位置相关服务（LBS）：
R-树索引

2017年

第四章 R-树索引

空间索引

- 思考：给定一张城市地图，如何快速有效地检索地标建筑物？

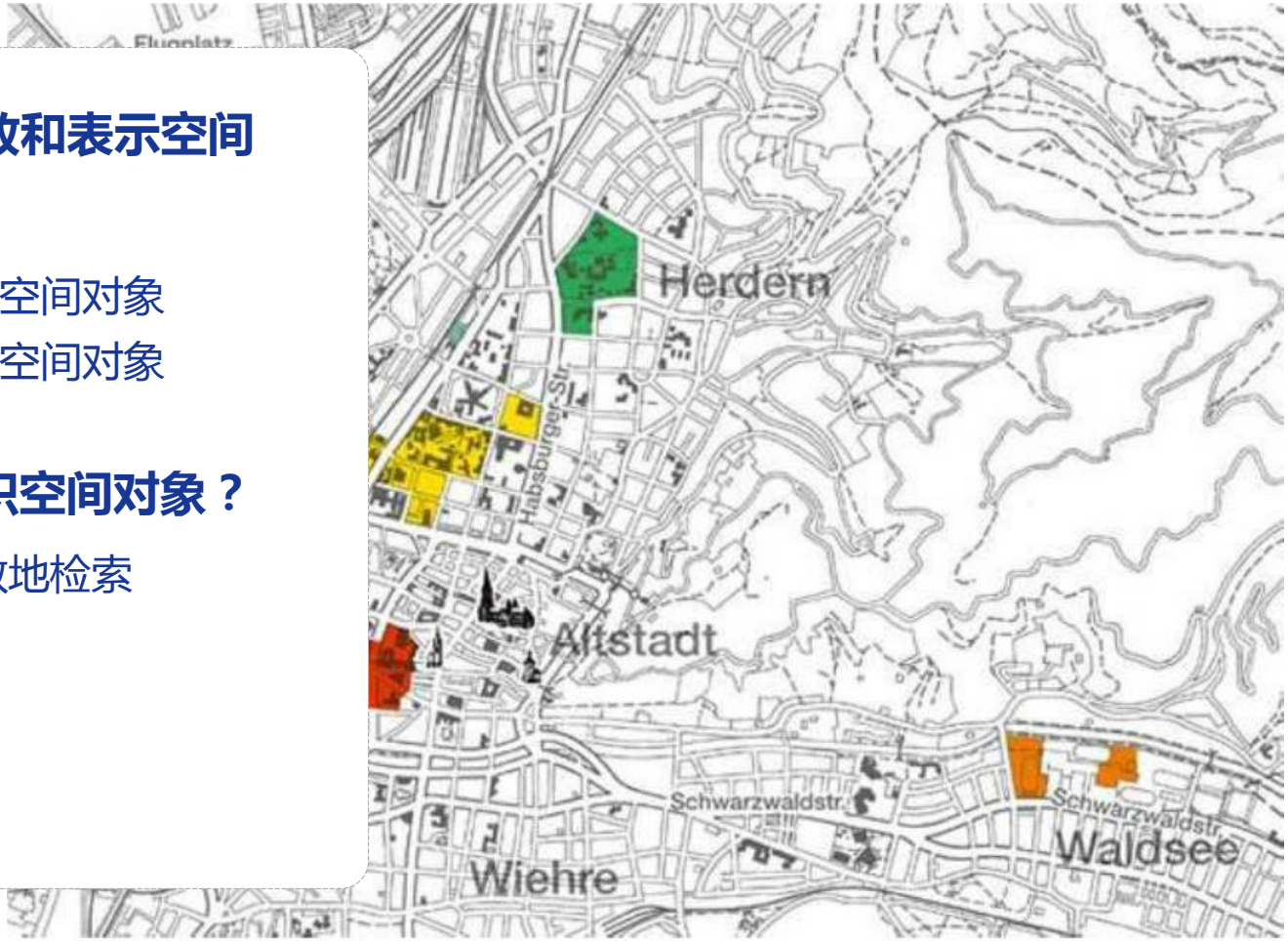
- **如何存放和表示空间对象？**

  高效存储空间对象

  简单表示空间对象

- **如何组织空间对象？**

  快速有效地检索



空间索引

❖ 思考：给定一张城市地图，如何快速有效地检索地标建筑物？

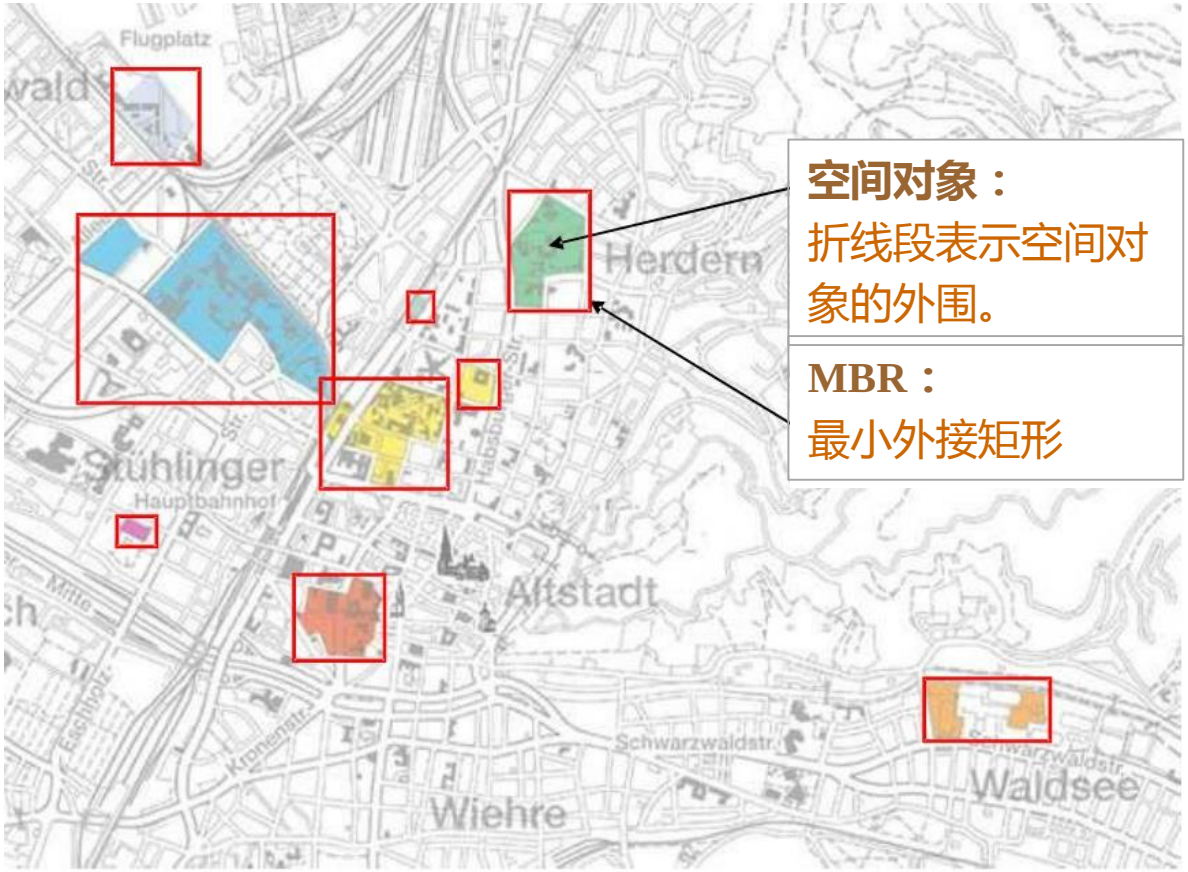
1. 存放和表示空间对象

- **空间对象**

  折线段表示空间对象的外围

- **最小外接矩形（MBR）**

  包容空间对象的最小外接矩形



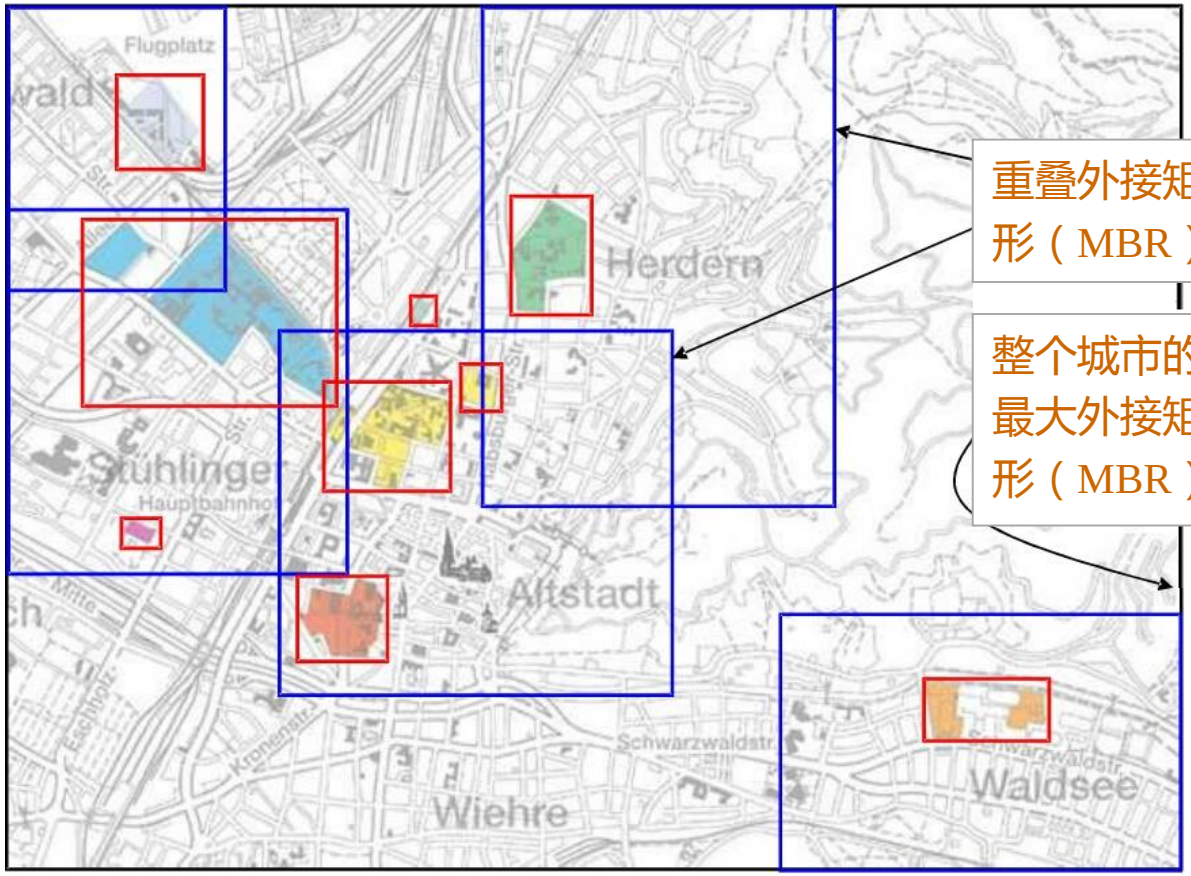
空间索引

❖ 思考：给定一张城市地图，如何快速有效地检索地标建筑物？

