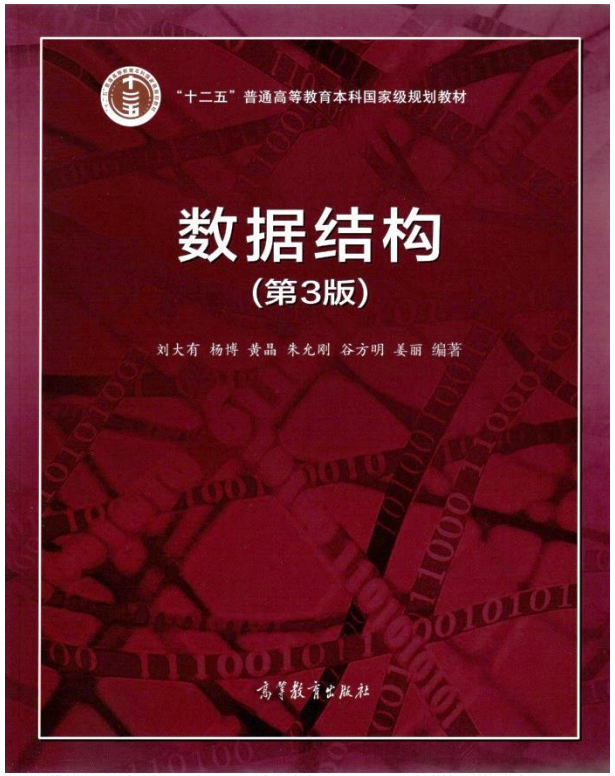
计算机学院王湘浩班

2024级



数据之法

结构之美

算法之道

队列及其应用

- 队列的性质
- 队列的实现
- 队列的应用

zhuyungang@jlu.edu.cn

Last updated on 2025.3

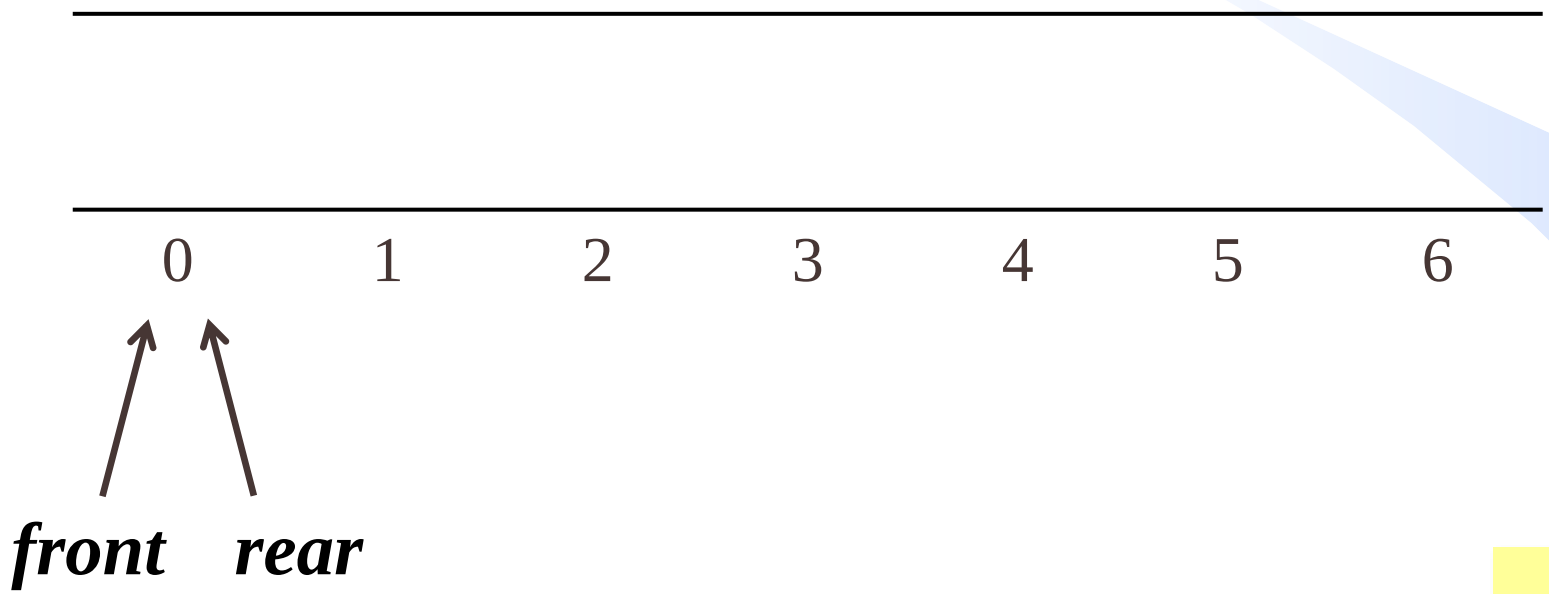
李煜东
2012年全国中学生信息学奥赛NOI金牌
2015年ACM-ICPC竞赛亚洲区域赛冠军
2017年毕业于北京大学
任职Google软件工程师

