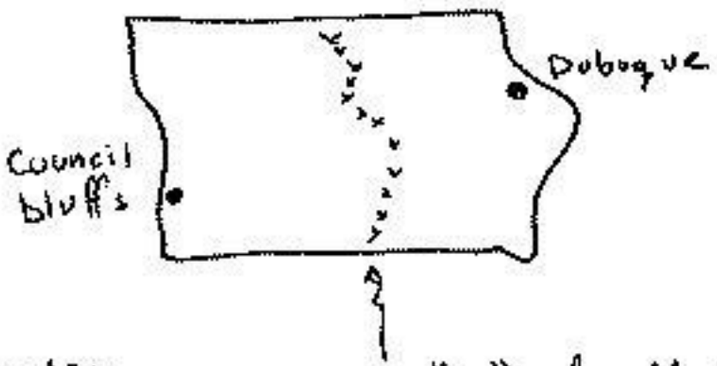
Problem:

Parallel Computation of Min-Distance Routes (Shortest Paths)

Calculate shortest paths between all locations.



Strategy — a "cut" of all roads along a line.

Let $x_1, x_2, \ldots, x_m$ be the points along the line;

Claim: Shortest (CouncilBluffs, Dubuque) =

$$\min_{k \in \{1,\ldots,m\}} \text{Shortest}(\text{CouncilBluffs}, x_k) + \text{Shortest}(x_k, \text{Dubuque})$$

proof: by contradiction.

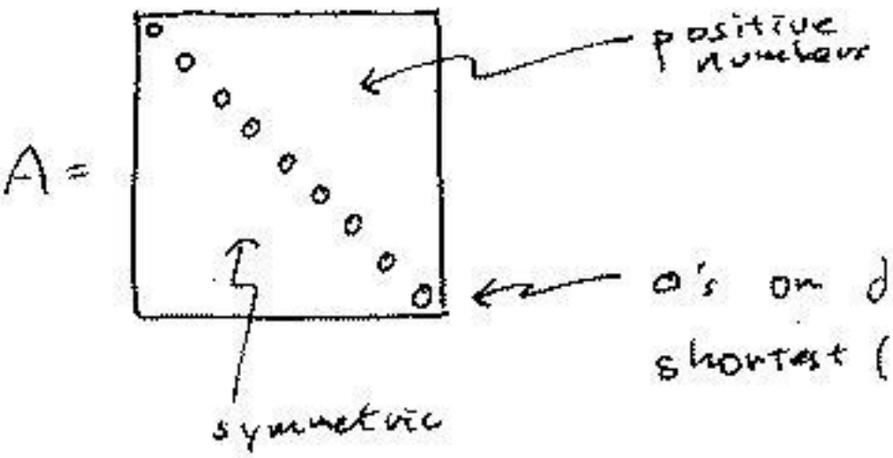
In fact, we can use a "shortcut" in some cases.

Suppose we know some path from a to b is 25 mi. Then, in computing

$$\min_k \text{shortest}(a, x_k) + \text{shortest}(x_k, b)$$

we can ignore any x_k for which $\text{shortest}(a, x_k) > 25$.

Usually this problem has a matrix representation:



Define a kind of "matrix multiply"

$A \times A$ by $(A \times A)[i][j] =$

$$\min_k \left(A[i][k] + A[k][j] \right)$$

Theorem

$A^n[i][j]$ is shortest (i, j)

where $n \geq$ rank (A)

↑ # rows / locations

How to compute A^n ?

