CRDT

Conflict-Free Replicated Data Type

1. Causal History – State based

For any replica x_i of x:

- Initially $C(x_i) = \phi$
- After operation f, $C(f(x_i)) = C(x_i) \cup \{f\}$
- After merge states x_i, x_j, $C(merge(x_i, x_j)) = C(x_i) \cup C(x_j)$
- Happens before: $f \to g \Leftrightarrow C(f) \subset C(g)$

2. Eventual Convergence

Two replicas x_i & x_j of x converge eventually if the following conditions are met:

- Safety: $\forall i, j : C(x_i) = C(x_j)$ implies that the abstract states of i & j are equivalent
- Liveness: $\forall i, j : f \in C(x_i)$ implies eventually $f \in C(x_j)$

3. Partial Order $\sqsubseteq_v$

Binary relation that is:

- reflexive : $a \sqsubseteq a$
- antisymmetric : $a \sqsubseteq b$, $b \not\sqsubseteq a$
- transitive : $a \sqsubseteq b$, $b \sqsubseteq c$, $a \sqsubseteq c$

4. Least Upper Bound (LUB) $\sqcup_v$

$m = x \sqcup_v y$ is a LUB of $\{x, y\}$ under $\sqsubseteq_v$ iff $x \sqsubseteq_v m$ & $y \sqsubseteq_v m$ and there is no $m' \sqsubseteq_v m$ such that $x \sqsubseteq_v m'$ and $y \sqsubseteq_v m'$

- Commutative $\Rightarrow x \sqcup_v y =_v y \sqcup_v x$
- idempotent $\Rightarrow x \sqcup_v x =_v x$
- associative $\Rightarrow (x \sqcup_v y) \sqcup_v z =_v x \sqcup_v (y \sqcup_v z)$

5. Join Semilattice

An ordered set $(S, \sqsubseteq_v)$ is a join semilattice iff $\forall x, y \in S$, $\underline{x \sqcup_v y \text{ exists}}$

Doubts

1. Can I assume a sparse graph?

$$V \leq E << V \times V$$

2. What does "merge with concurrent changes from other graph/replica" mean?

3. Is there a particular Data Type we want to store in the graph?

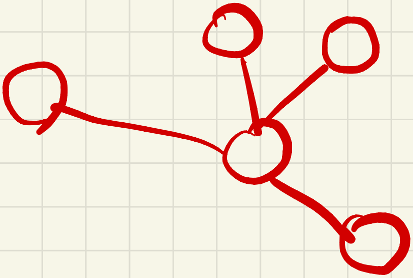
4. Should merge graph return a new graph or change in place?

5. Can remove before adding?

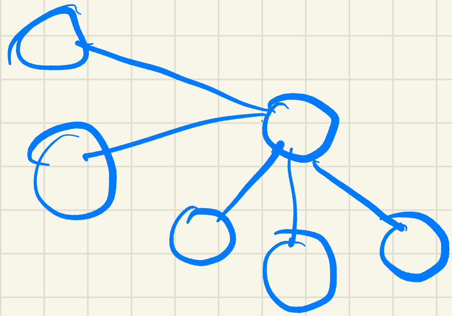
6. Prioritize addEdge or removeVertex?

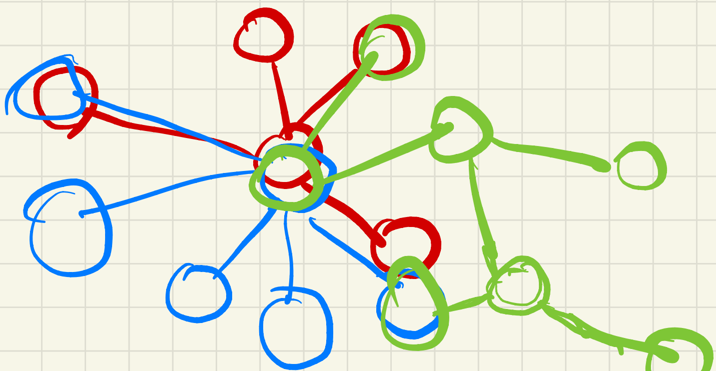
7. Should be an undirected or directed graph?

