实验舱
青少年编程
走近科学 走进名校

蛟龙四班

队列与宽度优先搜索

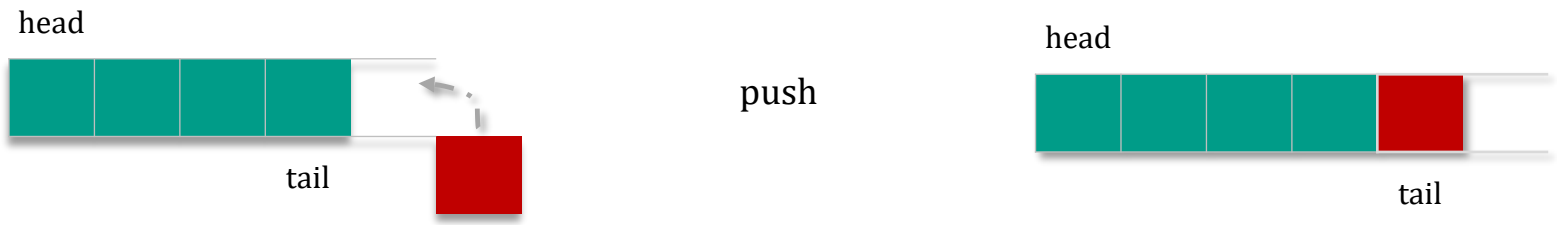
Mas

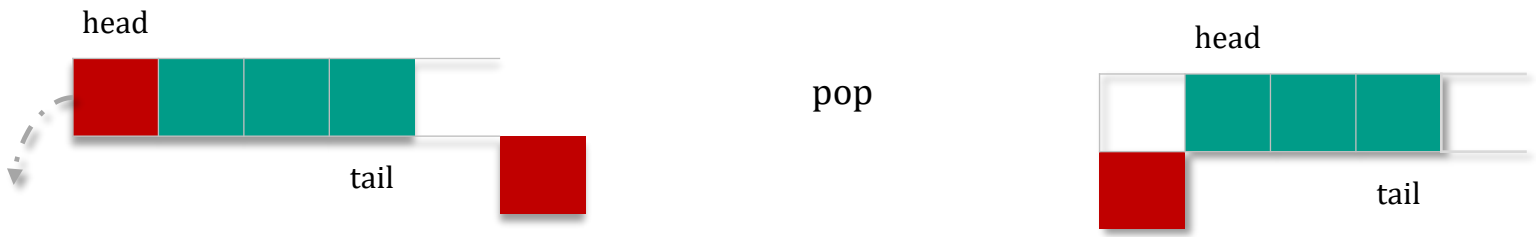
队列

队列($queue$),是一种操作受限制的线性的数据结构

其限制是只允许在队列的尾部进行插入,在前端进行删除

队列的修改具有为先进先出($first\ in\ first\ out$)的性质表,简称 $FIFO$ 表





队列的实现

```
int q[100005], h, t;
void push(int x)
{
  q[++t] = x;
}
void pop()
{
  if (t < h)
    return;
  ++h;
}
int front()
{
  if (t < h)
    return INF;
  return q[h];
}
```

C++ 中使用STL构造一个queue的语句为： queue <T> q

更多用法可以翻阅文档: http://cplusplus.com/reference/ queue / queue /?kw= queue

方法	功能
push	往队尾插入一个元素
pop	从队首删除一个元素
front	获取队首元素
size	获取队内元素个数

#418、沙龙舞会

【描述】

在电影《傲慢与偏见》里可以看到流行于 19 世纪的一种社交舞。男女排成两排，最前面的一对舞伴跳完后回到队伍的后面去。两队的人数可能不相同，所以配对的舞伴在不断变化。
现要求写一个程序，模拟上述舞伴配对问题。

【输入】

第一行两队的人数;
第二行舞曲的数目。

【输出】

配对情况。

【输入样例】

```
4 6
7
```

【输出样例】

```
1 1
2 2
3 3
4 4
1 5
2 6
3 1
```

```cpp
queue<int> qa, qb;
for (int i = 1; i <= n; i++)
  qa.push(i);
for (int i = 1; i <= m; i++)
  qb.push(i);
for (int i = 0; i < t; i++)
{
  cout << qa.front() << " " << qb.front() << endl;
  qa.push(qa.front()), qb.push(qb.front());
  qa.pop(), qb.pop();
}
```

