R-tree

- R-tree is a depth-balanced tree (just like a B-tree).

- 2 kinds of nodes:

minimum bounding box.

leaf nodes: $|(mbb_1, objID_1)|(mbb_2, objID_2)|\ldots$

1 data block

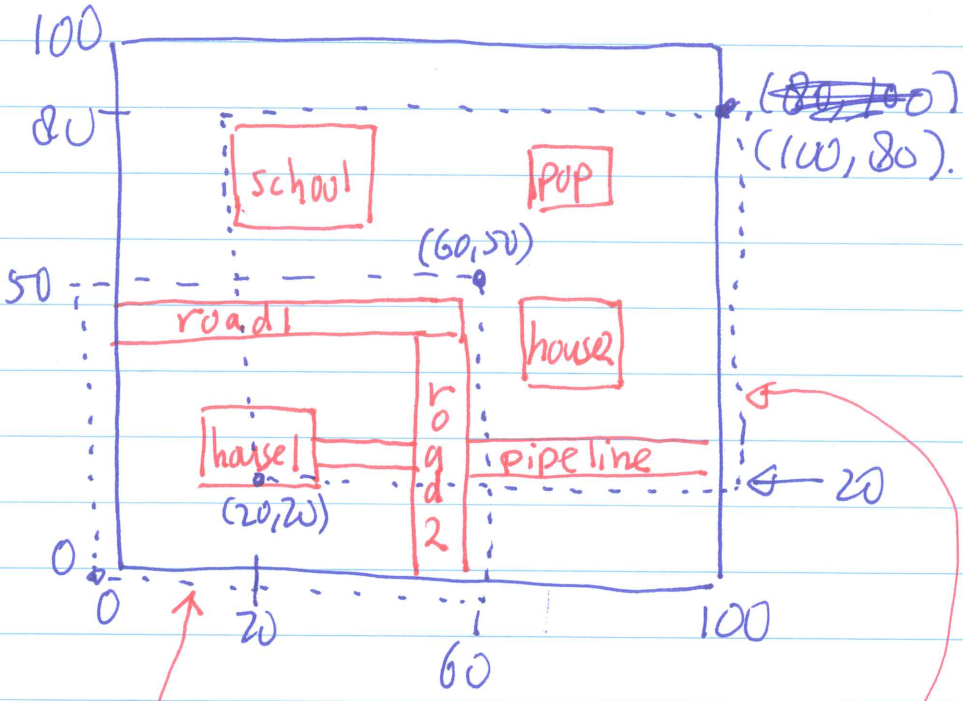
Non-leaf node: $|(bb_1, childID_1)|(bb_2, childID_2|\ldots$

bounding box of all entries in this child

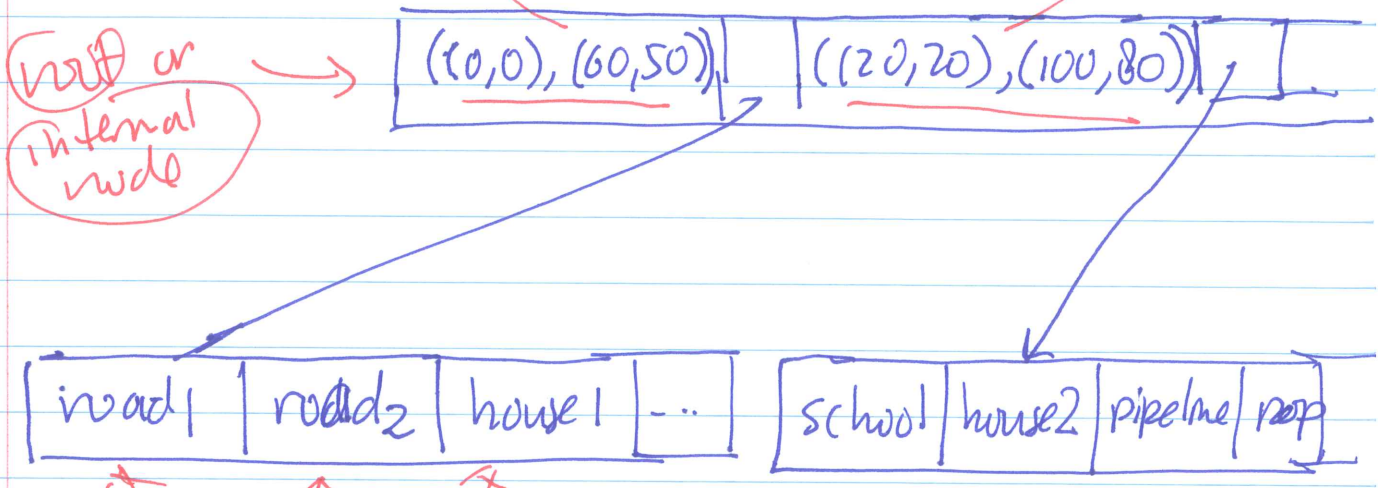
pointer to child node.

