Will look at many graph algorithms from parallel perspective

- Connectivity
- Matchings
- Maximal Ind. Set
- Shortest Paths
- Densest Subgraph

Main question(s) as usual: can we design work-efficient algorithms?

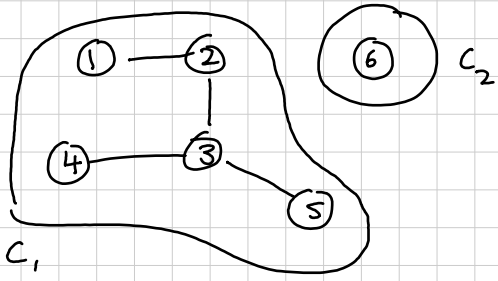
- Can we make the algorithms practical and theoretically-efficient?

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 algs + impls using BFS.

BFS(G, r): (G: not nec. undirected)

- compute a BFS tree rooted at r i.e. compute an array of Parents, $P[v]$

s.t. $P[v]$ = parent of v in BFS tree ($v \neq r$)

$P[r] = r$

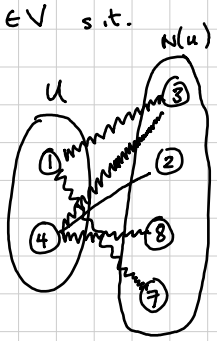
How do you solve BFS in parallel?

One way: just emulate sequential BFS and do it level-by-level.

- edge_map: primitive for traversal (useful beyond BFS).

```
edge_map(U, C, f) =
  parfor i ∈ [0 : |U|]
    N[i] = {v ∈ N⁺(U[i]) | f(u,v)}
  return Flatten(N)
```

- Take in a subset of vertices U, and an update function f.
- Return all vertices $v \in V$ s.t.
  - $(u,v) \in E$
  - $u \in U$
  - $f(u,v) = \text{true}$.



Output = [3, 7, 3, 8]

- 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.

Work: $O\left(|U| + \sum_{u \in U} \deg^+(u)\right)$

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 parent s.t. the tree formed by all $(u, \text{parent}[u])$ edges $\forall u \in V, u \neq r$ is a valid <u>BFS tree</u>

i.e. any non-tree edge (v, w) either lies in the same level, or between adjacent levels.

```
BFS(G, r) =
  F ← {r}  // initially only root in the frontier
  parfor u ∈ V:
    X[u] = 0
    P[u] = u
  X_r ← 1  // mark the root as visited
  while (|F| > 0):
    F ← edge_map(G, F, λ(u,v).
          if (test_and_set(X[u]) == 0) then P[u] = u and return true;
          return false;)
  return P
```

Def: The shortest path between $u, v \in V$ in G is denoted $\delta_G(u,v)$

Def: the radius of a vertex $u \in V$ is $\max_{v \in V} \delta_G(u,v) = \text{radius}_G(v)$

Def: the diameter of a graph G ($\text{diam}(G)$) is $\max_{v \in V} \text{radius}_G(v)$

- 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 many other related SSSP problems) efficiently in polylog(n) depth in general graphs.

What about graph probs. we used BFS for?

- For many probs, another powerful technique is applicable.

Low-Diameter Decomposition: (LDD)

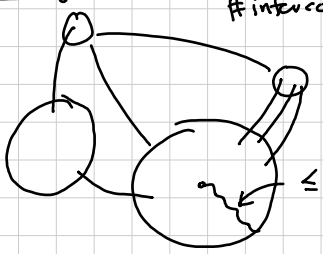
- idea: instead of searching from a single vertex, break graph up into some number of connected clusters s.t.
  (1) few edges are cut
  (2) internal diameters of each cluster are bounded (ensures that clusters can be efficiently explored in parallel; and generally a useful alg. property that algorithms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

More formally:

A (β, 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 diameter)
- The # edges $(u,v) \in E$ s.t. $u \in V_i$, $v \in V_j$ and $i \neq j$ is at most $\beta \cdot m$ (few inter-component edges).

pictorally:



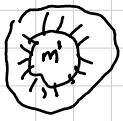
inter-component $\leq \beta \cdot m$

Apps include
- connectivity, spanners, hop-sets,
LSSTs (useful in sparse system solvers).
- First used in Dist. Comp; apps in metric embedding, linear system solvers, parallel algs.

