Snap-Stabilizing Prefix Tree for Peer-to-Peer Systems

Cédric Tedeschi[1]
Eddy Caron[1], Frédéric Desprez[1], Franck Petit[2]

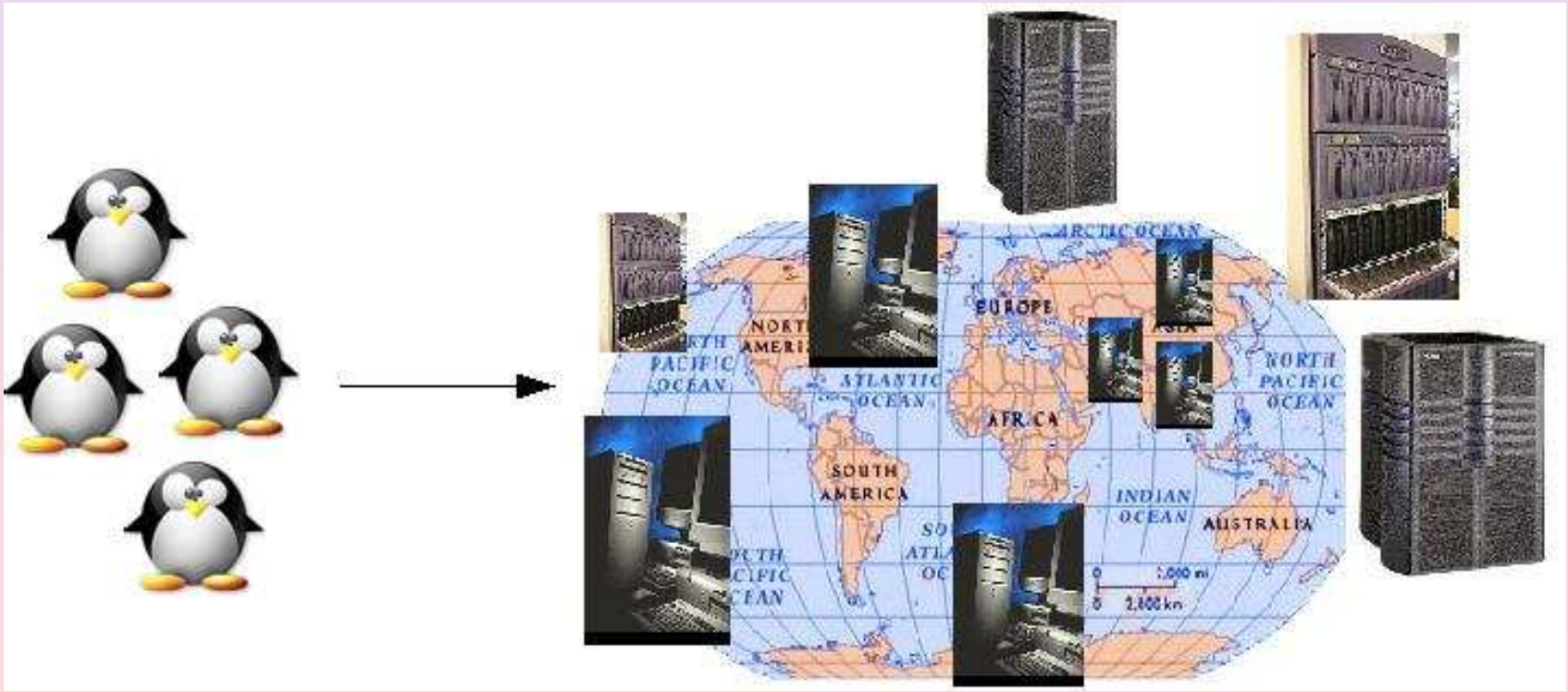
1 - Université de Lyon. LIP laboratory. UMR CNRS-ENS Lyon-UCB Lyon-INRIA 5668
2 - LaRIA laboratory. CNRS-UPJV

∼
Supported by ANR-05-CIGC-11

SSS 2007 - November 16, 2007

Context

- Service discovery in grid computing
  - Service (binary file, library) installed on servers
  - Servers declare their services, client discovers them
  - Need for maintaining this information



Context

- Service discovery in grid computing
  - Service (binary file, library) installed on servers
  - Servers declare their services, client discovers them
  - Need for maintaining this information
- Target platforms
  - large scale
  - no central infrastructure
  - dynamic joins and leaves of nodes
- P2P systems
  - Purely decentralized algorithms
  - Scalable algorithms to retrieve objects
  - Fault-tolerance

Trie-based overlays

A promising way to store and retrieve services

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Approaches
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - DLPT (Caron, Desprez, Tedeschi – 2005)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
- Drawback : fault-tolerance based on replication
  - Costly
  - Does not recover after arbitrary failures

Trie-based overlays

A promising way to store and retrieve services

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Approaches
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - DLPT (Caron, Desprez, Tedeschi – 2005)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
- Drawback : fault-tolerance based on replication
  - Costly
  - Does not recover after arbitrary failures

Trie-based overlays

A promising way to store and retrieve services

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Approaches
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - DLPT (Caron, Desprez, Tedeschi – 2005)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
- Drawback : fault-tolerance based on replication
  - Costly
  - Does not recover after arbitrary failures

Trie-based overlays

A promising way to store and retrieve services

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Approaches
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - DLPT (Caron, Desprez, Tedeschi – 2005)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
- Drawback : fault-tolerance based on replication
  - Costly
  - Does not recover after arbitrary failures

It remains the best effort approach.

Trie-based overlays

A promising way to store and retrieve services

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Approaches
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - DLPT (Caron, Desprez, Tedeschi – 2005)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
- Drawback : fault-tolerance based on replication
  - Costly
  - Does not recover after arbitrary failures

It remains the best effort approach. **Self-stabilization.**

Contributions

A snap-stabilizing protocol maintaining a prefix tree.

- Built in a *P2P oriented* architecture model
- Proven in the state model to be *snap-stabilizing*
- Worst case complexities
- Simulation results

Outline

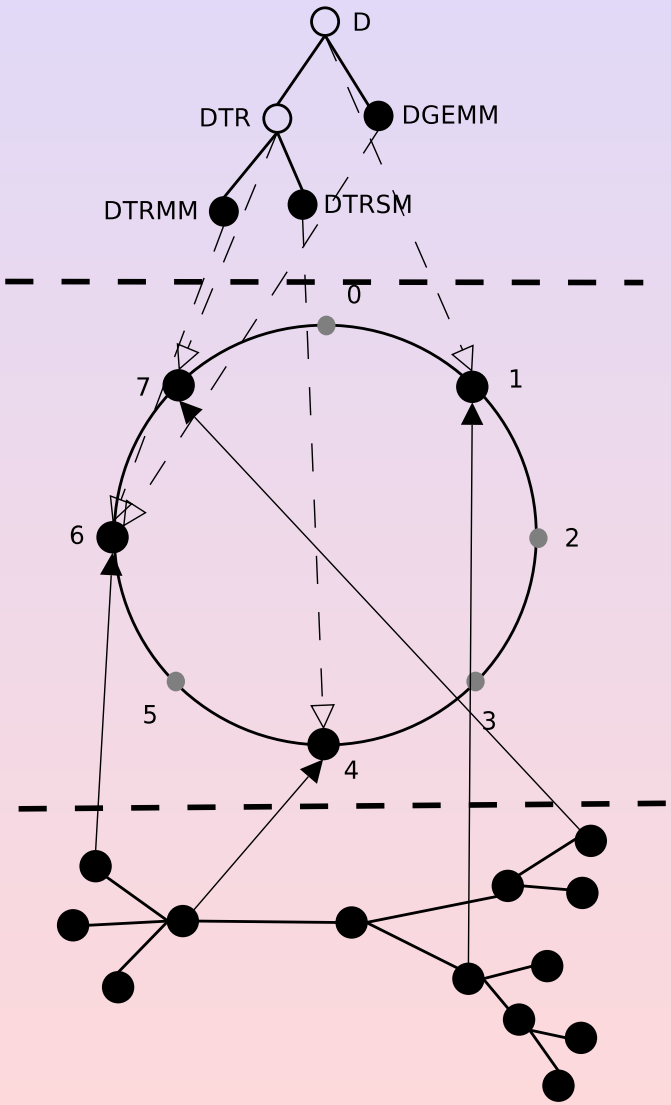
Outline

Architecture model : A two-layer P2P network

- A *physical* network of *peers*
  - P_1 can communicate with P_2 provided that P_1 *knows* P_2.
  - Each peer runs one or more logical *nodes*
- A *logical* tree of *nodes*
  - Logical prefix tree distributed among peers
  - Our protocol is run inside these nodes
  - Susceptible to changes during its reconstruction

Architecture model : A two-layer P2P network

- A *physical* network of *peers*
- A *logical* tree of *nodes*



State model (1/2)

- Nodes can read the variables of some other nodes and write only to their own variables
- Actions : $< guard > \rightarrow < statement >$
  - $< guard >$ bool. expr. of variables of p and its neighbors
  - $< statement >$
    - Executed only if its $< guard > = true$
    - Updates one or more variables of p
- The *state* of a node p is defined by the values of its variables
- The *configuration* of the system is the product of the states
- Unfair and distributed scheduling daemon

State model (2/2)

$\mathcal{C}$, the set of possible configurations of the system
$\mapsto$, a relation on $\mathcal{C}$.

A *computation* is a maximal sequence of configurations $e = (\gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1})$ such that for $i \geq 0$, $\gamma_i \mapsto \gamma_{i+1}$ or γ_i is a terminal configuration.

$\mathcal{E}$, the set of possible computations
$\mathcal{E}_\alpha$, the set of possible computations starting with a given $\alpha \in \mathcal{C}$.

A node *p* is *enabled* in γ if at least one guard is true on *p* in γ.
A node is *neutralized* in the step $\gamma_i \mapsto \gamma_{i+1}$ if *p* is enabled in γ_i and not enabled in γ_{i+1}.

Snap-stabilization

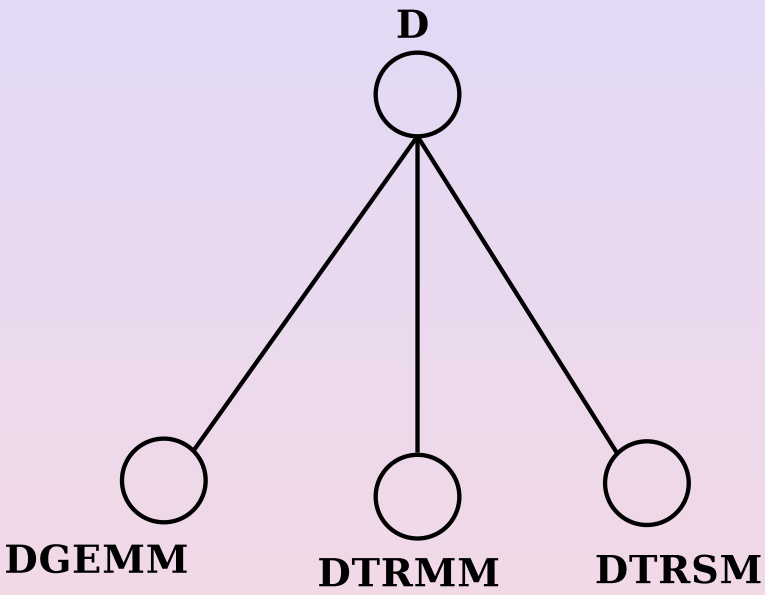
The first *round* e' of a computation $e \in \mathcal{E}$ is the minimal prefix of e containing the execution of one action or the neutralization of every enabled node from the first configuration.

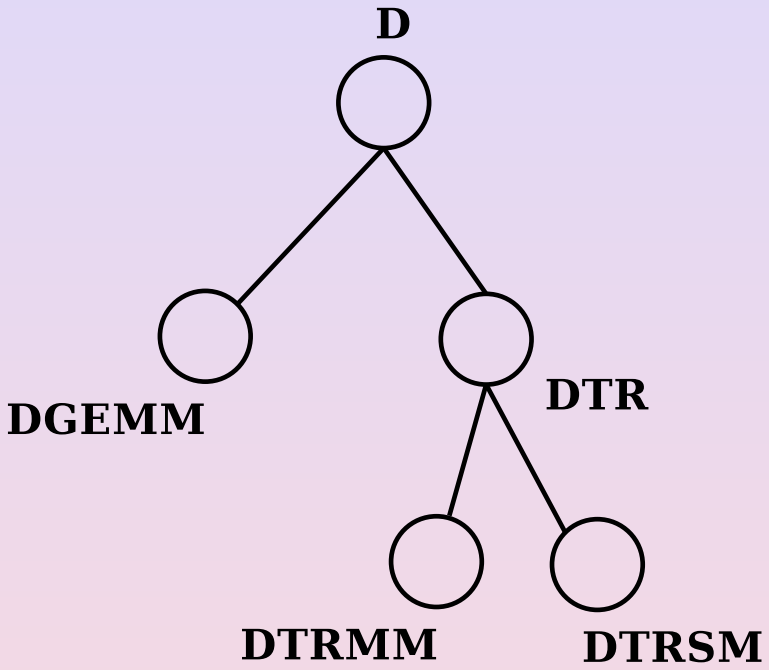
Let $\mathcal{X}$ be a set. We denote $x \vdash P$ the fact that $x \in \mathcal{X}$ satisfies the predicate P.

