15-210
Parallel and Sequential Algorithms and Data Structures

Lecture 16

Graph Contraction

Synopsis

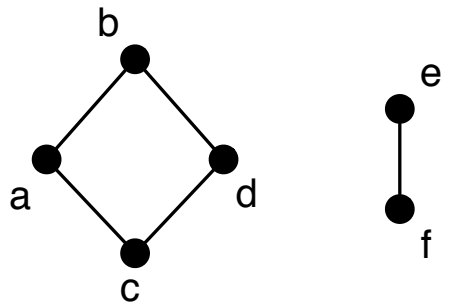
- Graph Contraction
- Finding Connected Components
- Edge Contraction
- Star Contraction

Motivation

- Most graph search algorithms were either
  - sequential, or
  - had span dependent on the diameter.
- Can we make these algorithms more parallel?
  - Polylogarithmic span: span is bounded by a polynomial in $\log n$
- We will look at contraction as a way to build parallel algorithms for some graph problems:
  - Graph Connectivity
  - Spanning Trees

Graph Connectivity

- Two vertices in an undirected graph are connected if there is a path between them.
- A graph is connected if all pairs of vertices are connected.
- The graph connectivity problems partitions a graph into its maximal connected subgraphs.



has two connected subgraphs: $\{a, b, c, d\}$ and $\{e, f\}$

Graph Connectivity

- BFS or DFS
  - Identify vertices of a connected component
  - Identify *all* connected components!
- BFS could be parallel but has span $\propto$ diameter d
- Each connected component needs to be done sequentially!

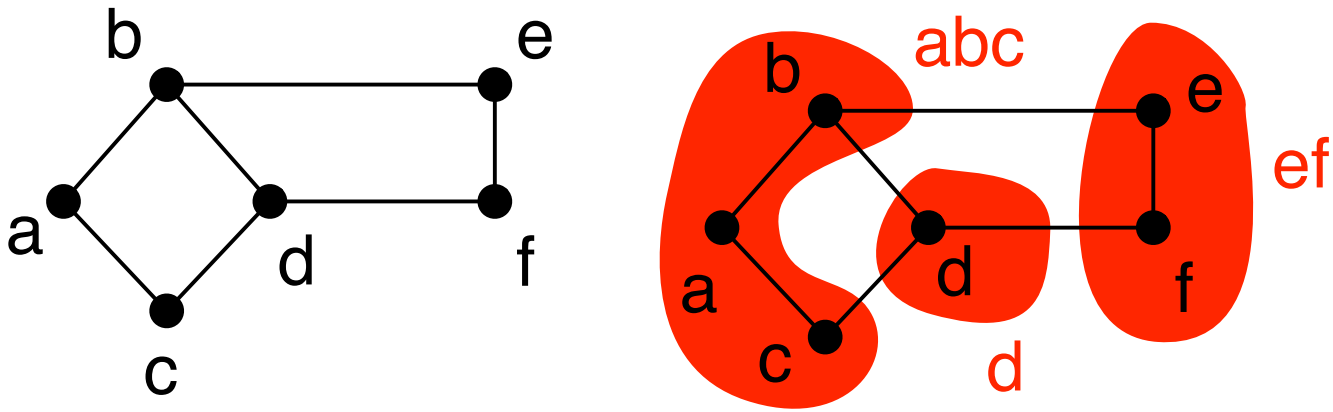
Graph Contraction

- Problem $\rightarrow$ Smaller Problem
- Shrink the size of the graph and solve the connectivity problem on the small graph.
  - Different components can be handled in parallel!
- Applicable to other problems
  - Spanning Trees
  - Minimum Spanning Trees

