)

- ▶ 图论中，拓扑排序 (Topological Sorting) 是一个有向无环图 (**DAG, Directed Acyclic Graph**) 的所有顶点的线性序列。且该序列必须满足下面两个条件：
  - ▶ 每个顶点出现且只出现一次。
  - ▶ 若存在一条从顶点 A 到顶点 B 的路径，那么在序列中顶点 A 出现在顶点 B 的前面。
- ▶ **注意：**有向无环图 (DAG) 才有拓扑排序。拓扑排序有一个或多个结果。

# 拓扑排序的过程

- ▶ 从 DAG 图中选择一个 没有前驱（即入度为0）的顶点并输出。
- ▶ 从图中删除该顶点和所有以它为起点的有向边。
- ▶ 重复 1 和 2 直到当前的 DAG 图为空或当前图中不存在无前驱的顶点为止。后一种情况说明有向图中必然存在环。

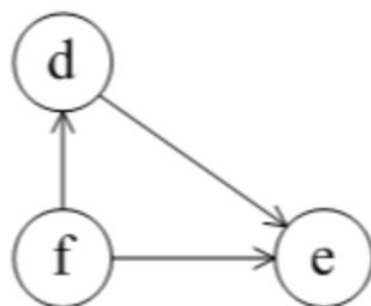
# 拓扑排序图示



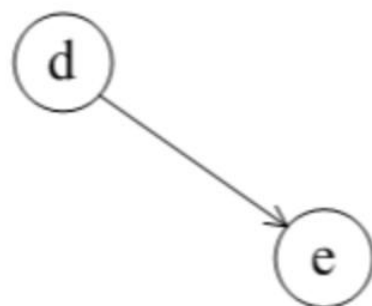
(a)图

(b)输出a

(c)输出c



(d)输出b



(e)输出f



(f)输出d (g)输出e

拓扑排序序列: acbfde



あなた たいま つるぎ えら ほど けんし せいちょう  
そして貴方は 退魔の剣に選ばれる程の剣士に成長した  
就这样 成长为被传说中的“退魔之剑”选中的人

## 例题1 突袭路线

# 突袭路线

为了解救公主，林克必须深入敌后。

在备战前，他拿出“关系分析仪”扫描敌营中每个士兵之间的关系。

关系分析仪的功能说明如下：

如果A的活动范围在B的眼皮底下，那么分析仪就会从B出发连一条射线指向A（ $B \rightarrow A$ ）。

经过扫描，林克得到全营敌兵的相互关系。有些敌人被多个同伴看顾，有些敌人背后一个替他守望的都没有。

林克决定从背后没有人的敌人开始，潜伏到其背后，突袭之，并且避免被其他人发现。

军营一共有 $n$ 个敌人，彼此之间的关系有 $m$ 条射线，请找到一条可以逐个击破敌人的路线图。

如果找不到这样一条突袭路线，请则输出-1.



# 建模

问题转化为：给定一个 $n$ 个点 $m$ 条边的有向图，点的编号是1到 $n$ ，图中可能存在重边和自环。

请输出任意一个该有向图的拓扑序列，如果拓扑序列不存在，则输出-1。

若一个由图中所有点构成的序列 $A$ 满足：对于图中的每条边 $(x, y)$ ， $x$ 在 $A$ 中都出现在 $y$ 之前，则称 $A$ 是该图的一个拓扑序列。

数据范围： $1 \leq n, m \leq 10^5$

## 输入

第一行包含两个整数 $n$ 和 $m$

接下来 $m$ 行，每行包含两个整数 $x$ 和 $y$ ，表示存在一条从点 $x$ 到点 $y$ 的有向边 $(x, y)$ 。

## 输出

共一行，如果存在拓扑序列，则输出拓扑序列。

否则输出-1。



# 算法:拓扑排序

- ▶ 用邻接表建图(数组模拟链表)
- ▶ 在BFS的框架上实现TopSort
- ▶ 用数组模拟队列



# 参考代码

```
//code by Andy
#include <iostream>
#include <algorithm>
#include <queue>
#include <cstring>

using namespace std;
#define For(a, begin, end) for (int a = (begin); a < (end); a++)
const int N = 10000007; //题目n的范围
```

