Graph Algorithms

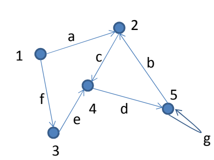
Overview

- Graph: abstract data type
  - G = (V,E) where V is set of nodes, E is set of edges $\subseteq$ VxV
- Structural properties of graphs
  - Power-law graphs, uniform-degree graphs
- Graph representations: concrete data type
  - Compressed-row/column, coordinate, adjacency list
- Graph algorithms
  - Operator formulation: abstraction for algorithms
  - Algorithms for single-source shortest-path (SSSP) problem
- Machine learning algorithms
  - Page-rank
  - Matrix-completion for recommendation systems

Structural properties of graphs

Graph-matrix duality

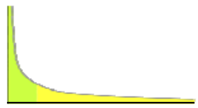
- Graph (V,E) as a matrix
  - Choose an ordering of vertices
  - Number them sequentially
  - Fill in |V|x|V| matrix
    - A(i,j) is w if graph has edge from node i to node j with label w
  - Called *adjacency matrix* of graph
  - Edge (u → v):
    - v is *out-neighbor* of u
    - u is in-neighbor of v
- Observations:
  - Diagonal entries: weights on self-loops
  - Symmetric matrix ↔ undirected graph
  - Lower triangular matrix ↔ no edges from lower numbered nodes to higher numbered nodes
  - Dense matrix ↔ clique (edge between every pair of nodes)



from \ to	1	2	3	4	5
1	0	a	f	0	0
2	0	0	0	c	0
3	0	0	0	e	0
4	0	0	0	0	d
5	0	b	0	0	g

Sparse graphs

- Terminology:
  - Degree of node: number of edges connected to it
  - (Average) diameter of graph: average number of hops between two nodes
- Power-law graphs
  - small number of very high degree nodes (see next slide for example)
  - low diameter
    - "six degrees of separation" (Karinthy 1929, Milgram 1967), on Facebook, it is 4.74
  - typical of social network graphs like the Internet graph or the Facebook graph
- Uniform-degree graphs
  - nodes have roughly same degree
  - high diameter
  - road networks, IC circuits, finite-element meshes
- Random (Erdös-Rènyi) graphs
  - constructed by random insertion of edges
  - mathematically interesting but few real-life examples



Node degree distribution of power-law graphs

Airline route map: power-law graph

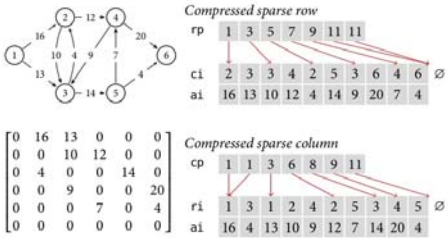
Road map: uniform-degree graph

Graph representations: how to store graphs in memory

Three storage formats:CSR,CSC,COO

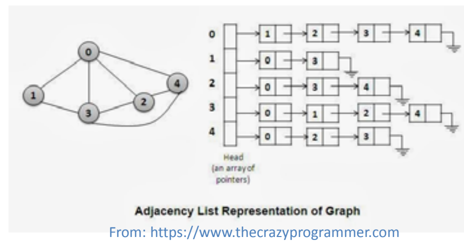
Coordinate storage

1	**2**	**4**	**5**	**3**	**1**	**2**	**3**	**4**	**5**
2	4	6	6	5	3	3	2	3	4
16	12	20	4	15	13	10	4	9	7



Labels on nodes are stored in a separate vector (not shown)

Adjacency list representation



From: https://www.thecrazyprogrammer.com

Permits you to add and remove edges from graph
Deleting edges: often it is more efficient to just to mark an edge as deleted rather than delete it physically from the list

