位置相关服务（LBS）：基于网格划分的索引

2017年

第二章 基于网格划分的索引

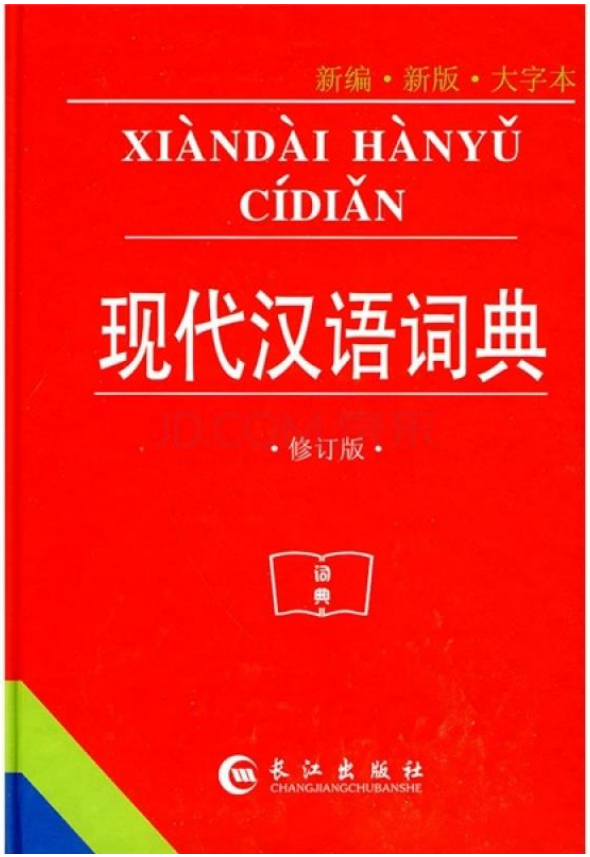
1. **索引的概念**
2. 空间索引
3. 网格索引的结构
4. 出租车位置更新算法
5. 作业

索引的类比

- 汉语字典中的汉字按页存放，一般都有汉语拼音目录（索引）、偏旁部首目录等
- 我们可以根据拼音或偏旁部首，快速查找某个字词

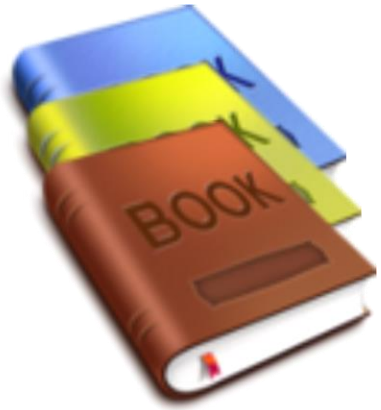
在一本书中，利用索引可以快速查找所需信息，无须阅读整本书。在一个数据集中，索引使程序无需对整个数据集进行扫描，就可以在其中找到所需数据。书中的索引是一个词语列表，其中注明了包含各个词的页码。索引也通过索引结构，快速地指向每个数据的存放位置。

索引使书籍中的记录有序化，它从逻辑上而不是物理上对记录进行排序。



索引的作用

- 一本工具书就像一个数据库，工具书的索引就像进入数据库的钥匙。
- 索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息
- 利用索引可以有效地管理大量的数据，并快速从数据中查找到特定的数据。



索引的概念

- 索引是加快索引表中数据的方法。
- 在数据库中，索引可以快速访问一条特定查询所请求的数据，无需遍历整个数据库。
- 通过使用索引可以大大提高数据检索速度，改善查询性能。

索引用于回答一个查询：索引包含了足够的信息来获得整个查询，我们根本不用去访问表。在这种情况下，索引则作为一个较小的表来使用。

索引的优缺点

索引的优点

- 大大加快数据的检索速度，这是创建索引的最主要的原因
- 加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。
- 在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

索引的优缺点

❖ 索引的缺点

- 创建索引和维护索引要耗费时间，并随着数据量的增加而增加
- 索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间
- 当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的维护速度

目　录

1 索引的概念

2 空间索引

3 网格索引的结构

4 出租车位置更新算法

5 作业

索引结构的本质：空间换时间

划分和有序组织数据

|

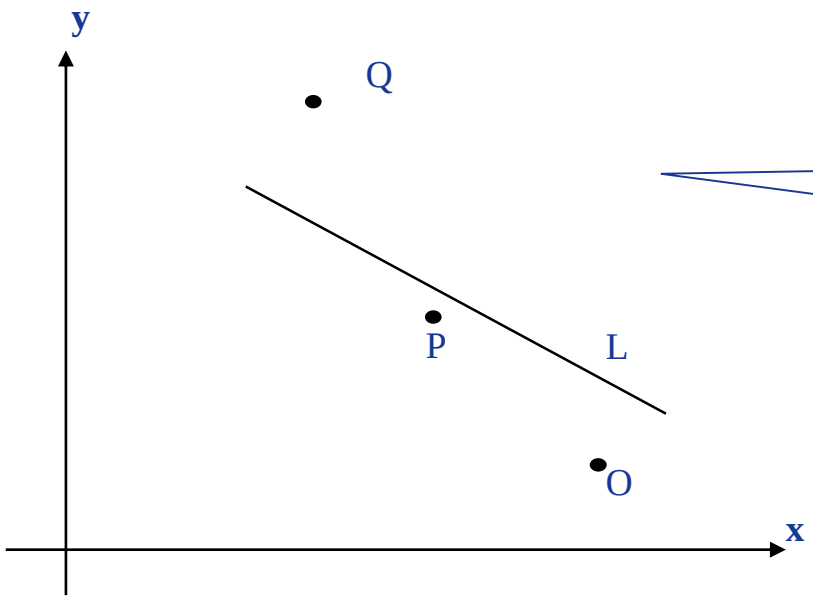
减少不必要磁盘访问

|

节省查询时间

问题引入

空间查询不能通过直接匹配得到



很难找到一种排序规则来完全表明这4个空间对象的位置关系

如图，设在一个平面上有线段（L）和点（O、P、Q）；同时假设在数据库中存放点的坐标信息以及线段的端点信息。

当处理查询“查找离L最近的点”时，是无法通过直接的匹配进行处理的。离端点最近的点（如O、Q）也许并不是期望的答案。

空间索引

一维索引技术的局限性

空间/时空数据库	空间/时空对象 非精确匹配

时空索引特性

高维数据	数据类型复杂	点、线、区域、多边形、平面、多面体 不同于普通数据库的查询
空间/时空索引	高维空间索引无法排序	不是简单的关键字匹配 相似度检索，而非精确匹配的检索 常规索引方法不能直接用于空间数据库的索引

引入空间索引技术的必要性

空间索引方法的分类

空间索引方法的分类

(1)基于最小包容块方法
（Minimum Bounding Rectangle，简称MBR）

(2)基于空间细分的方法

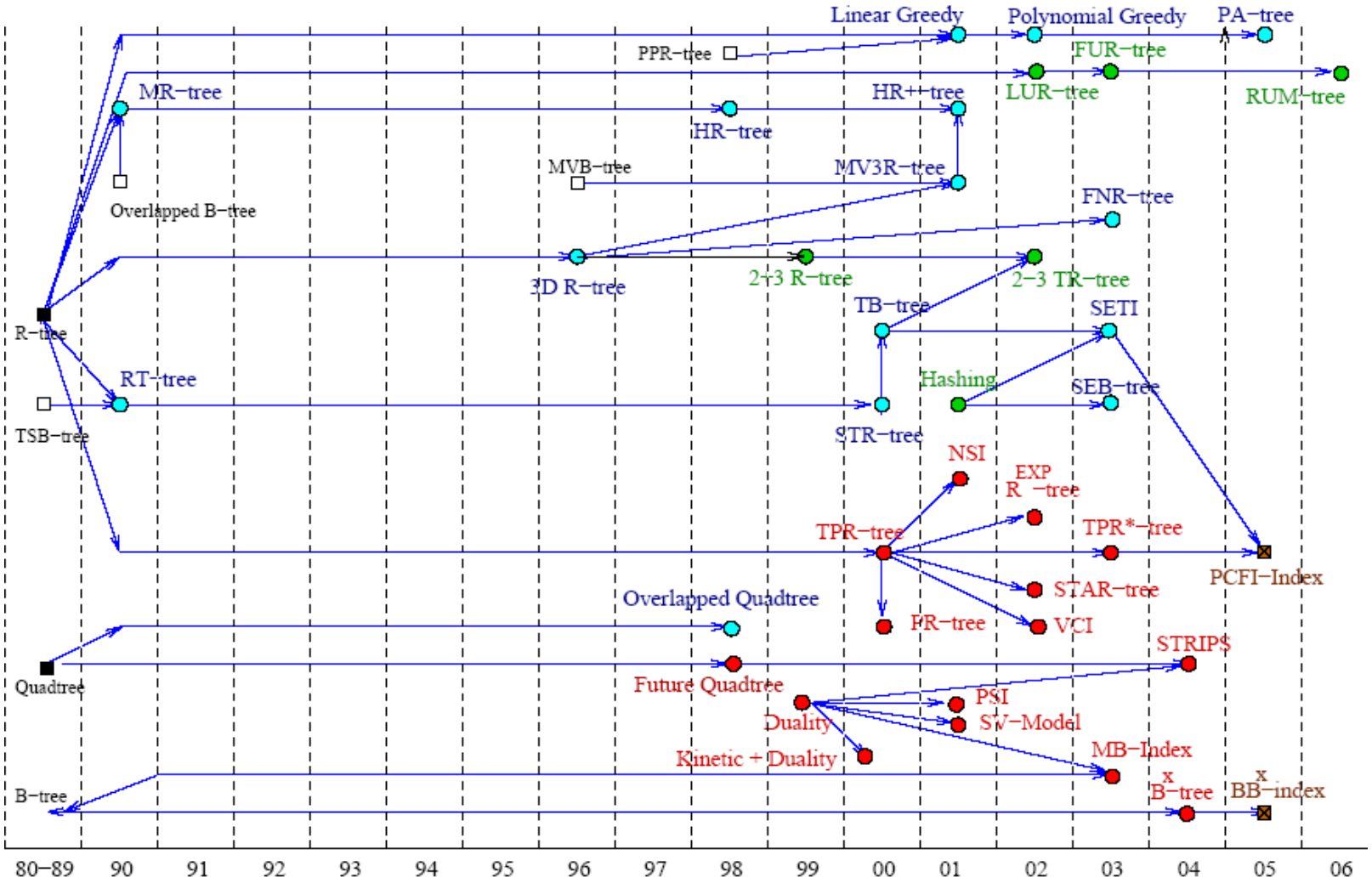
—— 将空间划分为均匀大小

—— 将空间划分为规则的不均匀大小

—— 将空间划分为随意大小

基于 MBR 的方法	基于空间细分的方法
多级 Grid files R-files PLOP hashing skd 树 GBD 树 R 树 Packed R 树 R*树 Buddy 树	Grid files EXCELL mkd 树 R+树 Cell 树 BANG 文件

空间索引方法的发展


Linear Greedy
Polynomial Greedy
PA−tree
PPR−tree
FUR−tree
LUR−tree
RUM−tree
MR−tree
HR+−tree
HR−tree
MVB−tree
MV3R−tree
Overlapped B−tree
FNR−tree
3D R−tree
2−3 R−tree
2−3 TR−tree
TB−tree
SETI
R−tree
RT−tree
Hashing
SEB−tree
TSB−tree
STR−tree
NSI
EXP
R −tree
TPR−tree
TPR*−tree
PCFI−Index
STAR−tree
Overlapped Quadtree
PR−tree
VCI
STRIPS
Quadtree
Future Quadtree
PSI
Duality
SV−Model
MB−Index
Kinetic + Duality
x
B−tree
x
BB−index
B−tree
80−89
90
91
92
93
94
95
96
97
98
99
00
01
02
03
04
05
06