2. 组织空间对象

- **R-树结构**

  层次树状结构

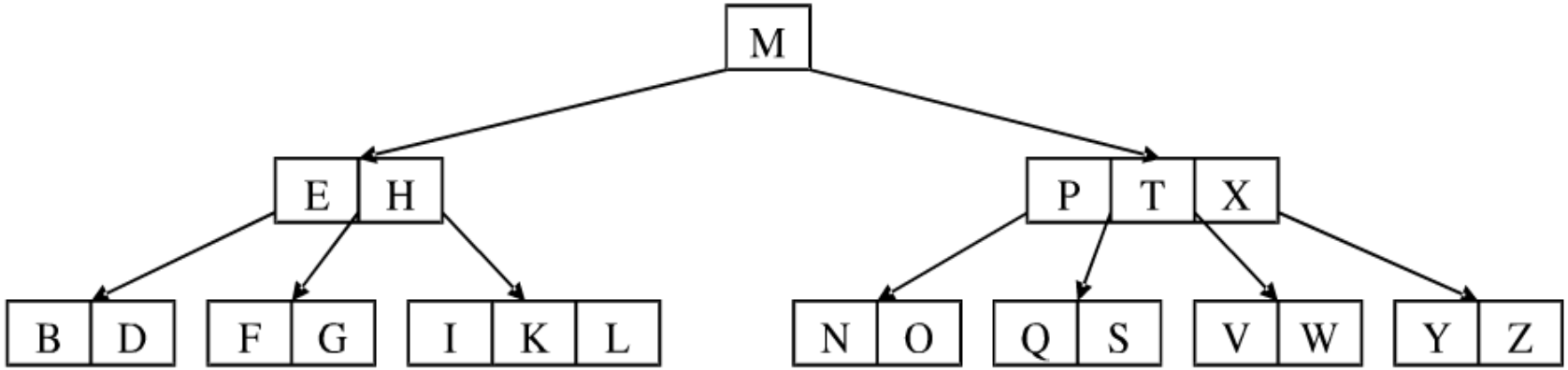


空间索引

- 注意：为了快速有效地检索空间对象
  - 支持二维空间
  - 支持范围查询
  - 查询结果：返回在查询区域Q内或与查询区域Q相交的空间对象

B-tree

- B-树是一棵有序、动态、多路查找树
- 关键字及所有子树都按查找树的形式呈现



- 每个节点包括一系列关键字及子树
- B-树能采用层高相对较小的树形结构存储数据量较多的数据集
  - 将遍历整个数据集的时间缩短为遍历树的层高的时间

R-tree

- R-tree
  - 1984年由Guttman提出
  - 是B树在高维数据空间的扩展
  - 用原始数据的最小边界矩形表示数据
  - 其插入、删除、更新操作都类似于B + -tree树
  - 能够有效支持的数据的维数：20维以下
  - 可以进行点查询和范围查询