# 邻接表,队列(都用数组模拟)

```
int n, m;
int head[N], edge[N], nextVertex[N], idx; //邻接表
int q[N], d[N]; //队列, 入度

void add(int a, int b)
{
    edge[idx] = b;
    nextVertex[idx] = head[a];
    head[a] = idx++;
}
```

# 拓扑排序

```
bool topsort()
```

```
{
```

```
    int qHeadIdx = 0, qTailIdx = -1; //队头, 队尾
```

```
    //从前往后遍历入度为0的点, 插入队列
```

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

```
        if (!d[i])
```

```
            q[++qTailIdx] = i; //数组模拟队列, 入队是尾指针+1
```

# 拓扑排序

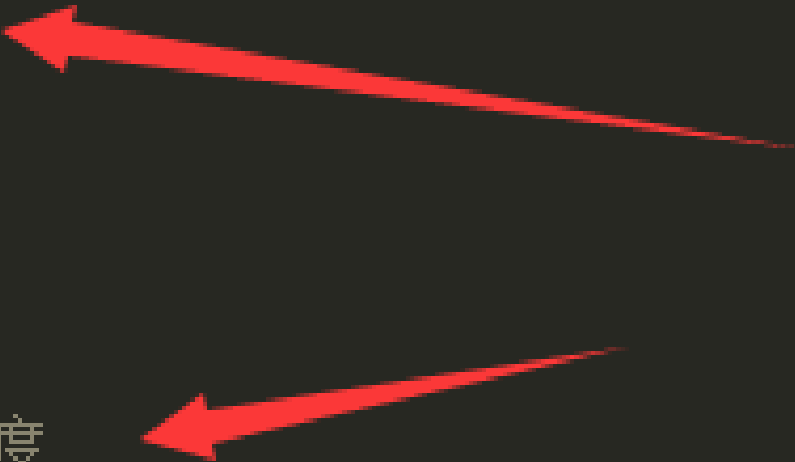
```
while (qHeadIdx <= qTailIdx) //如果头指针小于尾指针
{
    int t = q[qHeadIdx++]; //取队列头元素，，出队只是把头指针往后移动一位
    //遍历t的临边，空指针初始化为-1
    for (int i = head[t]; i != -1; i = nextVertex[i])
    {
        int j = edge[i]; //取到出边
        d[j]--; //入度减一
        if (d[j] == 0)
            q[++qTailIdx] = j; //如若入度为0，压入队列
    }
}
//判断是否所有顶点都入队
//也就是头指针qTailIdx是否等于n-1,即所有点都进入队列了
return qTailIdx==n-1;
```

```
}
```

# 建图和初始化

```
int main()
{
    cin >> n >> m;           //读入n,m
    memset(head, -1, sizeof(head)); //初始化Head数组指向-1顶点

    For(i, 0, m){
        int a, b;
        cin >> a >> b;
        add(a, b); //插入边
        d[b]++;    //更新入度
    }
```



# 顺序输出队列中的元素就是拓扑序列

```
if (topsort())  
{//数组队列里面的元素就是拓扑序  
    for (int i = 0; i < n; i++) printf("%d ", q[i]);  
}  
else  
    puts("-1");  
return 0;  
}
```