Example:

Data:



R-tree:



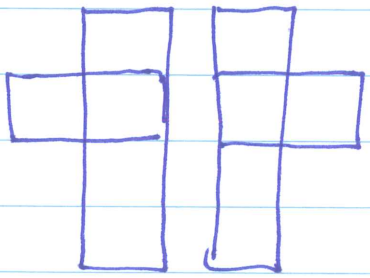
objects that lie ENTIRELY inside the parent's bounding box.

Note:

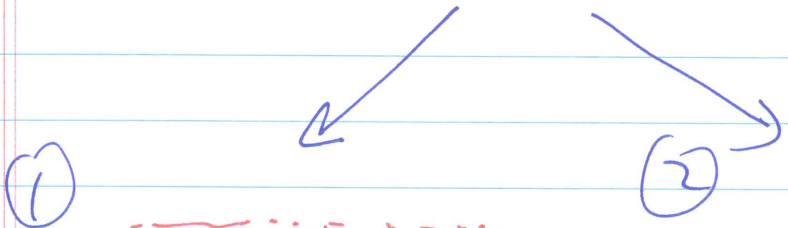
The bounding boxes can overlap !!!

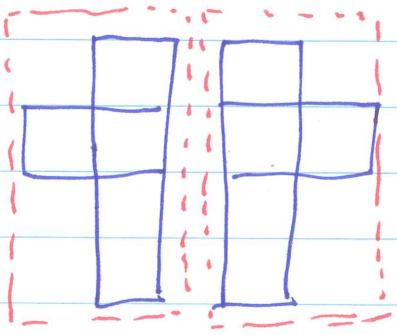
How to decide on bounding boxes:

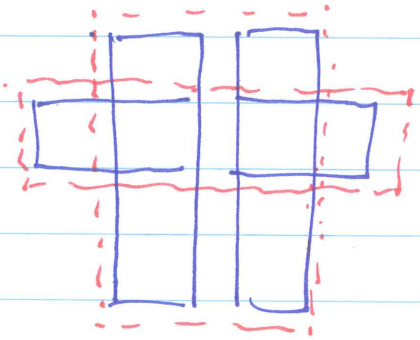
Objects:



How to put objects into 2 bounding boxes ???







Guidelines:

(1) minimize the total areas of the bounding boxes.

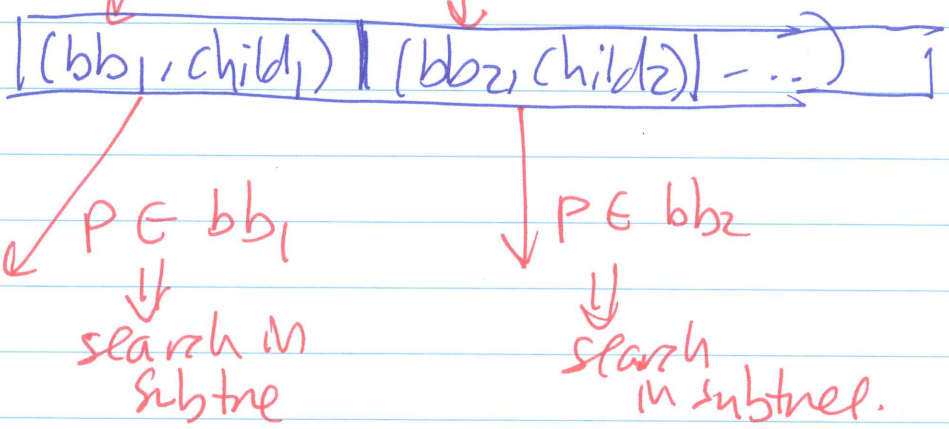
(2) Minimize the overlapping areas between bounding boxes.

