CS 7280/4973 – Data Str & Alg Scalable Comp Spring 2025

Lecture Graphs Representation & Computations — March 24

Prof. Prashant Pandey *Scribe: Ben Mash*

1 Overview

So far in this course we have looked at scalability in terms of space, query time, and runtime.
Main topics throughout the course:

1. Integer Data $\longrightarrow$ vEB, x-fast, and y-fast trees [**space + operation time**]
2. String Data $\longrightarrow$ Tries, Suffix Array/Trees, Burroughs-Wheeler Transform, FM-Index [**space**]
3. Hashing $\longrightarrow$ Hash functions, Hash Tables [**fast operations**]
4. Filters $\longrightarrow$ membership querries [**space**]
5. Sketehces $\longrightarrow$ Cardinality $\longrightarrow \begin{cases} \text{NLL} \\ \text{frequency estimation (MG, CS, CMS)} \end{cases}$
6. Nearest Neighbor $\longrightarrow \begin{cases} \epsilon\text{NN} \\ \text{LSH, minhash} \\ \text{NNS} \end{cases}$
7. Graphs — covering today

Application to consider — Metagenomics:
- Looking at biological samples from a large population of organisms
- Need to examine distance between all samples in order to find samples that come from the same organism
- Important to be able to compute NNS on edit distances

2 Graphs

Vertices: model objects
Edges: model relationships between objects
Edges can be directed and/or weighted
Both edges and nodes can have metadata associated with them

$$\begin{tikzcd} V_1 \arrow[r, "\text{edge}"] & V_2 \end{tikzcd}$$

Applications of Graphs: Social networks, collaboration networks, transportation networks, connectomics

Consider twitter:
Nodes: users — 100s of millions
Edges: users mentions in same tweet
Nearest Neighbor Search — can use distance on graph, i.e. 1 hop neighbors, 2 hop neighbors

Identifiying Important Nodes

- Betweenness Centrality — nodes which many paths pass through
- Triangle Finding — find small cliques such as 3 mutual friends
- PageRank
  - **web graph** — one of the largest public graphs
  - each webpage is a node, links between pages are edges
  - rank pages by the in-degree of their node
  - powered original Google Search:
    - look up each query word in an inverted index hashtable:

      $$word \mapsto \{\text{all documents containing } word\}$$

    - take intersection of the sets corresponding to each query word:

      $$\bigcap_{w \in query} \{\text{all documents containing } w\}$$

    - return documents sorted by descending page rank
  - recently, many search engines have moved to searching documents via vector embedding similarity

3 Computations on Graphs

1. **Social network queries**
   - find all friends who went to the same school
   - finding common friends with someone
   - reccomending people whom you might know
2. **Finding clusters** — groups that are well-connected internally and less connected externally
   - finding people with similar interests
   - detecting fraudulent websites

- document clustering
- unsupervised learning

3. **Subgraph finding**
   - finding/counting specific subgraphs within a larger graph
   - finding recurrent subgraphs

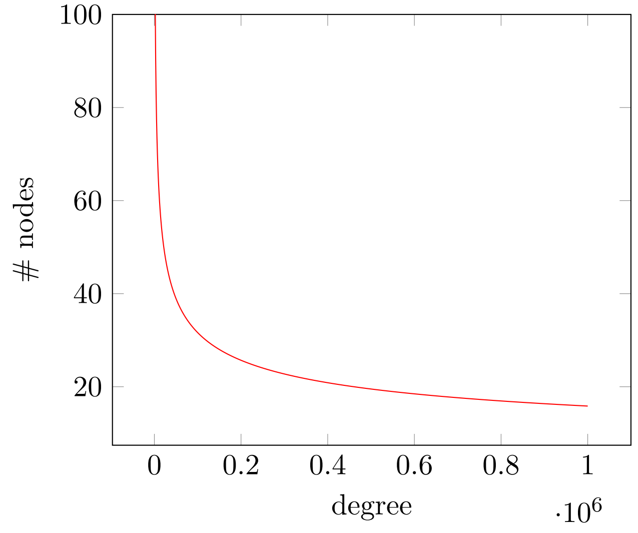
4 Properties of Real World Graphs

Most graphs approximately follow a highly-skewed **power law** distribution:

$$\#\text{ nodes with degree } d \approx a \cdot d^{-P}$$

$$\text{s.t. } 2 < P < 3$$

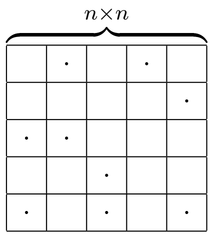
Example power law plot:



5 Representing Graphs

In this section, let $n = \#nodes$, and $m = \#edges$.