本课程重点

- 基于网格划分的索引
- R-tree索引

目　录

1 索引的概念

2 空间索引

3 **网格索引的结构**

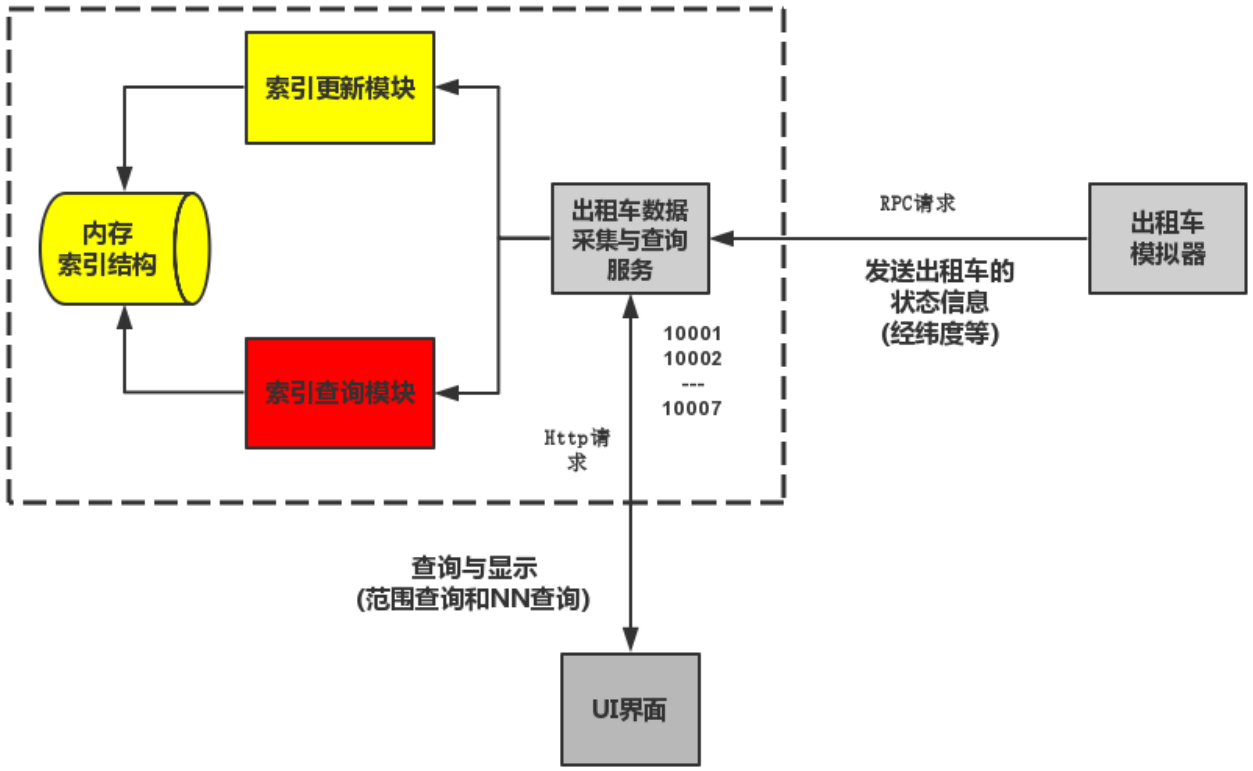
4 出租车位置更新算法

5 作业

预备知识

出租车状态信息

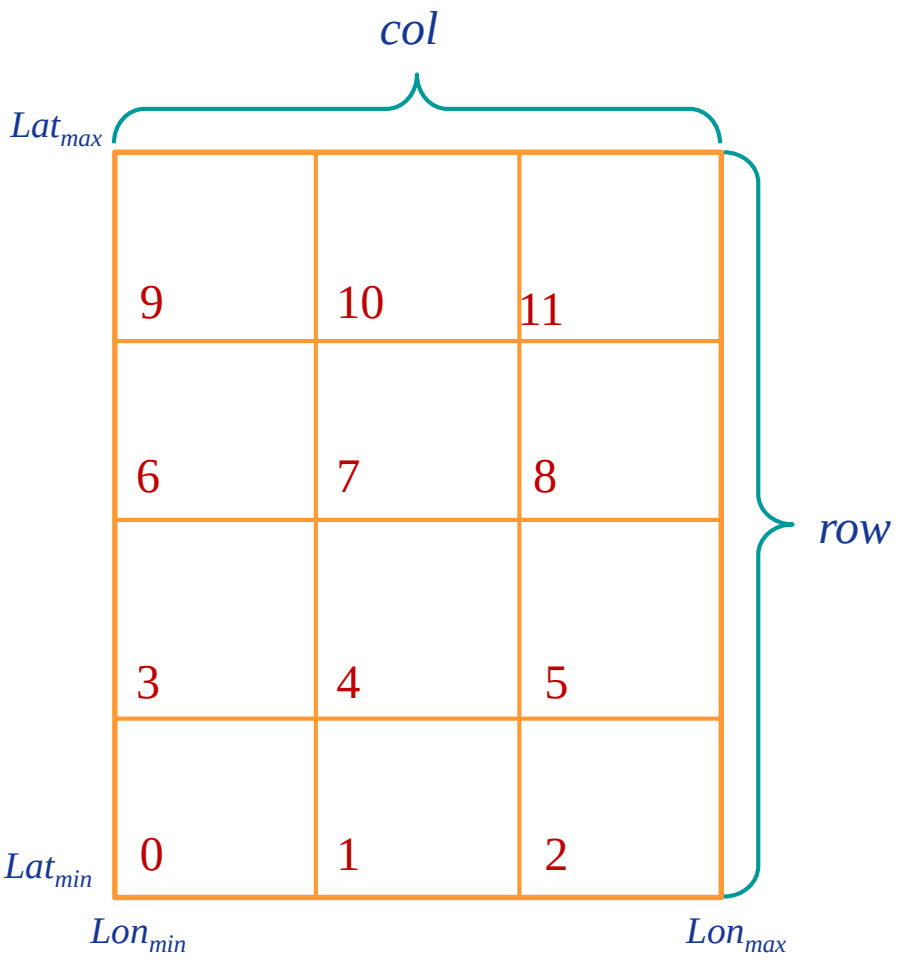
- 出租车唯一标识号
- 经纬度信息
- 时间戳
- ….



网格索引的结构

网格索引

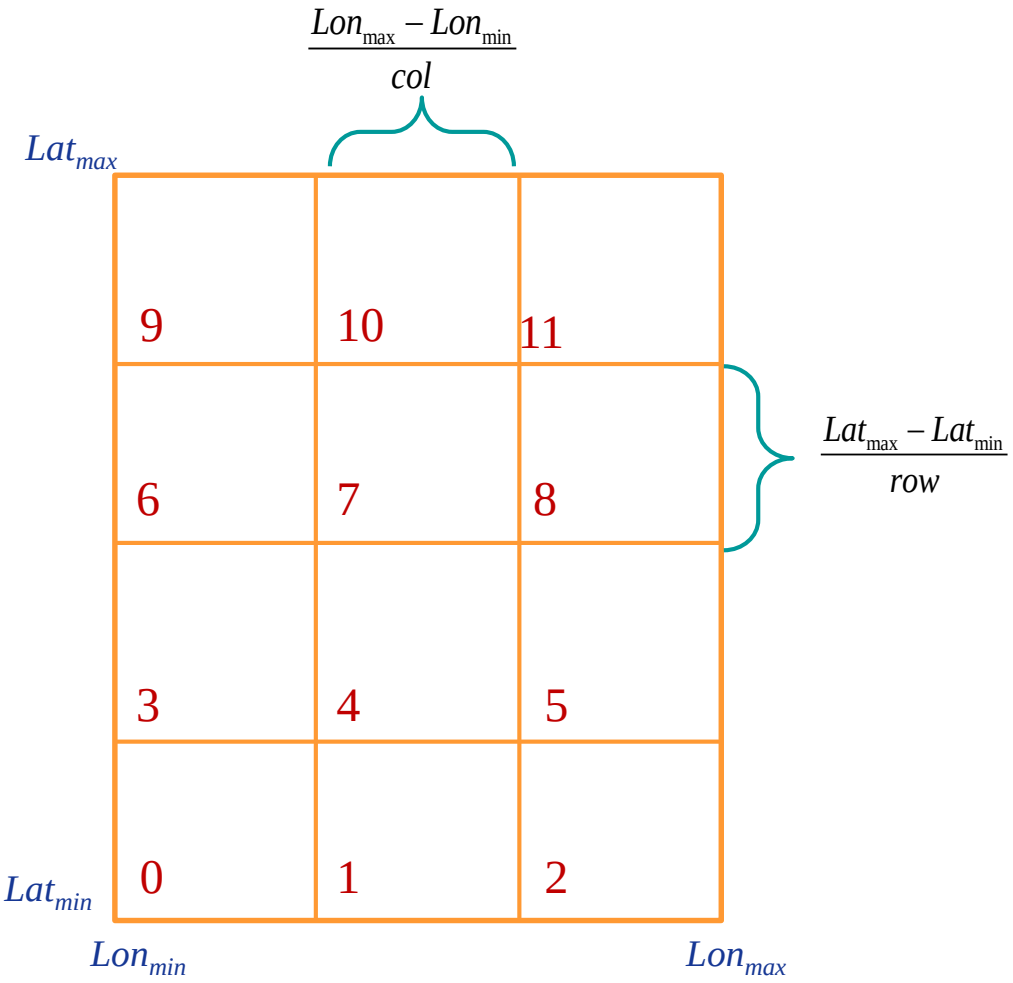
- 空间区域是一个[I_{lon}, I_{lat}]大小的空间，其中I_{lon} = [Lon_{min}, Lon_{max}]，I_{lat} = [Lat_{min}, Lat_{max}]。
- 将空间区域划分为$row \times col$个相同大小的网格，每一个小格子称为$Cell$



网格索引的结构

网格索引

- 空间区域是一个$[I_{lon}, I_{lat}]$大小的空间，其中$I_{lon} = [Lon_{min}, Lon_{max}]$，$I_{lat} = [Lat_{min}, Lat_{max}]$。
- 将空间区域划分为$row \times col$个相同大小的网格，每一个小格子称为*Cell*
- 网格的列宽$width = (Lon_{max} - Lon_{min})/col$
- 网格的行高$height = (Lat_{max} - Lat_{min})/row$

