# CSE 421 Winter 2025
# Lecture 3: Running Time, BFS

Nathan Brunelle

http://www.cs.uw.edu/421

# Stable Matching Problem

**Perfect matching:** everyone is matched to precisely one person from the other group

**Stability:** self-reinforcing, i.e. no pair has incentive to defect from their assignment.

- For a matching $M$, an unmatched pair $p$-$r$ from different groups is *unstable* if $p$ and $r$ prefer each other to current partners.
- Unstable pair $p$-$r$ could each improve by ignoring the assignment.

**Stable matching:** perfect matching with no unstable pairs.

**Stable matching problem:** Given the preference lists of $n$ people from each of two groups, find a stable matching between the two groups if one exists.

favorite ↓ least favorite ↓

|  | 1st | 2nd | 3rd |
|---|---|---|---|
| X | A | B | C |
| Y | B | A | C |
| Z | A | B | C |

*Group **P** Preference Profile*

favorite ↓ least favorite ↓

|  | 1st | 2nd | 3rd |
|---|---|---|---|
| A | Y | X | Z |
| B | X | Y | Z |
| C | X | Y | Z |

*Group **R** Preference Profile*

# Propose and Reject Algorithm Example

```
Initialize each person to be free.
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p′)
        replace (p′,r) by (p,r)    //p now engaged, p′ now free
    else
        r rejects p
}
```

favorite ↓ least favorite ↓

| | 1st | 2nd | 3rd |
|---|---|---|---|
| X | A | B | C |
| Y | B | A | C |
| Z | A | B | C |

*Group **P** Preference Profile*

favorite ↓ least favorite ↓

| | 1st | 2nd | 3rd |
|---|---|---|---|
| A | Y | X | Z |
| B | X | Y | Z |
| C | X | Y | Z |

*Group **R** Preference Profile*

Tentative Matches:

| | |
|---|---|
| X | A |
| Y | B |
| Z | C |

## What if we reverse the order of $P$?

```
Initialize each person to be free.
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p′)
        replace (p′,r) by (p,r)    //p now engaged, p′ now free
    else
        r rejects p
}
```

Tentative Matches:

| | |
|---|---|
| X | A |
| Y | B |
| Z | C |

favorite ↓ … least favorite ↓

| | 1st | 2nd | 3rd |
|---|---|---|---|
| X | A | B | C |
| Y | B | A | C |
| Z | A | B | C |

*Group **P** Preference Profile*

favorite ↓ … least favorite ↓

| | 1st | 2nd | 3rd |
|---|---|---|---|
| A | Y | X | Z |
| B | X | Y | Z |
| C | X | Y | Z |

*Group **R** Preference Profile*

## What if we reverse $P$ and $R$?

```
Initialize each person to be free.
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p′)
        replace (p′,r) by (p,r)    //p now engaged, p′ now free
    else
        r rejects p
}
```

Tentative Matches:

| | |
|---|---|
| A | Y |
| B | X |
| C | Z |

favorite ↓ least favorite ↓

| | 1st | 2nd | 3rd |
|---|---|---|---|
| X | A | B | C |
| Y | B | A | C |
| Z | A | B | C |

*Group **P** Preference Profile*

favorite ↓ least favorite ↓

| | 1st | 2nd | 3rd |
|---|---|---|---|
| A | Y | X | Z |
| B | X | Y | Z |
| C | X | Y | Z |

*Group **R** Preference Profile*

## Understanding the Solution

**Q:** For a given problem input, there may be several stable matchings.
Do all executions of Gale-Shapley yield the same stable matching?
If so, which one?

