# CS 315 Data Structures Fall 13

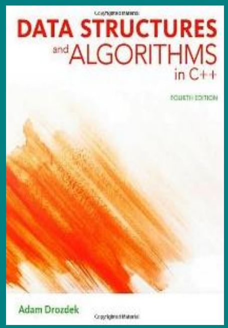
Instructor:

B. Ravikumar
Office: 116 I Darwin Hall
Phone: 664 3335
E-mail: cs315spring11@gmail.com

Course Web site:

http://piazza.com

Textbook:




Data Structures and Algorithms in C++ by Drozdek, Cengage Publishing.

Course Schedule

Lecture:
M W 8:30 – 9:45, Stevenson 3039

Lab:
M 5 to 7:50, Darwin 28

# Data Structures – central themes

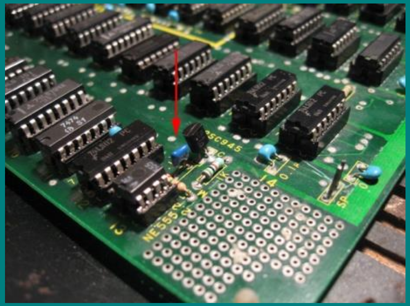
- (more advanced) programming
  - larger problems (than what was studied in CS 215)
  - advanced features like recursion, classes and library
  - new data types (images, audio, video, text)
  - algorithm design

- performance issues
  - comparison of algorithms
  - time, storage requirements

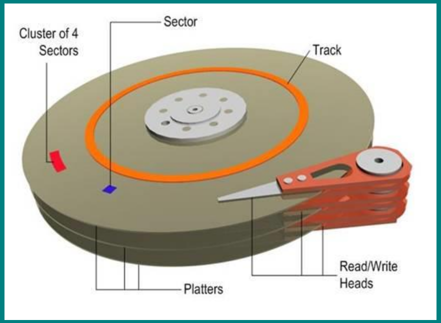
- applications
  - image processing
  - data/image compression
  - web search, parsing
  - board games
  - graphs and their algorithms

# Data Structures – the central issues

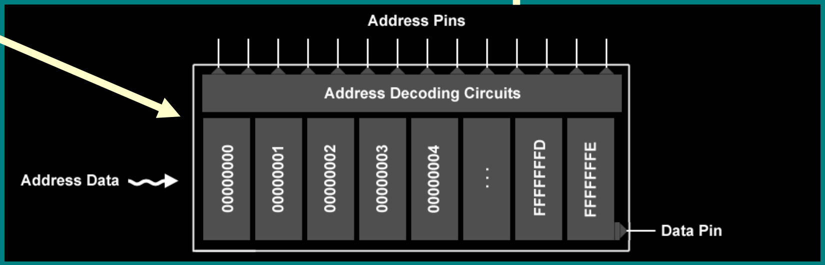
How to organize data in memory so that the we can solve the problem most efficiently?



CPU




Hard disk




RAM

# Data Structure – the central issues

Topics of concern:

- software design, problem solving and applications

- reactive programs:
  - query response

- trade-offs between operations

- efficiency, simplicity, scalability, …

# Course Goals

- Learn to use fundamental data structures:
  - arrays, linked lists, stacks and queues
  - hash table
  - priority queue
  - binary search tree

- Improve programming skill
  - recursion, classes, algorithm design, implementation
  - build projects using different data structures

- Analytical and experimental analysis
  - quantitative reasoning about the performance of algorithms (time, storage, etc.)
  - comparing different data structures

# Course Goals

- Projects/applications:
  - image storage, manipulation
  - Image labeling
  - compression (text, image, etc.)
  - computing with unlimited size integers
  - Discrete-event simulation
  - Index generation
  - Geometric problems
    - image rotation, image stitching
  - Game tree search
  - evaluating arithmetic expression

# Data Structures – key to software design

- Data structures play a key role in every type of software.

