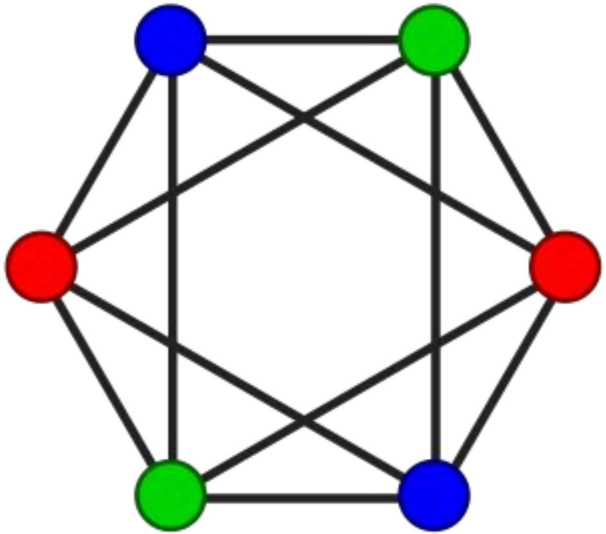
Ordering Heuristics for Parallel Graph Coloring

Authors: William Hasenplaugh, Tim Kaler, Tao B. Schardl, Charles E. Leiserson

Presentation by Ethan LaBelle

Definition: Graph-Coloring

- Definition: Vertex Coloring
  - Assignment of a color to each vertex of an undirected graph G = (V, E), such that for every edge (u, v) in E, u.color != v.color
- Find optimal vertex coloring (fewest colors)
- NP-complete problem
- In practice, approximation algorithms are sufficient



Motivation

- Scheduling data graph computations
  - Sequence of update on vertices of a graph
  - New value of a vertex depends on value of vertex and adjacent vertex values
  - Vertices of same color can be update in parallel
  - Fewer colors ⇔ more parallelism
- Other real world applications:
  - Register allocation via Graph Coloring

Properties of Good Parallel Ordering

- Quality ordering
- Scalable
- Work Efficient

Greedy Algorithm

GREEDY(G)

let $G = (V, E, \rho)$
for $v \in V$ in order of decreasing $\rho(v)$
    $C = \{1, 2, \ldots, \deg(v) + 1\}$
    for $u \in v.adj$ such that $\rho(u) > \rho(v)$
        $C = C - \{u.color\}$
    $v.color = \min C$

ρ- priority function

What is the required work?
Is this procedure parallelizable?

- Colors a graph with degree Δ in at most Δ + 1 colors

Greedy Algorithm

GREEDY(G)

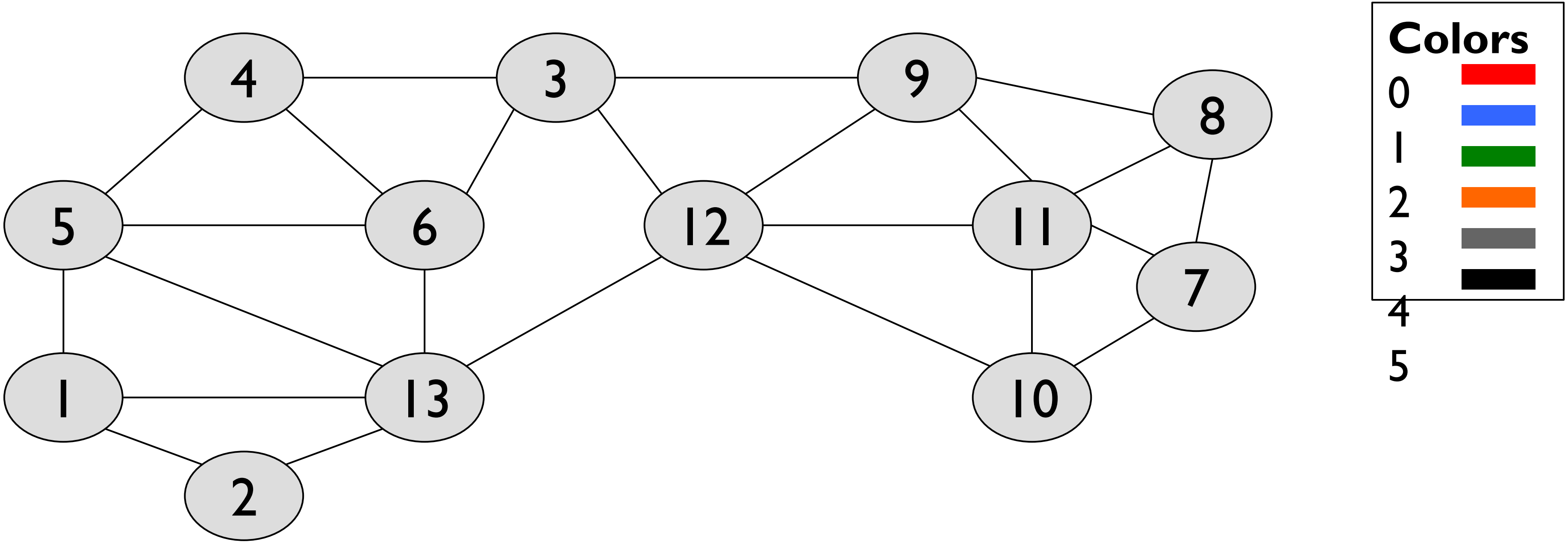
```
let G = (V, E, ρ)
for v ∈ V in order of decreasing ρ(v)
    C = {1, 2, ..., deg(v) + 1}
    for u ∈ v.adj such that ρ(u) > ρ(v)
        C = C − {u.color}
    v.color = min C
```

ρ- priority function

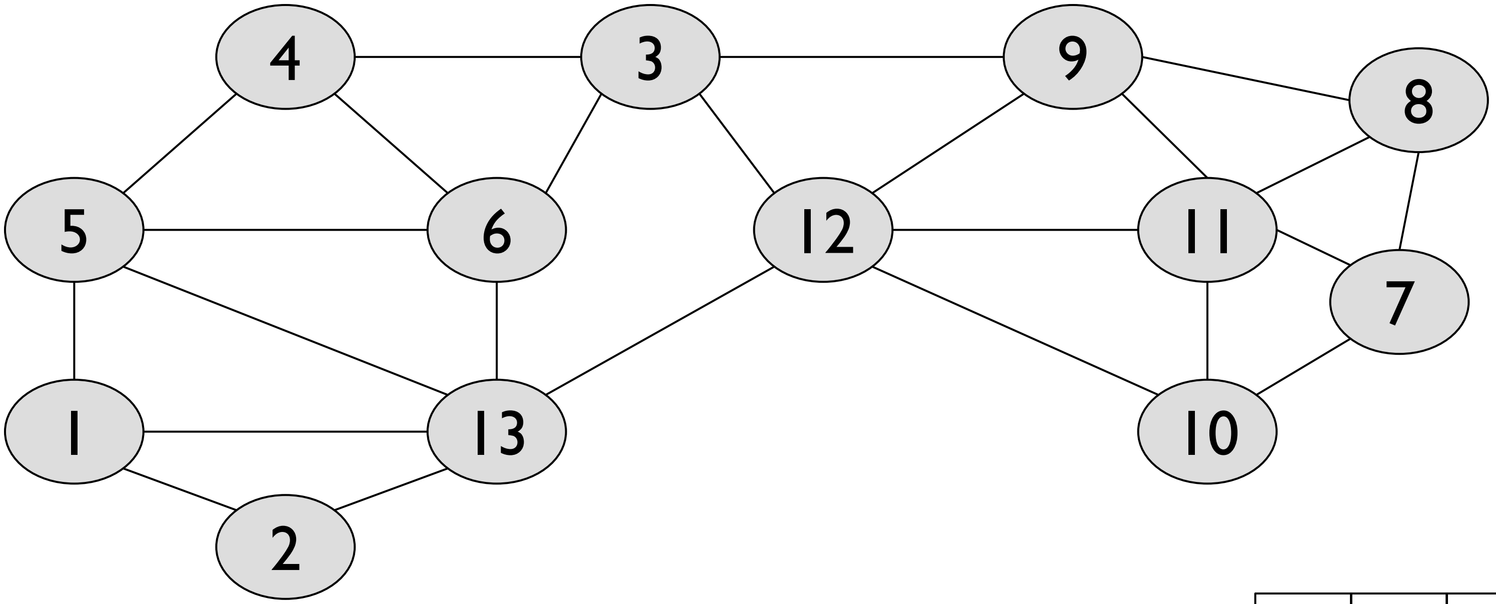
What is the required work?
Is this procedure parallelizable?

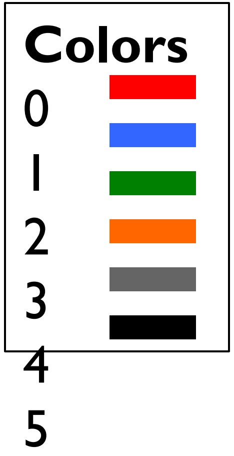
- Colors a graph with degree Δ in at most Δ + 1 colors