**Def:** $\boldsymbol{p}$ in $\boldsymbol{P}$ and $\boldsymbol{r}$ in $\boldsymbol{R}$ are valid partners iff there is some stable matching containing $(\boldsymbol{p}, \boldsymbol{r})$

**Def:** Proposer-optimal assignment: Each proposer is matched with their **best** valid partner
(their most preferred among all of their valid partners)

**Claim:** All executions of Gale-Shapley yield a proposer-optimal assignment!

- I.e. if we pair up each proposer with its best valid partner, the resulting pairs will be the same as Gale-Shapley
  - Gale-Shapley returns $\{(p, best(p)) | p \in P\}$
- Not obvious that proposer-optimal assignment is perfect, let alone stable
- Simultaneously best for each and every proposer

# Proposer Optimality

Want to prove that these rejections never happen

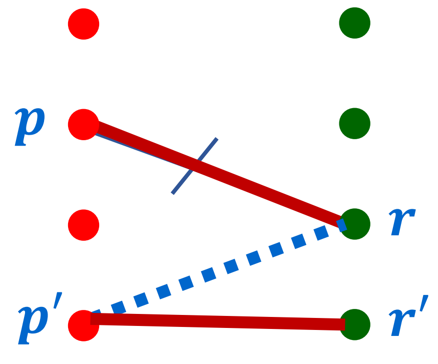
**Claim:** Any Gale-Shapley matching $\boldsymbol{M}$ is proposer-optimal.

**Proof:** (By contradiction)

Suppose that there are some proposers in $\boldsymbol{M}$ not matched to their best valid partners

$\Rightarrow$ Each must have been rejected by a valid partner, since they propose in decreasing preference order.

- Among all of these, choose the first time a proposer $\boldsymbol{p}$ is rejected by a valid partner.
- Call that rejecting valid partner $\boldsymbol{r}$. Let $\boldsymbol{p'}$ be the proposer who $\boldsymbol{r}$ prefers to $\boldsymbol{p}$ s.t either ($\boldsymbol{r}$ was tentatively paired with $\boldsymbol{p'}$) or ($\boldsymbol{p'}$ replaced $\boldsymbol{p}$) when that rejection happened.



Let $\boldsymbol{M'}$ be a stable matching containing $(\boldsymbol{p}, \boldsymbol{r})$. ← Must exist since $(\boldsymbol{p}, \boldsymbol{r})$ are valid partners

Let $\boldsymbol{r'}$ be the partner of $\boldsymbol{p'}$ in $\boldsymbol{M'}$. This $\Rightarrow$ $(\boldsymbol{p'}, \boldsymbol{r'})$ are valid partners.

- Since $\boldsymbol{r}$ rejecting $\boldsymbol{p}$ was the first rejection by a valid partner, when that happened, $\boldsymbol{r'}$ had not rejected $\boldsymbol{p'}$ since $(\boldsymbol{p'}, \boldsymbol{r'})$ are valid partners $\Rightarrow$ $\boldsymbol{p'}$ hadn't proposed to $\boldsymbol{r'}$.
- $\Rightarrow$ $\boldsymbol{p'}$ prefers $\boldsymbol{r}$ to $\boldsymbol{r'}$

But we already said that $\boldsymbol{r}$ prefers $\boldsymbol{p'}$ to $\boldsymbol{p}$.

$\Rightarrow$ $\boldsymbol{p'}$-$\boldsymbol{r}$ is unstable in $\boldsymbol{M'}$.

$\Rightarrow$ $\boldsymbol{M'}$ is not stable. Contradiction

| | |
|---|---|
| $r$ | ... $p'$ ... $p$ |
| $r'$ | |

| | |
|---|---|
| $p$ | |
| $p'$ | ... $r$ ... $r'$ |

# Non-obvious consequence of proposer optimality

- That proof worked no matter which free proposer was selected in each step!

- There is just one proposer-optimal stable matching

So all the orders of selecting free proposers in the Gale-Shapley algorithm yield the same stable matching!

## Stable Matching: Summary so far

**Stable matching problem:** Given preference profiles of two groups of $\boldsymbol{n}$ people, find a stable matching between them.

No pair of people both prefer to be with each rather than with their assigned partner

**Gale-Shapley algorithm:** Finds a stable matching in $O(\boldsymbol{n}^2)$ time.

**Proposer-optimality:** In Gale-Shapley matching, each proposer gets best partner possible among all stable matchings

**Q:** Does proposer-optimality come at the expense of the other side?

# Receiver Pessimality

**Receiver-pessimal assignment:** Each receiver is gets their worst valid partner.

**Claim:** Gale-Shapley produces a receiver-pessimal stable matching $M$.

**Proof:** (By contradiction again)

Suppose $(p, r)$ matched in $M$, but $p$ is not worst valid partner for $r$.