The protocol is **snap-stabilizing** for the specification $SP_{\mathcal{P}}$ on $\mathcal{E}$ iff :

$$\forall \alpha \in \mathcal{C} : \forall e \in \mathcal{E}_\alpha :: e \vdash SP_{\mathcal{P}}$$

The distributed structures maintained





Prefix Heap

A labeled rooted tree s.t. each node label is the **PGCP** of all its children labels.

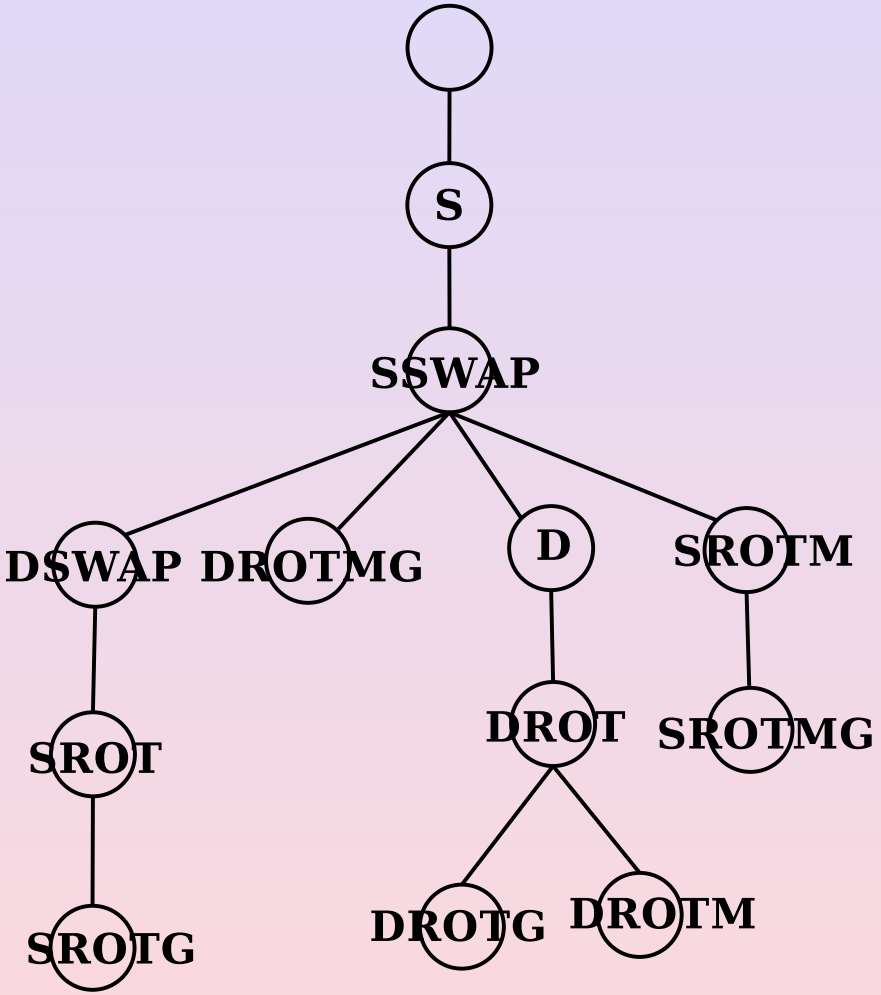
PGCP Tree

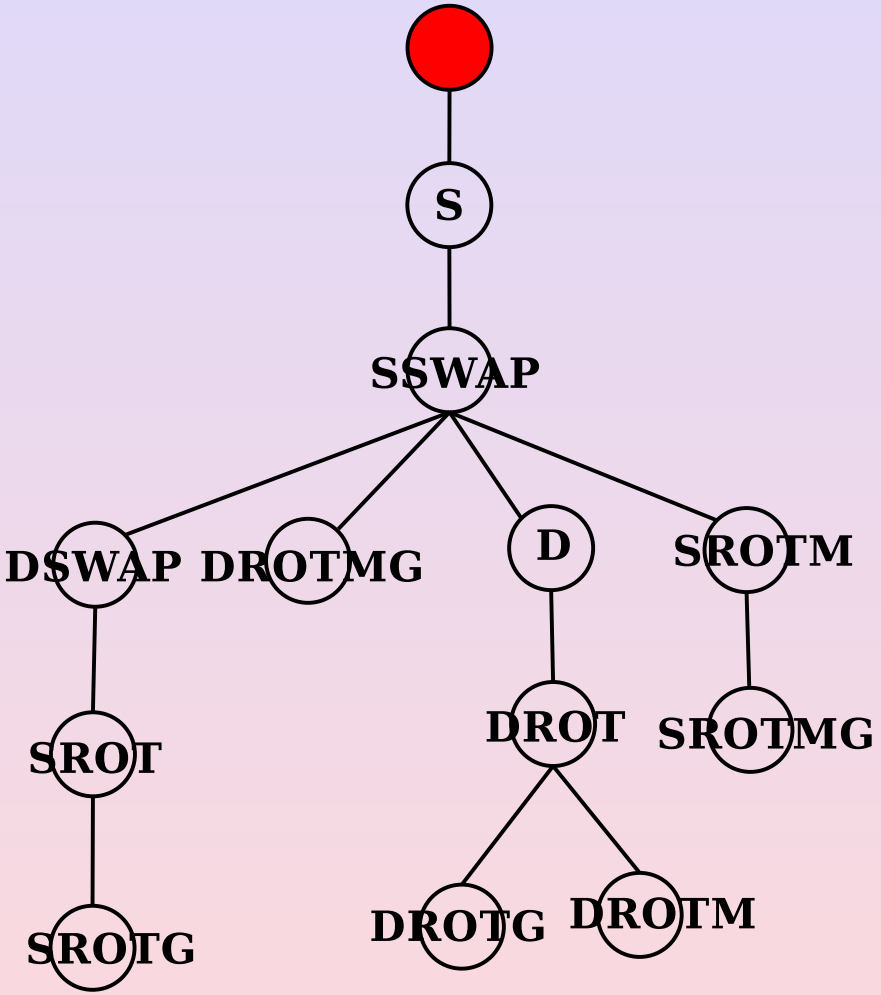
A labeled rooted tree s.t. each node label is the **PGCP** of any pair of its children labels.