Graph A

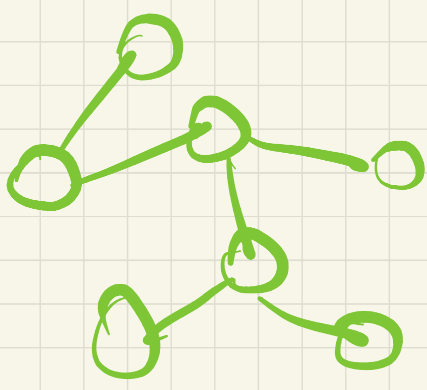


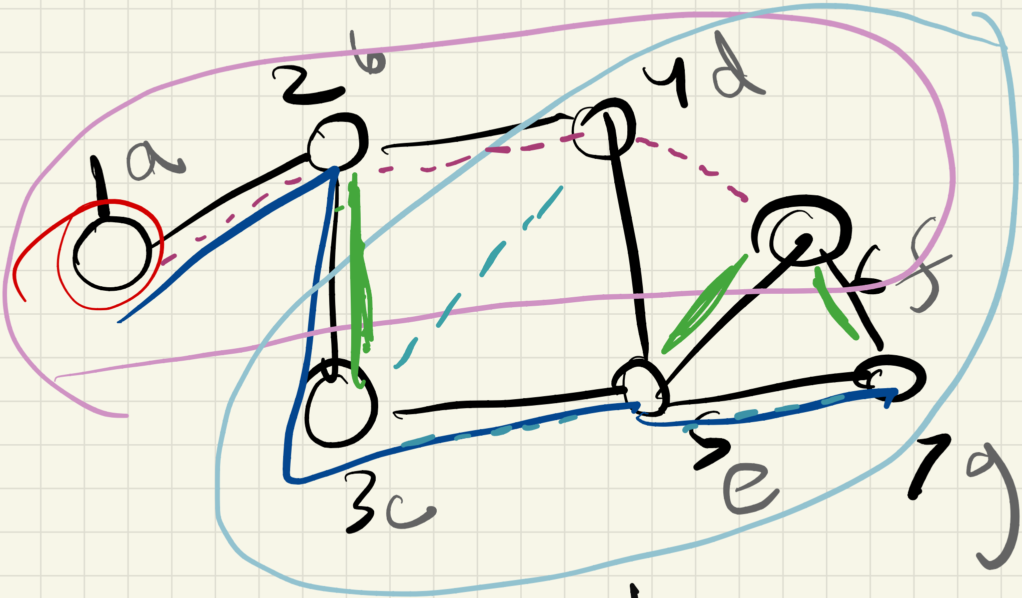
Graph B

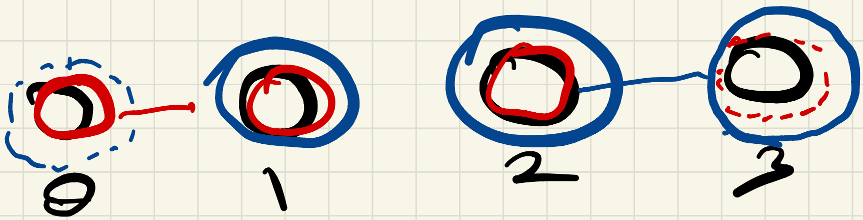




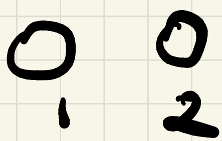
Graph C

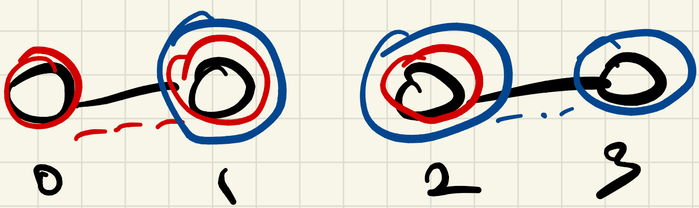




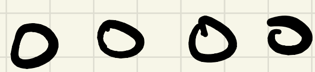


merge ⇒





merge ⇒



addVertex(u) addEdge(u, v)

addVertex(u) removeEdge(u, v)

removeVertex(u) addEdge(u, v)

removeVertex(u) removeEdge(u, v)