#2798、双端队列

题目描述

对于一个队列 Q ,你需要实现以下几个操作

- `0 d x` ,当 d 为 0 时,将 x 放入队首,当 d 为 1 时将 x 放入队尾
- `1 idx` ,输出下标为 idx 的元素(下标从 0 开始)
- `2 d` ,当 d 为 0 时,删除队首元素,当 d 为 1 时,删除队尾元素

输入格式

第一行输入一个正整数 n

接下来 n 行给出 n 个操作

> *保证所有指令都合法*

输出格式

对于每个 `1 idx` 操作输出一行

输入样例

```
11
0 0 1
0 0 2
0 1 3
1 0
1 1
1 2
2 0
2 1
0 0 4
1 0
1 1
```

输出样例

```
2
1
3
4
1
```

输出规模

对于全部的数据 $1 \leq n \leq 4 \times 10^5, -10^9 \leq x \leq 10^9$

#2798、双端队列

双端队列是指一个可以在队首/队尾插入或删除元素的队列

具体地,双端队列支持的操作有 4 个:

- 在队首插入一个元素
- 在队尾插入一个元素
- 在队首删除一个元素
- 在队尾删除一个元素

方法	功能
push_back	往队尾插入一个元素
push_front	往队首插入一个元素
pop_back	删除队尾元素
pop_front	删除队首元素

数组模拟双端队列的方式与普通队列相同

C++ 中使用STL构造一个 deque 的语句为： deque <T> q

更多用法可以翻阅文档: http://cplusplus.com/reference/deque/deque/?kw= deque

#419、循环调度

描述

现有 n 个任务按顺序进入队列，CPU 通过循环调度法逐一处理这些任务，每个任务最多处理 q 毫秒。

如果 q 毫秒之后任务尚未处理完，那么该任务将被移动至队伍的最末尾，CPU 随即开始处理下一个任务。

例如，假设 q 是 100，有如下任务序列：

$A(150) - B(80) - C(200) - D(200)$

首先，A 被处理 100 ms，然后带着剩余的 $50ms$ 移到队尾。

$B(80) - C(200) - D(200) - A(50)$

B 被处理，在 $180ms$ 时完成，B 移出队列。

$C(200) - D(200) - A(50)$

C被处理，剩余 $100ms$ 移至队尾。

$D(200) - A(50) - C(100)$

之后同样处理，一直到所有的任务处理完毕。

请编写一个程序，模拟循环调度法。

输入

第一行，两个整数 n 和 q。接下来 n 行整数，表示每个任务处理需要的时间 t_i。

输出

按照任务完成的先后顺序输出各个任务完成时的结束时间，一行一个整数。

输入样例

```
5 100
150
80
200
350
20
```

输出样例

```
180
400
450
550
800
```

数据说明

对于全部的数据 $1 \leq n \leq 100000, 1 \leq q \leq 1000, 1 \leq t_i \leq 50000$

#2282、Windows消息队列

优先队列维护

题目描述

消息队列是 $Windows$ 系统的基础。对于每个进程，系统维护一个消息队列。如果在进程中有特定事件发生，如点击鼠标、文字改变等，系统将把这个消息加到队列当中。

同时，如果队列不是空的，这一进程循环地从队列中按照优先级获取消息。请注意优先级值低意味着优先级高。请编辑程序模拟消息队列，将消息加到队列中以及从队列中获取消息。

输入格式:

输入首先给出正整数 $N(N \leq 10^5)$，随后 N 行，每行给出一个指令——`GET` 或 `PUT`，分别表示从队列中取出消息或将消息添加到队列中。

如果指令是 `PUT`，后面就有一个消息名称、以及一个不超过 100 的正整数表示消息的优先级，此数越小表示优先级越高。

消息名称是长度不超过 10 个字符且不含空格的字符串；

题目保证输入至少有一个 `GET`。

输出格式:

对于每个 `GET` 指令，在一行中输出消息队列中优先级最高的消息的名称和参数。如果消息队列中没有消息，输出 `EMPTY QUEUE!`。对于 `PUT` 指令则没有输出。