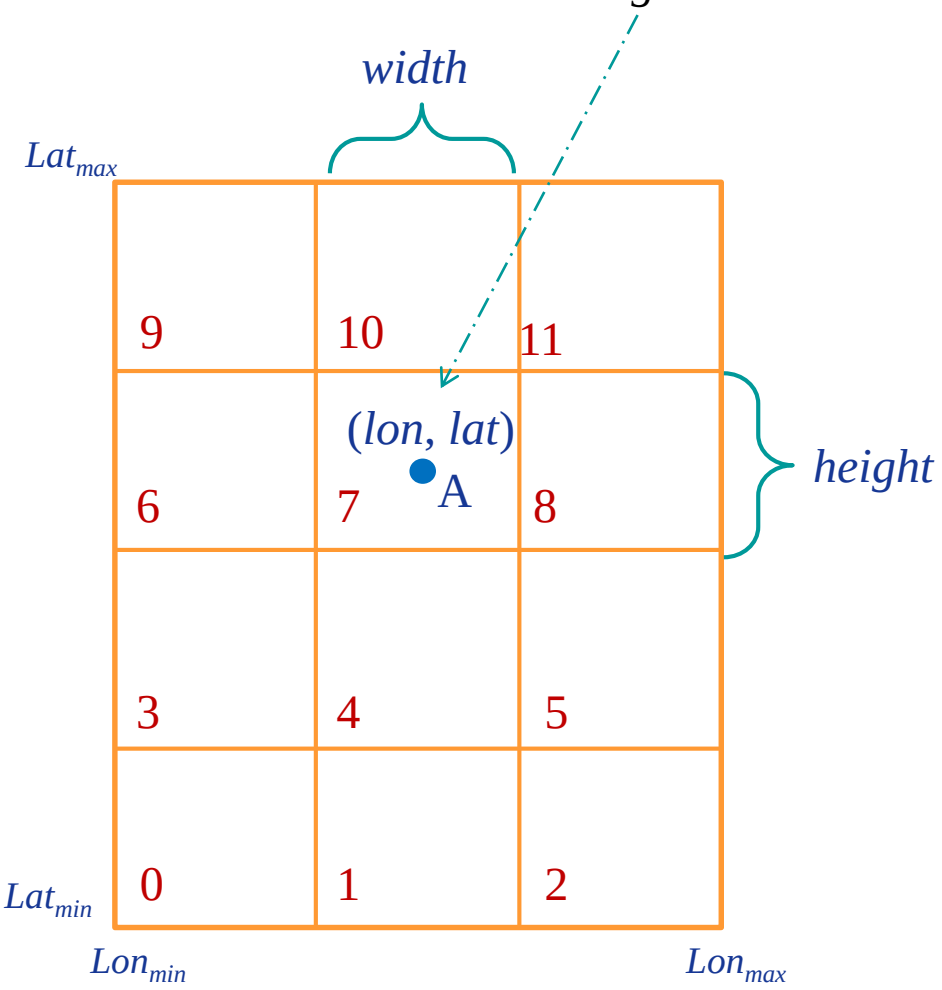


网格索引的结构

网格索引

- 空间区域是一个$[I_{lon}, I_{lat}]$大小的空间，其中$I_{lon} = [Lon_{min}, Lon_{max}]$，$I_{lat} = [Lat_{min}, Lat_{max}]$。
- 将空间区域划分为$row \times col$个相同大小的网格，每一个小格子称为*Cell*
- 网格的列宽$width = (Lon_{max} - Lon_{min})/col$
- 网格的行高$height = (Lat_{max} - Lat_{min})/row$
- $Cell\ Id = col \times (\boldsymbol{lat} - Lat_{min}) / height + (\boldsymbol{lon} - Lon_{min}) / width$

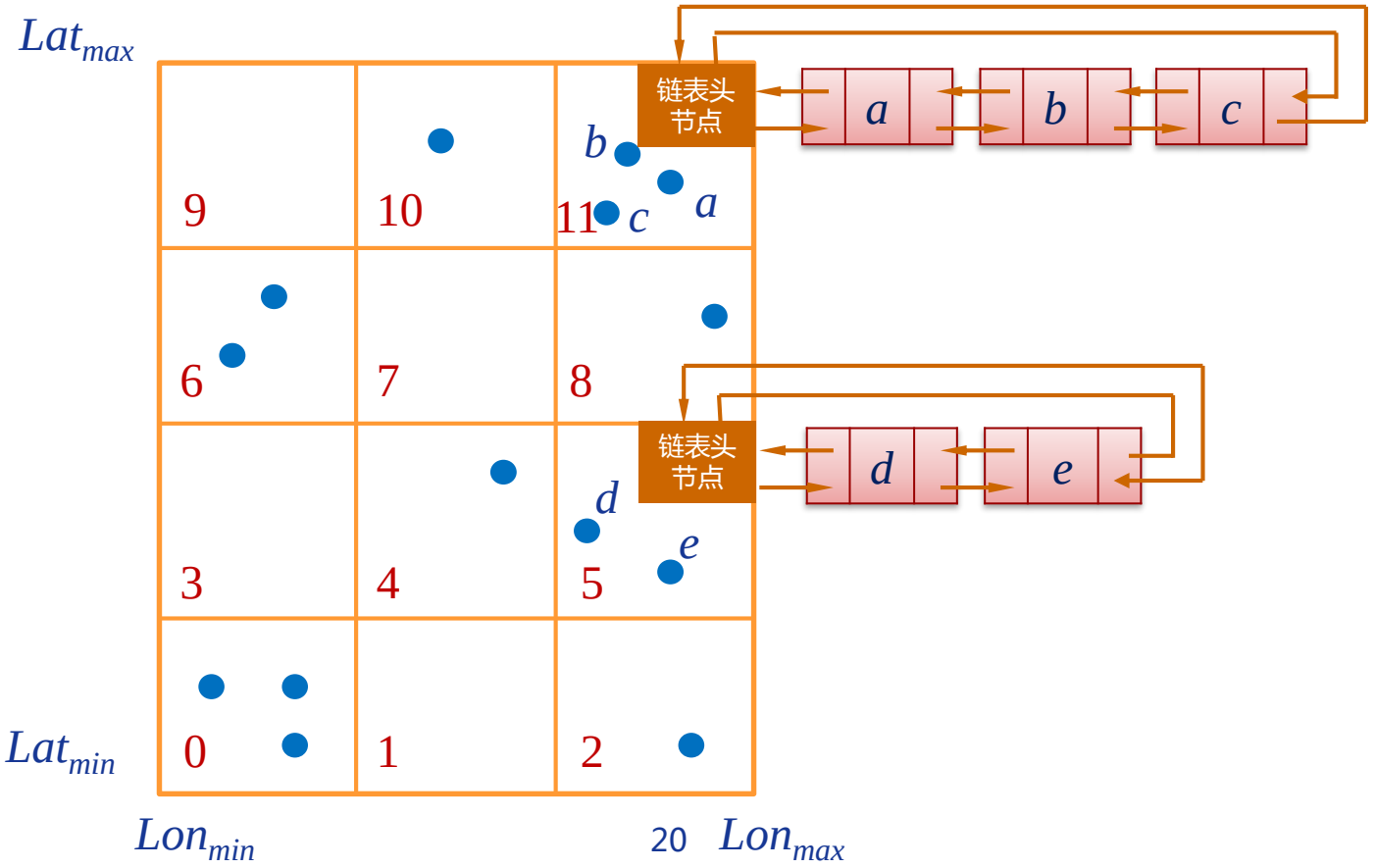
$$CellId = col \times \frac{lat - Lat_{\min}}{height} + \frac{lon - Lon_{\min}}{width}$$



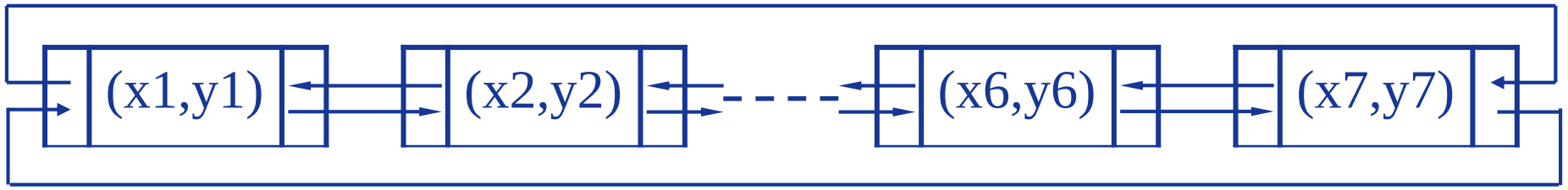
网格索引的结构

网格索引

- 每个网格为一个索引单元
- 每个网格中的出租车以双向循环链表的形式存入对应的索引单元



双向循环链表



为了表示这种既有数据又有指针的情况，引入了链表结构。

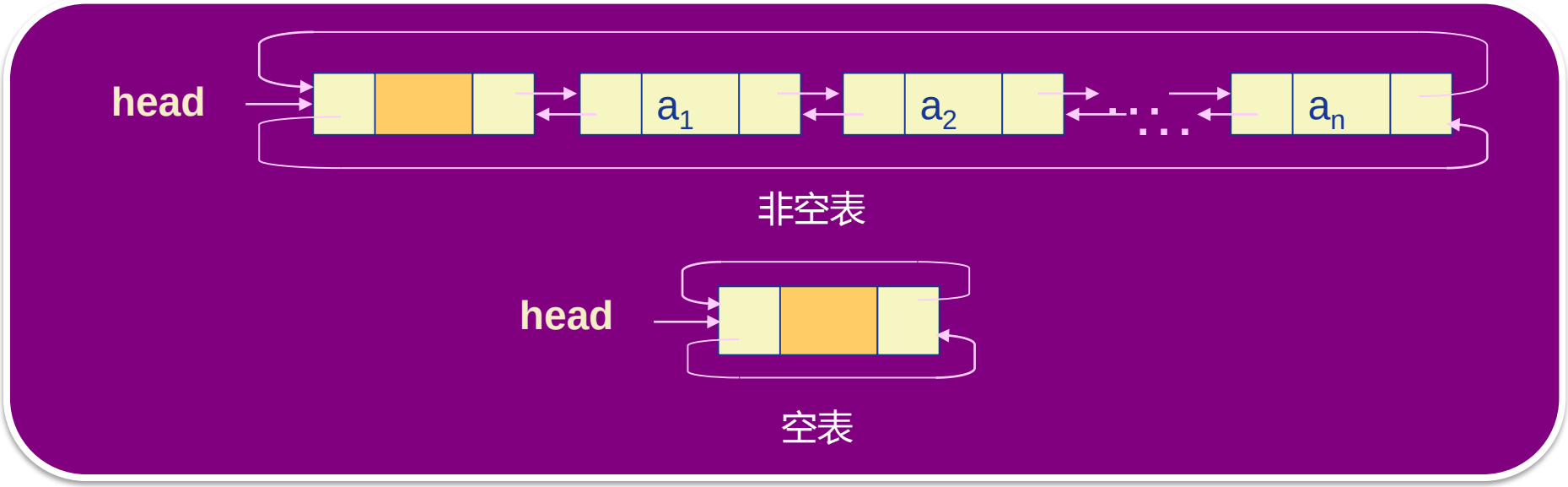
动态性体现为：

- 链表中的元素个数可以根据需要增加和减少；
- 元素的位置可以变化，即可以从某个位置删除，再插入到一个新的地方；

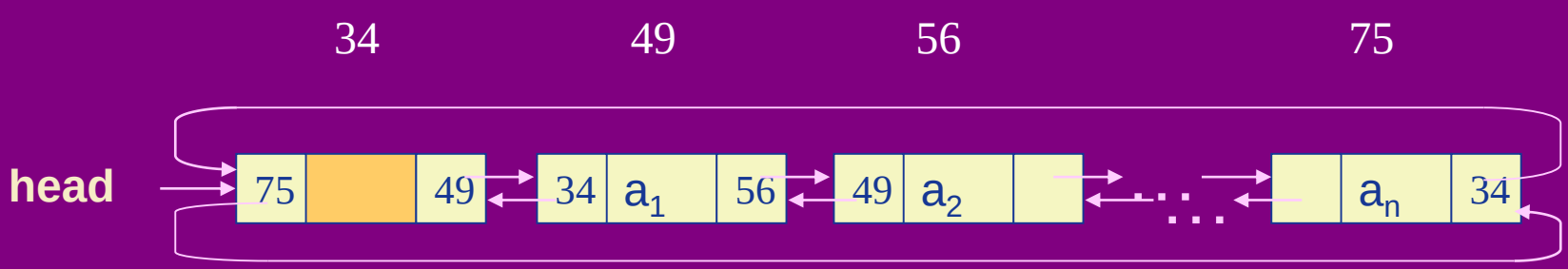
双向循环链表

- 双向循环链表的存储结构
  - 链表中每个结点有两个指针域，一个前驱指针域，指向直接前趋元素结点，另一个后继指针域，指向直接后继元素结点。

  - 双向循环链表通常采用带表头结点的循环链表形式。



双向循环链表



1、链表中的元素称为“结点”，每个结点包括两个域：数据域和指针域；

2、双向循环链表通常有一个头指针(head)，用于指向链表头；

3、结点里的后继指针是存放下一个结点的地址;