在大约一年半的时间里，我依靠3000道以上的巨大刷题量，才从一名连深度优先搜索都写不对的初学者，成长为NOI金牌得主，并入选国家集训队。

在思维的迷宫里，有的人凭天生的灵感直奔终点；有的人以持久的勤勉，铸造出适合自己的罗盘。

队列

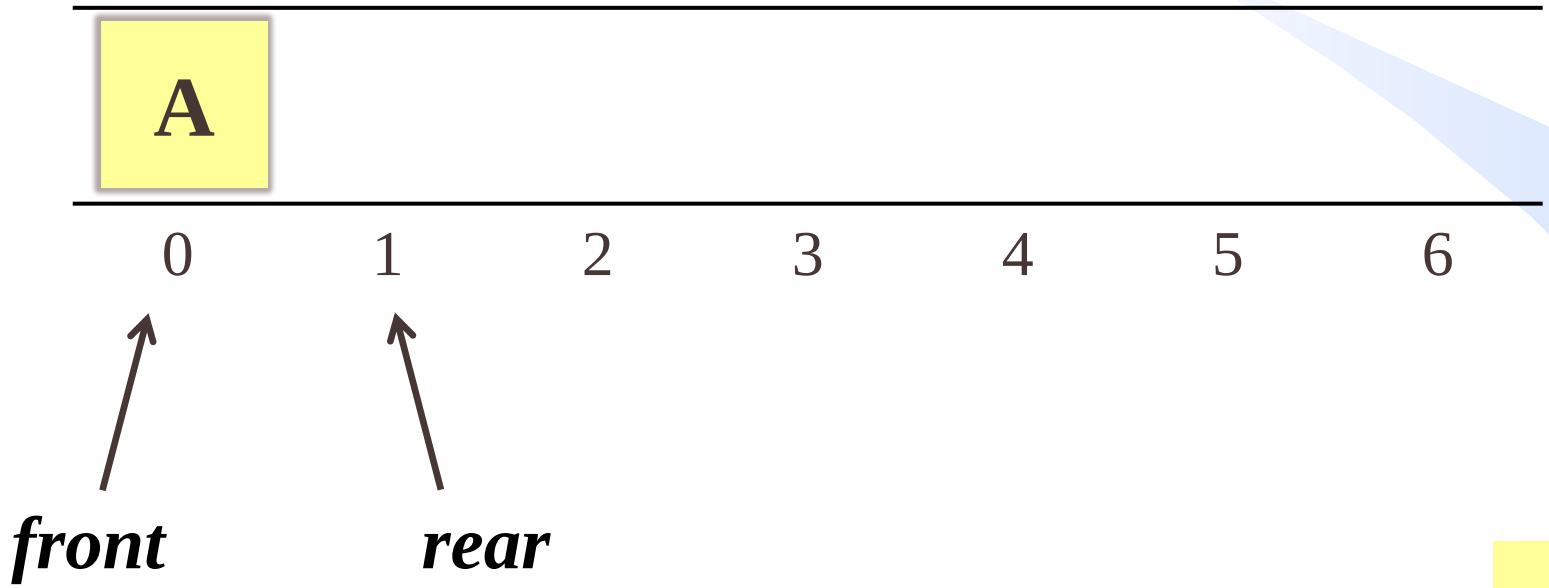
front指向队首元素，rear指向队尾元素的下一个位置



空队

队列

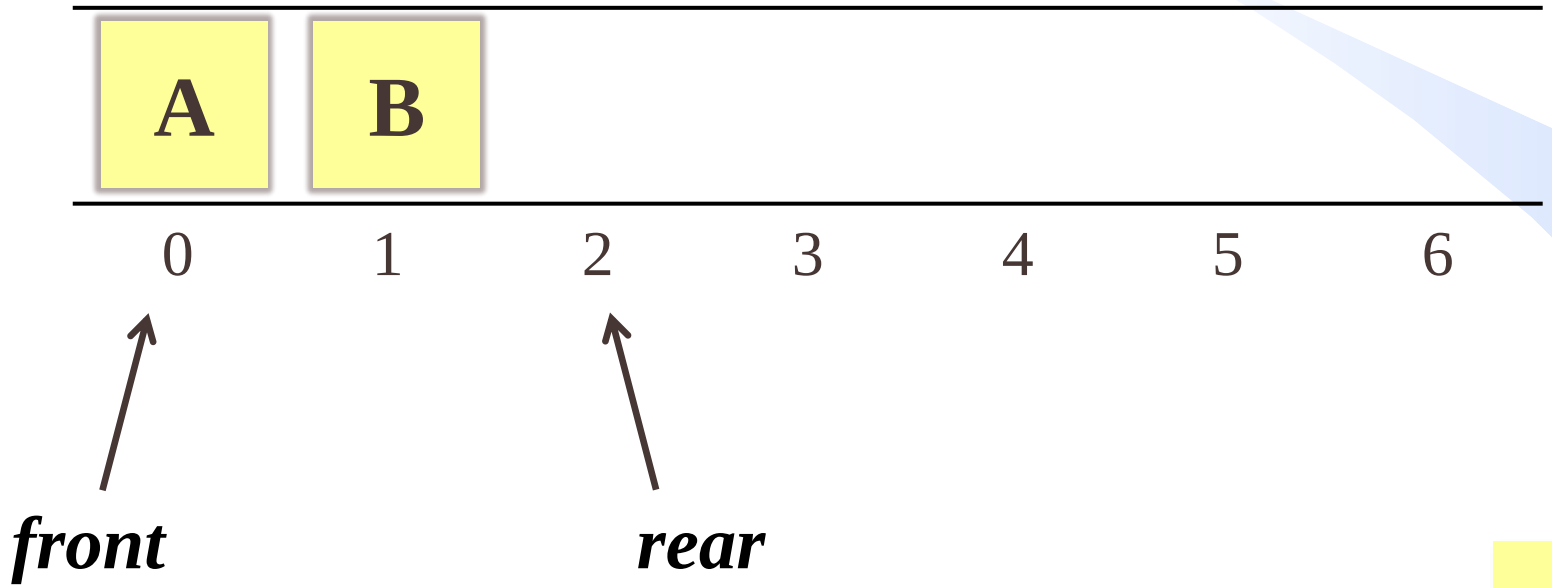
front指向队首元素，rear指向队尾元素的下一个位置



入队

队列

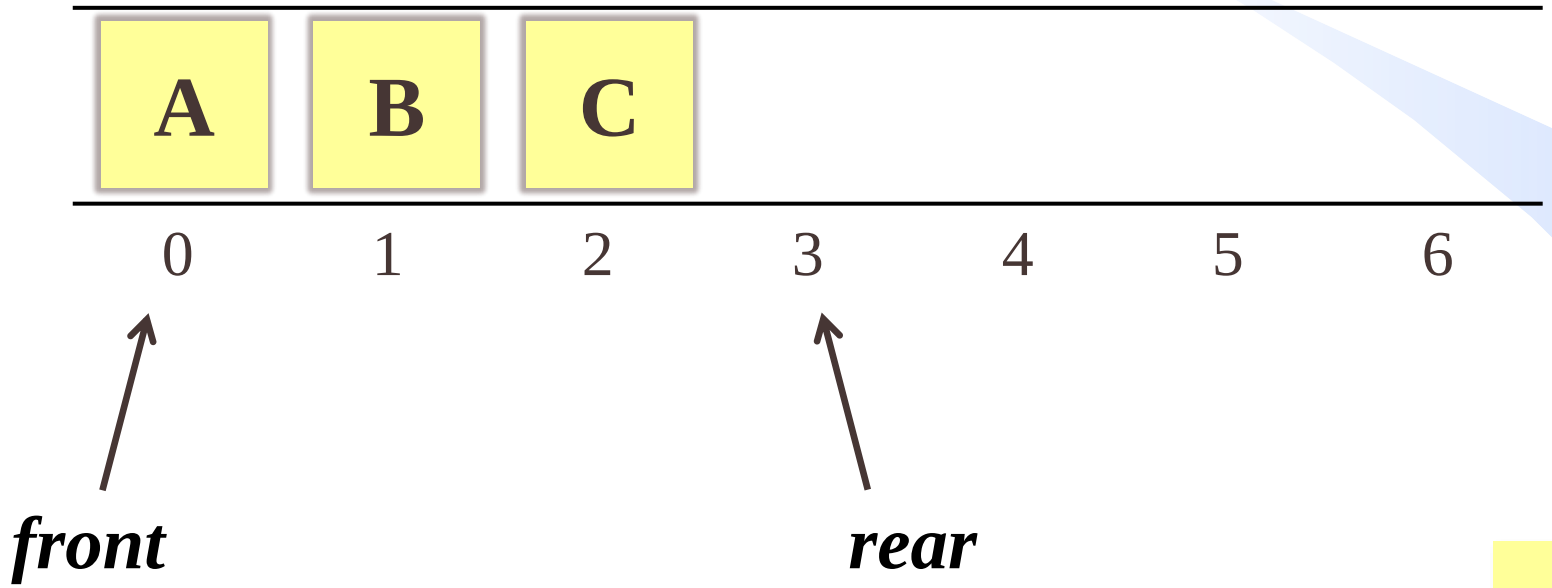
front指向队首元素，rear指向队尾元素的下一个位置



入队

队列

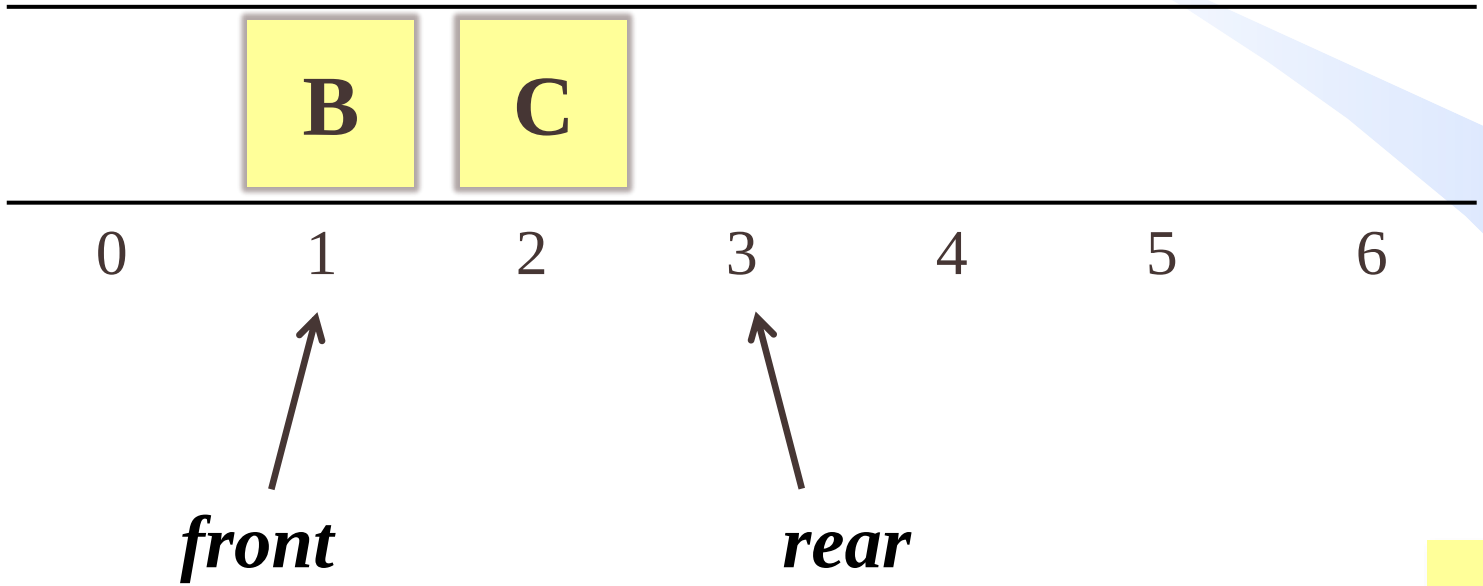
front指向队首元素，rear指向队尾元素的下一个位置



入队

队列

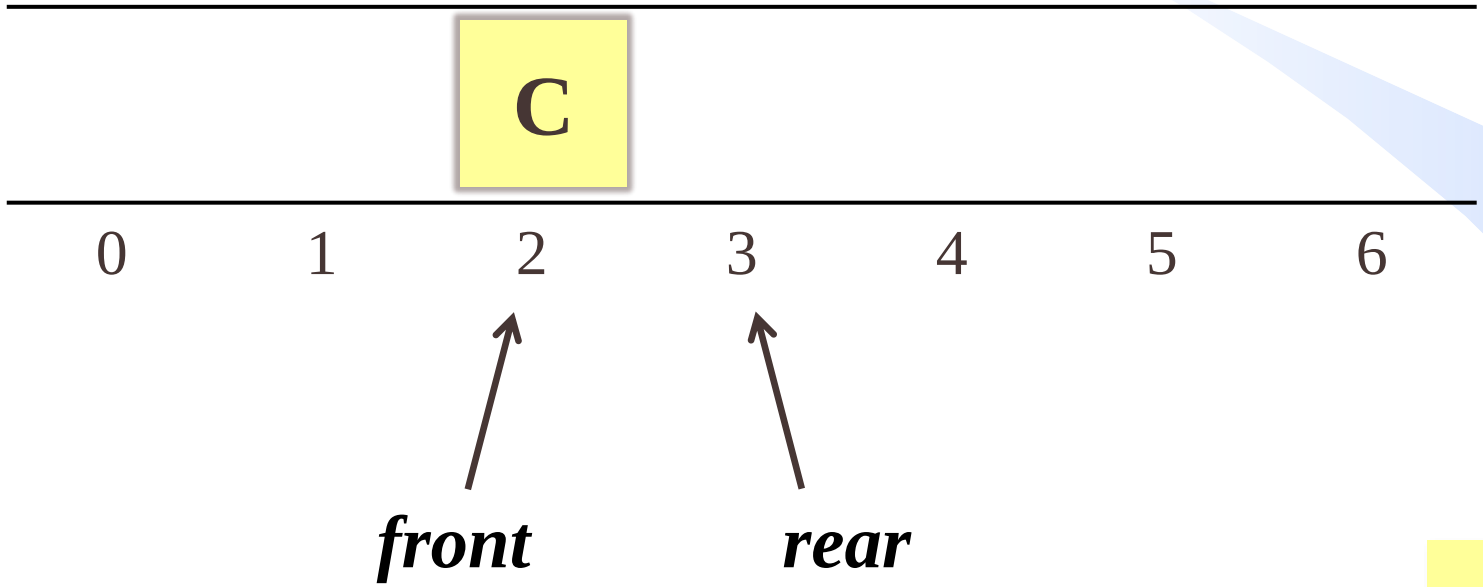