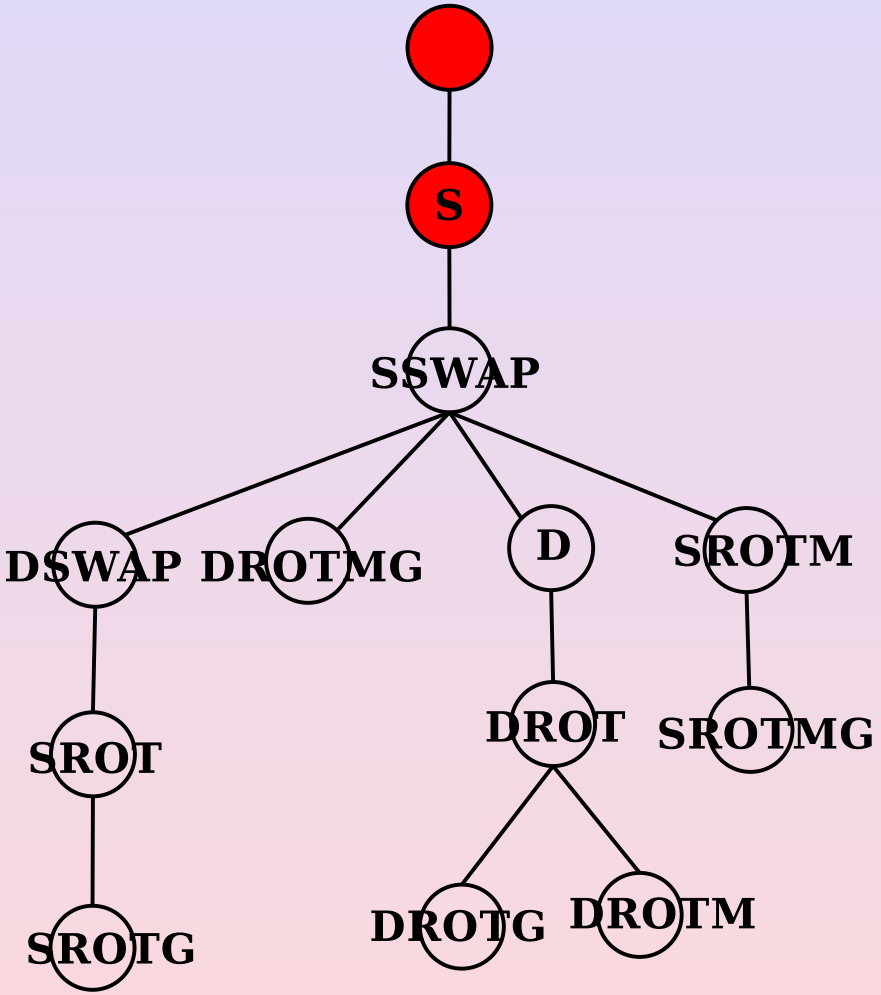
Outline

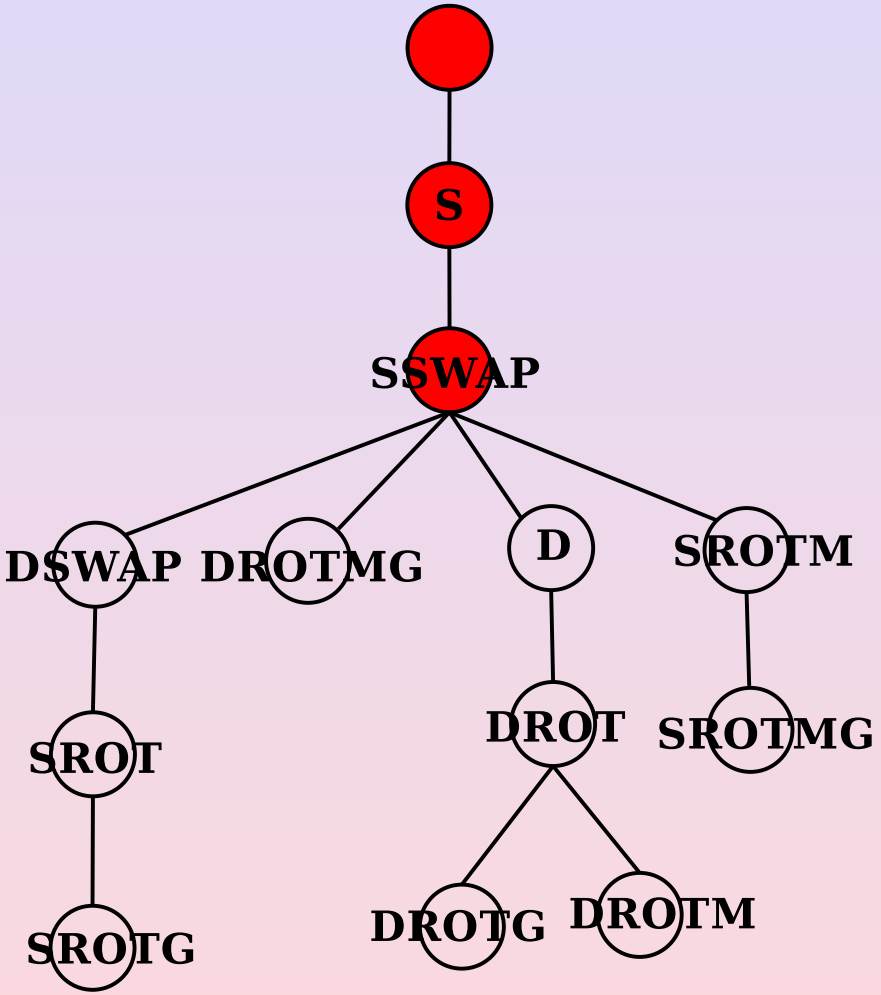
The algorithm

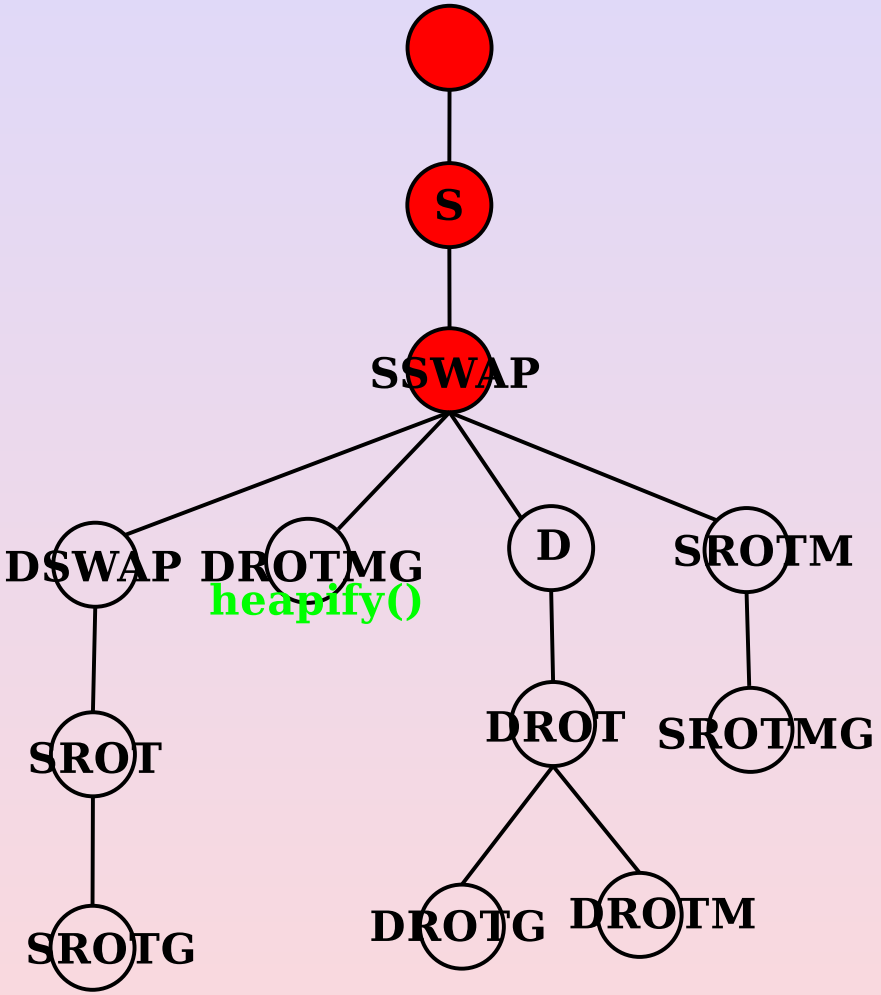
- Assumptions
  - The initial graph is a rooted connected tree
  - NEWNODE(*lbl*, *st*, *chldn*)
  - DESTROY(*p*)
  - HEAPIFY() locally creates a heap
  - REPAIR() locally builds a PGCP Tree starting from a heap
- Three Phases inspired by the snap-stabilizing PIF [Bui, Datta, Petit, Villain – 1999]
  - **Broadcast** phase : top-down wave initiating the algorithm
  - **Heapify** phase : down-top traversal building a prefix heap
  - **Repair** phase : final top-down wave building a PGCP tree
- Three node states
  - *I* (*initial*)
  - *B* (*broadcast*)
  - *H* (*heapified*)