Example: Greedy Coloring



Example: Greedy Coloring

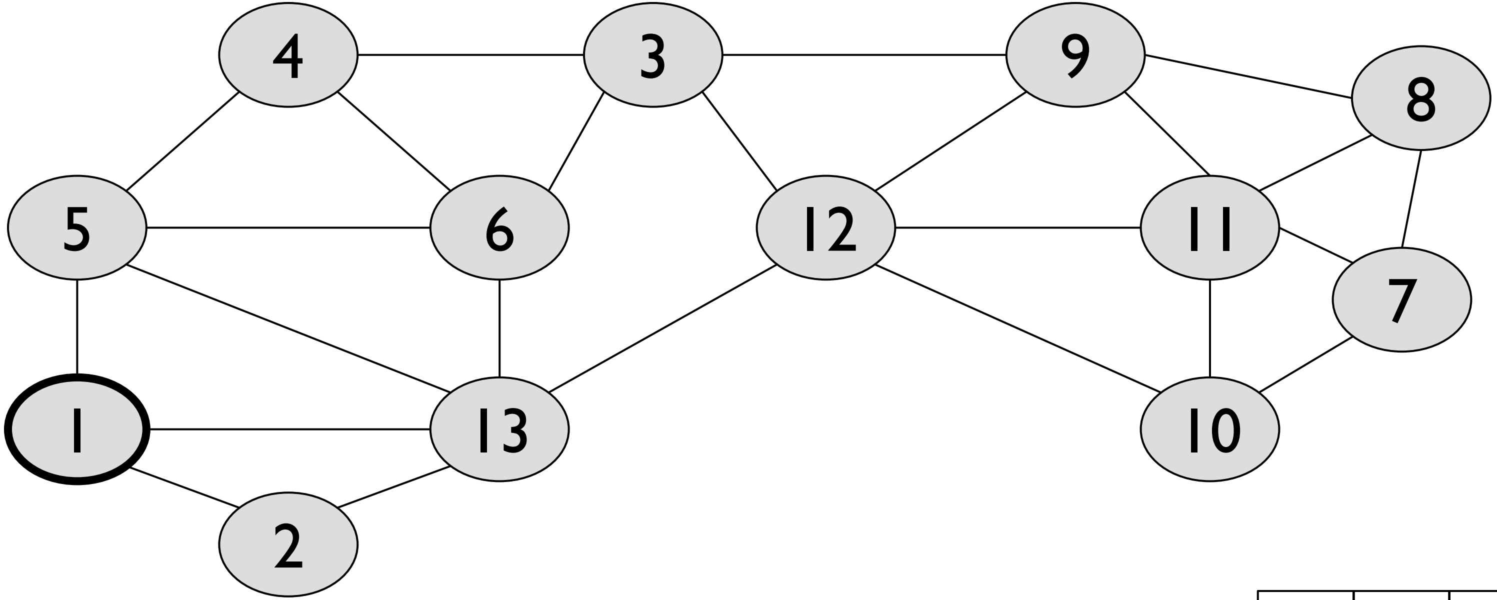


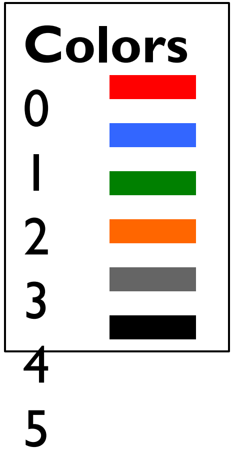


5	4	3	2	1	0

Neighboring Colors

Example: Greedy Coloring

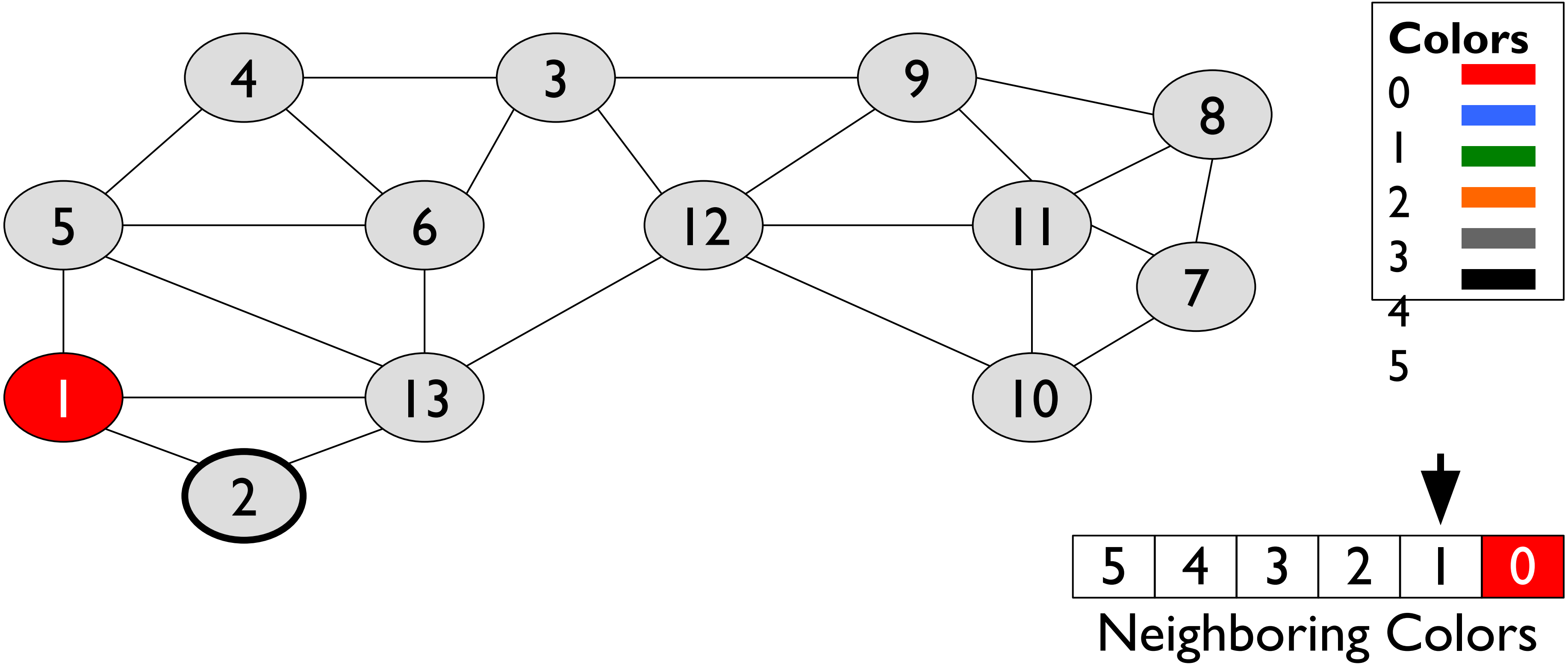




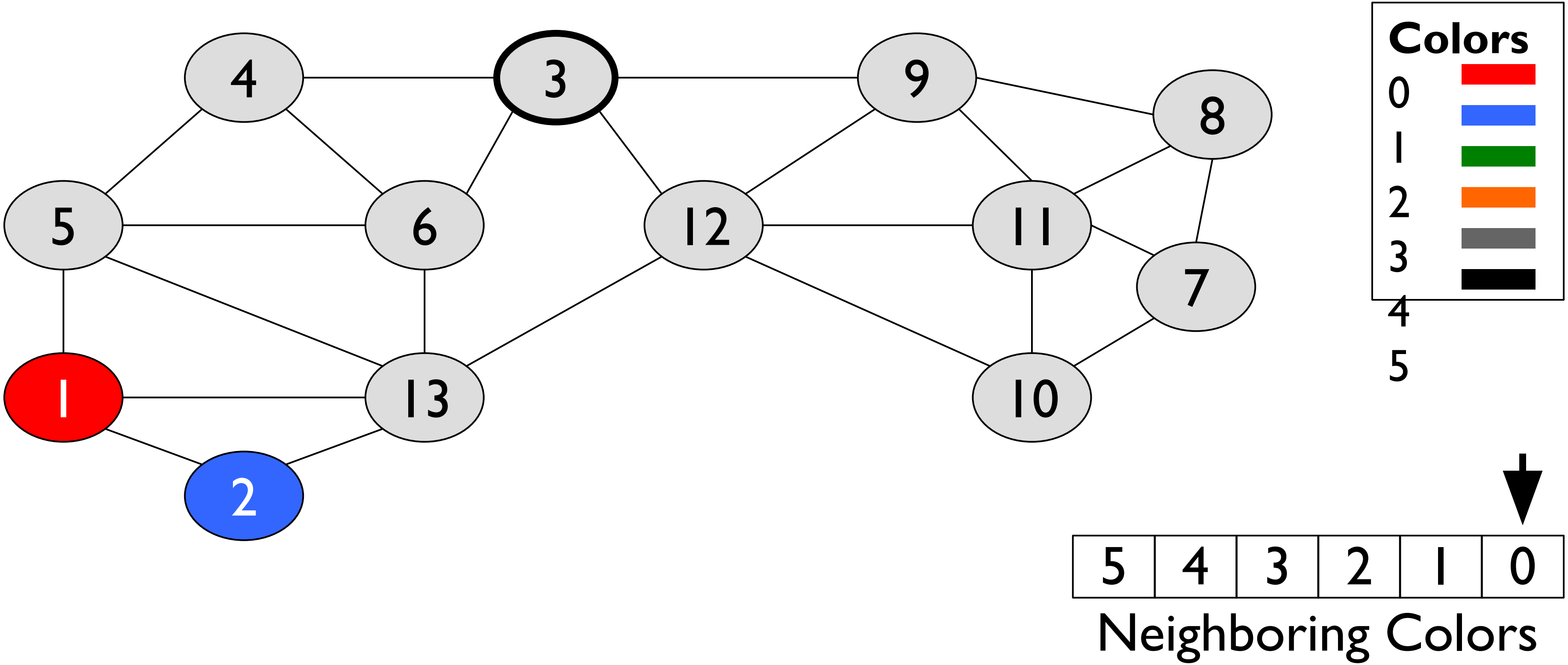
5	4	3	2	1	0

Neighboring Colors

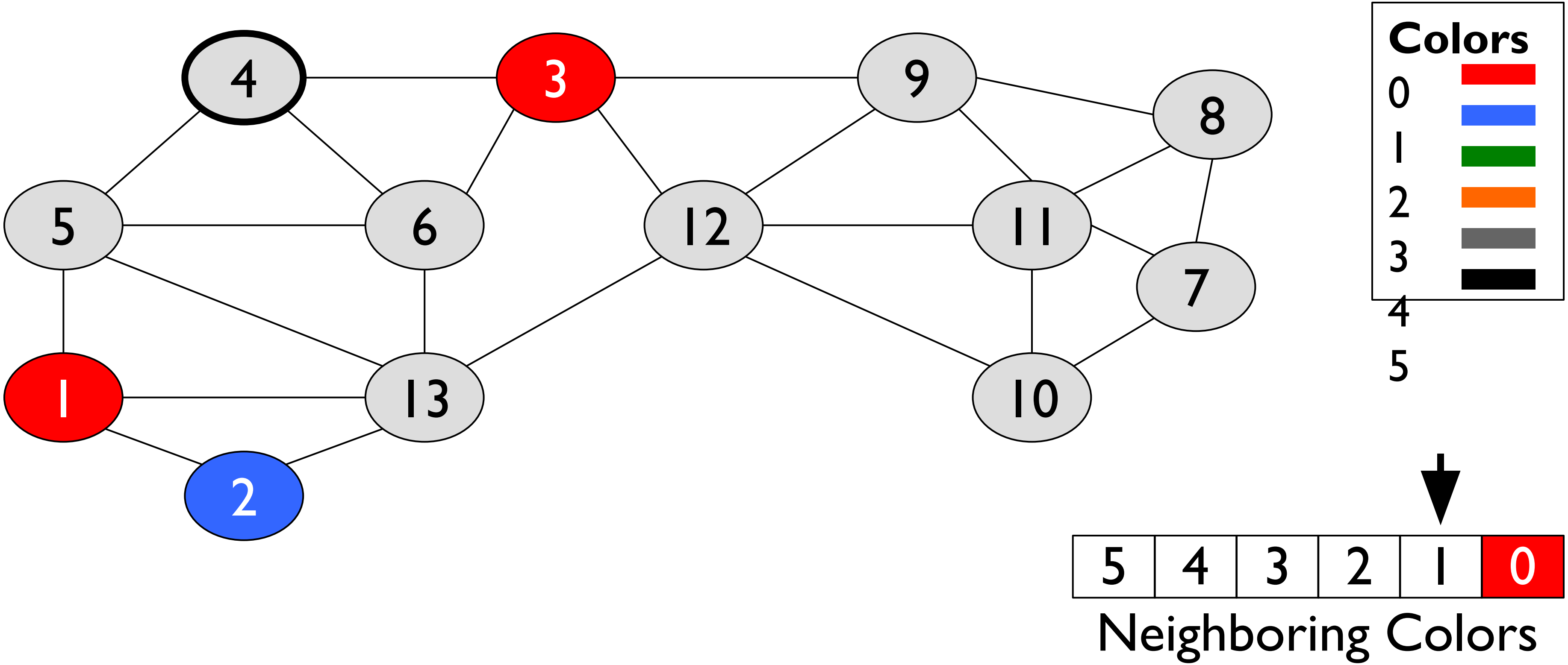
Example: Greedy Coloring



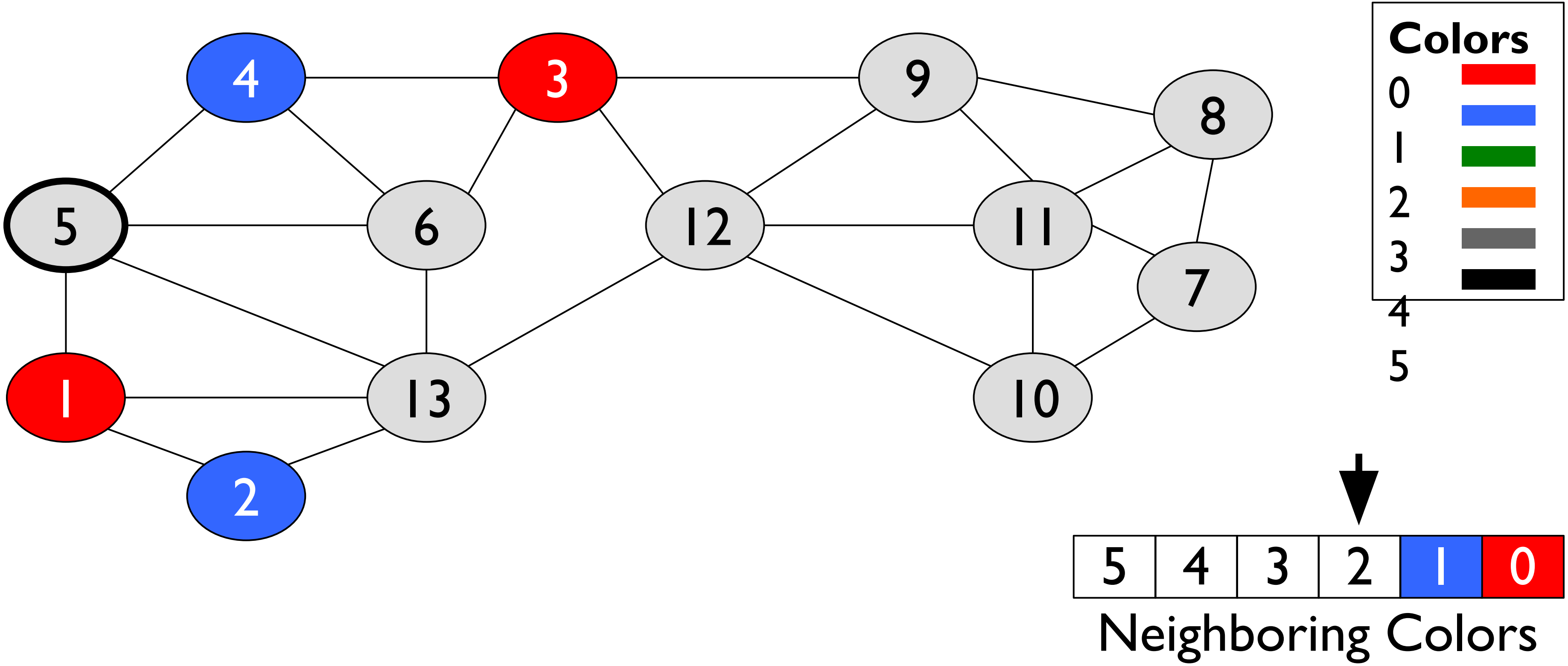
Example: Greedy Coloring



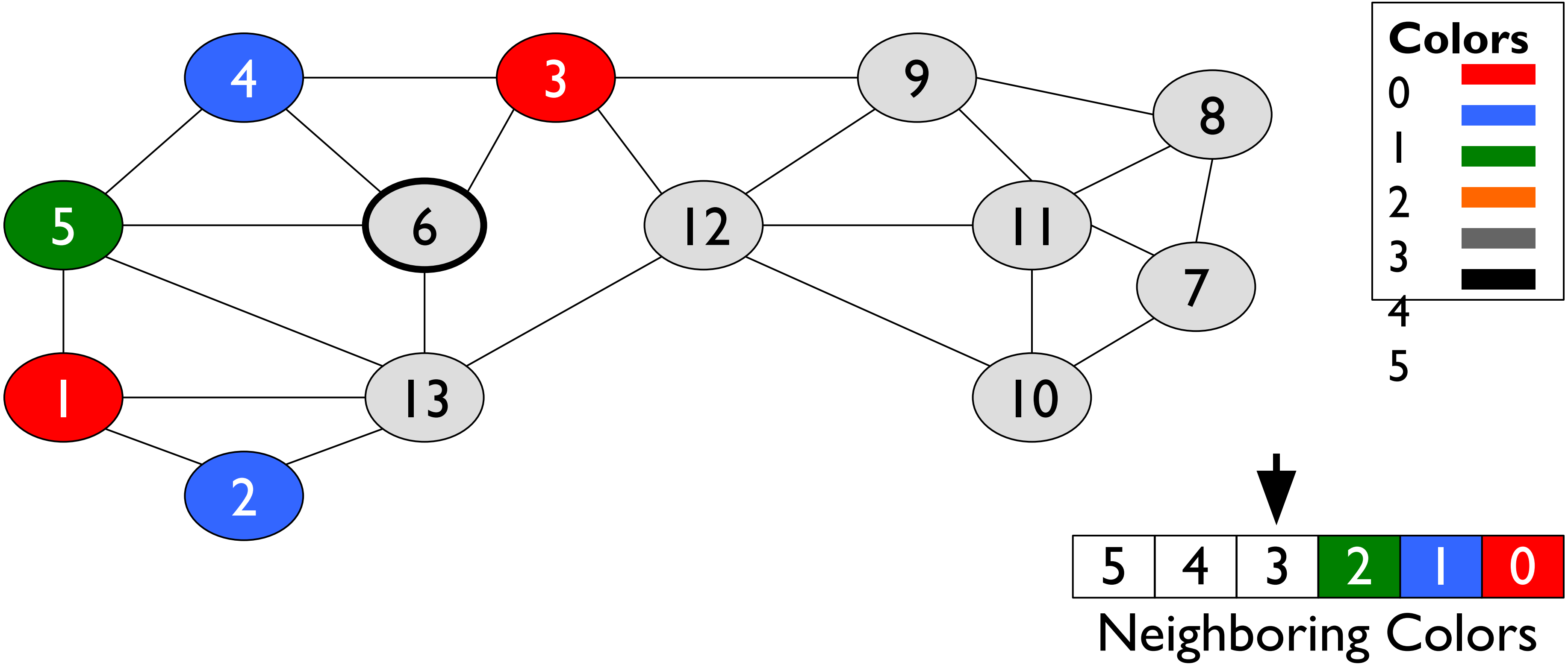
Example: Greedy Coloring



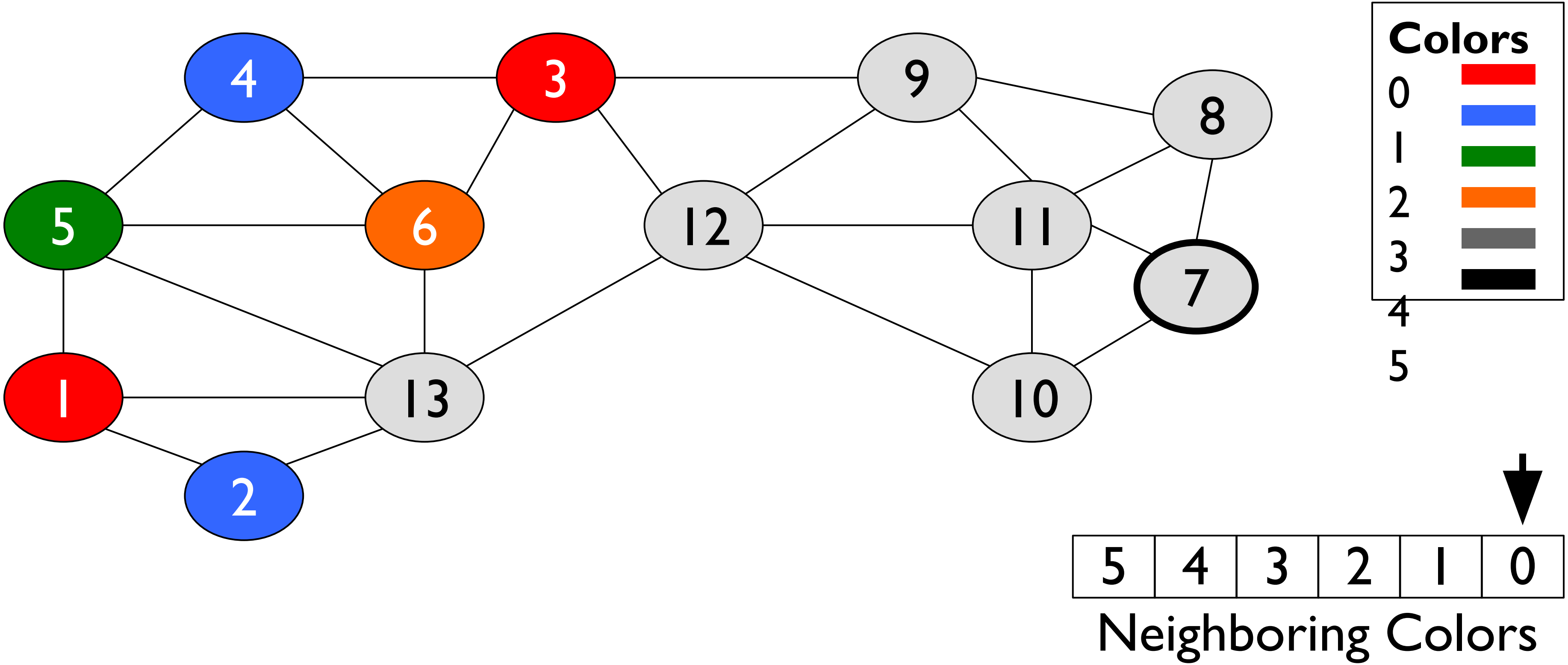
Example: Greedy Coloring



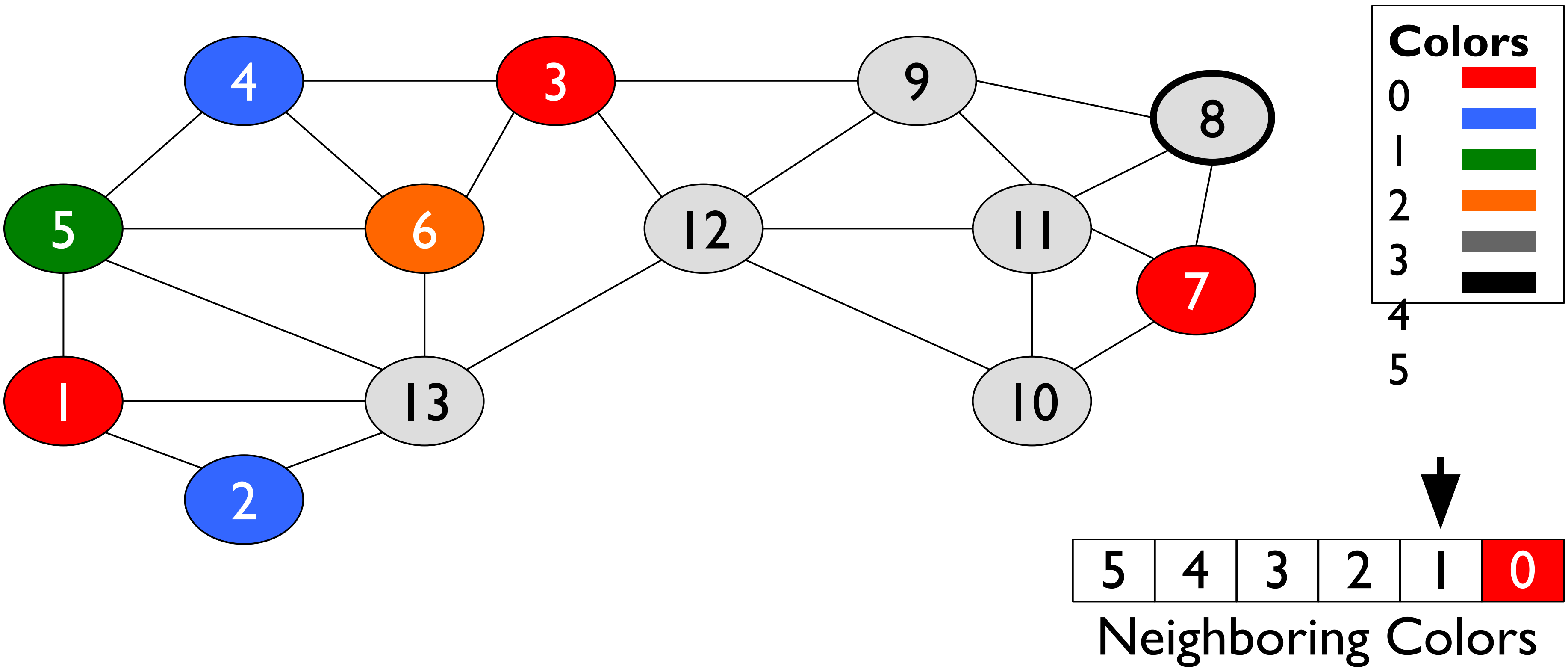
Example: Greedy Coloring



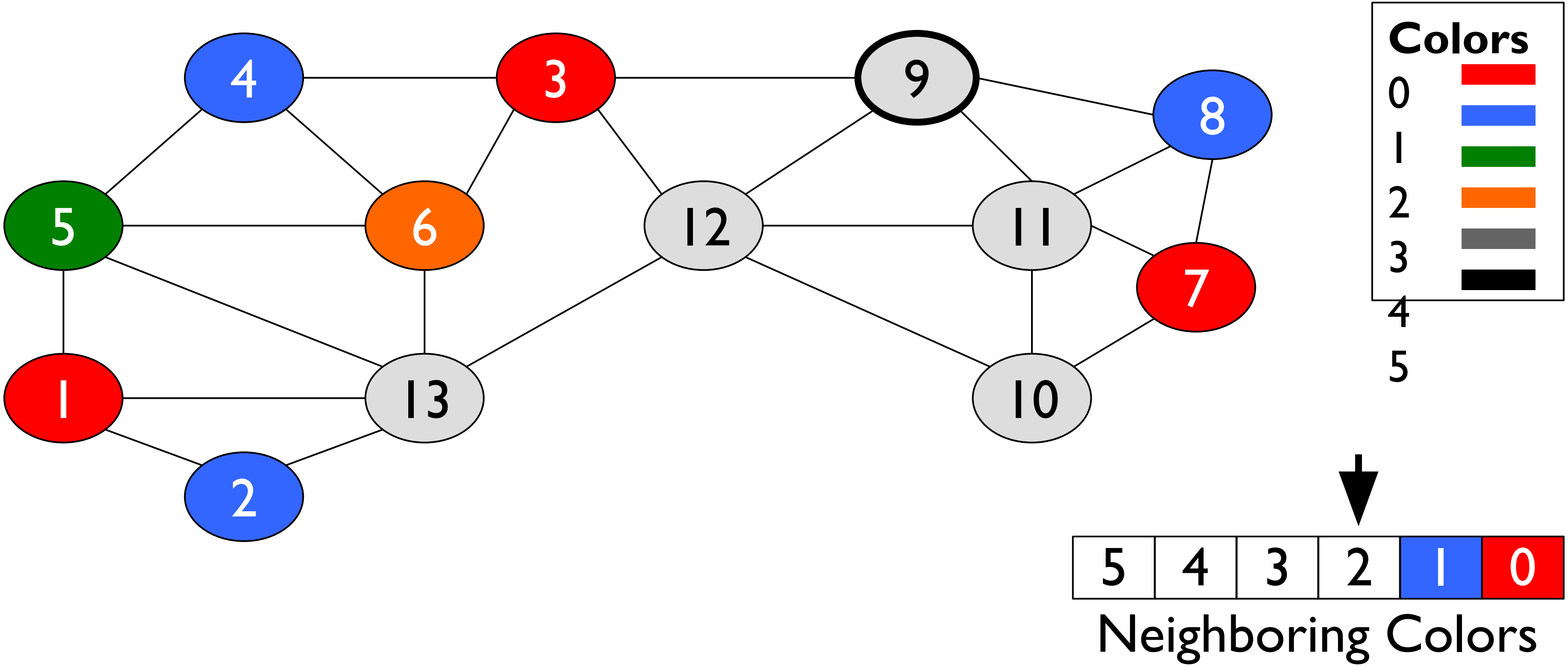
