Scientific Computation

Spring, 2019

Lecture 9

Notes

- Lab 4 solutions and example Dijkstra code have been posted
- Today's office hour is in 6M 20

Today

- Wrap up discussion of graph search
- Spreading processes on networks

Graph search

BFS: search in unweighted graphs

- **Maintain *queue* of explored nodes**
- ***O(N+M)* operations *if* items are added to and removed from queue in O(1) time**
- **Should use** dequeue **rather than** list **in Python**
- **Applications include: finding connected components, shortest path**

Graph search

BFS: search in unweighted graphs

- **Maintain *queue* of explored nodes**
- ***O(N+M)* operations *if* items are added to and removed from queue in O(1) time**
- **Should use** dequeue **rather than** list **in Python**
- **Applications include: finding connected components, shortest path**

DFS: search in unweighted graphs

- **Maintain *stack* of explored nodes**
- ***O(N+M)* operations *if* items are added to and removed from stack in O(1) time**
- **Should use** list **in Python**
- **Applications include: finding connected components, topological sort in DAGs**

Graph search

BFS: search in unweighted graphs

- **Maintain *queue* of explored nodes**
- ***O(N+M)* operations *if* items are added to and removed from queue in O(1) time**
- **Should use** dequeue **rather than** list **in Python**
- **Applications include: finding connected components, shortest path**

DFS: search in unweighted graphs

- **Maintain *stack* of explored nodes**
- ***O(N+M)* operations *if* items are added to and removed from stack in O(1) time**
- **Should use** list **in Python**
- **Applications include: finding connected components, topological sort in DAGs**

Dijkstra: search in weighted graphs (non-negative weights)

- **Maintain *priority queue* of unexplored nodes with provisional shortest distances**
- ***O(N^2)* operations *if* naïve search for "closest" unexplored node is used**
- **Can use** heapq **rather than dictionary or list for better performance in Python**
- **Applications include: finding connected components, shortest path**

Dijkstra

Recap:

Maintain: 1) set of *"Explored"* nodes where shortest distances have been assigned and 2) set of *"Unexplored"* nodes where provisional distances have been set

Initialization: All nodes in U. Target node has distance=0, all other nodes in *U* with distance=infinity

At beginning of any iteration:

1. **All nodes in *U* with finite distances: distance = shortest distance via explored nodes only**
2. **Node in *U* with shortest distance: distance = shortest distance via all nodes**

Dijkstra

Recap:

Maintain: 1) set of *"Explored"* nodes where shortest distances have been assigned and 2) set of *"Unexplored"* nodes where provisional distances have been set

Initialization: All nodes in U. Target node has distance=0, all other nodes in *U* with distance=infinity

At beginning of any iteration:

1. **All nodes in *U* with finite distances: distance = shortest distance via explored nodes only**
2. **Node in *U* with shortest distance: distance = shortest distance via all nodes**

During iteration:

- **Move "shortest-distance" node, n*, from *U* to *E***
- **Maintain condition 2: consider paths to unexplored nodes via n*, update provisional distance when appropriate**

- **Cost: For each iteration, O(N) operations to find n* -- we can do better!**

Dijkstra

Cost: For each iteration, O(N) operations to find n* -- we can do better!

- **Need to arrange data so that it is easy to:**
  1. **Extract n***
  2. **Update provisional distances**

- **A *binary heap* provides these operations in $O(\log_2 N)$ time**

Dijkstra

Cost: For each iteration, O(N) operations to find n* -- we can do better!