4、结点里的前驱指针是存放前一个结点的地址。

双向循环链表

双向循环链表的定义

```
typedef struct Queue {
    struct Queue *next;     // 指向下一个结点
    struct Queue *prev;     // 指向上一个节点
} Queue;
```

注意：一般来说，双向循环链表的头结点，不存储数据。

构造头结点

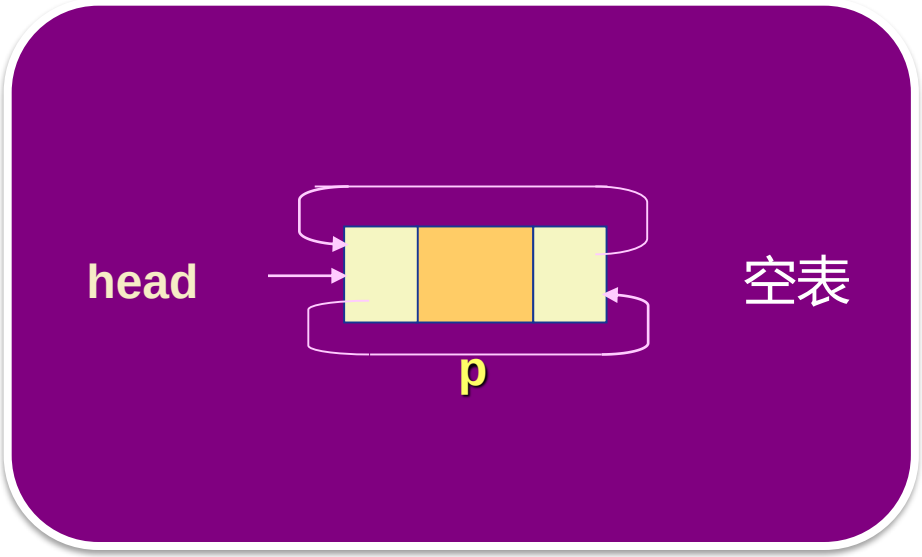
双向循环链表的头结点

算法1：CreateHead(p)

输入：无

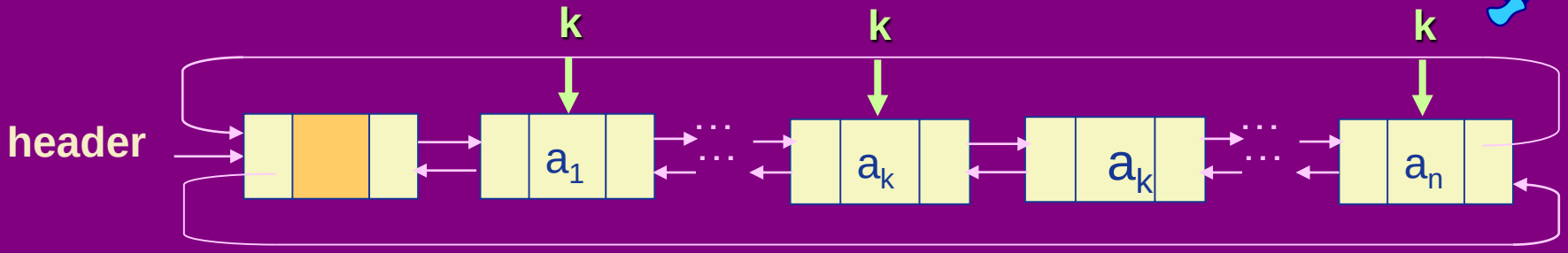
输出：双向循环链表头结点head

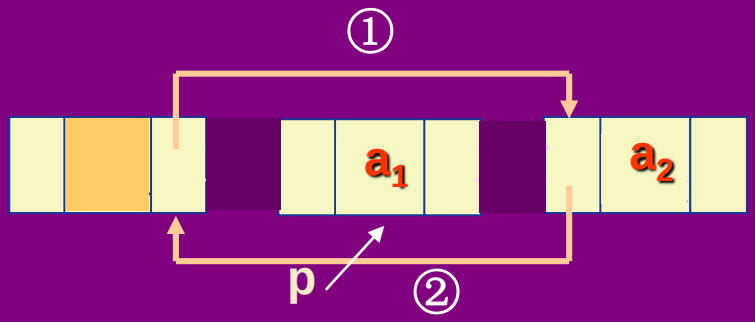
1) **p->prev = p;**
2) **p->next = p;**
3) **head = p;**



双向循环链表的删除

- 在双向循环链表中删除位置k的结点





① p->prev->next=p->next

② p->next->prev=p->prev

③ p->next=Null

④ p->prev=Null

双向循环链表的删除

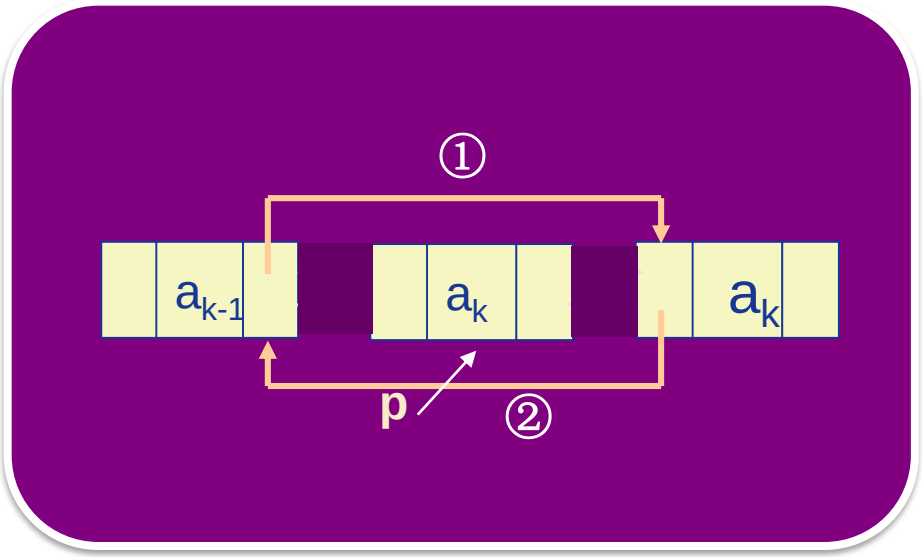
双向循环链表的节点删除

算法2：QueueRemove(p)

输入：双向循环链表指针p

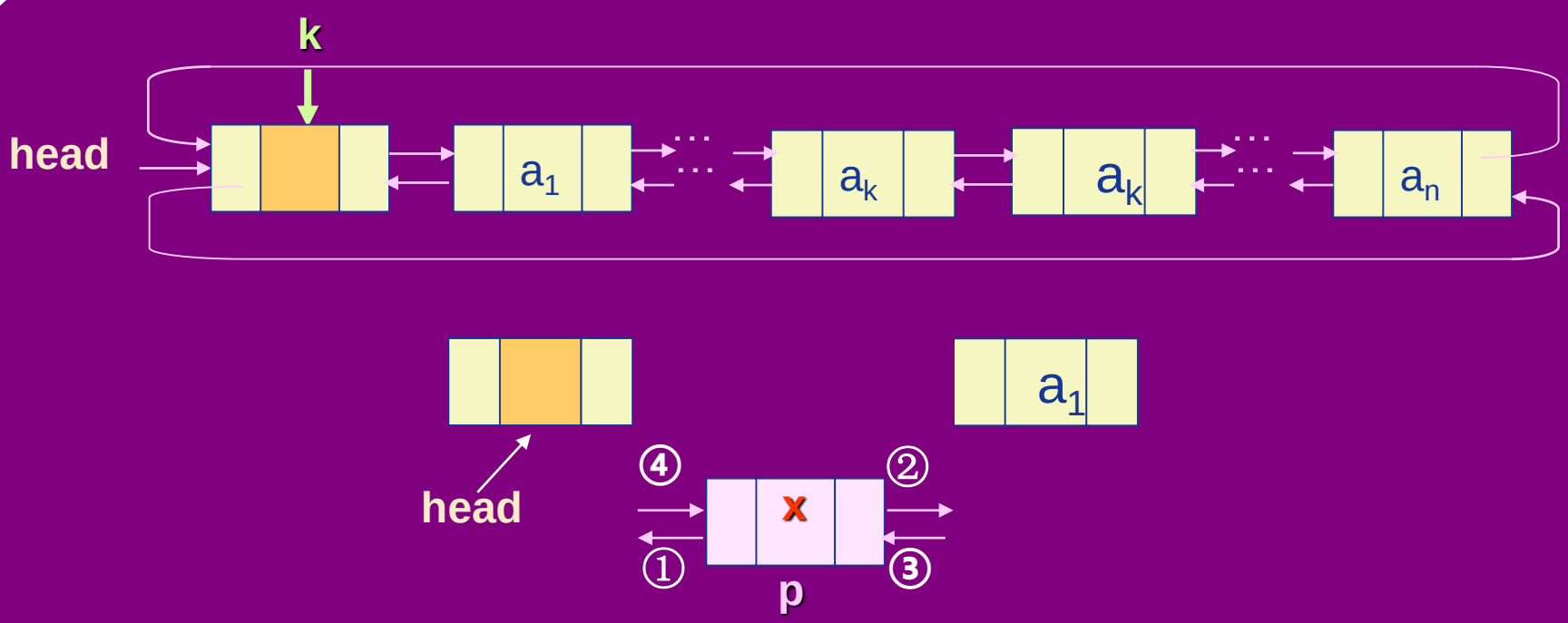
输出：无

1) **p->next->prev = p->prev;**
2) **p->prev->next = p->next;**
3) **p->next = NULL;**
4) **p->prev= NULL;**



双向循环链表的插入

- 将出租车的位置信息插入链表最前面



① p->prev=head	③ head->next->prev=p
② p->next=head->next	④ head->next=p

双向循环链表的插入

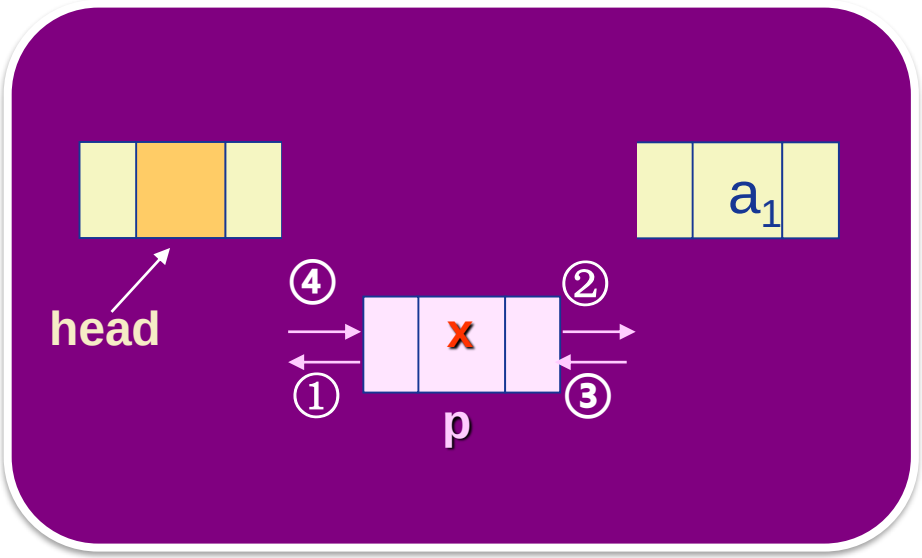
双向循环链表的节点增加

算法3：QueueInsertHead(head, p)