- Data structure deals with how to store the data internally while solving a problem in order to
    - Optimize
        - the overall running time of a program
        - the response time (for queries)
        - the memory requirements
        - other resources (e.g. band-width of a network)
        - Simplify software design
        - make solution extendible, more robust

# Abstract vs. concrete data structures

- Abstract data structure (sometimes called ADT -> Abstract Data Type) is a collection of data with a set of operations supported to manipulate the structure

- Examples:
  - stack, queue insert, delete
  - priority queue insert, deleteMin
  - Dictionary insert, search, delete

- Concrete data structures are the implementations of abstract data structures:
  - Arrays, linked lists, trees, heaps, hash table

- A recurring theme: Find the best mapping between abstract and concrete data structures.

# Abstract Data Structure (ADT) container supporting operations

- Dictionary
  - search
  - insert
  - Delete

  primary operations

  - deleteMin
  - Range search
  - Successor
  - Merge

  secondary operations

- Priority queue
  - Insert
  - deleteMin

  primary operations

  - Merge, split etc. Secondary operations

# Linear data structures

- key properties of the (1-dim.) array:
  - a sequence of items are stored in consecutive physical memory locations.
  - main advantage: array provides a constant time access to k-th element for any k.
    (access the element by: Element[k].)
  - other operations are expensive:
    - Search
    - Insert
    - delete

# 2-dim. arrays

- Used to store images, tables etc.
- Given row number r, and column number s, the element in A[r, s] can be accessed in one clock cycle.
- (usually row major or column major order is used.)
- Other operations are expensive.
- Sparse array representation
  - Used to compress images
  - Trade-offs between storage and time

# Linked lists

- Linked lists:

  order is important

  - Storing a sequence of items in non-consecutive locations of the memory.
  - Not easy to search for a key (even if sorted).
  - Inserting next to a given item is easy.
  - Array vs. linked list:
    - Don't need to know the number of items in advance. (dynamic memory allocation)
    - disadvantages

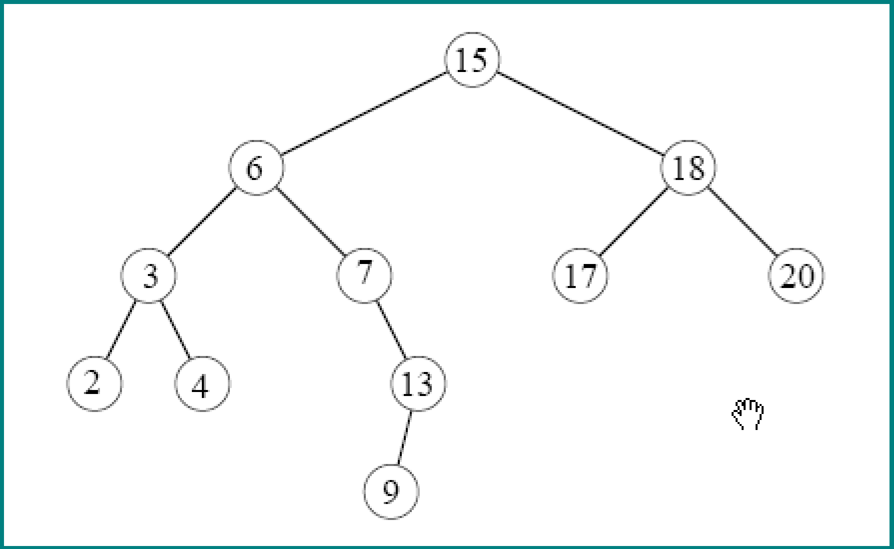
# stacks and queues

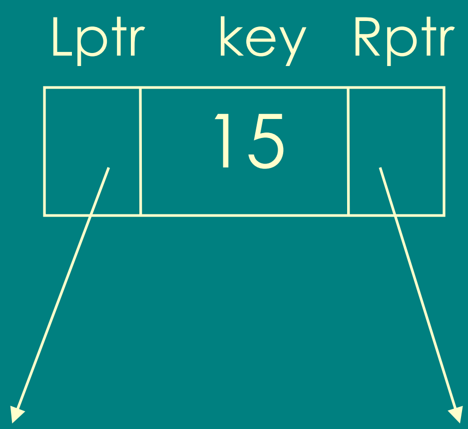
- stacks:
  - insert and delete at the same end.
  - equivalently, last element inserted will be the first one to be deleted.
  - very useful to solve many problems
    - Processing arithmetic expressions

- queues:
  - insert at one end, deletion at the other end.
  - equivalently, first element inserted is the first one to be deleted.