- **Need to arrange data so that it is easy to:**
  1. **Extract n***
  2. **Update provisional distances**

- **A *binary heap* provides these operations in $O(\log_2 N)$ time**

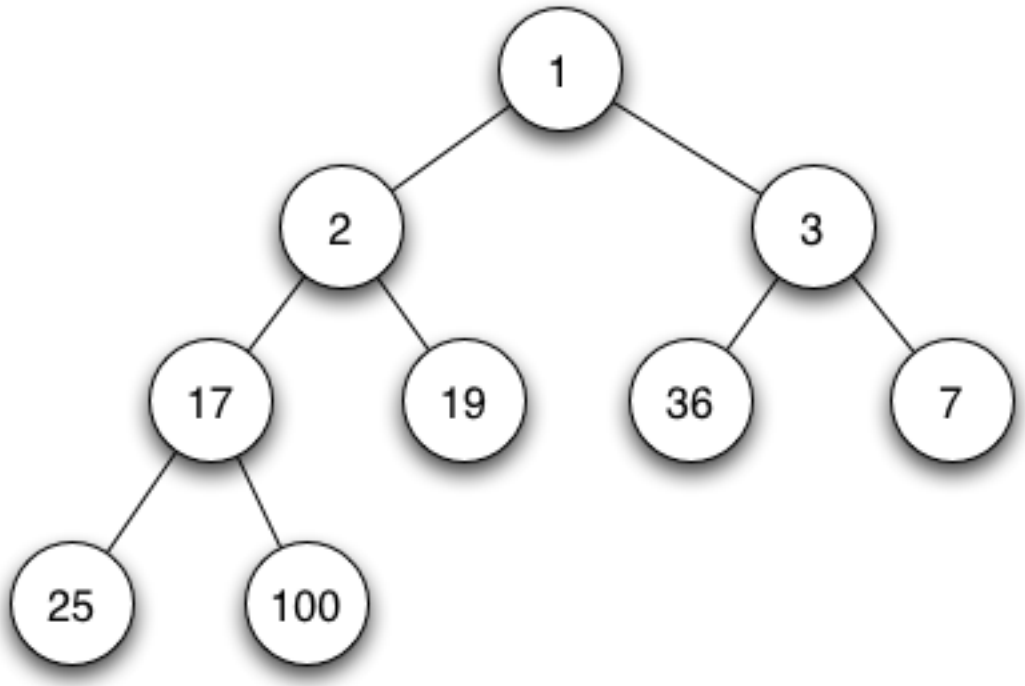
- **We won't go through the full Dijstra+heap implementation, but will outline the key elements**

- **A binary heap arranges nodes in a list, L; the order of the nodes is determined by the weights with the "smallest-distance" node in L[0]**

- **When n* is popped from the heap, the list is re-ordered in in $O(\log_2 N)$ time**

Binary heap

The arrangement of elements in L corresponds to a binary tree:



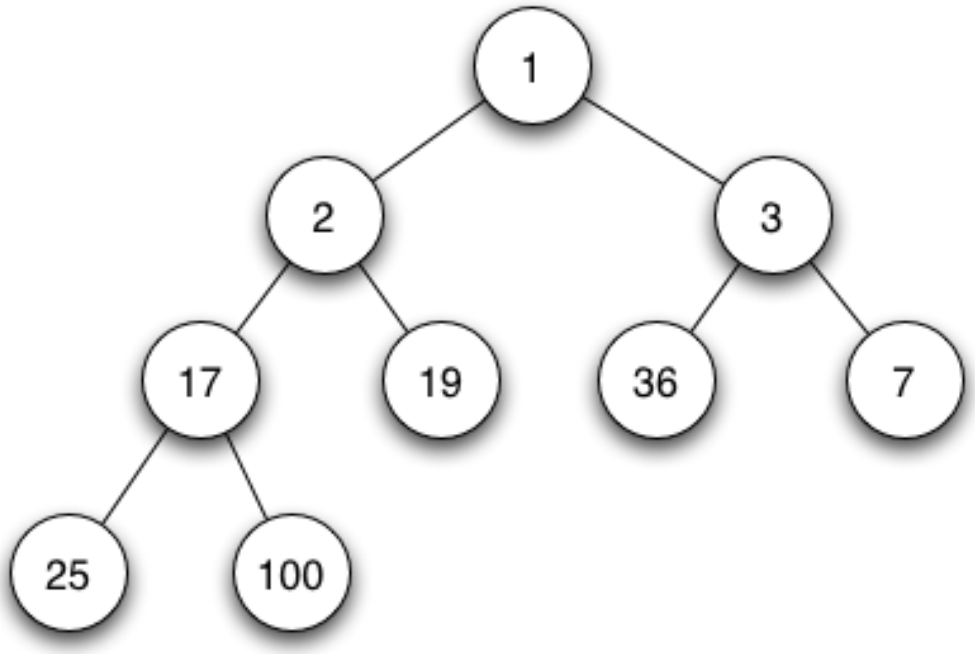
from wikipedia

L=

1	2	3	17	19	36	7	25	100

Binary heap

The arrangement of elements in L corresponds to a binary tree:



from wikipedia

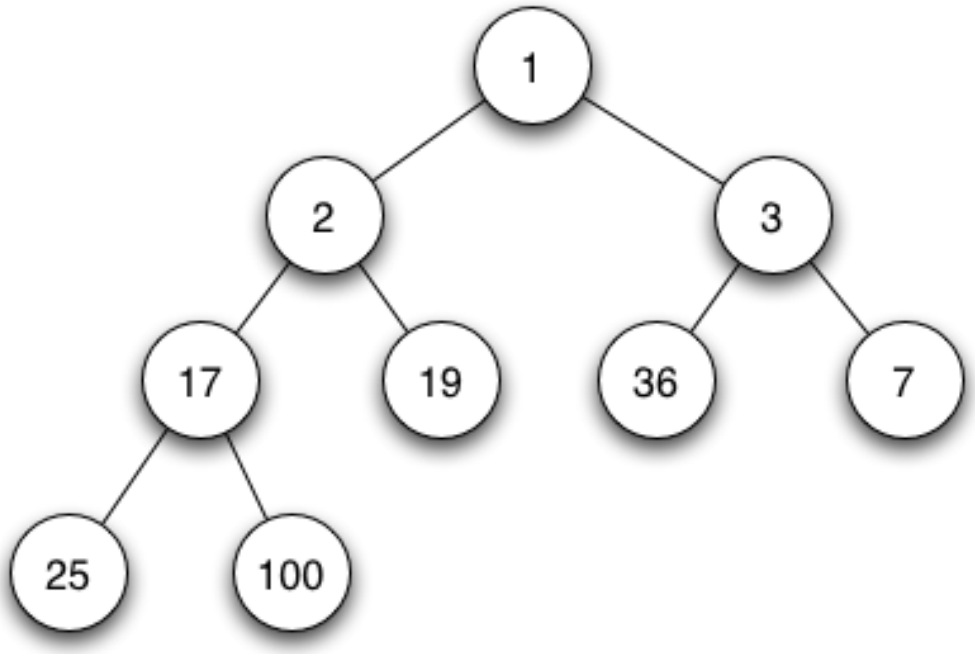
- **Nodes are ordered by keys (provisional distances)**
- **Python *heapq* builds ordered lists and provides $\log_2 N$ "min-removal"**
- **Does not provide $\log_2 N$ key modification (update provisional weight) – has to be manually coded**

L=

1	2	3	17	19	36	7	25	100

Dijkstra w/ heap

The arrangement of elements in L corresponds to a binary tree:



from wikipedia

- **Heap of unexplored nodes**

1. **Remove n* and re-structure heap**
2. **Adjust weights of neighbors of n* (if needed) and re-structure heap**

Step 1: $O(N \log_2 N)$
Step 2: $O(M \log_2 N)$

Is this better?

L=

1	2	3	17	19	36	7	25	100

Graph search

Final notes:

- **Other algorithms are available for weighted graphs with negative weights**
- **NetworkX provides shortest-path functions**
  - **Important to understand strengths/weaknesses and cost of underlying algorithms (especially when working on large networks)**
  - **Important to think about how data is organized (stacks, queues, heaps)**
  - **Not all networks can be represented as NetworkX graphs!**

Spreading processes on networks

- The importance of complex networks is intimately connected to the importance of spreading processes on complex network
- Picture below: Epidemic spreading via global air transportation network

