Programming, Problem Solving, and Algorithms

CPSC203, 2019 W1

Announcements

Project 1 is released. Due 11:59p, Oct 17.

“Problem of the Day” continues!

Today:

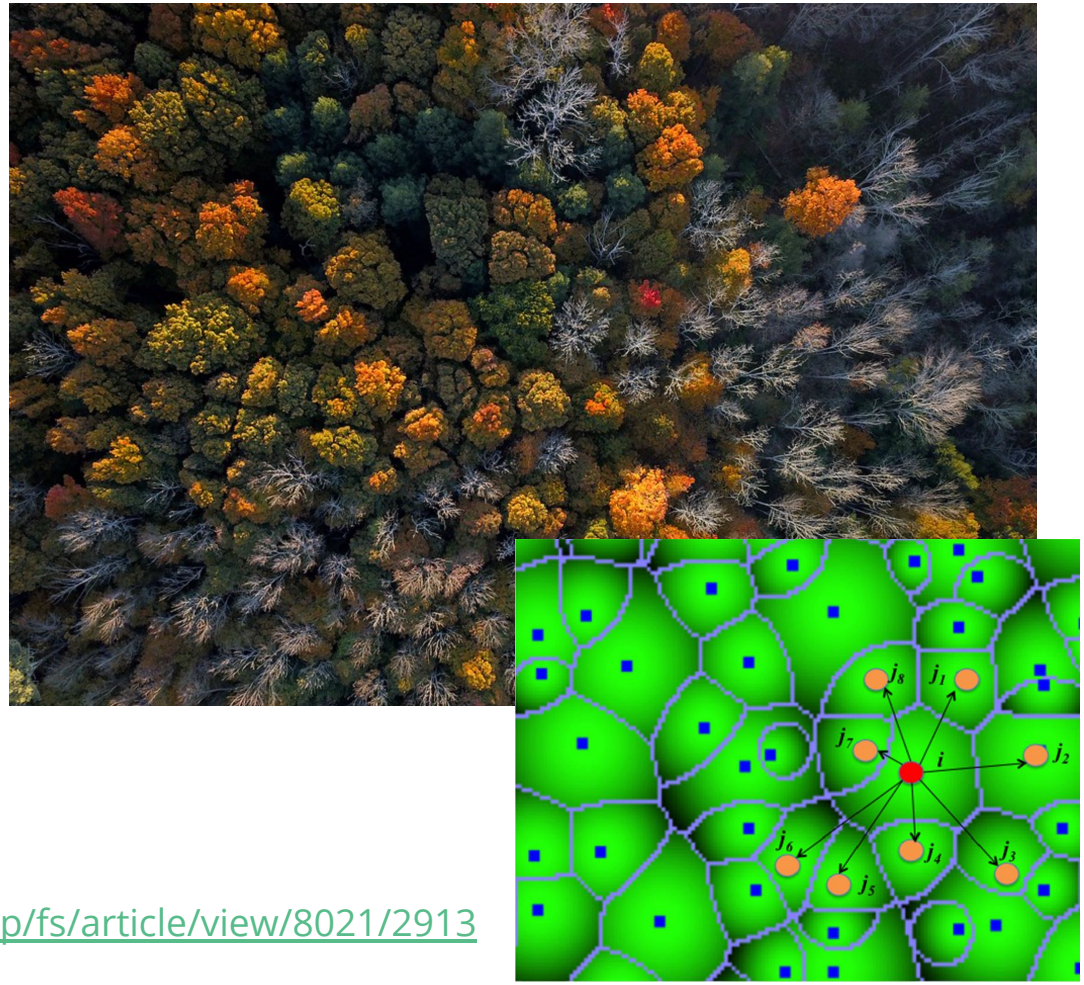
Voronoi Diagrams

Voronoi Diagrams

Given a (finite) set of “centers” c_1, c_2, … c_k, a Voronoi region, R_j consists of the set of points nearer to center c_j, than to any other center.

Together, the R_j regions compose the Voronoi Diagram of a plane.

The applications of this structure go far beyond our coffee fix!!