R-树的概念

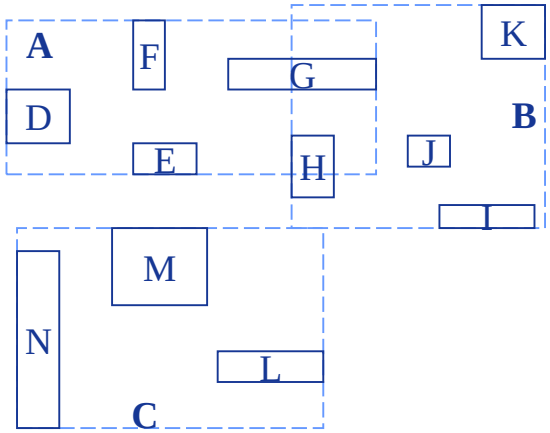
R树是一棵高度平衡的树

- 由中间节点和叶节点组成
- 叶节点存放实际数据对象的最小外接矩形
- 中间节点通过聚集其子节点的外接矩形形成，包含所有子节点的外接矩形

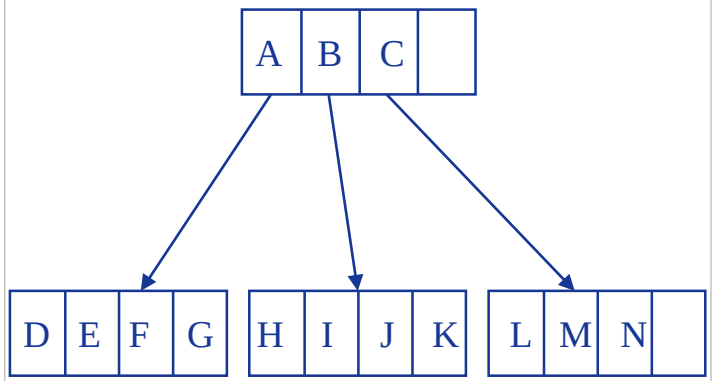
R树的特点

- 根节点若非叶子节点，则至少有两个子节点；
- 每个非根叶节点和非叶节点包含的数据项均介于m和M之间；
  （注：$m \leq M/2$）
- 所有叶子节点在同一层次。

空间对象



R树示例图



目　　录

R-树的数据结构

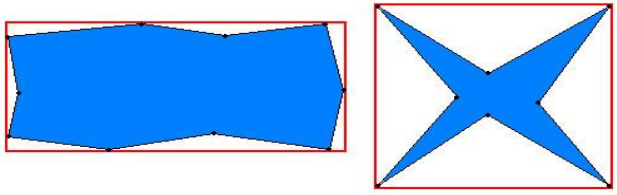
叶子结点的每一条索引记录为：

$$E = (I, \textit{tuple-identifier})$$

- I是数据对象的最小外接矩形
- *tuple-identifier*是指向数据对象的指针

表示数据的方法

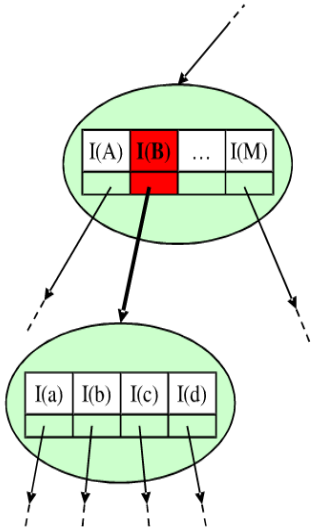
- 由于高维数据类型复杂，无法在索引结构中直接表示原始数据
- R-树中用原始数据的最小外接矩形（MBR)来表示数据