Example: Greedy Coloring



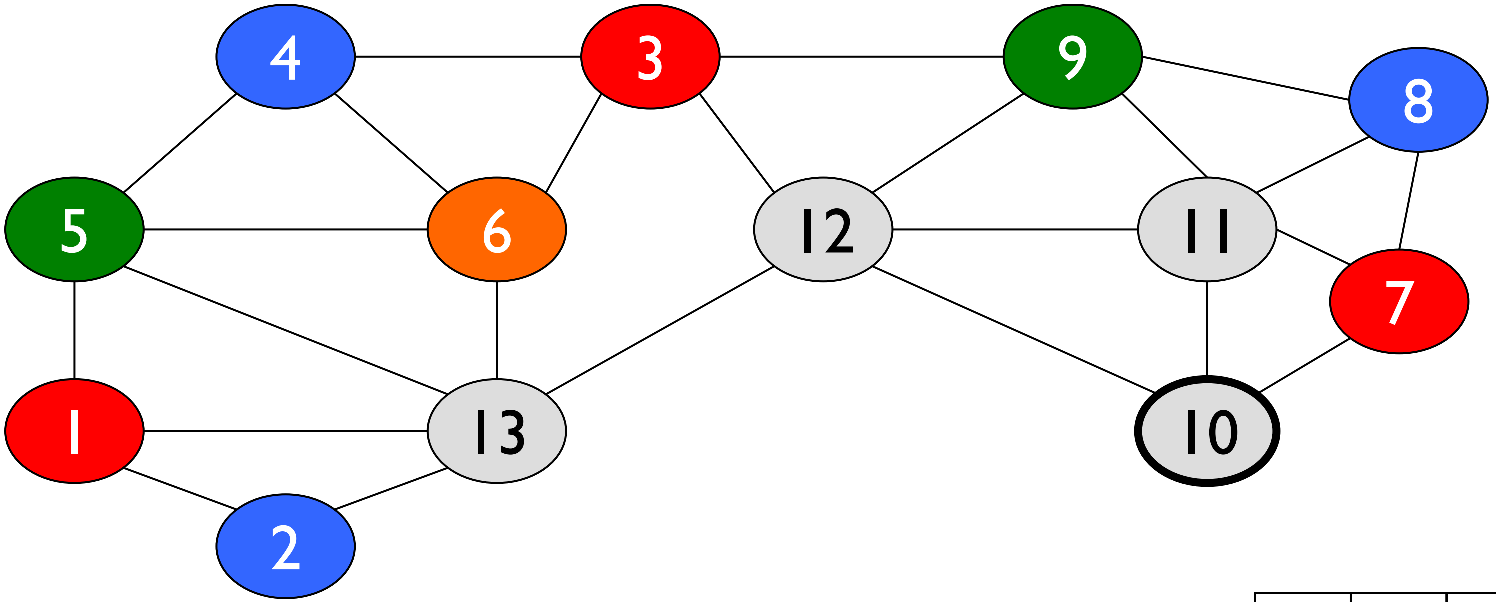
Example: Greedy Coloring

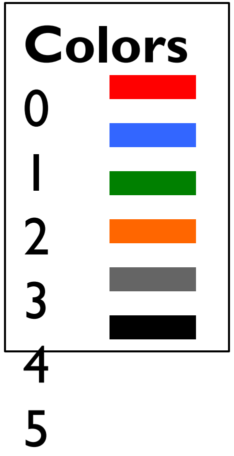


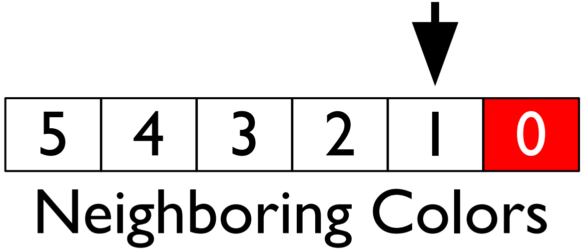
Example: Greedy Coloring



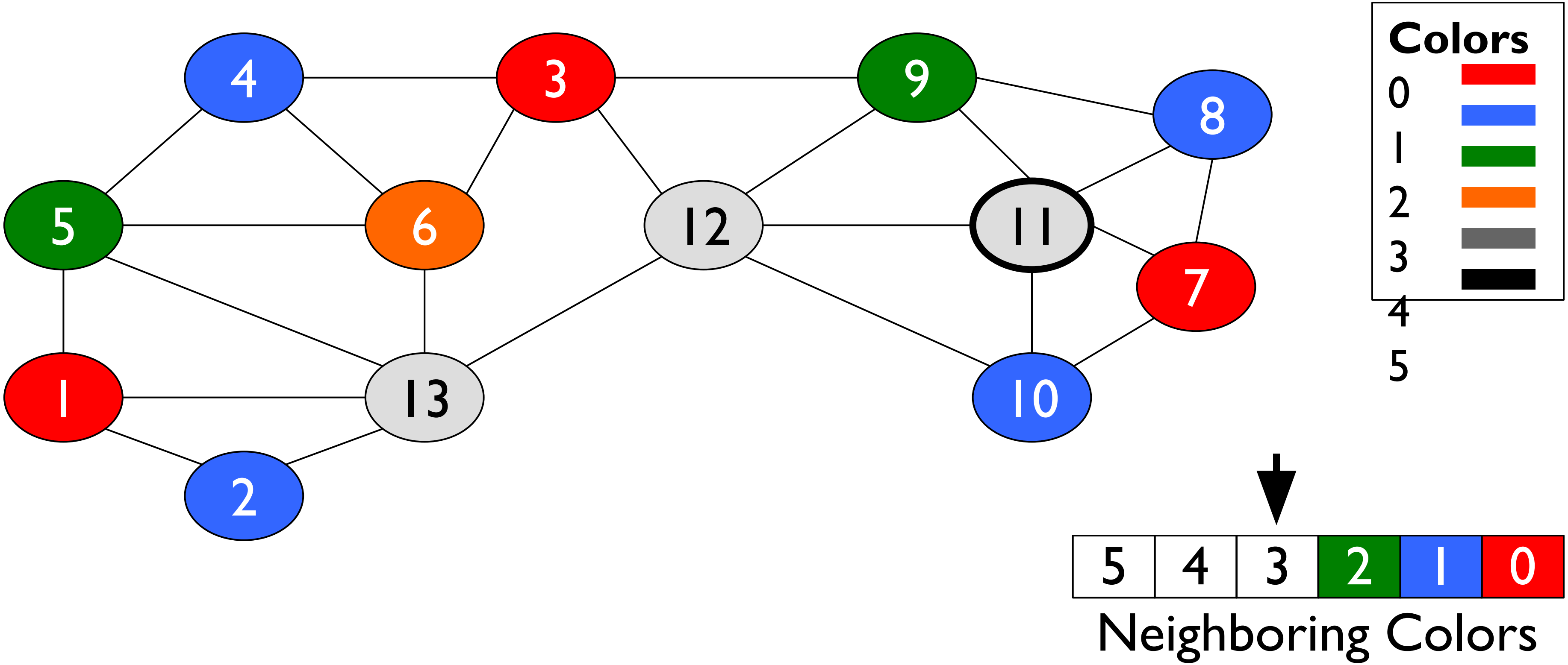
Example: Greedy Coloring



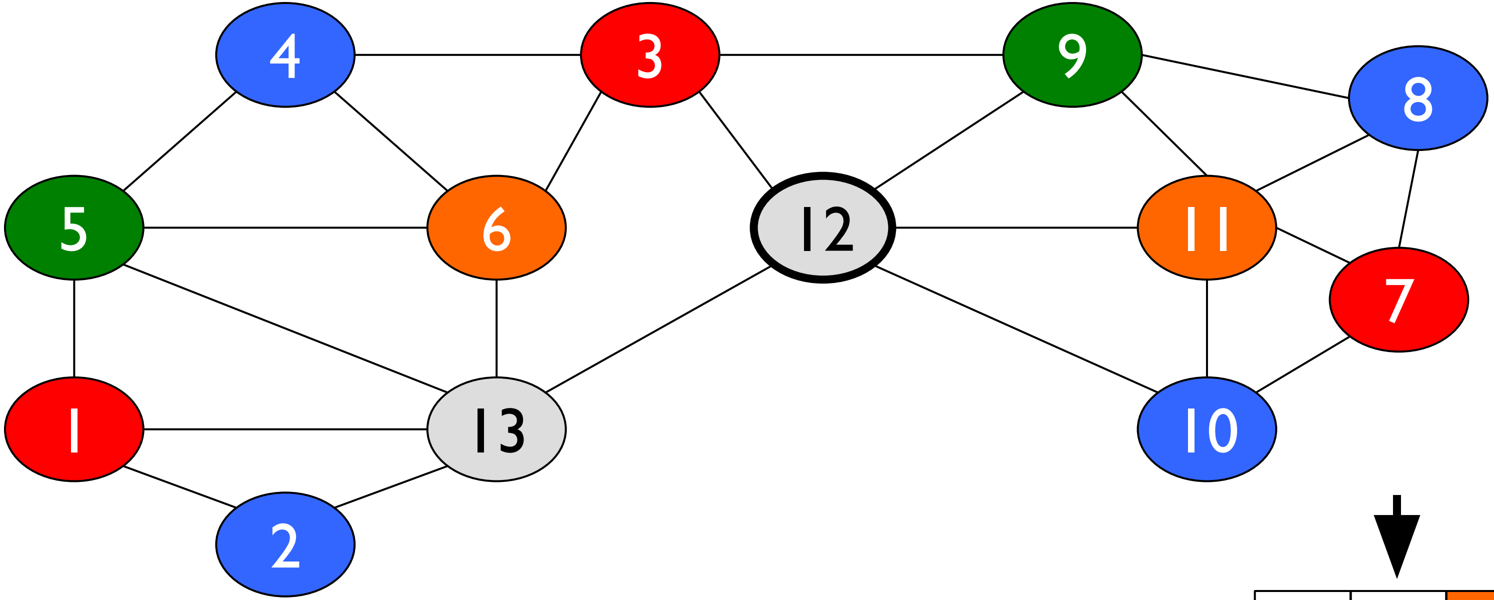


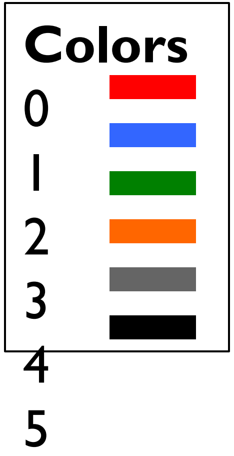


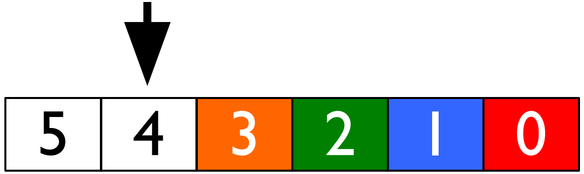
Example: Greedy Coloring



Example: Greedy Coloring

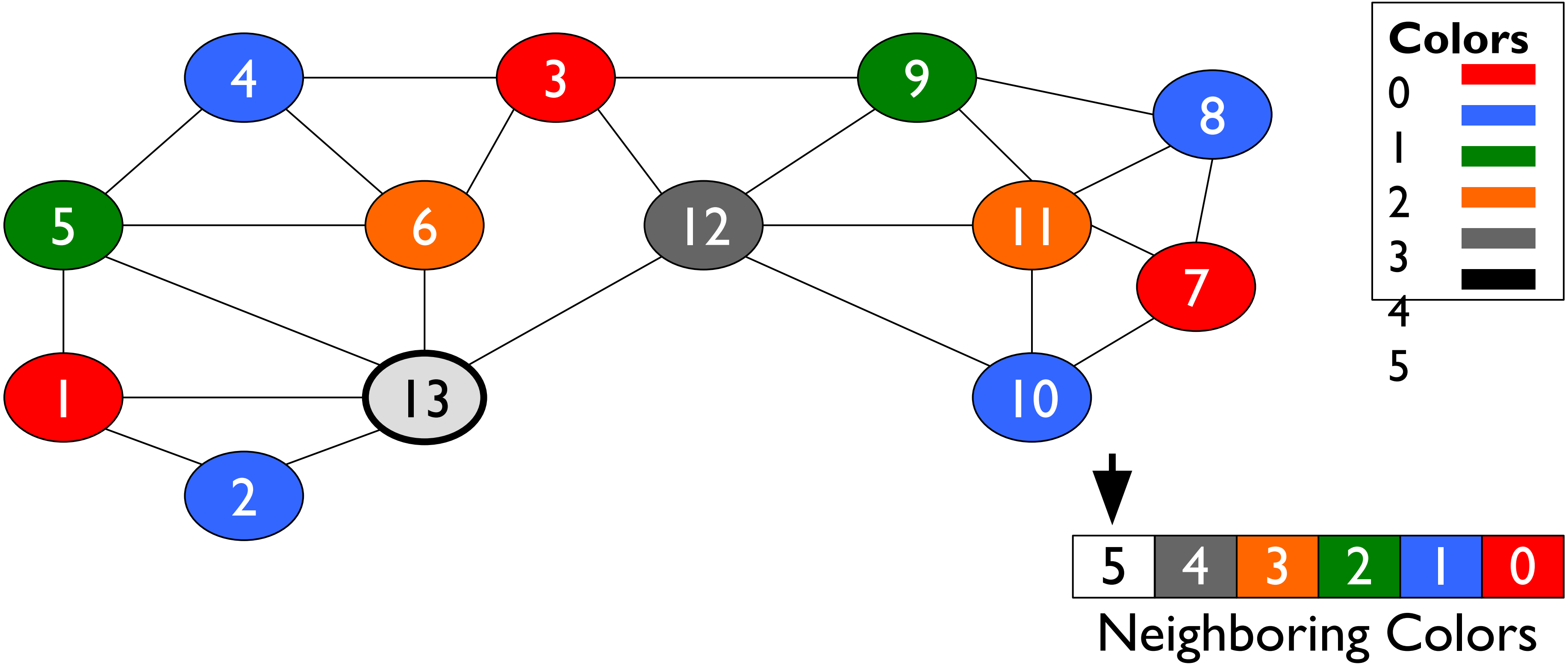




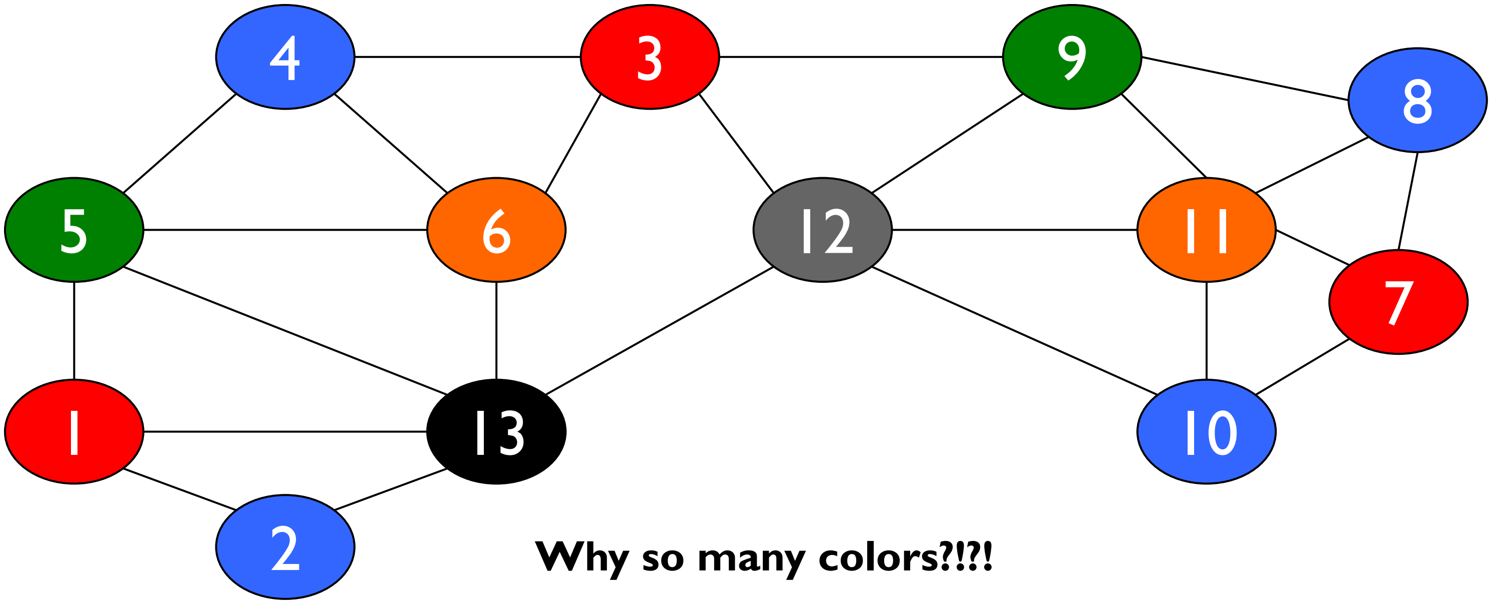


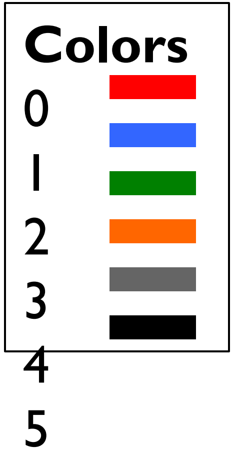
Neighboring Colors

Example: Greedy Coloring



Example: Greedy Coloring





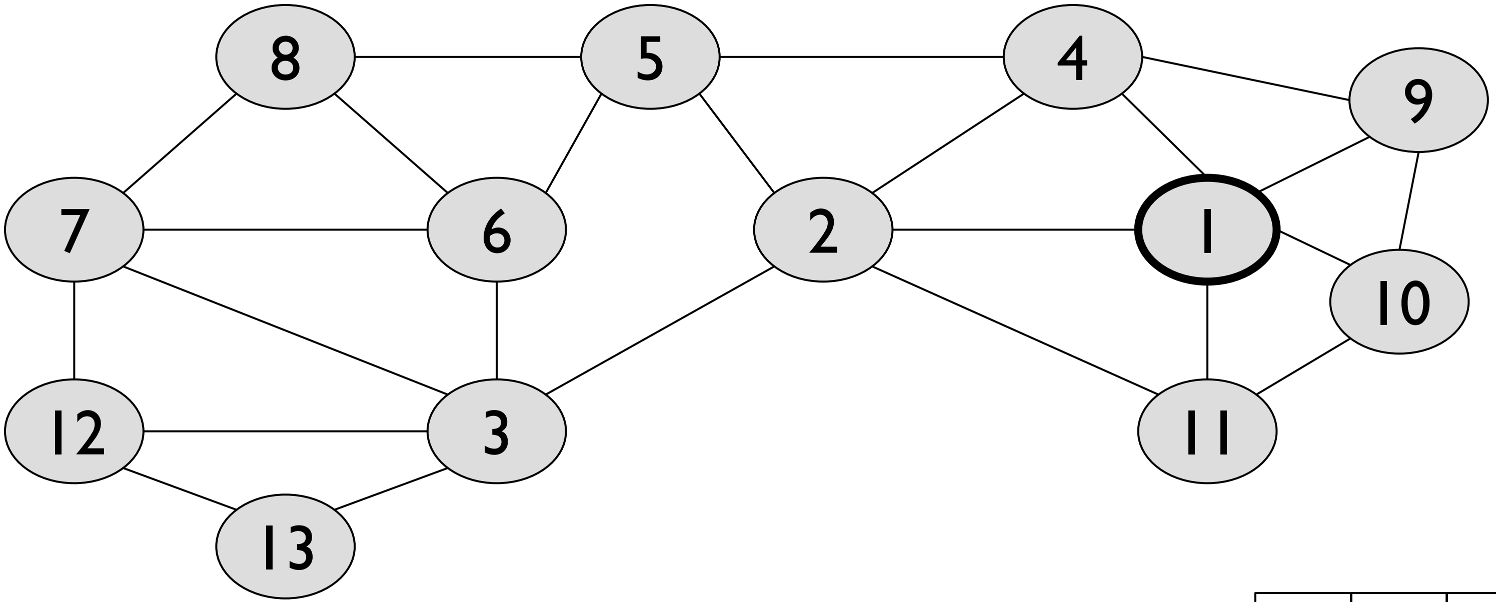
Why so many colors?!?!