1. **Adjacency Matrix**
   - $n \times n$ bit matrix
   - each cell at index (x, y) represents whether an edge exists from node x to node y
   - Add Edge: $O(1)$
   - Space: $O\left(n^2\right)$



2. **Edge List**
   - m-length list of $(source,\ destination)$ tuples
   - Add Edge: $O(m)$
   - Space: $O(m)$

$S\ D$

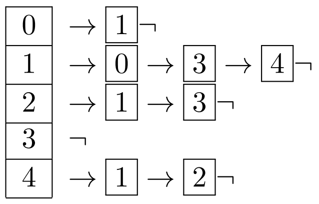
(0,1)

(2,3)

(8,3)

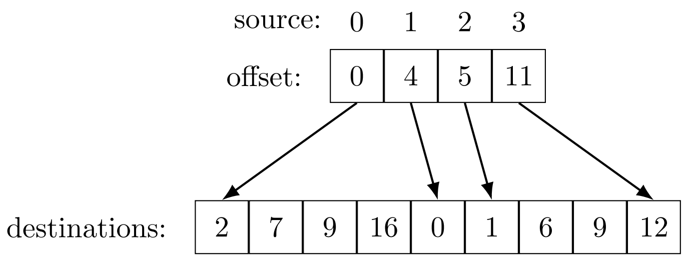
$\vdots$

3. **Adjacency List**
   - n-length array of linked lists
   - Add Edge: $O(\deg(n)) \quad \leq \quad \underset{\text{loose upper bound}}{O(m)}$
   - Space: $O(m+n)$



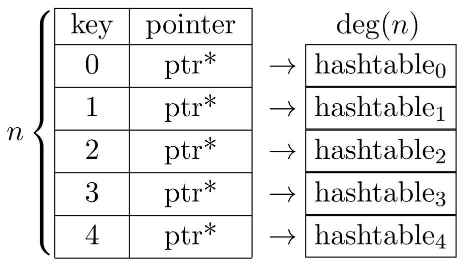
4. **Compressed Sparse Row**

- for each node, store an index into a list containing destination nodes
- Add Edge: n/a (static, no inserts)
- Space: $O(m+n)$, but no constant overhead, so really $= (m+n)$
- this representation achieves high cache efficiency



5. **Dhruv's Proposal**

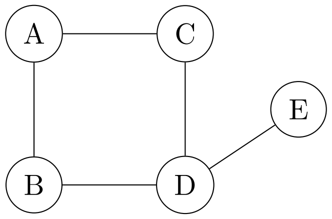
- maintain a hashtable mapping source nodes to a hashset of destination nodes
- Add Edge: $O(1)$
- Space: $O(m+n)$
- this representation has been investigated but it has some dominating bottlenecks in practive
- consider the power law: $\sim 90\%$ of hashtables will have only 1 or 2 items



6 Breadth-first Search

Given a source node s, visit all nodes in order of distance from s.

Example:



Starting at source D, one possible order could be D, B, C, E, A.

We also often return the distance from source to each visited node:
$A : 2,\ B : 1,\ C : 1,\ D : 0,\ E : 1$

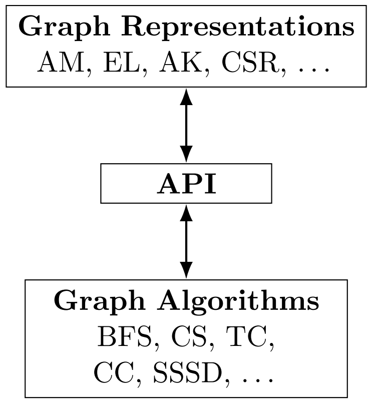
BFS can be calculated in $O(m + n)$ time and is the foundation of many graph algorithms.

7 Ligra

Ligra is a lightweight graph *interface* for graph algorithms.

Presented at PPOPP 2013 by Shun and Blellock of CMU [1].

Graph systems are often divided in 2 parts:



Graph APIs can be defined which allow for any graph algorithm to be run on any graph representation.

Most/all graph algorithms could be written using only **get-neighbors**, but it would be inefficient.

Instead, we must consider parallelism opportunities and practical runtime.

In order to provide this, Ligra builds upon compare-and-swap atomic instructions, which allow parallel systems to be built without explicit locking behavior.

Ligra Interface:

With graph $G : (V, E)$ and vertex subset $U \subseteq V$

1. size$(U) \to N$

   returns $|U|$

2. EdgeMap $(G, U, F : (U \times V \to \mathit{bool}), C : (V \to \mathit{bool})) \to$ vertex subset

   for an unweighted graph $G : (V, E)$, EdgeMap applies a function F to all edges with a source vertex in U and a target vertex satisfying C, and returns the subset for which F returned true.