输入：双向循环链表的头结点指针head，待增加的节点指针p

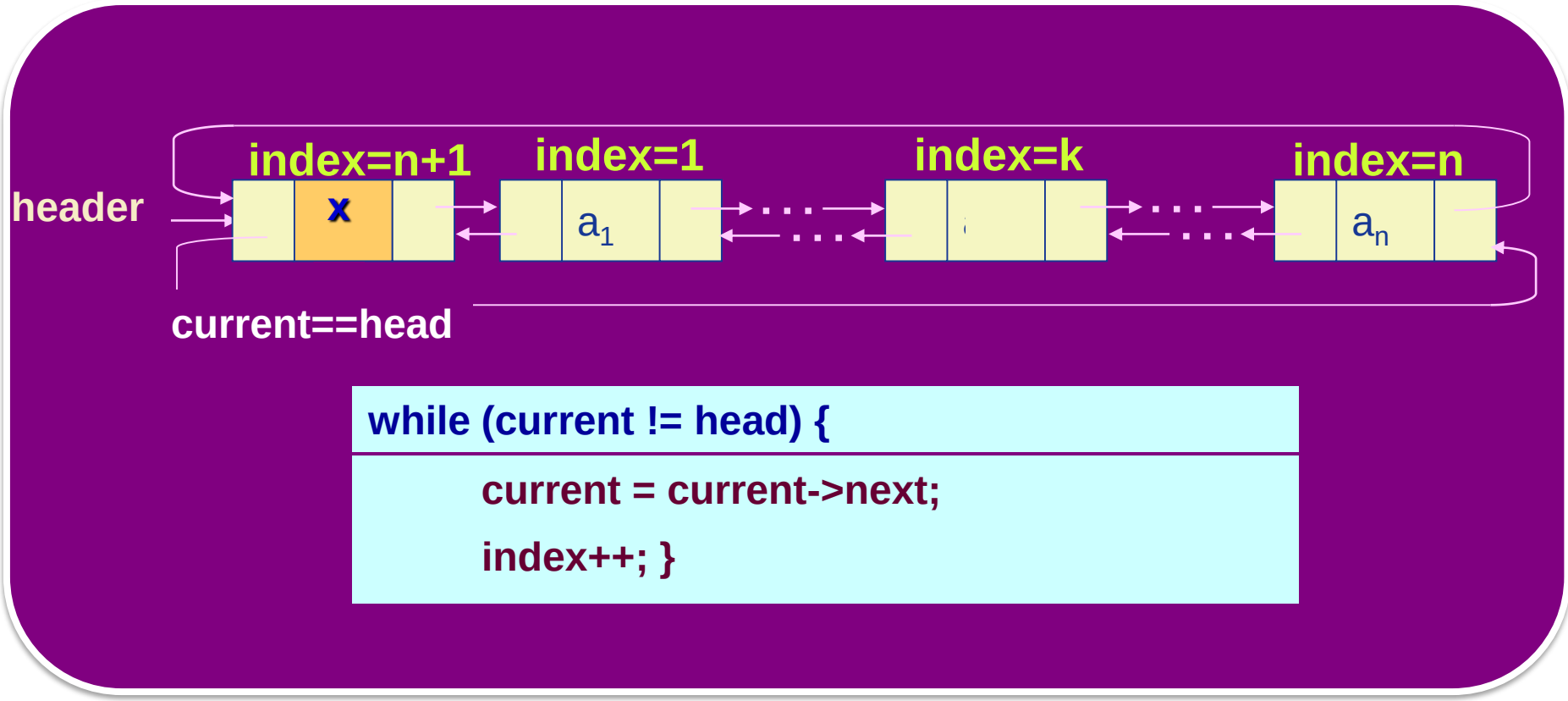
输出：无

1) **p->prev = head;**
2) **p->next = head->next;**
3) **head->next->prev = p;**
4) **head->next = p;**



双向链表的遍历

- 基本思想：从位置1结点开始依次向后扫描，直到最后一个元素。




```
while (current != head) {
        current = current->next;
        index++; }
```

双向链表的遍历

双向循环链表的遍历

算法4：QueueVisit (head)

输入：双向循环链表的头结点指针head

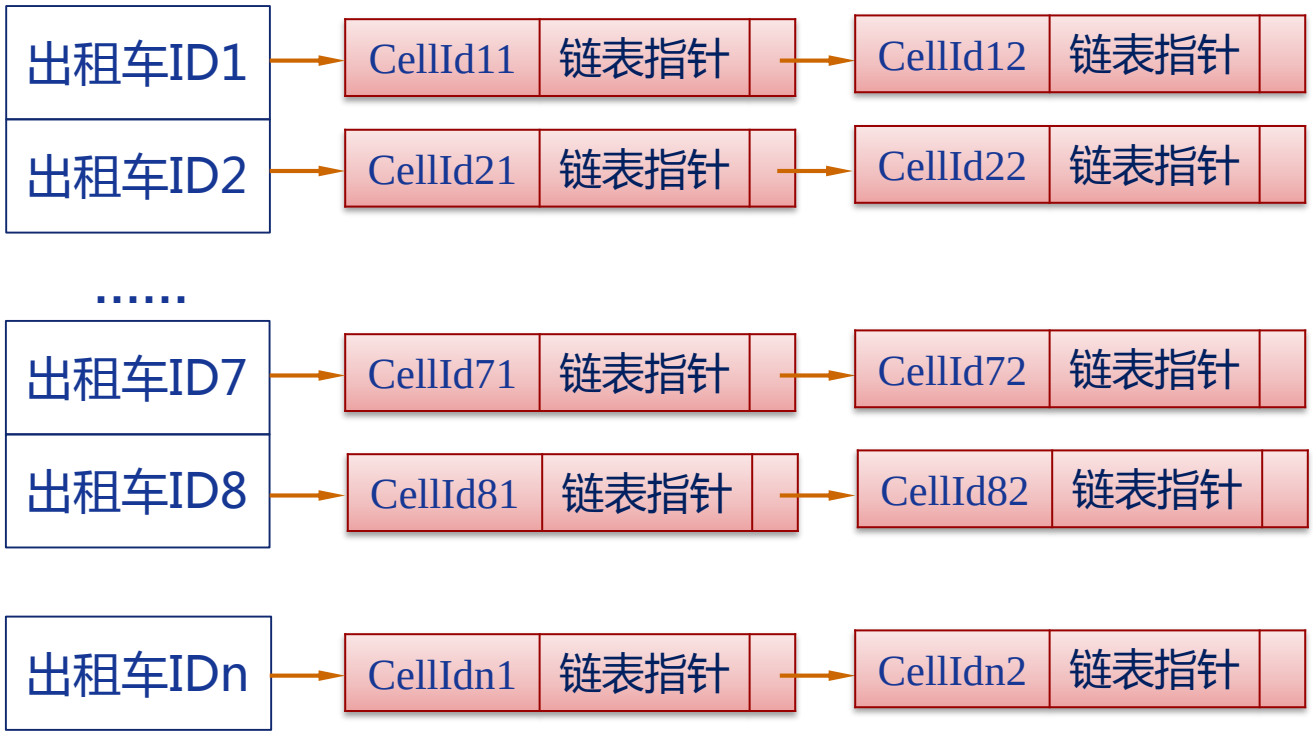
输出：无

```
1)  current = head->next;
2)  while (current != head) {
3)      Visit(current);
4)      current = current->next;
5)  }
```

网格索引的结构

网格索引

- 通过Hash表来定位当前出租车所在的Cell编号(链地址法)。



为什么引入哈希表

- 车牌号“京B12345”的出租车位置更新时，如何判断所在网格，以及链表位置？
  - 在网格的链表中逐个比较查找

基于比较的查找太慢？如何实现快速查找？

哈希表：按照内容查找，不用比较，立即取得所查找的记录。

通过记录的存放位置和关键字之间的对应关系，方便地根据记录的关键字检索到对应记录的信息。

什么是哈希表

- 哈希表又称散列表
- 哈希表存储的基本思想是：
  - 以数据表中的每个记录的关键字k为自变量；
  - 通过哈希函数H(k)计算出函数值。

> 哈希函数：记录的存放位置与关键字之间的对应关系。
>
> 即：记录的存储位置loc = H(k)

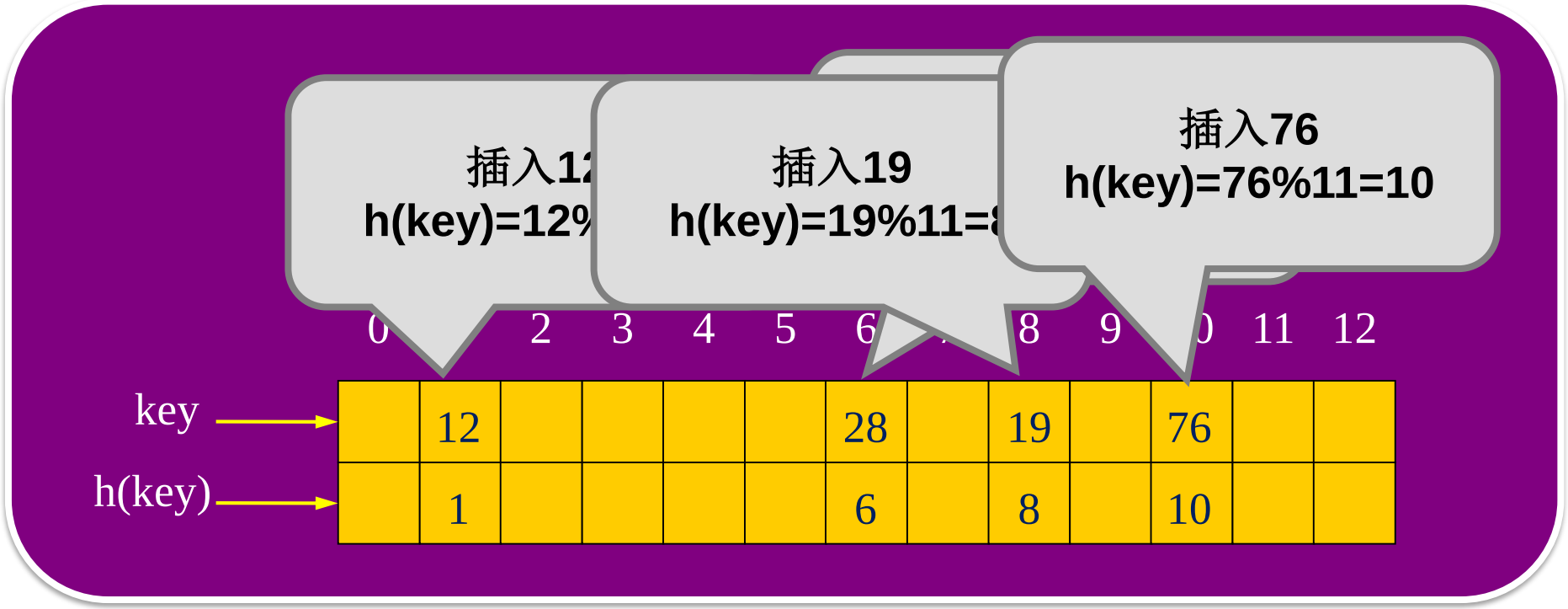
  - 这个函数值可以理解为一块连续存储空间（即数组空间）的单元地址（即下标），将该记录存储到这个单元中。
  - 按这种方法建立的表称为哈希表或散列表。

