第四章　栈与队列

张史梁
slzhang.jdl@pku.edu.cn

排队问题

- 银行排队
- 公交车排队
- 打饭排队
- ….

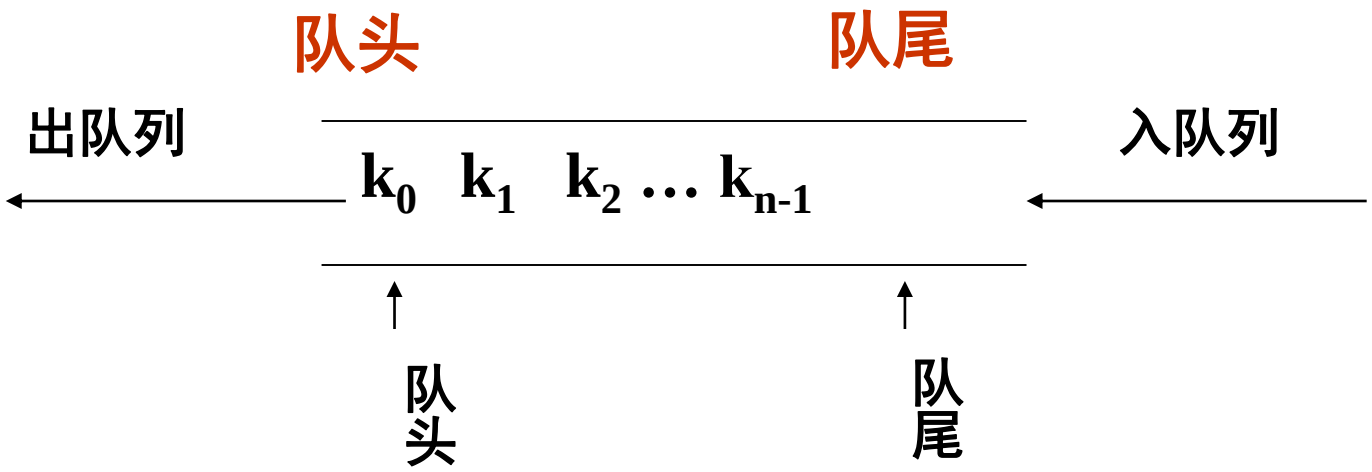
- 如何保证等待线性表中较先等待的实体能够较早地使用资源（被处理）？

队列 — 内容提要

- 定义和操作
- 队列的实现
- 队列的应用

队列的定义

- 先进先出（First In First Out, FIFO)
  - 限制访问的线性表
    - 按照达到的顺序来释放元素
    - 只允许在表的一端（队尾）插入，另一端（队头）删除。
- $q=(k_0\ k_1\ k_2 \ldots k_{n-1})$



队列

- 允许进行删除的这一端叫队列的头。
- 允许进行插入的这一端叫队列的尾。
- 当队列中没有任何元素时，称为空队列。
- 队列的插入操作通常称为进队列或入队列。
- 队列的删除操作通常称为退队列或出队列。

队列的主要操作

- 入队列(enQueue)
- 出队列(deQueue)
- 取队首元素(getFront)
- 判断队列是否为空(isEmpty)