Graph Contraction

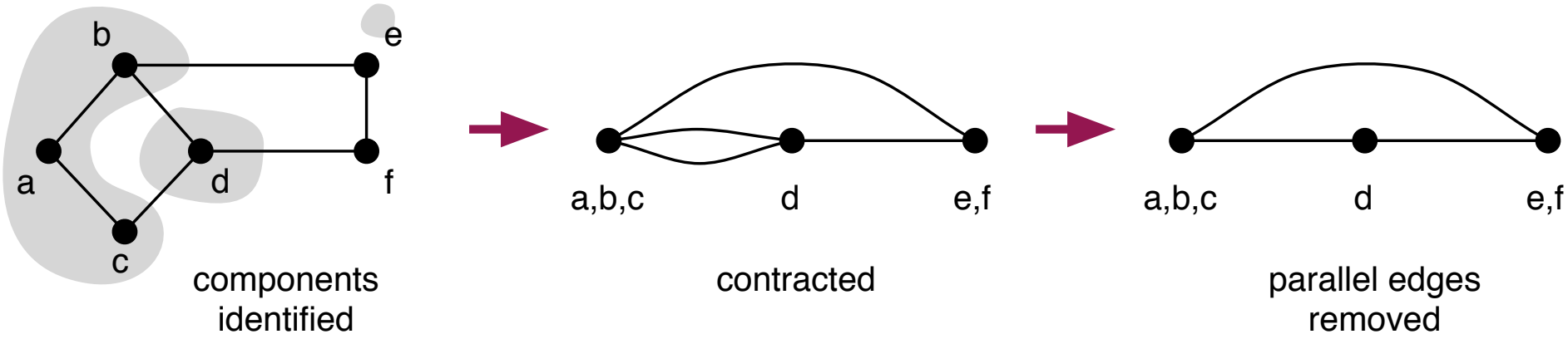
$$\texttt{contract} : \texttt{graph} \rightarrow \texttt{partition}$$

- Takes a graph $G(V, E)$ and returns a partitioning of V into connected subgraphs.
  - Not necessarily maximally connected subgraphs (yet)
  - But vertices in a partition are connected.

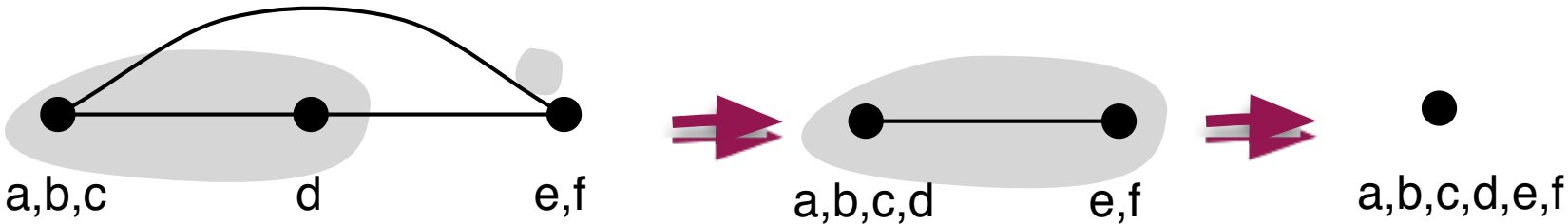


$$\{\{a, b, c\}, \{d\}, \{e, f\}\}$$

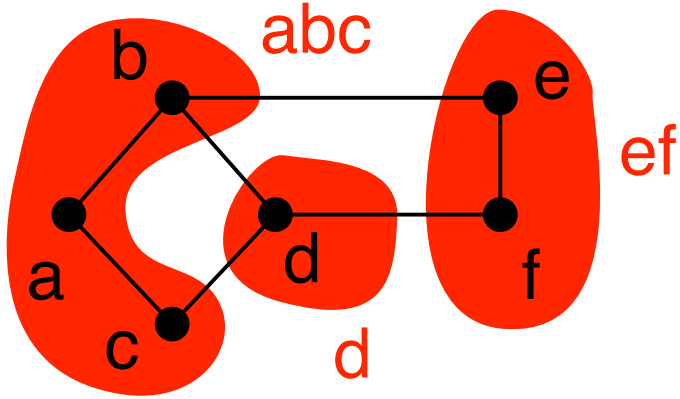
Graph Contraction



- If the graph contracts on each round, eventually each maximal connected component will shrink down to a single vertex!