First solve A^2 problem in parallel:

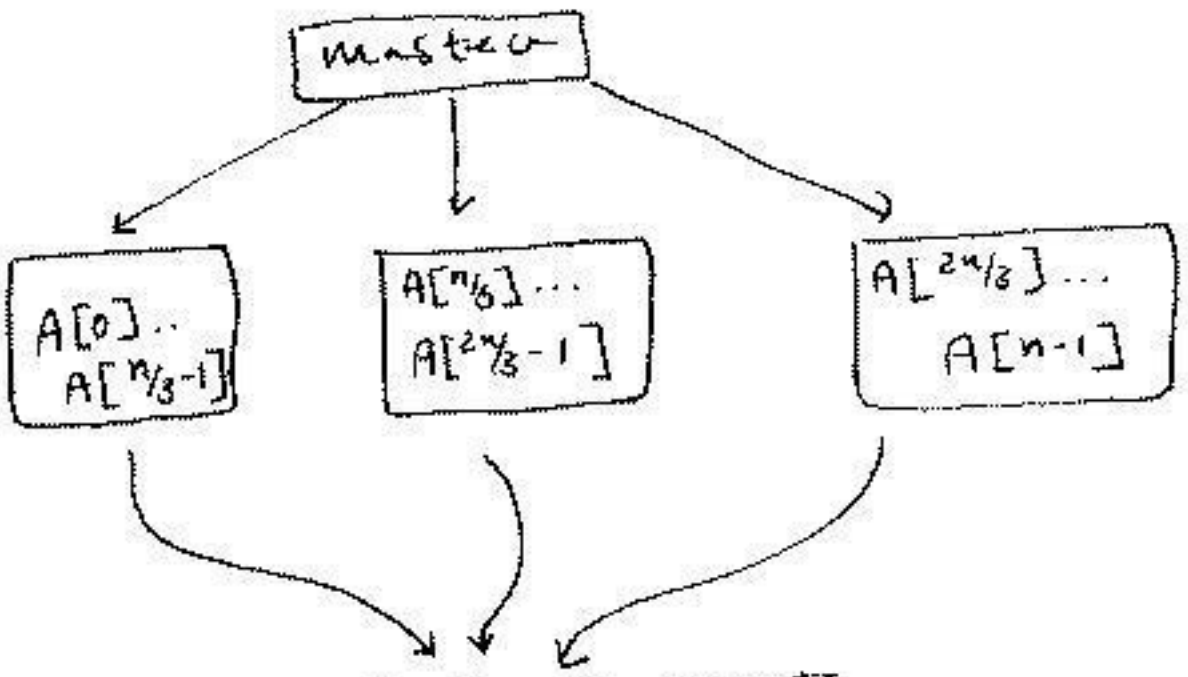
- result (specialist) for each $A^2[i][j]$
- OR, batch by row
- OR, agenda style

Later, continue with A^4, A^{16}, A^{256}, ...

What is and is not agenda parallel?

Example:

Find cds(A), common divisors



This is "specialist" in some ways, but a small change can make it into agenda.

Consider the Thread invoked by

```
T = cdsThread( 0, n/3-1)
T. start()
```

```
class cdsThread(threading.Thread)
def __init__(self, a, b)
```

defines start & end for this thread.

Alternatively:

```
L.Out(("calcA", 0, n/3-1))
T = cdsThread()
T. start
```

```
def run()
  Inp(("calcA", None, None))
```

here is where thread learns of start, end.

So what is the difference?

Mapping of which thread does what is dynamic in second form -- this is agenda style.

Also, we can use finer granularity.

Example: Each thread could do this:

```
while True:
    t, start, end = L.In(("calcA", None, None))
    if end - start > 0:
        L.Out(("calcA", start+1, end))
    for j in range(A[start]):
        if A[start] % j == 0:
            L.Out(("divisor", start, j))
```

we need to have other process/threads to collect & intersect results.

Open "Problems" with this style of coding:

1. how to know when to end a thread (otherwise they are blocked at In)

   - not easy just to use Inp!

2. how to know how many tuples to expect in output?

   ex: (("divisor", 12, 1)
   ("divisor", 12, 2)
   ("divisor", 12, 5)

   : can we expect more?

One way to solve would be to "relax" the Linda syntax:

allow tuples & dictionaries inside of Linda tuples.

We can simulate this already in Python!

Look UP repr and eval

in core functions of Python!