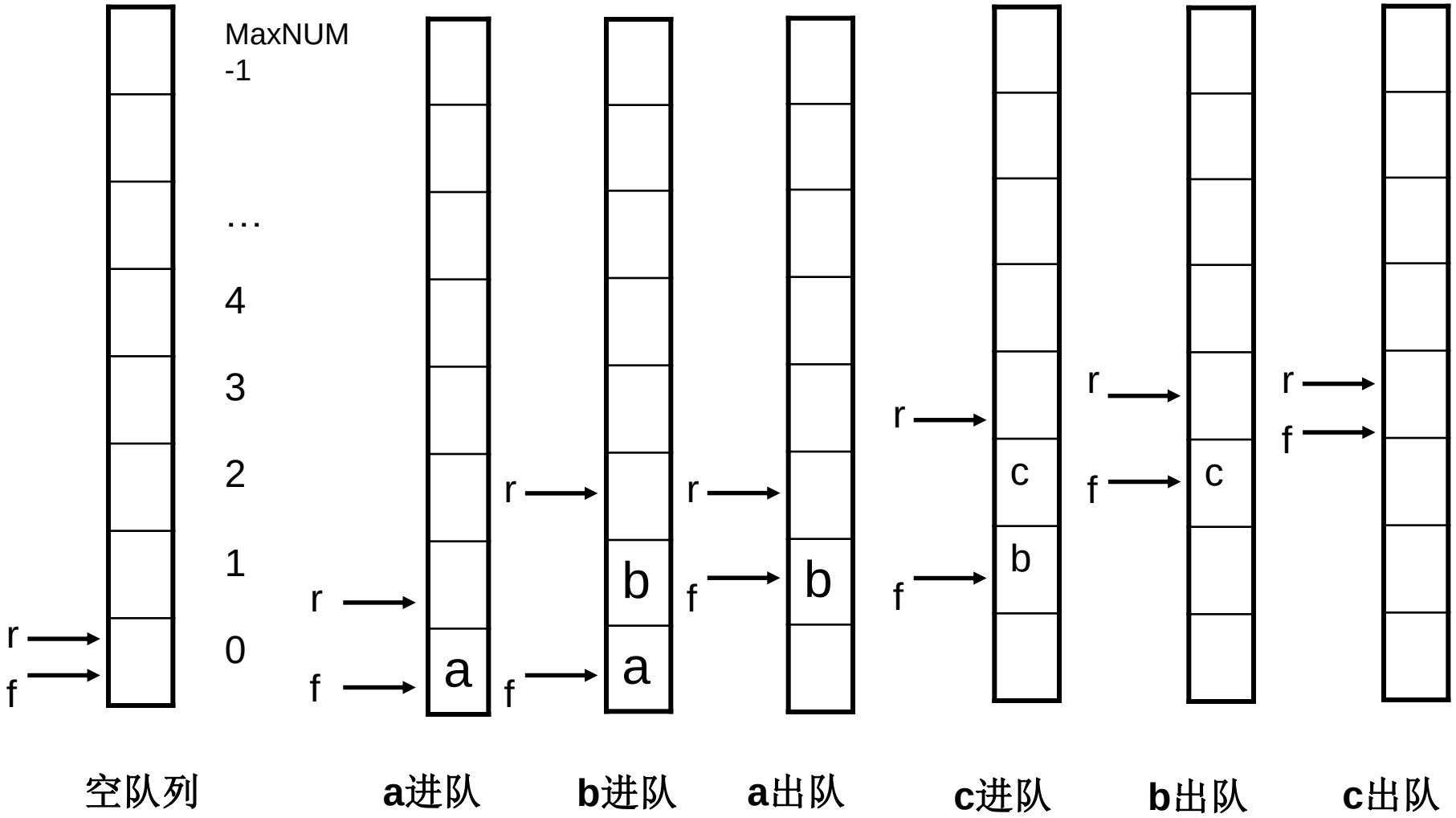
队列 – 内容提要

- 定义和操作
- 队列的实现
- 队列的应用

队列的实现

- 顺序队列
  - 关键是如何防止假溢出
- 链式队列
  - 用单链表方式存储，队列中每个元素对应链表中的一个节点

顺序队列示意



队列的顺序表示

```
① #define MAXNUM 100        /*队列中最大元素个数*/
② struct  SeqQueue
③ {
④       DataType *elem;     // DataType elem[MAXNUM];
⑤       int f, r;
⑥ };
⑦ typedef  struct SeqQueue  *PSeqQueue;

⑧ PSeqQueue pq;
```

- 要分配一块连续的存储区域来存放队列里的元素
- 用两个变量分别指示当前队列的头和尾元素的位置
  - 头变量：要删除的位置-队头元素所在的位置
  - 尾变量：要插入的位置-队尾元素所在位置的下一个位置

队列的顺序表示

```
① #define MAXNUM 100        /*队列中最大元素个数*/
② struct  SeqQueue
③ {
④       DataType *elem;       // DataType elem[MAXNUM];
⑤       int f, r;
⑥ };
⑦ typedef  struct SeqQueue  *PSeqQueue;

⑧ PSeqQueue pq;
```

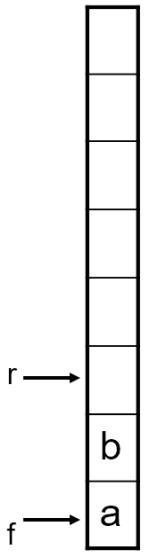
初始时：pq->f=pq->r=0

当前队列头部的元素：pq->elem[pq->f]

空队列判断：pq->r==pq->f **==0？？？**

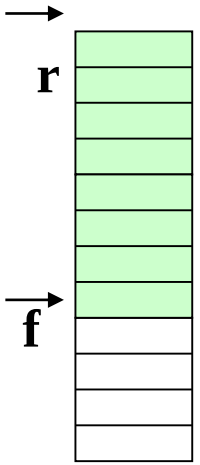
当前队列中元素的个数: (pq->r) - (pq->f)

插入元素：　pq->elem[pq->r]=key, pq->r +=1;