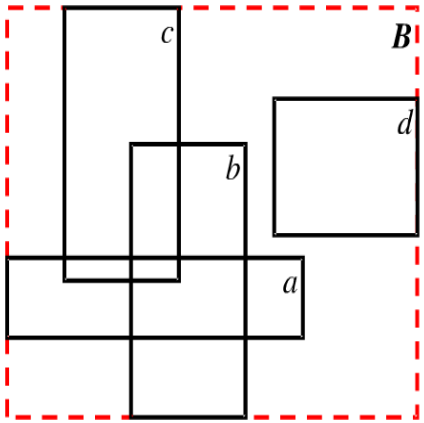


非叶子结点的每一条索引记录为：

$$E = (I, \textit{child-pointer})$$

- I是包含所有孩子结点外接矩形的最小矩形
- *child-pointer*是指向子树的指针

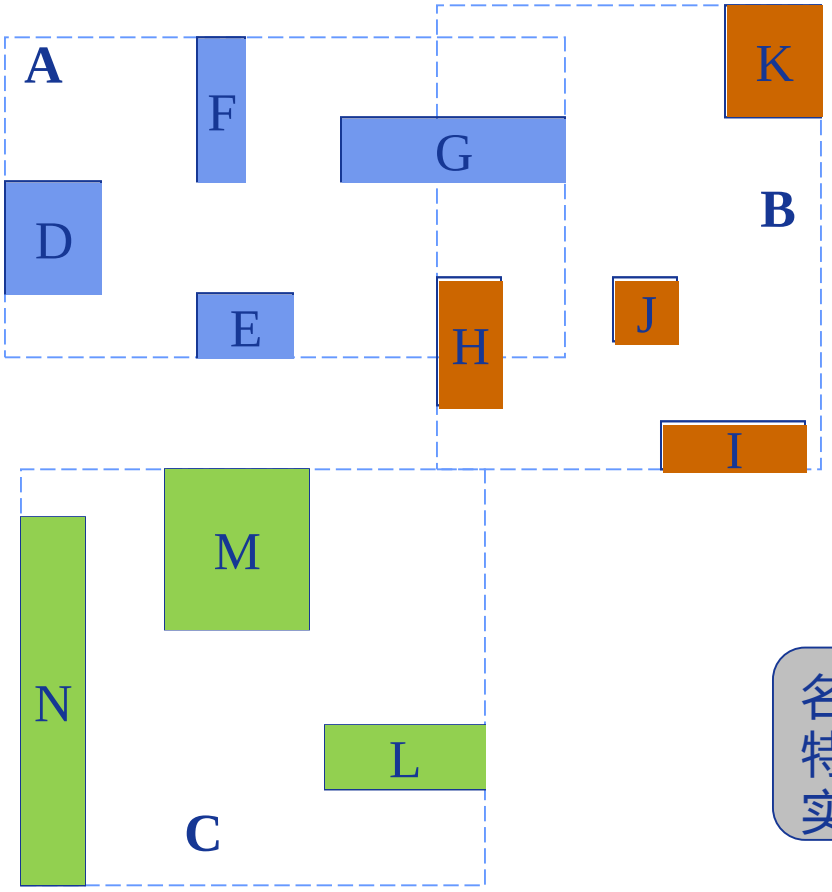


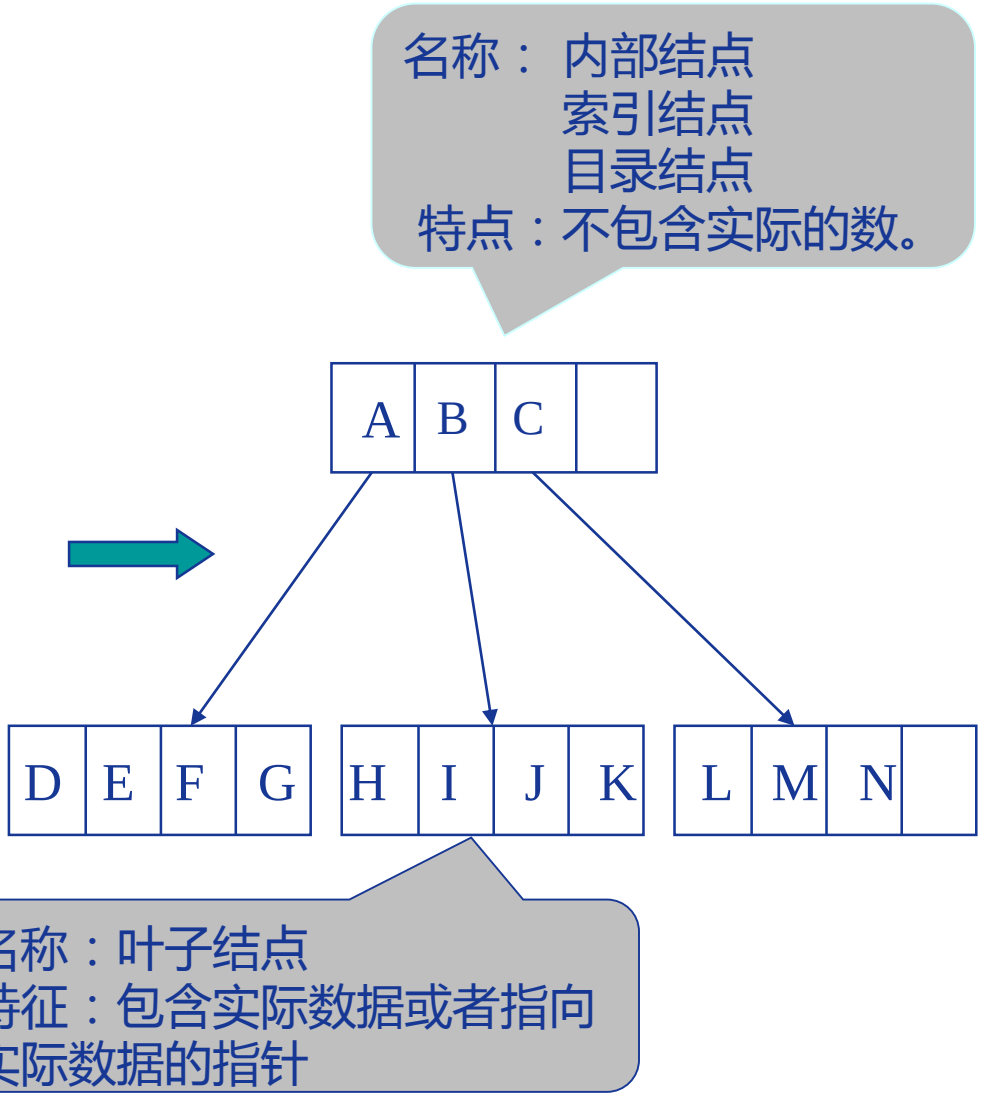


一个R树的例子

- 设M=4，m=2。

注：R树并非唯一的





目　　录

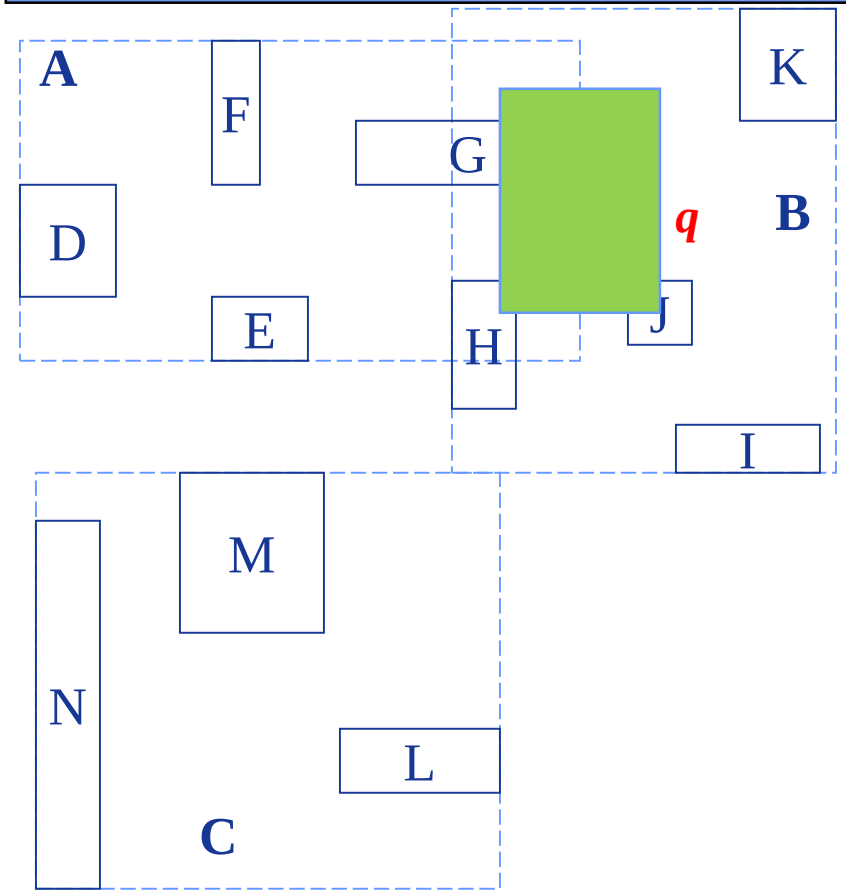
R-tree查询

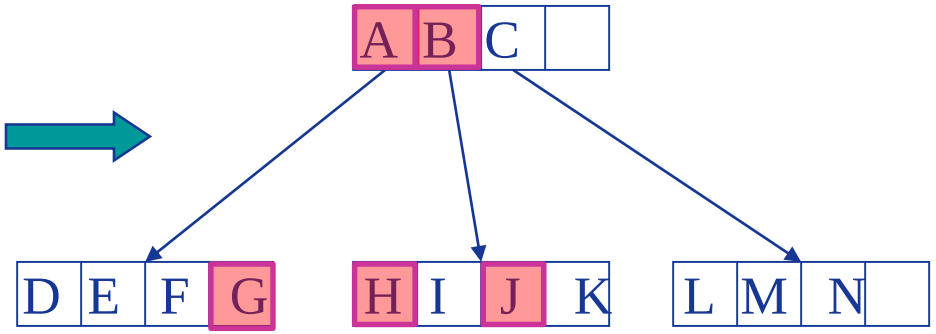
- R树的查询与B树极为相似。不同之处在于在查找的过程中可能有回朔。
- 查询从根结点开始，逐层向下搜索。
  - 当处理R树的某个结点T时，需要对T的每一个基本数据项E进行检查。
    - 如果E对应的MBR与查询区域Q有交叉，则查询在E对应的子树继续进行，反之则放弃E而转向E的下一个兄弟结点。
  - 当处理到叶子结点时，要对其中的每个基本数据项进行检查，将所有在空间上与查询区域相交的空间对象ID均作为结果返回。

