Orthogonal Range Query
Assume: Each node $p$ stores:
- $p.pt$: splitting point
- $p.cutDim$: cutting dim
- $p.size$: no. of pts in $p$'s subtree
- Tree stores ptr. to root and bounding box for all pts.
- Recursive helper stores current node $p$'s cell

### Cases:
- $p == null$ → fell out of tree → $0$
- Query rect disjoint from $p$'s cell → $0$
  - return $0$
  - no point of $p$ contributes to answer
- Query rect contains $p$'s cell → $p.size$
  - every point of $p$'s subtree contributes to answer
- Otherwise:
  - Rect. + cell overlap → Recurse on both children

### Kd-Tree Queries II

$$\text{class Rectangle}{\{}$$

private Point low, high
public Rect (Point $l$, Point $h$)
  " boolean contains(Point $q$)"
  " boolean contains(Rect $c$)"
  " Rect leftPart (int cd, Points)"
  " Rect rightPart ("", "")"

$$\text{\}}$$

```java
int rangeCount(Rect $R$, KDNode $p$, Rect cell)
if ($p == null$) return $0$ // fell out of tree
else if ($R.isDisjointFrom(cell)$) return $0$ // overlap
else if ($R.contains(cell)$) return $p.size$ // take all
else {
  int $ct = 0$ // partial overlap
  if ($R.contains(p.pt)$) $ct += p.size$ // $p$ pt in range
  $ct += rangeCount(R, p.left, cell.leftPart(p.cutDim, p.pt))$
  return $ct$
}
```
Theorem: Given a balanced kd-tree storing n pts in $\mathbb{R}^2$ (using alternating cut dim), orthog. range queries can be answered in $O(n^{1/2})$ time.

Analysis: How efficient is our algorithm?
- Tricky to analyze
  - At some nodes we recurse on both children $\Rightarrow O(n)$ time?
  - At some we don't recurse at all!

Solving the Recurrence:
- Macho: Expand it
- Wimpy: Master Thm (CLRS)

Master Thm:
\[ T(n) = aT(n/b) + f(n) \]

For $a = 2, b = 4, d = \log_2 4 = 2$,
\[ T(n) = n^{\log_2 4} = n^2 \]

Since tree is balanced a child has half the pts + grandchild has quarter.

Recurrence: $T(n) = 2 + 2T(n/4)$
Recursively, $T(n) = 2 + T(n/4)$
Each half of $n$ gets recursed. $T(n) = n^{\log_2 4} = n^2$
If we consider 2 consecutive levels of kd-tree, $l$ stabs at most 2 of 4 cells:

Lemma: Given a kd-tree (as in Thm above) and horiz. or vert. line $l$, at most $O(n^{1/2})$ cells can be stabbed by $l$.

Proof: w.l.o.g. $l$ is horiz.
Cases: $p$ splits vertically

Stabbing: 3 cases
- cell is disjoint (easy)
- cell is contained (easy)
- cell partially overlaps or is stabbed by the query range (hard!)

Kd-Tree Queries

Orthog. range queries can be answered in $O(n^{1/2})$ time.

How many cells are stabbed by $R$? (worst case)
Simpler: Extend $R$'s sides to 4 lines and analyze each one.
Scapegoat Trees:
- Arne Anderson (1989)
- Galperin + Rivest (1993) rediscovered/extended
- Amortized analysis - $O(\log n)$ for dictionary ops amortized (guaranteed for find)
  - Just let things happen
  - If subtree unbalanced - rebuild it

Recap:
- Seen many search trees
- Restructure via rotation
- Today: Restructure via rebuilding
  
  - Sometimes rotation not possible
  - Better mem. usage

Example:

```
A: 0 1 2 3 4 5
```

```
P: b
a
```

```
j = \left\lceil \frac{k}{2} \right\rceil = 3
```

```
Time = O(k)
```

Overview:
Insert:
- Same as standard BST
  - if depth too high \( \rightarrow \log n \)
  - trace search path back
  - find unbalanced node - scapegoat
  - rebuild this subtree

Find:
- Same as std. BST
- Tree height \( \leq \log_{3/2} n \approx 1.71 \log n \)

Delete:
- Same as std. BST
- If num. of deletes is large rel. to \( n \) rebuild entire tree!

How? Maintain \( n, m \leftarrow 0 \)
Insert: \( n++; m++ \)
Delete: \( n--; \) \( m > 2n \) rebuild

How to rebuild? rebuild(p):
- Inorder traverse p's subtree \( \rightarrow \) array \( A[ ] \)
  
  - buildSubtree(A)
  
  - buildSubtree(A[0..k-1]):
    - if \( k = 0 \) return null
    - \( j \leftarrow \left\lceil \frac{k}{2} \right\rceil \); \( x \leftarrow A[j] \) median
    - \( L \leftarrow \) buildSubtree(A[0..j-1])
    - \( R \leftarrow \) buildSubtree(A[j+1..k-1])
    - return Node(x, L, R)
Announcements 3/30

- Midterm: Grades are published
  - Check out solutions before requesting regrades

  
<table>
<thead>
<tr>
<th>Score Range</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>90-100</td>
<td>A</td>
</tr>
<tr>
<td>80-90</td>
<td>B</td>
</tr>
<tr>
<td>60-80</td>
<td>C</td>
</tr>
<tr>
<td>&lt;60</td>
<td>D</td>
</tr>
</tbody>
</table>

- Prog Assign 2
  Part 1 (20%) - due Wed Apr 5
  Part 2 (80%) - due Wed Apr 19

- 4XX Info Session - 5pm in Gannon.

Rebuild offset

\[ \text{Rebuild node } u \text{ when: } \]
\[ u.\text{insert} t > u.\text{size} \frac{1}{2} + \text{rebuild offset} \]

\[ \approx 5 - 10 \]
Quick-and-Dirty kd-Tree Analysis:

- Ideal case:
  - Uniformly distributed pts
  - kd-Tree subdivision ~ grid $\sqrt{n} \times \sqrt{n}$
  - How many grid squares stab the query range?

- Good for average case, not worst case
Programming Assignment 2: Sliding Midpoint kd-Tree

- Extended binary tree
  - internal nodes: split but no data
  - external nodes: data but no splitting

- Sliding midpoint splitting rule
  - Splitting rule for squares

- Cut-dim
  - which side of cell is longer (if tied use x (vertical))

- Cut-value
  - mid point by default

- But: Two exceptions

  - if all pts are equal along cut dimension - flip to other
  - if all pts on same side of mid point - slide splitting plane to closest point

- Rebalancing
  - Whenever the number of insertions grows too large (~ half size of subtree)
    rebuild the entire subtree from scratch.

- How?
  - Traverse subtree
  - BulkCreate - Build tree recursively by applying sliding mid point