普通顺序队列的缺陷

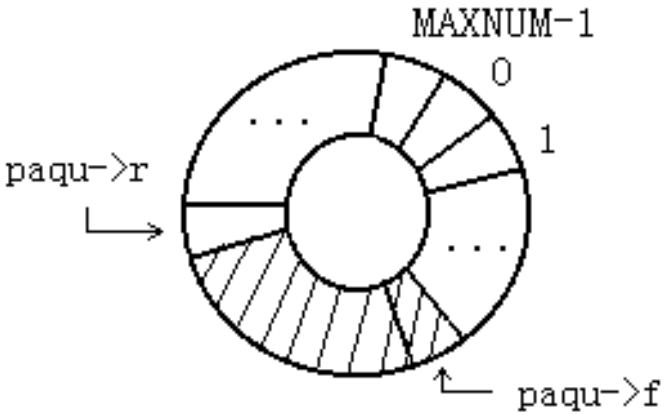
- 队列溢出：
  - 当队列满时，再做进队列操作，这种现象称为上溢
  - 当队列空时，再做删除操作，这种现象称为下溢
- 插入
  - pq->elem[pq->r] = x
  - pq->r = pq->r+1　先插入，后增加尾变量
- 删除
  - pq->f = pq->f+1
- 当pq->r ==MAXNUM
  - 再做插入就会产生溢出，而实际上这时队列的前端还有许多空的可用的位置，这种现象称为假溢出。

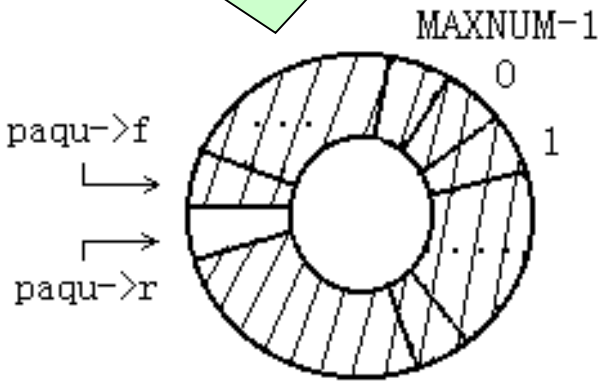


循环队列

- 解决方法：
  - 把数组paqu->q[MAXNUM]从逻辑上看成一个环
- 循环队列判空满条件：
  - 附设一变量，头尾相碰时判断头追上尾还是尾追上头
  - 少用一个空间，当尾+1等于头时为满；当头等于尾时为空

队空条件：f = = r;
队满条件：(r+1)%MAXNUM == f





循环队列基本运算的实现

- PSeqQueue creatEmptyQueue_seq(void);
  - 创建一个空队列，返回指向空队列的指针
- int isEmptyQueue_seq(PSeqQueue paqu);
  - 判断paqu所指的队列是否为空队列，空则返回1，否则返回0
- void enQueue_seq(PSeqQueue paqu，DataType x);
  - 进队列运算，插入一个值为x的元素
- void deQueue_seq(PSeqQueue paqu);
  - 出队列运算，删除一个元素
- DataType frontQueue_seq(PSeqQueue paqu);
  - 当paqu所指的队列不空时，求队列头部元素的值，队列保持不变

循环队列的基本运算实现

```
① /*进队列运算，往papu所指的队列中插入一个值为x的元素 */
② void enQueue_seq( PSeqQueue paqu, DataType x )
③ {
④     if( (paqu->r + 1) % MAXNUM == paqu->f ) /* 满队列不能再插入 */
⑤         printf( "队列已经满，不能再插入！ \n" );
⑥     else
⑦     {  /* 在队尾处插入，并且修改队尾指针 */
⑧         paqu->elem[paqu->r] = x;
⑨         paqu->r = (paqu->r + 1) % MAXNUM;
⑩     }
⑪ }
```

先插入，后修改指针，和栈不同！

循环队列的基本运算实现

```
/*出队列运算,当队列不空时,从paqu所指的队列中删除一个元素*/
① void deQueue_seq( PSeqQueue paqu )
② {
③     if( paqu->f == paqu->r ) /* 空队列不能再删除*/
④         printf( "Empty Queue.\n" );
⑤     else
⑥         paqu->f = (paqu->f + 1) % MAXNUM;
⑦ }
```