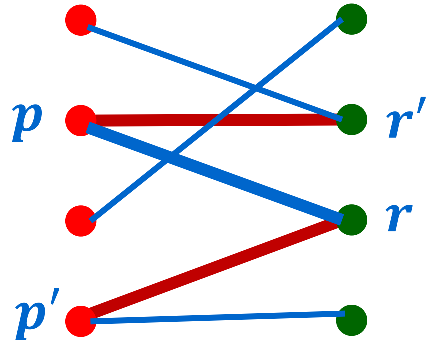
$\Rightarrow$ there exists some other stable matching $M'$ in which $r$ is paired with a proposer, say $p'$, whom $r$ likes less than $p$.

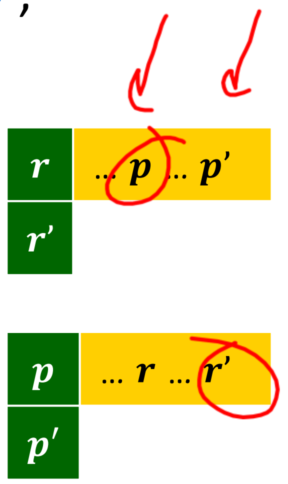
Let $r'$ be the partner of $p$ in $M'$.

Since $M$ is proposer-optimal, $p$ prefers $r$ to $r'$

$\Rightarrow$ $p$-$r$ is an unstable in $M'$

$\Rightarrow$ $M'$ is not stable. ∎





## Extensions: Matching Residents to Hospitals

**Original:** Proposers ≈ hospitals, Receivers ≈ med school residents.

**Variant 1:** Some participants declare others as unacceptable. ← e.g. resident $r$ unwilling to work in Cleveland

**Variant 2:** Unequal number of proposers and receivers.

e.g. hospital $h$ wants to hire $3$ residents

**Variant 3:** Limit on # of pairs person participates in can be >1.

**Def:** Matching $M$ is unstable if there is a hospital $h$ and resident $r$ such that:

- $h$ and $r$ are acceptable to each other; and
- either $r$ is unmatched, or $r$ prefers $h$ to her assigned hospital; and
- either $h$ does not have all its places filled, or $h$ prefers $r$ to at least one of its assigned residents.

## Application: Matching Residents to Hospitals

**NRMP:** (National Resident Matching Program)

- Original use just after WWII
- Ides of March: 23,000+ residents legally bound by the outcome
- Pre-1995 NRMP favored hospitals (they proposed)
- Changed in 1995 to favor residents (after a lawsuit)

**Rural hospital dilemma:**

- Certain hospitals (mainly in rural areas) were unpopular and declared unacceptable by many residents.
- Rural hospitals were under-subscribed in NRMP matching.
- Q: Find stable matching that benefits "rural hospitals"?

**Rural hospital theorem:** Rural hospitals get exactly same residents in every stable matching!

## The original paper

The title of the 1962 Gale-Shapley paper was “College Admissions and the Stability of Marriage”

- The propose-and-reject algorithm was clearly inspired by Western traditions of proposals
- The fact that the result is much more advantageous to the proposing side even in this non-binding scenario took some time to be appreciated

Though Gale had died by then, Shapley and Roth shared the 2012 Nobel Prize in Economic Sciences for their work on stable assignments.

## Lessons Learned

- Powerful ideas learned in course.
  - Isolate underlying structure of problem.
  - Create useful and efficient algorithms.
- Potentially deep social ramifications.
- Technique: sometimes useful to consider the first time something bad might happen for an algorithm in order to rule it out.

# Deceit: Machiavelli Meets Gale-Shapley

**Q:** Can there be an incentive to misrepresent your preference profile?

- Assuming you know that propose-and-reject algorithm will be run and who will be proposers.
- And assuming that you know the preference profiles of **all** other participants.