R-tree查询

查询：对于给定的查询区域q, 查询所有与q相交或包含于q中的空间对象。

- 从根节点开始，对所有内部节点查找与q相交的子节点
- 到达叶子节点后，输出所有与q相交的空间对象。





R-tree查询

影响查询效率的主要因素：

1、重叠区域（ Overlap ）：导致查询时遍历多个路径

为了提高查询效率，在构建R-tree的时候要尽量减少重叠区域的面积

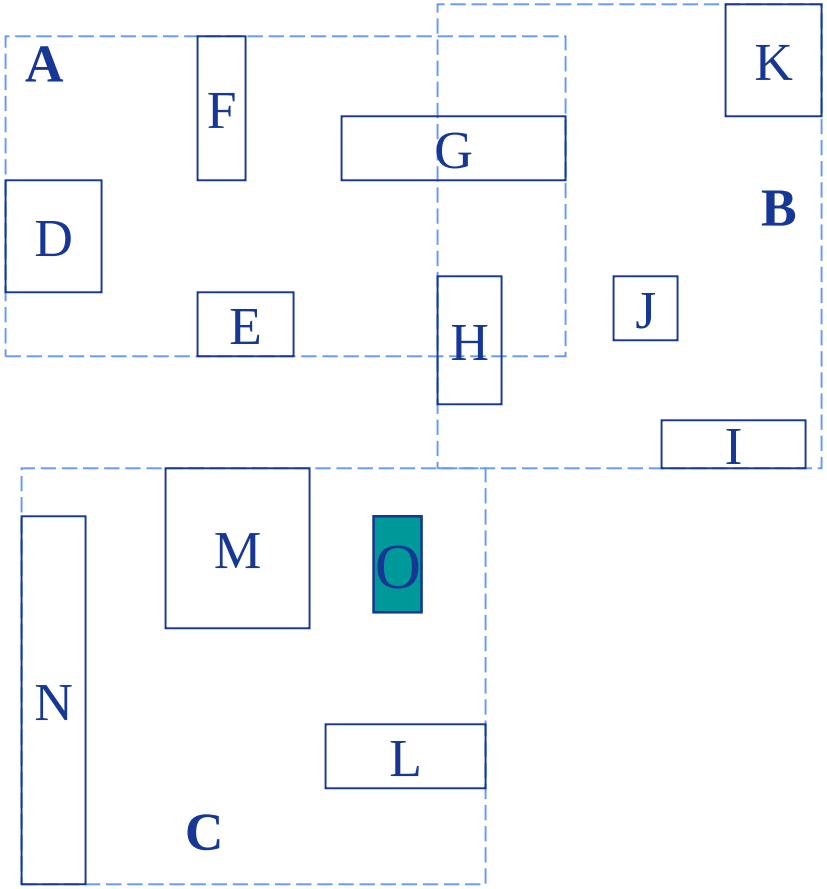
目　　录

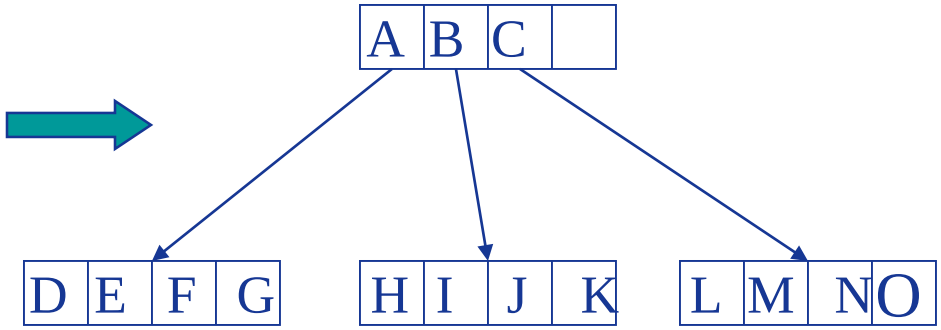
R-tree插入

- R树的插入与B树的类似，可以归纳为一个递归过程。
- 插入的大致步骤如下：
  - 从根结点出发，选择其中一个孩子插入新的空间对象，直到叶子结点。
  - 当新的空间对象的插入使叶子结点中的单元个数超过M时，需要进行结点的分裂操作。
    - 分裂操作是将溢出的结点分为若干部分。
    - 在其父结点删除原来对应的单元，并加入由分裂产生的相应的单元。
    - 如果这样引起父结点的溢出，则继续对父结点进行分裂操作。
    - 分裂操作保证了空间对象插入后R树仍能保持平衡。

R-tree插入

（1）向R-tree中插入一个数据，首先从根节点出发，在所经过的所有内部节点中选择这样的项：它所指向的子节点为了容纳下该数据面积需要扩大的最小;如出现相同的情况，则选择面积较小的那个。