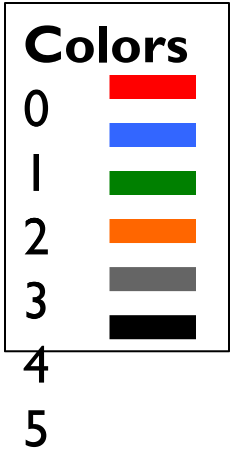
The order in which we color the vertices influences the number of colors.

Definitions: Ordering Heuristics (ρ(v))

- FF: First fit
- R: Random
- LF: Largest degree first
- SL: Smallest degree last
  - Remove all lowest degree vertices and recursively color graph
- ID: Incidence-degree
  - “Color degree”
- SD: Saturation degree
  - “Distinct color degree”

Example: Largest-First

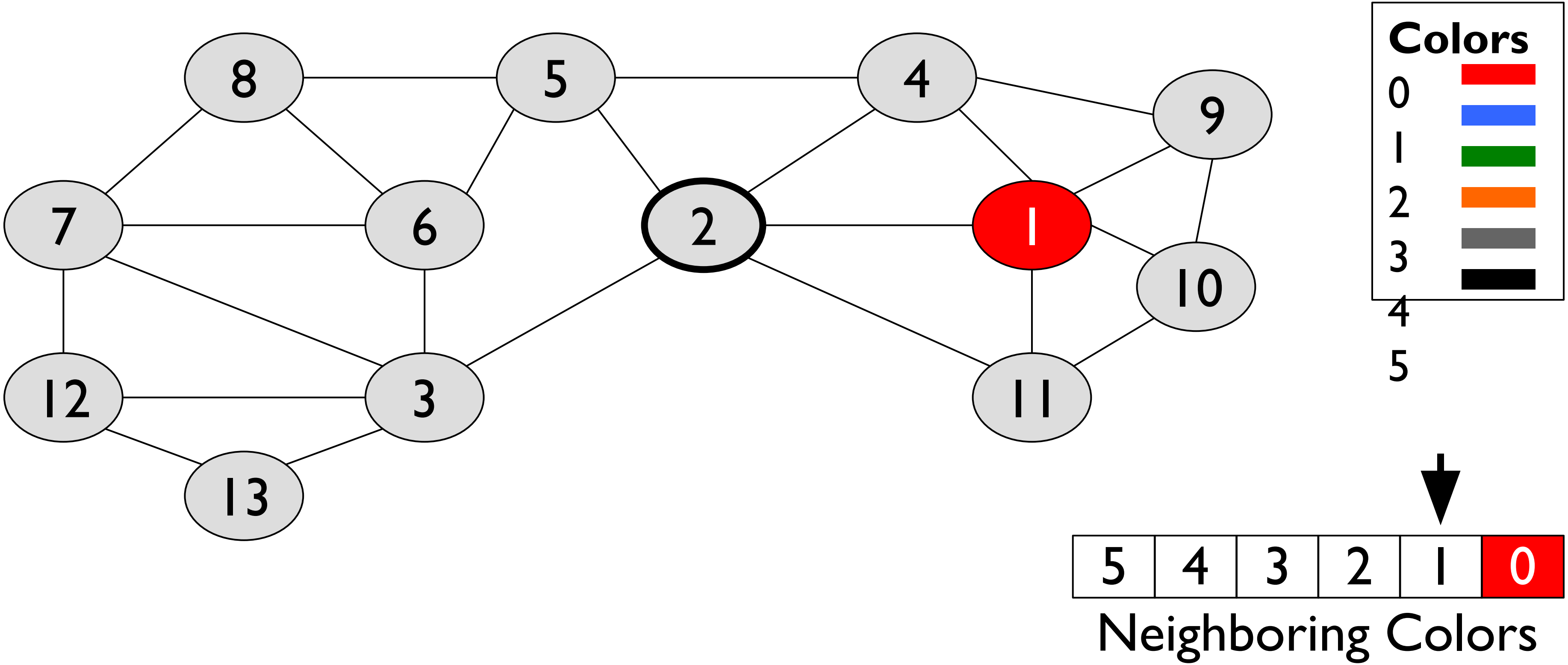




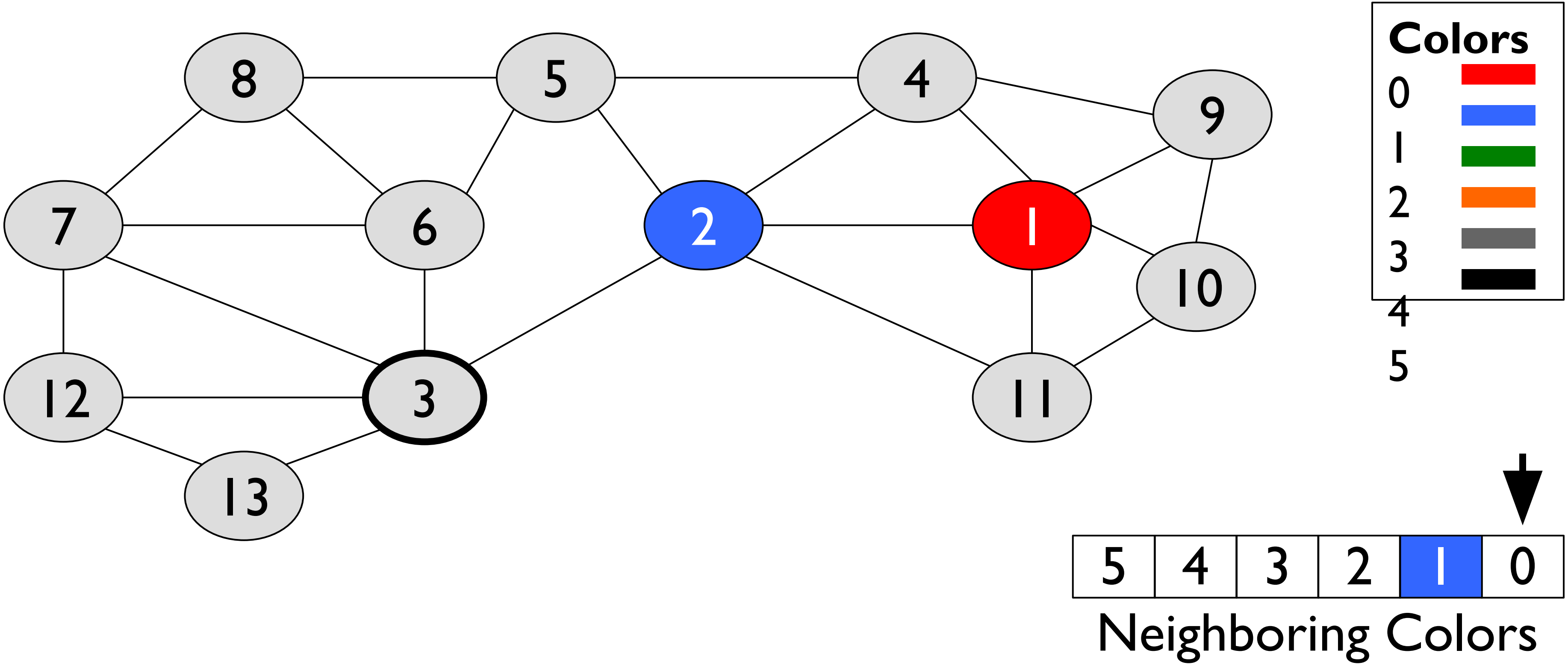
5	4	3	2	1	0

Neighboring Colors

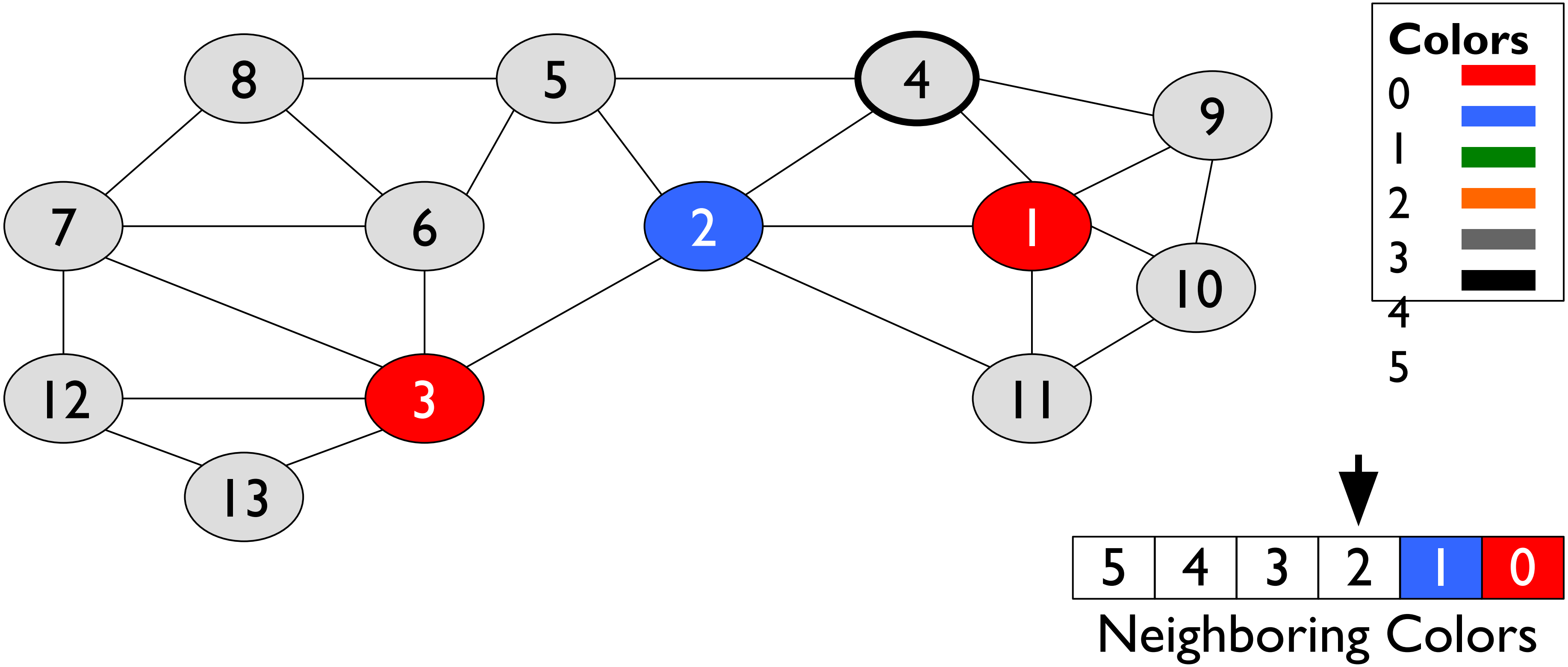
Example: Largest-First



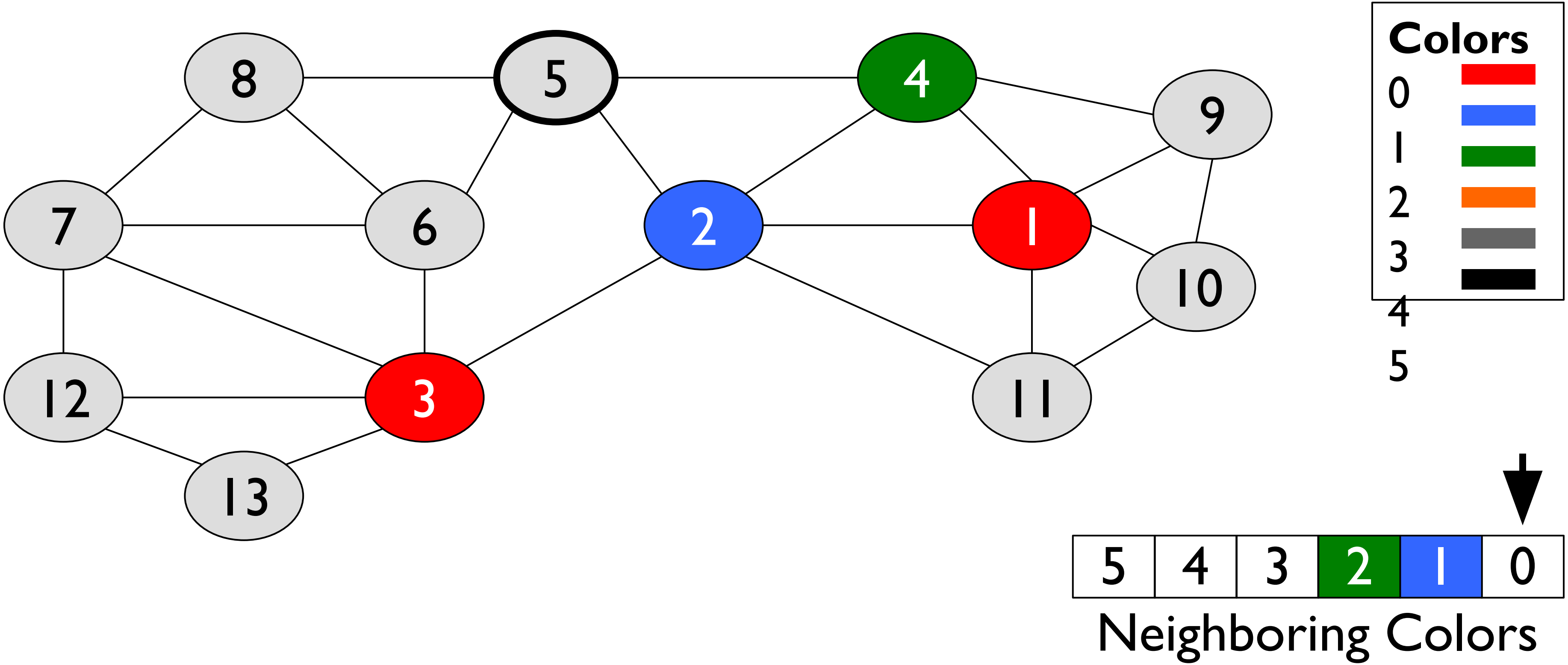
Example: Largest-First



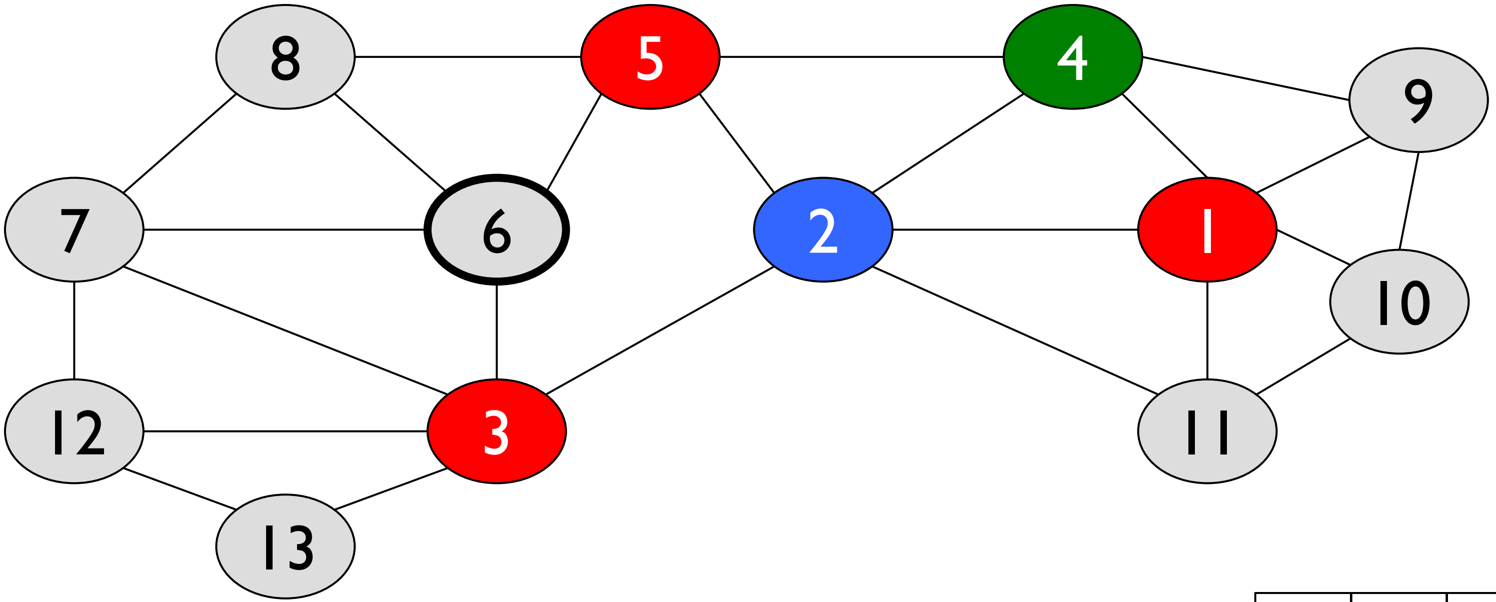
Example: Largest-First