front指向队首元素，rear指向队尾元素的下一个位置



出队

队列

front指向队首元素，rear指向队尾元素的下一个位置

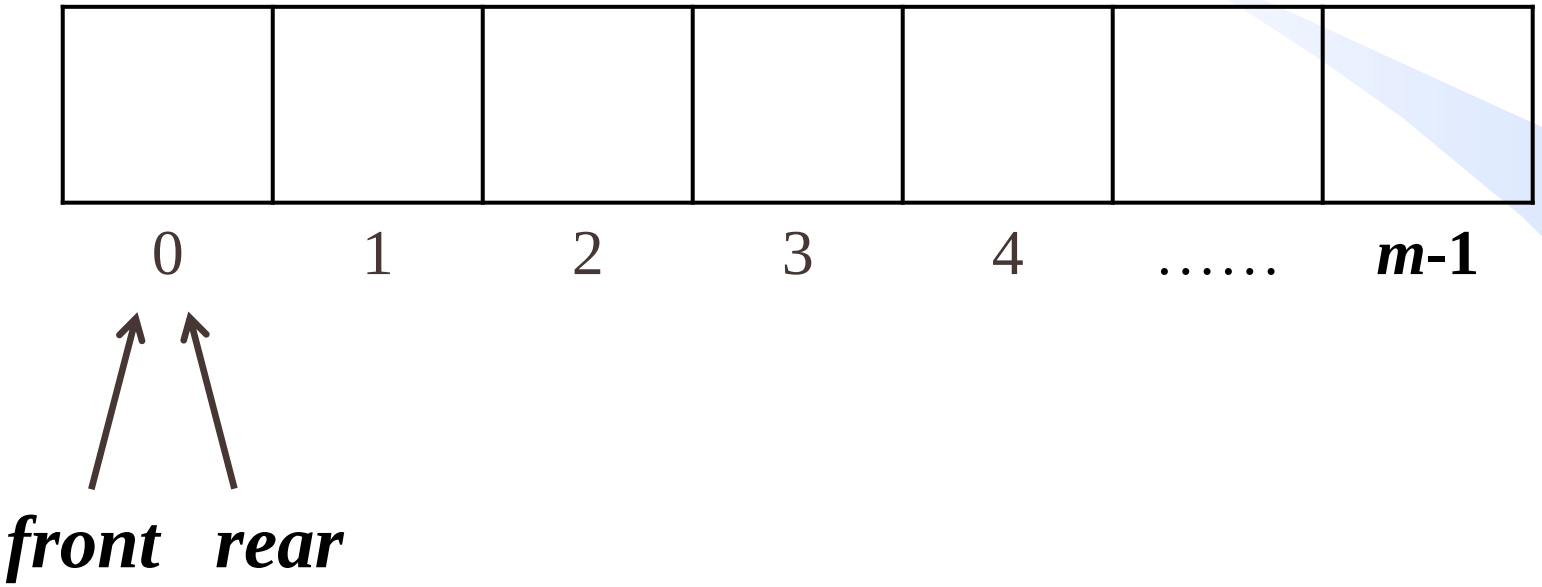


出队

先进先出

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

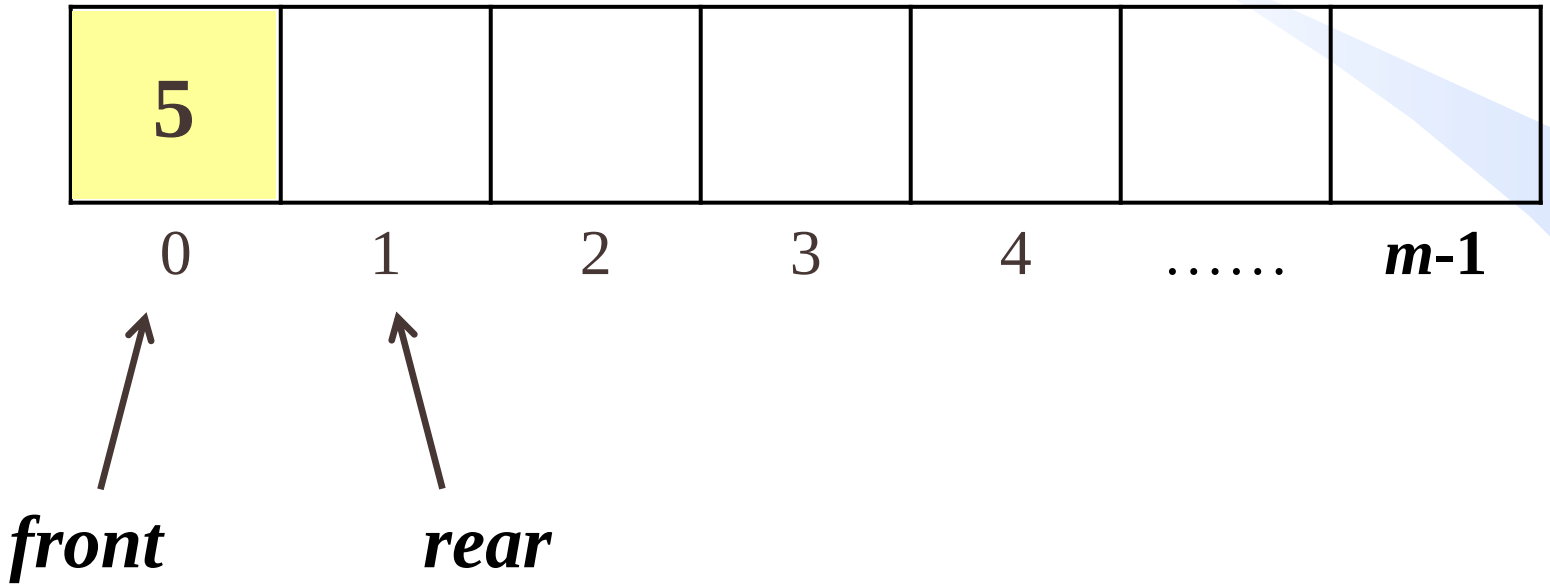


队空

front == rear

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

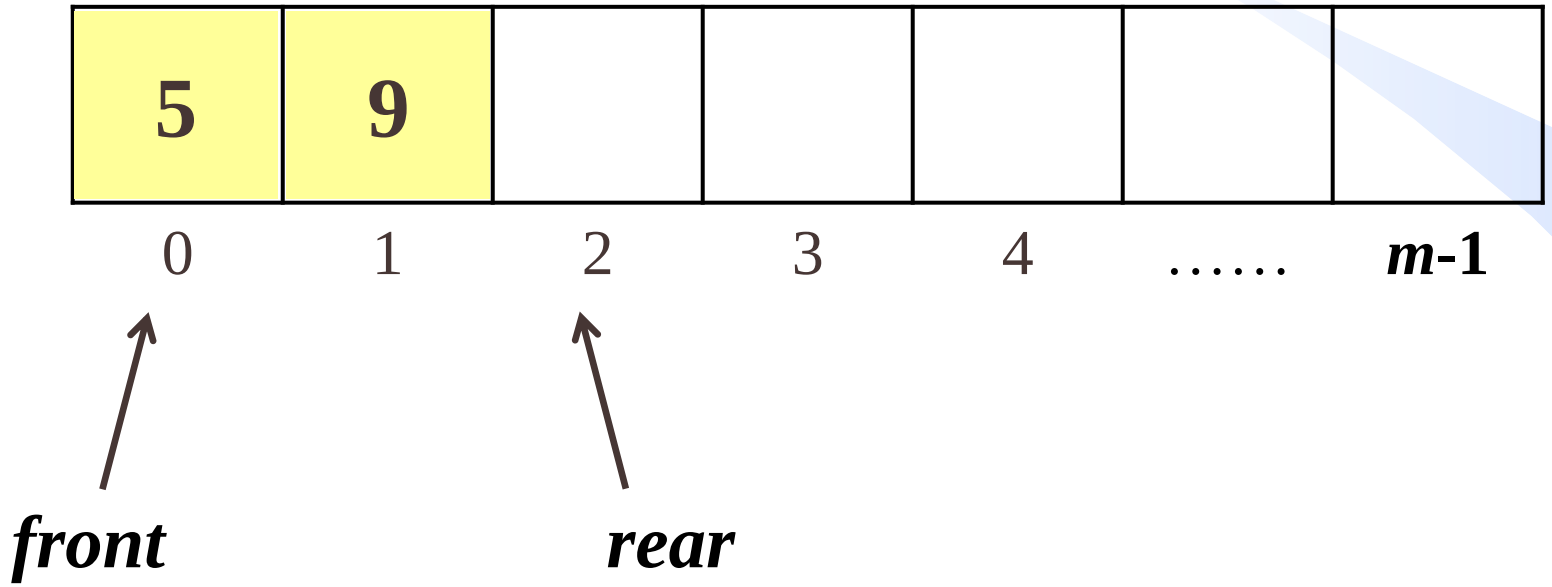


元素K入队

A[rear++]=K

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

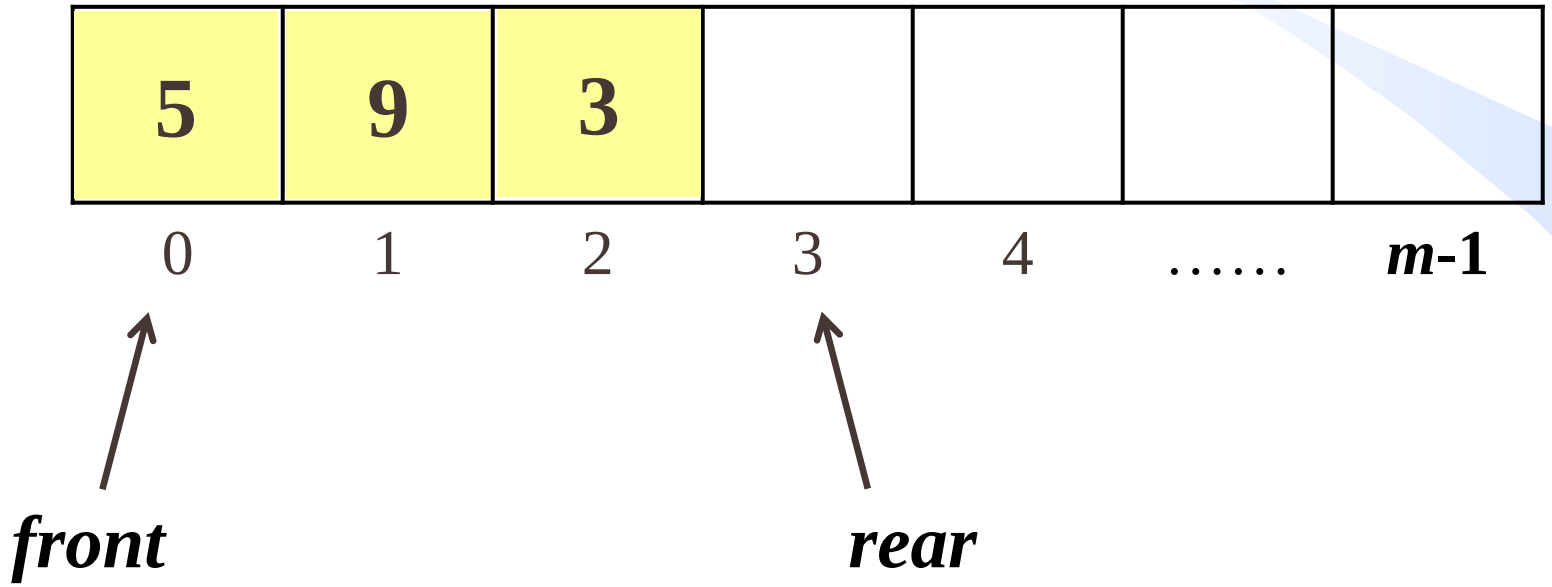