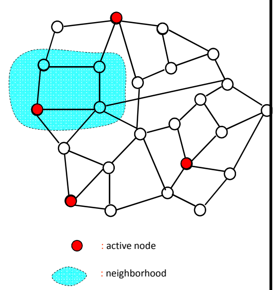
Graph algorithms

Overview

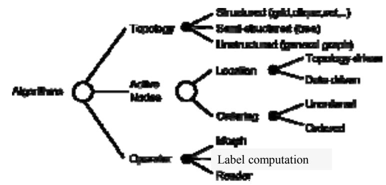
- Algorithms: usually specified by pseudocode
- We take a different approach:
  - operator formulation of algorithms
  - data-centric abstraction in which data structures play central role
- Advantages of operator formulation abstraction:
  - Connections between seemingly unrelated algorithms
  - Sources of parallelism and locality become evident
  - Suggests common set of mechanisms for exploiting parallelism and locality for all algorithms

Operator formulation of algorithms

- Algorithm = Operator + Schedule
- Operator: local view of algorithm
  - Active node/edge: place in graph where some computation is needed
  - Operator: specification of computation
  - Activity: application of operator to active node
  - Neighborhood: Set of nodes/edges read/written by activity
- Schedule: global view of algorithm
  - Unordered algorithms:
    - active nodes can be processed in any order
    - all schedules produce the same answer but performance may vary
  - Ordered algorithms:
    - problem-dependent order on active nodes



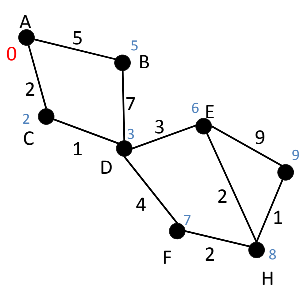
TAO analysis: terminology



- Active nodes
  - Topology-driven algorithms
    - Algorithm is executed in rounds
    - In each round, all nodes/edges are initially active
    - Iterate till convergence
  - Data-driven algorithms
    - Some nodes/edges initially active
    - Applying operator to active node may create new active nodes
    - Terminate when no more active nodes/edges in graph
- Operator
  - Morph: may change the graph structure by adding/removing nodes/edges
  - Label computation: updates labels on nodes/edges w/o changing graph structure
  - Reader: makes no modification to graph

Graph problem:SSSP

- Problem: single-source shortest-path (SSSP) computation
- Formulation:
  - Given an undirected graph with positive weights on edges, and a node called the source
  - Compute the shortest distance from source to every other node
- Variations:
  - Negative edge weights but no negative weight cycles
  - All-pairs shortest paths
  - Breadth-first search: all edge weights are 1
- Applications:
  - GPS devices for driving directions
  - social network analyses: centrality metrics



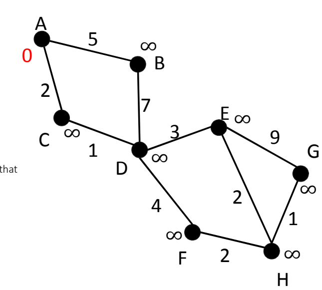
Node A is the source

SSSP Problem

- Many algorithms
  - Dijkstra (1959)
  - Bellman-Ford (1957)
  - Chaotic relaxation (1969)
  - Delta-stepping (1998)
- In textbook presentations, they seem unrelated to each other
- Common structure:
  - Each node has a label d that is updated repeatedly
    - initialized to 0 for source and ∞ for all other nodes
    - during algorithm: shortest known distance to that node from source
    - termination: shortest distance from source
  - All of them use the same *operator*

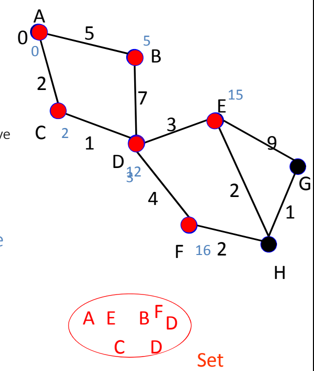
```
relax-edge(u,v):
  if d[v] > d[u]+w(u,v)
 then d[v] ← d[u]+w(u,v)

relax-node(u):
  relax all edges connected to u
```