☆ These 2 goals are conflicting

→ goal (1) is primary goal

Search with R-tree

- Given a point $p(x, y, \ldots)$.
- Search Algorithm:



- Pseudo Code:

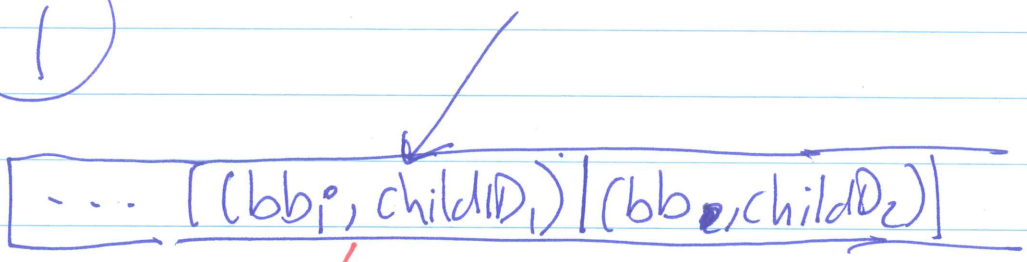
```
N = root;
~~for each~~

if (N == leaf)
{  for each object ∈ N:
       if (p ∈ mbb(object))
           add object to output.
}
else for each area ∈ N
       if (p ∈ area)
           search in ~~area~~ child.
}
```

Sketch of Insert Algorithm:

Insert object (mbb_x, $objID_x$).

mbb_x ← min. bounding box

(1)



There is a rectangle bb_i such that:

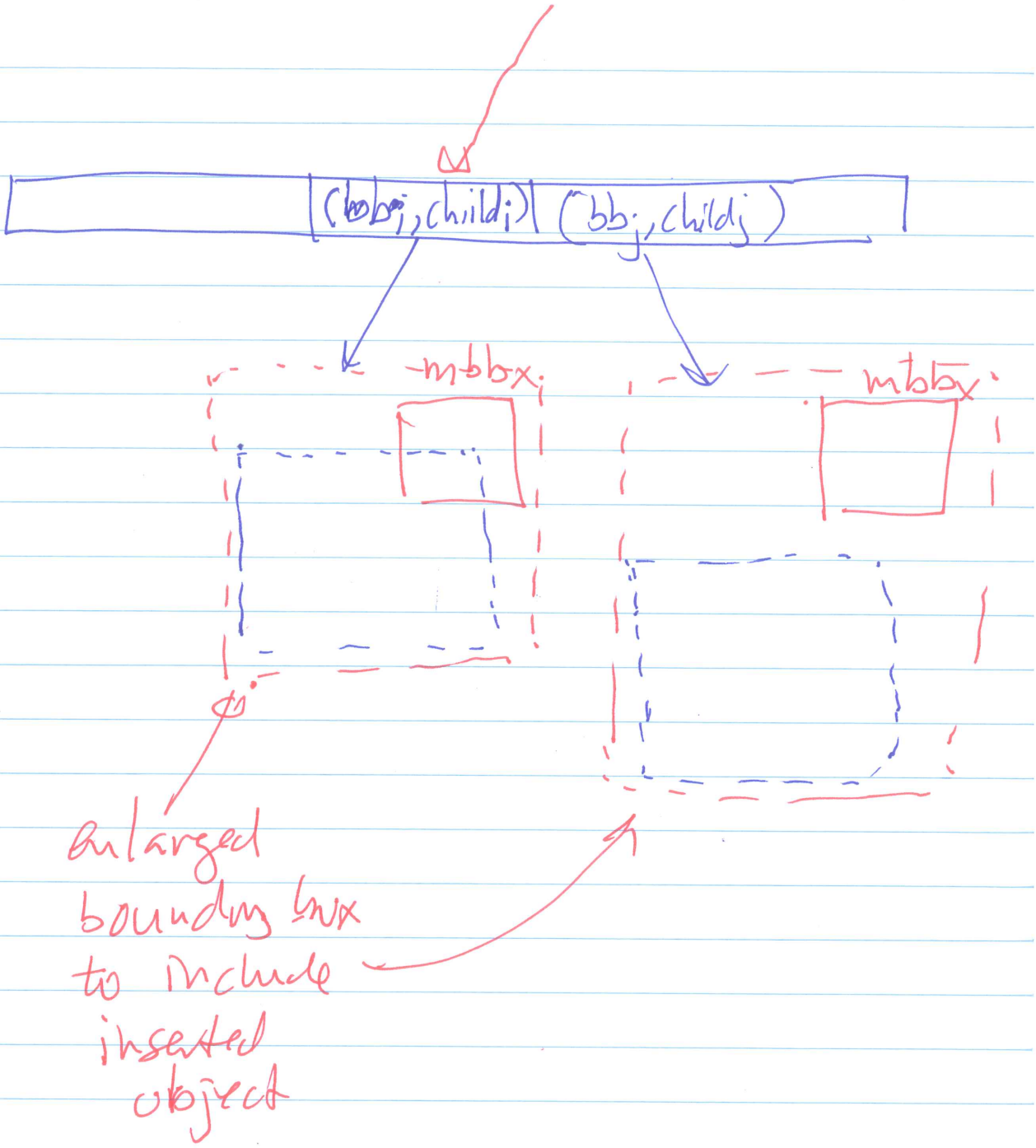
$mbb_x \subseteq bb_i$

mbb_x ← new object.

