BST 应用——区间查询问题

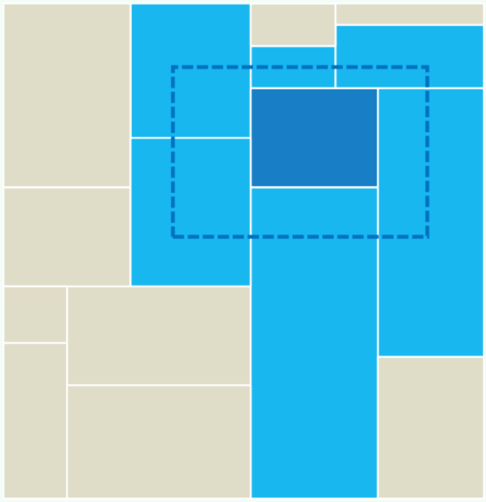
kd-Tree

建树复杂度：$\mathrm{O}(nlogn)$　递推式$T(n) = 2T(n/2) + \mathrm{O}(n)$

查询复杂度：$O(r + \sqrt{n})$　递推式$T(n) = 2T(n/4) + \mathrm{O}(1)$

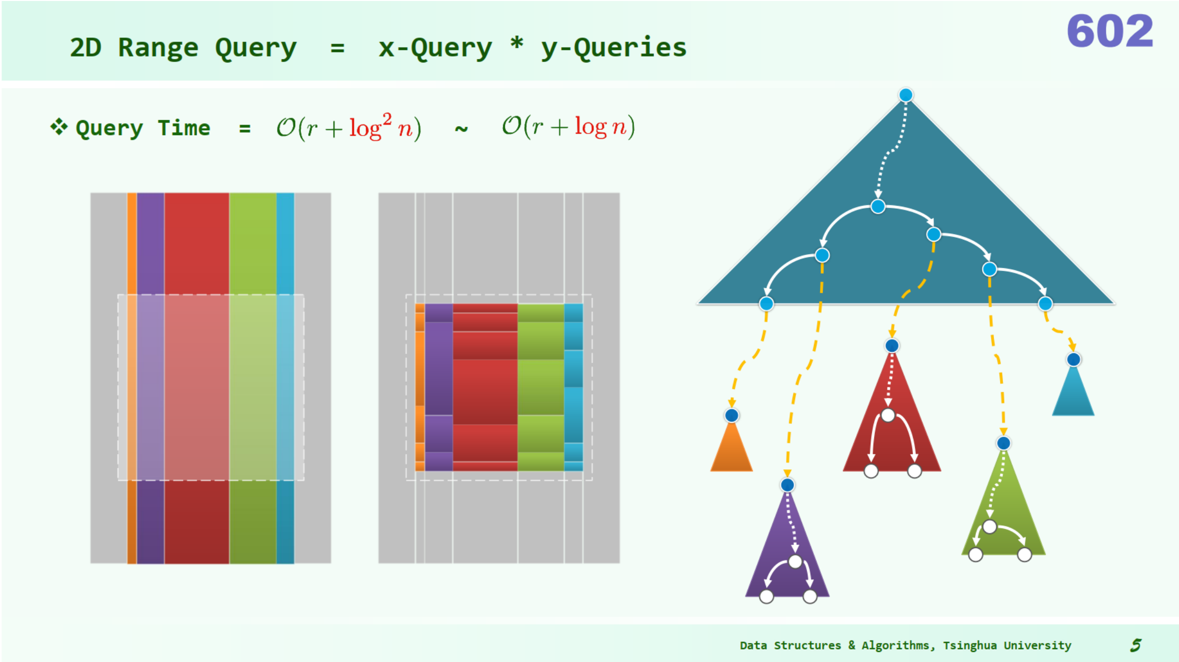
kdSearch(v,R)：热刀来切~~千~~(logn)层巧克力

```
❖ if ( isLeaf( v ) )
     if ( inside( v, R ) ) report(v)
     return
❖ if ( region( v->lc ) ⊆ R )
     reportSubtree( v->lc )
  else if ( region( v->lc ) ∩ R ≠ ∅ )
     kdSearch( v->lc, R )
❖ if ( region( v->rc ) ⊆ R )
     reportSubtree( v->rc )
  else if ( region( v->rc ) ∩ R ≠ ∅ )
     kdSearch( v->rc, R )
```



kdsearch的总次数是所有和查询区域有交集但未完全包含的区间个数。而区间有交集代表查询区域至少有一条边和点对应的区域有交集。所以总搜索的区间个数小于四条边延长后有交集的总区间个数。而一条边至多和孙辈节点分成的四个区域中的两个有交集。因此有递推式 $T(n) = 2T(n/4) + \mathrm{O}(1)$