修改指针后注意：%maxnum取模操作

队列的实现

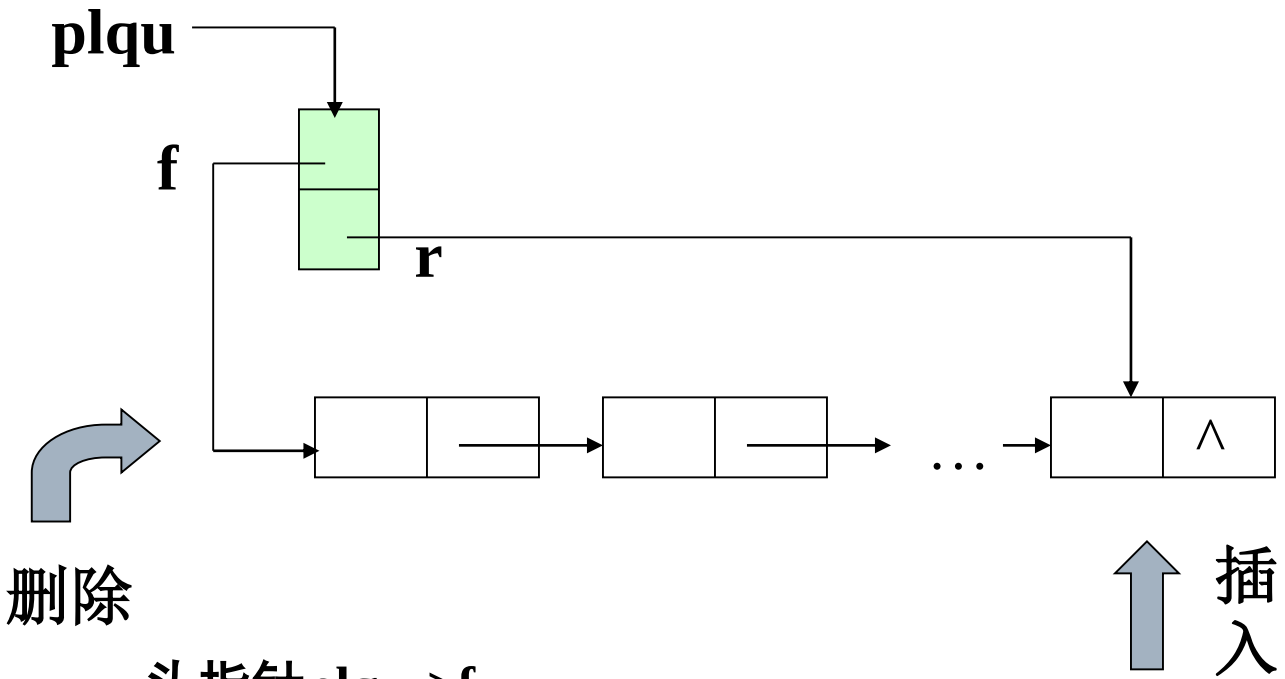
- 顺序队列
  - 关键是如何防止假溢出
- 链式队列
  - 用单链表方式存储，队列中每个元素对应链表中的一个节点

队列的链式表示

- 队列的链接表示就是用一个线性链表来表示队列，队列中的每个元素对应链表中的一个结点

```
① struct  Node                          /* 结点结构 */
② {
③         DataType  info;
④          struct Node *   link;
⑤ };
⑥ typedef struct Node *PNode;
⑦ struct  LinkQueue     /*链接队列类型定义 */
⑧ {
⑨          Pnode     f;   /*头指针*/
⑩          Pnode     r; /*尾指针*/
⑪ };
⑫ typedef struct LinkQueue, *PLinkQueue;    /*队列的指针类型 */
```

PLinkQueue plqu; /* plqu是指向链接队列的一个指针变量 */



头指针plqu->f

尾指针plqu->r

队列为空plqu->f = plqu->r = NULL

链表的头对应队列头，链表的尾对应队列的尾

链式队列中基本运算的实现

- PSeqQueue creatEmptyQueue_link(void);
  - 创建一个空队列，返回指向空队列的指针
- int isEmptyQueue_link(PLinkQueue plqu);
  - 判断plqu所指的队列是否为空队列，空返回1，否则返回0
- void enQueue_link(PLinkQueue plqu，DataType x);
  - 进队列运算，表示往plqu所指的队列中插入一个值为x的元素
- void deQueue_link(PLinkQueue plqu);
  - 出队列运算，表示从plqu所指的队列中删除一个元素
- DataType frontQueue_link(PLinkQueue plqu);
  - 当plqu所指的队列不空时，求队列头部元素的值，队列保持不变

链式队列基本运算-插入

```
① void enQueue_link( PLinkQueue plqu, DataType x )
② {
③     PNode p = (PNode)malloc( sizeof( struct Node ) );    /*申请结点*/
④     if ( p == NULL )  return;
⑤      p->info = x;
⑥      p->link = NULL;                    /* 最后一个元素，其无后继 */

⑦      if (plqu->f == NULL)               /* 原来为空队列 */
⑧      {
⑨          plqu->f = p;          plqu->r = p;          /* 头尾指针皆指向p */
⑩      }
⑪      else
⑫      {
⑬          plqu->r->link = p;          plqu->r = p;
⑭       }
⑮  }
```

链式队列基本运算-删除

```
① void deQueue_link( PLinkQueue plqu )
② {
③     PNode p;
④     /*首先判断队列是否为空，空队列不能删除*/
⑤    if( plqu->f == NULL ) printf( "Empty queue.\n " );
⑥     else
⑦     {
⑧         /* 删除头指针指向的结点，并且修改头指针 */
⑨         p = plqu->f;
⑩         plqu->f = plqu->f->link;
⑪         free(p);
⑫     }
⑬ }
```

```
else if (plqu->f==plqu->r){
    p=plqu->f;
    plqu->f = NULL;
    plqu->r = NULL;
    free(p)
}
```