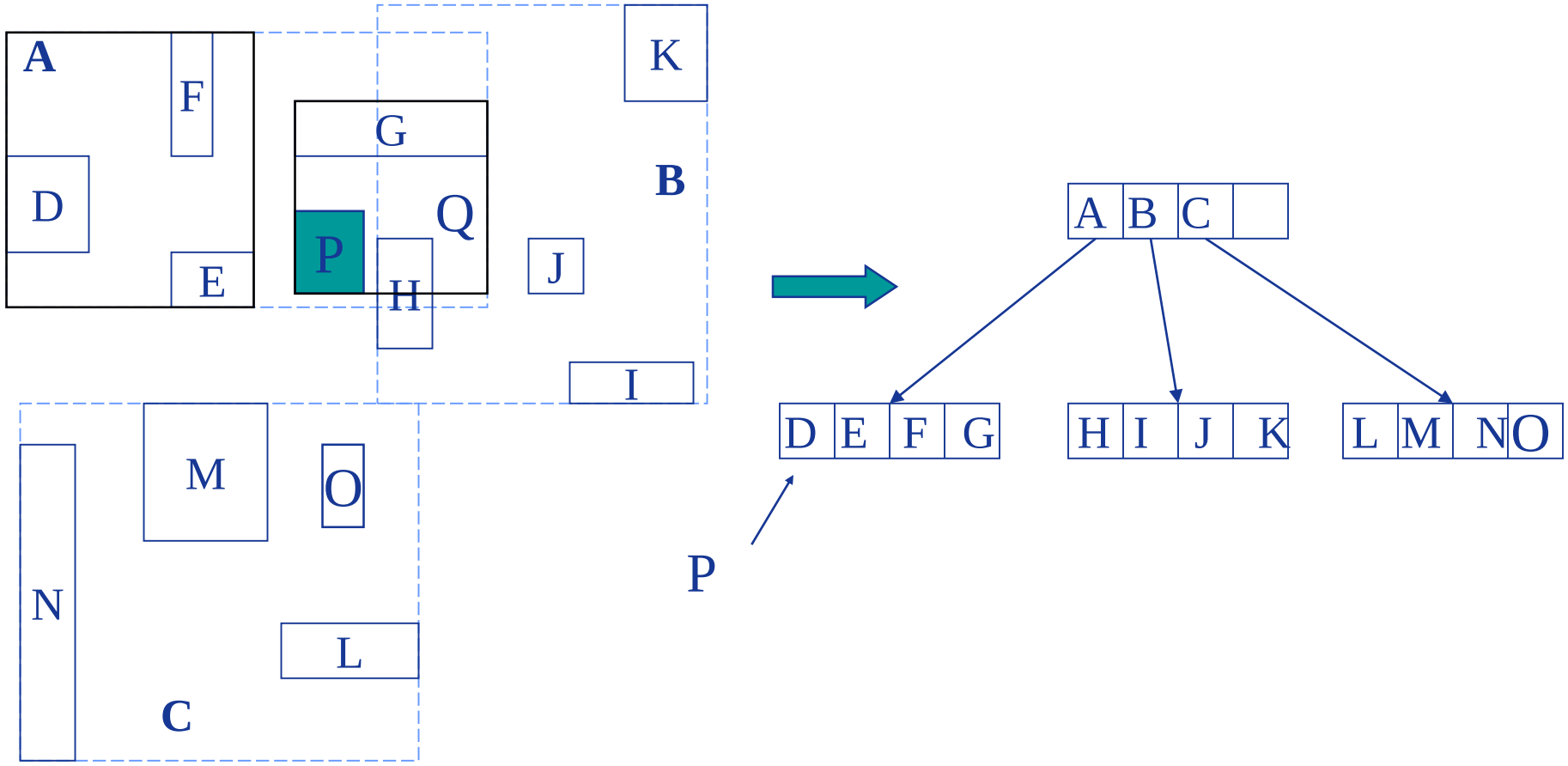




（2）到达叶子节点后，如节点中仍有剩余的空间，则将数据插入，然后依次修改祖先节点中的相应项的数据。

R-tree插入

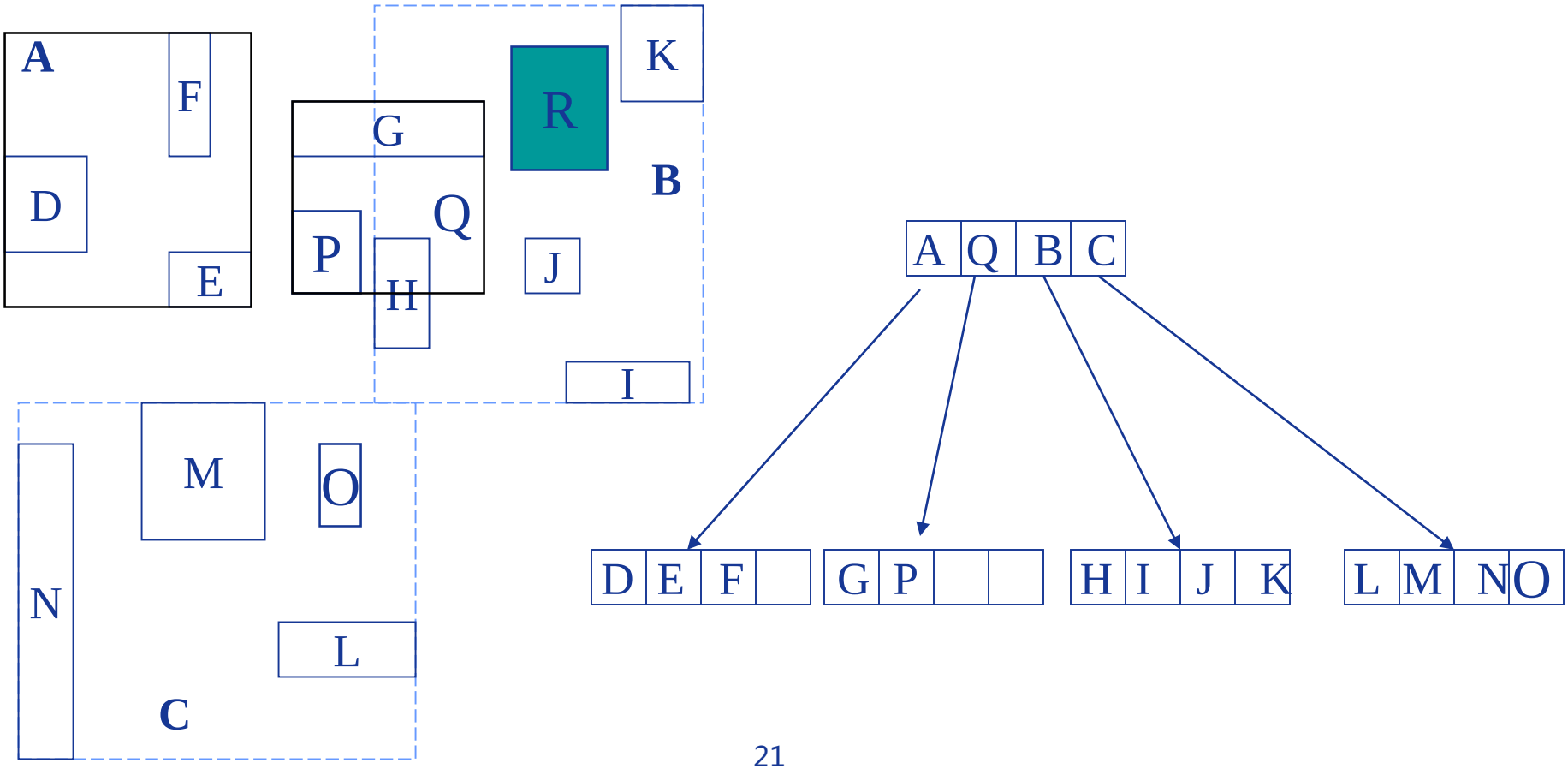
(3)如果叶子节点中已经没有足够的空间，则要进行分裂操作，并产生一个新的项，插入到其父节点中