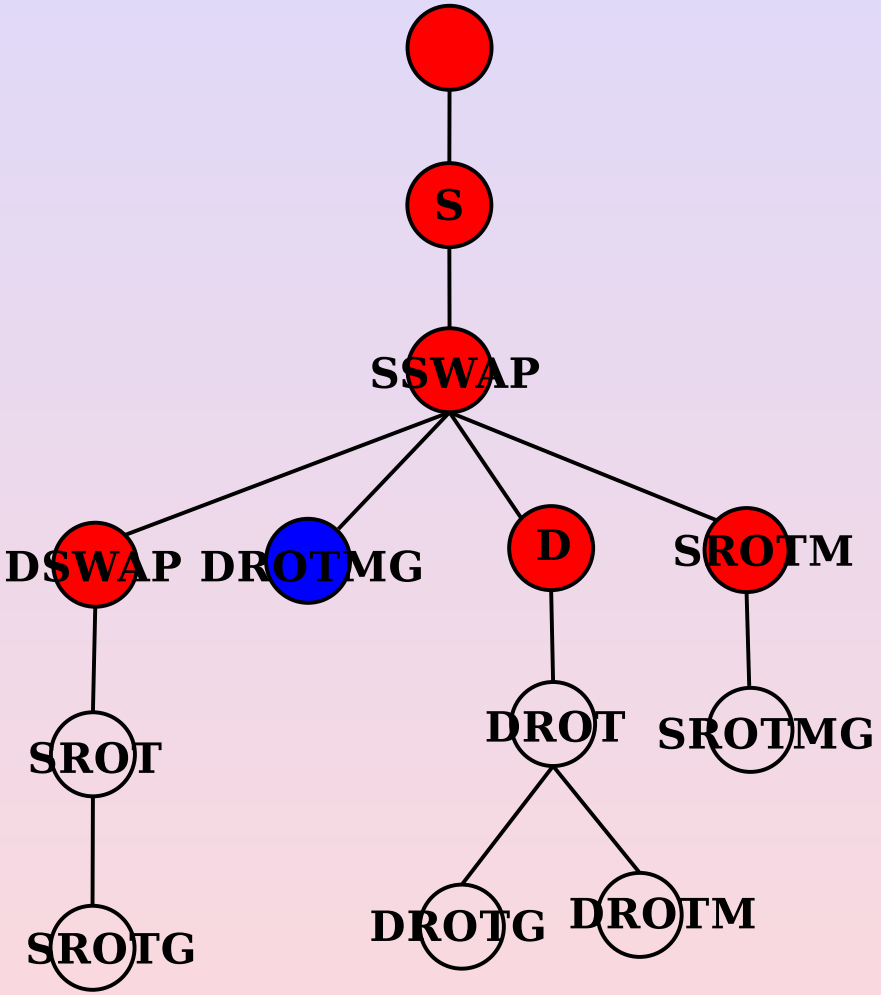

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

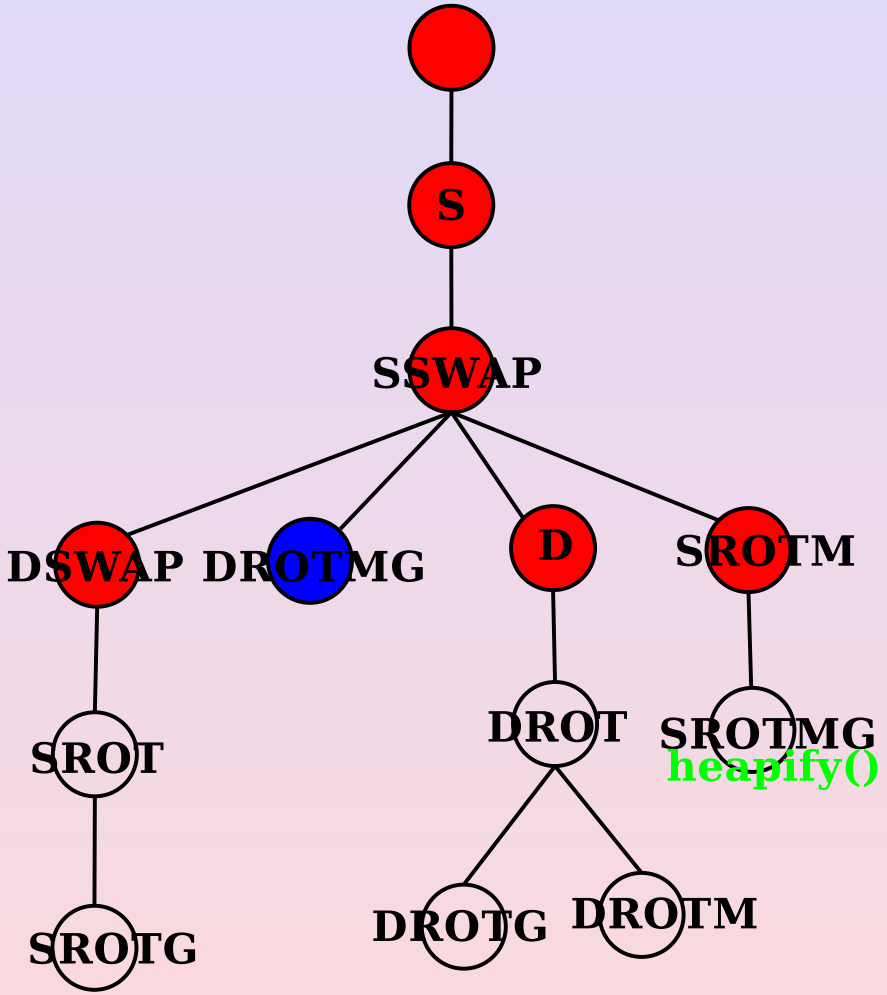

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

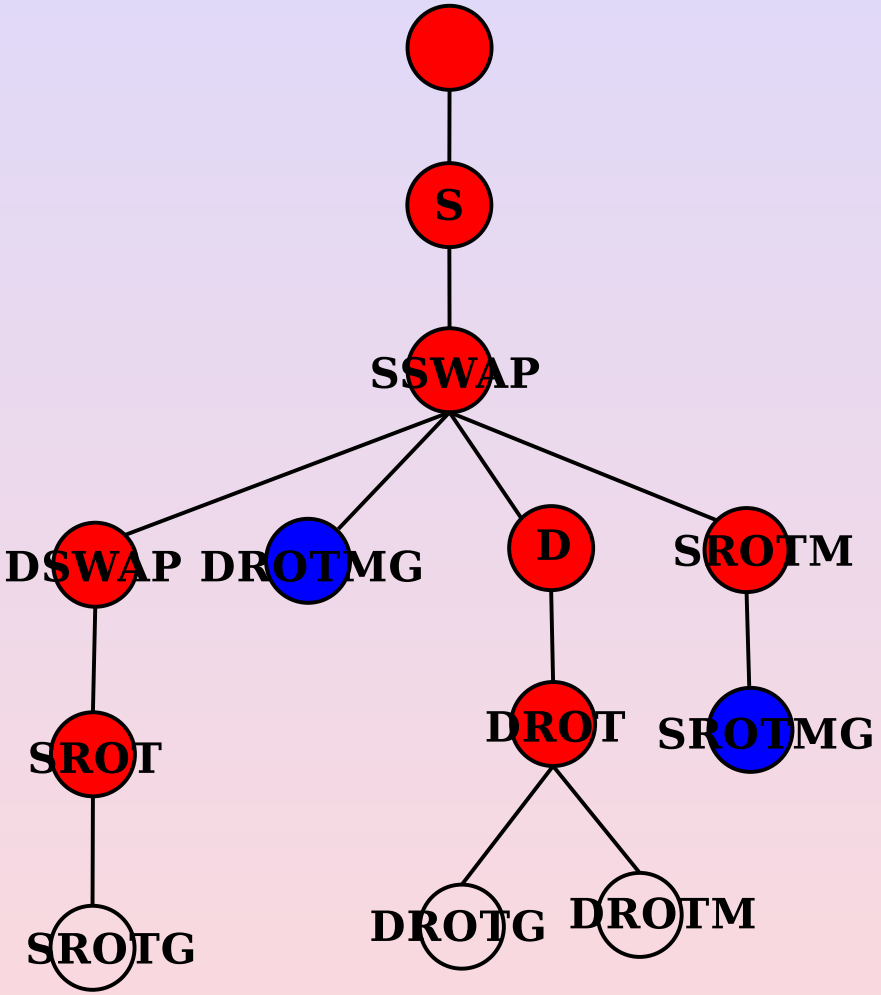

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

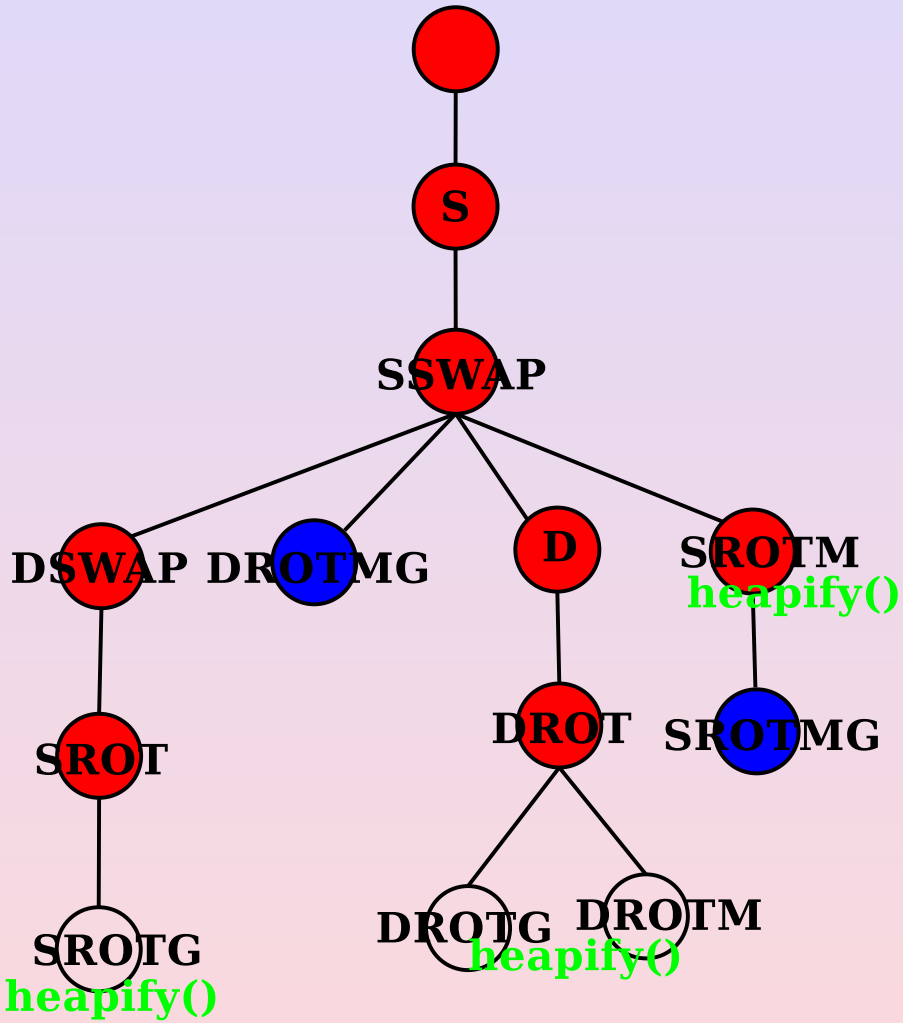

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

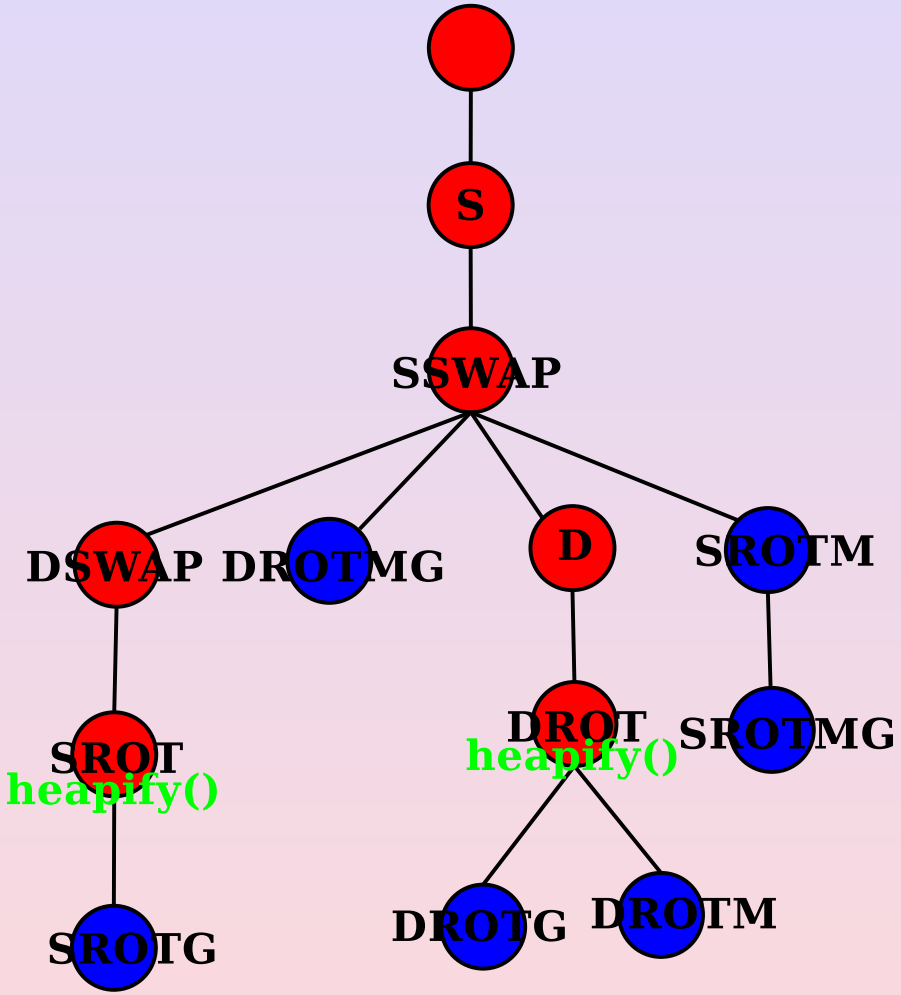

S
SSWAP
DSWAP
DROTMG
heapify()
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

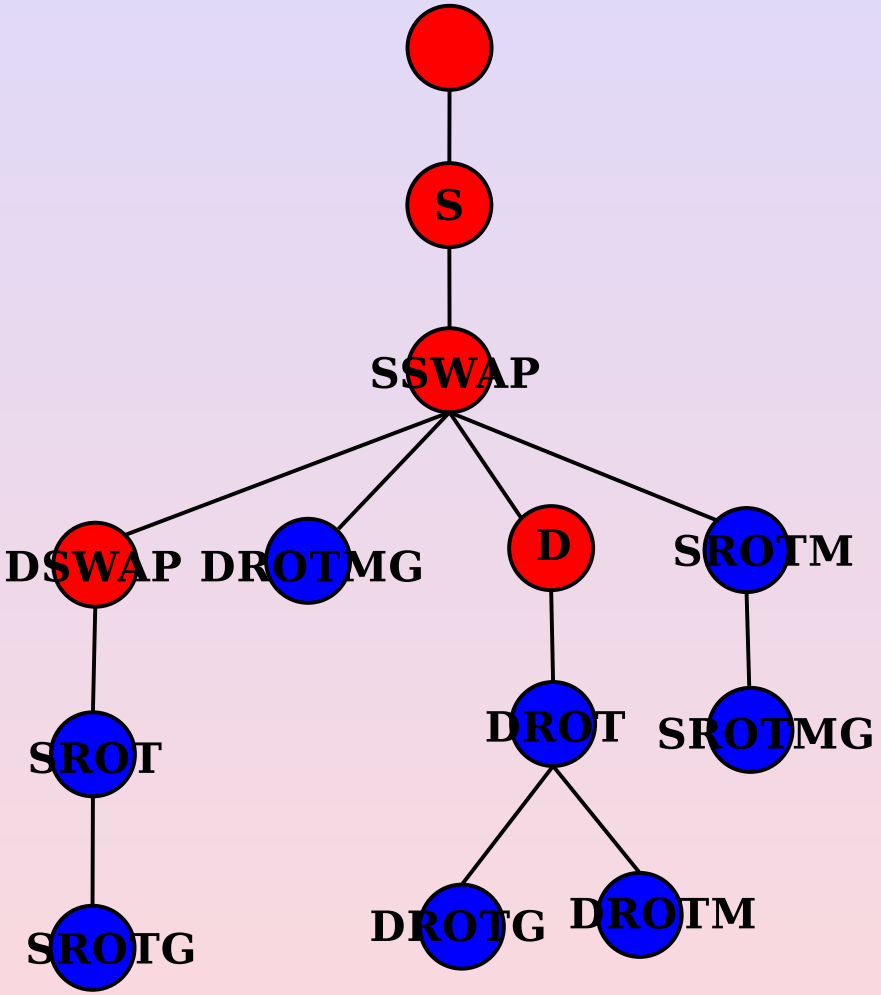

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

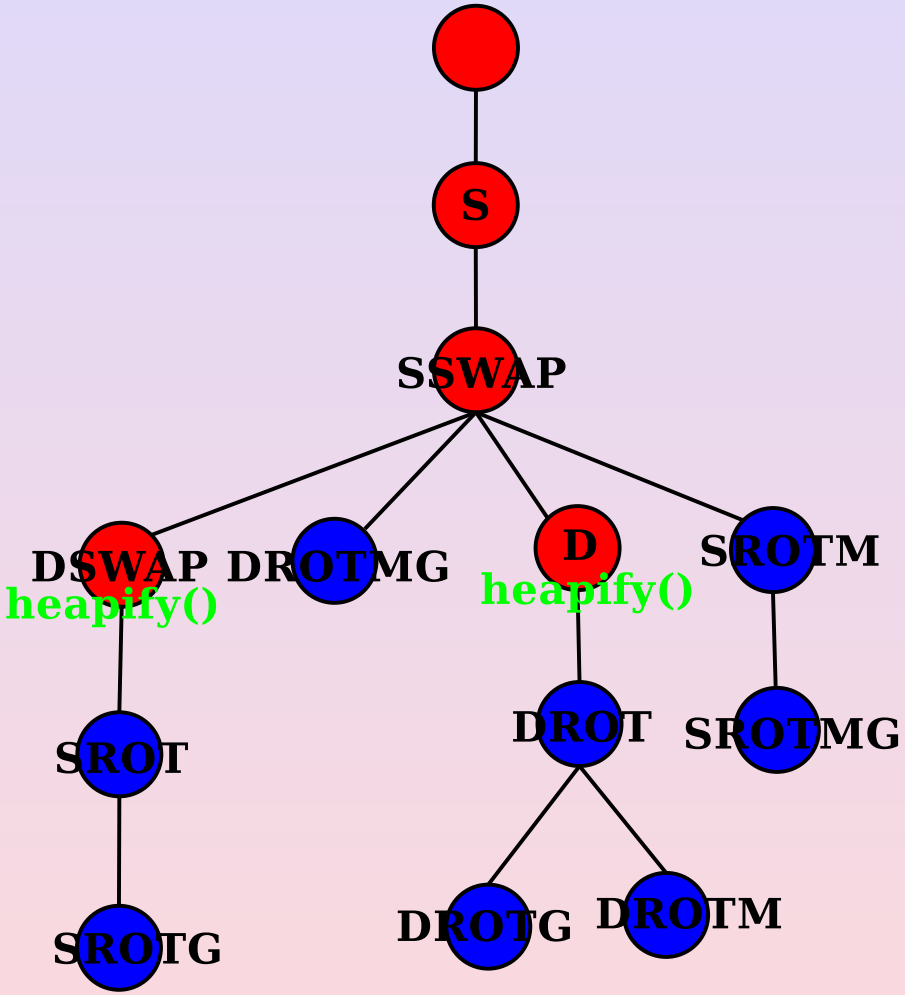

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
heapify()
SROTG
DROTG
DROTM