元素K入队

A[rear++]=K

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

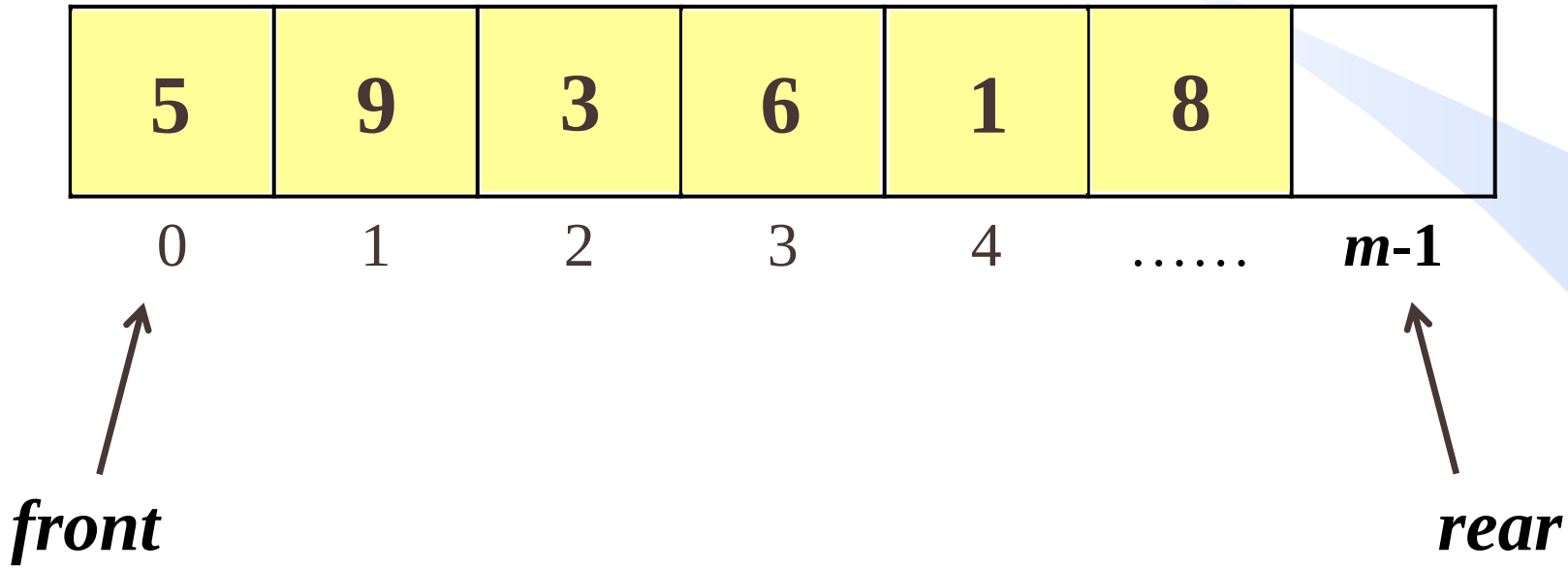


元素K入队

A[rear++]=K

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

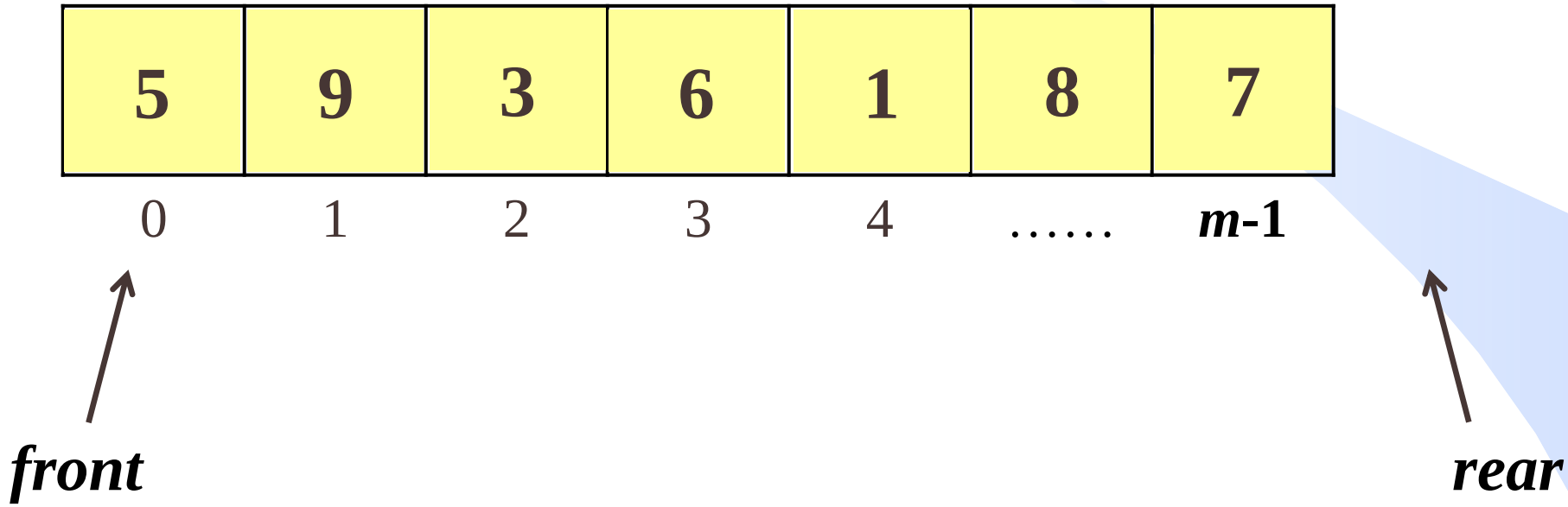


元素K入队

A[rear++]=K

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

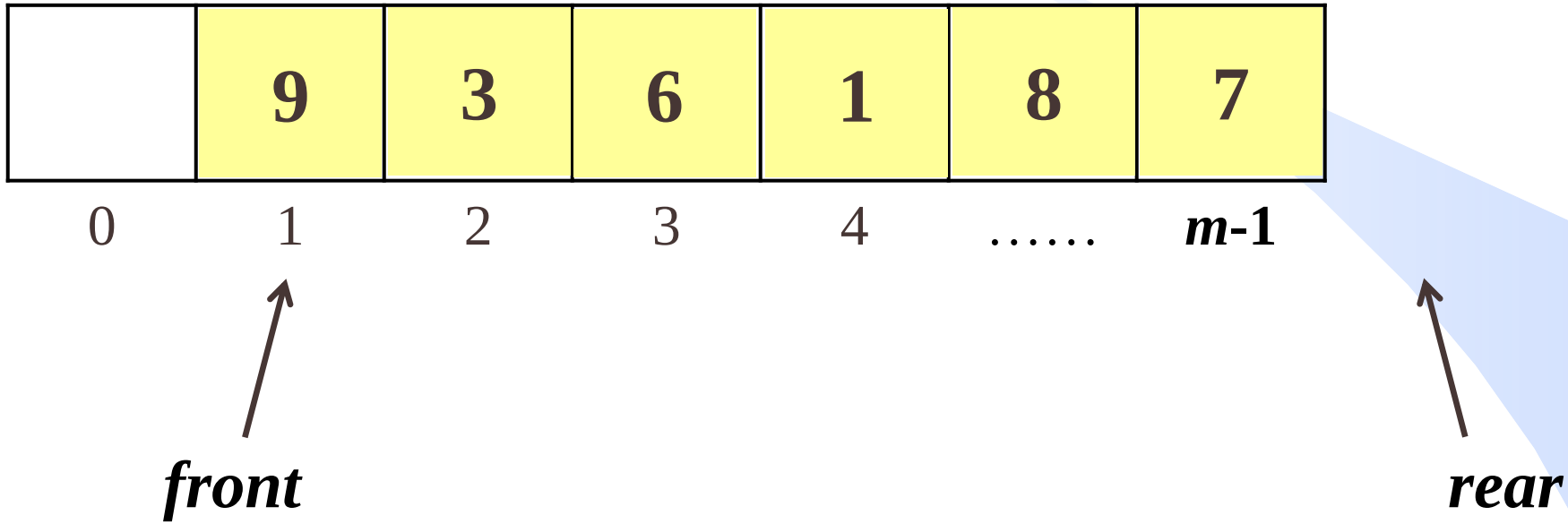


队满

```
rear == Maxsize
```

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

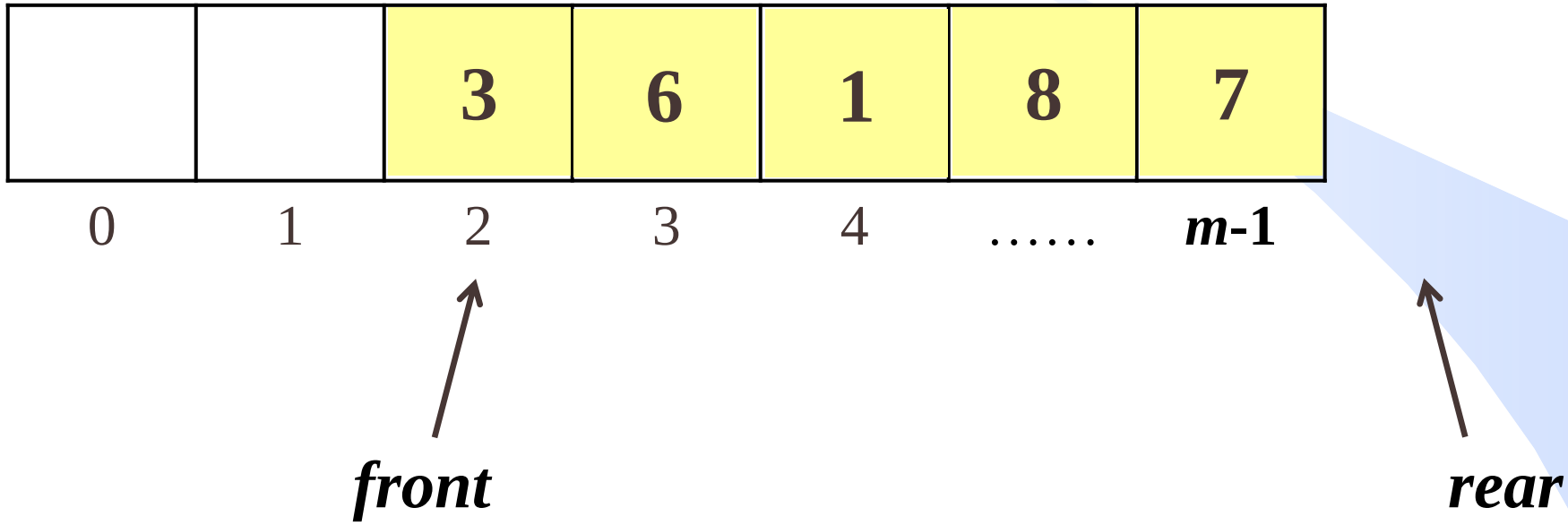


