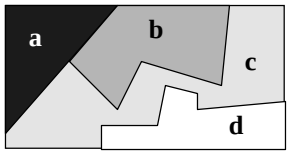
Region-based representations

- A **region map** is a partition of the plane.
  - The components of the partition are called **regions**.
  - Associated with each region is a set of properties, including its name and its type.
  - Regions are comprised of **cells** which are primitive elements with given size(s) and shape(s) and location(s).

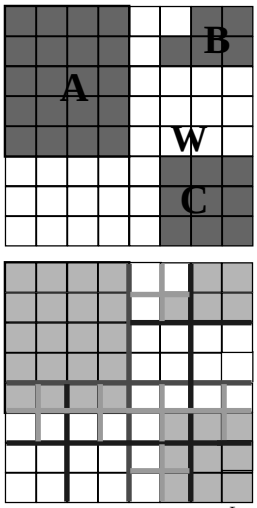


Operations on regions

- Want to develop data structures for region maps that efficiently support a variety of basic operations:
  - **Name queries**: Given a region's name identify all of the cells that belong to the region
  - **Point queries**: Given a point's location, find the region that contain this location. For that region, identify all of the cells in the region.
  - Given two region maps, create a third corresponding to the union, intersection, set difference, etc. of the two given regions.
  - Given a line(segment), find the region(s) through which the line (segment) passes.
  - Range and nearest neighbor queries

Cells

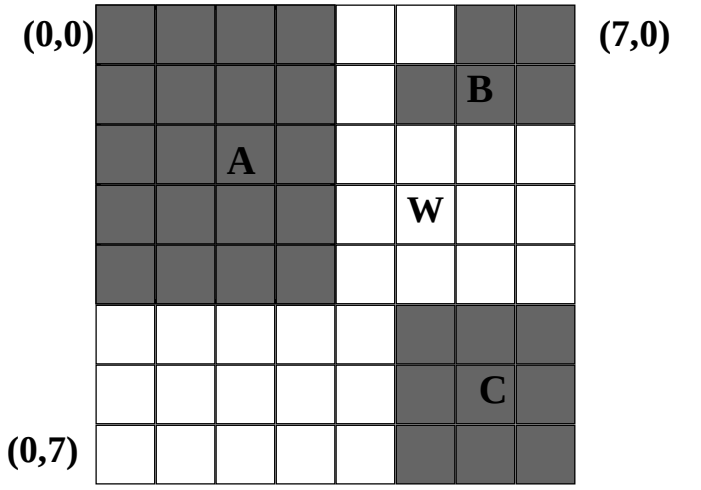
- We will consider two types of region maps:
  - simple arrays
  - quadtrees



Array representations of region maps

- Cells that make up regions are the “pixels” of the array
- Each pixel in the array stores the name of the region to which it belongs
  - actually a pointer to the record that contains descriptive information about the region.
  - For maps with “background” regions, some of these pointers might be null
    - » example: map of lakes

Our example map



Array representations

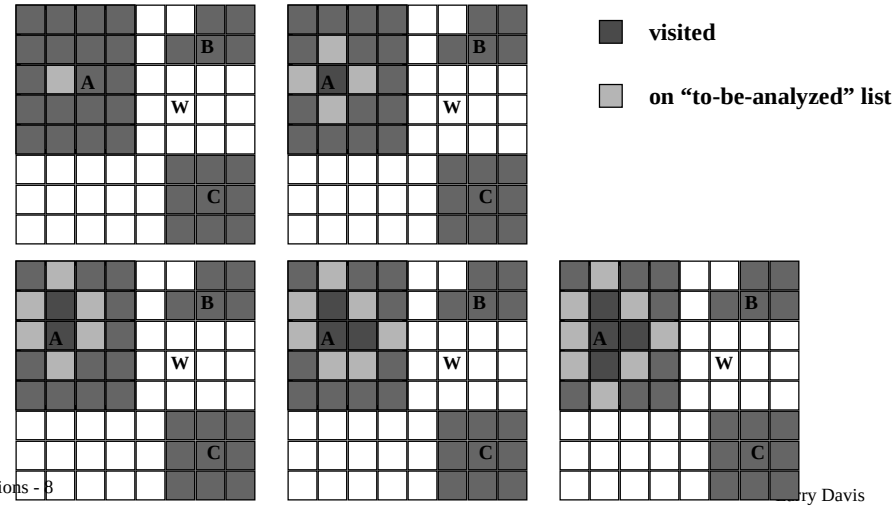
- Need an auxiliary data structure that organizes the **names** of regions - BST, hash table, etc.
  - each record in this data structure contains a pointer to at least one pixel in the array that belongs to the corresponding region
- Given a region
  - Point queries are easy to compute
    - » compute the pixel in which the point lies
      - keep in mind that the pixels correspond to nontrivial square regions, and we might want to locate points represented to finer precision than pixel coordinates.
  - Name queries involve some search of the array to find all of the pixels in a given region