Representing Partitions



$$\downarrow$$

$$\{\{a, b, c\}, \{d\}, \{e, f\}\}$$

$$\downarrow$$

$$(\{a, d, e\}, \{a \mapsto a, b \mapsto a, c \mapsto a, d \mapsto d, e \mapsto e, f \mapsto e\})$$

Contracting Graphs

```
1  fun contractGraph((V, E), i) =
2  if |E| = 0 then (V, E)
3  else let
4       (V', P) = partitionGraph((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6  in
7     contractGraph((V', E'), i + 1)
8  end
```

- Ignore i for the time being!
- V' is the set of representative vertices
- P maps every $v \in V$ to a $v' \in V'$.
- E' is the set of edges in the contracted graph.
  - Self-loops are removed!

Counting Components

```
1  fun numComponents((V, E), i) =
2  if |E| = 0 then |V|
3  else let
4       (V', P) = partitionGraph((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6  in
7     numComponents((V', E'), i + 1)
8  end
```

Computing Components

```
1  fun components((V, E), i) =
2  if |E| = 0 then {v ↦ v : v ∈ V}
3  else let
4       (V', P) = partitionGraph((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6       P' = components((V', E'), i + 1)
7  in
8     {v ↦ P'[P[v]] : v ∈ V}
9  end
```

Computing Components

```
1  fun components((V, E), i) =
2  if |E| = 0 then {v ↦ v : v ∈ V}
3  else let
4       (V', P) = partitionGraph((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6       P' = components((V', E'), i + 1)
7  in
8     {v ↦ P'[P[v]] : v ∈ V}
9  end
```

- Base case: Every vertex maps to itself!

Computing Components

```
1  fun components((V, E), i) =
2  if |E| = 0 then {v ↦ v : v ∈ V}
3  else let
4       (V', P) = partitionGraph((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6       P' = components((V', E'), i + 1)
7  in
8     {v ↦ P'[P[v]] : v ∈ V}
9  end
```

- (Recursively) find components of the contracted graph

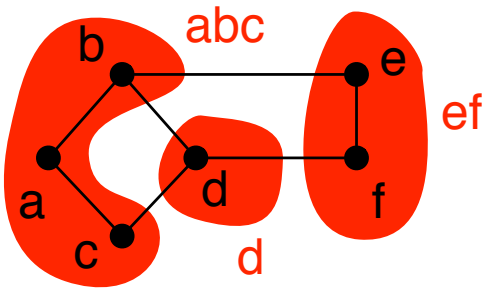
Computing Components

```
1  fun components((V, E), i) =
2  if |E| = 0 then {v ↦ v : v ∈ V}
3  else let
4       (V', P) = partitionGraph((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6       P' = components((V', E'), i + 1)
7  in
8     {v ↦ P'[P[v]] : v ∈ V}
9  end
```

- Map each vertex to the representative vertex of its partition!

Computing Components

```
1  fun components((V, E), i) =
2  if |E| = 0 then {v ↦ v : v ∈ V}
3  else let
4       (V', P) = partitionGraph((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6       P' = components((V', E'), i + 1)
7  in
8     {v ↦ P'[P[v]] : v ∈ V}
9  end
```