出队

```
return A[front++];
```

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置

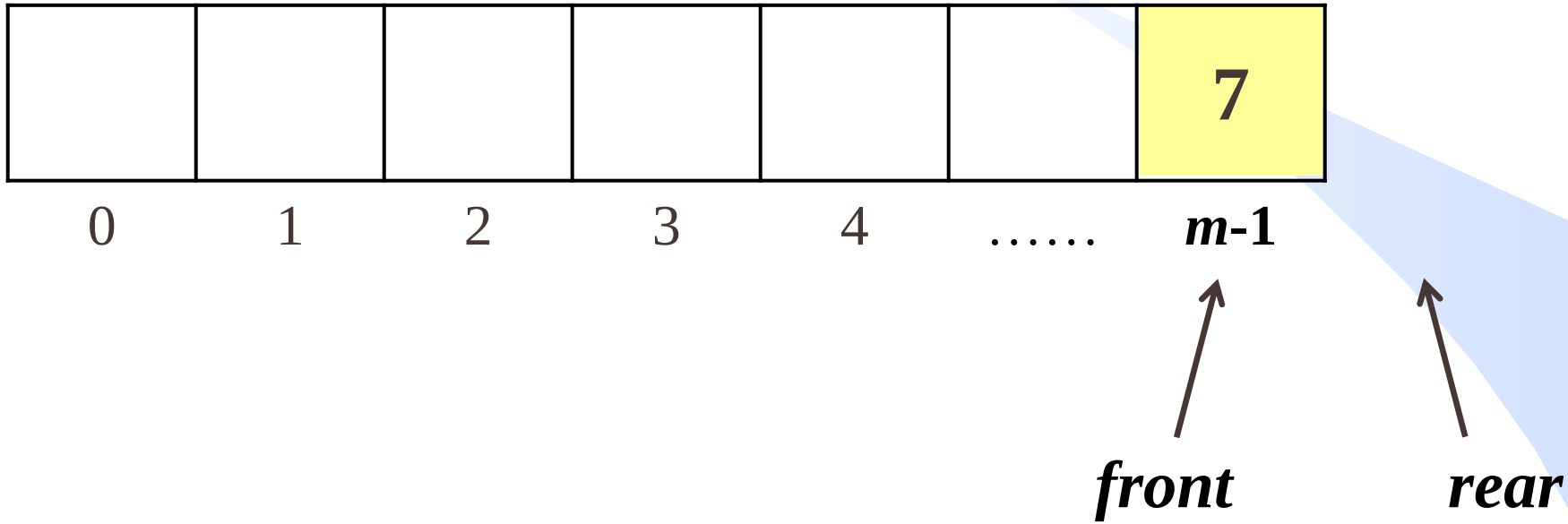


出队

```
return A[front++];
```

用数组实现队列

front指向队首元素，rear指向队尾元素的下一个位置



假“溢出”：队实际没满，但元素已无法入队

$(rear+1)\%m$

解决方案
循环队列

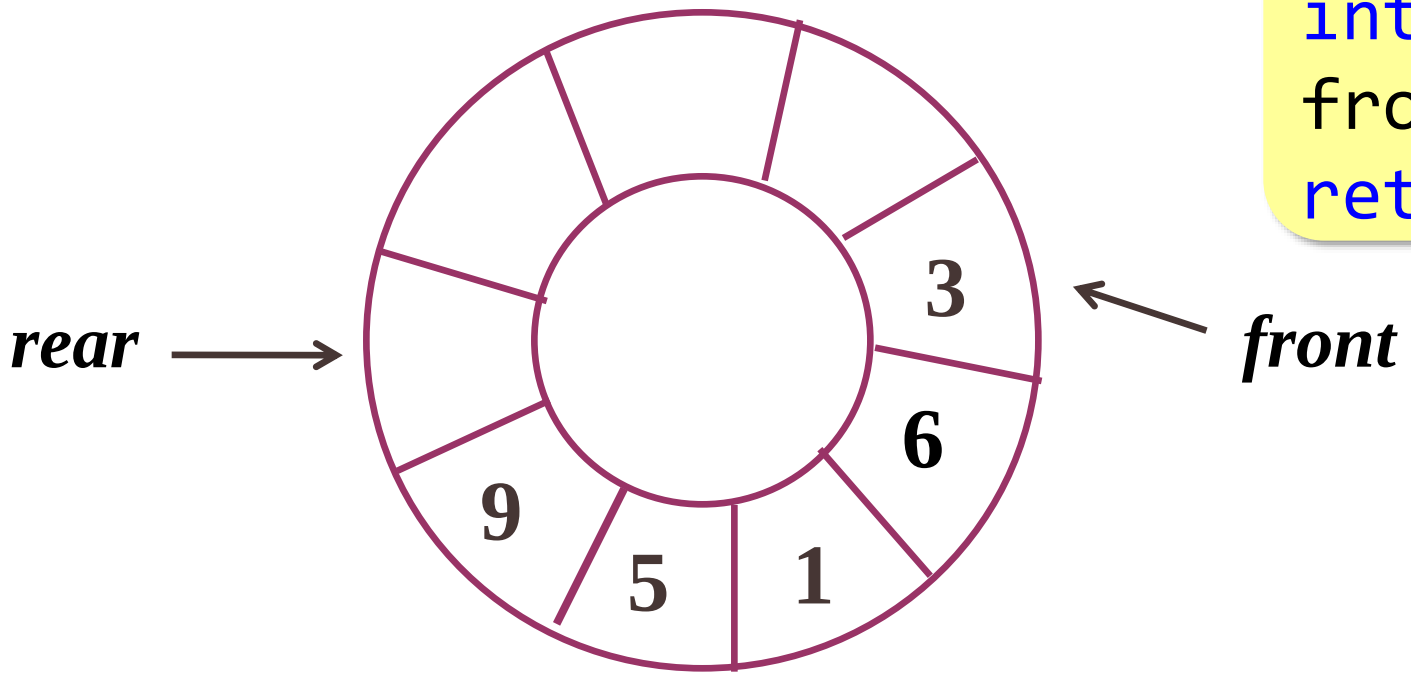
循环队列的数组实现

A

假定数组是循环的，即采用环状模型来实现队列。出队和入队即将*front*和*rear*顺时针移动一位。

出队

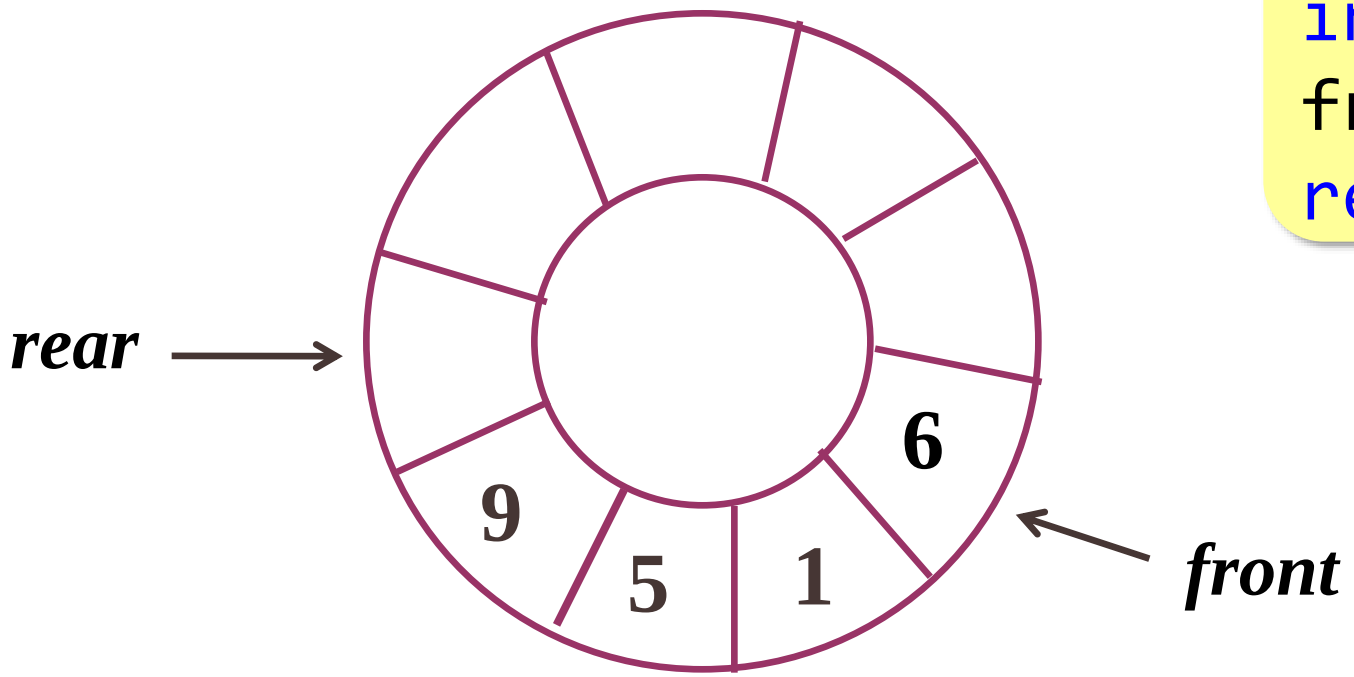
```
int K=A[front];
front=(front+1)%m;
return K;
```