**Fact:** No, for proposers. Yes, for some receivers. No mechanism can guarantee a stable matching and be cheatproof.

| | 1st | 2nd | 3rd |
|---|---|---|---|
| X | A | B | C |
| Y | B | A | C |
| Z | A | B | C |

*Group **P** Preference List*

| | 1st | 2nd | 3rd |
|---|---|---|---|
| A | Y | X | Z |
| B | X | Y | Z |
| C | X | Y | Z |

*Group **R** True Preference List*

| | 1st | 2nd | 3rd |
|---|---|---|---|
| A | Y | Z | X |
| B | X | Y | Z |
| C | X | Y | Z |

*A pretends to prefer Z to X*

# Complexity analysis

- Problem size $\boldsymbol{n}$
  - **Worst-case complexity**:

    maximum # steps algorithm takes on any input of size $\boldsymbol{n}$

  - **Best-case complexity:**

    minimum # steps algorithm takes on any input of size $\boldsymbol{n}$

  - **Average-case complexity**:

    Expected # steps algorithm takes on inputs of size $\boldsymbol{n}$

## Complexity

- The complexity of an algorithm associates a number $\boldsymbol{T(n)}$, the worst/average-case/best time the algorithm takes, with each problem size **n**.

- Mathematically,
  - $\boldsymbol{T}$ is a function that maps positive integers giving problem size to positive real numbers giving number of steps.

- Sometimes we have more than one size parameter
  - e.g. $\boldsymbol{n}$=# of vertices, $\boldsymbol{m}$=# of edges in a graph.