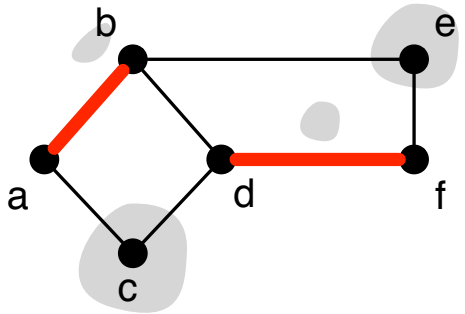
- After 4: $V' = \{a, d, e\}$
  $P = \{a \mapsto a, b \mapsto a, c \mapsto a, d \mapsto d, \quad e \mapsto e, f \mapsto e\}$
- After 6: $P' = \{a \mapsto a, d \mapsto a, e \mapsto a\}$
- 8 returns: $\{a \mapsto a, b \mapsto a, c \mapsto a, d \mapsto a, e \mapsto a, f \mapsto a\}$

Implementing Contract

- Edge Contraction:Only pairs of vertices connected by an edge are contracted.
- Star Contraction: Vertices around a "center star" collapse to the "star"
- Tree Contraction: disjoint trees within the graph are identified and vertices in a tree are collapsed to the root.
- Parallel
- Reduce graph size (vertices/edges?) by a constant factor every round.
  - Will lead to $O(\log n)$ rounds!.

Edge Contraction

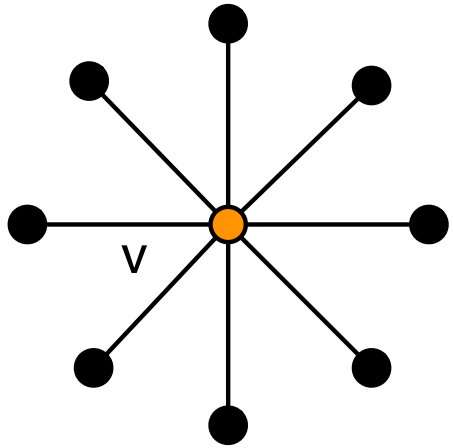
- Find disjoint edges – edges can not share vertices.



- Vertex matching problem
- Can be done in parallel
  - Each edge picks a random priority in $[0, 1]$
  - Any edge which has highest priority for both vertices gets selected.
- It turns out this has some problems!

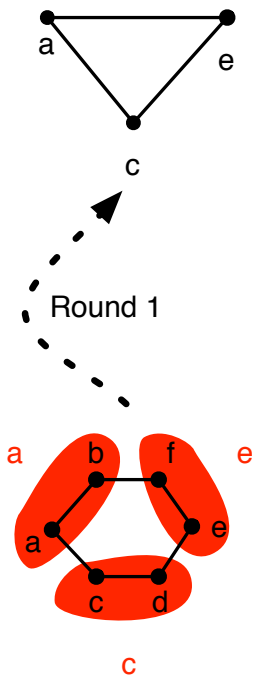
Edge Contraction

- Consider a graph like



- How many edges can be contracted each round?
- How many rounds are needed to contract to 1 node?
- Not very parallel!