# Non-linear data structures

- Various versions of trees
  - Binary search trees
  - Height-balanced trees etc.




Main purpose of a binary search tree → support dictionary operations efficiently

# Priority queue

- Max priority key is the one that gets deleted next.
  - Equivalently, support for the following operations:
    - insert
    - deleteMin
- Useful in solving many problems
  - fast sorting (heap-sorting)
  - shortest-path, minimum spanning tree, scheduling etc.

# Hashing

- Supports dictionary operations very efficiently (most of the time).

- Main advantages:
  - Simple to design, implement
  - on average very fast
  - not good in the worst-case.

# Applications

- arithmetic expression evaluation
- data compression (Huffman coding, LZW algorithm)
- image segmentation, image compression
- backtrack searching
- finding the best path to route in a network
- geometric problems (e.g. rectangle area)

# Some projects from past semesters

- Unlimited precision arithmetic:
  Compute values like 1000! =

4 023 872 600 770 937 735 437 024 339 230 039 857 193 748 642 107 146 325 ⋱
437 999 104 299 385 123 986 290 205 920 442 084 869 694 048 004 799 ⋱
886 101 971 960 586 316 668 729 948 085 589 013 238 296 699 445 909 ⋱
974 245 040 870 737 599 188 236 277 271 887 325 197 795 059 509 952 ⋱
761 208 749 754 624 970 436 014 182 780 946 464 962 910 563 938 874 ⋱
378 864 873 371 191 810 458 257 836 478 499 770 124 766 328 898 359 ⋱
557 354 325 131 853 239 584 630 755 574 091 142 624 174 743 493 475 ⋱
534 286 465 766 116 677 973 966 688 202 912 073 791...

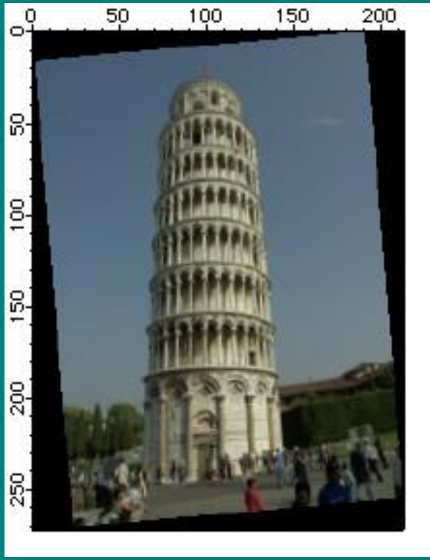
(concepts : recursion, linked list)

# Some projects from past semesters

- Image manipulation: (concept: arrays, library, algorithm analysis)