实现立即查找！

哈希表

例如：

- 要将关键字值序列（ 12，19，28，76 ），存储到编号为0到12的表长为13的哈希表中。
- 计算存储地址的哈希函数可取除11的取余数算法H(k)=k % 11。则构造好的哈希表如图所示。

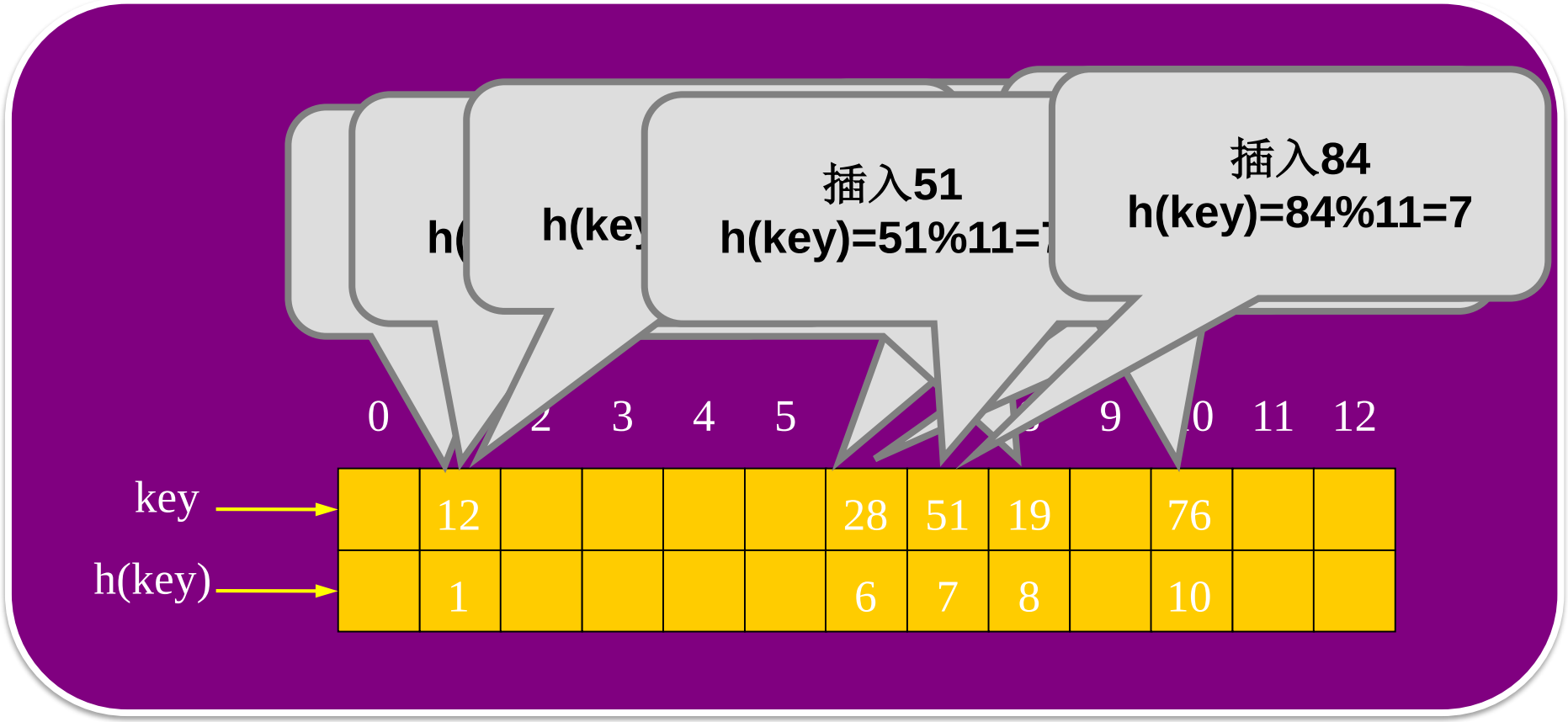


哈希表

- 理想情况下，哈希函数在关键字和地址之间建立了一个一一对应关系，从而使得查找只需一次计算即可完成。
- 由于关键字值的某种随机性，使得这种一一对应关系难以发现或构造。
- 因而可能会出现不同的关键字对应一个存储地址。即$k1 \neq k2$，但$H(k1)=H(k2)$，
- 这种现象称为冲突。

哈希表冲突

- 已知一组关键字为（12，19，23，28，39，51，56，76，84），哈希表长m=13，哈希函数为：h(key) = key%11



哈希表

对于哈希技术，主要研究两个问题：

（1）如何设计哈希函数以使冲突尽可能少地发生。

（2）发生冲突后如何解决。

哈希表的构造

- 构造哈希函数的目标是：
  - 使哈希地址尽可能均匀地分布在连续的内存单元地址上，以减少冲突发生的可能性；
  - 同时使计算尽可能简单以达到尽可能高的时间效率；
  - 根据关键字的结构和分布不同，可构造出与之适应的各不相同的哈希函数。

（1）直接定址法

（2）数字分析法

（3）除留取余法

……

哈希表的构造

除留取余法

- 除留余数法采用取模运算（%），把关键字除以某个不大于哈希表表长的整数得到的余数作为哈希地址。哈希函数的形式为：

$$h(key) = key \% p$$

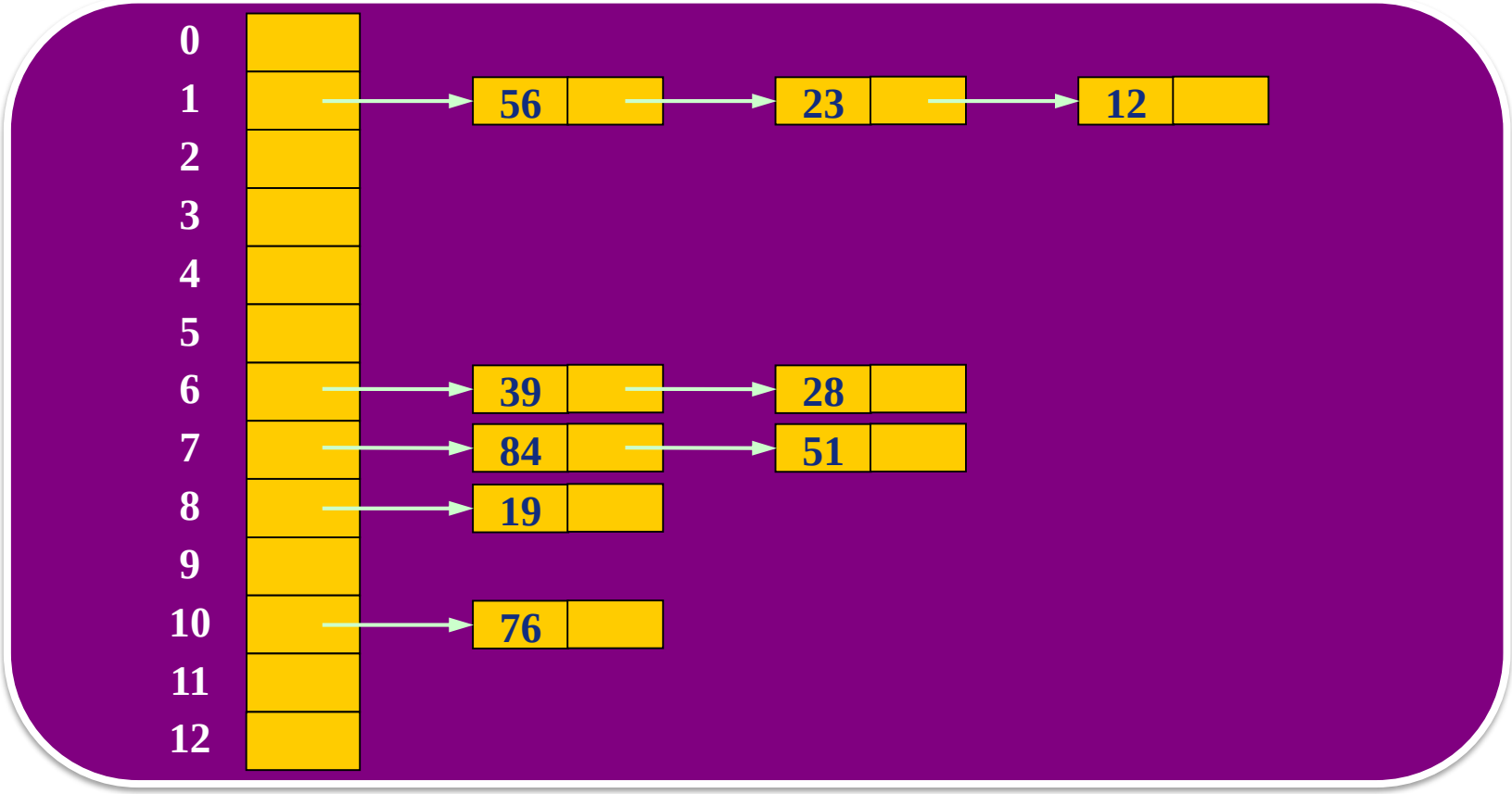
- 除留余数法的关键是选好p，使得记录集合中的每个关键字通过余数转换后映射到哈希表范围内任意地址上的概率相等，从而尽可能减少发生冲突的可能性。

哈希冲突解决方法

- 链地址法
  - 将所有关键字为同义词的结点链接在同一个链表中。每个链表中除了表头指针存储在哈希表数组中以外，所有元素都存储在数组以外的空间。

哈希冲突解决方法

- 关键字为（12，19，23，28，39，51，56，76，84），哈希表长m=13，哈希函数为h(key) = key % 11，如果使用链地址法进行存储，元素插入单链表时总是插在表头作为第一个结点。设插入顺序为（12，28，19，23，39，56，76，51，84），其结果将如图所示