如果删除前只有一个节点？

顺序队列与链式队列的比较

- 顺序队列
  - 固定的存储空间
  - 方便访问队列内部元素
- 链式队列
  - 可以满足大小无法估计的情况
  - 访问队列内部元素不方便
  - 链式队列为何不用双链表实现？

队列 – 内容提要

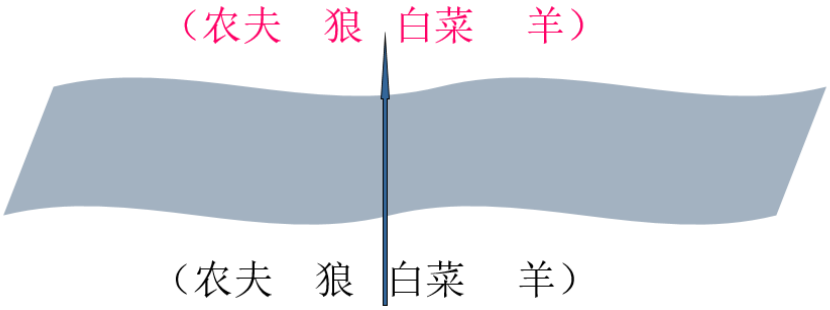
- 定义和操作
- 队列的实现
- 队列的应用

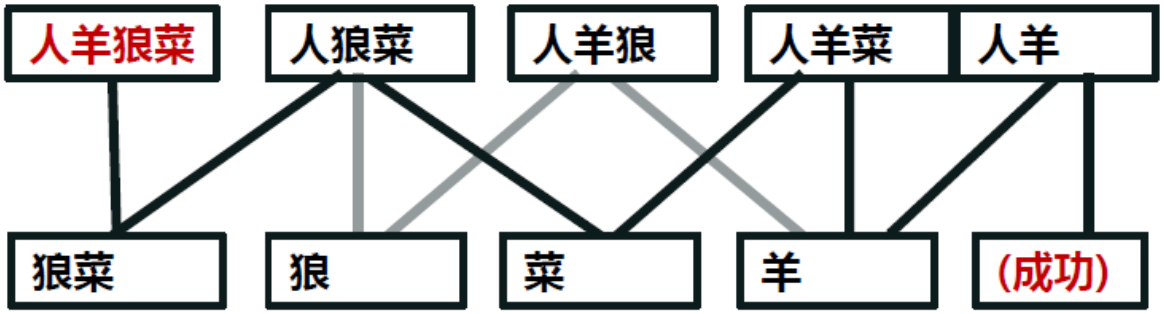
队列的应用

- 只要满足先来先服务特性的应用均可采用队列作为其数据组织方式或中间数据结构
- 调度或缓冲
  - 消息缓冲器
  - 邮件缓冲器
  - 操作系统的各种管理任务
  - 计算机的硬设备之间的通信也需要队列作为数据缓冲
- 宽度优先搜索

队列的应用—农夫过河问题

- 农夫过河问题
  - 只有人能撑船，船上只有两个位置（包括人）
  - 狼羊、羊菜不能在没人时共处





数据抽象

- 每个角色的位置进行描述
  - 农夫、狼、菜和羊，四个目标各用一位（bit）（假定按照农夫、狼、白菜、羊次序），目标在起始岸位置：0，目标岸：1

0	0	0	0

  - 如0101 表示农夫、白菜在起始岸，而狼、羊在目标岸（此状态为不安全状态）

0	1	0	1

数据的表示

- 用整数status 表示上述四位二进制描述的状态
  - 整数0x08 表示的状态

1	0	0	0

  - 整数0x0F 表示的状态

1	1	1	1

- 如何从上述状态中得到每个角色所在的位置？
  - 设计一个状态判断函数
  - 函数返回值为真（1），表示所考察人或物在目标岸
  - 否则，表示所考察人或物在起始岸

角色位置判断函数

```
bool farmer(int status)
{  return ((status & 0x08)  != 0);     }

bool wolf(int status)
{  return ((status & 0x04)  != 0);     }

bool cabbage(int status)
{  return ((status & 0x02)  != 0);     }

bool goat(int status)
{  return ((status & 0x01)  != 0);     }
```

人	狼	菜	羊
1	x	x	x
x	1	x	x
x	x	1	x
x	x	x	1

安全状态判断

```
① bool safe(int status)                    // 返回 true:安全，false:不安全
② {
③     if ((goat(status) == cabbage(status)) &&  (goat(status) != farmer(status)))
④             return(false);               // 羊吃白菜
⑤     if ((goat(status) == wolf(status)) &&  (goat(status) != farmer(status)))
⑥             return(false);               // 狼吃羊
⑦     return(true);                                  // 其它状态为安全
⑧ }
```

人	狼	菜	羊
0	1	0	1

算法抽象

- 问题变为：
  - 从状态0000（整数0）出发，
  - 寻找全部由安全状态构成的状态序列，
  - 以状态1111（整数15）为最终目标。
  - 状态序列中每个状态都可以从前一状态通过农夫（可以带一样东西）划船过河的动作到达。
  - 序列中不能出现重复状态