循环队列的数组实现

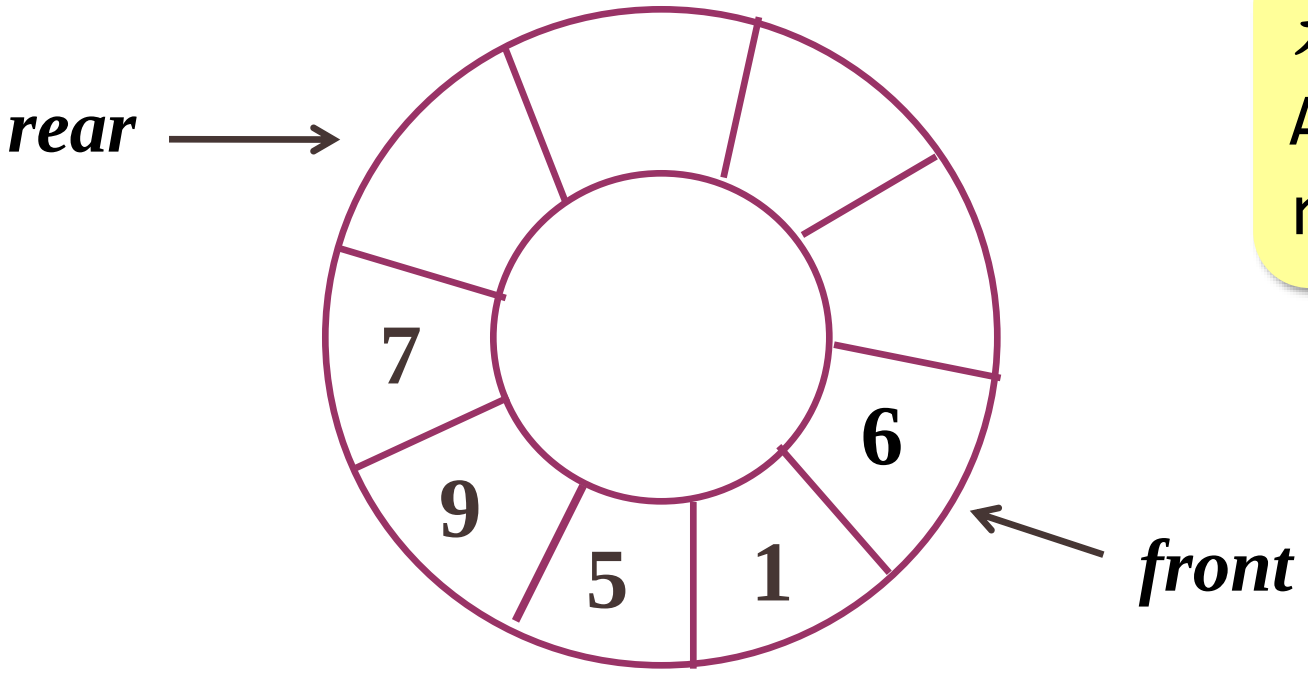
假定数组是循环的，即采用环状模型来实现队列。出队和入队即将*front*和*rear*顺时针移动一位。

```
出队
int K=A[front];
front=(front+1)%m;
return K;
```



循环队列的数组实现

假定数组是循环的，即采用环状模型来实现队列。出队和入队即将*front*和*rear*顺时针移动一位。

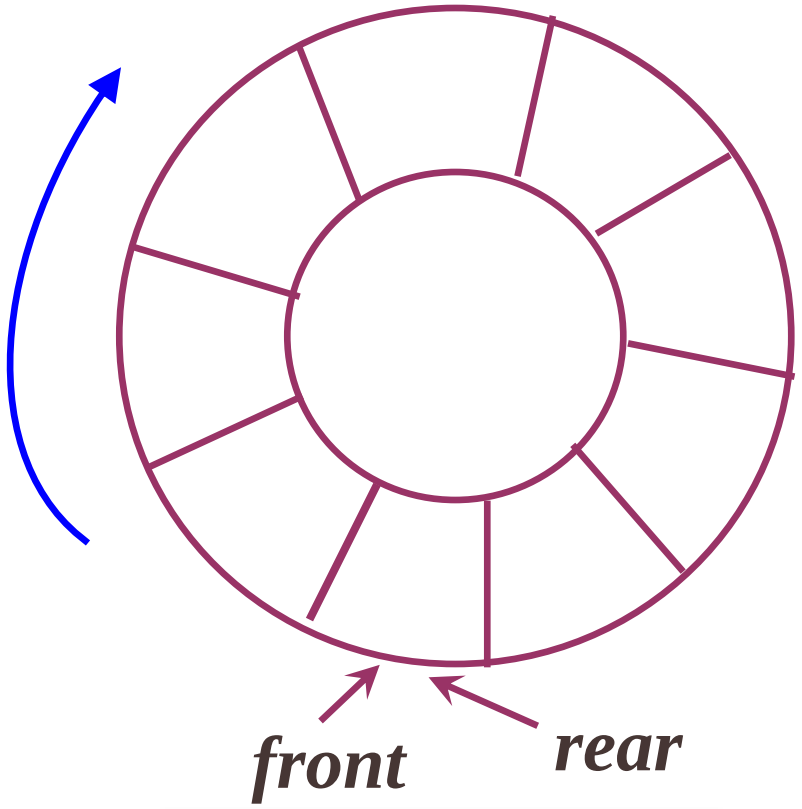


元素K入队

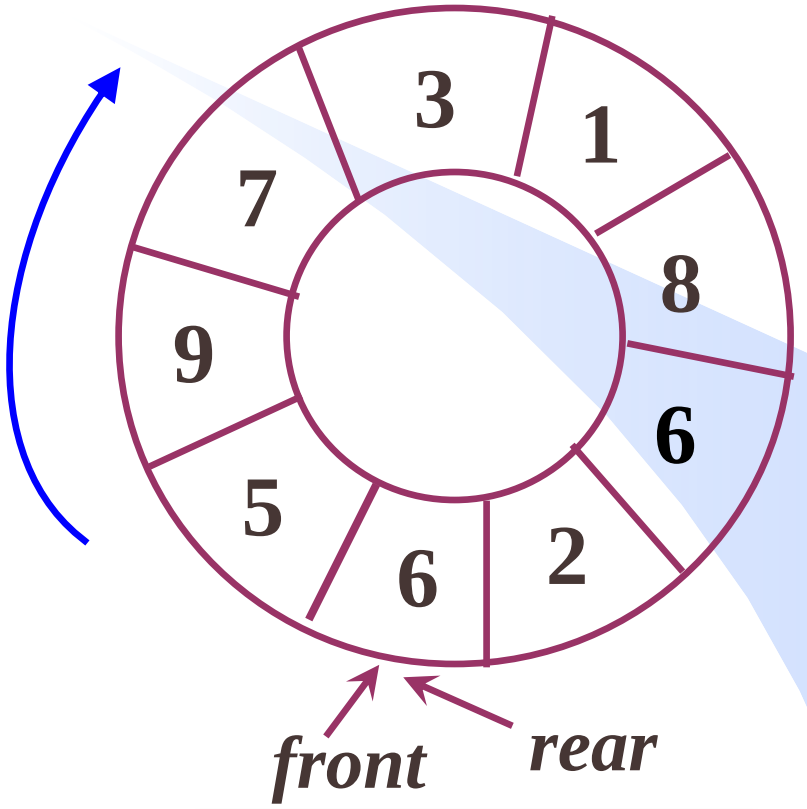
```
A[rear]=K;
rear=(rear+1)%m;
```

循环队列的数组实现

front指向队首元素，rear指向队尾元素的下一个位置



队空
front==rear

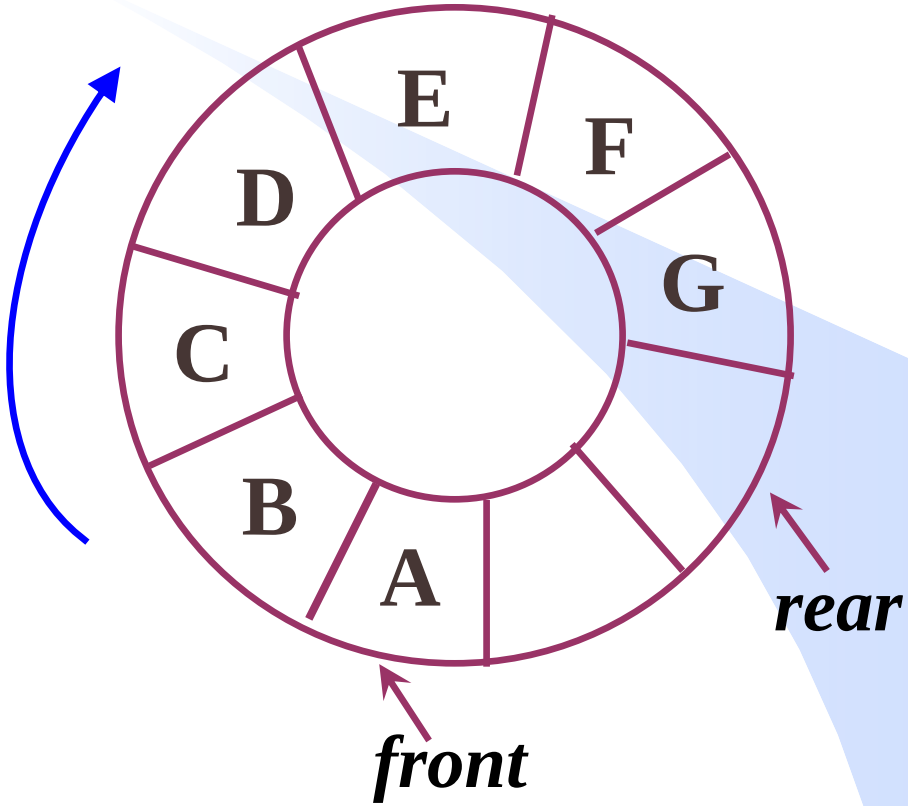


队满
front==rear

循环队列的数组实现

A

- 实现方案1：加一个变量*count*保存队列元素个数
  - ✓初始：front=0,rear=0,count=0;
  - ✓队空：count==0
  - ✓队满：count==m
  - ✓队中元素个数：count