Sequentially, easy to compute using "ball growing"
- pick an uncovered vertex v
- repeatedly grow ball by BFS'ing from v until # edges incident to current frontier is $\leq \beta$ fraction of the # internal edges. Remove this ball from G 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)

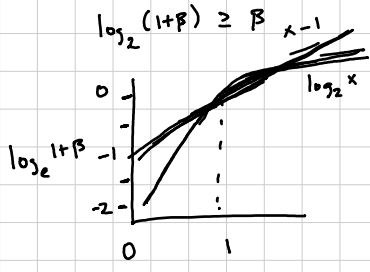
- What about diameter?
  - Each new BFS level is added b.c. stopping criteria not yet applicable; i.e. more than β-fraction of internal edges on the boundary



boundary $> \beta m'$ $\Rightarrow$ $m'' \geq m' + \beta m' = (1+\beta)m'$

$\Rightarrow$ in each BFS level, the num. of internal nodes grows by a $(1+\beta)$ factor.

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



$\Rightarrow$ Seq alg gives a $(\beta, O(\log n/\beta))$ decomposition using $O(n+m)$ time.

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

Bad example for seq?

Good example for seq?

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

MPX gives a $(\beta, O(\log n/\beta))$-decomp in
- $O(m+n)$ expected work and $O(\log^2 n)$ 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.

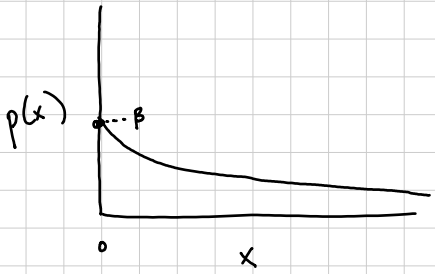
- Challenge (1): which balls do we grow in parallel?
  Idea: use "exponentially-shifted" start times to start growing from v.

- 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(β) if

$$Pr[X = x] = \beta e^{-\beta x}$$



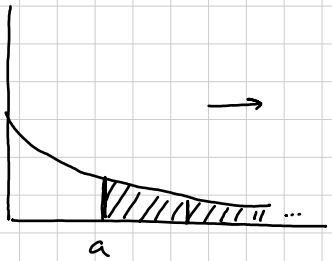
- CDF of X is (integrate)

$$Pr[X \le x] = 1 - e^{-\beta x}$$

- Memorylessness property:

$$Pr[X > a+b \mid X > a] = Pr[X > b]$$

i.e. if we look at the tail of the distribution past a point a and rescale it so that the total prob = 1, then dists are the same.



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

→ Sample $\delta_v \sim Exp(\beta)$. If we use δ_v as start times, then almost all vertices start in the first few timesteps

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

→ Exp dist is subtracted, so start times have a backwards exponential dist (increases over time).

In particular,

- very few vertices start in first unit of time
- exponentially growing number start on each subsequent timestep.

Which ball should v be assigned to?

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

($T_v + d(u,v)$: shifted distance.)

- Ignoring whether this generates a (β, d)-decomp, can we impl efficiently? Turns out to be relatively easy, using simultaneous BFS (also multi-BFS).

```
parfor v∈V : δ_v ← Exp(β)
δ_max ← max_{v∈V} δ_v
parfor v∈V:
  T_v ← δ_max − δ_v
  C_v ← unvisited      // initially all vertices unvisited
  γ_v ← ∞              // how the balls compete to see which one wins a vertex.
                       (based on earliest T_v)
l ← 0                  // # visited / started balls
r ← 1
while (l < |V|):
  F ← F ∪ {v∈V | (T_v < r) ∧ (C_v == unvisited)}   // vertices "wake up" and start a ball if ⌊T_v⌋ steps elapsed and not yet visited by another BFS / ball.
  l ← l + |F|
  edge_map(G, F, λ(u,v).
    if (C_v == unvisited)
      c ← C_u
      WriteMin(γ_v, T_c − ⌊T_c⌋))   // fractional part of T_c
  F ← edge_map(G, F, λ(u,v).
      c ← C_u
      if (γ_v == T_c − ⌊T_c⌋)
        C_v ← c
        return true
      return false
  r ← r+1
return C
```

Let's first analyze work/span:

- # rounds $(r) \leq \lceil \delta_{max} \rceil$ since all vertices will be either covered or added by this point.

- Sorting centers by start time easy to do in parallel in $O(n)$ work and $O(\log(n) + \delta_{max})$ depth (eg using parallel counting sort (see earlier lecture: n ints in $[0:m]$ in $O(n)$ work, $O(\log n + m)$ span. $m \to \delta_{max}$)

- Each vertex appears in the frontier once, so each edge visited at most twice
  $\Rightarrow$ **total work is $O(n+m)$**

- Each round has $O(\log n)$ depth. **Therefore overall depth is $O(\delta_{max} \log n)$**

Let's bound δ_{max}.