算法分析

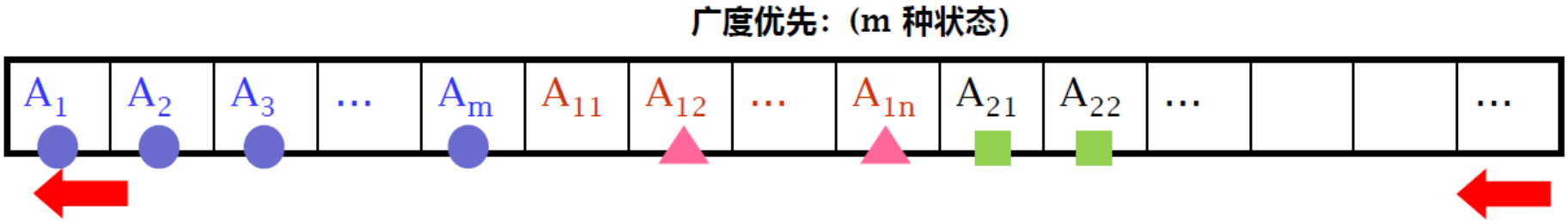
- 该问题的求解可以使用试探法，每一步都搜索当前状态下所有可能的选择，对合适的选择再考虑下一步的各种方案。
- 两种不同的搜索策略：

  广度优先搜索：搜索该步的所有可能状态，再进一步考虑后面的各种情况；（队列应用）

  深度优先搜索：沿某一状态走下去，不行再回头。（栈应用）

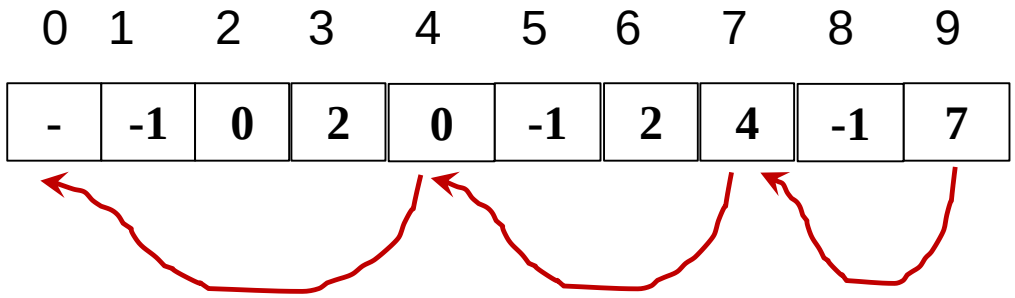
广度优先搜索算法分析

- 采用队列做辅助结构，把本步可以到达的所有状态都放在队列中
- 从队列中顺序取出状态，对其处理，处理过程中再把下一步可以到达的状态放在队列中
- 由于队列的操作按照先进先出原则，因此只有前一步的所有情况都处理完后才能进入下一步。



算法设计

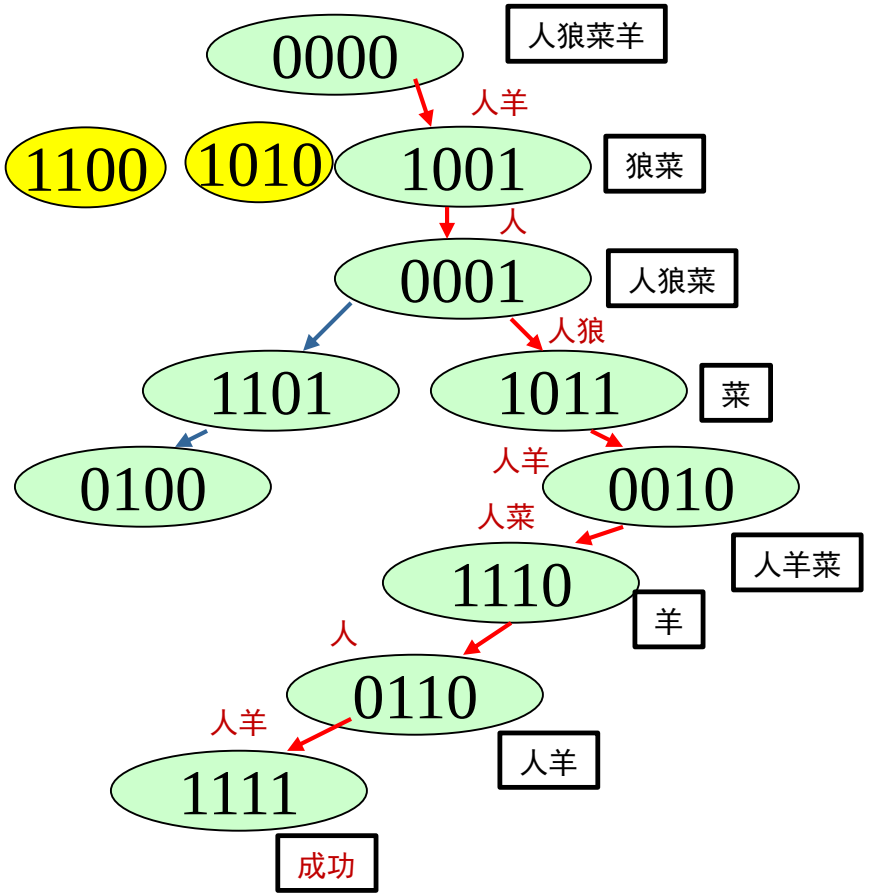
- 还需要定义一个数据结构，记录已被访问过的各个状态，以及已被发现的能够到达当前这个状态的路径
  - 用顺序表**route** 的第***i***个元素记录状态***i***是否已被访问过
  - 若**route[i]** 已被访问过，则记入前驱状态值；**-1**表示未被访问
  - **route** 的大小（长度）为**16**