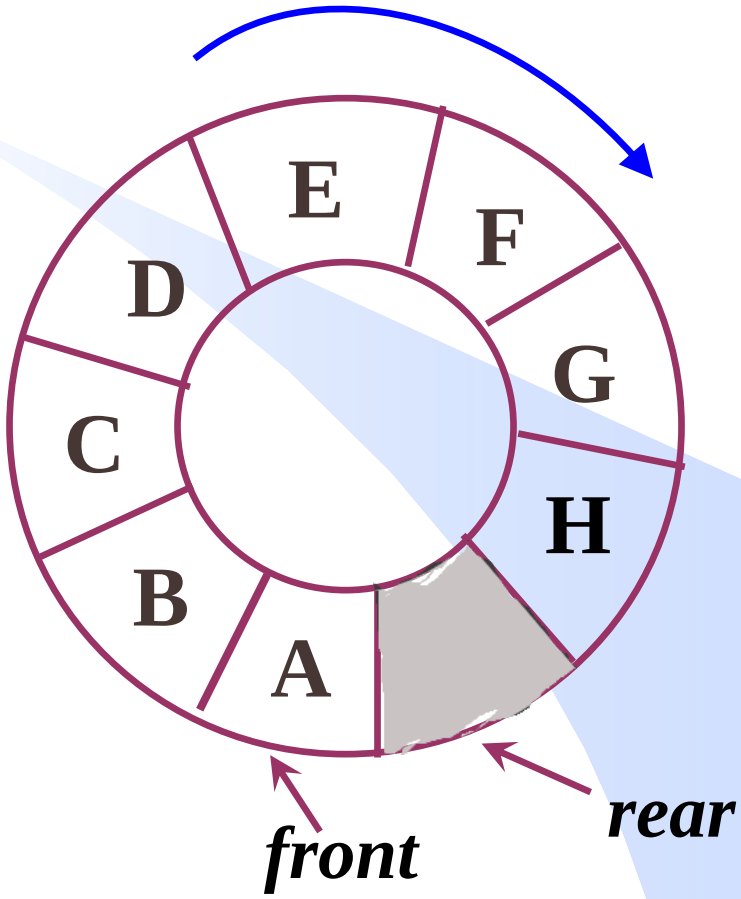


- 优点
  - ✓队列每个空间都能用上
- 缺点
  - ✓多了一个变量
  - ✓入队、出队需要对count加减1

时间换空间

循环队列的数组实现

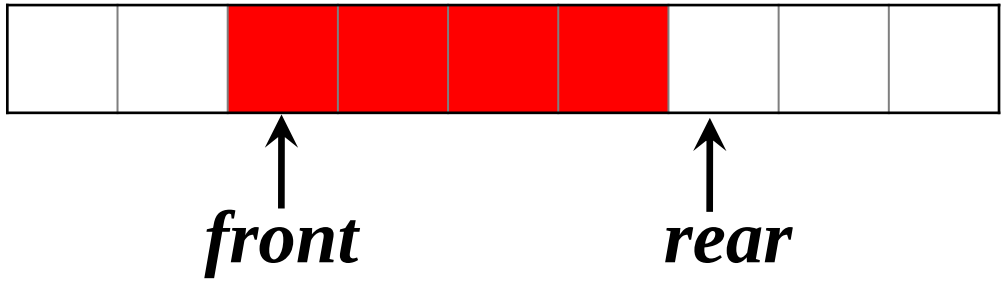
- 实现方案2：队尾后一个位置永远空着
  - ✓初始：front=0,rear=0;
  - ✓队空：front==rear
  - ✓队满：(rear+1)%m==front
  - ✓队中元素个数：



循环队列的数组实现

- 实现方案2：队尾后一个位置永远空着
  - ✓初始：front=0,rear=0;
  - ✓队空：front==rear
  - ✓队满：(rear+1)%m==front
  - ✓队中元素个数：(rear-front+m)%m

rear≥front：rear-front

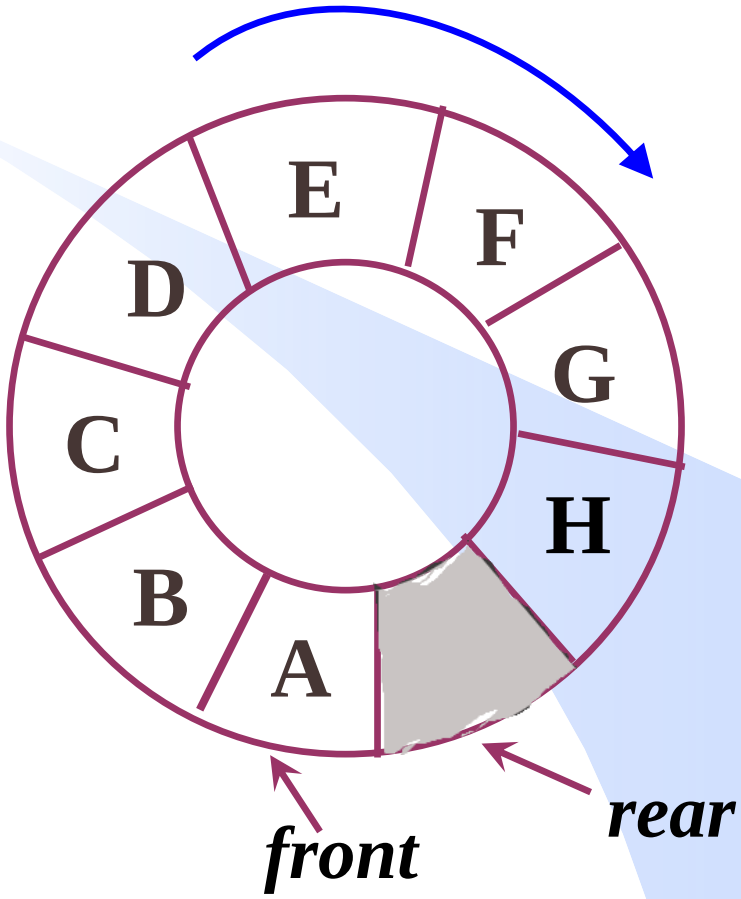


rear<front：rear-front+m

循环队列的数组实现

- 实现方案2：队尾后一个位置永远空着
  - ✓初始：front=0,rear=0;
  - ✓队空：front==rear
  - ✓队满：(rear+1)%m==front
  - ✓队中元素个数：(rear-front+m)%m

- 优点
  - ✓不需要count变量
  - ✓入出队也无需对count加减1
- 缺点
  - ✓浪费1个空间



空间换时间

课下思考

A

有一个用数组*C*[1..*m*]实现的环形队列，*m*为数组的长度。假设*f*为队头元素在数组中的位置，*r*为队尾元素的后一位置（按顺时针方向）。队列中元素个数为_____【国家电网、阿里笔试题】

A. (*m*+*r*-*f*) % *m*　　B. *r*-*f*　　C. (*m*-*r*+*f*) % m

D. (*m*-*r*-*f*) % *m*　　E. (*r*-*f*) % *m*

循环队列的数组实现

A

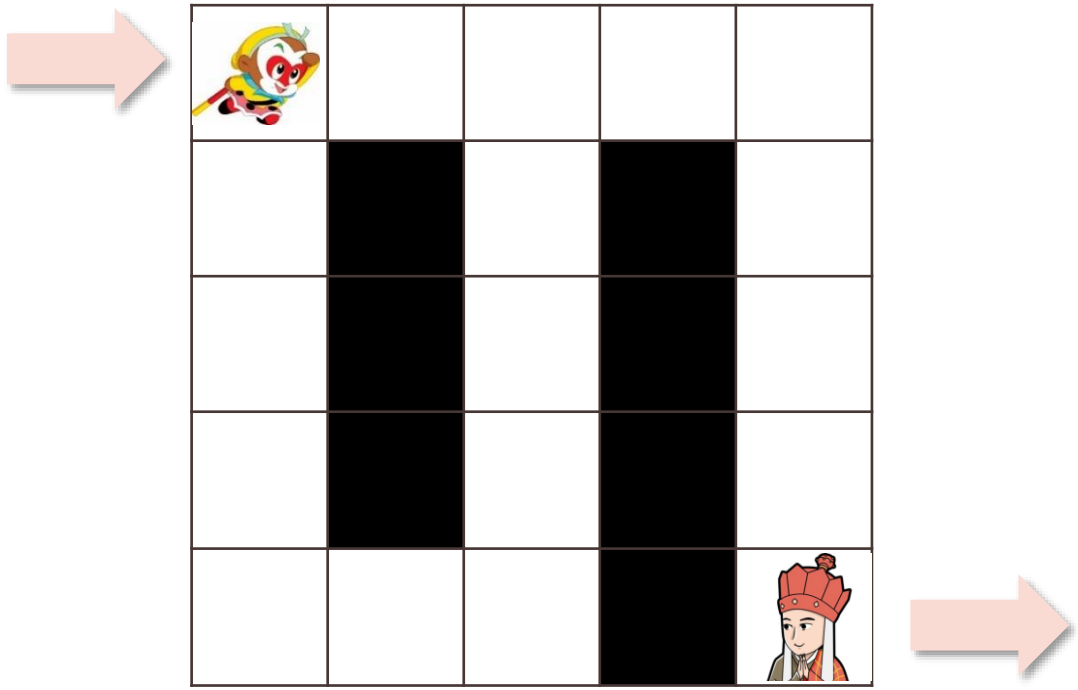
根据具体问题采用更简单的实现方法

```
const int MaxSize = 1e4 + 10;
template <class T>
class Queue{
private:
      T A[MaxSize];
      int front = 0, rear = 0;
public:
      bool empty() { return front==rear; }
      bool full() { return (rear+1) % MaxSize == front; }
      void enQueue(T K) {
            assert(!full());
            A[rear] = K; rear = (rear+1) % MaxSize;
      }
      T deQueue() {
            assert(!empty());
            T K = A[front]; front = (front+1) % MaxSize;
            return K;
      }
};
```

迷宫寻径

B

一个迷宫由n行m列格子组成($1 \le n,m \le 40$)。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），计算步数要包括起点和终点。【北京大学2019年保研机试，吉林大学2024年保研夏令营机试，Openjudge 3752】