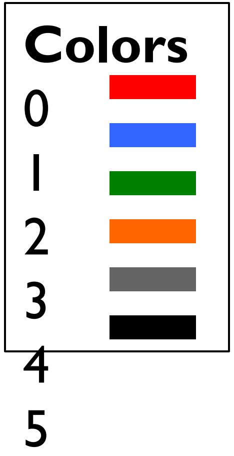


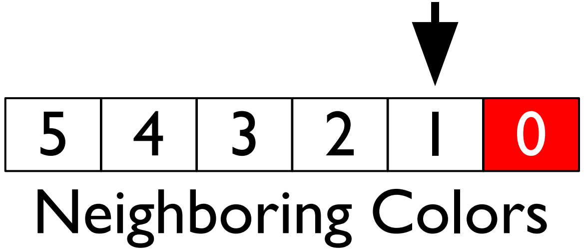
Example: Largest-First



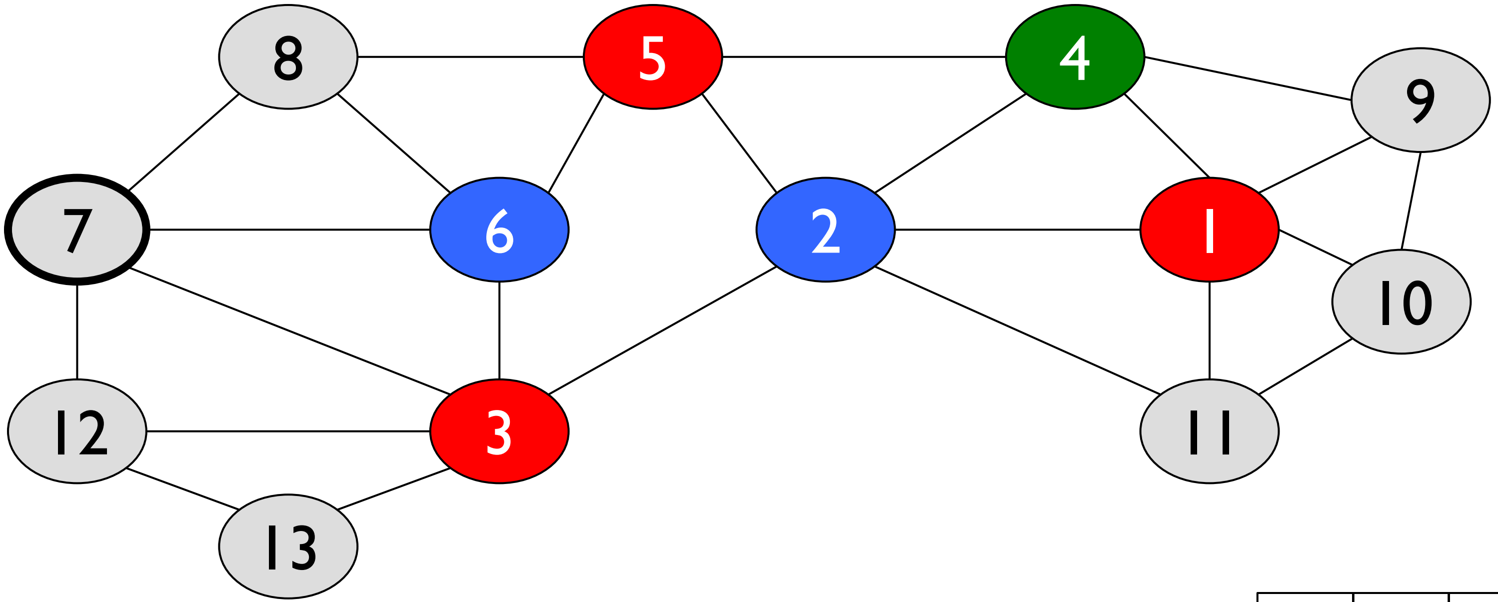
Example: Largest-First

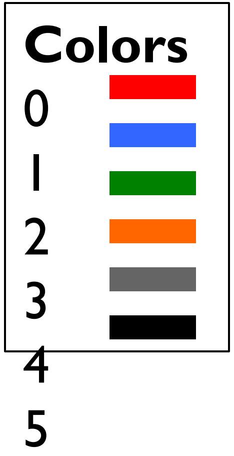


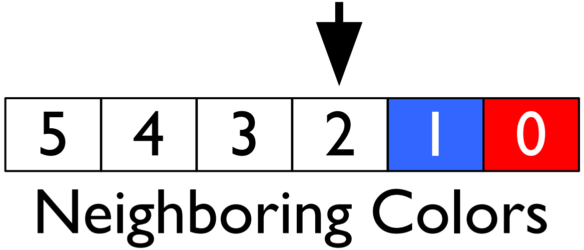




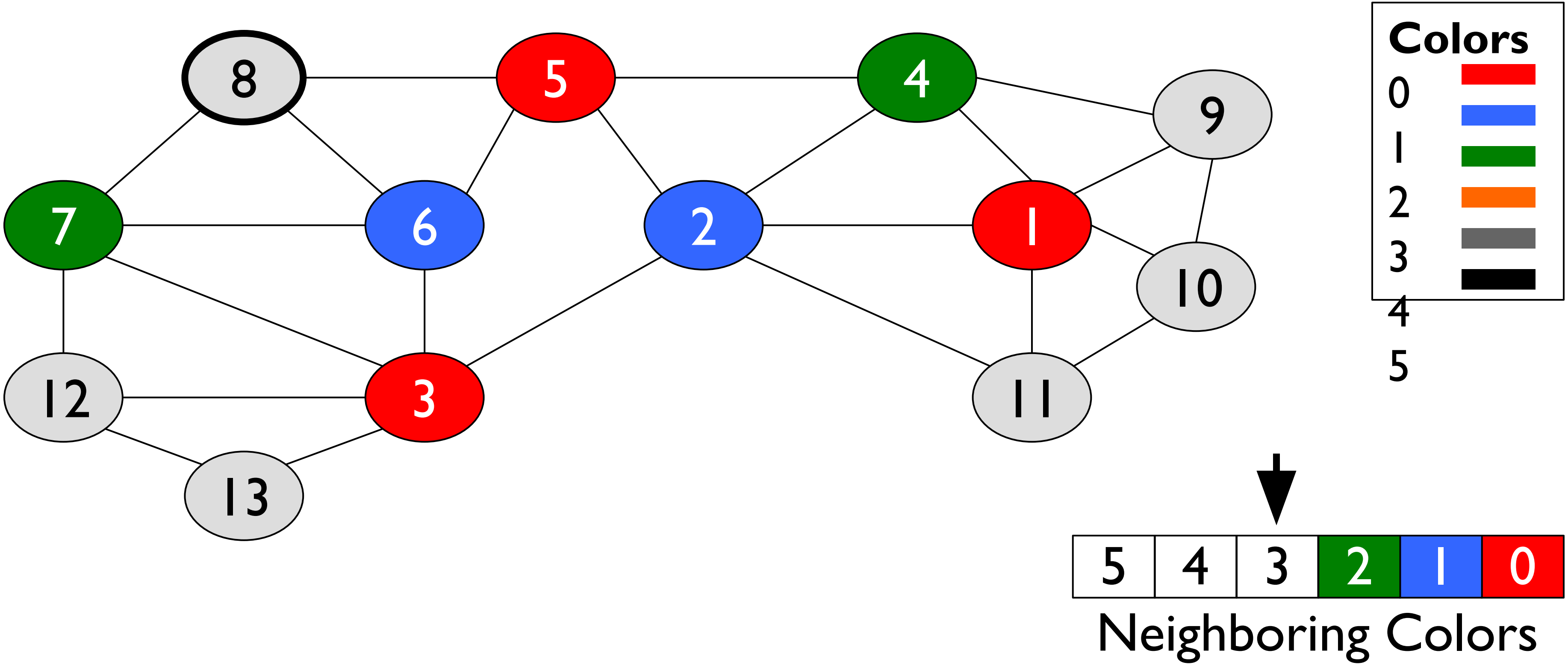
Example: Largest-First



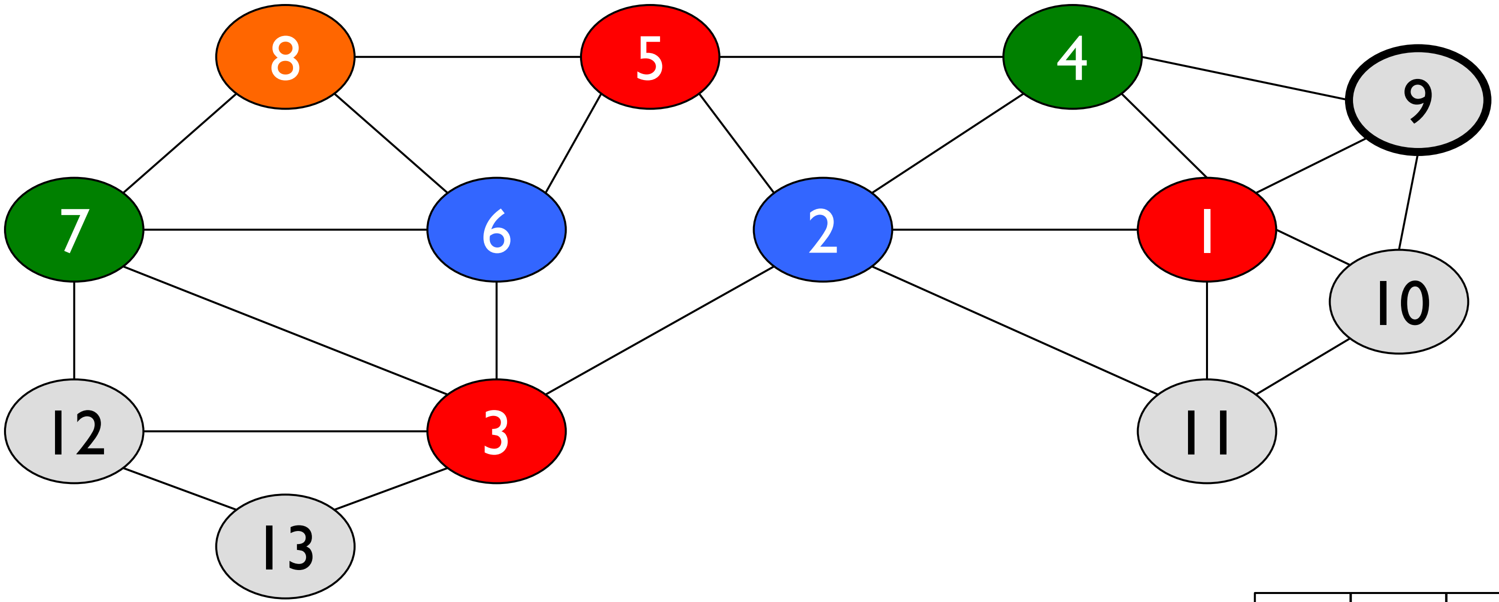


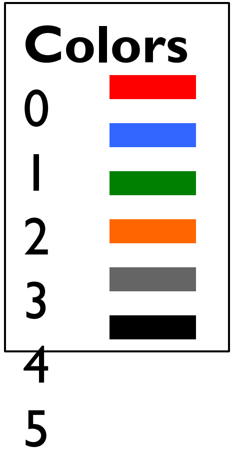


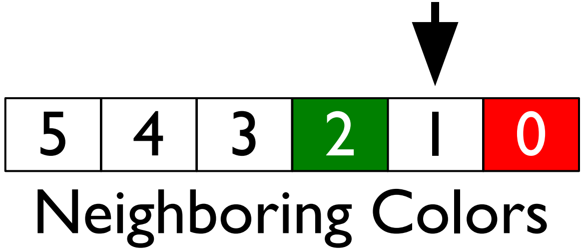
Example: Largest-First



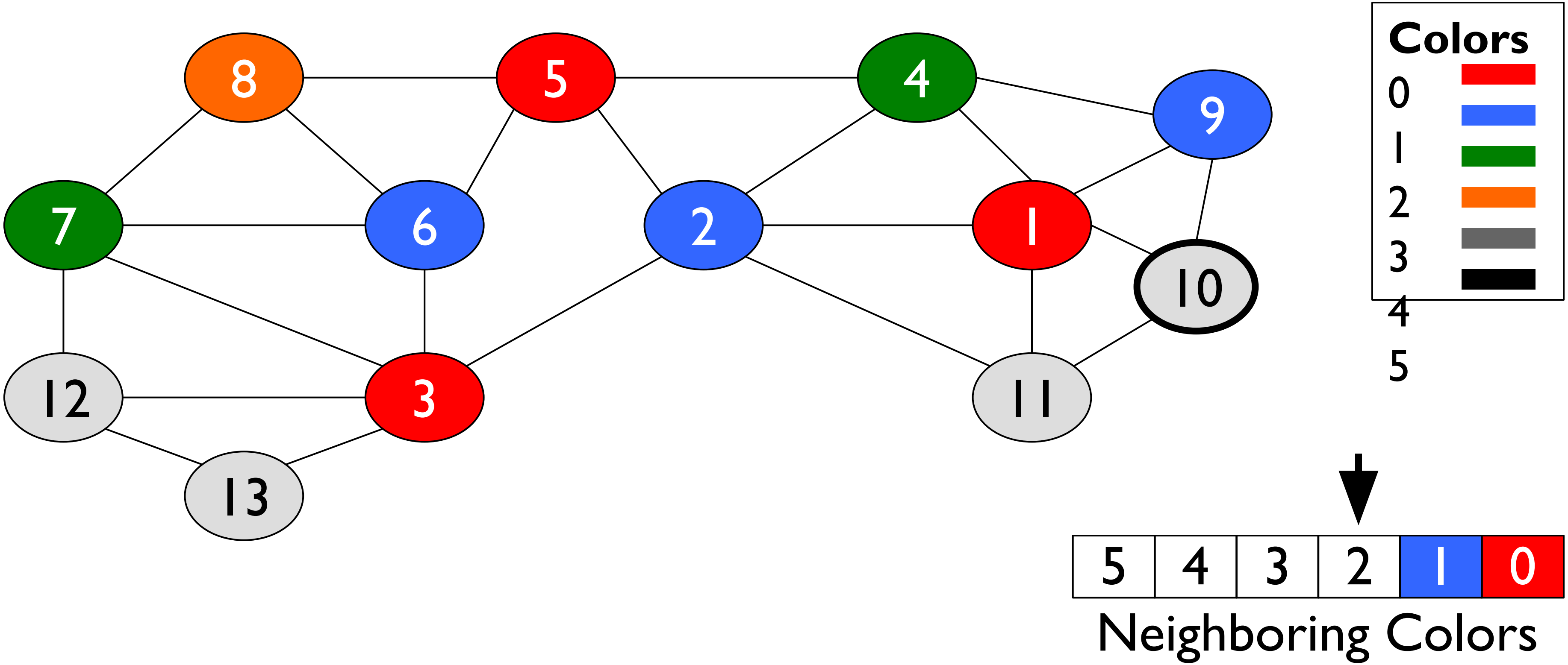
Example: Largest-First



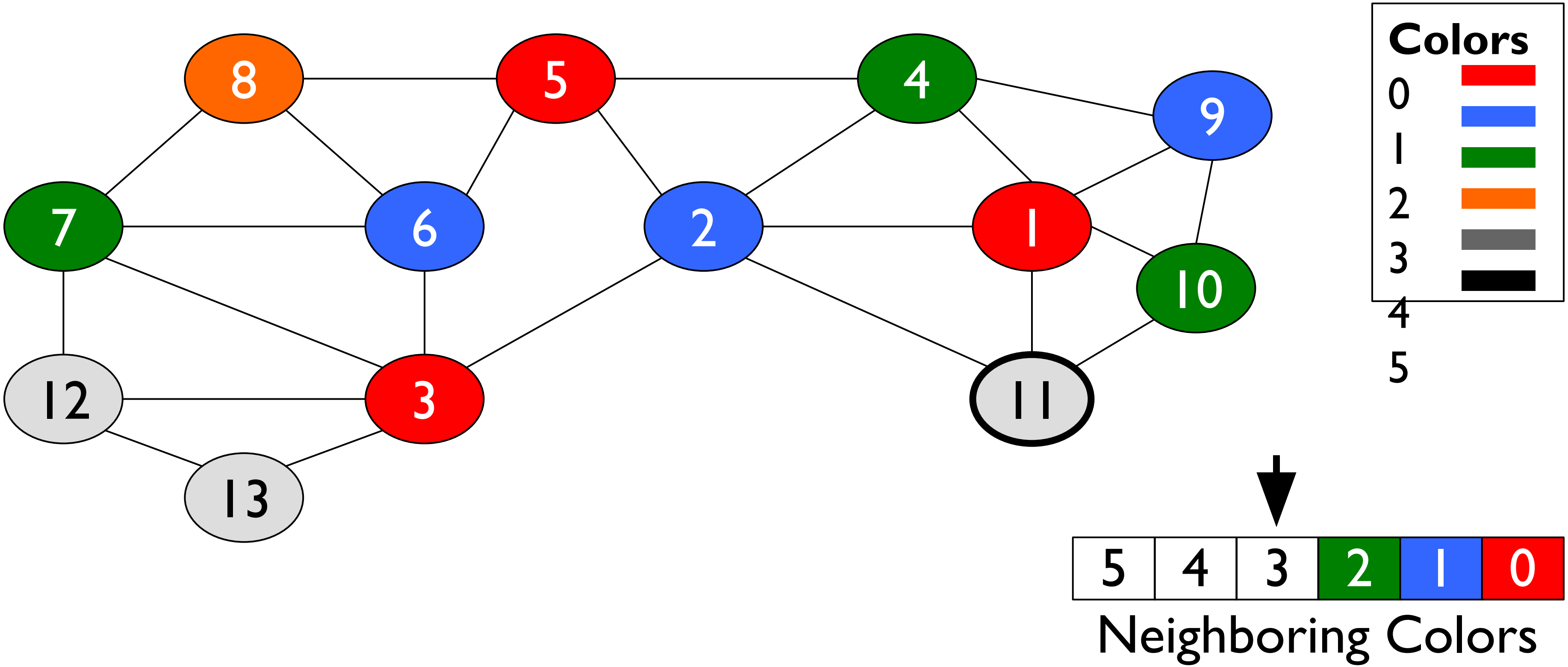




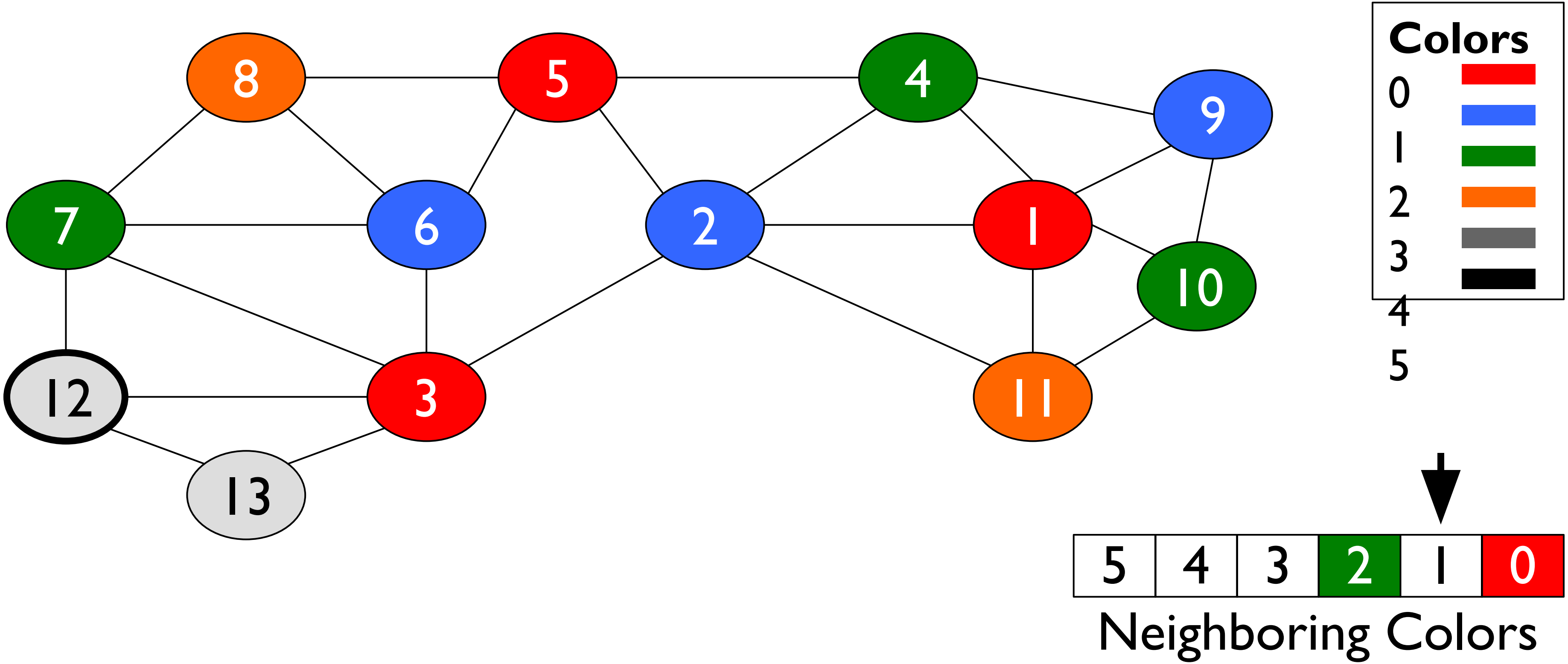
Example: Largest-First



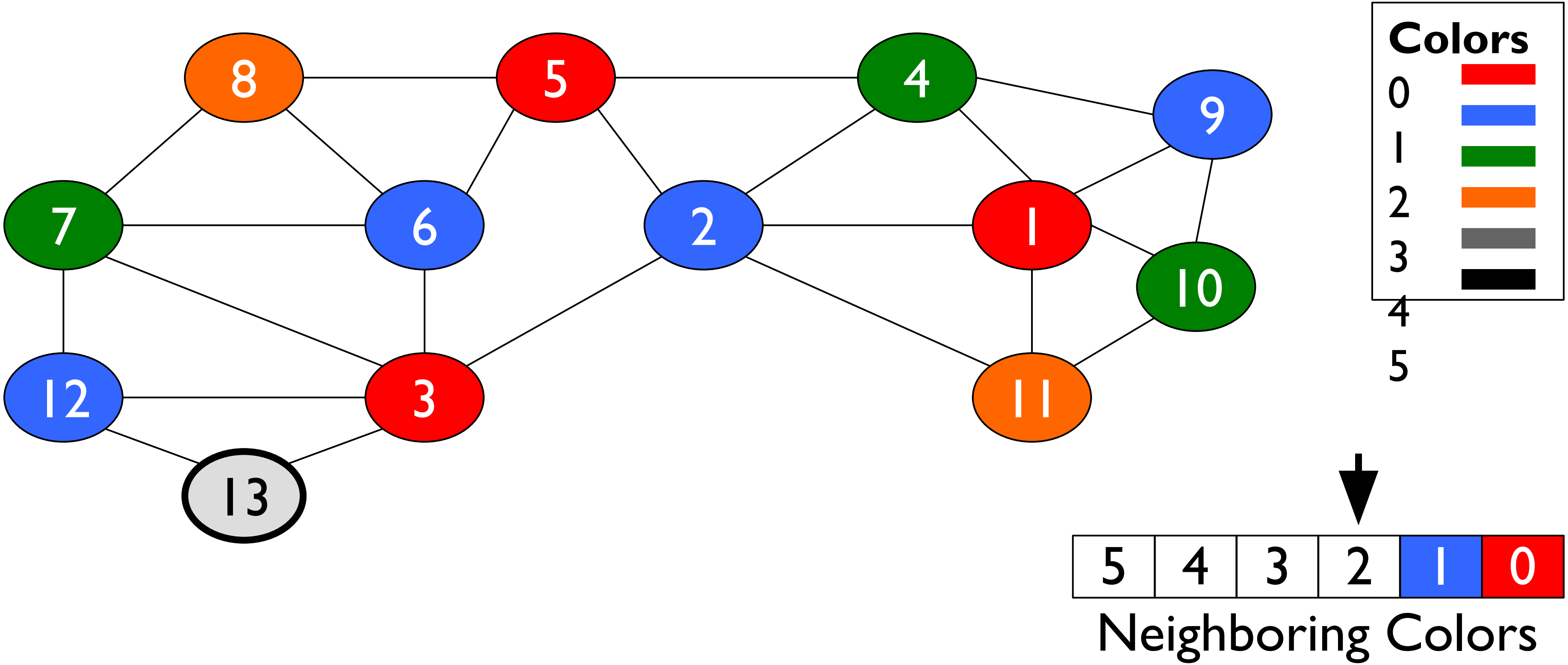
Example: Largest-First



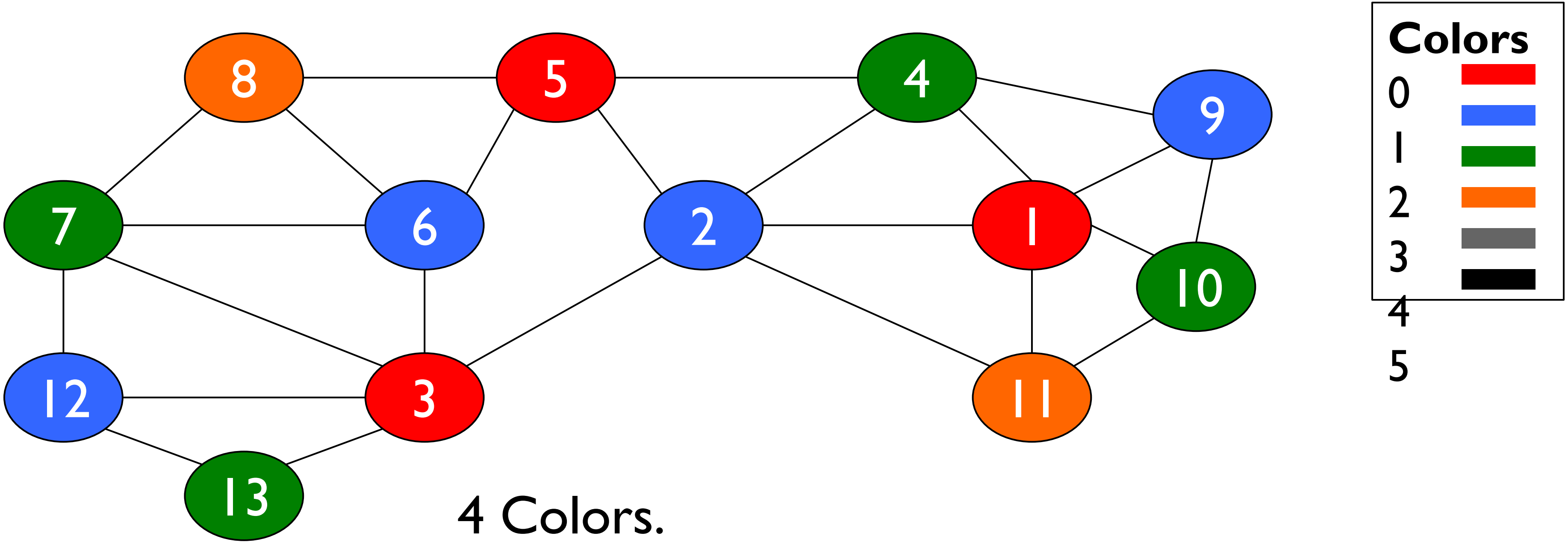