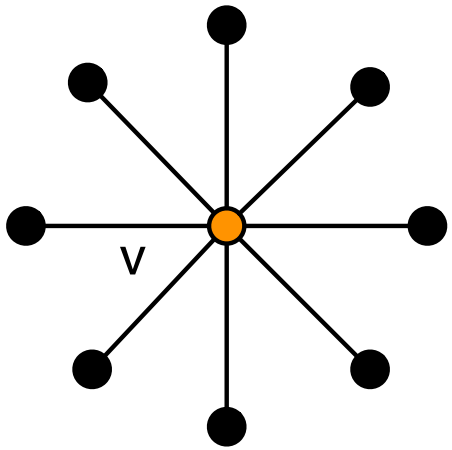
Edge Contraction for Cycle Graphs



- Edges flip a coin
- Edges get a random number

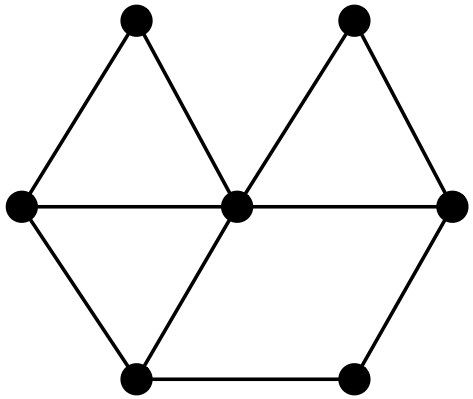
Star Graphs

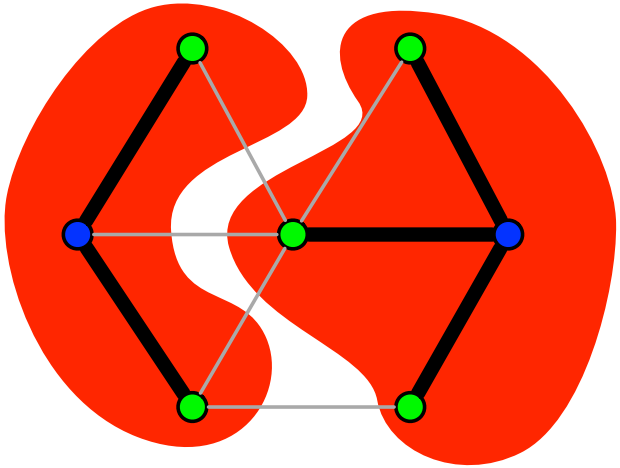
- A star (sub)graph $G = (V, E)$ is an undirected graph with a center vertex $v \in V$, and a set of edges $E = \{\{v, u\} : u \in V \setminus \{v\}\}$.



Star Contraction

- Star subgraphs can be contracted in parallel!





- How do we find disjoint stars?

Finding Disjoint Stars

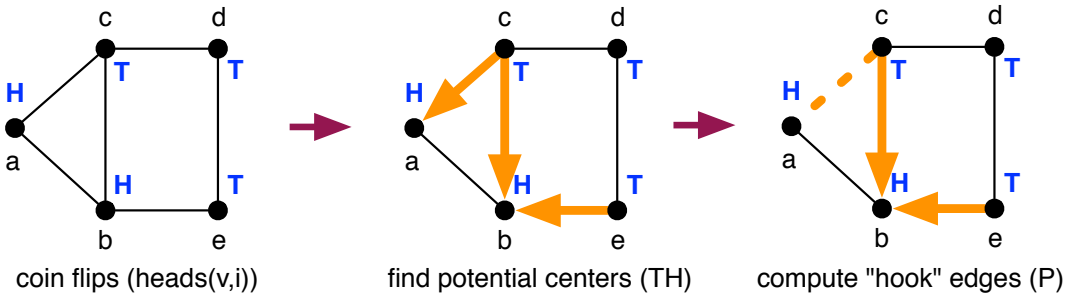
- Each vertex throws a coin
  - Heads $\rightarrow$ vertex is a star-center
  - Tails $\rightarrow$ vertex is a *potential satellite* (Why *potential*?)
- Each satellite then selects a center.
- What is the probability that a vertex with degree d is removed?