3. VertexMap $(U, F : (vert \to bool)) \to$ vertex subset

   applies F on every vertex in U and returns the subset for which F returned true.

∗ **VertexMap and EdgeMap can both run in parallel**

∗ **Vertex Subset: can be represented by a set of integers for sparse subsets, or as a bit vector for dense subsets**

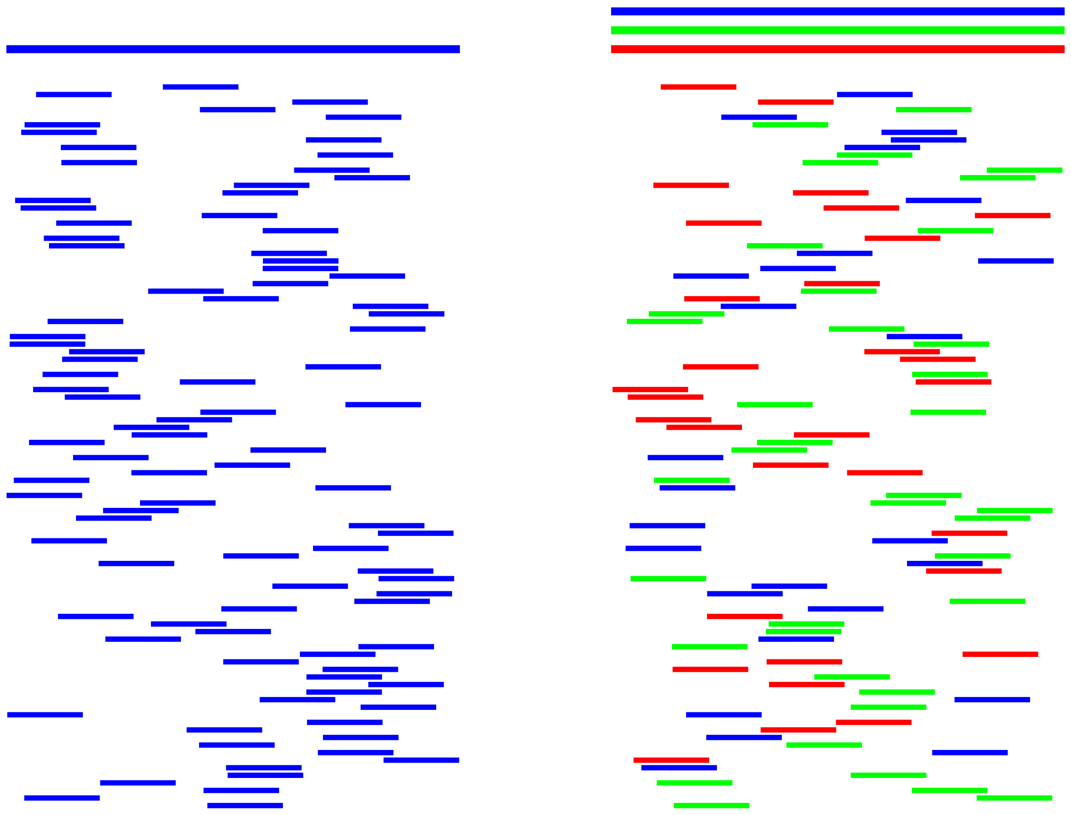
8 BFS with Ligra

Algorithm 1 Ligra BFS

```
$parents \leftarrow \overset{\text{list of size } n}{[-1, \ldots, -1]}$

procedure UPDATE($s, d$):
    return CAS $(parents[d], -1, s)$          ▷ ensure we only visit each node $d$ once
end procedure

procedure COND($i$):
    return $parents[i] == 1$
end procedure

procedure BFS($G, r$):                          ▷ $r$ = start node
    $parents[r] \leftarrow r$
    $frontier \leftarrow \{r\}$                 ▷ $frontier$ is a vertex subset
    while size($frontier$) $\neq 0$ do          ▷ this while loop can be performed in parallel
        $frontier \leftarrow$ EdgeMap($G$, $frontier$, UPDATE, COND)
    end while
end procedure
```

9 Metagenomic Assembly

In **Genomic Assembly**, millions of genome samples from a single organism are aligned in attempt to form a single string that contains the entire genome. This is an NP-Hard problem, although there are many heuristics that are used to make it more tractable.



Left: genomic assembly, Right: metagenomic assembly

In **Metagenomic Assembly**, multiple organisms are present in the sample, and their genomes must be assembled separately. Subgraph finding can be used to determine whether a known genome is present in the sample of genome segments. This is an example of an application of large scale graph algorithms outside of the typical examples of social networks or webpages.

References

[1] Julian Shun and Guy Blelloch Ligra: A Lightweight Graph Processing Framework for Shared Memory *PPoPP*, pp. 135-146, 2013.