Region labeling algorithm

- Goal: Find all pixels in the region, R, containing location (x,y)
  - Find pixel containing (x,y). Mark this cell as "visited" and
    1. put its address in **output list**
    2. Put its address in the "to-be-analyzed" list.
  - Until the "t0-be-analyzed" list is empty, do the following
    - » take a point, p, off that list
    - » find its four (horizontal and vertical) neighboring cells in the array
    - » For each neighbor of p which
      - has not been previously visited, and
      - is part of object R

    add it to the output list, mark it as visited and put it on the "to-be-analyzed" list

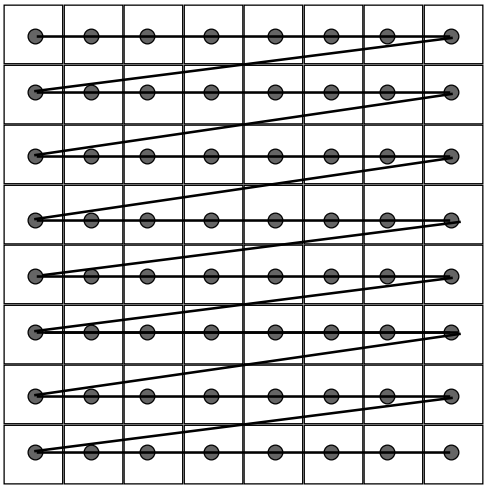
Region labeling - region A



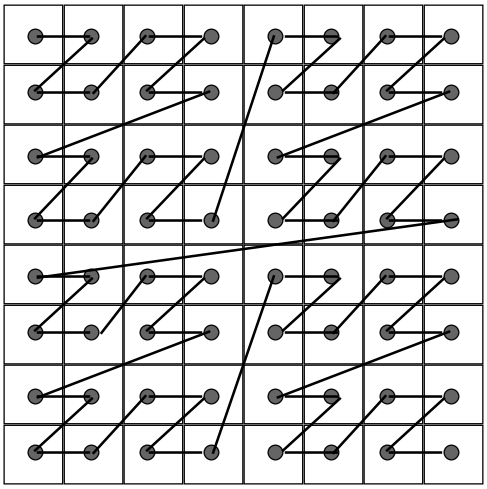
Storage allocation functions for region map arrays

- Can view storage allocation functions as **space filling curves**
  - pass through each position once and only once
    - » lexicographic storage allocation function is one example
    - » but we will impose other requirements so that they can be adapted to addressing more complex cells than pixels - i.e,, blocks

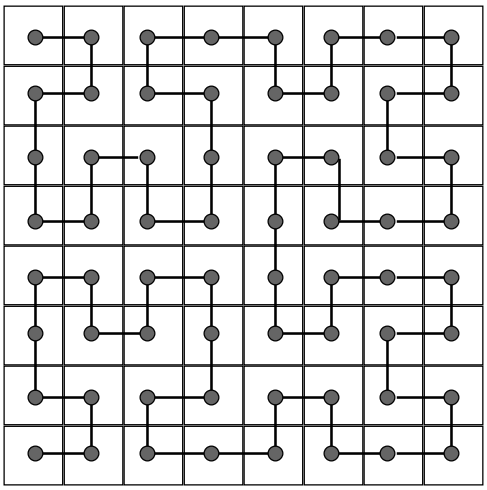
Space filling curves - row major order



Morton order



Peano-Hilbert order



Desireable properties of space filling curves

- If p is a pixel and a is our storage allocation function, then it is simple to compute a(p) and a^{-1}.
  - For the Morton ordering, mapping is obtained by interleaving the bits in the binary representations of the coordinates of the position of a cell.
- Two positions that are adjacent in the 2-D plane are neighbors along the space filling curve
  - impossible to satisfy for all points adjacent in the place
  - For Peano-Hilbert, every cell is a 4-adjacent neighbor of the previous element in the sequence.
- Neighbor finding should be efficient
  - for fast queries