Random Coin Tosses

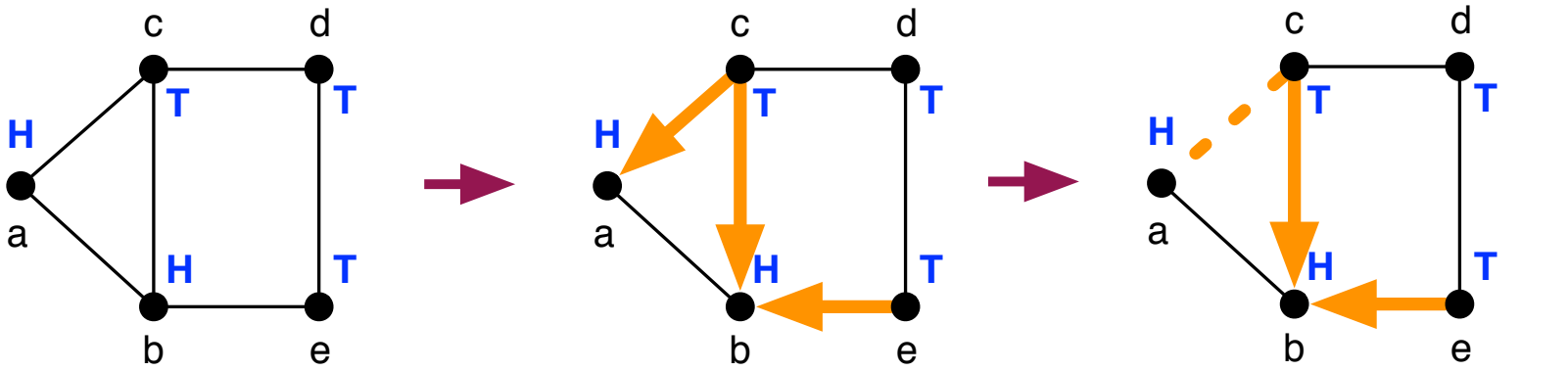
- Pretend each vertex has a potentially infinite sequence of random coin flips
- $\texttt{heads}(v, i) : \texttt{vertex} \times \texttt{int} \to \texttt{bool}$ provides access to these coin tosses.
- This can be implemented with a pseudorandom number generator.

Star Contraction

```
1  fun starPartition(G = (V, E), i) =
2  let
3      % select edges that go from a tail to a head
4        TH = {(u, v) ∈ E | ¬heads(u, i) ∧ heads(v, i)}
5      % make mapping from tails to heads, removing duplicates
6        P = ∪(u,v)∈TH {u ↦ v}
7      % remove vertices that have been remapped
8        V' = V \ domain(P)
9      % Map remaining vertices to themselves
10       P' = {u ↦ u : u ∈ V'} ∪ P
11 in (V', P') end
```



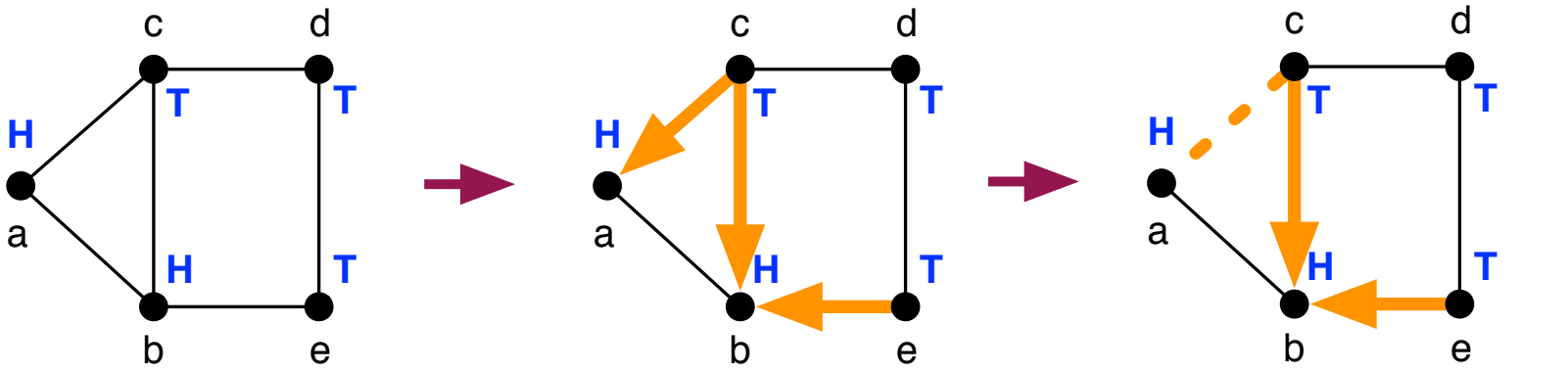
Star Contraction



$$TH = \{(u, v) \in E \mid \neg heads(u, i) \wedge heads(v, i)\}$$

- $TH = \{(c, a), (c, b), (e, b)\}$.