算法实现

- moveTo整数队列，元素为可以安全到达的中间状态。
- route顺序表，记录各个状态被访问过的情况（大小为16数组）：
  - -1表示未被访问，否则记录前驱状态值的下标。
  - 最后，（route[15]为非负值时）利用route建立正确的状态路径。
- 计算结果

  route顺序表:15, 6, 14, 2, 11, 1, 9, 0



教学网提供代码：农夫过河问题

练习

- 将1，2，3，……, n,顺序入栈，其输出序列为p1, p2, p3,……,pn, 若p1=n, 则pi为__?

- 假设以数组A[m]存放循环队列的元素，其头指针是front，当前队列有k个元素，则队列的尾指针为_____

- 设栈S和队列Q的初始状态为空，元素a、b、c、d、e、f依次通过栈S，一个元素出栈后即进入队列Q。若这6个元素出队列的顺序是b、d、c、f、e、a，请写出它们进栈出栈的顺序。

补充内容：运算符与表达式

- C语言中的运算符（**Operator**）规定了对操作数的处理规则：
  - 算术运算符：+，-，*，/，%，++，--
  - 关系运算符：>，<，>=，<=，==，!=
  - 逻辑运算符：!，&&，||
  - 位运算符：>>，<<，&，|，^，~
  - 赋值运算符：=，及其扩展赋值运算符如+=，-=，*=，/=等
  - 条件运算符：?：

