Will look at many graph algorithms from parallel perspective:
- Connectivity
- Matching
- Maximal Ind. Set
- Shortest Paths
- Densest Subgraph

Main question(s) as usual: can we design work-efficient algorithms?
- Can we make the algorithms practical and theoretically efficient?

Start with graph connectivity.
Input: $G(V, E)$ (undirected), $s, t \in V$
Let $n=|V|$, $m=|E|$
Output: Are $s, t$ connected in $G$?

Usual idea: compute components of each $v \in V$, $C[v]$.
Check if $C[s] = C[t]$.

Sequentially, easy to solve in $O(n+m)$ time, eg using BFS/DFS.
- Also practical in essentially linear time using union-find.

Parallel BFS: Understand some basic ideas in graph algos + impls using BFS.

not nec. undirected
BFS($G, x$):
- compute a BFS tree rooted at $x$, i.e. compute an array of Parents, $P[v]$
  s.t. $P[v]=\text{parent of } v \text{ in BFS tree } (v \neq x)$
  $P[x] = x$

How do you solve BFS in parallel?
One way: just emulate sequential BFS and do it level-by-level.
- edge_map: primitive for traversal (useful beyond BFS).
\[ \text{edge-map}(U, U, f) = \]
\[ \quad \text{perform } u \in [0: |U|] \]
\[ \quad N[i] = \{ v \in N^+(U[i]) \mid f(u, v) \} \]
\[ \quad \text{return Flatten}(N) \]

- Usually ensure the output of edge-map is a set
- by ensuring that a vertex \( v \in N(U) \) is atomically
- acquired by only one vertex in \( U \).

- Note: this flatten-based impl is not very practical. Next Tuesday's
- lecture will discuss more practical ideas.

\[ \text{Work: } O(|U| + \sum_{u \in U} \deg^*(u)) \]

\[ \text{Depth: } O(\log(n)) \]

Back to BFS: we will need the test-and-set primitive from HW1.

- Test-And-Set(x) (TS): take a reference to memory location \( x \),
- check if value \( x = 0 \) and if so atomically set it to 1, returning 0;
  - if value \( x = 1 \), it returns 1.

BFS output: parents array \( \text{parent} \) s.t. the tree formed by all
\((u, \text{parent}[u])\) edges \( u \in V, u \neq r \) is a valid BFS tree
\( \text{i.e. any non-tree edge } (V, w) \text{ either lies in the same level, or between adjacent levels.} \)

BFS \((G, r) = \)
\[ F \leftarrow \{ r \} \quad \text{// initially only root in the frontier} \]
\[ \quad \text{perform } u \in V: \]
\[ \quad \text{X}[u] = 0 \]
\[ \quad \text{P}[u] = u \]
\[ \quad \text{X}[r] = 1 \quad \text{// mark the root as visited} \]
\[ \quad \text{while } |F| > 0: \]
\[ \quad \quad F \leftarrow \text{edge-map}(G, F, \lambda(u, v): \]
\[ \quad \quad \quad \text{if test-and-set}(X[k]) = 0 \text{ then } P[u] = u \text{ and return true;} \]
\[ \quad \quad \quad \text{return false;} \]
\[ \text{return } P \]
Def: The shortest path between u,v \in V in G is denoted \( S_G(u,v) \)

Def: the radius of a vertex \( u \in V \) is max \( \forall v \in V \) \( S_G(u,v) \) = radius \( G \) \( (u) \).

Def: the diameter of a graph \( G \) \( (\text{Diam}(G)) \) is max \( \forall v \in V \) radius \( G \) \( (v) \).

- The parallel BFS algorithm runs in \( O(n+m) \) work and \( O(\text{diam}(G) \log n) \) depth.

Unfortunately, in general no good bounds on \( \text{diam}(G) \) in RW graphs. Often small, so BFS-based approaches can be good, but many important graphs have high diameter. Currently we don’t know how to solve BFS (or other SSSP problems) in \( \text{polylog}(n) \) depth in general graphs.

What about graph people used BFS for?
- For many prob, another powerful technique is applicable.

---

**Low-Diameter Decomposition (LDD)**

- Instead of searching from a single vertex, the idea is to break graph up into some number of connected clusters s.t.
  1. few edges are cut
  2. internal diameters of each cluster are bounded (ensures that subgraphs can be efficiently explored in parallel and generally a useful alg. property that algorithms using LDD can exploit).

More formally:

A \((\beta, d)\)-decomposition for \( 0 < \beta < 1 \) is a partition of \( V \) into clusters \( V_1, \ldots, V_k \) s.t.