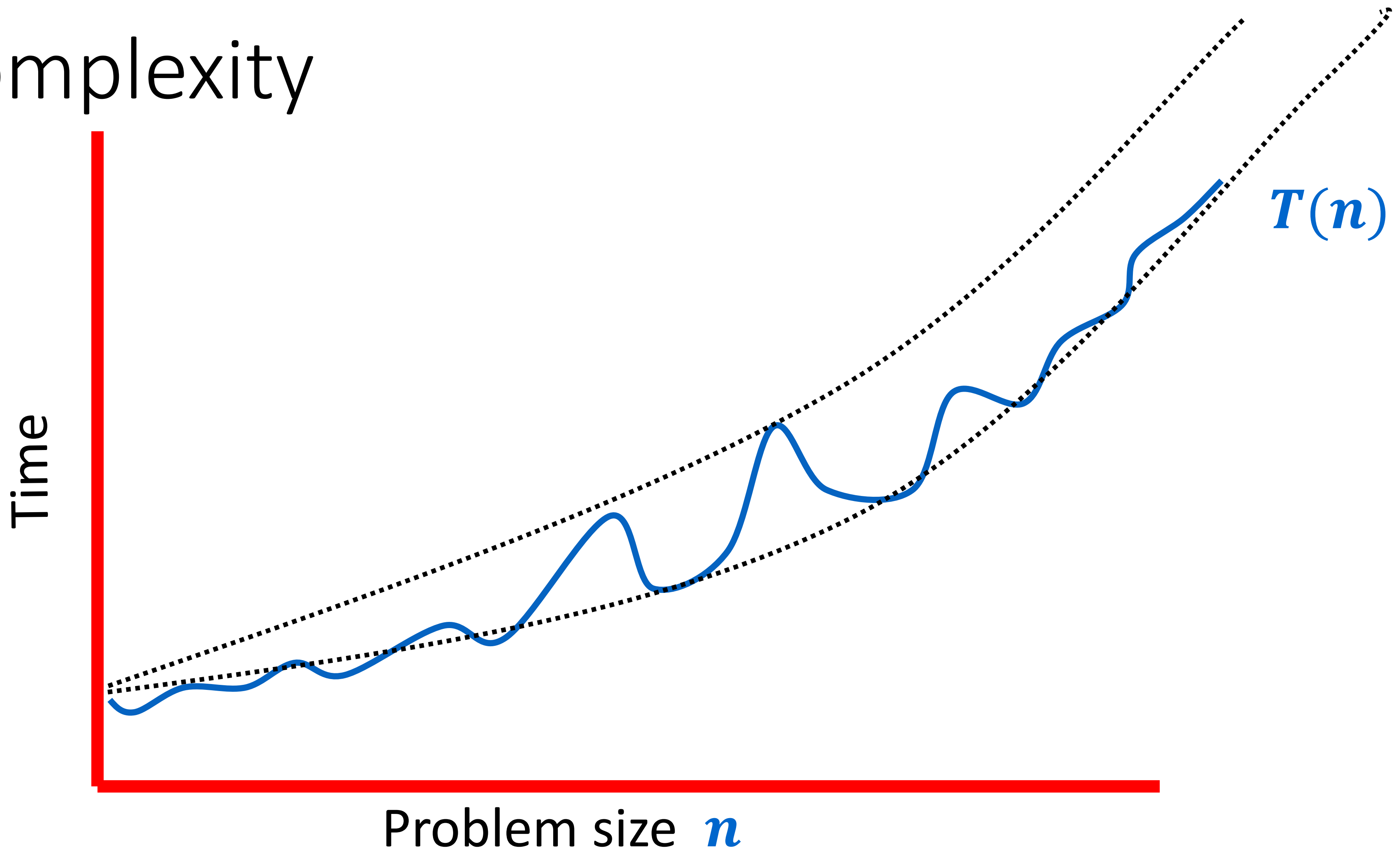
# Efficient = Polynomial Time

- Polynomial time
  - Running time $T(n) \leq cn^k + d$ for some $c, d, k \geq 0$

- Why polynomial time?
  - If problem size grows by at most a constant factor then so does the running time
    - e.g. $T(2n) \leq c\,(2n)^k + d = 2^k cn^k + d \leq 2^k\left(cn^k + d\right) = 2^k\, T(n)$
    - polynomial-time is exactly the set of running times that have this property

  - Typical running times are small degree polynomials, mostly less than $n^3$, at worst $n^6$, not $n^{100}$

## Complexity



## O-notation etc

- Given two positive functions $f$ and $g$
  - $f(n)$ is $O(g(n))$ iff there is a constant $c > 0$ so that $f(n)$ is eventually always $\leq c \cdot g(n)$
  - $f(n)$ is $o(g(n))$ iff for every constant $c > 0$ $f(n)$ is eventually always $\leq c \cdot g(n)$
  - $f(n)$ is $\Omega(g(n))$ iff there is a constant $\varepsilon > 0$ so that $f(n) \geq \varepsilon \cdot g(n)$ for infinitely many values of $n$
  - $f(n)$ is $\omega(g(n))$ iff for every constant $c > 0$ $f(n)$ is eventually always $\geq c \cdot g(n)$
  - $f(n)$ is $\Theta(g(n))$ iff $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$

Note: The definition of "$f(n)$ is $\Omega(g(n))$" is *almost* the same as "$f(n)$ is **not** $o(g(n))$"

The definition of "$f(n)$ is $\Omega(g(n))$" is *almost* the same as "$f(n)$ is **not** $o(g(n))$"