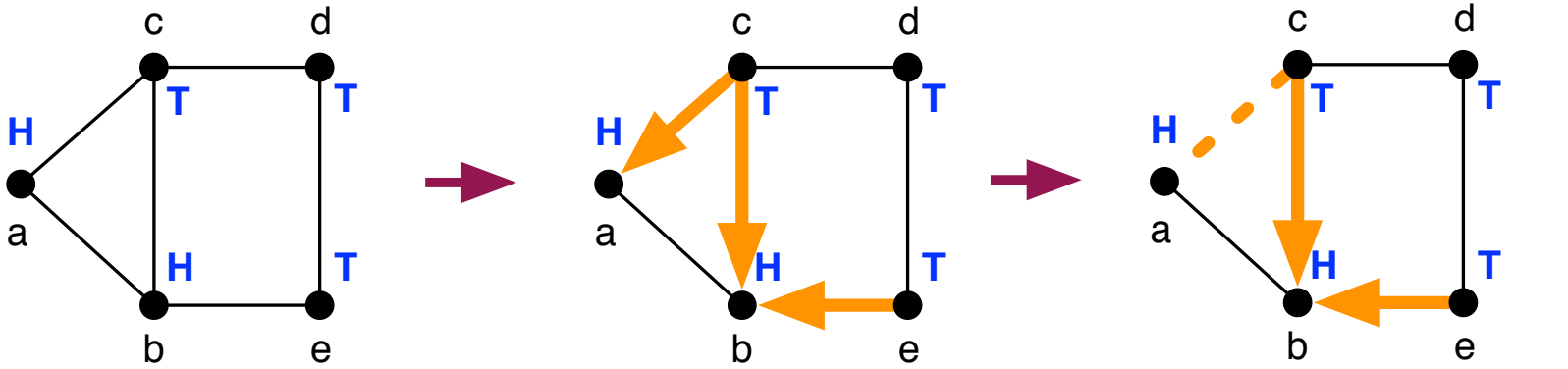
Star Contraction



$$P = \cup_{(u,v) \in TH} \{u \mapsto v\}$$

- $TH = \{(c, a), (c, b), (e, b)\}$
- $P = \{c \mapsto b, e \mapsto b\}$

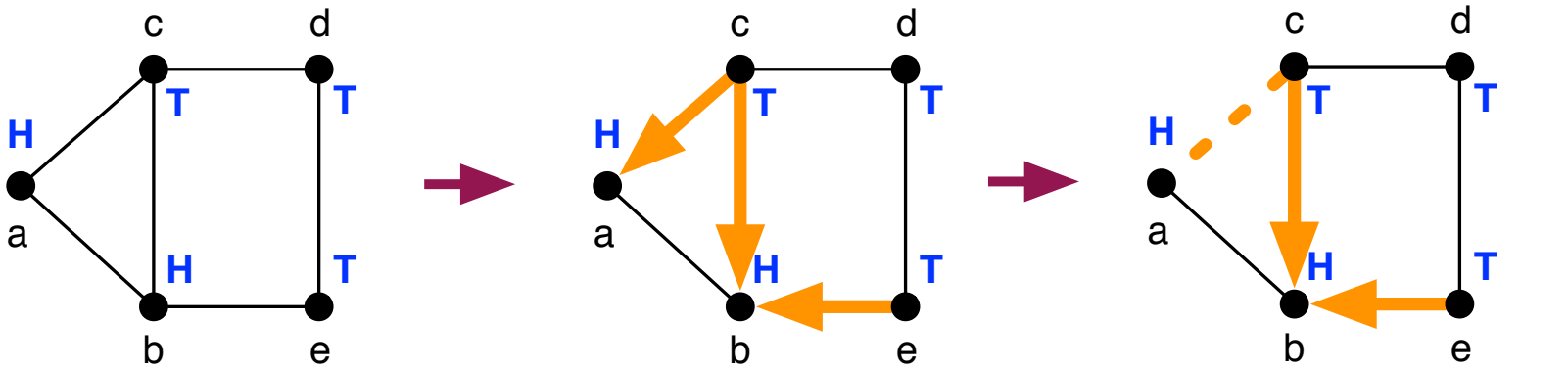
Star Contraction



$$V' = V \setminus \mathit{domain}(P)$$

- $P = \{c \mapsto b, e \mapsto b\}$
- $\mathit{domain}(P) = \{c, e\}$
- $V' = \{a, b, d\}$

Star Contraction



$$P' = \{u \mapsto u : u \in V'\} \cup P$$

- $P = \{c \mapsto b, e \mapsto b\}$, $V' = \{a, b, d\}$
- $P' = \{a \mapsto a, b \mapsto b, c \mapsto b, d \mapsto d, e \mapsto b\}$

Analysis of Star Contraction

Lemma

For a graph G with n non-isolated vertices, let X_n be the random variable indicating the number of vertices removed by `starPartition`$(G, \_)$. Then, $\mathbf{E}[X_n] \geq n/4$.

- H_v: vertex v comes up heads, T_v: vertex v comes up tails
- R_v: vertex v is removed in contraction
- v has at least one neighbor u.
- $T_v \wedge H_u$ implies R_v
  - If v is a tail, join $u's$ star or some other star.
- $\mathbf{Pr}[R_v] \geq \mathbf{Pr}[T_v]\mathbf{Pr}[H_u] = 1/4$
- Expected total $\geq n/4$

Analysis of Star Contraction

```
1   fun starPartition(G = (V, E), i) =
2   let
3       % select edges that go from a tail to a head – O(m) work, O(1) span
4         TH = {(u, v) ∈ E | ¬heads(u, i) ∧ heads(v, i)}
5       % make mapping from tails to heads, removing duplicates
6       % O(n) work, O(log n) span