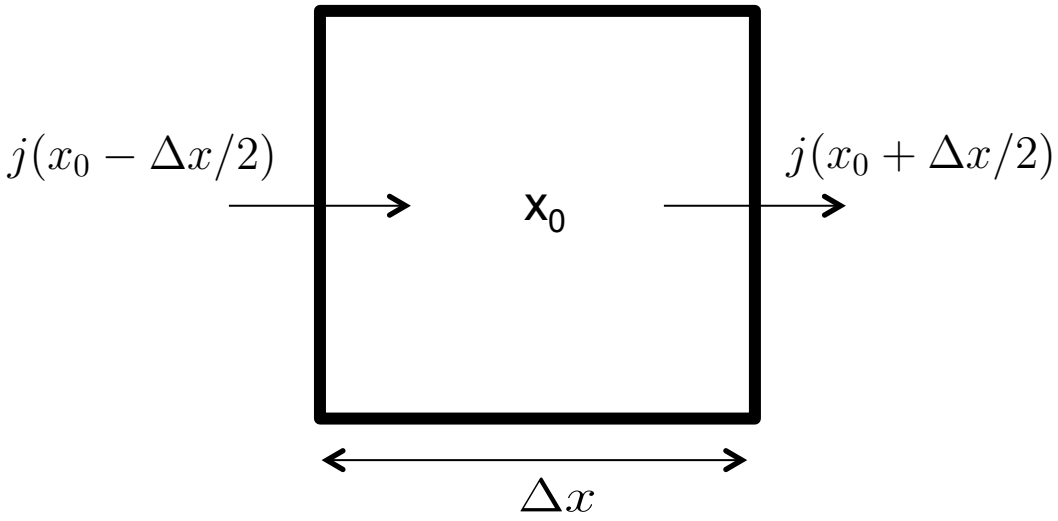
Image from Brockmann & Helbing, The Hidden Geometry of Network-Driven Contagion Phenomena

Other examples:

- Memes spreading on social networks
- Viruses spreading via the internet
- Blackouts

Spreading processes on networks

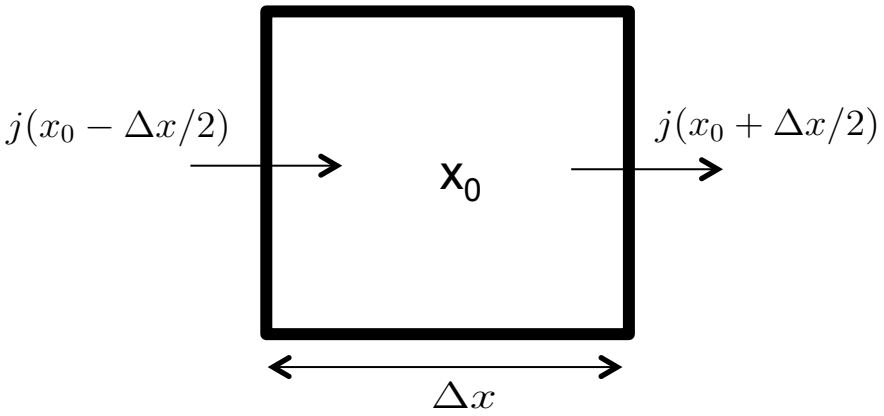
- How can/should we model these sorts of processes?
- The starting point is to think about diffusion
  - Basic idea: Flux of f is proportional to the gradient of f: $J = -\mathcal{D}\nabla f$
    - Examples: concentration of component in mixture, (thermal) energy



Spreading processes on networks

Basic idea: Flux of f is proportional to the gradient of f: $J = -\mathcal{D}\nabla f$