R-tree插入

1: D E F G + P → D E F + G P

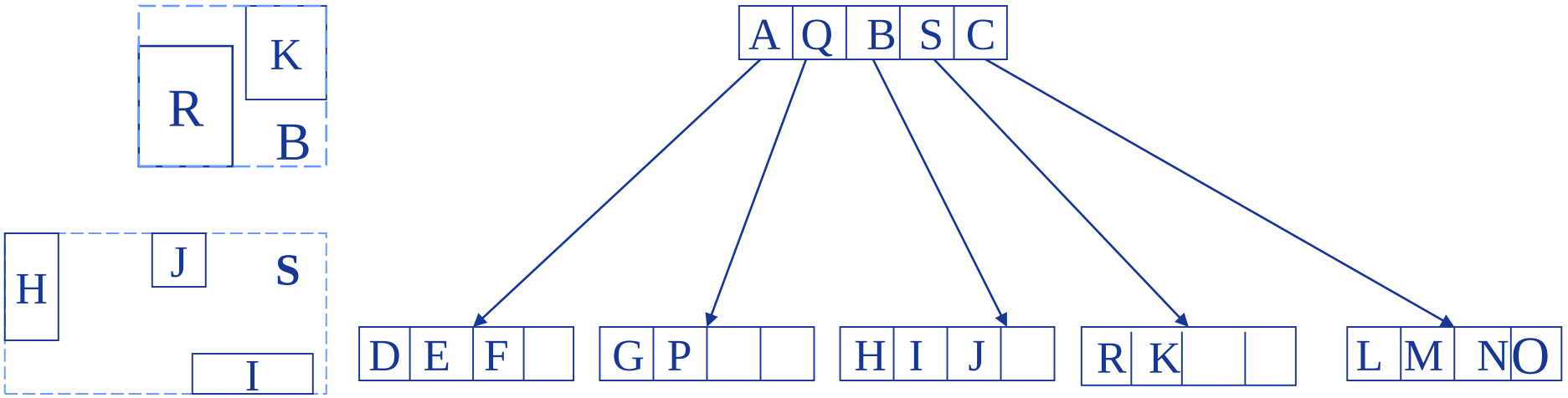
2: 生成一个新的项Q，插入到父节点中



R-tree插入

1: H I J K + R → H I J + R K

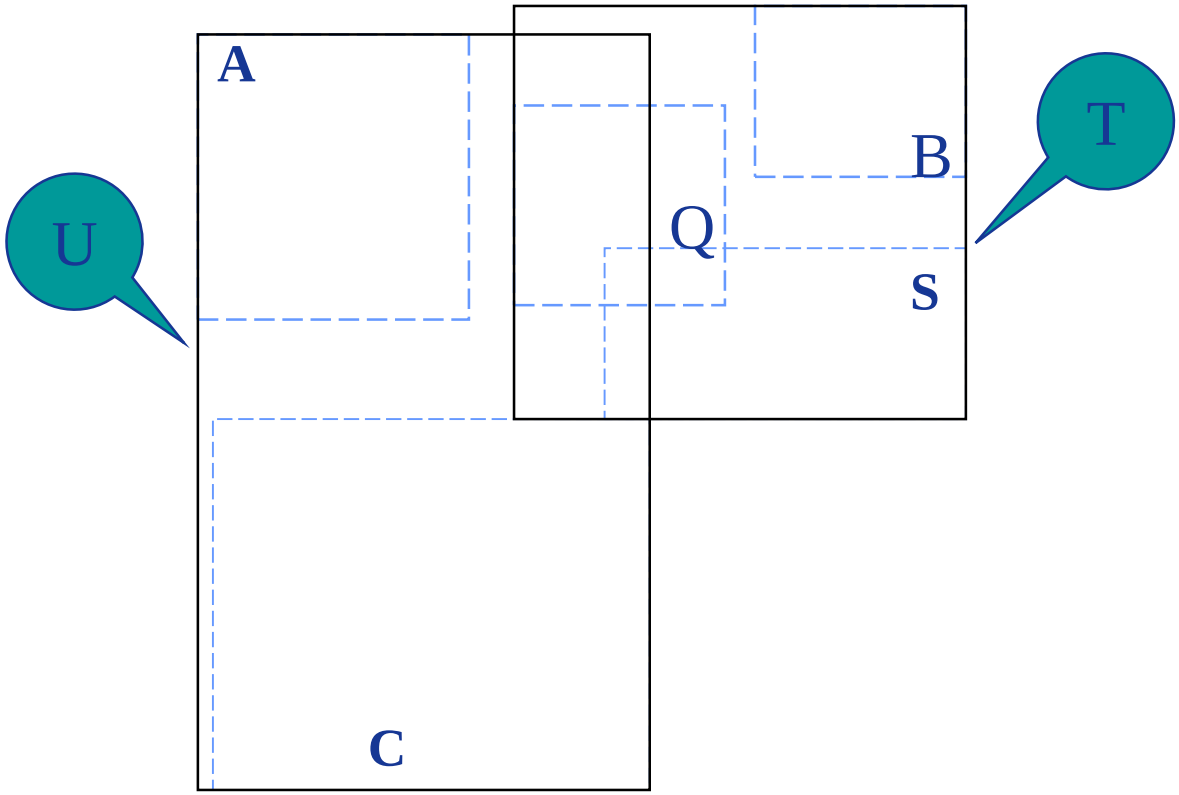
2: 生成一个新的项S，插入到父节点中



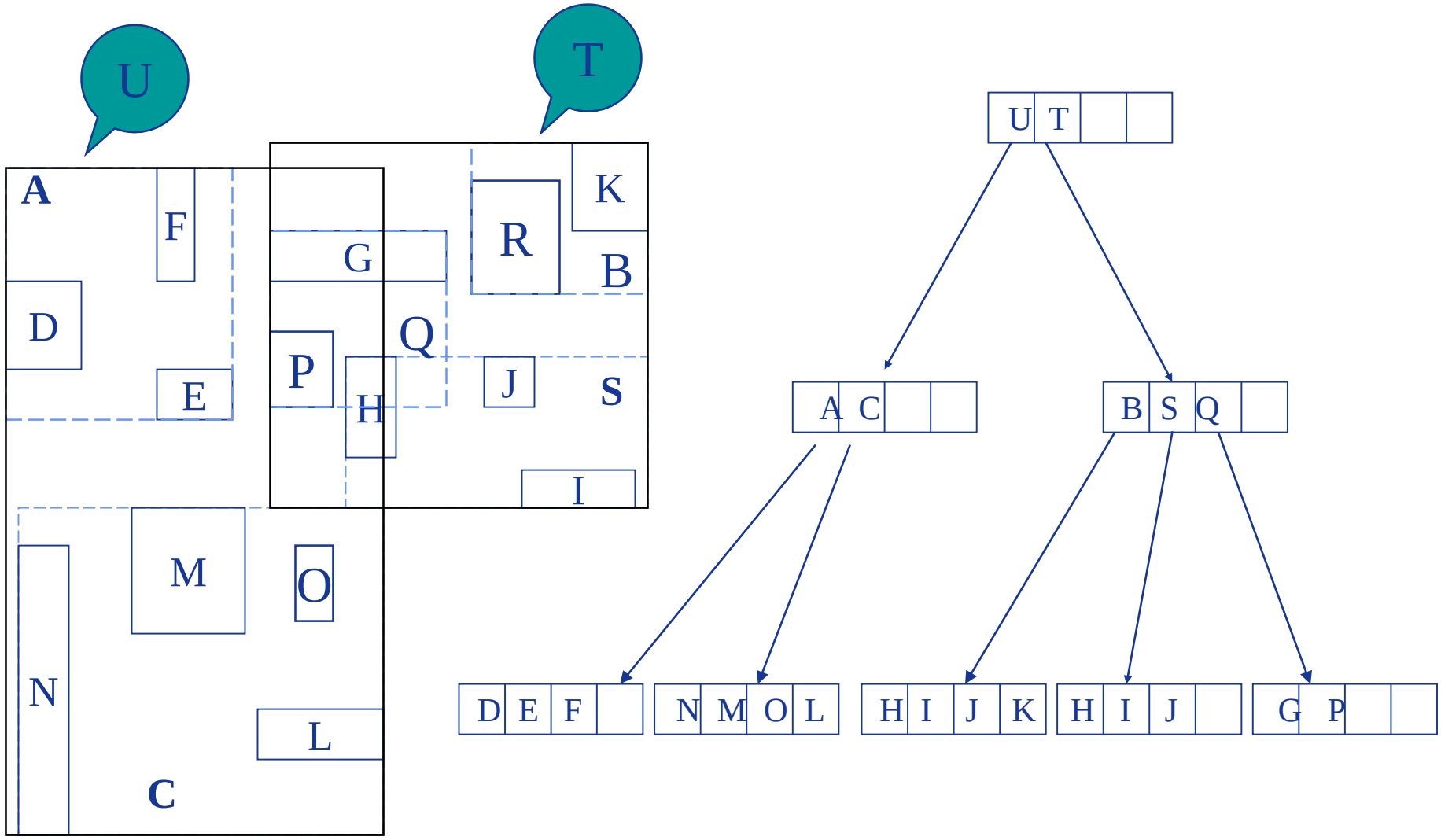
R-tree插入

3: 此时根节点已经没有剩余的空间，根节点要发生分裂

4：



R-tree插入

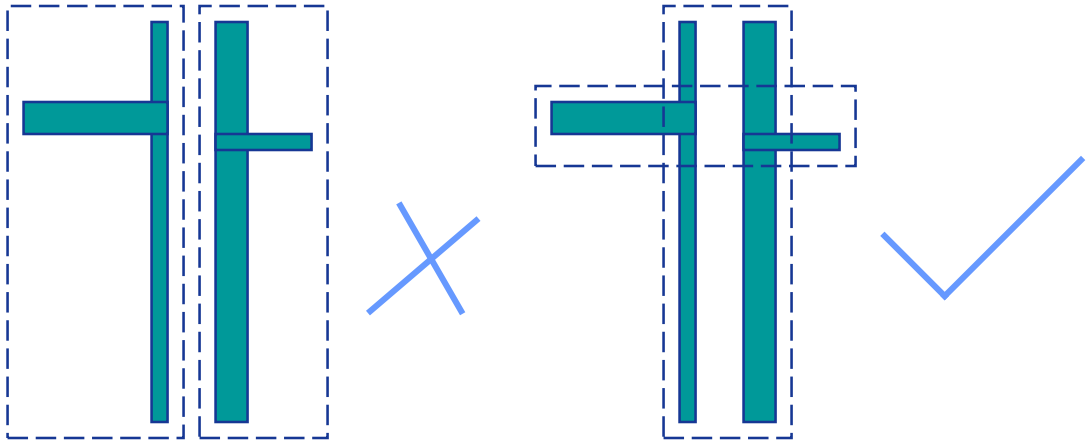


分裂算法

分裂算法希望能使以后的查询操作中，尽量少的对分裂后的两个子节点同时进行查询。

决定是否对子节点进行查询访问，要根据查询区域是否与子节点的MBR相交来决定

分裂算法的原则：分裂之后使两个子节点所覆盖的区域面积之和尽量的小。



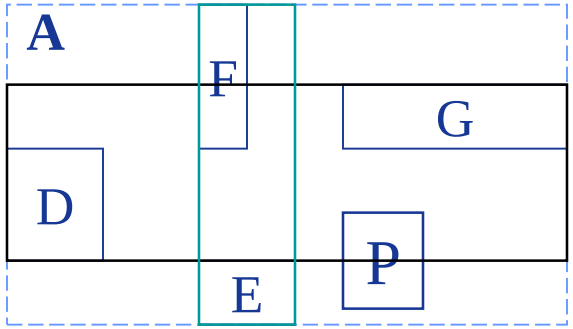
第一种分裂算法：穷尽组合法

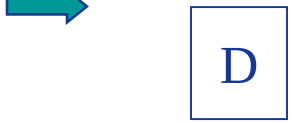
- 对每一种分裂的可能情况都进行一次计算，从中选择最好的一种做为分裂结果。
- 优点：分裂效果好
- 缺点：时间代价过大

第二种分裂算法：二次花费算法

（1）从要分裂的节点中首先选择两个数据，作为分裂后的新节点的种子；

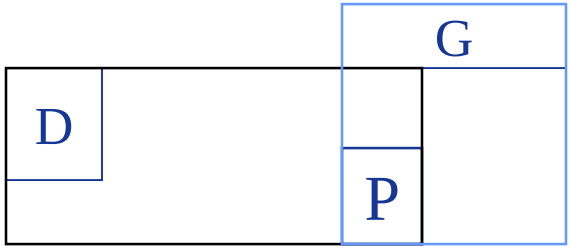
原则：其最小外接矩形面积最大的两个。

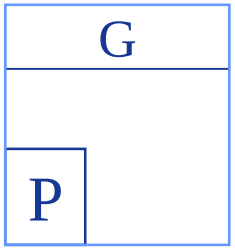




（2）依次从剩余的数据选择这样的项：两个新节点为了容纳下该项，需要扩大的面积之差最大。