# image manipulation

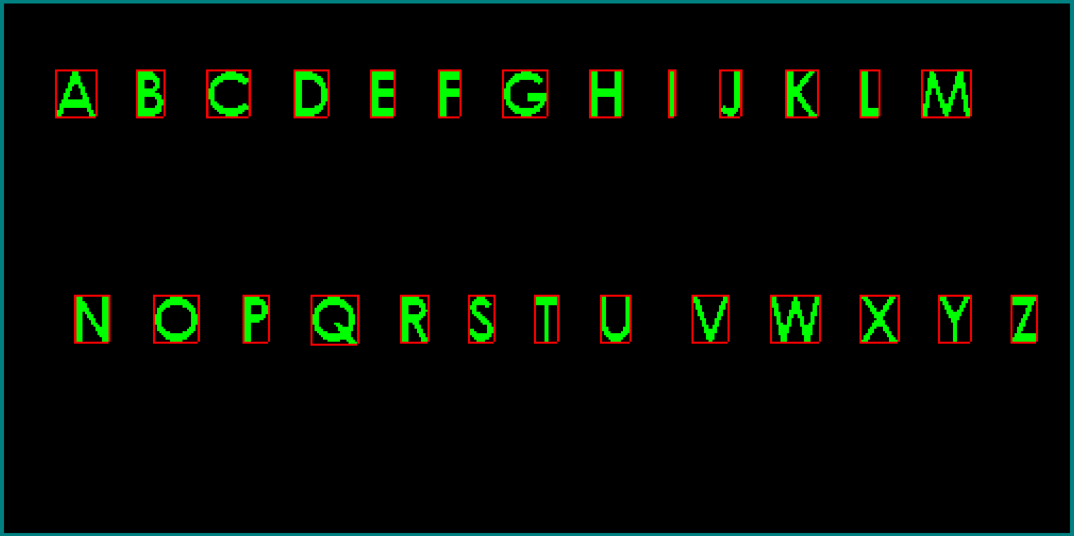
# Image rotation



- Bounding box construction: OCR is one of the early success stories in software applications.

Scan a printed page and recognize the characters in it.

First step: bounding box construction.



# Final step:

## Input:

In 1830 there were but twenty-three miles of railroad in operation in the United States, and in that year Ken-tucky took the initial step in the work west of the Alleghanies. An Act to incorporate the Lexington & Ohio Railway Company was approved by Gov. Metcalf, January 27, 1830. It provided for the construction and re-

Output: "In 1830 there were but twenty-three miles of railroad in operation in the United States, and in that year Kentucky took … "

- Spelling checker: Given a text file T, identify all the misspelled words in T.

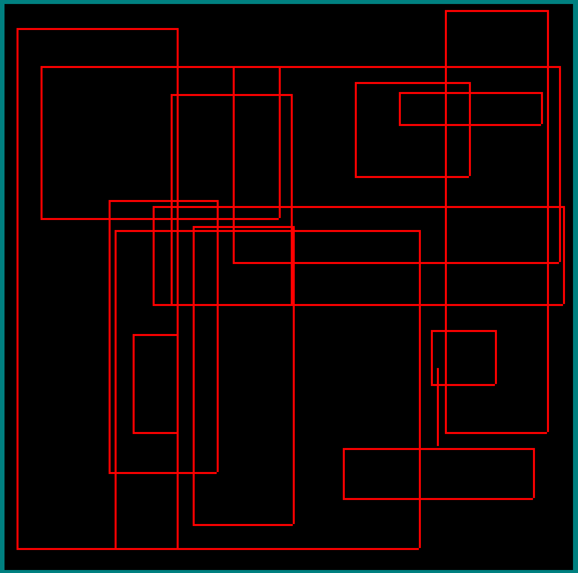
Idea: build a hash table H of all the words in a dictionary, and search for each word of the text T in the table H. For each misspelled word, suggest the correct spelling.

(hashing, strings, vectors)

- Peg solitaire (backtracking, recursion, hash table)