- **Examples: concentration of component in mixture, (thermal) energy**



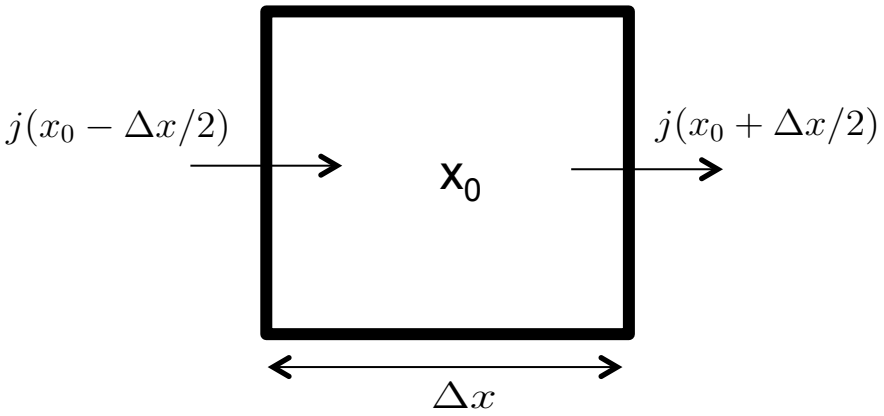
A concrete example:

- **Consider particles undergoing 1-D random walks on a grid**
- **We are interested in the particle density,** $n(x_i,t)$**, the number of particles per unit length and the particle number,** $N(x_i,t) = n(x_i,t)\,\Delta x$

Spreading processes on networks

Basic idea: Flux of f is proportional to the gradient of f: $J = -\mathcal{D}\nabla f$

- **Examples: concentration of component in mixture, (thermal) energy**



A concrete example:

- **Consider particles undergoing 1-D random walks on a grid**
- **We are interested in the particle density, $n(x_i,t)$, the number of particles per unit length and the particle number, $N(x_i,t) = n(x_i,t)\,\Delta x$**

- **1-D random walk: during time Δt, a particle moves a distance $+\Delta x$ or $-\Delta x$ with equal probability.**
- **During a time step, half of particles leave x_0 via right boundary, other half via left. The flux (over time Δt) at the right edge of the box is then:**

$$j(x_0 + \Delta x/2) = j_{out} - j_{in} = -0.5\,(N(x_0 + \Delta x) - N(x_0))$$

- **Rearranging:** $\frac{j(x_0+\Delta x/2)}{\Delta t} = -\frac{0.5\Delta x^2}{\Delta t}\frac{(n(x_0+\Delta x)-n(x_0))}{\Delta x}$

Diffusion in 1-D

Empirically (or from kinetic theory), we know we can define a constant diffusivity:

$$\mathcal{D} = \Delta x^2 / \Delta t$$

- **Finally, taking the limit as Δt, Δx → 0:** $J(x) = lim_{\Delta x, \Delta t \to 0} \frac{j(x)}{\Delta t} = -\mathcal{D} \frac{\partial n}{\partial x}$
- **This is Fick's first law**
- **The net flux into a 'box' corresponds to the rate of change of particles in the box:**

$$\frac{\partial n}{\partial t} = -\frac{\partial J}{\partial x} = \mathcal{D} \frac{\partial^2 n}{\partial x^2}$$

- **In three dimensions,** $\frac{\partial c}{\partial t} = \mathcal{D} \nabla^2 c$**, and c is a concentration (e.g. particles/volume)**
- **This is the standard diffusion equation which arises in a number of settings: ion transport, heat flow, mixing of a chemical species in a liquid**

Diffusion in networks

- With networks, we no longer have spatial derivatives
- Can we use similar ideas?
- Need to think about flux along a link between two nodes
- Then the rate of change at a node will be the sum of fluxes from its neighbors

Diffusion in networks

- With networks, we no longer have spatial derivatives
- Can we use similar ideas?
- Need to think about flux along a link between two nodes
- Then the rate of change at a node will be the sum of fluxes from its neighbors
- The diffusive flux from node j to node i is taken to be: $J_{ij} = -\mathcal{D}\left(f_i - f_j\right)$

  which leads to: $\frac{\partial f_i}{\partial t} = \sum_j J_{ij}$ and the sum is over all nodes with links to node i
- We can rewrite the sum using the network's adjacency matrix:

$$\frac{\partial f_i}{\partial t} = \sum_{j=1}^{N} A_{ij} J_{ij}$$

$$\frac{\partial f_i}{\partial t} = -\sum_{j=1}^{N} A_{ij} \mathcal{D}\left(f_i - f_j\right)$$

- And this is a representative model for diffusion/spreading in networks