Neighbor finding

- Given a point (x,y) and its code (row-major code, Morton code), find one of its vertical or horizontal neighbors quickly
  - row-major order: formula for sequential allocation of arrays tells us how to advance through the array by one row (keeping column index fixed) or one column (keeping row index fixed)
  - Morton code: separate code into x and y coordinates by bit de-interleaving, then add (subtract) 1 from appropriate coordinate and interleave to obtain Morton code of neighbor

Drawbacks of array access structure

- Large background regions
  - these are the regions never accessed by queries
  - but we would need array elements for each of them
- Redundant representation of objects
  - can represent an object using a smaller number of larger blocks
- Solutions
  - Use the space filling curve to construct an index of cell positions that can be organized using a tree
  - Aggregate large blocks of pixels in single “super-cells”, and index using a generalization of the space-filling curve index.

Binary tree representation of array-based region maps

- Suppose that our map contains large, empty regions
  - then an array is wasteful of storage
- Can represent the map as a binary search tree (maybe a b-tree) of pixels
  - given a pixel, p, we use a(p) as the key for constructing the binary search tree

Region map data structures - array

A1	A2	A3	A4	W1	W2	B1	B2
A5	A6	A7	A8	W3	B3	B4	B5
A9	A10	A11	A12	W4	W5	W6	W7
A13	A14	A15	A16	W8	W9	W10	W11
A17	A18	A19	A20	W12	W13	W14	W15
W16	W17	W18	W19	W20	C1	C2	C3
W21	W22	W23	W24	W25	C4	C5	C6
W26	W27	W28	W29	W30	C7	C8	C9

Binary search trees for Morton ordering

- Create a binary search tree containing a node for each pixel in an object
  - The BST is organized around the Morton order number of a pixel
  - Each node in the BST contains
    - » pointer to record describing region containing corresponding cell
    - » region name
    - » Morton order number (code) for pixel position
- Could also use a B-tree or other hierarchical data structure

Morton codes

(0,0)