# Asymptotic Notation intuition

$f(n)$ is…

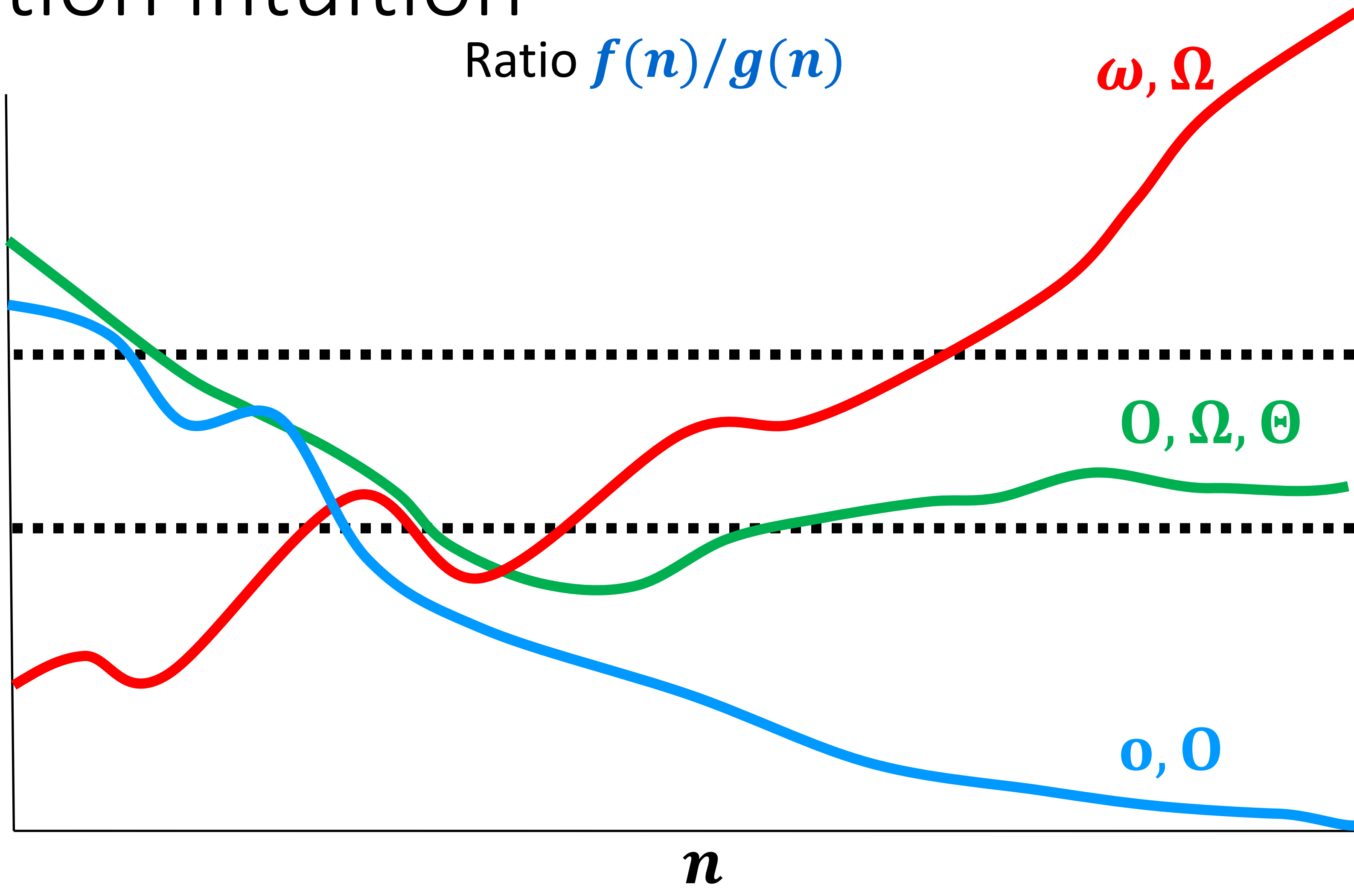
$O(g(n))$: ratio eventually below some line forever