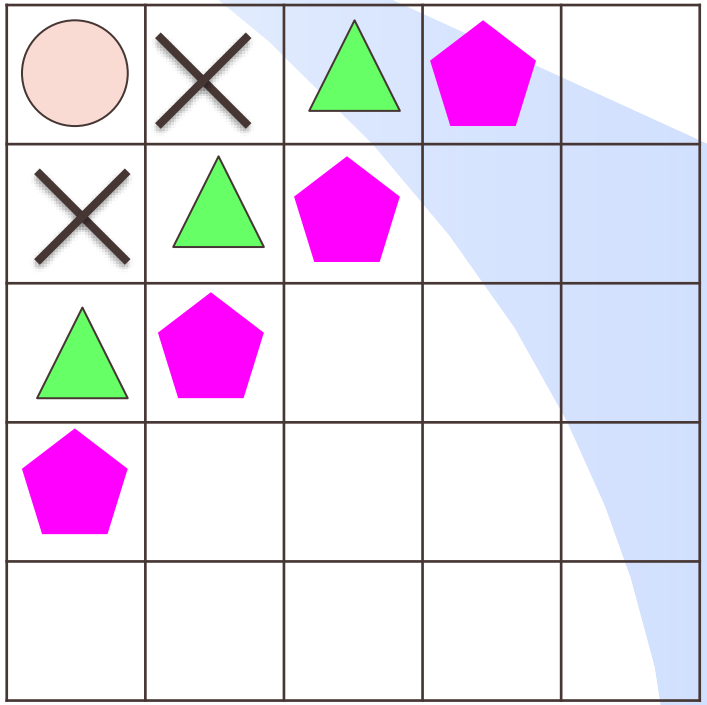


搜索过程

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

广度优先搜索
Breadth First Search, BFS



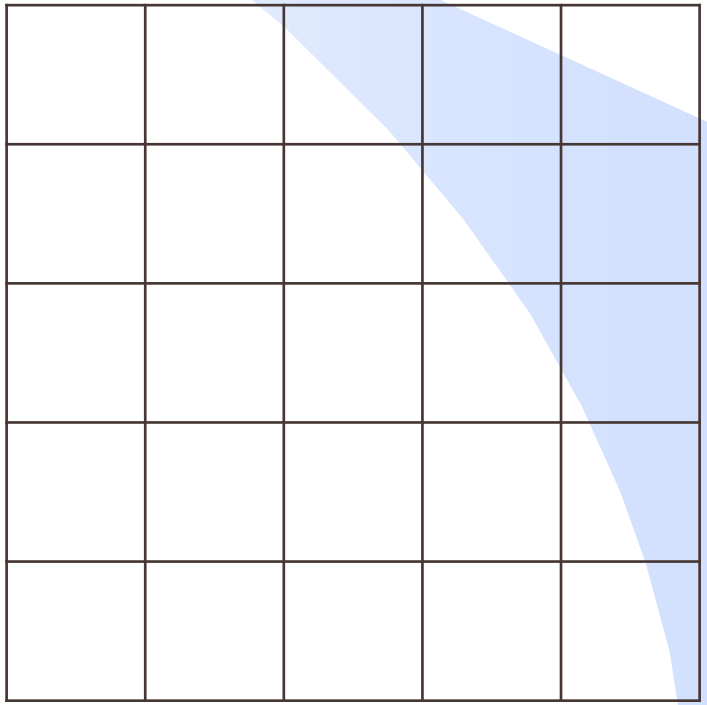
性质：搜索过程中每个格子第一次被访问时，找到该格子与起点的最短距离。

算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队



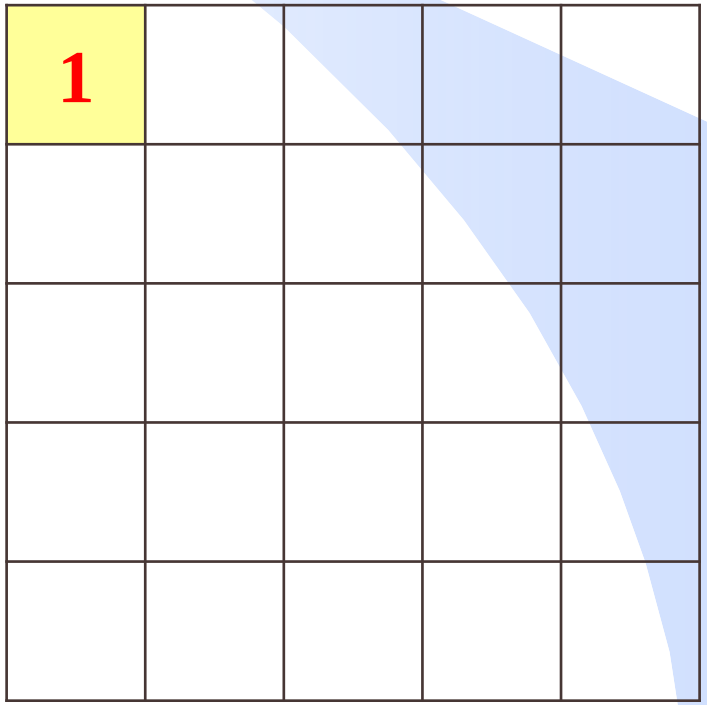
算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队

1



算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队

2	3

1	2			
3				

算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队

3	4	5

1	2	4		
3	5			

算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队

4	5	6

1	2	4		
3	5			
6				

算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队

5	6	7	8

1	2	4	7	
3	5	8		
6				

算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队

6	7	8	9

1	2	4	7	
3	5	8		
6	9			

算法实现

B

- 初始位置(0, 0)花费1步即可到达；
- 利用(0, 0)，访问距起点2步的点；
- 利用2步到达的格子，访问距起点3步的点；
- ……
- 如此下去，直至访问到出口。

队列：按访问的顺序把各点依次入队

7	8	9	10

1	2	4	7	
3	5	8		
6	9			
10				

算法实现

B

起点入队

```
while( 队列不空 ) {
        出队一个点p
        把p的未访问的相邻点入队并访问
        如果相邻点是出口则退出
}
```

8	9	10	11	12

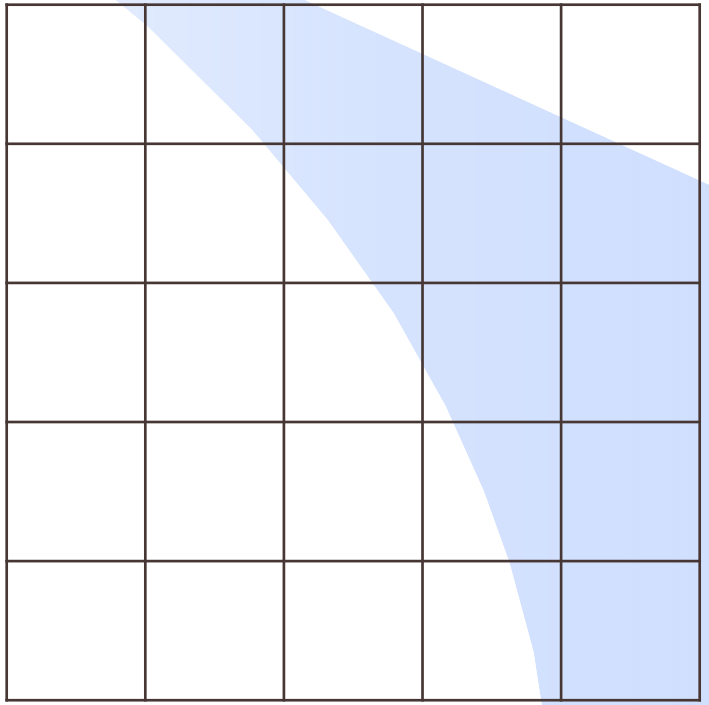
1	2	4	7	11
3	5	8	12	
6	9			
10				

算法实现

B

- 二维数组map存储地图。
- 二维数组vis（规模与map相同）:vis[*i*][*j*]=0表示点(*i*, *j*)没被访问过，vis[*i*][*j*]=1表示点(*i*, *j*)被访问过。
- 二维数组dist（规模与map相同）: dist[*i*][*j*]表示起点到点(*i*, *j*)的最短距离。

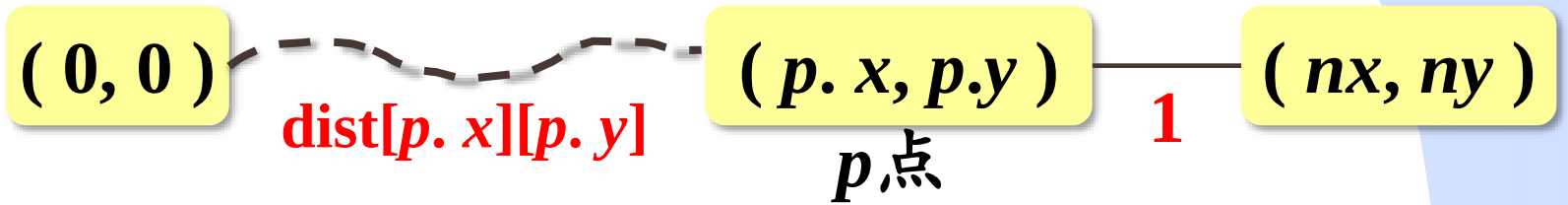
```
const int maxn = 50;
const int MaxSize = maxn*maxn;
struct point {
      int x, y;
      point(int a=0, int b=0) { x = a; y = b; }
};
int dx[4] = { -1,0,1,0 }, dy[4] = { 0,1,0,-1 };
char map[maxn][maxn];
int vis[maxn][maxn]={{0}};
int dist[maxn][maxn]={{0}};
```



```
int BFS(int n,int m){
    Queue<point> Q;
    Q.enQueue(point(0, 0));    //起点入队
    dist[0][0] = 1; vis[0][0] = 1;
    while (!Q.empty()) {
        point p = Q.deQueue();              //出队一个点
        for (int i = 0; i < 4; i++) {       //将当前点相邻的合法点入队
            int nx = p.x + dx[i], ny = p.y + dy[i];
            if (feasible(nx,ny)) {          //若点(nx, ny)合法、可以走
                vis[nx][ny] = 1;
                dist[nx][ny] = dist[p.x][p.y] + 1;
                if (nx==n-1 && ny==m-1) return dist[nx][ny];
                Q.enQueue(point(nx, ny)); //点(nx, ny)入队
            }
        }
    }
    return -1;
}
```

vis[*i*][*j*]表示点(*i*,*j*)是否被访问过，初始时所有值均为0

dist[*i*][*j*]表示起点到点(*i*,*j*)的最短距离



深度优先搜索vs广度优先搜索

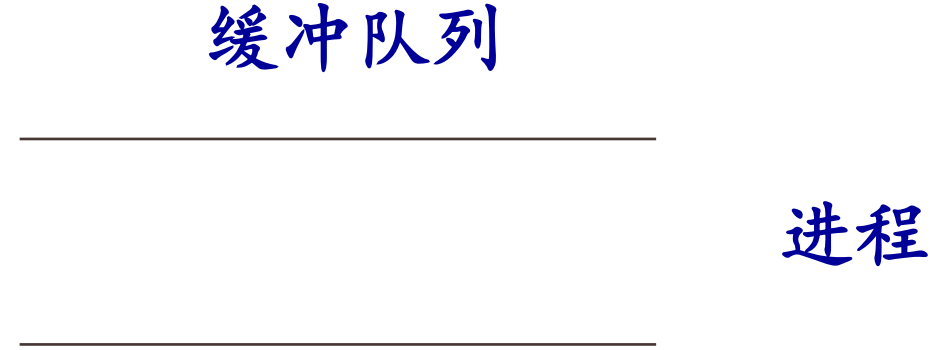
B

- 深度优先搜索（Depth First Search，DFS）：一条道走到黑，不撞南墙不回头，一般通过回溯法实现。
- 广度优先搜索（Breath First Search，BFS）：地毯式搜索，层层推进，常用于找最短路径。

D

队列的应用

- 缓冲



D

队列的应用

CPU

就绪进程队列

C++ STL中的队列 queue<T>

C

成员函数	作用
bool empty()	判断队空
int size()	返回队列中元素个数
void push(T x)	将x入队
void pop()	出队
T front()	返回队头元素

例:

```
queue<int> q;  //创建一个队列，其元素为int型
q.push(5);
int x = q.front();
q.pop();
```

自愿性质OJ练习题

- ✓LeetCode 622（循环队列）
- ✓LeetCode 225（用队列实现栈）
- ✓LeetCode 232（用栈实现队列）
- ✓Openjudge 3752（迷宫求最短距离）
- ✓POJ 3032（简单队列模拟）
- ✓POJ 2259（简单队列模拟）