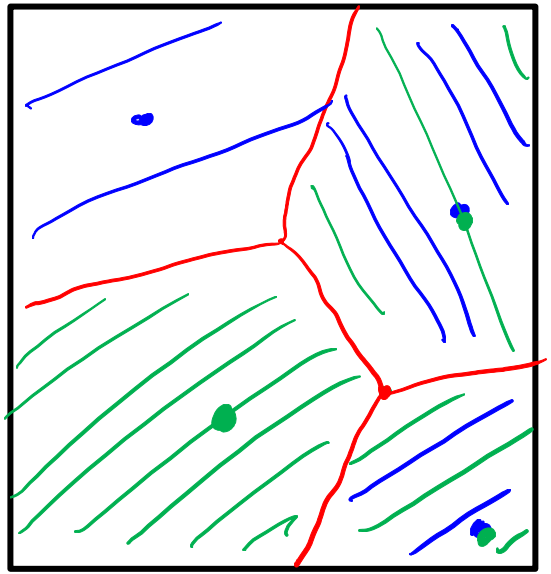


http://revistas.inia.es/index.php/fs/article/view/8021/2913

Pointillism

A Sunday on La Grande Jatte, Georges Seurat

The Idea...



1) Select a subset of points from the original image. How?
2) Use those points, with their colors, as centers in a new image of the same size.
3) Build the voronoi diagram in the new image, using the colors for the original image.

The quality of the new image

depends on # of points chosen.

density of points.

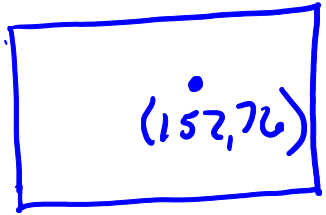
Planning

Point: Tuple[int, int]: (width, height) bounds on values are $x \in [0..width-1]$ $y \in [0..height-1]$

Color: Tuple[int, int, int]: (r, g, b) $r \in [0..255]$ $g \in [0..255]$ $b \in [0..255]$

Center: point + a color

Centers:



choose x from 0..width → 152
choose y from height → 76
[density * width] * [density * ht]
# of points. density is a %

Image: Image

# of pixels is (density)^2 * size of image

Planning

Data flow:

1) Read in img we are simulating - photo
2) Choose the collection of centers

gather inputs

3) Build new image - for each pt find color of nearest ctr + place it.
4) Write out new image.

Demo and Analysis

orig (.1) # of ctrs .01 * size
new (.25) # of ctrs .0625 * size

https://github.students.cs.ubc.ca/cpsc203-2019w-t1/LecVor

w width # of centers c
h height

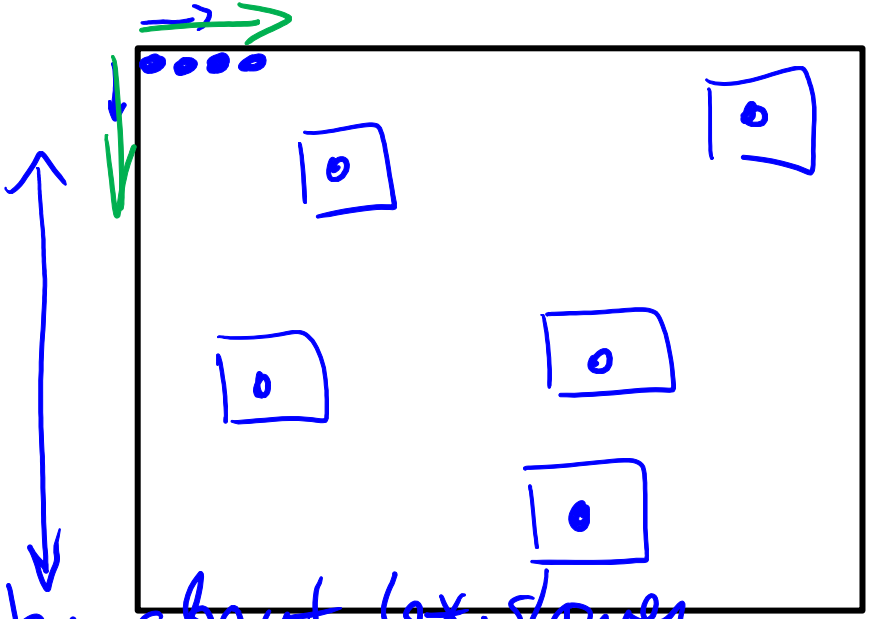
How much work is done?