y

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	46	58	59	62	63

x

Examples

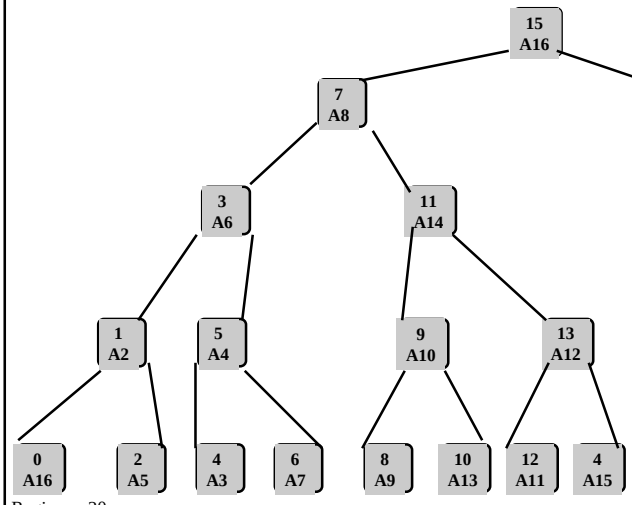
(3,4) = (011, 100) <--> 011010 = 26

(6,3) = (110, 011) <--> 101101 = 45

Regions - 19

Larry Davis

BST for example

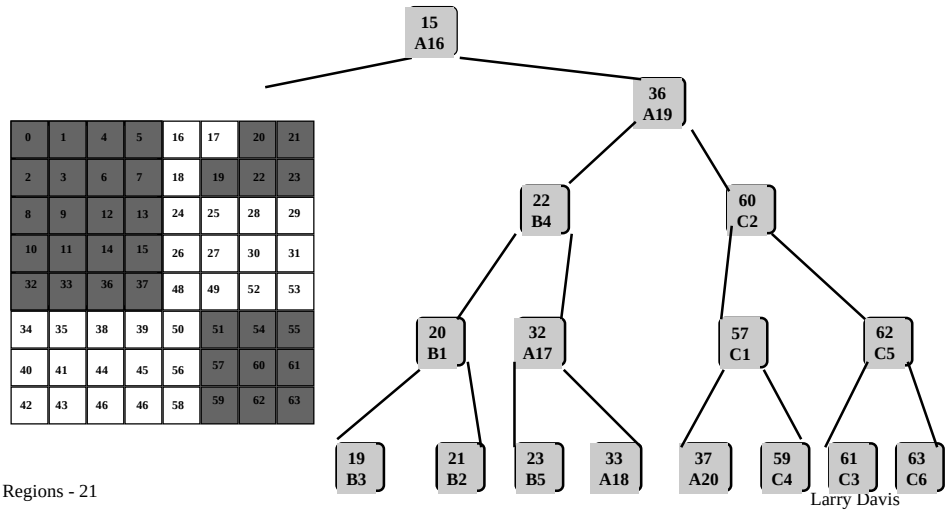


0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	46	58	59	62	63

Regions - 20

Larry Davis

BST for example



Answering queries using the BST

- Point query:
  - identify position of pixel containing given location(x,y)
  - compute the Morton code, M, for that position
  - Search the BST for M - this finds one pixel in the region, O, containing (x,y)
  - Recursively:
    - » construct Morton codes for neighbors of M
    - » search BST for those neighbors; add any contained in O and not previously visited, and recurse on those pixels

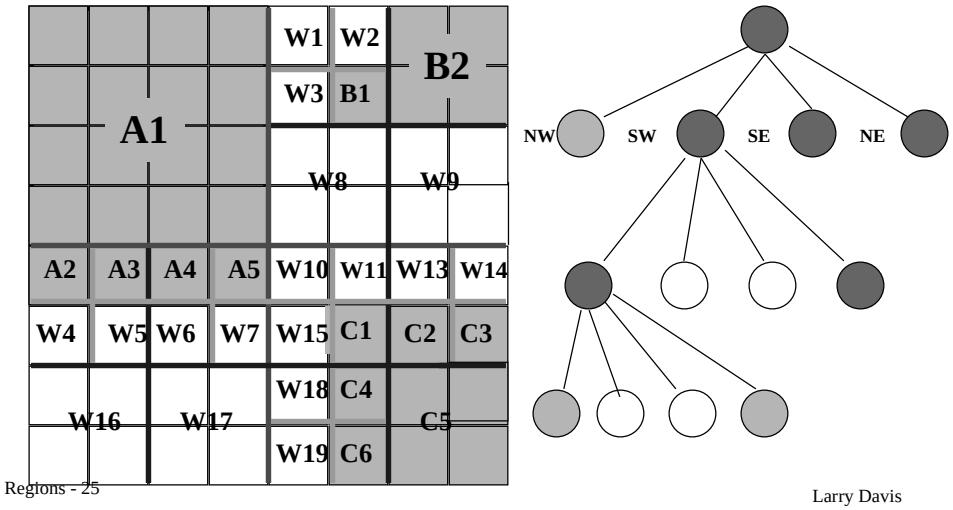
Representations using blocks

- Relax requirement that all cells be the same size
- Two ways to define natural blocks
  - space decomposition process: set of blocks that can be used to represent an object is independent of the object and is determined by the coordinate system of the representation
  - object decomposition process: set of blocks is determined by the object shape, size and position

Regular decompositions

- We require all of the blocks to have the same shape (square) and limit their
  - sizes to powers of 2
  - positions of upper left corners of blocks of size 2^i to coordinates that are multiples of 2^i
- Called a region quadtree
  - construct by a top down process that splits (into 4) any block containing more than one object type
  - construct by bottom-up process that scans pixels in Morton order, constructing blocks on the fly.
- Two ways to represent
  - 4-ary tree
  - binary tree of Morton-coded quadtree blocks