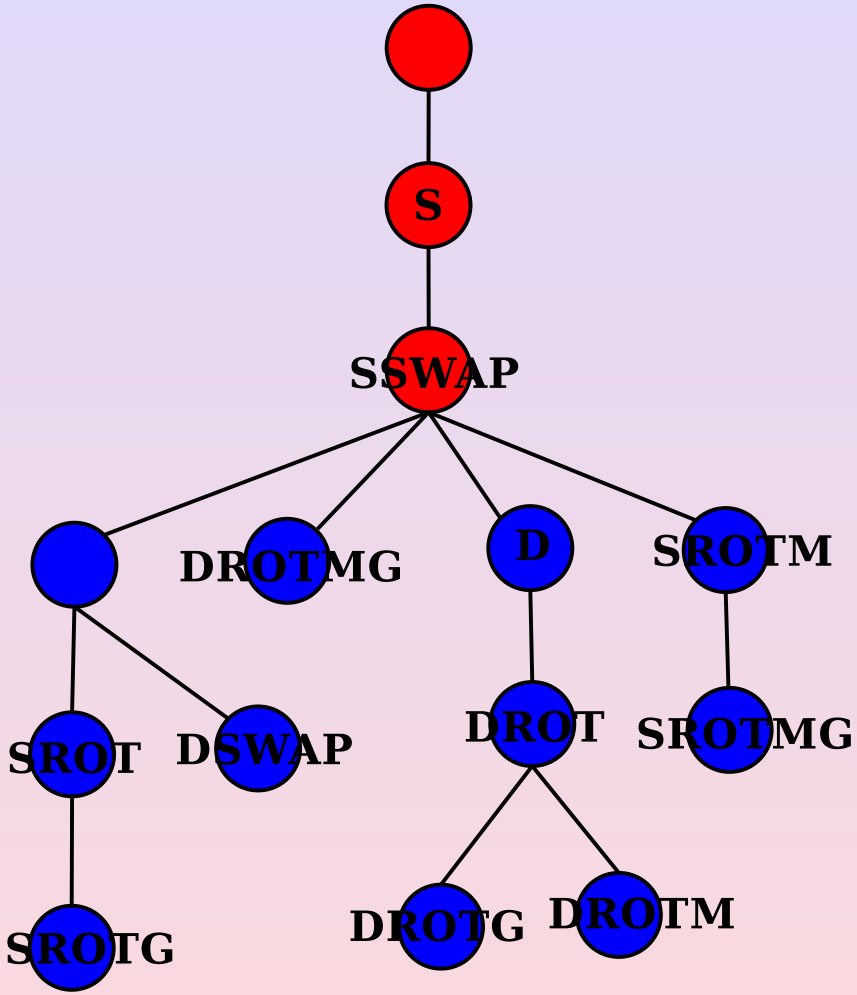

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

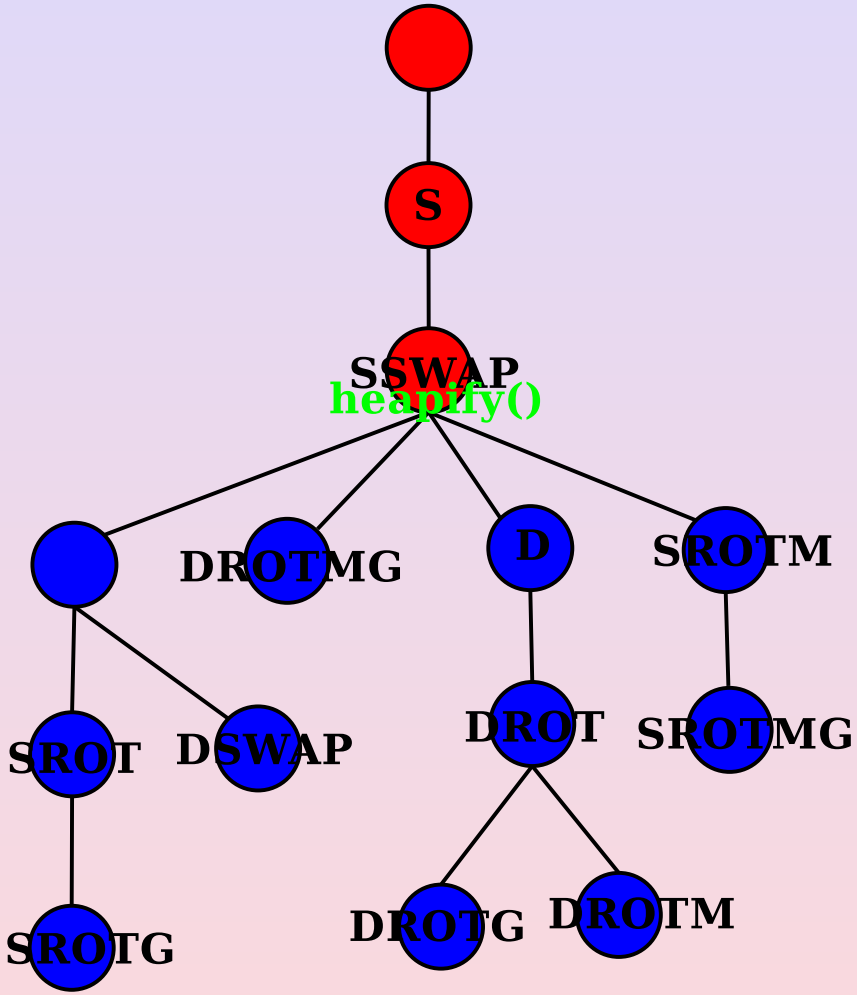

S
SSWAP
DSWAP
DROTMG
D
SROTM
heapify()
SROT
DROT
SROTMG
SROTG
heapify()
DROTG
DROTM
heapify()

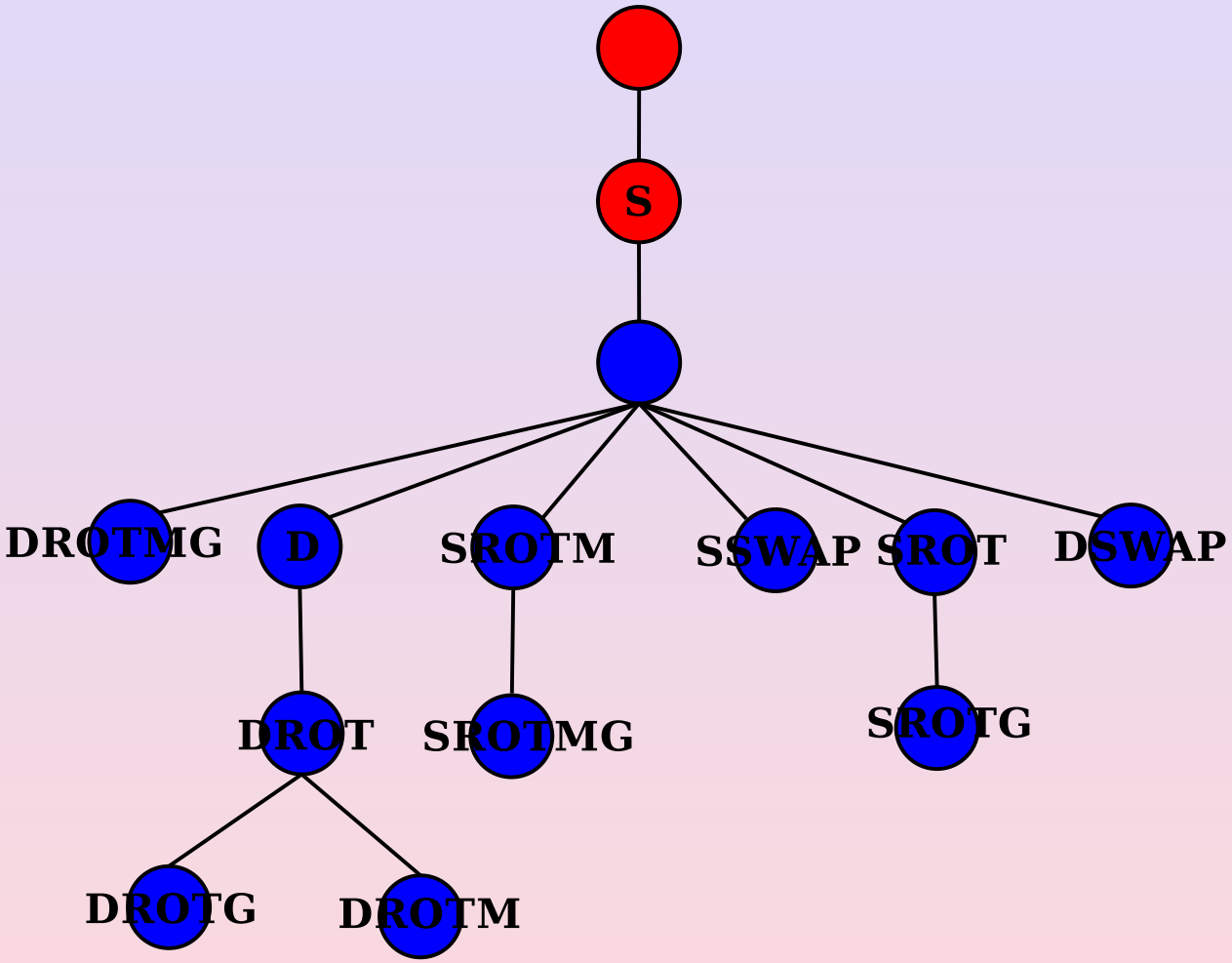

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
heapify()
DROT
heapify()
SROTMG
SROTG
DROTG
DROTM

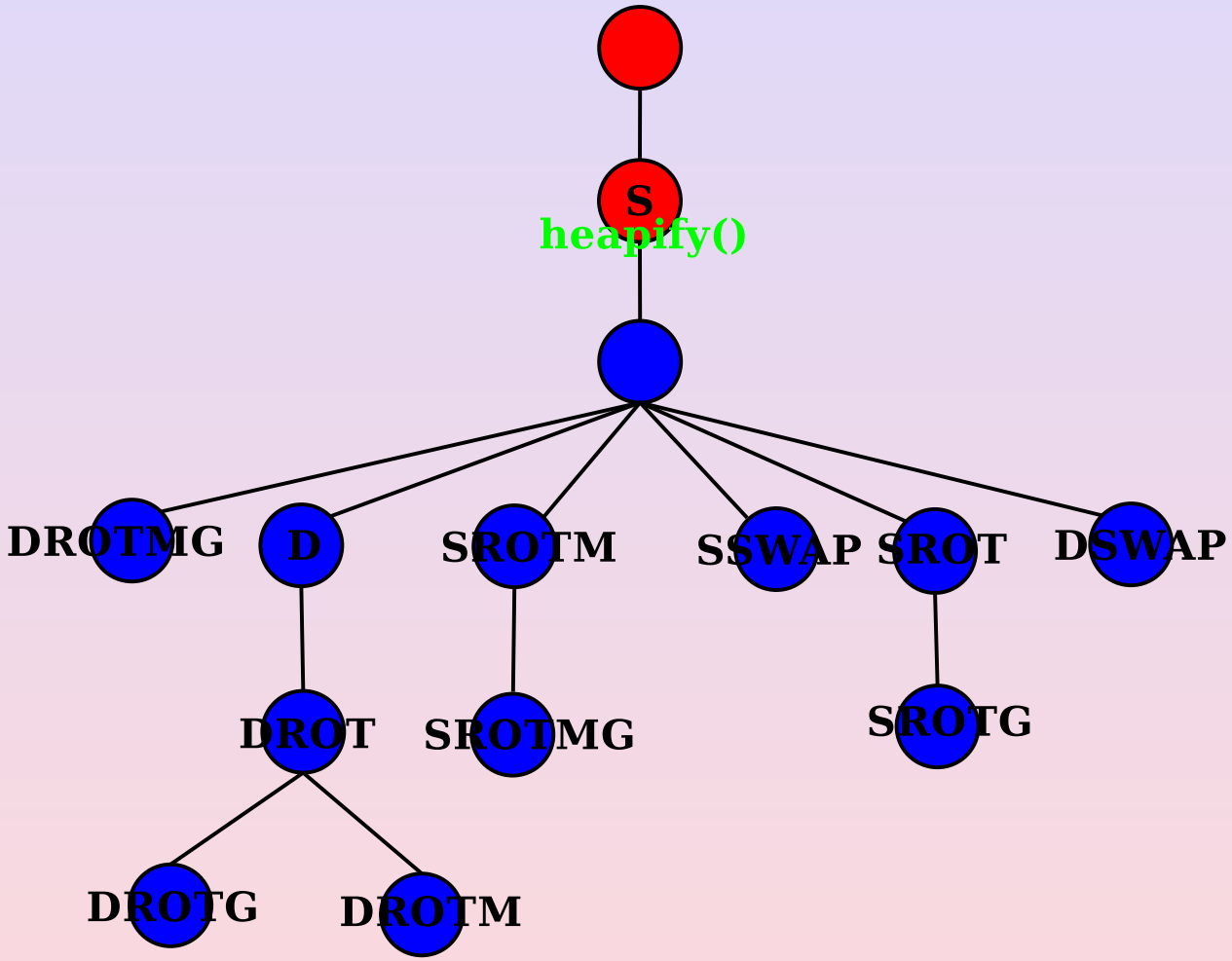

S
SSWAP
DSWAP
DROTMG
D
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

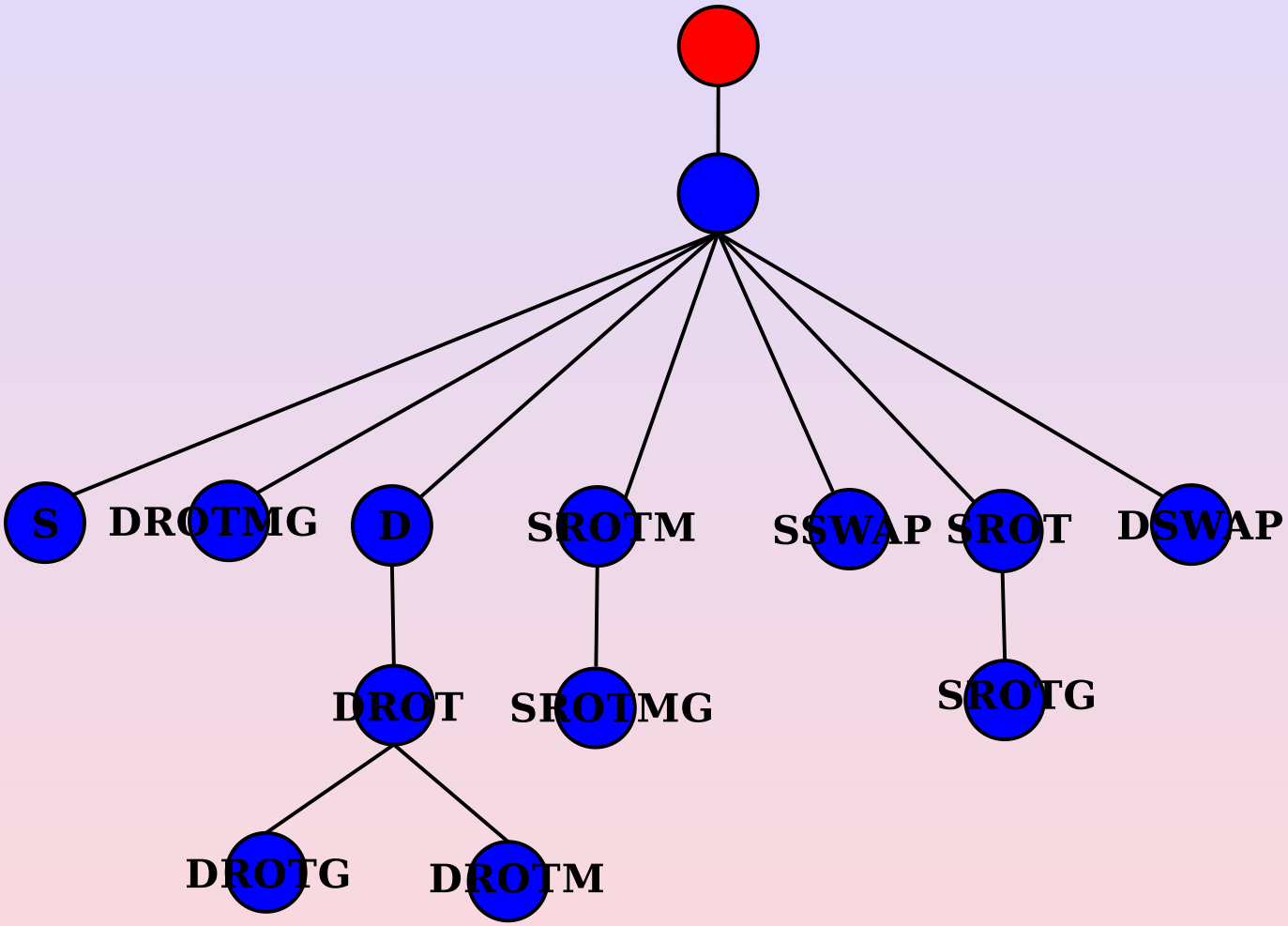

S
SSWAP
DSWAP
heapify()
DROTMG
D
heapify()
SROTM
SROT
DROT
SROTMG
SROTG
DROTG
DROTM