目　录

1 索引的概念

2 空间索引

3 网格索引的结构

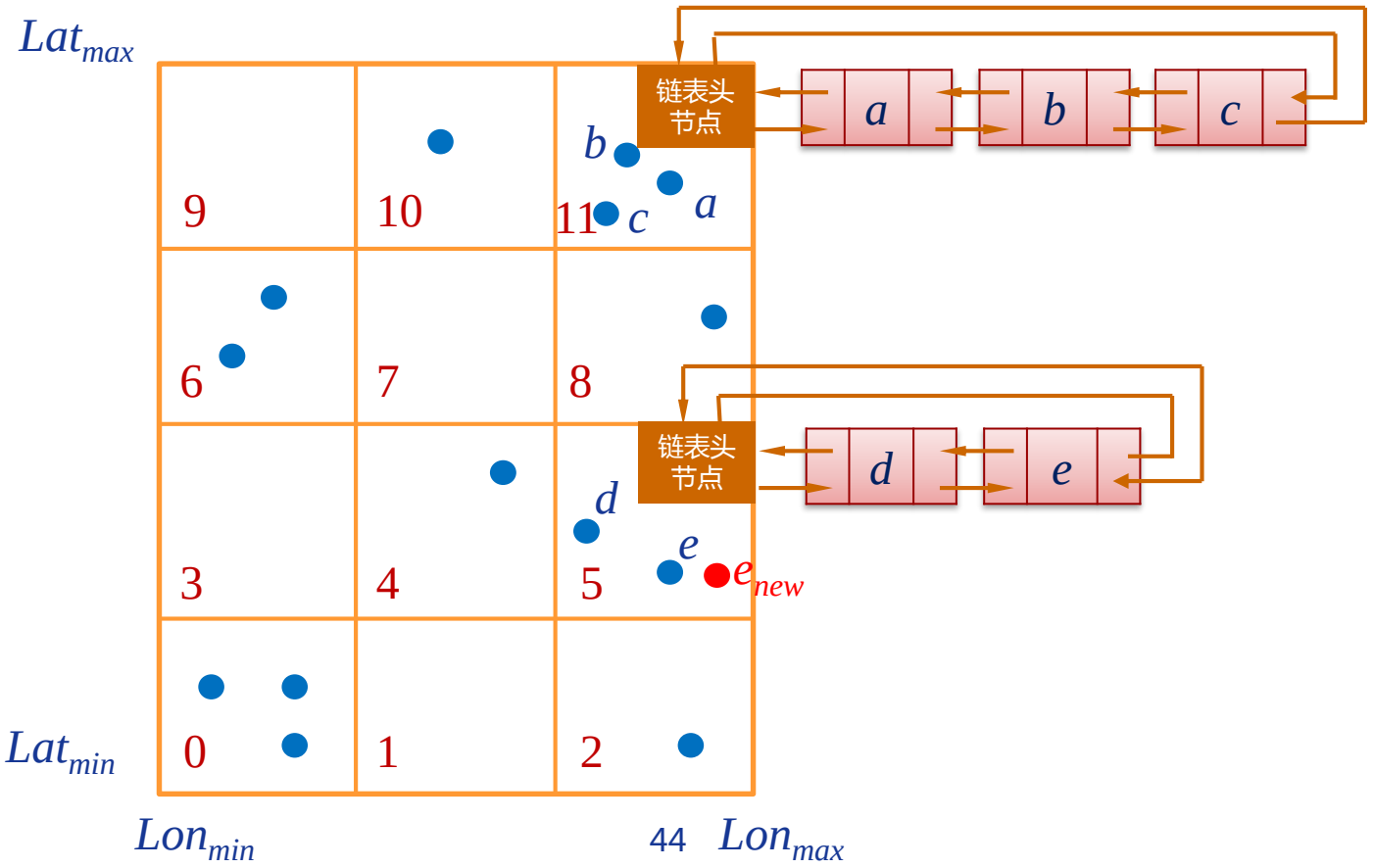
4 **出租车位置更新算法**

5 作业

出租车位置更新

出租车e进行了位置更新

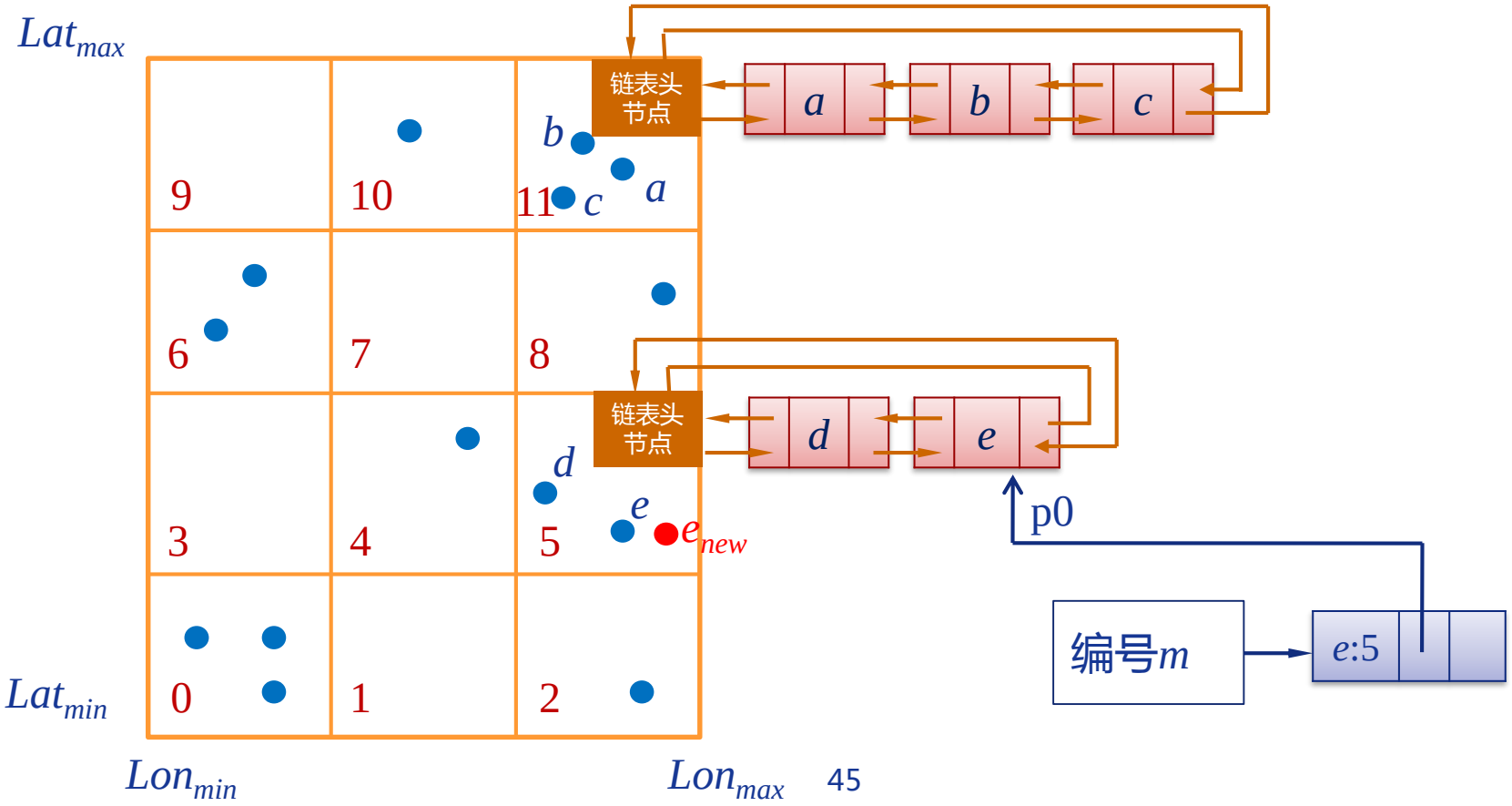
- 计算出租车的新位置e_{new}的经纬度(lon，lat)对应的cell_id；
- cell_id = 5



出租车位置更新

出租车*e*进行了位置更新

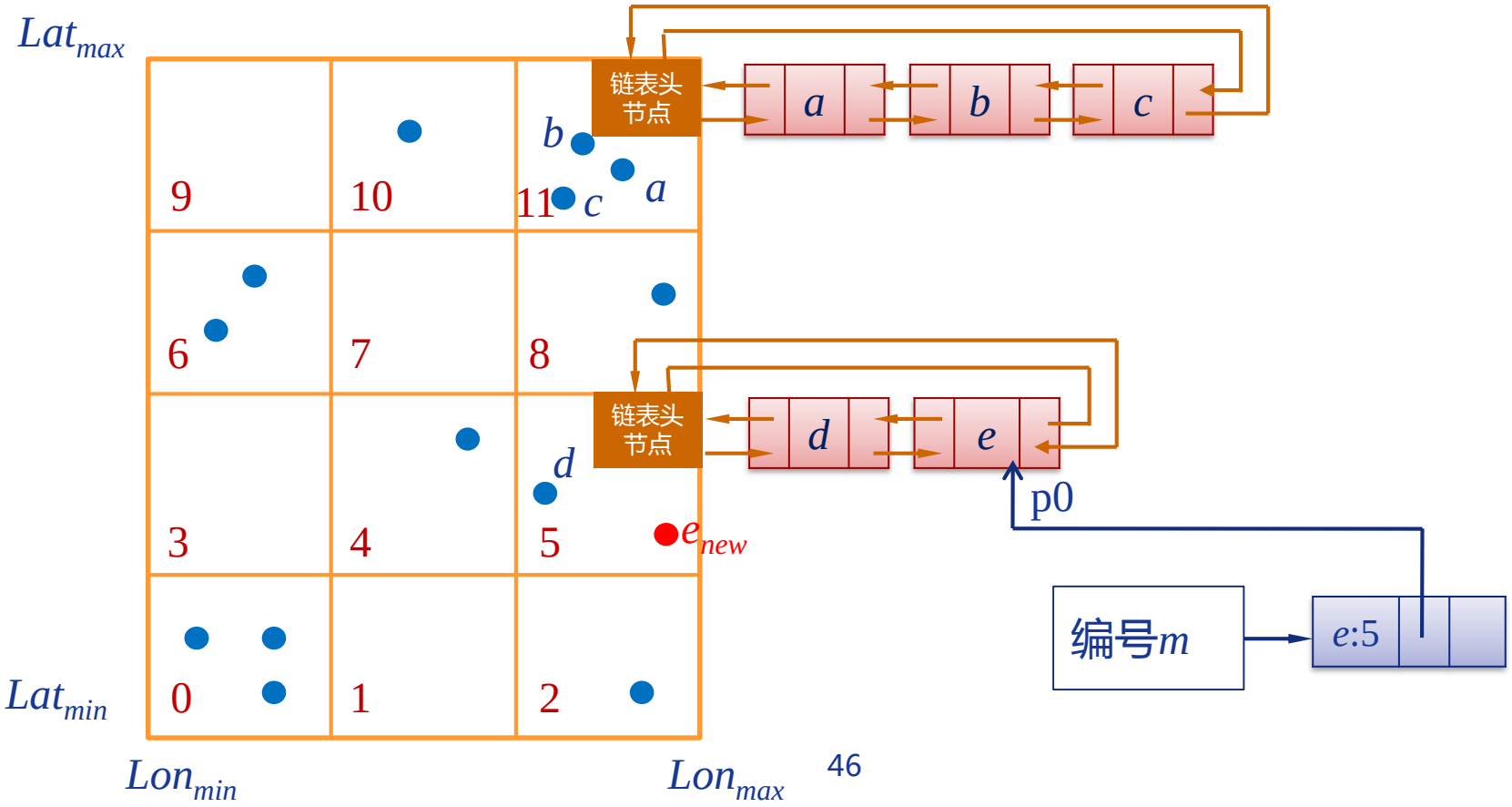
- 查找*e*在Hash表中上一次更新的网格编号cell_id0以及链表指针p0
- cell_id0 = 5