  - Differences between algorithms: schedule



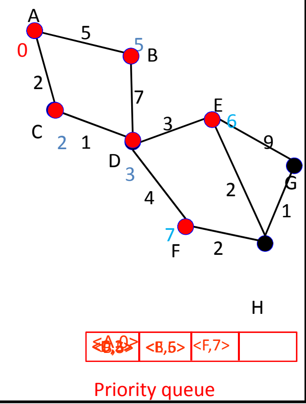
Chaotic relaxation (1969)

- Active node
  - node whose label has been updated
  - initially, only source is active
- Schedule
  - pick active node at random
  - use a (work)-set or multiset to track active nodes
- TAO: unordered, data-driven algorithm
- Main inefficiency: number of node relaxations depends on the schedule
  - can be exponential in the size of graph
- Parallelization:
  - ??



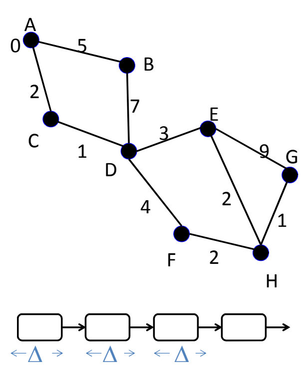
Dijkstra's algorithm (1959)

- Active nodes
  - node whose label has been updated
  - initially, only source is active
- Schedule for processing nodes
  - prefer nodes with smaller labels since they are more likely to have reached final values
- Implementation of work-set
  - priority queue ordered by node label
- Work-efficient ordered algorithm
  - node is relaxed just once
  - O(|E|*lg(|V|))
- Parallelization: ??
- Main inefficiency:
  - as we will see later, there is little parallelism for most graphs



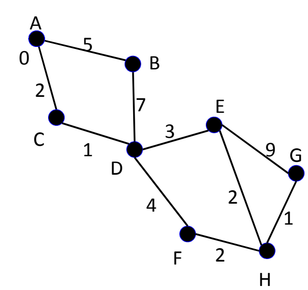
Delta-stepping (1998)

- Controlled chaotic relaxation
  - Exploit the fact that SSSP is robust to priority inversions
  - "soft" priorities
- Implementation of work-set:
  - parameter: Δ
  - sequence of sets
  - nodes whose current distance is between $n\Delta$ and $(n+1)\Delta$ are put in the n^{th} set
  - nodes in set n are completed before processing of nodes in set (n+1) are started
- Δ = 1: Dijkstra
- $\Delta = \infty$: Chaotic relaxation
- Picking an optimal Δ :
  - depends on graph and machine
  - high-diameter graph → large Δ
  - find experimentally



Bellman-Ford (1957)

- Algorithm:
  - Execute algorithm in rounds
  - In each round, iterate over all nodes and apply relaxation operator
  - Do this |V| times
  - In practice, terminate rounds when no node changes value in a round
- Work-efficiency:
  - O(|E|*|V|)
  - In each round, we may visit many nodes where there is no work to do
  - However, we do not need a worklist, so there is one less problem for the implementation to worry about
- TAO analysis:
  - topology-driven
  - each round is unordered
- Parallelization of rounds
  - ???



Summary of SSSP Algorithms

- Chaotic relaxation
  - unordered, data-driven algorithm
    - use sets/multisets for work-set
  - amount of work depends on schedule: can be exponential in size of graph
- Dijkstra's algorithm
  - ordered, data-driven algorithm
    - use priority queue for work-set
  - O(|V|log(|E|)): work-efficient but little parallelism
- Delta-stepping
  - controlled chaotic relaxation: parameter Δ
  - Δ permits trade-off between parallelism and work-efficiency
- Bellman-Ford algorithm
  - unordered, topology-driven algorithm
  - O(|V||E|) time

Machine learning

- Many machine learning algorithms are sparse graph algorithms
- Examples:
  - Page rank: used to rank webpages to answer Internet search queries
  - Recommender systems: used to make recommendations to users in Netflix, Amazon, Facebook etc.