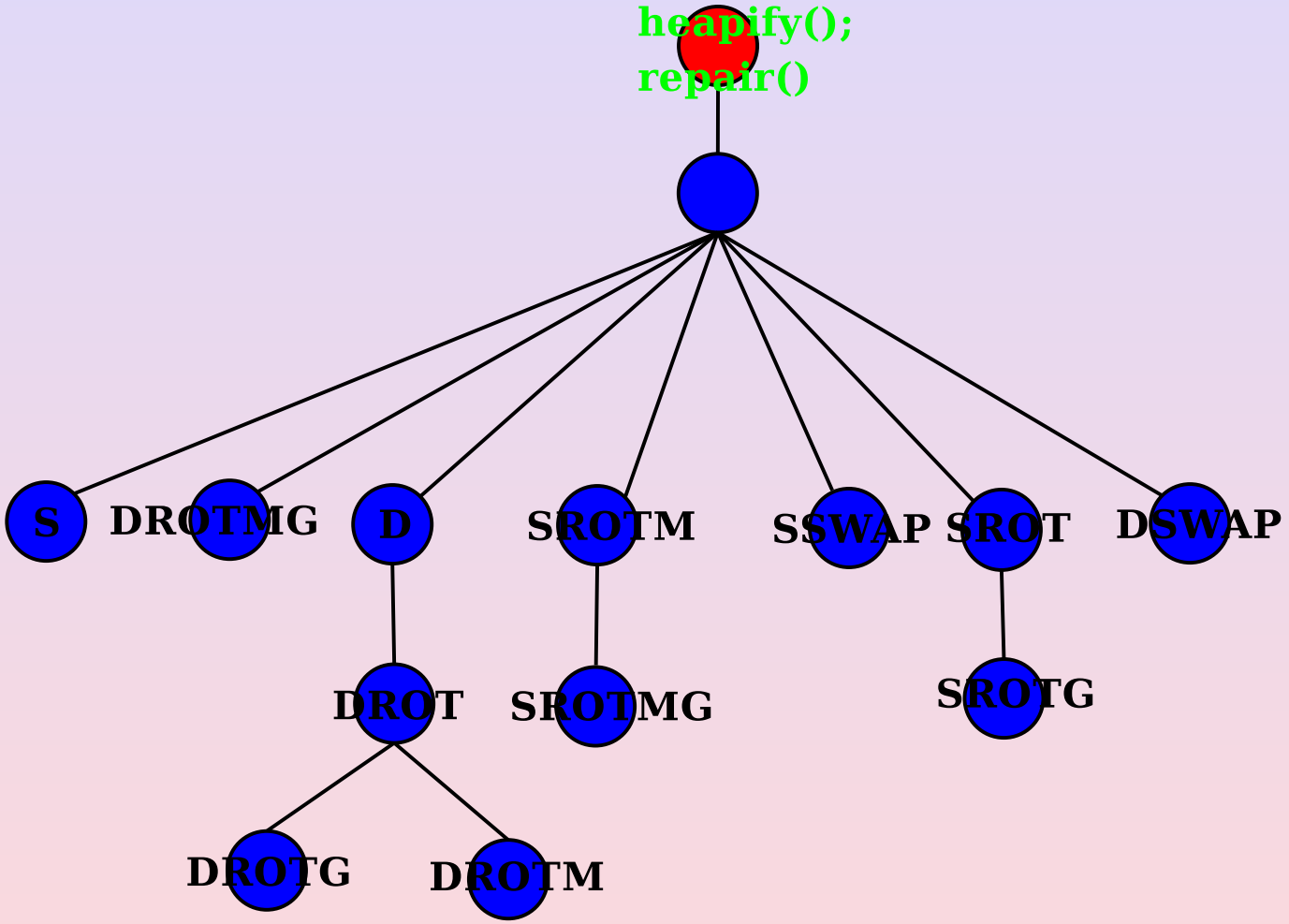

S
SSWAP
DROTMG
D
SROTM
SROT
DSWAP
DROT
SROTMG
SROTG
DROTG
DROTM

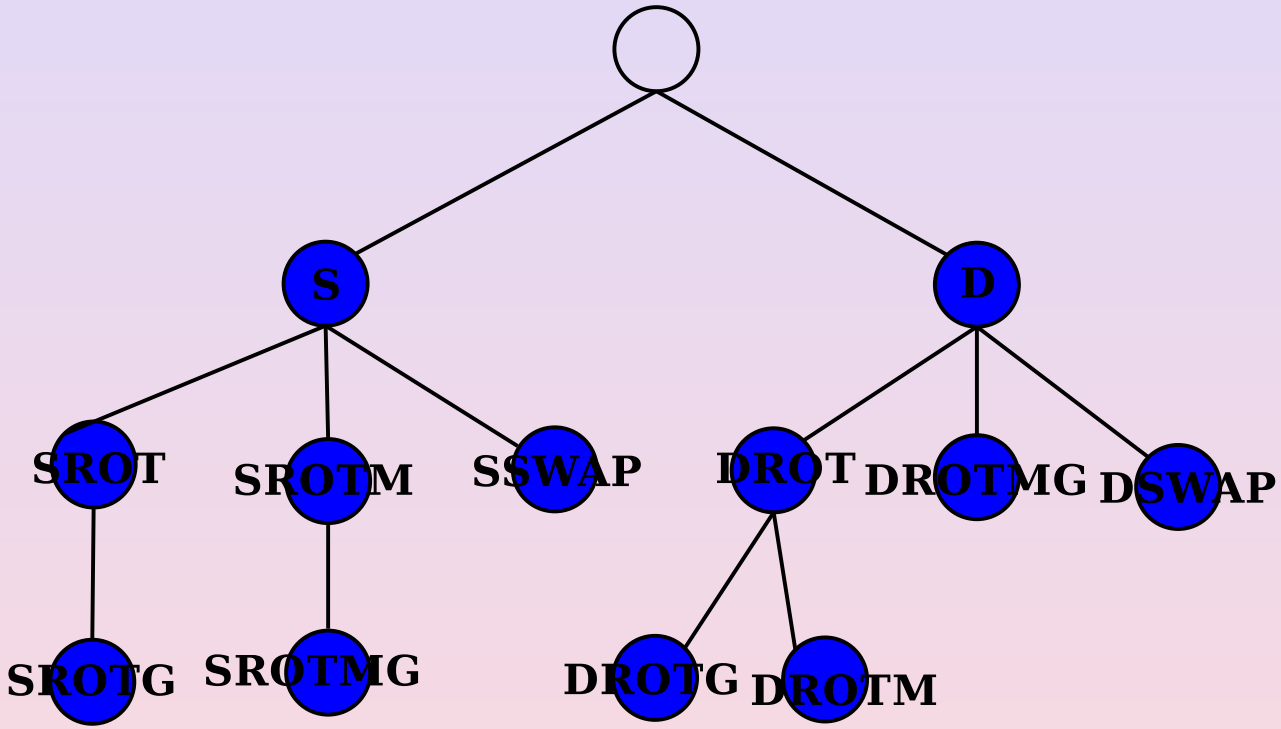

S
SSWAP
heapify()
DROTMG
D
SROTM
SROT
DSWAP
DROT
SROTMG
SROTG
DROTG
DROTM

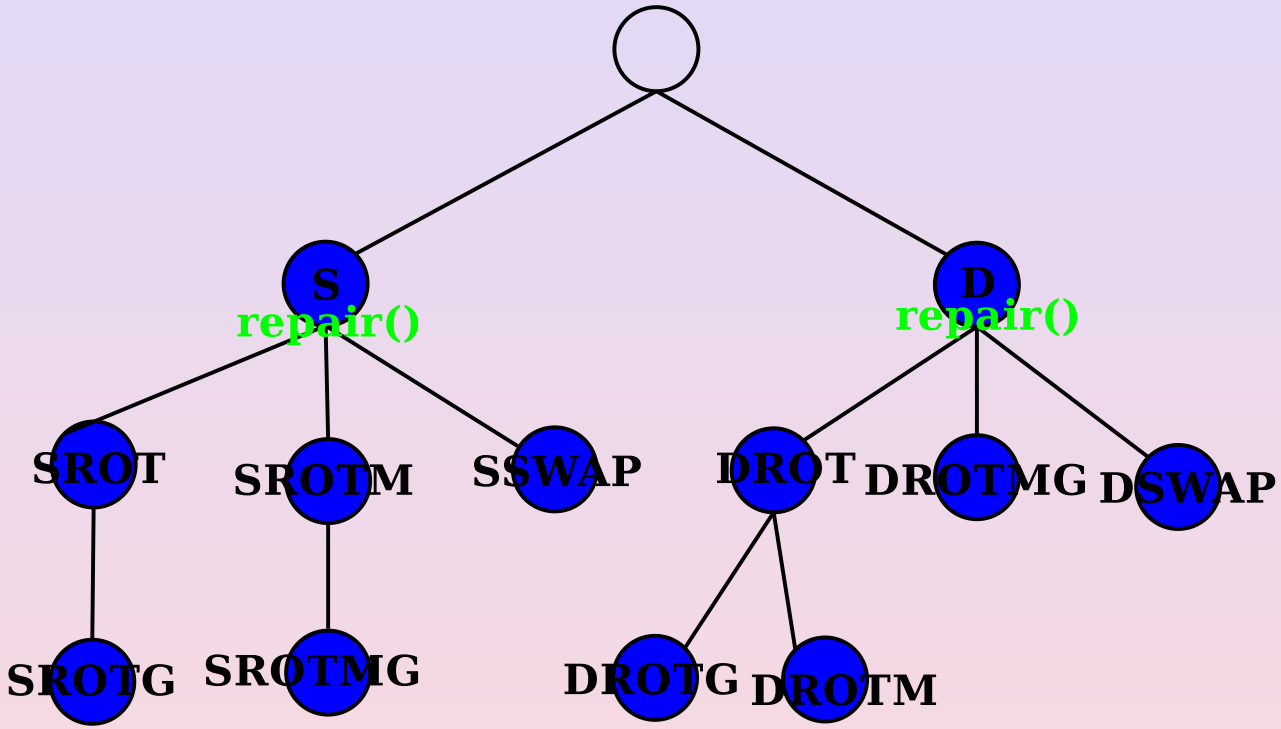

S
DROTMG
D
SROTM
SSWAP
SROT
DSWAP
DROT
SROTMG
SROTG
DROTG
DROTM