- Shortest path between any \( u,v \in V_i \) using only vertices in \( V_i \) is at most \( d \) (strong diameters)
- The \# edges \( (u,v) \in E \) s.t. \( u \in V_i, v \in V_j \) and \( i \neq j \) is at most \( \beta m \) (few inter-component edges).
Sequentially, easy to compute using "ball growing"
- pick an uncovered vertex \( v \)
- repeatedly grow log \( \beta \) "ball" by \( \text{BFS-ing from } v \) until \( \beta \) fraction of the internal edges is reached. Remove that ball once this condition is met.

- Not hard to see that b.c. of the stopping condition, the \# cut edges is guaranteed to be \( \leq \beta m \). (Few inter-comp edges)

- What about diameter?
  Each new BFS level is added b.c. stopping criteria not yet applicable; i.e. more than \( \beta \)-fraction of internal edges on the boundary.

\[
\# \text{ boundary } > \beta m^* \Rightarrow m^* + \beta m^* = (1+\beta)m^*
\]

\[
\Rightarrow \text{ in each BFS level, the num. of internal nodes grows by a } (1+\beta) \text{ factor.}
\]

This can happen at most \( \log_{1+\beta} m = O\left(\frac{\log m}{\beta}\right) \)

\[
\Rightarrow \text{ seq alg gives a } (\beta, O(\log \frac{m}{\beta})) \text{ decomposition}
\]

using \( O(n+m) \) time.

Since the balls are inspected one after the other, the algorithm (in the worst case) is fully sequential.

---

Next: We will see a fully parallel, work-efficient alg. by Miller, Peng, Xu [MPX]

MPX gives a \( (\beta, O(\log \frac{m}{\beta})) \)-decomp in
- \( O(m+n) \) expected work
- \( O(\log m) \) depth whp.
Main idea is to grow multiple balls (carefully) in parallel from different vertices. We will mimic the sequential process, but carefully handle conflicts.

- **Challenge (1):** which balls do we grow in parallel?
  - Idea: use "exponentially-shifted" start times to start growing from $v$.

- **Challenge (2):** how to prove few-cut-edges property? Strong diameter?
  - Idea: use properties of exp. dist.

**Exp dist quick refresher:**

- $X$ non-negative C.R.V. is exponential ($\beta$) if
  \[ P(x = x) = \beta e^{-\beta x} \]

- CDF of $X$ is (integrate)
  \[ P(x \leq x) = 1 - e^{-\beta x} \]

- Memorylessness property:
  \[ P(x > a + b \mid x > a) = P(x > b) \]

  i.e. if we look at the tail of the distribution past a point $a$ and rescale it so that the total $p(a) = 1$, the dist. are the same.

---

How shall we decide when to start growing a ball at $v$?

→ Sample $S_v \sim \text{Exp} (\beta)$. If we use $S_v$ as start time, then almost all vertices start in the first few timesteps.

Instead, use "shifted" start times: $T_v = S_{\max} - S_v$

→ Exp dist is subtracted, so start time have a backward exponential dist (increases over time).
In particular,
- very few vertices start in first unit of time
- exponentially growing number start on each subsequent timestamp.

Which ball should \( v \) be assigned to?

\[
C_u \leftarrow \arg \min_{v \in V} (T_v + d(u,v))
\]

Shifted distance.

- Ignoring whether this generates a \((\beta,d)\)-decomp, can we implement efficiently? Turns out to be relatively easy, using simultaneous BFSs (aka multi-BFSs).

\[
\text{parfor } v \in V : S_v \leftarrow \text{Exp}(\beta)
\]

\[
\delta_{\max} \leftarrow \max_{v \in V} S_v
\]

\[
\text{parfor } v \in V :
T_v \leftarrow \delta_{\max} - S_v
\]

\[
C_v \leftarrow \text{unvisited} \quad \text{// initially all vertices unvisited}
\]

\[
\gamma_v \leftarrow \infty \quad \text{// how the balls compete to see which one visits a vertex}
\]

\[
I \leftarrow 0 \quad \text{// visited/started balls}
\]

\[
r \leftarrow 1
\]

\[\text{while } (I < |V|):
\]

\[
F \leftarrow F \cup \{ v \in V : (T_v < r) \land (C_v = \text{unvisited}) \} \quad \text{BFS / ball}
\]

\[
l \leftarrow l + |F|
\]

\[
\text{edge-map}(G,F,\lambda(u,v)).
\]

\[
\text{if } (C_v = \text{unvisited})
\]

\[
c \leftarrow C_u
\]

\[
\text{WriteMin}(\gamma_v, T_c - \lfloor T_c \rfloor)
\]

\[
F \leftarrow \text{edge-map}(G,F,\lambda(u,v)).
\]

\[
r \leftarrow r + 1
\]

\[
c \leftarrow C_u
\]

\[
\text{if } (\gamma_v = T_c - \lfloor T_c \rfloor)
\]

\[
C_v \leftarrow c
\]

\[
\text{return true}
\]

\[
\text{return false}
\]
Let's first analyze work/span:

- # rounds \( r \) \leq \lceil \frac{S_{\text{max}}}{\beta} \rceil \) since all vertices will be either found or added by this point.