Example: Largest-First



Example: Largest-First

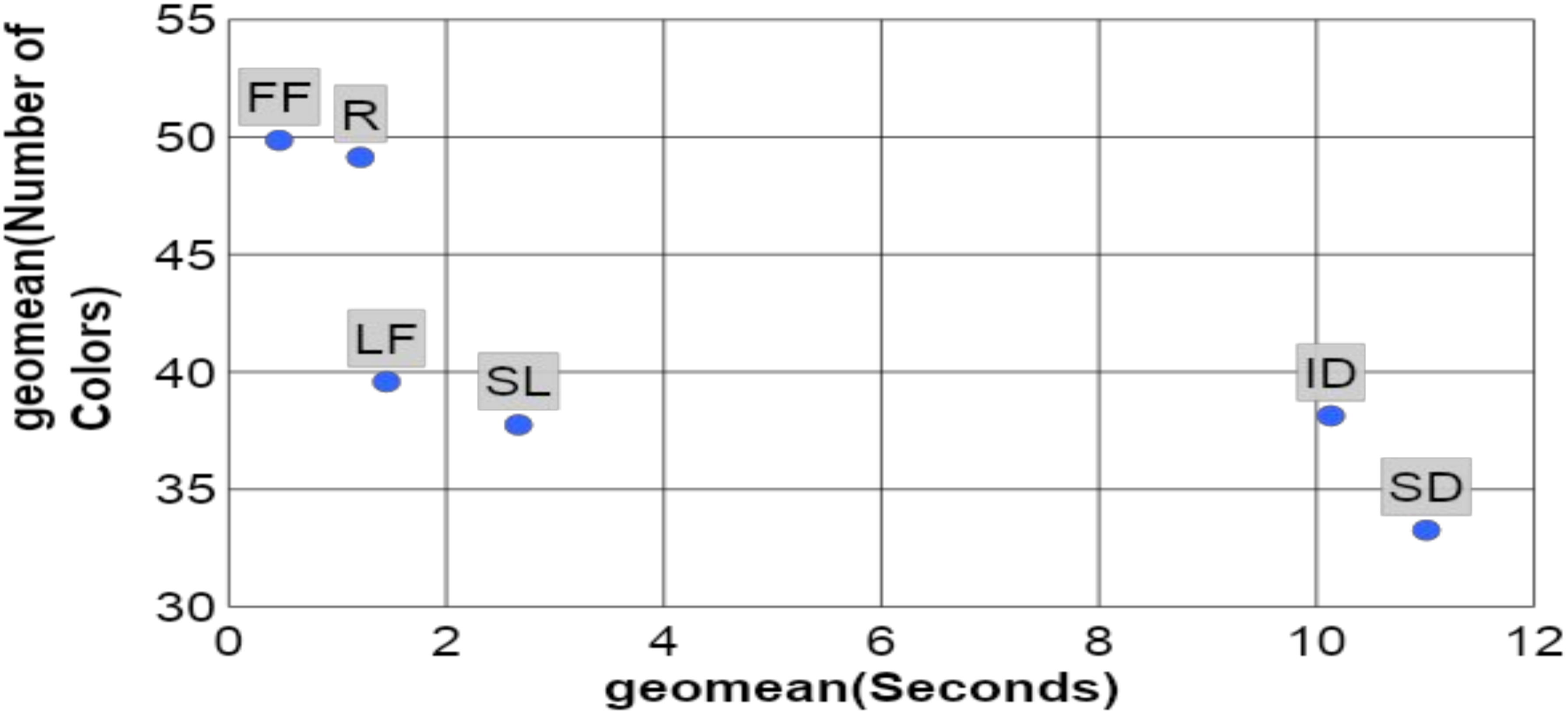


Example: Largest-First

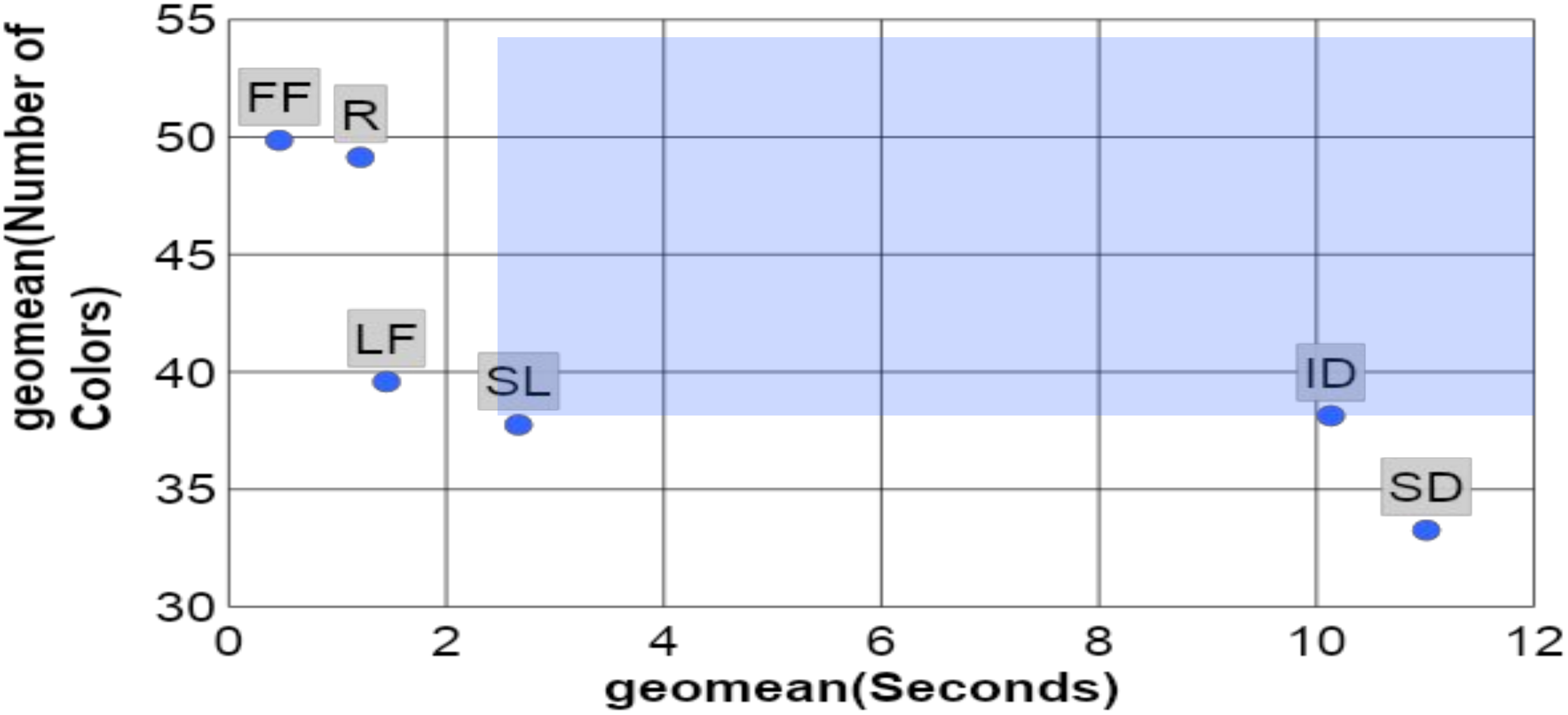


4 Colors.

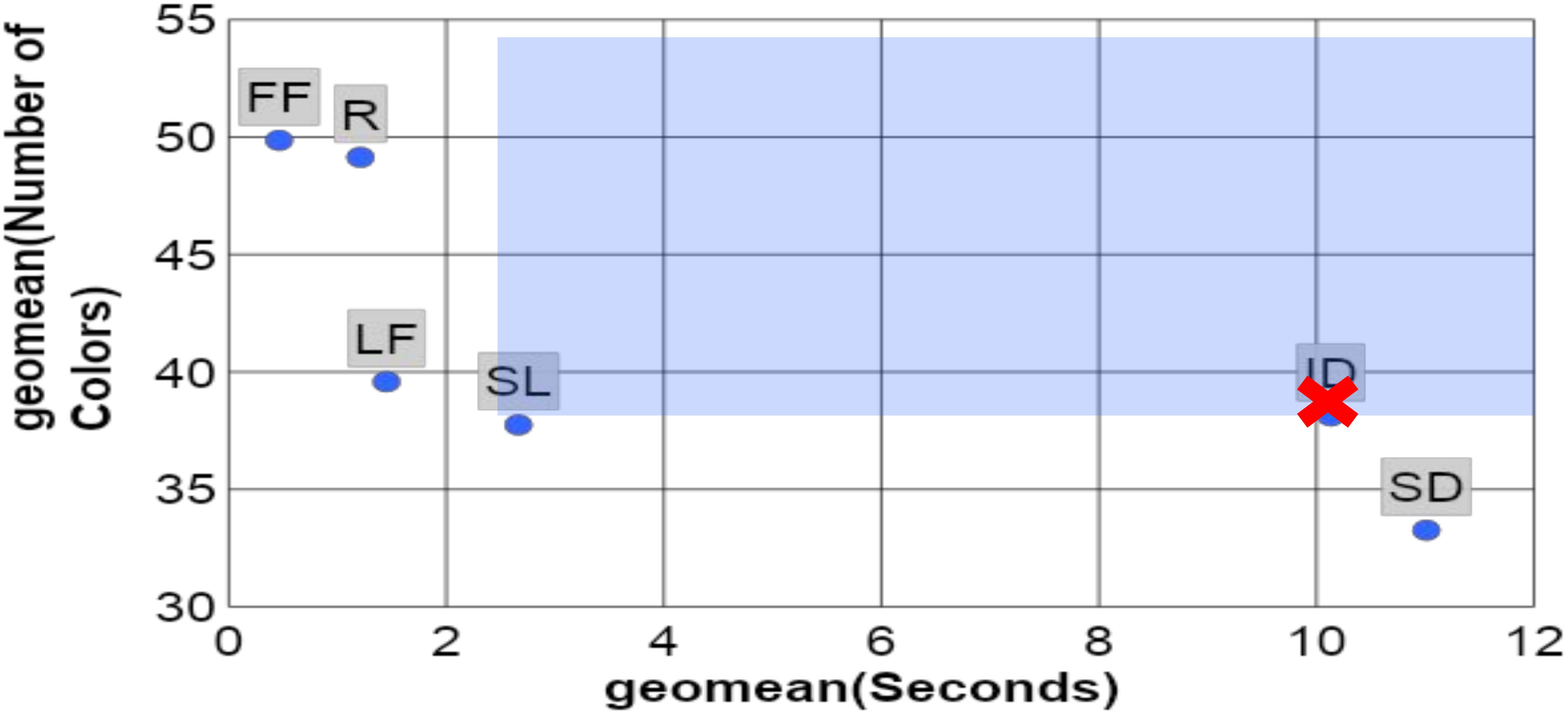
Quality vs. Serial Runtime



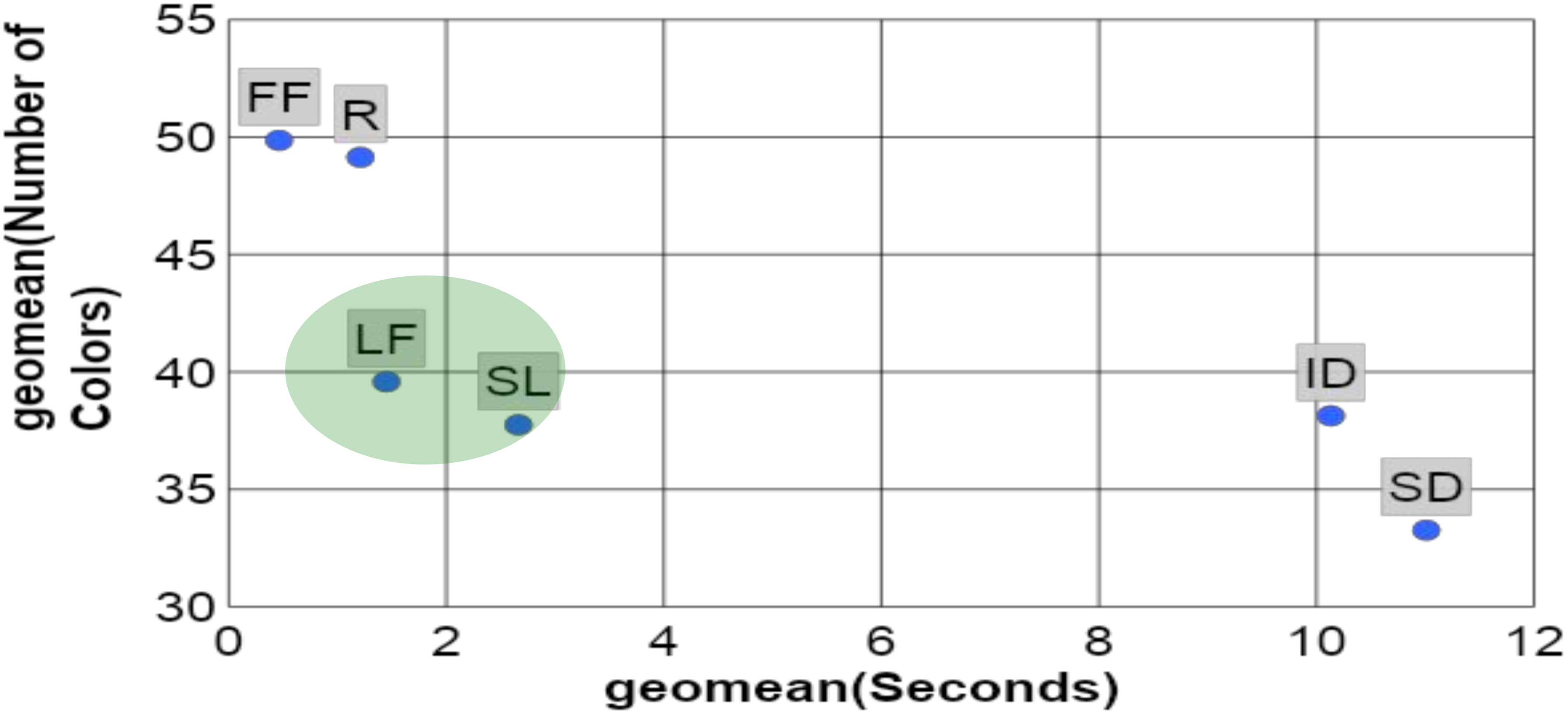
Quality vs. Serial Runtime



Quality vs. Serial Runtime



Quality vs. Serial Runtime



Parallel Greedy Coloring

```
JP(G)
7   let G = (V, E, ρ)
8   parallel for v ∈ V
9       v.pred = {u ∈ V : (u, v) ∈ E and ρ(u) > ρ(v)}
10      v.succ = {u ∈ V : (u, v) ∈ E and ρ(u) < ρ(v)}
11      v.counter = |v.pred|
12  parallel for v ∈ V
13      if v.pred == ∅
14          JP-COLOR(v)
```

```
JP-COLOR(v)
15  v.color = GET-COLOR(v)
16  parallel for u ∈ v.succ
17      if JOIN(u.counter) == 0