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

$$P\left[\delta_v > \frac{c\log n}{\beta}\right] = 1 - \underbrace{P\left[\delta_v \leq \frac{c\log n}{\beta}\right]}_{\text{Exp CDF}} = 1 - (1 - e^{-c\log n}) = \frac{1}{n^c}.$$

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

i.e. $$P_r\left[\delta_{max} > \frac{c\log n}{\beta}\right] \leq \frac{1}{n^{c-1}}$$

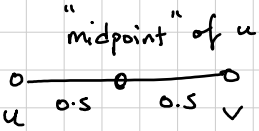
$\Rightarrow \delta_{max} \leq \frac{c\log n}{\beta}$ whp.

Therefore overall depth is $O(\frac{\log^2 n}{\beta})$ whp and $O(\log^2 n)$ whp for constant β.

This also shows the <u>strong diameter result</u> Cluster radius bounded by $2\delta_{max}$ $= O(\frac{\log n}{\beta})$ whp.

Next: only βm edges cut in expectation.

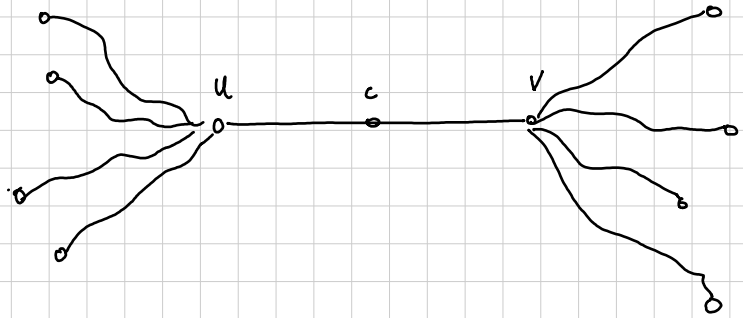
Actually show something a bit stronger: the first k balls that arrive at an edge (its "midpoint") or at a vertex are unlikely to arrive within a small window of time.



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

$$\{ \underbrace{T_v}_{\text{start time}} + \underbrace{d(v,u)}_{\text{dist}} : v \in V \}$$

differ by less than a is $\leq (a\beta)^{k-1}$



$$T_v = \delta_{max} - \delta_v$$

$$D_v(u) = T_v + d(v,u)$$

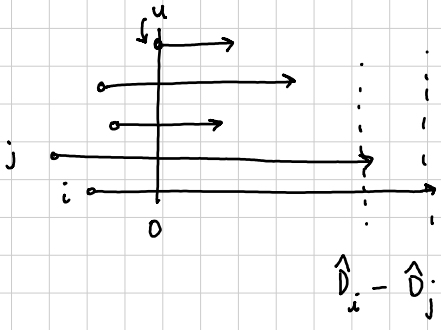
$$\hat{D}_v(u) = \delta_{max} - D_v(u) = \delta_v - d(v,u)$$

$$\arg\min_{v \in V} \overbrace{\left(\delta_{max} - \delta_v + d(v,u)\right)}^{D_v(u)}$$

$$\arg\max_{v \in V} \left(\delta_v - d(v,u)\right)$$

Consider the case of first and second balls. Analyze this like a "horse race" from all vertices to u.

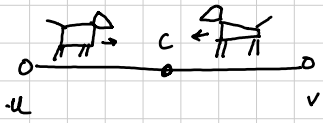
- length the horse runs is δ_v. After δ_v units of time, it gets tired and stops.
- horse is penalized by $d(v,u)$



$i = \underset{v \in V}{\text{argmax}}\ \hat{D}(v)$

Goal is to run as far from 0 as possible.

For an edge (u,v) $\hat{D}_i - \hat{D}_j < 1$ 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. distr.

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

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

$1 - e^{-x} < x$ since $e^x = 1 - x +$

The prob. that all $k-1$ other balls removed within a is therefore $< (\beta a)^{k-1}$.

For $k=2$, $a=1$, 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.

$\Rightarrow$ prob edge is inter-cluster $\leq (\beta a)^{2-1} = \beta$

By lin. of expectations, # inter-cluster edges $\leq \beta m$.

KC: $(1,2)$-ND

(r,s)-ND

c-$(1,2)$ and above is P-complete (for any constant $c \geq 3$)

ATLAS

vector retrieval models

c'-(r,s) P-complete for which c?

$1 < r < s$

c'-$(2,3)$-ND

map k-core directly