## 例题2 全面反击

# 可达性统计

加农复活后，他邪恶的势力掌控了海拉鲁大陆几乎所有的神庙。

为了避免路上战斗的时间，决定绕开路上的敌人，全力攻取神庙。

他打开地图，研究进攻路线。

若神庙A到神庙B有一条可以绕开敌人的路，林克就把路线标在地图上。

经过N个小时的努力，林克完成这宏伟的绘图工作。

他得到一张N个点M条边的有向无环图，每个点代表一个神庙的位置。

请分别统计从每个点出发能够到达的点的数量。

数据范围

$1 \leq N, M \leq 30000$





# 可达性统计

## 输入

第一行两个整数 $N, M$ ，接下来 $M$ 行每行两个整数 $x, y$ ，表示从 $x$ 到 $y$ 的一条有向边。

## 输出

输出共 $N$ 行，表示每个点能够到达的点的数量。

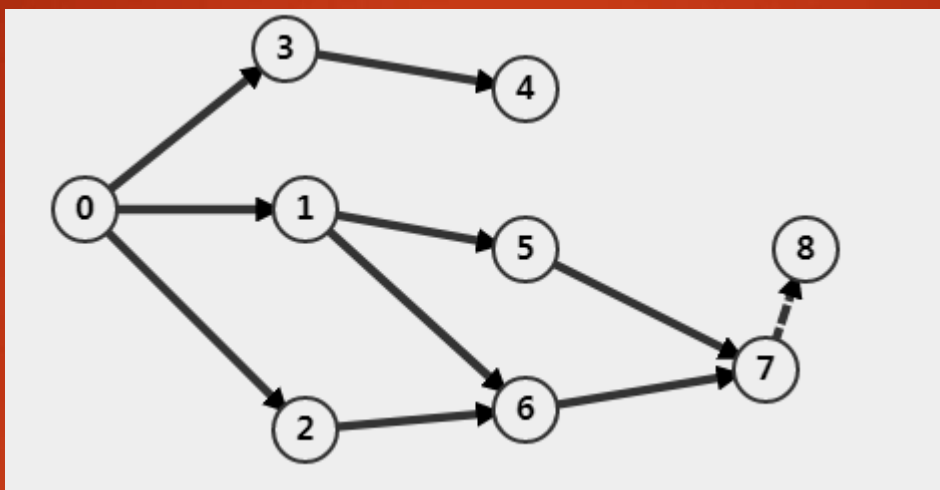
### 输入样例 1

```
10 10
3 8
2 3
2 5
5 9
5 9
2 3
3 9
4 8
2 10
4 9
```

### 输出样例 1

```
1
6
3
3
2
1
1
1
1
1
1
```

# $f(i)$ 定义为所有能从 $i$ 出发抵达的点的集合



- ▶ 如上图有  $f(0) = f(3) + f(1) + f(2)$
- ▶ 数据范围  $N, M \leq 300000$ , 最大的情况是 9000000000000  
因此使用 bitset 来标记顶点之间是否有边
- ▶ 算法: 1. 拓扑排序 2. 按照逆序的方式遍历拓扑顶点并统计各顶点的边数

# 参考代码

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <bitset> //todo 使用bit
#include <queue>
using namespace std;
#define MAXM 30007
int n, m, degree[MAXM], a[MAXM]; //!存储入度、拓扑序的顶点
int edge[MAXM], Next[MAXM], head[MAXM], tot, cnt;
bitset<MAXM> f[MAXM]; //todo 使用bit

void add(int x, int y)
{ // !在邻接表中添加一条有向边
    edge[++tot] = y, Next[tot] = head[x], head[x] = tot;
    degree[y]++; //入度++
}
```

```
void topsort()
{
    queue<int> q; //队列
    for (int i = 1; i <= n; i++)
        if (degree[i] == 0) //?所有入度为0的顶点入栈
            q.push(i);
    //?栈不空的时候拓扑排序
    while (q.size())
    {
        int x = q.front();
        q.pop();
        a[++cnt] = x; //!保存拓扑过程中的顶点
        //todo 拓展顶点x
        for (int i = head[x]; i; i = Next[i])
        {
            int y = edge[i];
            --degree[y]; //入度减1
            if (degree[y] == 0)
                q.push(y);
        }
    } // end of while
} // end of topsort
```

# 可达性统计

```
void calReachableNodes()
{
    //!逆序统计
    for (int i = cnt; i; i--)
    {
        int x = a[i]; //取拓扑排序过程中的顶点
        f[x][x] = 1;  //!当前顶点设置为访问过
        for (int i = head[x]; i; i = Next[i])
        {
            int y = edge[i];
            f[x] |= f[y]; //!求并集合
        }
    }
}
```

# 建图