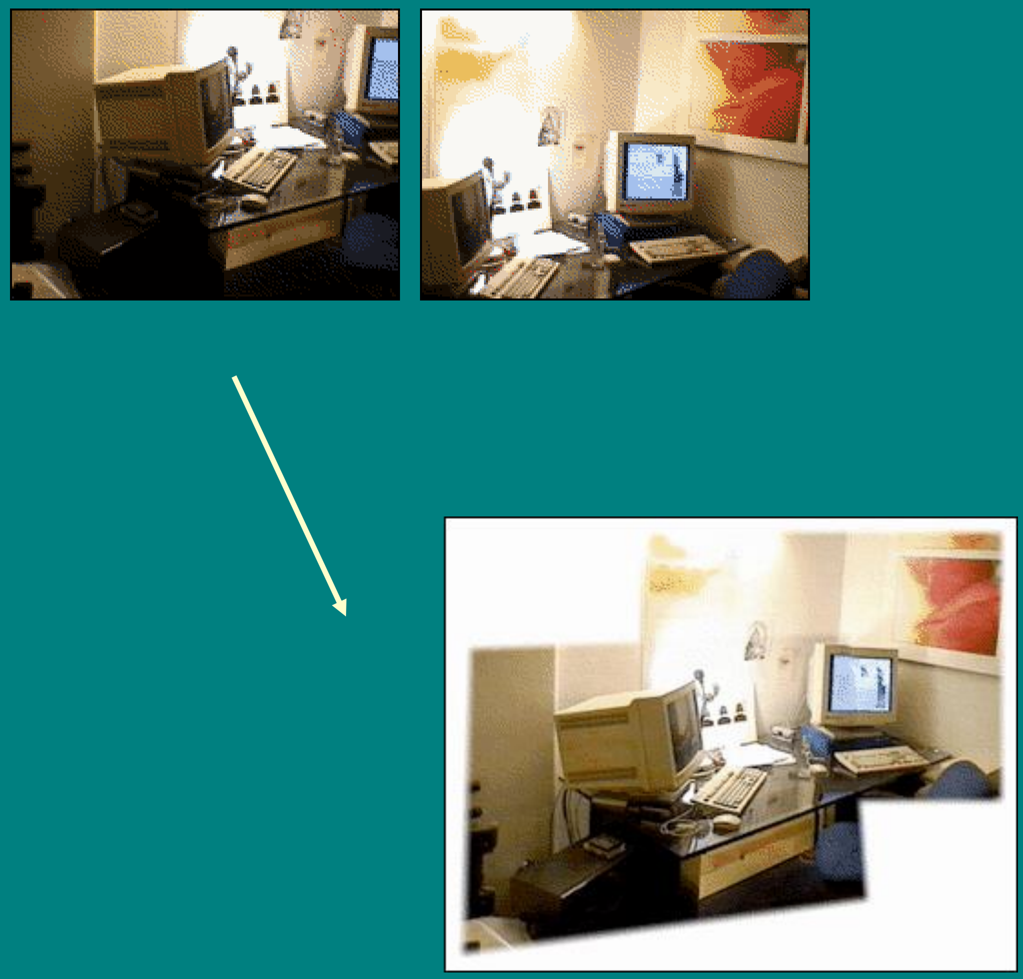
Find a sequence of moves that leaves exactly one peg on the board. (starting position can be specified. In some cases, there may be no solution.)

- Geometric computation problem – given a set of rectangles, determine the total area covered by them. Trace the contour, report all intersections etc.



Data structure: binary search tree.

• Given two photographs of the same scene taken from two different positions, combine them into a single image.



# Image compression

(Quadtree data structure)


original

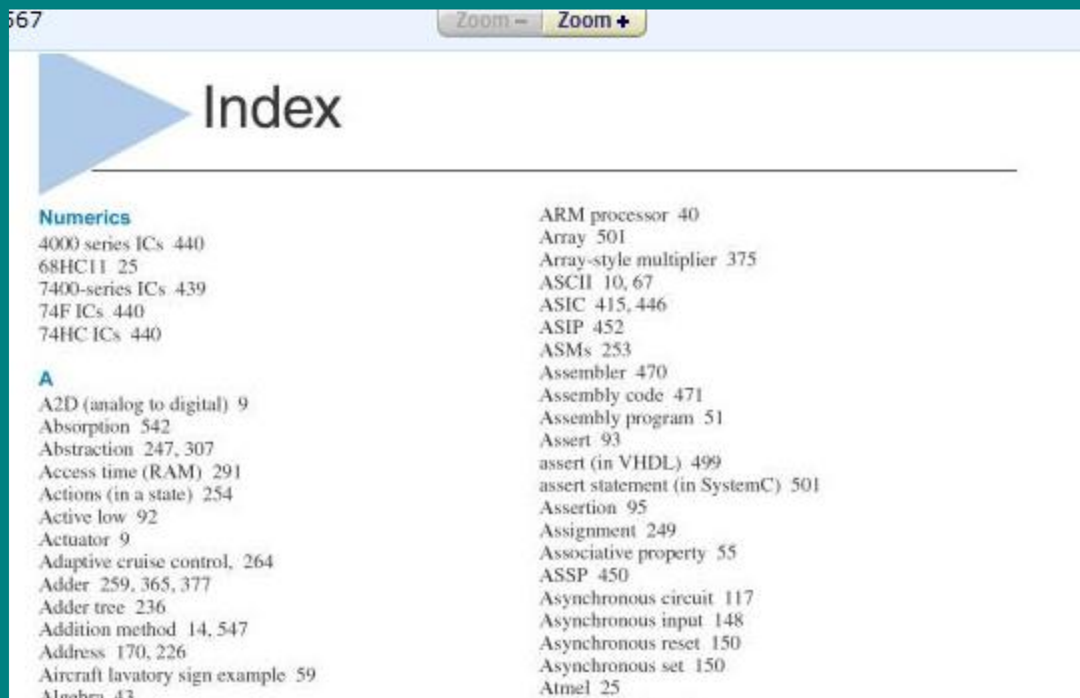

(compressed x10)

(compressed x 50)

# Index generation for a document

Index contains the list of all the words appearing in a document, with the line numbers in which they appear.