Insert (mbb_x, $objID_x$) in this subtree.

(2) Diff. case:

There is no rectangle that contains mbb_x completely.



⇒ Pick the child that:

the enlargement needed to include the new object (mbb_x...) has minimal area!!!

- Repeat this procedure until:
  you reach a leaf node.

- Once you reach a leaf node,
  the easy case is:
  (1) Leaf node has space to hold
  (mbb_x, $obj\,ID_x$)
  ⇒ insert object in leaf
  DONE.
  (2) Leaf node is full
  (1) split leaf node.
  (decide on 2 bounding boxes.)
  (2) Move a containing bb up into the parent node

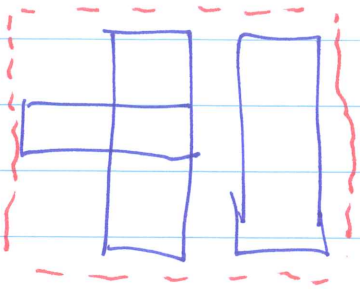
I will be sketchy here

(don't go into:

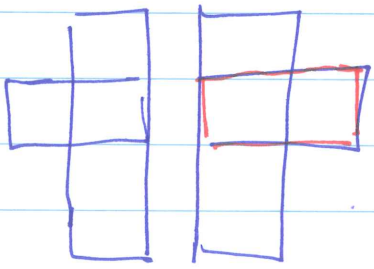
(1) how to split a node.

(Algorithm is complicated).

eg:



Add.

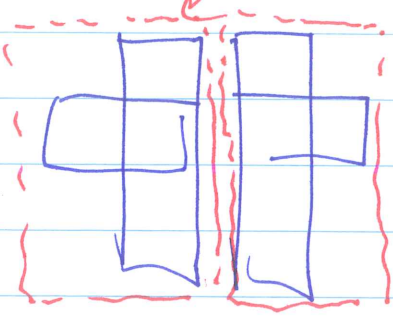


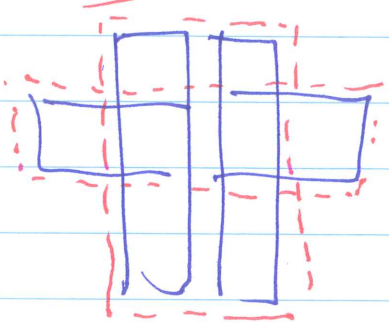
overflow

split

choice 1

choice 2.

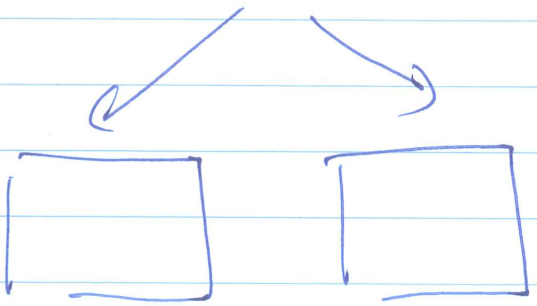




Algorithm??

The "naive" alg. is $O(2^n)$

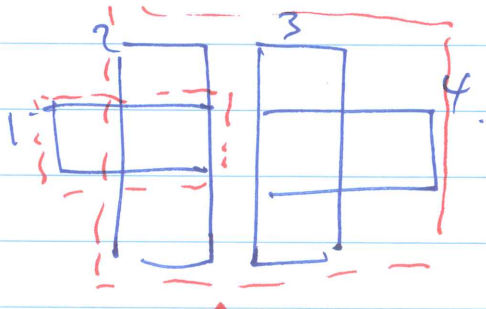
n areas/object



partitioned into 2 bounding boxes

1 object : choices : 2
n objects : # choices $= 2^n$.

eg:



∅	1 2 3 4
1	2 3 4
2	1 3 4
3	1 2 4
4	1 2 3.
12	3 4

and so on.

An $O(n^2)$ alg. for partitioning is available — ~~not~~ discussed.

Use of R-tree:

- R-tree is highly effective for answer this kind of questions:

Given P (region or Point.)

Find Objects that contains P.

(Where am I? → query type.

{ P(x, y) in house }.