然后将该数据放入面积需要扩大较小的节点中。





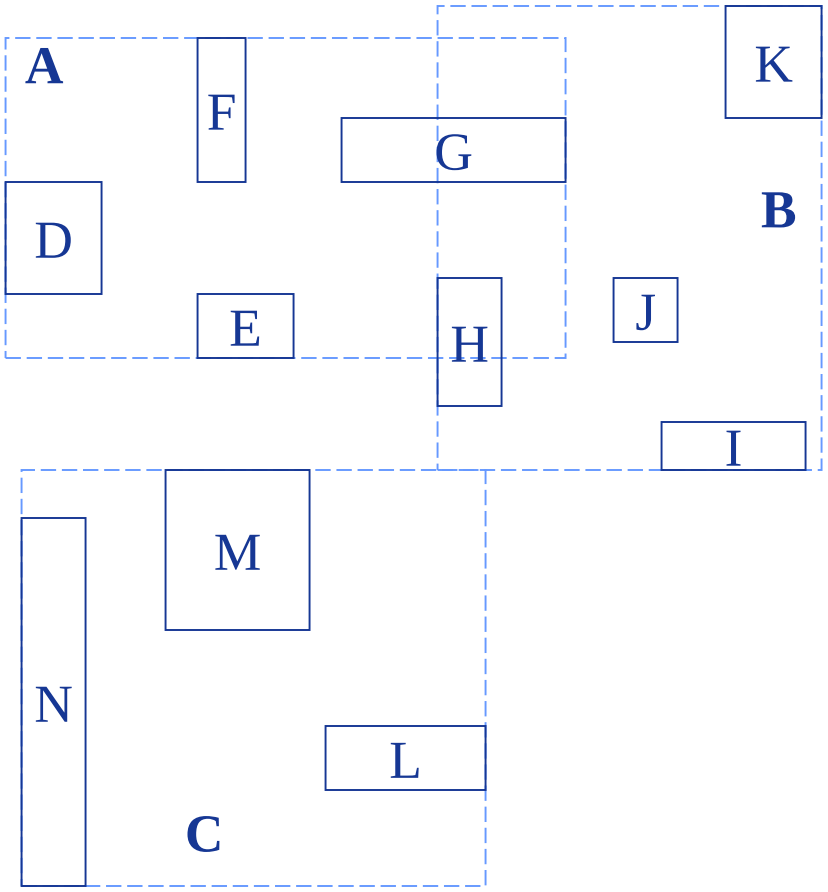
目　　录

R-tree删除

- 从R树中删除一个空间对象与插入类似
- 删除的步骤大致如下：
  - 首先从R树中查找到该空间对象所在的叶子结点，这就是R树的查找。
  - 查找到该空间要素所在的叶子结点后，删除其对应的单元。
  - 如果删除后该叶子结点单元个数少于m，需要进行R树的压缩操作
    - 压缩操作是将单元数过少的结点删除。
    - 并将因进行结点调整而被删除的空间对象重新插入到R树中。
    - 压缩操作使得R树的每个结点单元数不低于m这个下限，从而保证了R树结点的平衡和利用率。

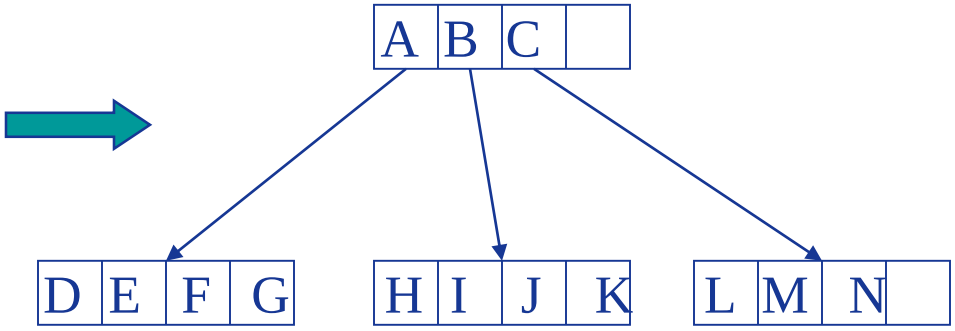
R-tree删除

（1）从R-tree中删除一个数据，首先要进行一次精确匹配查询，查找在R-tree中是否存在该数据，如不存在，删除失败；如存在，则从叶子节点中删除该数据



（2）如删除后节点中仍有足够多的项，则依次修改祖先节点中的数据，删除结束；

如节点中已经没有足够多的项，则将该节点从树中删除，并将节点中的所有项重新插入到树中。



R-tree

- R-tree主要存在以下缺陷
  - 内部节点存在重叠现象，由于节点间数据的相互重叠，在进行查询的时候，可能要遍历多条路径，查找多个叶子节点，最坏的情况下，要遍历所有的路径。
  - R-tree只对20维以下的数据较为有效，当维数逐渐增大的时候，内部节点中的重叠现象迅速恶化，导致查询性能急剧下降，使索引变的毫无意义，也就是通常所说的“维数灾难”。

R-tree的变种

- 为了能够减少内部节点之间的重叠问题，在R-tree的基础上又相继提出了以下一些索引结构
  - R+ -tree
  - R* -tree

目　　录

作业

- 实现R-树索引：插入与删除

谢谢！