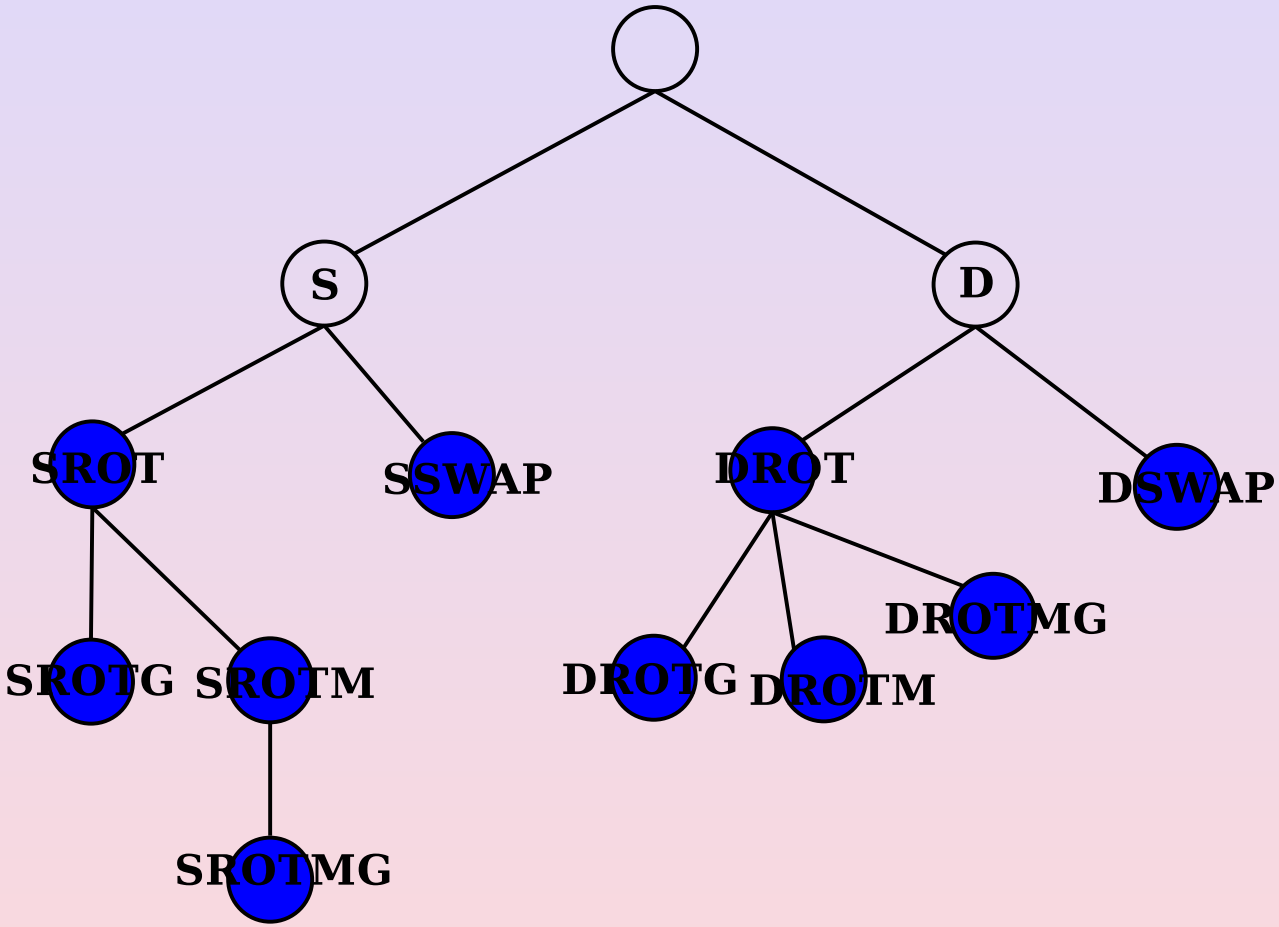

S
heapify()
DROTMG
D
SROTM
SSWAP
SROT
DSWAP
DROT
SROTMG
SROTG
DROTG
DROTM

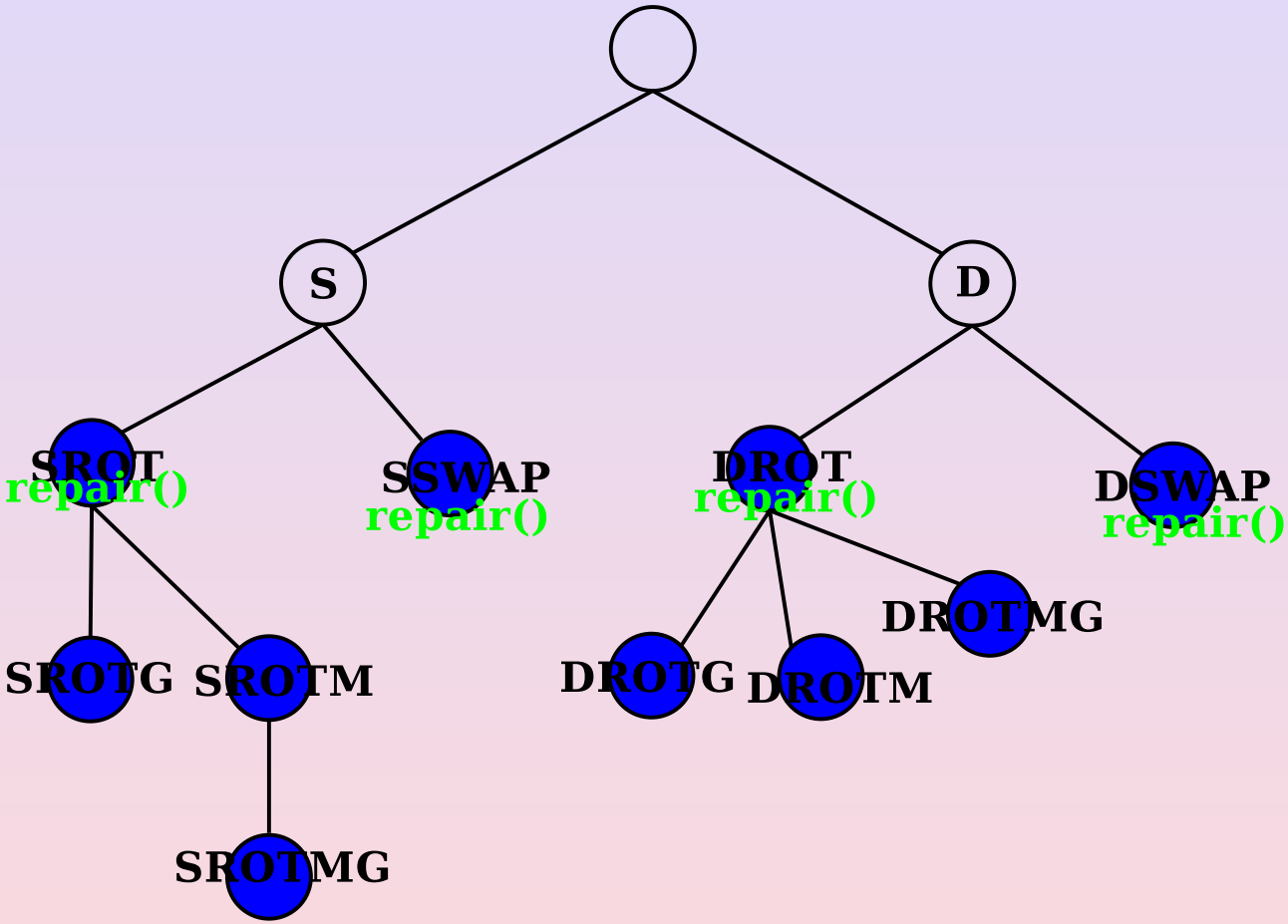

S
DROTMG
D
SROTM
SSWAP
SROT
DSWAP
DROT
SROTMG
SROTG
DROTG
DROTM

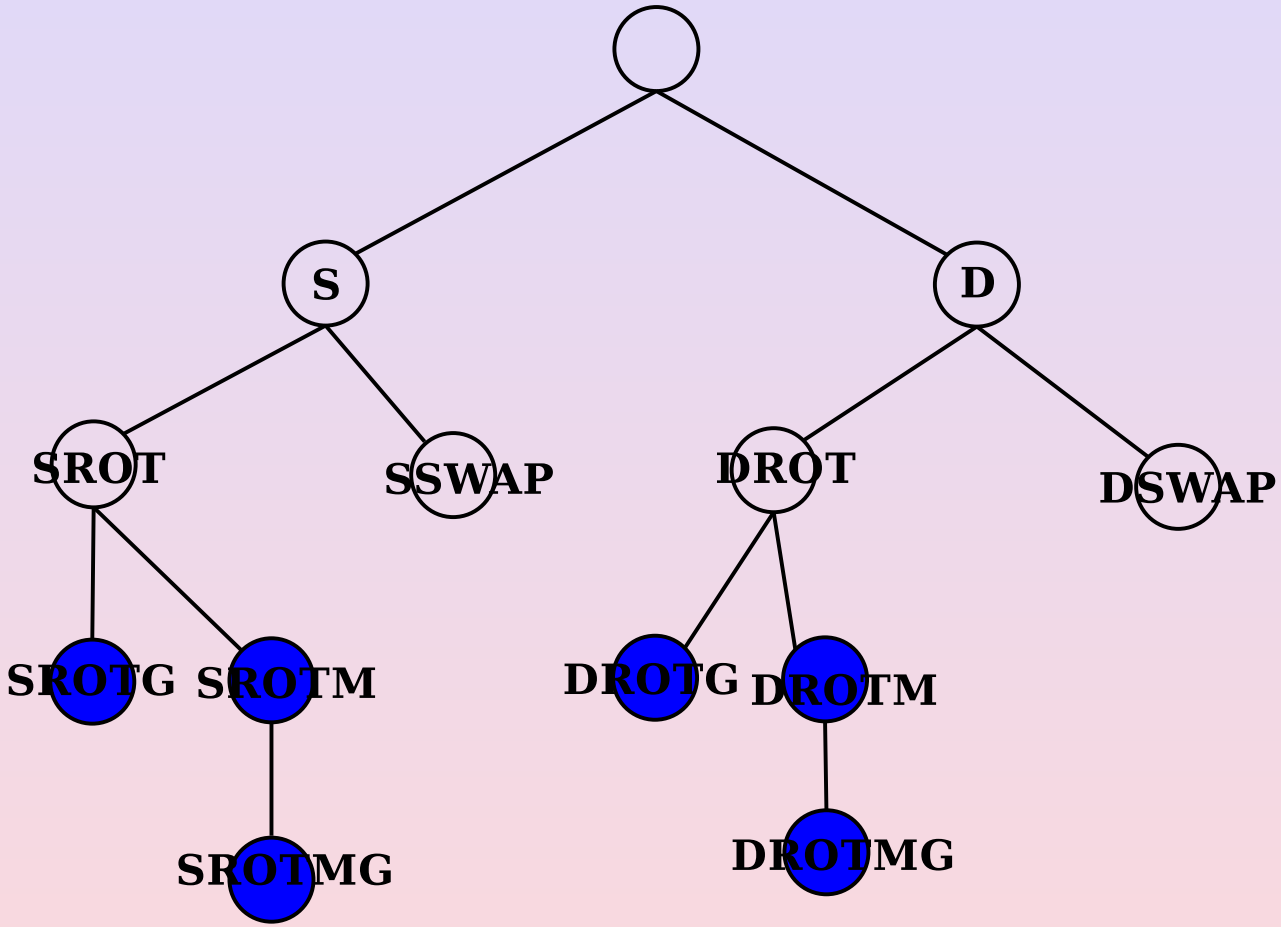

heapify();
repair()
S
DROTMG
D
SROTM
SSWAP
SROT
DSWAP
DROT
SROTMG
SROTG
DROTG
DROTM