Typical index for a book looks:

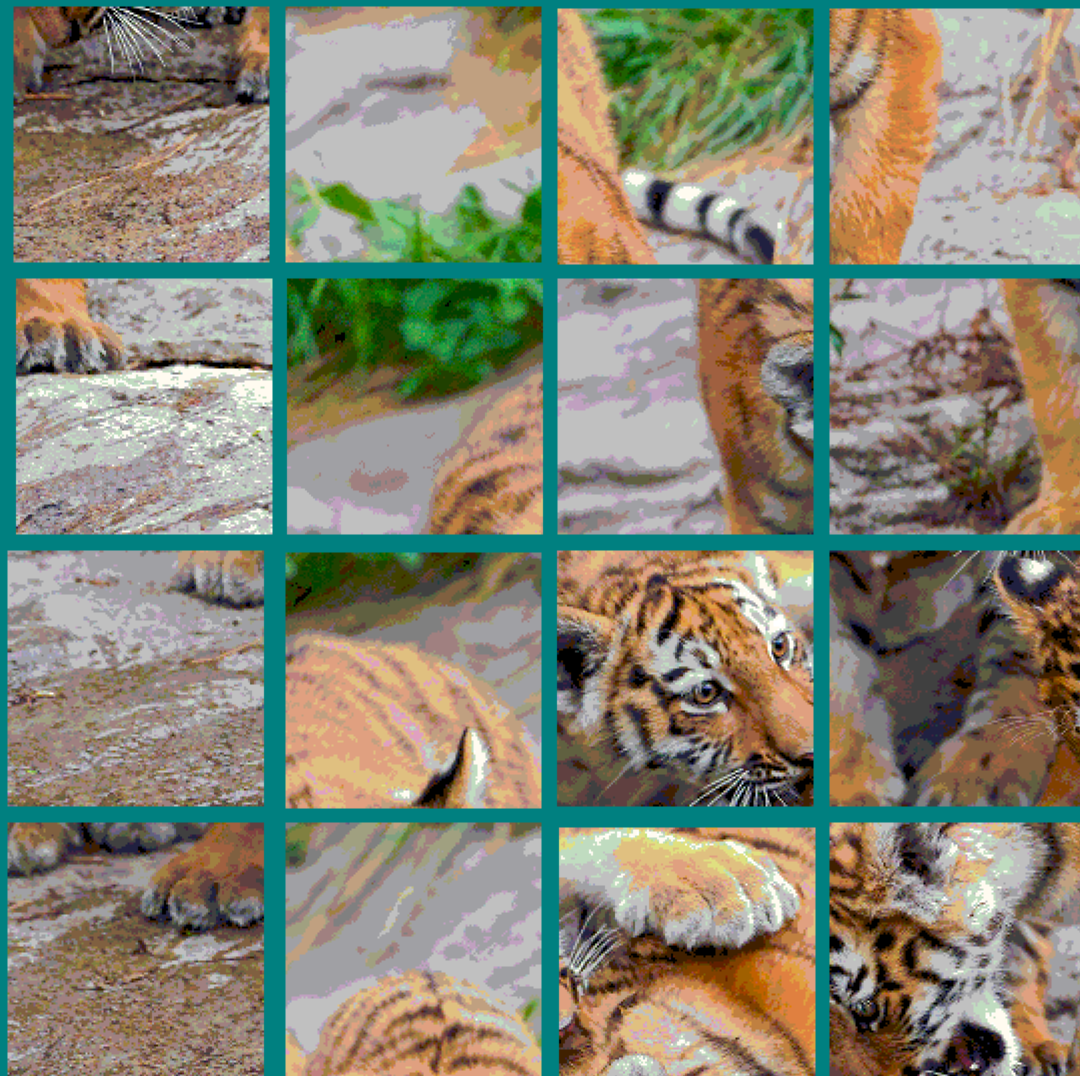



## Index

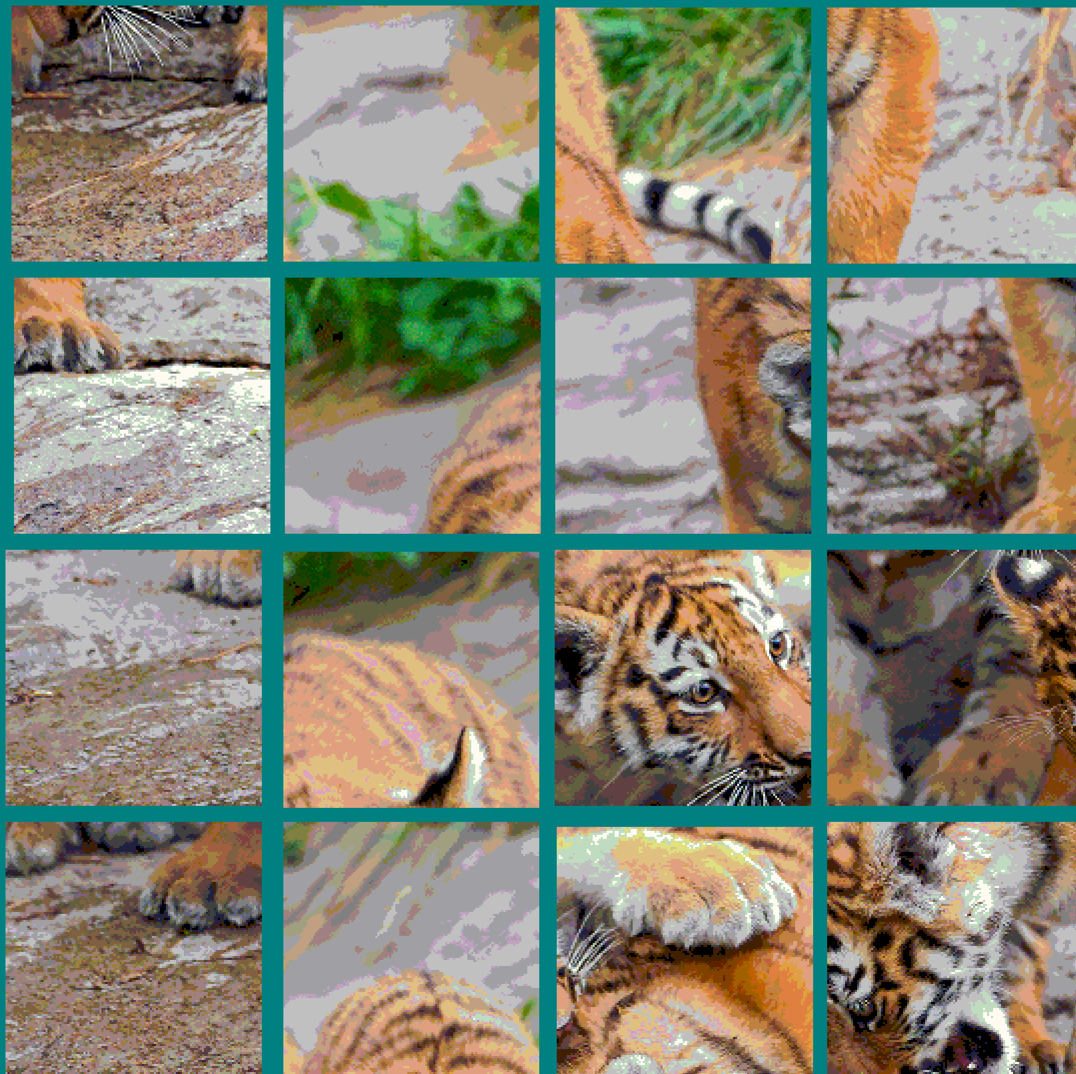


Data structure binary search tree, hash table

# Image stitching



# Image stitching



solution