18          JP-COLOR(u)
```

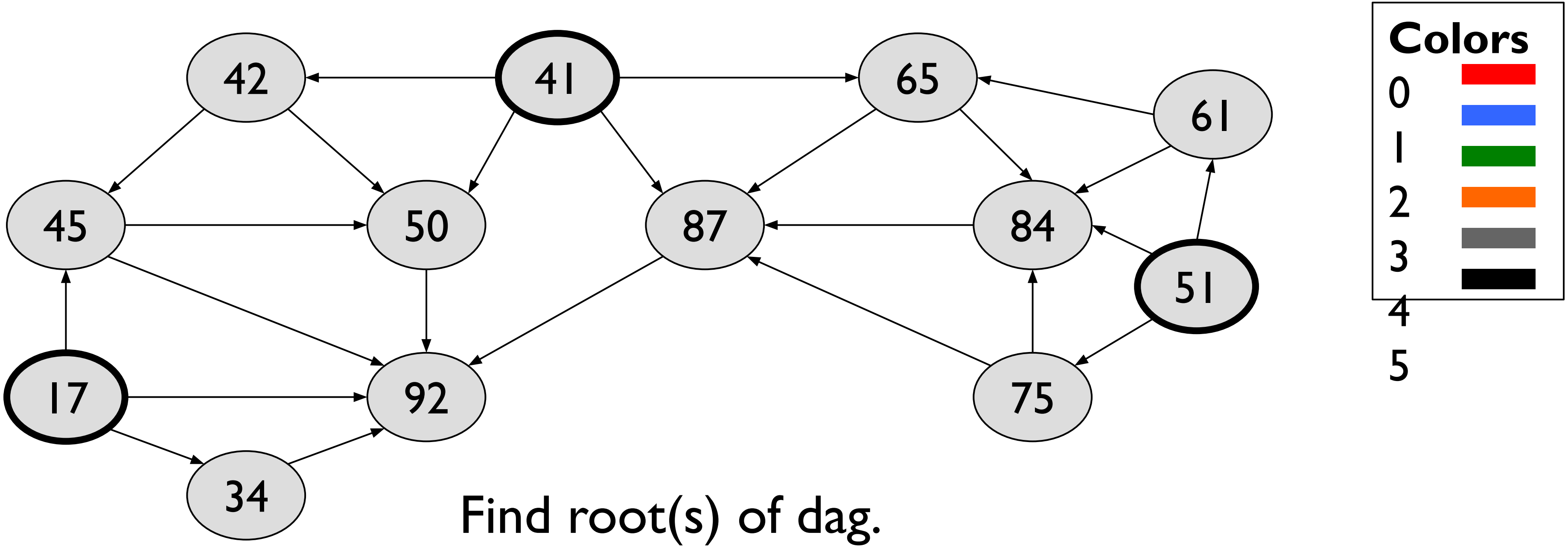
```
GET-COLOR(v)
19  C = {1, 2, ..., |v.pred| + 1}
20  parallel for u ∈ v.pred
21      C = C − {u.color}
22  return min C
```

Jones and Plassmann [35]

Line 17:

- JOIN(u.counter) checks if u's predecessors have been colored

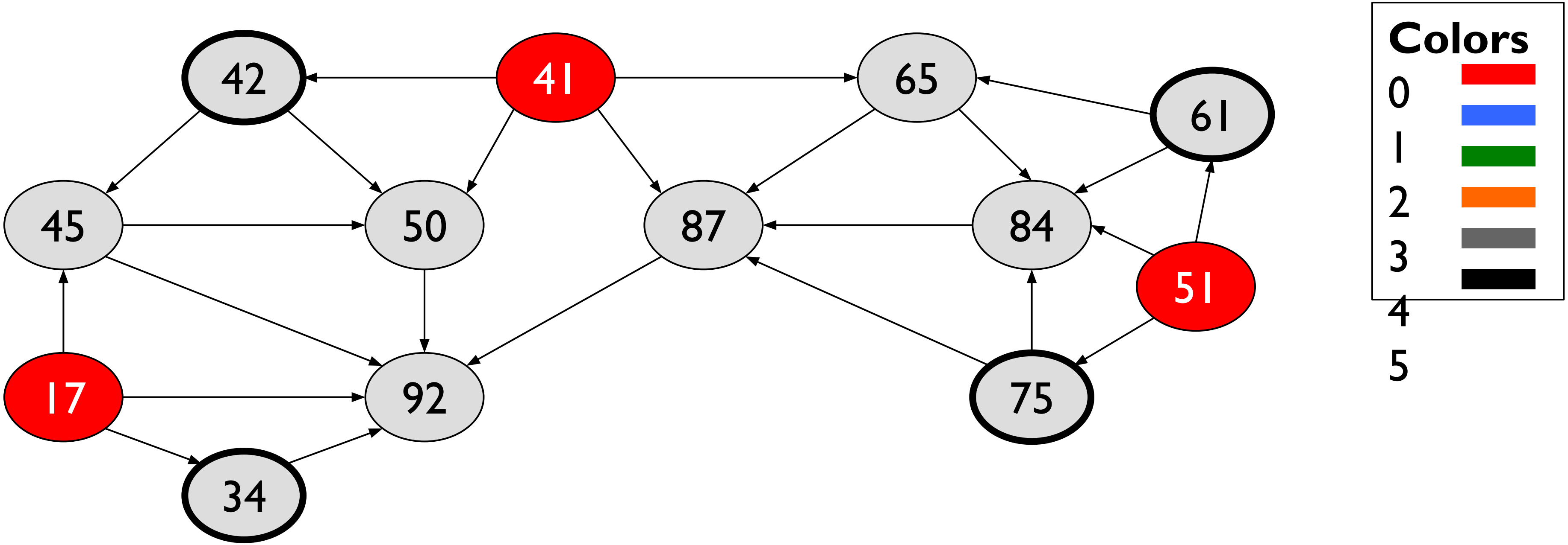
Example: Jones-Plassmann



Find root(s) of dag.

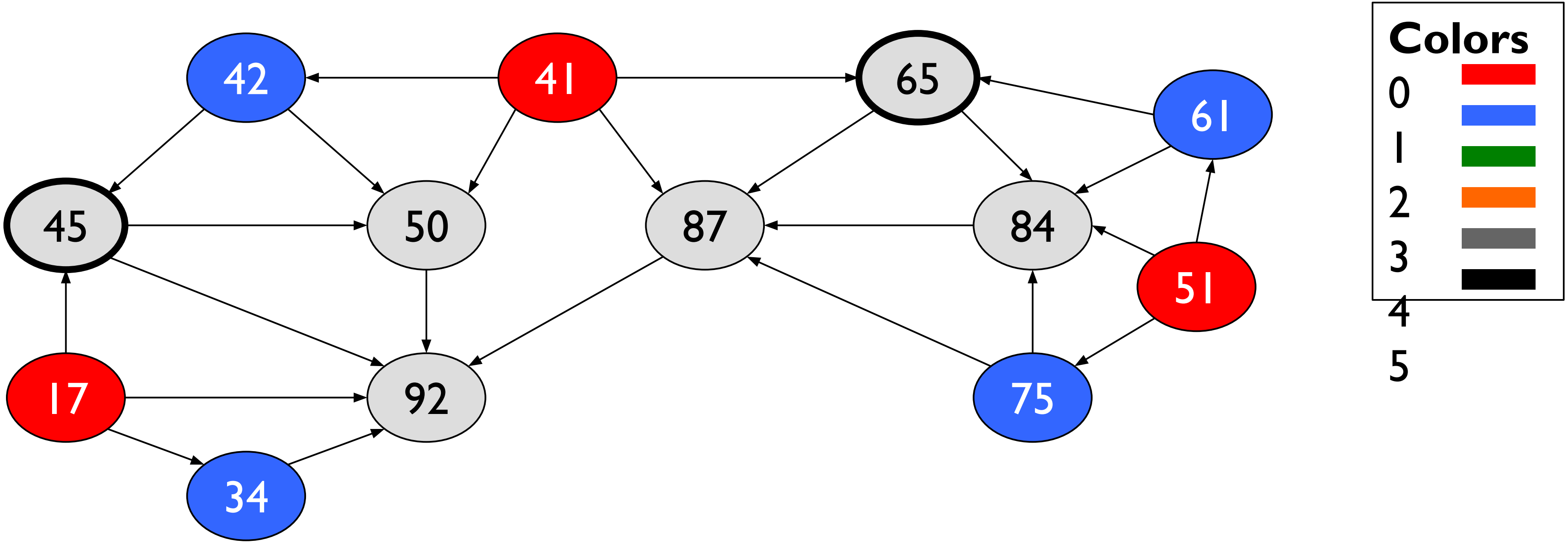
Jones and Plassmann - SIAM J. Scientific Computing, 1993

Example: Jones-Plassmann



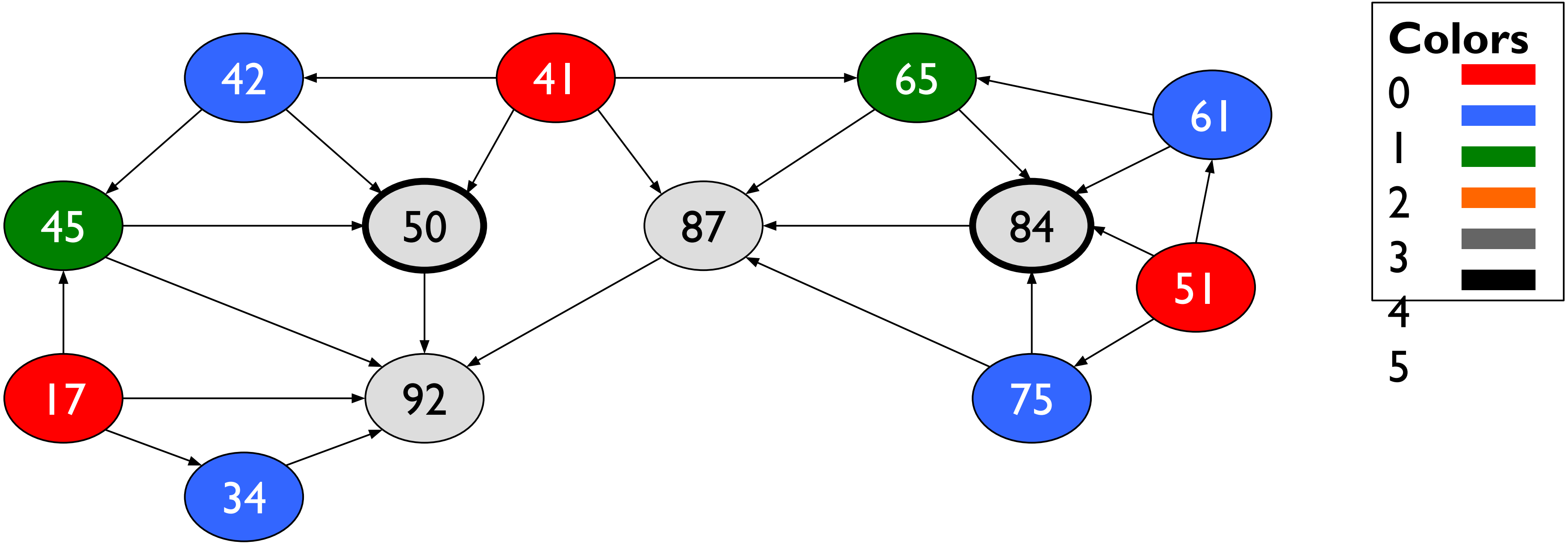
Jones and Plassmann - SIAM J. Scientific Computing, 1993

Example: Jones-Plassmann



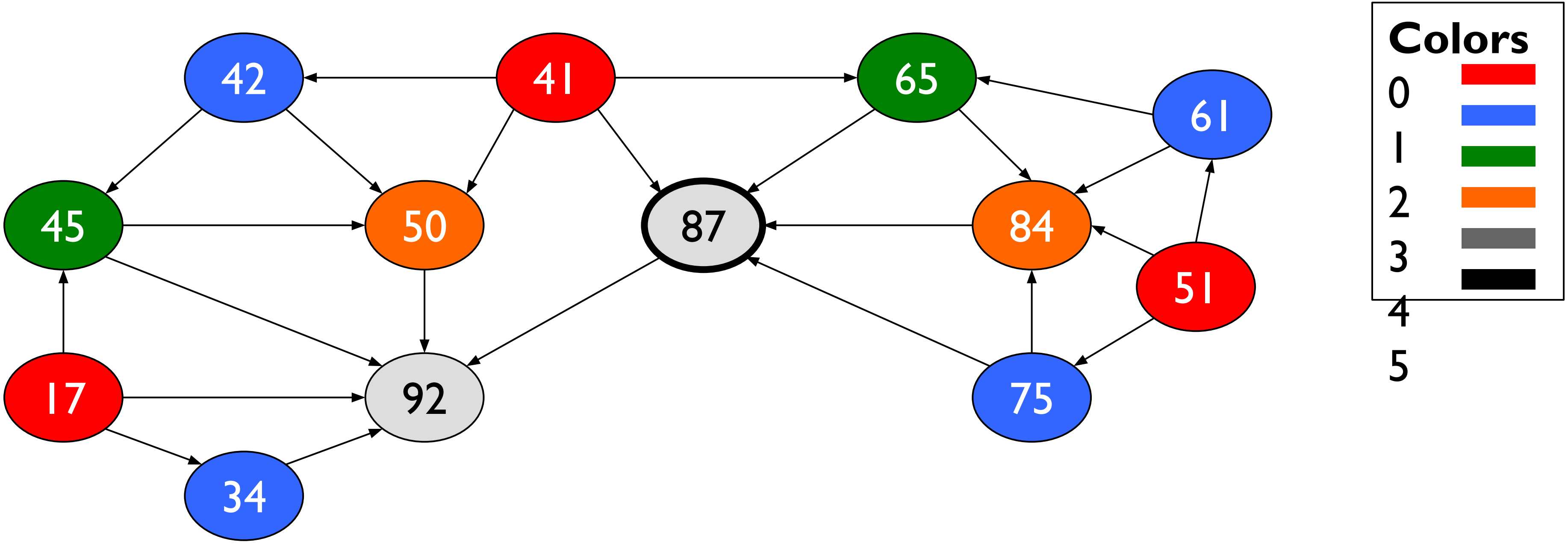
Jones and Plassmann - SIAM J. Scientific Computing, 1993