7         P = ∪(u,v)∈TH {u ↦ v}
8       % remove vertices that have been remapped
9       % O(n) work, O(log n) span
10        V' = V \ domain(P)
11      % Map remaining vertices to themselves -O(n) work, O(log n span
12        P' = {u ↦ u : u ∈ V'} ∪ P
13  in (V', P') end
```

- n nodes, m edges
- $O(n + m)$ work, $O(\log n)$ span.

Analysis of Connectivity

```
1  fun numComponents((V, E), i) =
2  if |E| = 0 then |V|
3  else let
4       (V', P) = starPartition((V, E), i)
5       E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
6  in
7     numComponents((V', E'), i + 1)
8  end
```

- $S(n) = S(n') + O(\log n)$
- $n' = n - X_n$ and $\mathbf{E}[X_n] = n/4$, so $\mathbf{E}[n'] = 3n/4$
- $S(n) \in O(\log^2 n)$

Analysis of Connectivity

- We can remove a constant fraction of vertices every round.
- For each vertex removed, we remove at least one edge.
- Consider a hypothetical contraction

round	vertices	edges
1	n	m
2	$n/2$	$m - n/2$
3	$n/4$	$m - 3n/4$
4	$n/8$	$m - 7n/8$

- Number of edges does not go below $m - n$.

Analysis of Connectivity

$$W(n, m) \leq W(n', m) + O(n + m),$$

- As before, $\mathbf{E}[n'] = 3n/4$, so $\mathbf{E}[W(n, m)] \in O(n + m \log n)$

Tree Contraction

- Identify disjoint trees and contract them.
- For every tree of t vertices contracted, $t - 1$ edges are removed.
- Number of edges also go down geometrically at every round.
- Leads to $O(m)$ work and $O(\log^2 n)$ span.