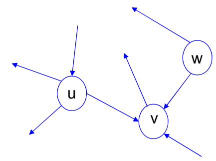
Web search

- When you type a set of keywords to do an Internet search, which web-pages should be returned and in what order?
- Basic idea:
  - offline:
    - crawl the web and gather webpages into data center
    - build an index from keywords to webpages
  - online:
    - when user types keywords, use index to find all pages containing the keywords
  - key problem:
    - usually you end up with tens of thousands of pages
    - how do you rank these pages for the user?

Ranking pages

- Manual ranking
  - Yahoo did something like this initially, but this solution does not scale
- Word counts
  - order webpages by how many times keywords occur in webpages
  - problem: easy to mess with ranking by having lots of meaningless occurrences of keyword
- Citations
  - analogy with citations to articles
  - if lots of webpages point to a webpage, rank it higher
  - problem: easy to mess with ranking by creating lots of useless pages that point to your webpage
- PageRank
  - extension of citations idea
  - weight link from webpage A to webpage B by "importance" of A
  - if A has few links to it, its links are not very "valuable"
  - how do we make this into an algorithm?

Web graph

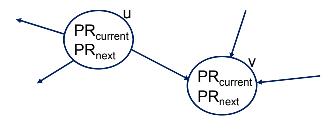


Webgraph from commoncrawl.org

- Directed graph: nodes represent webpages, edges represent links
  - edge from u to v represents a link in page u to page v
- Size of graph: commoncrawl.org (2012)
  - 3.5 billion nodes
  - 128 billion links
- Intuitive idea of pageRank algorithm:
  - each node in graph has a weight (pageRank) that represents its importance
  - assume all edges out of a node are equally important
  - importance of edge is scaled by the pageRank of source node

PageRank (simple version)

Graph G = (V,E)
|V| = N



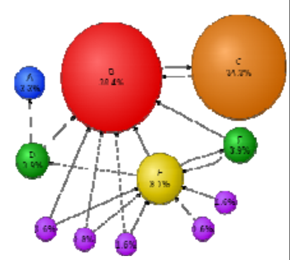
- Iterative algorithm:
  - compute a series PR_0, PR_1, PR_2, … of node labels
- Iterative formula:
  - $\forall v \in V.\ PR_0(v) = 1/N$
  - $\forall v \in V.\ PR_{i+1}(v) = \sum_{u \in \text{in-neighbors}(v)} \frac{PR_i(u)}{\text{out-degree}(u)}$
- Implement with two fields $PR_{current}$ and PR_{next} in each node

Page Rank (contd.)

- Small twist needed to handle nodes with no outgoing edges
- Damping factor: d
  - small constant: 0.85
  - assume each node may also contribute its pageRank to a randomly selected node with probability (1-d)
- Iterative formula
  - $\forall v \in V.\ PR_0(v) = \frac{1}{N}$
  - $\forall v \in V.\ PR_{i+1}(v) = \frac{1-d}{N} + d * \sum_{u \in \text{in-neighbors}(v)} \frac{PR_i(u)}{\text{out-degree}(u)}$

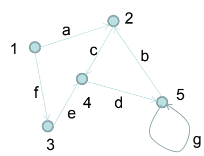
PageRank example

- Nice example from Wikipedia
- Note
  - B and E have many in-edges but pageRank of B is much greater
  - C has only one in-edge but high pageRank because its in-edge is very valuable
- Caveat:
  - search engines use many criteria in addition to pageRank to rank webpages



Matrix-vector multiplication

- Matrix computation: $\underline{y} = A\underline{x}$
- Graph interpretation:
  - Each node i has two values (labels) x(i) and y(i)
  - Each node i updates its label y using the x value from each out-neighbor j, scaled by the label on edge (i,j)
  - Topology-driven, unordered algorithm
- Observation:
  - Graph perspective shows dense MVM is special case of sparse MVM
  - What is the interpretation of $\underline{y} = A^T\underline{x}$?
- Page-rank can be expressed as generalized MVM
  - Reinterpret + and * operations