Example: Jones-Plassmann



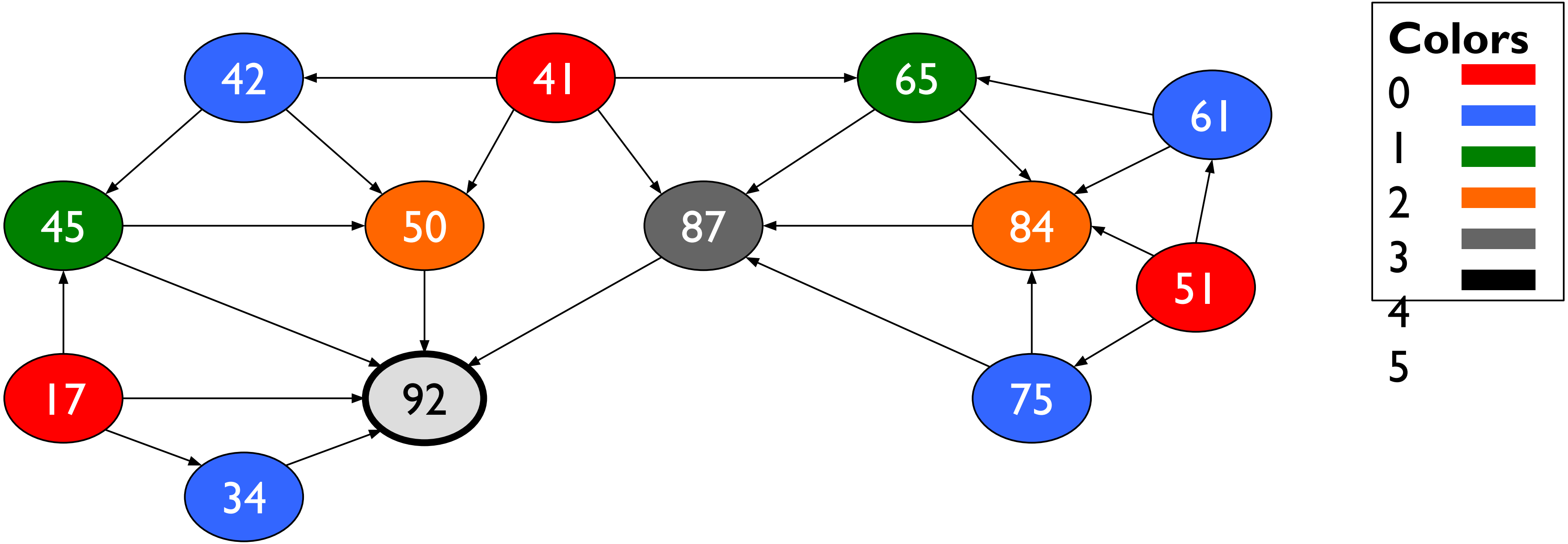
Jones and Plassmann - SIAM J. Scientific Computing, 1993

Example: Jones-Plassmann



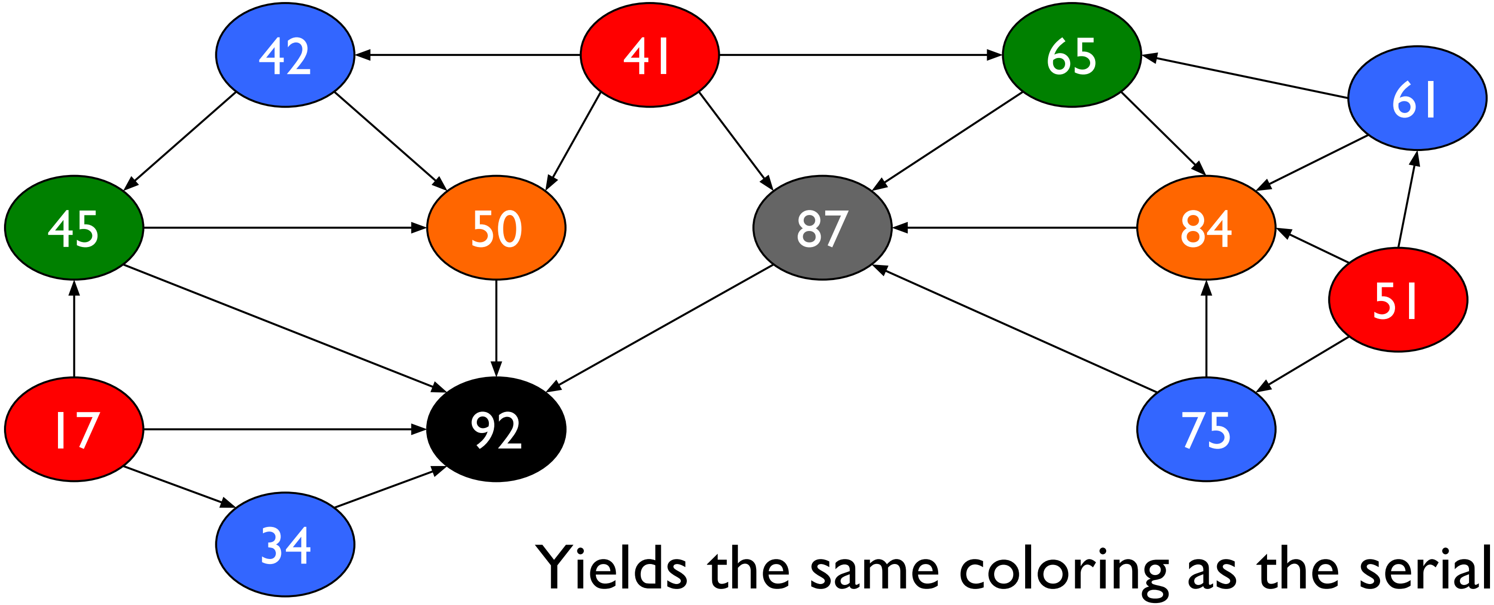
Jones and Plassmann - SIAM J. Scientific Computing, 1993

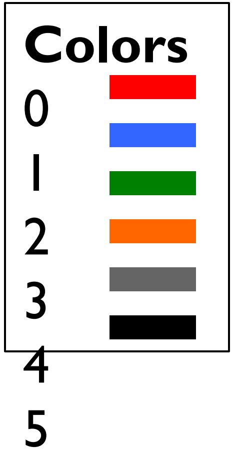
Example: Jones-Plassmann



Jones and Plassmann - SIAM J. Scientific Computing, 1993

Example: Jones-Plassmann





Yields the same coloring as the serial **Greedy** algorithm.

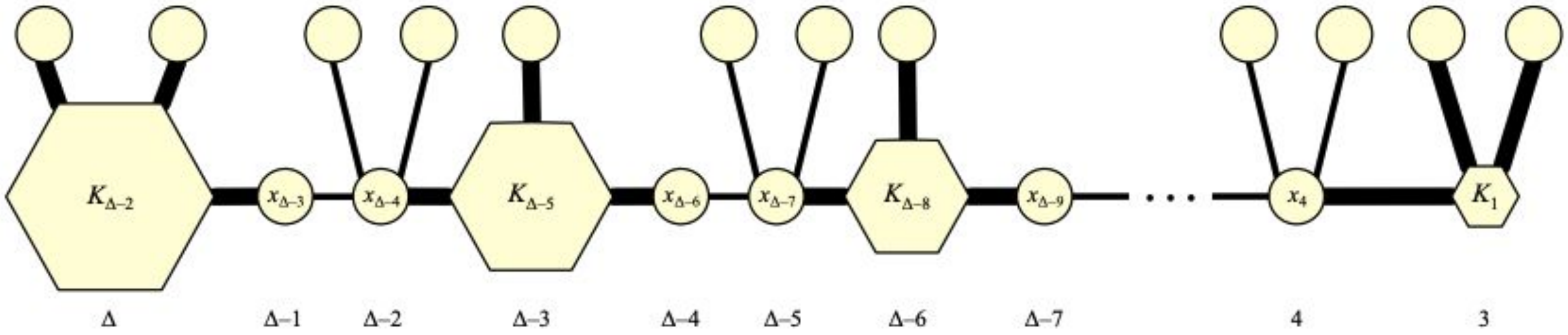
Jones and Plassmann - SIAM J. Scientific Computing, 1993

Analysis

- Linear work in size of the graph
- Traditional heuristics vulnerable to adversarial inputs causing worst case $\Omega(V)$ span
    - Why?

Adversarial Input for JP-LF

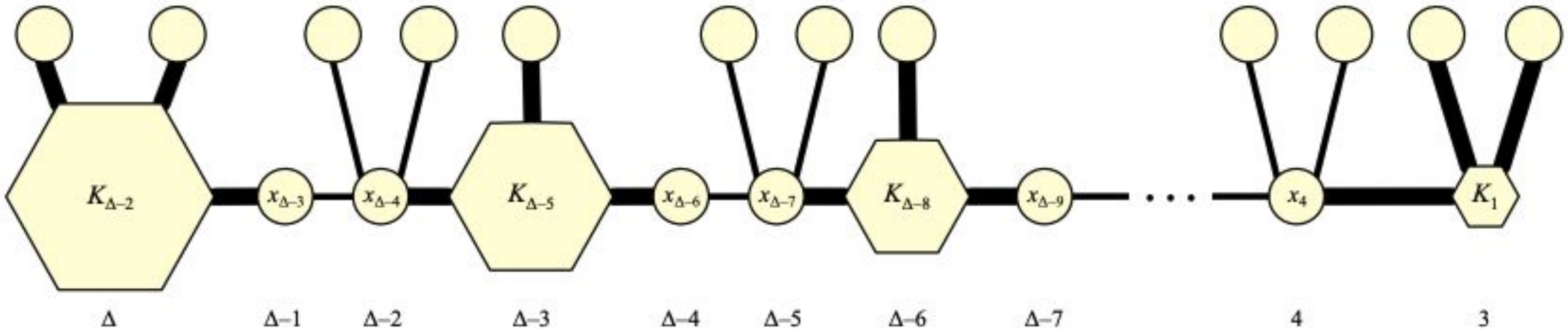
THEOREM 7. *For any* $\Delta > 0$*, there exists a* Δ*-degree graph* $G = (V,E)$ *such that* JP-LF *colors* G *in* $\Omega(\Delta^2)$ *span and* JP-R *colors* G *in* $O(\Delta \lg \Delta + \lg^2 \Delta \lg V / \lg \lg V)$ *expected span.*



LLF Ordering Heuristics

- Largest-log-degree-first
- $\rho(v) = \langle \lceil \log(deg(v)) \rceil, \rho R(v) \rangle$
- ρR is a random priority function

Clique-Chain with JP-LFF



SLL Ordering Heuristic

```
SLL-ASSIGN-PRIORITIES(G, r)
let G = (V, E)
i = 1
U = V
let Δ be the degree of G
let ρ_R ∈ R be a random priority function
for d = 0 to lg Δ
    for j = 1 to r
        Q = {u ∈ U : |u.adj ∩ U| ≤ 2^d}
        parallel for v ∈ Q
            ρ(v) = ⟨i, ρ_R(v)⟩
        U = U − Q
        i = i + 1
return ρ
```

Analysis

- JP-R, JP-LLF, JP-SLL work efficient
- Span bounds:
  - JP-R: $O(\lg V + \lg\Delta \cdot \min\{\sqrt{E}, \Delta + \lg\Delta\lg V/\lg\lg V\})$
  - JP-LLF: $O(\lg\Delta\lg V + \lg\Delta(\min\{\sqrt{E},\Delta\} + \lg 2\Delta\lg V/\lg\lg V))$
  - JP-SLL: $O(\lg\Delta\lg V + \lg\Delta(\min\{\sqrt{E},\Delta\} + \lg 2\Delta\lg V/\lg\lg V))$

Empirical Evaluation

Benchmark suite: 8 real-world graphs and 10 synthetic graphs.

Serial Heuristic	***Parallel Heuristic***	***Color Ratio***	***Efficiency***	***Speedup***
FF	R	1.011	0.417	7.039
LF	LLF	1.021	1.058	7.980
SL	SLL	1.037	1.092	6.082

Color Ratio: Ratio of the number of colors used by the parallel heuristic to the serial heuristic.

Efficiency: Ratio of serial heuristic running time to the parallel heuristic run on a single core.

Speedup: The 12-core speedup of the parallel heuristic.

“Coarse Hierarchy” In Coloring Quality

FF < R < LLF < LF < SLL < SL

Implementation Techniques

- Join trees for reducing memory contention on atomic counters
  - (Line 17)
- Bit vectors for assigning colors
  - (Line 19) Word containing adjacent colors, maintained during joins
- Software prefetching
  - (Line 16)

```
JP(G)
7  let G = (V, E, ρ)
8  parallel for v ∈ V
9      v.pred = {u ∈ V : (u, v) ∈ E and ρ(u) > ρ(v)}
10     v.succ = {u ∈ V : (u, v) ∈ E and ρ(u) < ρ(v)}
11     v.counter = |v.pred|
12 parallel for v ∈ V
13     if v.pred == ∅
14         JP-COLOR(v)
```

```
JP-COLOR(v)
15 v.color = GET-COLOR(v)
16 parallel for u ∈ v.succ
17     if JOIN(u.counter) == 0
18         JP-COLOR(u)
```

```
GET-COLOR(v)
19 C = {1, 2, ..., |v.pred| + 1}
20 parallel for u ∈ v.pred
21     C = C − {u.color}
22 return min C
```

“Coarse Hierarchy” In Coloring Quality

FF < R < LLF < LF < SLL < SL < **SD?**

Bonus: Saturation Degree

```
GREEDY-SD(G)
let G = (V, E)
for v ∈ V
    v.adjColors = ∅
    v.adjUncolored = v.adj
    PUSHORADDKEY(v, Q[0][|v.adjUncolored|])
s = 0
while s ≥ 0
    v = POPORDELKEY(Q[s][max KEYS(Q[s])])
    v.color = min({1, 2, ..., |v.adjUncolored| + 1} − v.adjColors)
    for u ∈ v.adjUncolored
        REMOVEORDELKEY(u, Q[|u.adjColors|][|u.adjUncolored|])
        u.adjColors = u.adjColors ∪ {v.color}
        u.adjUncolored = u.adjUncolored − {v}
        PUSHORADDKEY(u, Q[|u.adjColors|][|u.adjUncolored|])
        s = max{s, |u.adjColors|}
    while s ≥ 0 and Q[s] == ∅
        s = s − 1
```

- “Saturation Table” Q

THEOREM 13. GREEDY-SD *colors a graph* $G = (V, E)$ *according to the* SD *ordering heuristic in* $\Theta(V + E)$ *time.*

Bonus: Saturation Degree

```
GREEDY-SD(G)
let G = (V, E)
for v ∈ V
    v.adjColors = ∅
    v.adjUncolored = v.adj
    PUSHORADDKEY(v, Q[0][|v.adjUncolored|])
s = 0
while s ≥ 0
    v = POPORDELKEY(Q[s][max KEYS(Q[s])])
    v.color = min({1, 2, ..., |v.adjUncolored| + 1} − v.adjColors)
    for u ∈ v.adjUncolored
        REMOVEORDELKEY(u, Q[|u.adjColors|][|u.adjUncolored|])
        u.adjColors = u.adjColors ∪ {v.color}
        u.adjUncolored = u.adjUncolored − {v}
        PUSHORADDKEY(u, Q[|u.adjColors|][|u.adjUncolored|])
        s = max{s, |u.adjColors|}
    while s ≥ 0 and Q[s] == ∅
        s = s − 1
```

- “Saturation Table” Q

THEOREM 13. GREEDY-SD *colors a graph* $G = (V, E)$ *according to the* SD *ordering heuristic in* $\Theta(V + E)$ *time.*

- Ordering is determined during serial coloring. How to parallelize?

Acknowledgements

- Professor Leiserson
- Will, Tim

Results

- Overall, JP-LLF obtains a geometric-mean speedup — the ratio of the runtime on 1 core to the runtime on 12 cores — of 7.83 on the eight real-world graphs and 8.08 on the ten synthetic graphs.
- Similarly, JP-SLL obtains a geometric-mean speedup of 5.36 and 7.02 on the real-world and synthetic graphs, respectively.

Incidence Degree

- Iteratively colors an uncolored vertex with the largest number of colored neighbors

Smallest Degree Last

- First remove all lowest degree vertices
- Recursively color the new graph
- Add the removed vertices back and color

Saturation Degree

- Color an uncolored vertex whose colored neighbors use the largest number of distinct colors

Lemma 1

The helper routine GET-COLOR, shown in Figure 2, can be implemented so that during the execution of JP on a graph $G = (V,E,\rho)$, a call to GET-COLOR(v) for a vertex $v \in V$ costs $\Theta(k)$ work and $\Theta(\lg k)$ span, where $k = |v.pred|$.

Proof:

- Represent set of colors as an array
- Use sentinels to represent removed elements
    - Lines 20-21 require $\Theta(k)$ work and $\Theta(\lg k)$ span
- Implement min as a parallel reduction
    - $\Theta(k)$ work and $\Theta(\lg k)$ span
- QED

Theorem 2

Given a Δ-degree graph $G = (V,E,\rho)$ for some priority function ρ, let $G\rho$ be the priority dag induced on G by ρ, and let L be the depth of $G\rho$. Then JP(G) runs in $\Theta(V + E)$ work and $O(L\lg\Delta + \lg V)$ span

Lemma 3

The number of length-k simple paths in any Δ- degree graph G = (V,E) is at most $|V| \cdot \min\{\Delta^{k-1}, (2|E|/(k-1))^{k-1}\}$.

Lemma 4

$$g(\alpha,\beta) = e^2 \frac{\ln\alpha}{\ln\beta} \ln\left(e \frac{\beta \ln\alpha}{\alpha \ln\beta} \right) .$$

Define the function g(α,β) for α,β > 1.

Then for all β ≥ e 2 , α ≥ 2, and β ≥ α, we have g(α,β) ≥ 1.

Theorem 5

Let $G = (V,E)$ be a Δ-degree graph, let $n = |V|$ and $m = |E|$, and let $G\rho$ be a priority dag induced on G by a random priority function $\rho \in R$. For any constant $\varepsilon > 0$ and sufficiently large n, with probability at most $n{-}\varepsilon$, there exists a directed path of length $e2 \cdot \min\{\Delta, \sqrt{}\, m\} + (1 + \varepsilon)\min\{e\ 2\ \ln\Delta\ln n/\ln\ln n, \ln n\}$ in $G\rho$.

Corollary 6

COROLLARY 6. *Given a graph* $G = (V, E, \rho)$, *where* $\rho \in \mathrm{R}$ *is a random priority function, the expected depth of the priority dag* G_ρ *is* $O(\min\{\sqrt{E}, \Delta + \lg\Delta \lg V / \lg\lg V\})$, *and thus* JP-R *colors all vertices of* G *with* $O(\lg V + \lg\Delta \cdot \min\{\sqrt{E}, \Delta + \lg\Delta \lg V / \lg\lg V\})$ *expected span.*

Theorem 8

THEOREM 8. *There exists a class of graphs such that for any* $G = (V, E, \rho)$ *in the class and for any priority function* $\rho \in$ SL, JP-SL *incurs* $\Omega(V)$ *span and* JP-R *incurs* $O(\lg V / \lg \lg V)$ *span.*

Theorem 9

THEOREM 9. *Let* $G = (V, E)$ *be a* Δ*-degree graph, and let* G_ρ *be the priority dag induced on* G *by a priority function* $\rho \in$ LLF. *The expected length of the longest directed path in* G_ρ *is* $O(\min\{\Delta, \sqrt{E}\} + \lg^2 \Delta \lg V / \lg\lg V)$.

Corollary 10

COROLLARY 10. *Given a graph* $G = (V, E, \rho)$ *for some* $\rho \in$ LLF, JP-LLF *colors all vertices in G with expected span* $O(\lg V + \lg \Delta(\min\{\sqrt{E}, \Delta\} + \lg^2 \Delta \lg V / \lg \lg V))$. □

Corollary 12

COROLLARY 12. *Given a graph* $G = (V, E, \rho)$ *for some* $\rho \in$ SLL, JP-SLL *colors all vertices in* G *with expected span* $O(\lg \Delta \lg V + \lg \Delta (\min\{\sqrt{E}, \Delta\} + \lg^2 \Delta \lg V / \lg \lg V))$.

Definition: Vertex-Coloring

- Assignment of a color to each vertex of an undirected graph G = (V, E), such that for every edge (u, v) in E, u.color != v.color