1) Read image: → w * h

2) Choose centers: c

3) Build new image: c * w * h

4) Write out new image: w * h

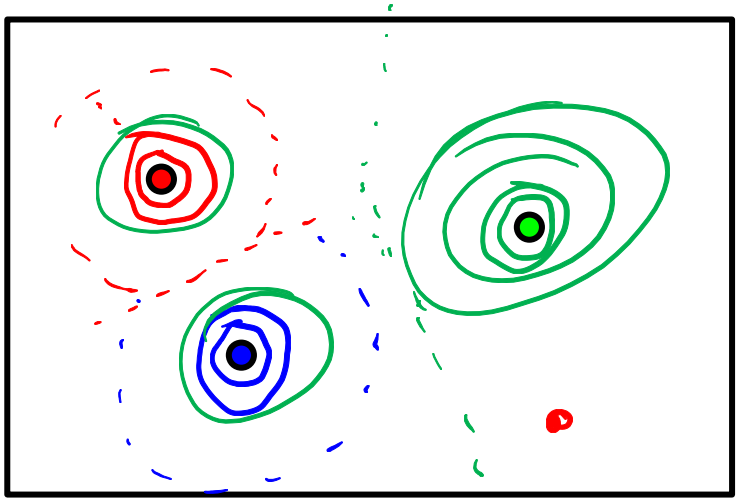


new Img (finer one) will be about 6x slower.

Can we do better?

The running time of the original algorithm: is dominated by $c \times w \times h$ (largest term)

What would be better? $w * h$



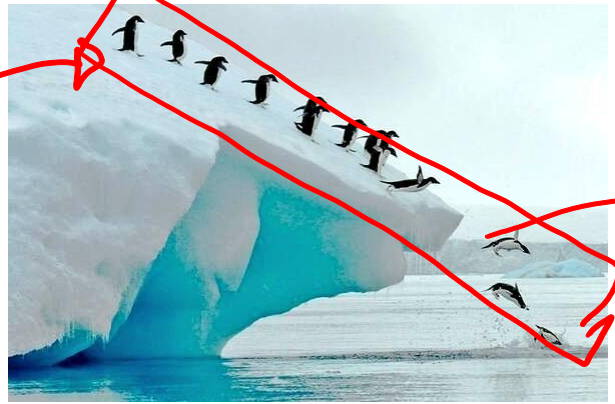
Orchestrate a fill from each center, growing out at the same rate.

Each pixel is processed exactly once, not once per center as before.

This means we can have lots of centers!

Data Structure: Queue

To orchestrate the fill, we'll use a data structure called a QUEUE.



enqueue

dequeue

First in First Out structure

Queue:

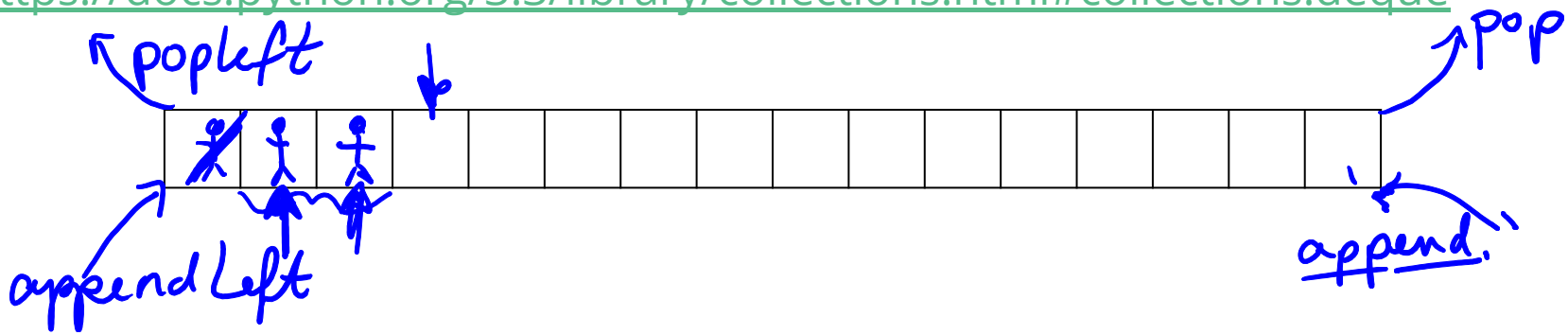
enqueue(k) -- places data k onto the structure, at the "end" insert