```
int main()
{
    cin >> n >> m; // 点数、边数
    for (int i = 1; i <= m; i++)
    {
        int x, y;
        cin >> x >> y;
        add(x, y); //?建图
    }
    topsort();          //?拓扑排序，顶点存储在全局数组a中
    calReachableNodes(); //!逆序计算可达顶点数
    for (int i = 1; i <= n; i++)
        cout << f[i].count() << endl;
}
```

# AC!!



已接受

时间: 116ms 内存: 111MB 语言: C++ 用户: andy

ID	状态	内存	时间	分数	实际时间	信号
1	已接受	3MB	2ms	20	2ms	0
2	已接受	4MB	3ms	20	5ms	0
3	已接受	7MB	19ms	20	26ms	0
4	已接受	39MB	80ms	20	127ms	0
5	已接受	111MB	116ms	20	222ms	0



## Boss02 Daruk的实力



# Daruk的实力

英杰Daruk是Gorons族力气最大的人。他能够轻松粉碎巨大的岩石。



Daruk的攻击力上限是 $W$ ，他可以一次性粉碎重量之和不超过 $W$ 的任意多个岩石。

假设有 $N$ 个重量不同的岩石，其中第 $i$ 个岩石的重量是 $G[i]$ 。

请问，Daruk一次攻击最多能粉碎的最大重量的岩石是多少？

# 数据太大,无法暴力枚举!

## 数据范围

$$1 \leq N \leq 46,$$

$$1 \leq W, G[i] \leq 2^{31} - 1$$

## 输入

第一行两个整数，分别代表W和N。

以后N行，每行一个正整数表示G[i]。

## 输出

一个整数，表示Daruk一次攻击能粉碎的最大岩石重量。

# 双层深度优先搜索算法:

- ▶ 将所有岩石按重量从大到小排序
- ▶ 先将前K块岩石能凑出的所有重量打表,然后排序判重
- ▶ 搜索剩下的N-K块岩石的选择方式中,能够使总和不超W的选择方法(使用二分法)
- ▶ 时空优化:  $N \leq 46$ , 并且前K块打表的时间复杂度较低,后面排序较高,为了是计算的时间前后平衡,K不选23乃是选 $223+3$ 则后面计算变为 $N-K$ 为 $23-3=20$ .

# 参考代码

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
```

```
//?N个重量不同的岩石，其中第i个岩石的重量是g[i]
int n, half, m, g[50];
unsigned int w, ans, a[(1 << 24) + 1];
```

# 预处理

```
int main() {  
  
    //读入别代表w和n  
    cin >> w >> n;  
    //N行，每行一个正整数表示G[i]  
    for (int i = 1; i <= n; i++) cin>>g[i];  
    //?排序  
    sort(g + 1, g + n + 1);  
    //?逆序，从大到小排序，g[0]是最大的岩石重量  
    reverse(g + 1, g + n + 1);  
  
    //?运算复杂度的平衡优化  
    half = n / 2 + 3; //todo 前半多取3
```

# 双层深搜

```
//?运算复杂度的平衡优化
half = n / 2 + 3; //todo 前半多取3

//todo 前半程深度优先搜索
dfs1(1, 0); //从第1层，累计重量为0开始深搜
sort(a + 1, a + m + 1);
//?去重
m = unique(a + 1, a + m + 1) - (a + 1);

//todo 后半程深搜
dfs2(half, 0);
//输出结果
cout << ans << endl;
```

```
}
```

# 前半程深搜

//todo 深搜打表

```
void dfs1(int i, unsigned int sum) {  
    if (i == half) {///  
        a[++m] = sum;  
        return;  
    }  
    dfs1(i + 1, sum);///  
    if (sum + g[i] <= w) dfs1(i + 1, sum + g[i]);///  
}
```

# 后半程深搜

//todo 后半程深搜

```
void dfs2(int i, unsigned int sum) {  
    if (i == n + 1) {  
        calc(sum); //查表、二分搜索最大的重量  
        return;  
    }  
    dfs2(i + 1, sum); //?后半程的深搜  
    if (sum + g[i] <= w) dfs2(i + 1, sum + g[i]);  
}
```



# 二分查找最大重量

//todo 二分查找最大的重量（查表法）

```
void calc(unsigned int val) {  
    int rest = w - val;  
    int l = 1, r = m;  
    while (l < r) {  
        int mid = (l + r + 1) / 2;  
        if (a[mid] <= rest) l = mid; else r = mid - 1;  
    }  
    ans = max(ans, a[l] + val);  
}
```