出租车位置更新

出租车*e*进行了位置更新

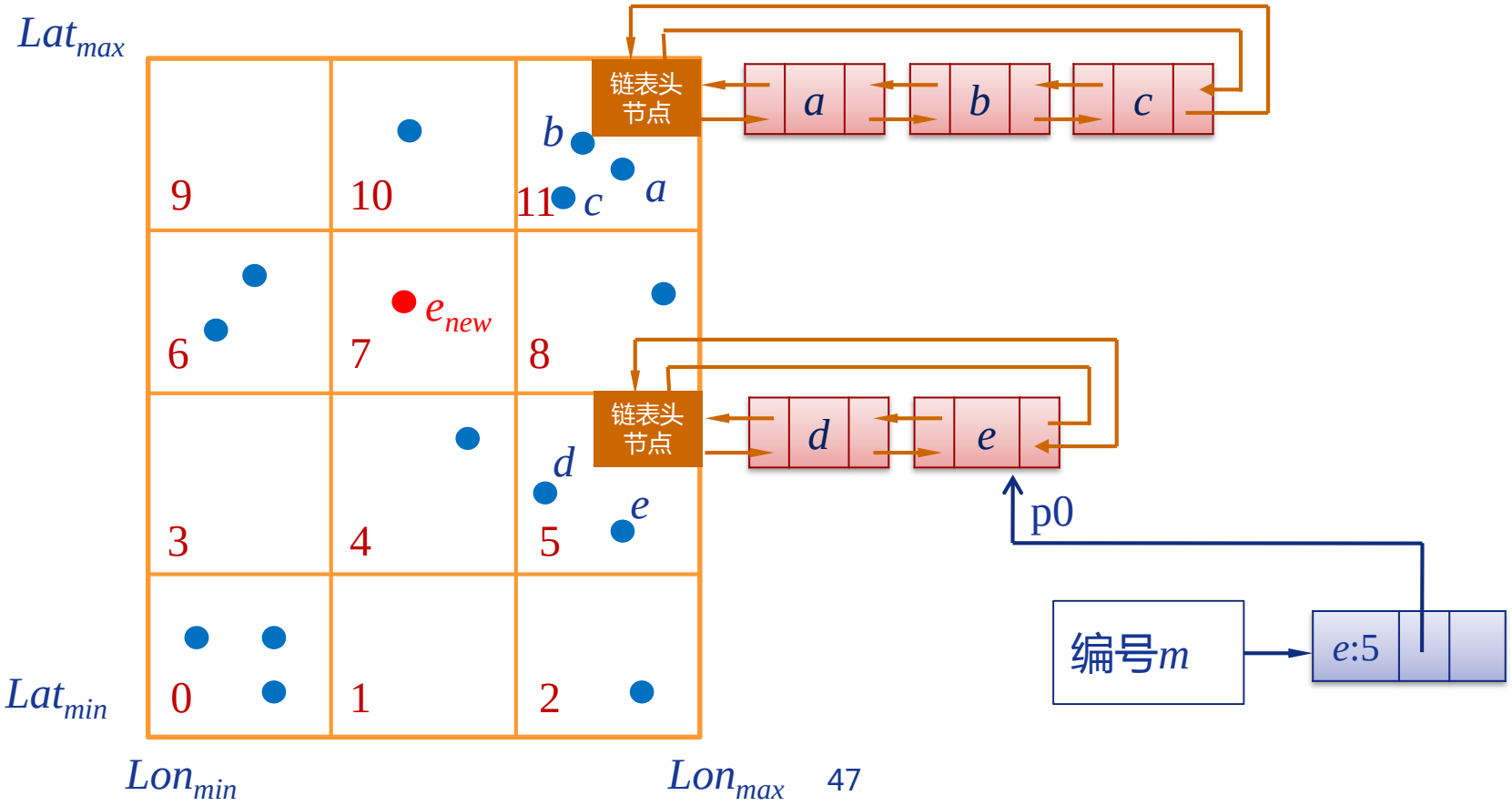
- 直接修改p0所指向出租车状态信息
- 如果更新前后的网格编号相等，更新结束
- cell_id0 = cell_id = 5，更新结束



出租车位置更新

出租车*e*进行了位置更新

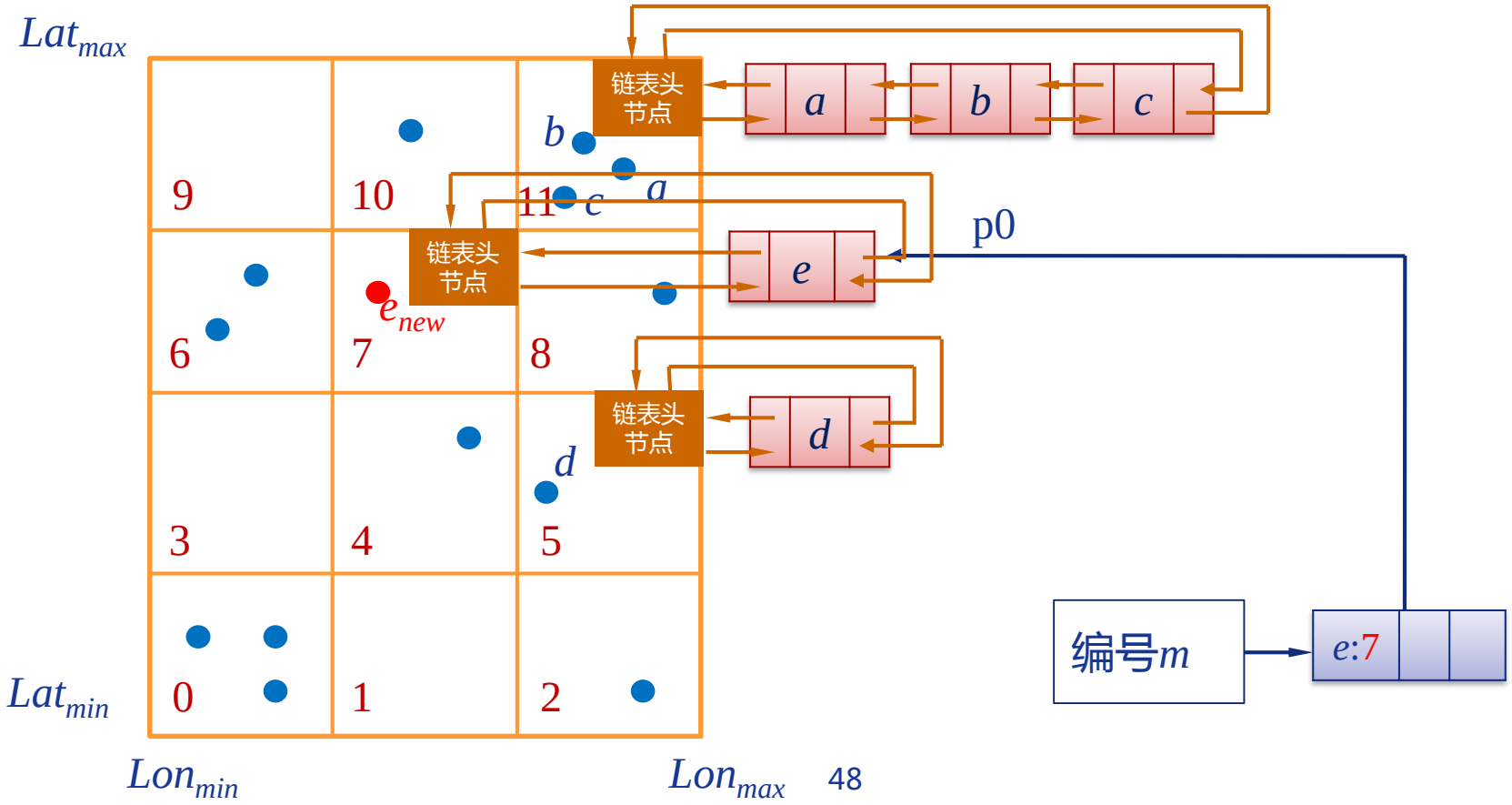
- 如果更新前后的网格编号不同，更新Hash节点的cell_id
- 将p0从cell_id0的双向循环俩表中删除，并将p0添加到cell_id的双向循环链表中；



出租车位置更新

出租车*e*进行了位置更新

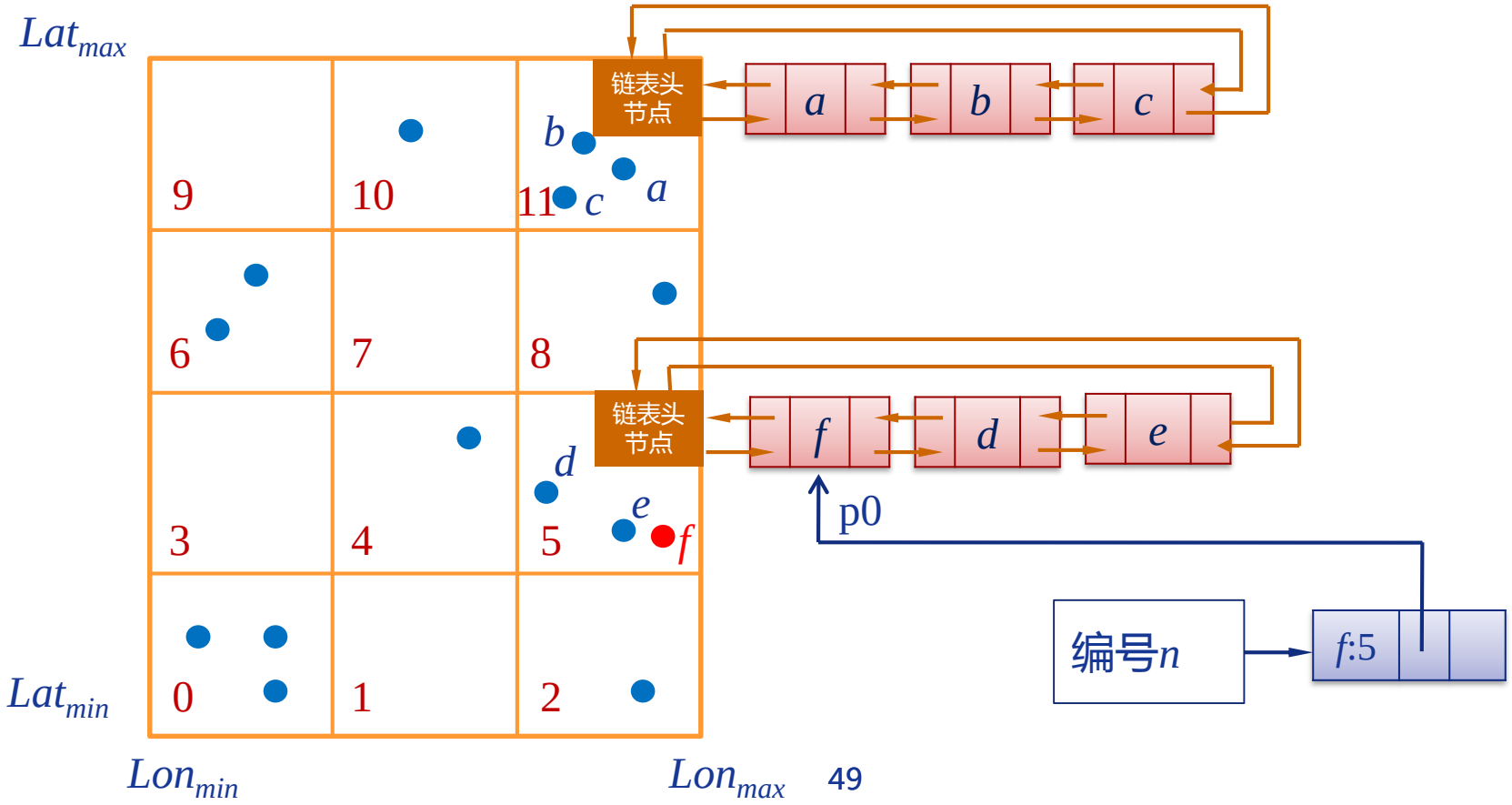
- 如果更新前后的网格编号不同，更新Hash节点的cell_id
- 将p0从cell_id0的双向循环俩表中删除，并将p0添加到cell_id的双向循环链表中；



出租车位置更新

出租车*f*进行了位置更新

- 如果在Hash表没有查找到该出租车，重新分配链表节点p0,赋值之后将p0插入到cell id的双向循环链表中，更新Hash表。



出租车位置更新算法

出租车位置更新算法

算法5：Grid_Update(lon, lat, id, timestamp)

输入：出租车的经纬度、唯一标识号id、时间戳以及网格属性

输出：无

1) **计算出租车位置(lon，lat)对应的cell_id；**
2) **根据id在Hash表中查找该出租车的上一次更新的cell_id0以及链表指针p0,如果没有找到，转5）；**
3) **直接修改p0所指向出租车状态信息后，如果cell_id0等于cell_id，返回, 否则，继续；**
4) **更新Hash节点的cell_id，将p0从cell id0的双向循环俩表中删除，并将p0添加到cell id的双向循环链表中；返回；**
5) **重新分配链表节点P0,赋值之后将P0插入到cell id的双向循环链表中，更新Hash表。**

目　录

1 索引的概念

2 空间索引

3 网格索引的结构

4 出租车位置更新算法

5 作业

作业

- 实现基于网格的索引
- 实现出租车的位置更新

谢谢！