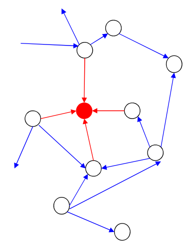


	1	2	3	4	5	
1	0	a	f	0	0	x1
2	0	0	0	c	0	x2
3	0	0	0	e	0	x3
4	0	0	0	0	d	x4
5	0	b	0	0	g	x5

A x

PageRank discussion

- Vertex program (Pregel):
  - value at node is updated using values at immediate neighbors
  - very limited notion of neighborhood but adequate for pageRank and some ML algorithms
- CombBlas: combinatorial BLAS
  - generalized sparse MVM: + and * in MVM are generalized to other operations like ∨ and ∧
  - adequate for pageRank
- Interesting application of TAO
  - standard pageRank is topology-driven
  - can you think of a data-driven version of pageRank?

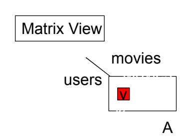


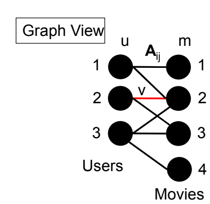
Recommender system

- Problem
  - given a database of users, items, and ratings given by each user to some of the items
  - predict ratings that user might give to items he has not rated yet (usually, we are interested only in the top few items in this set)
- Netflix challenge
  - in 2006, Netflix released a subset of their database and offered $1 million prize to anyone who improved their algorithm by 10%
  - triggered a lot of interest in recommender systems
  - prize finally given to BellKor's Pragmatic Chaos team in 2009

Data structure for database

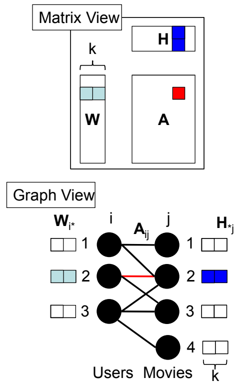
- Sparse matrix view:
  - rows are users
  - columns are movies
  - A(u,m) = v is user u has given rating v to movie m
- Graph view:
  - bipartite graph
  - two sets of nodes, one for users, one for movies
  - edge (u,m) with label v
- Recommendation problem:
  - predict missing entries in sparse matrix
  - predict labels of missing edges in bipartite graph





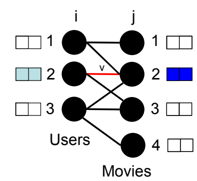
One approach: matrix completion

- Optimization problem
  - Find m×k matrix **W** and k×n matrix **H** (k << min(m,n)) such that **A** ≈ **WH**
  - Low-rank approximation
  - **H** and **W** are dense so all missing values are predicted
- Graph view
  - Label of user nodes i is vector corresponding to row W_{i*}
  - Label of movie node j is vector corresponding to column H_{*j}
  - If graph has edge (u,m), inner product of labels on u and m must be approximately equal to label on edge



One algorithm:SGD

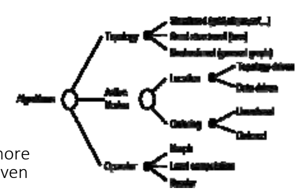
- Stochastic gradient descent (SGD)
- Iterative algorithm:
  - initialize all node labels to some arbitrary values
  - iterate until convergence
    - visit all edges (u,m) in some order and update node labels at u and m based on the residual
- TAO analysis:
  - active edges: topology-driven, unordered
- What algorithm does this remind you of?
  - Bellman-Ford



Summary of discussion of algorithms

What we have learned

- Operator formulation:
  - data-centric view of algorithms
- TAO classification
- Location of active nodes
  - Topology-driven algorithms
  - Data-driven algorithms
  - Data-driven algorithm may be more work-efficient than topology-driven one
- Ordering of active nodes
  - Unordered algorithms
  - Ordered algorithms
- Some problems
  - have both ordered and unordered algorithms (e.g. SSSP)
  - have both topology-driven and data-driven algorithms (e.g. SSSP, pageRank)



Questions

- What are the sources of parallelism and locality in algorithms?
- Can the operator formulation help us in answering this question?
- How do we exploit parallelism and locality efficiently?