# AC!!!



已接受

时间: 534ms 内存: 17MB 语言: C++ 用户: andy

ID	状态	内存	时间	分数	实际时间	信号
1	已接受	3MB	0ms	10	2ms	0
2	已接受	3MB	9ms	10	11ms	0
3	已接受	7MB	113ms	10	118ms	0
4	已接受	3MB	25ms	10	50ms	0
5	已接受	3MB	64ms	10	82ms	0
6	已接受	3MB	65ms	10	104ms	0
7	已接受	16MB	389ms	10	406ms	0
8	已接受	17MB	534ms	10	554ms	0
9	已接受	3MB	88ms	10	89ms	0
10	已接受	3MB	1ms	10	2ms	0



## Boss03 古希卡文的九宫之谜(2)

# 题目分析

给定任意一个 $3 \times 3$ 的矩阵,如何把该矩阵变换为标准矩阵?



初始矩阵

标准矩阵

变换规则定义：把



与其上、下、左、右四个方向之一的数字交换（如果存在）。

# 题目分析

上述题目拥有如下的变换过程：



现在，给你一个初始网格，请你求出得到正确排列至少需要进行多少次交换。

为了区分，我们把特殊字符0,标记为x.

# 建模与数据输入输出分析

题面大意翻译如下：

```
1 2 3
x 4 6
7 5 8
```

使得网格变为如下排列（标准矩阵）：

```
1 2 3
4 5 6
7 8 x
```

交换过程如下：

1 2 3	1 2 3	1 2 3	1 2 3
x 4 6	4 x 6	4 5 6	4 5 6
7 5 8	7 5 8	7 x 8	7 8 x

现在，给你一个初始网格，请你求出得到正确排列至少需要进行多少次交换。

## 输入

可以把 $3 \times 3$ 的矩阵转换为String表示：

```
1 2 3
x 4 6
7 5 8
```

输入矩阵可以表示为：1 2 3 x 4 6 7 5 8

标准矩阵可以表示为：8 7 6 5 4 3 2 1 x

因此，输入1行字符串，每行是由空格隔开的9个字符，代表 $3 \times 3$ 的矩阵。

## 输出

输出占一行，包含一个整数，表示最少交换次数。

如果不存在解决方案，则输出"impossible"。

# 输入输出分析

输入样例 1 

2 3 4 1 5 x 7 6 8

输出样例 1

19

输入样例 2 

4 x 6 1 3 7 2 8 5

输出样例 2

impossible

- ▶ 求最少步数,可以用bfs搜索
- ▶ 如何开辟一个数组使得这个数组中的元素, 能够和矩阵的所有状态 (长度为9的字符串的全排列) 一一对应

# 全排列哈希

<https://www.acwing.com/solution/acwing/content/2481/>

- 我们熟知的数一般都是常进制数，所谓常进制数就是该数的每一位都是常数进制的

$k$ 进制数上的每一位都逢 $k$ 进一，第 $i$ 位的位权是 $k^i$

- 这里要介绍一种变进制数，用来表示字符串的排列状态

这种数的第 $i$ 位逢 $i$ 进一，第 $i$ 位的位权是 $i!$

用 $d[i]$ 来表示一个变进制数第 $i$ 位上的数字

一个 $n$ 位变进制数的值就为 $\sum_{i=0}^{n-1} d[i] \times i!$



# 全排列哈希

这是一个最大的9位变进制数

876543210

它对应的十进制数为

$$8 \times 8! + 7 \times 7! + 6 \times 6! + \dots + 1 \times 1! + 0 \times 0! = 9! - 1 = 362879$$

我们可以找到一个9位变进制数，与一个9位无重复串的某种排列一一对应

# 全排列哈希

用  $d[i]$  表示字符串中的第  $i$  位与其前面的字符组成的逆序对个数

字符串的一种排列对应的变进制数的值为  $\sum_{i=0}^{n-1} d[i] \times i!$

这是字符串 `123x46758` 的与  $d[i]$  的对应关系

$i$	0	1	2	3	4	5	6	7	8
$s[i]$	1	2	3	x	4	6	7	5	8
$d[i]$	0	0	0	0	1	1	1	3	1

它对应的变进制数的值为

$$1 \times 4! + 1 \times 5! + 1 \times 6! + 3 \times 7! + 1 \times 8! = 56304$$

# 用以下函数求字符串的一种排列对应的哈希值

```
int permutation_hash(char s[], int n)    //求长度为n的字符串某种排列的哈希值
{
    int ans = 0;
    for(int i = 0; i < n; i++)
    {
        int d = 0;
        for(int j = 0; j < i; j++)
            if(s[j] > s[i]) d++;          //求s[i]与其前面的字符组成的逆序对个数
        ans += d * fact[i];
    }
    return ans;
}
```