输入样例:

```
9
PUT msg1 5
PUT msg2 4
GET
PUT msg3 2
PUT msg4 4
GET
GET
GET
GET
```

输出样例:

```
msg2
msg3
msg4
msg1
EMPTY QUEUE!
```

#2786、飞花令

题目描述

前段时间**人类最长飞花令**引起网友热议，出题者逐一给出 π 小数点后的数字，答题者“飞”出含有此数字的诗词，五位选手旗鼓相当，一来一往，直到突破小数点后第 203 位。
左手数学，右手诗歌，这场语文和数学的完美结合，让人大开眼界。不过作为总裁判的你，马上要将计算机编程引入这场**人类最长飞花令**，自动评测谁是优胜者。假如有 3 位选手参赛，则我们分别用大写字母 `A` 、 `B` 、 `C` 表示三人，而这也是三位选手的答题顺序。他们的答题过程我们用一个字符串表示，字符串中有 $0 \sim 9$ 十个数字及 `#` 号。

- $0 \sim 9$ 十个数字分别代表选手针对圆周率相应数字所答出诗句。如果数字相同，则代表该选手答对了，继续在场上等待作答；如果数字不同，则代表该选手答错了，判输并退出作答。
- 如果出现 `#` 号则代表相应选手未能在限定时间内作答，同样判输并退出作答。如果有选手未能“飞”出针对某数字的诗词，则后续的下一位选手继续就该数字**飞花**。

当场上只留下一位选手时，他/她就是最后的胜利者。

输入格式

输入数据有三行，第一行为选手个数 $n(1 \leq n \leq 6)$
第二行为圆周率，小数点后的数字不超过 200 位,长度足够判断出胜利者
第 3 行为 n 位选手的作答过程

输出格式

输出代表胜利者的大写字母。

输入样例

```
3
3.141592653589
314145926#
```

输出样例

```
A
```

#2786、飞花令

将给出的圆周率去掉小数点

并将所有人入队

使用两个变量维护圆周率和答案序列的当前位置

如果回答正确,将当前人员放回队尾

否则不放回

```
cin >> n;
cin >> PI;
PI.erase(PI.begin() + 1);
cin >> str;
queue<char> q;
for (int i = 0; i < n; i++)
  q.push(i + 'A');
while (pos1 < PI.size() && pos2 < str.size() && q.size() > 1)
{
  auto cur = q.front();
  q.pop();
  if (str[pos2] == PI[pos1])
    q.push(cur), pos1++;
  pos2++;
}
cout << q.front();
```

#1298、数据流的大小

题目描述

在大数据时代，数据会源源不断的收集得到并丢弃。现在初始有一个空的队列，有如下三种操作：

> $add\ x$: *在队列尾添加元素* x
> $delete$: *删除队列首的元素。如果队列为空，则不删除任何元素。*
> $query$: *询问队列中的元素之和。如果队列为空，则答案为* 0 。