Example

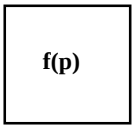


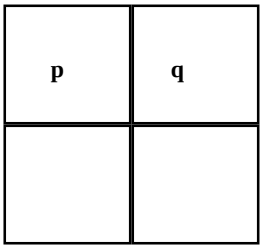
Representations of regions quadtrees

- Tree with fanout 4 (= 2^2)
- Point queries involve a
  - tree traversal to find the block containing the location,
  - and then a neighbor finding step to drive the recursive region labeling algorithm
- Given a node, p, in the quadtree, how can we find its smallest Eastern neighbor, q, whose size, |q|, is at least as large as |p|
  - if |q| > |p|, then q is a terminal node and we can check if it is part of the same object as p
  - if |q| = |p| then q might or might not be a terminal node. If it is not, then it is easy to find all of its westernmost descendants.

Finding the Eastern neighbor of p

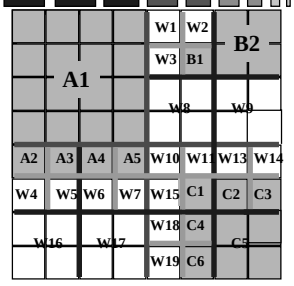
- Suppose p is
  - the NW (or SW) son of its father f(p). Then, the NE (or SE) son of f(p) - NE[f(p)] is p's Eastern neighbor, and f(p) is the common ancestor of p and its Eastern neighbor

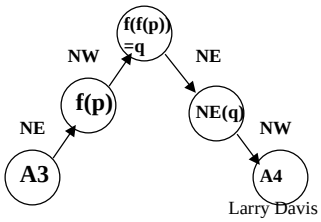




Finding the Eastern neighbor of p

- Suppose p is the NE son of f(p)
  - then, if f(p) is the NW (or SW) son of f(f(p)), then the Eastern neighbor of p is either the NE son of f(f(p)) or the NW son of NE[f(p)] is it exists.
  - Consider A3 (2 levels) and A5 (3 levels) as examples





Finding the Eastern neighbor of p

- More generally
  - acsend the tree as long as the link between a node and its father is a NE or SE link. Stop when a NW or SW link is encountered. This link is from the **lowest common ancestor,** r**,** of p and p's Eastern neighbor
  - maintain the sequence of directions traversed in ascending the tree to r.
  - descend the tree as far as possible using the mirror image moves (taken around the N-S axis) from this sequence
    - » for each NE move, go down NW
    - » for each SE move go down SE ...
  - This will arrive at the smallest Eastern neighbor of p whose size is greater than or equal to |p|

Morton code representations of region quadtrees

- Morton codes are consecutive within a quadtree block
- Can build a BST based on Morton codes of corners of quadtree blocks

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	46	58	59	62	63

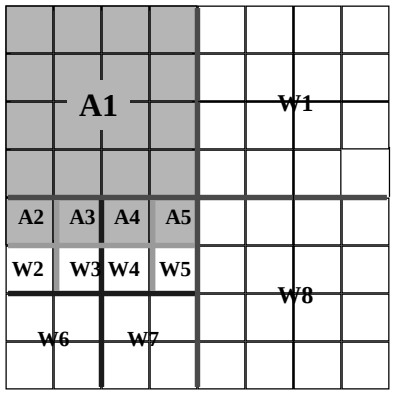
Morton code representations of region quadtrees

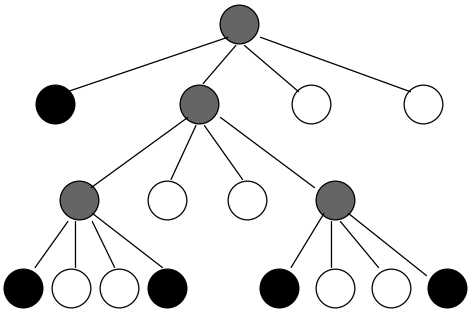
- Lookup is then efficient; given location (x,y)
  - compute its Morton code, m(x,y)
  - let M be the Morton code at the root of the BST, and let s be the size of the corresponding block, B.
    - » (x,y) is in B if M <=m(x,y) <= M + s^2 -1
  - branch left if M > m(x,y)
  - branch right if M < m(x,y)
- Neighbor finding is more complicated