逻辑运算表达式

使用 逻辑运算符号，

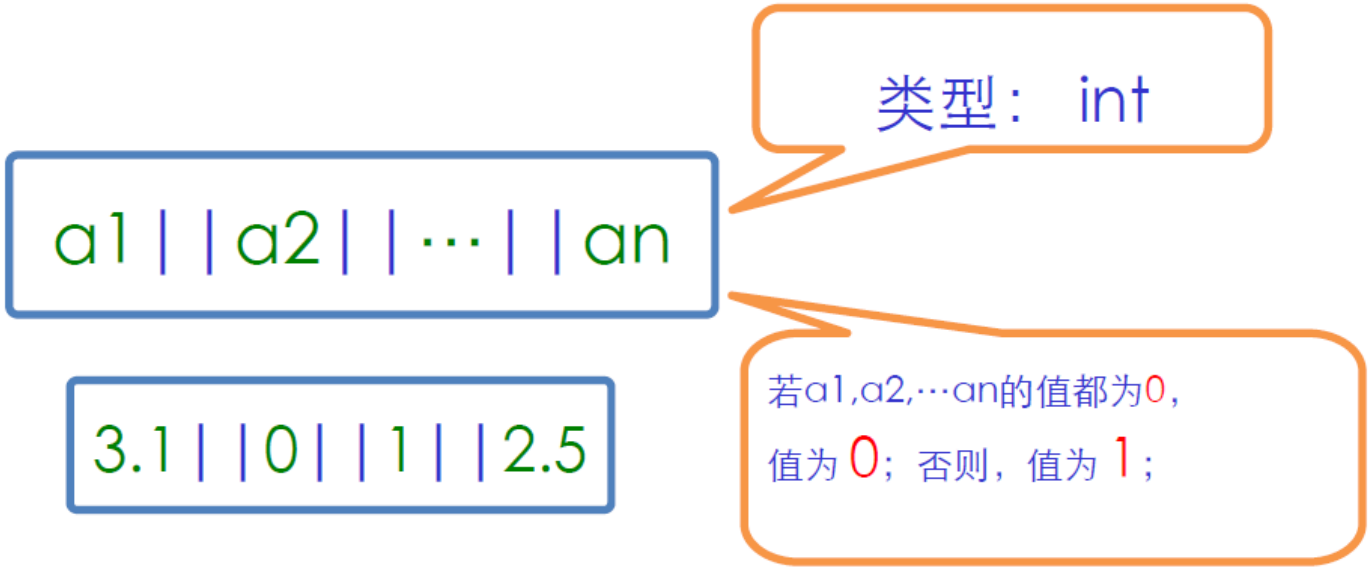
按照一定的规则，将操作数连接起来的表达式，称为 逻辑运算表达式

关于 操作数：

任何类型为int, float, 或 double的表达式，都可以作为逻辑运算表达式的操作数

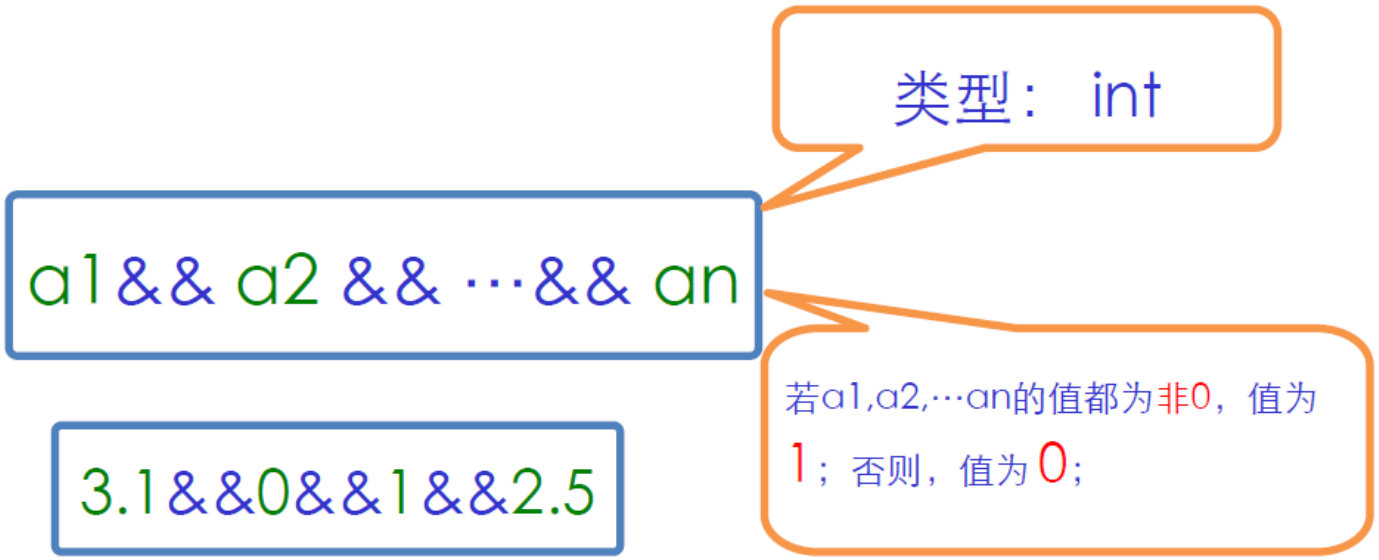
逻辑运算符号 逻辑或：||

- 假设a1, a2, …an是类型为int, float, 或double的表达式



逻辑运算符号 逻辑与：&&

- 假设a1, a2, …an是类型为int, float, 或double的表达式



例：判断某一年是否是闰年

- 程序输入
  - 一个表示年份的整数（如，2012）
- 程序输出
  - 如果这个年份是闰年，程序输出true;
  - 否则，程序输出false;

(x%400==0)||((x%4==0)&& (x%100!=0))

位运算表达式

使用 位运算符，

按照一定的规则，将操作数连接起来的表达式，称为 位运算表达式

关于 操作数：

必须是整型数据！

位运算表达式

- &(按位与)
- | (按位或)
- ^ (按位异或)
- ~ (按位非)
- <<(左位移)：将左侧操作数的二进制数值向左移动若干位（由右侧的操作数给出），移出去的位丢弃，空出的位用0填补
- >>(右位移)：将左侧操作数的二进制数值向右移动若干位（由右侧的操作数给出），移出去的位丢弃，空出的位用符号位（对有符号数）或0（对无符号数）来填补

例：异或

```
    1 1 0 0 0 1
^   0 1 1 1 1 1
-----------------
    1 0 1 1 1 0
```

C=A^B
A=C^B
B=C^A

- 设计一个算法来实现字符串逆序存储，要求不另设串存储空间；

  利用字符串的结束符'\0'，作为中间变量

  利用a=a+b;b=a-b;a=a-b;

  利用a=a^b;b=a^b;a=a^b;（异或）

位运算符号

- 无符号数：没有符号位，不管左移或右移，空出的位用0填补。

- 位移运算的实质（在不发生溢出时）
  - 左移：x<<n，相当于x* 2n
  - 右移：x>>n，相当于x/2n

位运算符号

- 有符号数：做位移运算时，符号位不参与移动
  - 左移时，空出的位用0填补；
  - 右移时，空出的位用符号位填补。(数据的补码表示)