- Sorting centers by start time can be done in parallel in \( O(n) \) work and \( O(\log(n) + S_{\text{max}}) \) depth (by using parallel counting sort.

See earlier lecture:

- \( n \) int's in \([0,m]\) in
- \( O(n) \) work, \( O(\log(n) + m) \) span.

- Each vertex appears in the frontier once, so each edge visited at most twice.

\[ \Rightarrow \text{total work is } O(ntm) \]

- Each round has \( O(\log n) \) depth. Therefore overall depth is \( O(S_{\text{max}} \log n) \).

Let's bound \( S_{\text{max}} \):

- What is the probability any vertex picks a shift \( > \frac{c\log n}{\beta} \)?

\[
P\left[ S_v > \frac{c\log n}{\beta} \right] = 1 - P\left[ S_v \leq \frac{c\log n}{\beta} \right] = 1 - \left( 1 - e^{-\frac{c\log n}{\beta}} \right) = \frac{1}{n^c}.
\]

\( \text{Exp} \text{ CDF} \)

Now, take the union-bound over all vertices. Prob that any vertex has \( S_v > \frac{c\log n}{\beta} \)

i.e., \( \Pr \left[ S_{\text{max}} > \frac{c\log n}{\beta} \right] \leq \frac{1}{n^c} \)

\[ \Rightarrow S_{\text{max}} \leq \frac{c\log n}{\beta} \text{ whp.} \]

Therefore overall depth is \( O\left( \frac{\log n}{\beta} \right) \) whp and \( O(c\log n) \) whp for constant \( \beta \).

This also shows the strong diameter result: cluster radius bounded by \( 2S_{\text{max}} = O\left( \frac{\log n}{\beta} \right) \) whp.
Next: only $m$ edges cut in expectation.

Actually show something a bit strange: the first $k$ balls that arrive at an edge (its "midpoint") or at a vertex are unlikely to arrive within a small window of time.

```
      midpoint of uv
          o---o
         u o s o s v
```

Lemma: For any vertex or midpoint of an edge, the probability that the smallest and $k$th smallest values of

$$\left\{ \frac{T_v + d(v,u)}{\text{startime dist}} : v \in V \right\}$$

differ by less than a $\delta$s $\leq (a\beta)^{k-1}$
Consider the case of first and second balls. Analyze this like a "horse race" from all vertices to u.
- length the horse runs is $\delta_v$. After $\delta_v$ units of time, it gets tired and stops.
- horse is penalized by $d(v,u)$

\[ i = \arg\max_{v \in V} \hat{D}(v) \]

Goal is to run as far from 0 as possible.

For an edge $(u,v)$ if $\hat{D}_i > \hat{D}_j < 1$ is exactly the event that $(u,v)$ is cut:

We can analyze this more generally for the top k winners:

Consider running the ball-growing process "backwards" in time. i.e. we gradually shrink balls, with points that start later being removed first.

Since we subtract exp dist to get start time, if we look at time going backwards, the distributions are proper exp dists.

Consider removing the the k-th ball. and consider the time $t$ at which it uncover u. k-1 balls still remain, covering u.

By memorylessness property of exp dist, they each are still distributed as $\text{Exp}(\beta)$, starting at $t$. Therefore the prob. any one of them is removed within a units of time is the CDF: $1 - e^{-t\beta} < \beta t$

$1 - e^{-t} < t$ since $e^x = 1 - x +$
The probability that all $k-1$ other balls removed within $a_i$ is therefore $< (\beta a_i)^{k-1}$.

For $k=2$, this is exactly the probability an edge is cut.

In forward time, if first ball arrives at time $t$, next ball at time $> t+1$ (at MP) then this edge is intra-cluster.

\[ \Rightarrow \text{prob edge is inter-cluster} \leq (\beta a_i)^{2-1} = \beta \]

By line of expectations, # inter-cluster edges $\leq \beta n$. 

KC: (1, 2) - ND
(r, s) - ND

\( c^{\ast}(r, s) \) p-complete for which \( c \)?

\[ 1 < c < s \]

\( c^{-1}(2, 3) \) - ND

map
k-core
directly

\[ a \]

\[ a \]