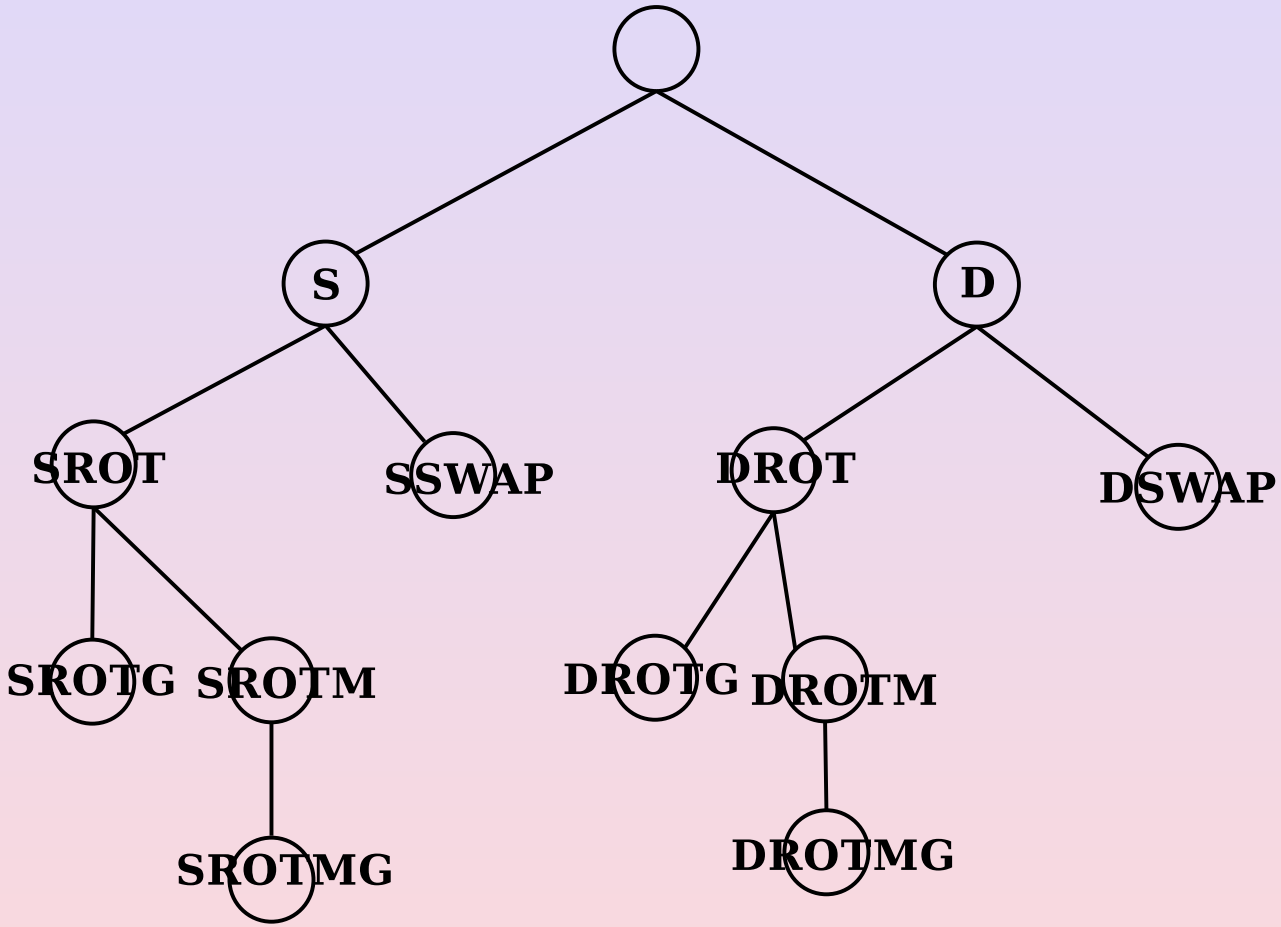

S
D
SROT
SROTM
SSWAP
DROT
DROTMG
DSWAP
SROTG
SROTMG
DROTG
DROTM


S
repair()
D
repair()
SROT
SROTM
SSWAP
DROT
DROTMG
DSWAP
SROTG
SROTMG
DROTG
DROTM


S
D
SROT
SSWAP
DROT
DSWAP
SROTG
SROTM
DROTG
DROTM
DROTMG
SROTMG


S
D
SROT
repair()
SSWAP
repair()
DROT
repair()
DSWAP
repair()
SROTG
SROTM
SROTMG
DROTG
DROTM
DROTMG


S
D
SROT
SSWAP
DROT
DSWAP
SROTG
SROTM
DROTG
DROTM
SROTMG
DROTMG


S
D
SROT
SSWAP
DROT
DSWAP
SROTG
SROTM
DROTG
DROTM
SROTMG
DROTMG

Sketch of proof

Lemma (1)

Starting from any configuration, the root node initiates the wave in a finite time.

Lemma (2)

Afther the execution of procedure HEAPIFY *by p, T_p is a prefix heap.*

Corollary (1)

After the Heapify phase ends, the whole tree is a prefix heap.

Sketch of proof

Lemma (1)

Starting from any configuration, the root node initiates the wave in a finite time.

Lemma (2)

Afther the execution of procedure HEAPIFY *by p, T_p is a prefix heap.*

Corollary (1)

After the Heapify phase ends, the whole tree is a prefix heap.

Sketch of proof

Lemma (1)

Starting from any configuration, the root node initiates the wave in a finite time.

Lemma (2)

Afther the execution of procedure HEAPIFY *by p, T_p is a prefix heap.*

Corollary (1)

After the Heapify phase ends, the whole tree is a prefix heap.

Lemma (3)

After the execution of procedure REPAIR *by p, the label of p is the PCP of any pair of children labels of p.*

Corollary (2)

After the Repair phase ends, the whole tree is a PGCP tree.

Theorem

The proposed algorithms provide a snap-stabilizing protocol building a PGCP Tree construction.

Lemma (3)

After the execution of procedure REPAIR *by p, the label of p is the PCP of any pair of children labels of p.*

Corollary (2)

After the Repair phase ends, the whole tree is a PGCP tree.

Theorem

The proposed algorithms provide a snap-stabilizing protocol building a PGCP Tree construction.

Lemma (3)

After the execution of procedure REPAIR *by p, the label of p is the PCP of any pair of children labels of p.*

Corollary (2)

After the Repair phase ends, the whole tree is a PGCP tree.

Theorem

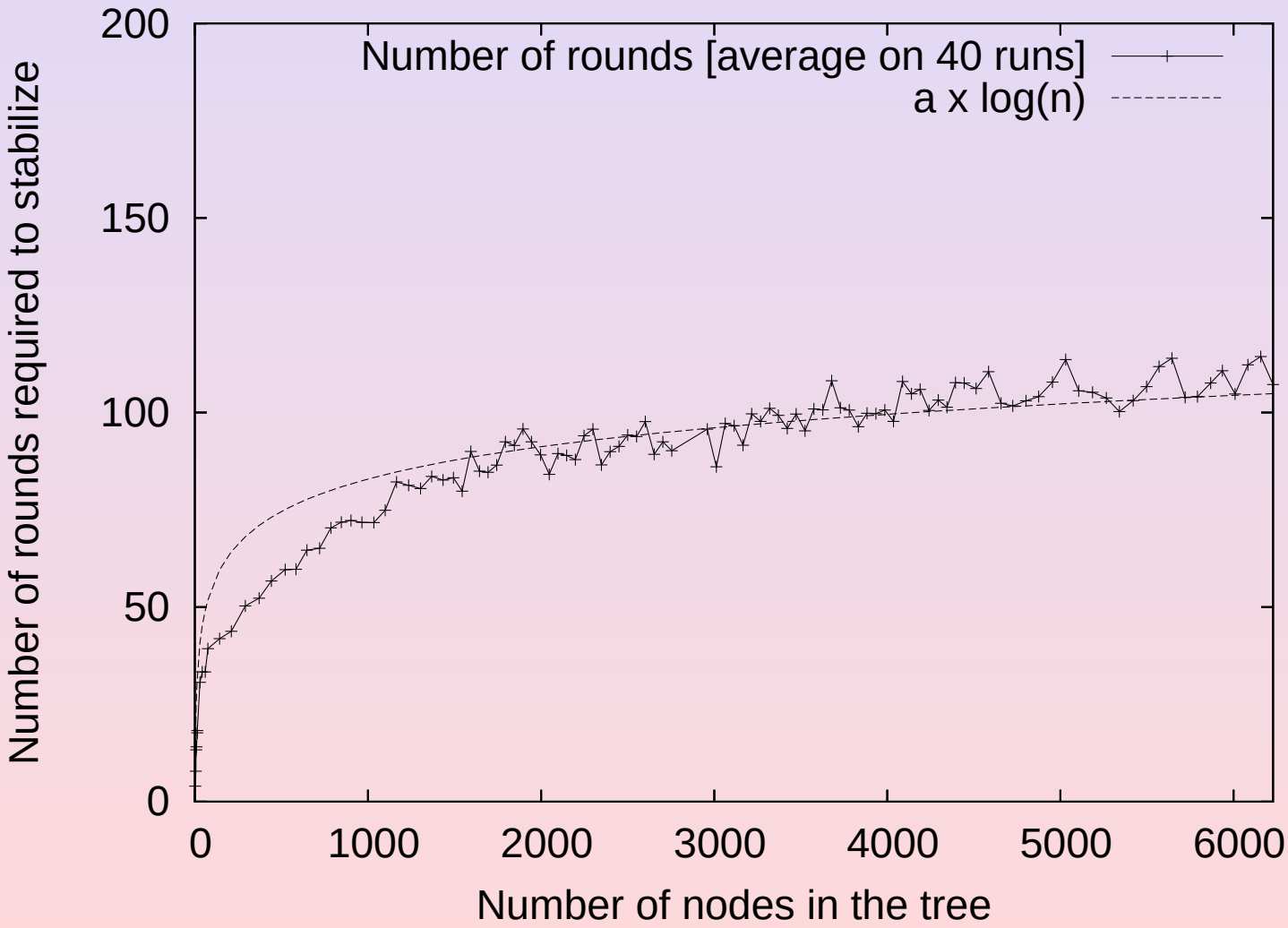
The proposed algorithms provide a snap-stabilizing protocol building a PGCP Tree construction.

Complexity

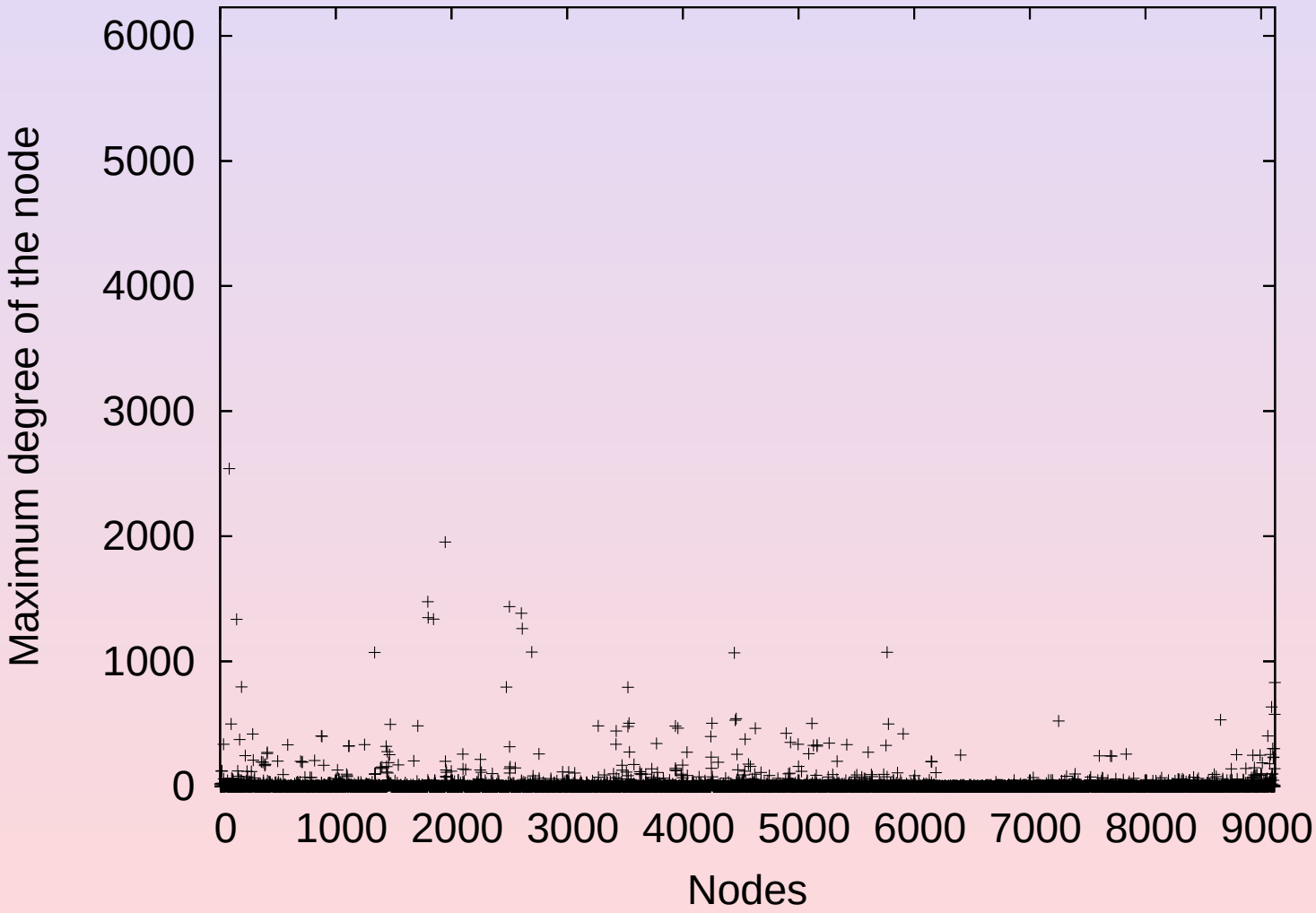
- Time to obtain a consistant PGCP Tree : $O(h + h')$ rounds
- Worst case : $O(n)$ rounds, $O(n^2)$ operations, $O(n)$ extra space

Outline

Number of discretized rounds



Average extra space required



Outline

Conclusion

- Contributions
  - The first snap-stabilizing prefix tree
  - An alternative to nodes replication
  - Proven to be snap-stabilizing in the state model and thus optimal in terms of stabilization time
  - Average latency in the height of the tree
- On-going and future work
  - Algorithm desgined in the state model, need to be designed for the MP model
  - Implementation and deployment over a real platform