dequeue() -- removes and returns the "first" element from the structure

Queues in Python

There is no built-in Queue type in Python3, so we use a `deque`. doubly ended queue

https://docs.python.org/3.3/library/collections.html#collections.deque



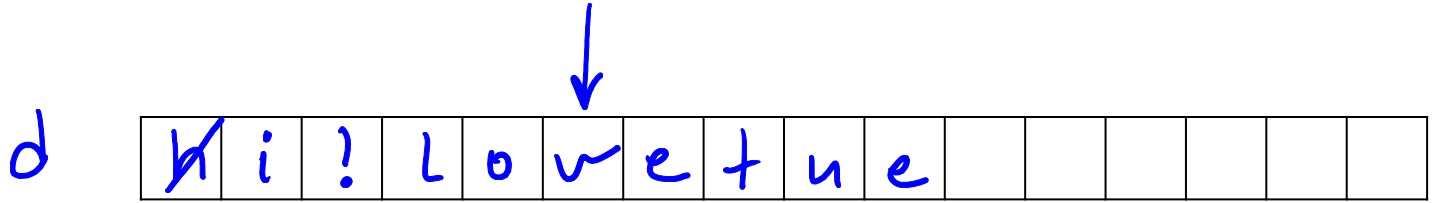
Queue:

enqueue(k) -- append

dequeue() -- popLeft

} to avoid shifting data

Example:



standard in Python3

```
from collections import deque

d = deque()
d.append('h')
d.append('i')
d.append('!')
print(d.popleft())
```

enqueue

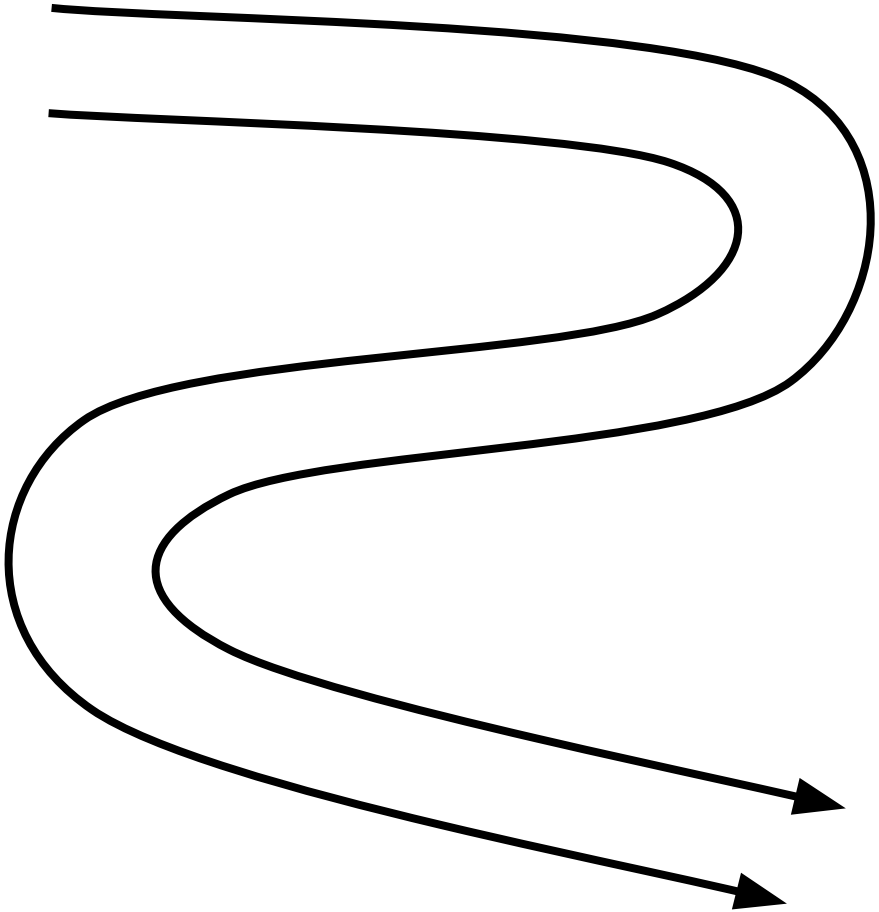
dequeue

→ h

Putting it together

00	10	20	30	40	50	60	70	80	90
01	11	21	31	41	51	61	71	81	91
02	12	22	32	42	52	62	72	82	92
03	13	23	33	43	53	63	73	83	93
04	14	24	34	44	54	64	74	84	94
05	15	25	35	45	55	65	75	85	95