那么你能对所有的询问都做出正确的回答吗？如果可以，那么恭喜你，你就是一名大数据工程师了！

题目输入

第一行是一个 n ，表示所有操作的次数。接下来 n 行，每行是三种操作形式之一，如样例输入所示。

题目输出

对于每个询问，单独输出一行表示答案。

数据范围

对于 50% 的数据有： $1 \leq n \leq 10^3$ 。

对于 100% 的数据有： $1 \leq n \leq 10^5, 1 \leq x \leq 10^6$ 。

样例输入

```
7
add 1
add 2
query
delete
query
add 4
query
```

样例输出

```
3
2
6
```

#1298、数据流的大小

使用一个变量sum记录对内元素之和

对于add操作$sum \mathrel{+}= x$

对于$delete$操作 $sum \mathrel{-}= q.front()$

对于每一种操作时间复杂度$O(1)$

#1182、走迷宫

题目描述

一个迷宫由 R 行 C 列格子组成，有的格子里有障碍物，不能走；有的格子是空地，可以走。

给定一个迷宫，求从左上角走到右下角最少需要走多少步(数据保证一定能走到)。只能在水平方向或垂直方向走，不能斜着走。

输入

第一行是两个整数， R 和 C ，代表迷宫的长和宽。（ $1 \leq R, C \leq 40$ ）

接下来是 R 行，每行 C 个字符，代表整个迷宫。

空地格子用 `.` 表示，有障碍物的格子用 `#` 表示。

迷宫左上角和右下角都是 `.`

输出

输出从左上角走到右下角至少要经过多少步（即至少要经过多少个空地格子）。计算步数要包括起点和终点。

输入样例

```
5 5
..###
#....
#.#.#
#.#.#
#.#..
```

输出样例

```
9
```

#1182、走迷宫

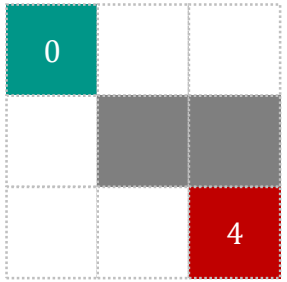
不难发现,BFS在遍历格点时时一层一层扩展的

从起点出发,在BFS遍历过程中记录当前的已经走了多少步

若v是从u移动过来的

那么 $step_v = step_u + 1$

在搜索过程中遇到了终点,此时的步数即为最小步数



最小步数

对于最小步数模型的BFS,队列内节点状态权值(步数)差值至多为1,队列内步数一定是**单调**的

初始状态下队列为空,满足单调性,设某一次队内步数满足单调性 $\{ X, X, X, X+1, X+1, X+1 \}$

对于下一次更新第一个X先出队，扩展得到 $X+1$ 入队,依然满足单调性(对于$\{ X, X, X, X\}$情况类似)

设队首元素为x当前步数为dis_x,队列后续某一个元素y

如果y能够将 dis_x 变得更小(y 能够通过到达 x),那么BFS最小步数无法保证正确性

由于队列的单调性满足$dis_y \geq dis_x$,若 $dis'_x = dis_y + z < dis_x \Rightarrow z < 0$

对于BFS最小步数模型而言,并不存在步数为负,与条件矛盾

同理可证明$Dijkstra$

#1170、最少步数

【题目描述】

在各种棋中，棋子的走法总是一定的，如中国象棋中马走“日”。

有一位小学生就想如果马能有两种走法将增加其趣味性，因此，他规定马既能按“日”走，也能如象一样走“田”字。

他的同桌平时喜欢下围棋，知道这件事后觉得很有趣，就想试一试，在一个（100×100）的围棋盘上任选两点 A、B，A 点放上黑子，B 点放上白子，代表两匹马。

棋子可以按“日”字走，也可以按“田”字走，俩人一个走黑马，一个走白马。

谁用最少的步数走到左上角坐标为 $(1,1)$ 的点时，谁获胜。现在他请你帮忙，给你 A、B 两点的坐标，想知道两个位置到 $(1,1)$ 点可能的最少步数。

【输入】

A、B 两点的坐标。

【输出】

最少步数。

#1170、最少步数

```
int bfs(int x, int y)
{
  memset(vis, 0, sizeof(vis));
  queue<node> q;
  vis[x][y] = true, q.push({x, y, 0});
  while (!q.empty())
  {
    cur = q.front(), q.pop();
    if (cur.x == 1 && cur.y == 1)
      return cur.step;
    for (int i = 0; i < 12; i++)
    {
      int tx = cur.x + dir[i][0], ty = cur.y + dir[i][1];
      if (tx > 0 && tx <= 100 && ty > 0 && ty <= 100 && !vis[tx][ty])
      {
        vis[tx][ty] = true;
        q.push({tx, ty, cur.step + 1});
      }
    }
  }
}
```

经典最小步数模型

BFS为什么不在出队时标记vis?

#1169、奇怪的电梯

【题目描述】

大楼的每一层楼都可以停电梯，而且第 i 层楼（$1 \leq i \leq n$）上有一个数字 $K_i(0 \leq K_i \leq n)$ 。

电梯只有四个按钮：开，关，上，下。上下的层数等于当前楼层上的那个数字。当然，如果不能满足要求，相应的按钮就会失灵。

例如：3 3 1 2 5 代表了 K_i（$K_1 = 3, K_2 = 3, \ldots\ldots$），从一楼开始。

在一楼，按“上”可以到 4 楼，按“下”是不起作用的，因为没有 -2 楼。

那么，从 A 楼到 B 楼至少要按几次按钮呢?

【输入】

共有二行，第一行为三个用空格隔开的正整数，表示 $n, A, B(1 \leq n \leq 200, 1 \leq A, B \leq n)$，第二行为 n 个用空格隔开的正整数，表示 K_i 。

【输出】

一行，即最少按键次数,若无法到达，则输出 -1 。

【输入样例】

```
5 1 5
3 3 1 2 5
```

【输出样例】

```
3
```

直接套用最小步数模型

当前楼层为i

那么下一步可到达$\min(i + k[i], n)$或者$\max(i - k[i], 0)$

谢谢观看