Set operations on quadtrees

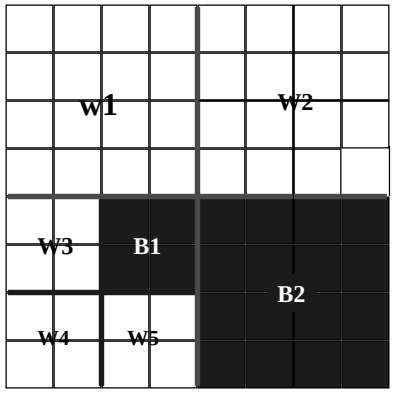
- Suppose we have two region quadtrees - one representing region A and the second region B. How can we efficiently compute the region quadtree representing:
  - $A \cap B$
  - $A \cup B$
- Process is very simple because the quadtrees share lines of decomposition

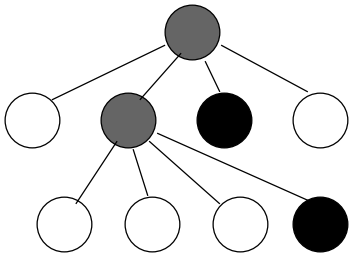
Examples





Examples

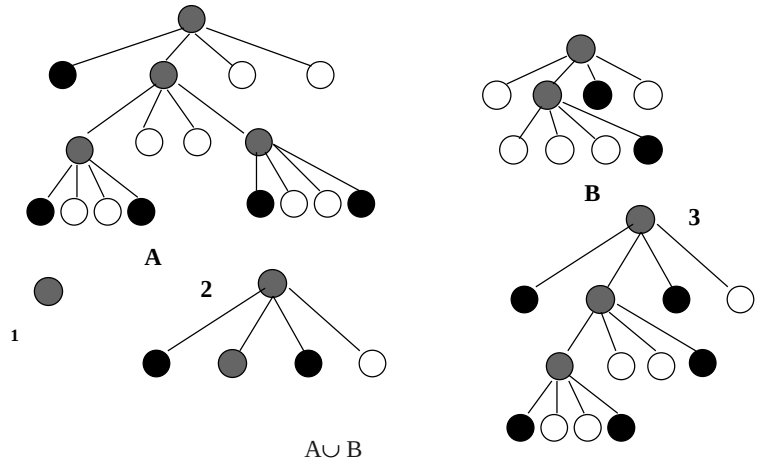


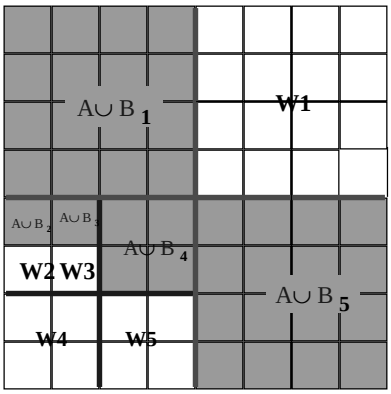
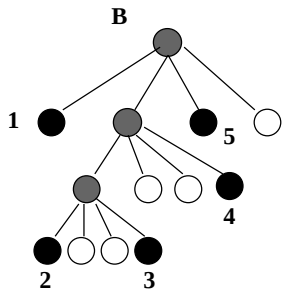


A∪ B

- Traverse both region quadtrees starting from their roots R_A and R_B. Create root of $R_{A \cup B}$
- Compare corresponding descendants of R_A and R_B, dir[R_A] and dir[R_B]
  - if black dir[R_A] or black dir[R_B] then black dir[$R_{A \cup B}$]
  - if white dir[R_A] then dir[$R_{A \cup B}$] <== dir[R_B]
  - if white dir[R_B] then dir[$R_{A \cup B}$] <== dir[R_A]
  - otherwise, dir[$R_{A \cup B}$] is mixed and we descend one level in R_A and R_B.
  - If we create four black brothers, we must collapse them to their father

A∪ B



A∪ B

A∪B