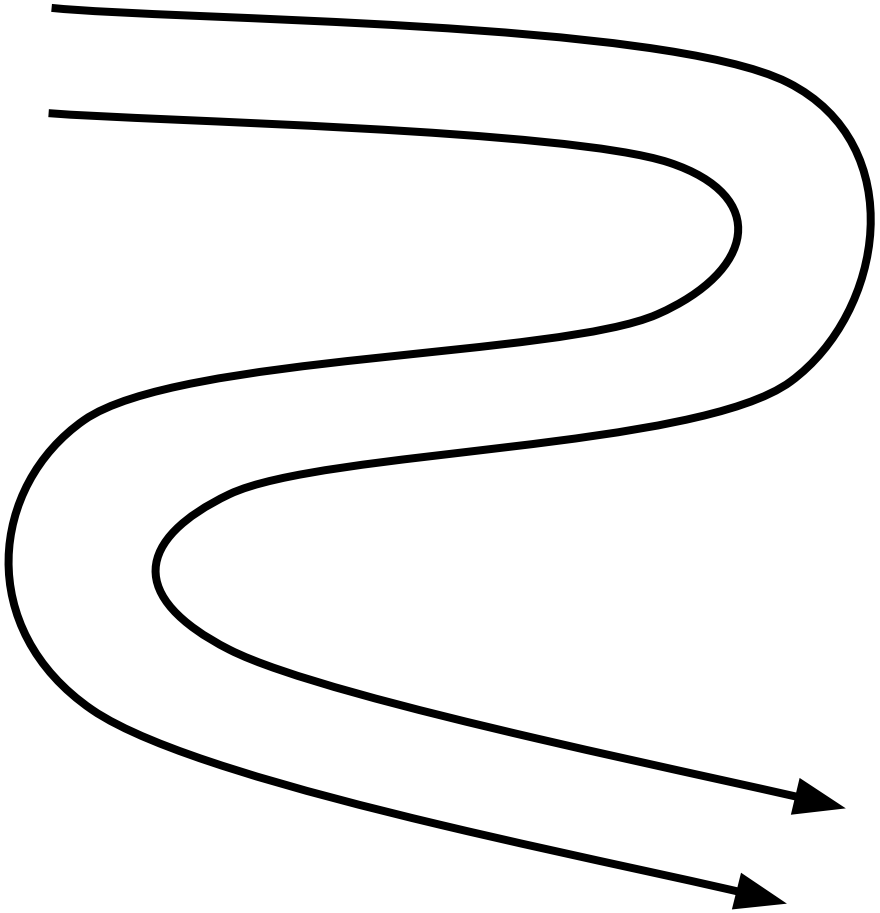
1) enqueue the center to start
2) while the queue is not empty:
   a) v = dequeue
   b) for each valid neighbor w, of v:
      i) color w
      ii) enqueue w



Putting it together

00	10	20	30	40	50	60	70	80	90
01	11	21	31	41	51	61	71	81	91
02	12	22	32	42	52	62	72	82	92
03	13	23	33	43	53	63	73	83	93
04	14	24	34	44	54	64	74	84	94
05	15	25	35	45	55	65	75	85	95

1) enqueue the center to start
2) while the queue is not empty:
   a) v = dequeue
   b) for each valid neighbor w, of v:
      i) color w
      ii) enqueue w



POTD #14 Tue

https://github.students.cs.ubc.ca/cpsc203-2019w-t1/potd14

Describe any snags you run into:

1. Line ___: _____
2. Line ___: _____
3. Line ___: _____
4. Line ___: _____
5. Line ___: _____

ToDo for next class...

POTD: Continue every weekday! Submit to repo.

Reading: TLACS Ch 10 & 12 (lists and dictionaries)

References:

https://en.wikipedia.org/wiki/Voronoi_diagram