# 全排列哈希+BFS 参考代码

//todo 使用变进制数设计hash映射

```
int fact[9];
```

```
bool visited[362880];
```

//todo 传入排列的字符串，传出对应的全排列id值

```
int permutation_hash(char s[]) //求长度为9的字符串某种排列的哈希值
```

```
{
```

```
    int ans = 0;
```

```
    for(int i = 0; i < 9; i ++)
```

```
    {
```

```
        int d = 0;
```

```
        for(int j = 0; j < i; j ++)
```

```
            if(s[j] > s[i]) d ++;
```

//求s[i]与其前面的字符组成的逆序对个数

```
        ans += d * fact[i];
```

```
    }
```

```
    return ans;
```

```
}
```

```
#include<cstring>
```

```
#include<iostream>
```

```
#include<queue>
```

```
using namespace std;
```

# 数据结构

```
//todo Point结构, 存储的每个序列
typedef struct{
    char s[10];
    int step;
    int k;           //'x'在第k位
}Point;

//?四个方向搜索
int dx[4] = {-1, 0, 1, 0};
int dy[4] = { 0, -1, 0, 1};
```

# 预处理,找到start点

```
int main()
{
    fact[0] = 1;
    for (int i = 1; i < 9; i++)
        fact[i] = fact[i - 1] * i; //预处理fact[i] = i!

    char c[2], str[10];
    Point start;
    for (int i = 0; i < 9; i++)
    {
        scanf("%s", &c);
        if (c[0] == 'x')
            start.k = i;
        start.s[i] = c[0];
    }
    start.s[9] = 0;
    start.step = 0;
```

# 广度优先搜索BFS

```
start.s[9] = 0;  
start.step = 0;  
int result = bfs(start);  
if (result == -1)  
    cout << "impossible" << endl;  
else  
    cout << result << endl;  
return 0;  
}
```

# 从p点开始搜索

```
int bfs(Point p)
{
    visited[permutation_hash(p.s)] = true;
    queue<Point> q;
    q.push(p);
    while (!q.empty())
    {
        p = q.front();
        q.pop();

        if (!strcmp(p.s, "12345678x"))
            return p.step;
    }
}
```



# 坐标变换

```
int x = p.k / 3; // 'x' 的行数
int y = p.k % 3; // 'x' 的列数
Point next;
next.step = p.step + 1;
for (int i = 0; i < 4; i++)
{
    int nx = x + dx[i];
    int ny = y + dy[i];
```

# BFS,采用全排列hash

```
if (nx >= 0 && nx <= 2 && ny >= 0 && ny <= 2)
{
    next.k = nx * 3 + ny; //求出'x'在字符串中的的新位置

    strcpy(next.s, p.s);
    next.s[9] = 0;
    next.s[p.k] = p.s[next.k]; //先用即将和'x'交换的字符覆盖'x'之前的位置
    next.s[next.k] = 'x';      //再给'x'的新位置赋值'x'
    //todo 得到next.s的这个字符串的hash的值，访问vis数组判断其是否已经被访问过
    int hash = permutation_hash(next.s);
    if (!visited[hash])
    {
        visited[hash] = true;
        q.push(next);
    }
}
}
return -1;
}
```

# AC!!!!



已接受

时间: 46ms 内存: 4MB 语言: C++ 用户: andy

ID	状态	内存	时间	分数	实际时间	信号
1	已接受	4MB	35ms	10	36ms	0
2	已接受	4MB	43ms	10	63ms	0
3	已接受	4MB	25ms	10	27ms	0
4	已接受	4MB	46ms	10	75ms	0
5	已接受	4MB	41ms	10	61ms	0
6	已接受	4MB	32ms	10	52ms	0
7	已接受	4MB	46ms	10	47ms	0
8	已接受	4MB	26ms	10	33ms	0
9	已接受	4MB	6ms	10	7ms	0
10	已接受	4MB	38ms	10	39ms	0