树套树（范围树，也即range tree）

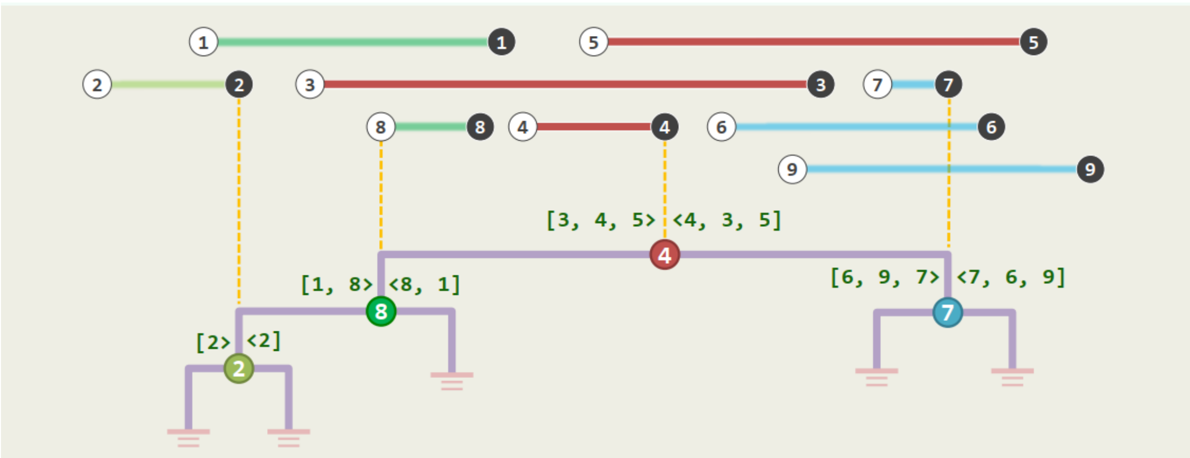


查询思路：先查询x方向，再查询y方向

查询复杂度：$O(r+log^2n)$

建树时间和空间复杂度均是$O(nlogn)$，其中空间复杂度的证明思路为：只需要求出每个结点重复出现的次数就可以，而一个结点出现且仅出现在其祖先的第二维度树中，因此每个结点出现的次数为$O(logn)$，总空间复杂度为$O(nlogn)$。时间复杂度证明的思路为：x方向上先做一次归并排序，相当于建好了第一层树，然后再在y方向上做归并，每归并一次复制得到的向量作为一棵树。

区间树（interval tree）



建树思路：思路一是先整体排序，然后每次取中点并由排序结果确定穿刺到的（也就是红色部分）左右端点的排序，然后向下递归。思路二是每次用中位数算法找到中点，在底层递归完成后得到排序向量，并由此确定左右端点的排序。

建树时间复杂度：中位数算法为$\mathrm{O}(n)$，自底向上的归并过程中可以实现排序，故复杂度为$\mathrm{O}(nlogn)$

查询思路：向下深入至叶，每次首先对该结点"穿刺"的线段做查询，然后根据查询位置和结点的相对位置关系继续神探直至根部

查询时间复杂度：由于每个结点都已经对线段左右端点做了排序，而树高不过$\mathrm{O}(logn)$，因此时间复杂度为$\mathrm{O}(logn+r)$

线段树（segment tree）

建树思路： 每次从所有左右边界取中点，如果该区间内不存在细分线段，则不继续向下递归

建树时间复杂度： $\mathrm{O}(nlogn)$

建树空间复杂度： 每条线段被切割成$O(logn)$条小线段存储，因此空间为$O(nlogn)$

查询时间复杂度： $\mathrm{O}(logn + r)$