$o(g(n))$: ratio eventually below every line forever

$\Omega(g(n))$: ratio eventually above some line forever

$\omega(g(n))$: ratio eventually above every line forever

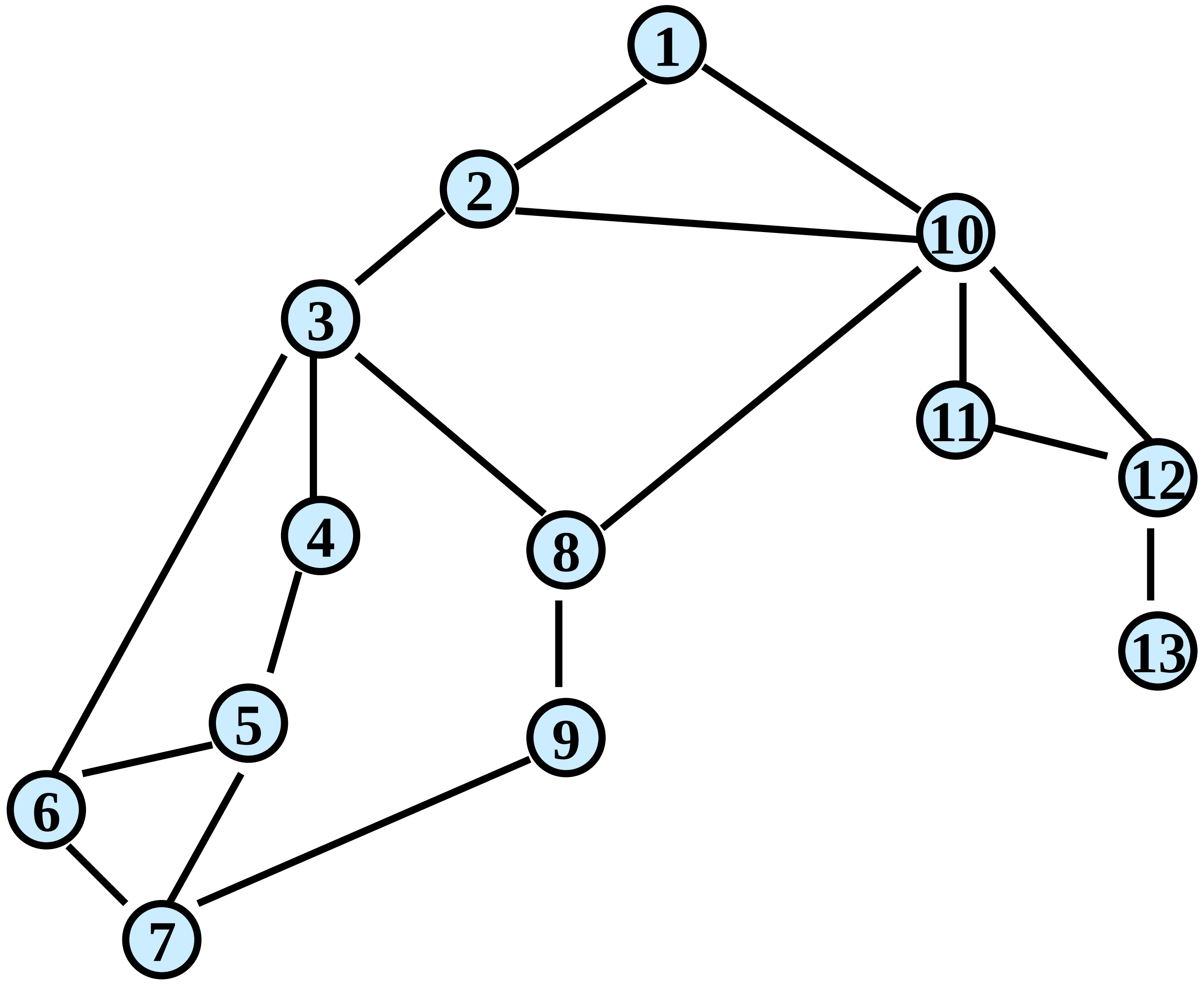
$\Theta(g(n))$: both $O$ and $\Omega$



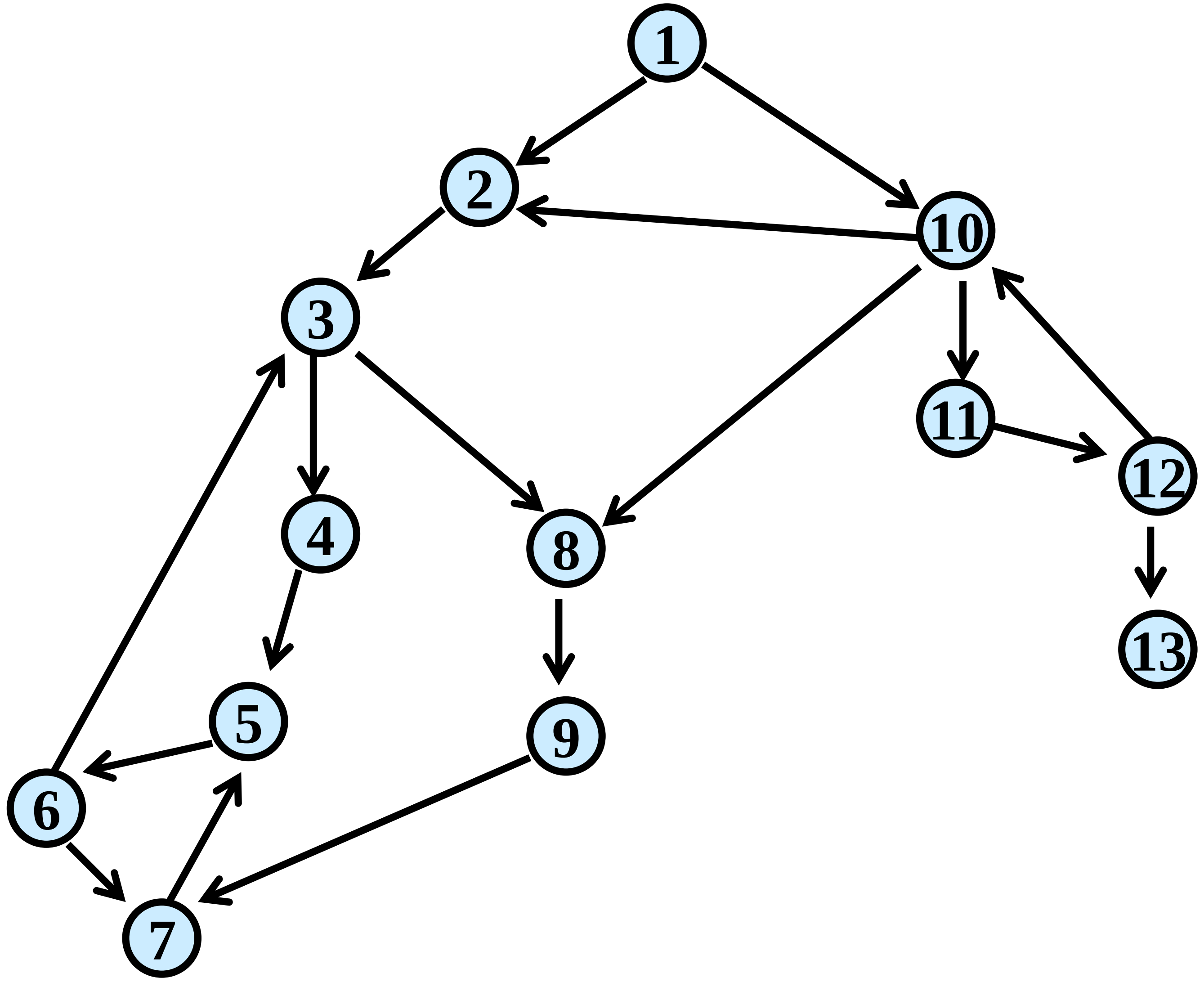
# Introduction to Algorithms

- **Graph Search/Traversal**

# Undirected Graph G = (V,E)



# Directed Graph G = (V,E)



## Graph Traversal

Learn the basic structure of a graph

Walk from a fixed starting vertex $\boldsymbol{s}$ to find all vertices reachable from $\boldsymbol{s}$

# Generic Graph Traversal Algorithm

**Given:** Graph graph $G = (V, E)$ vertex $s \in V$

**Find:** set $R$ of vertices reachable from $s \in V$

```
Reachable($s$):
    Add $s$ to $R$
    while there is a $(u, v) \in E$ where $u \in R$ and $v \notin R$
        Add $v$ to $R$
    return $R$
```

# Generic Traversal Always Works

**Claim:** At termination, $\boldsymbol{R}$ is the set of nodes reachable from $\boldsymbol{s}$

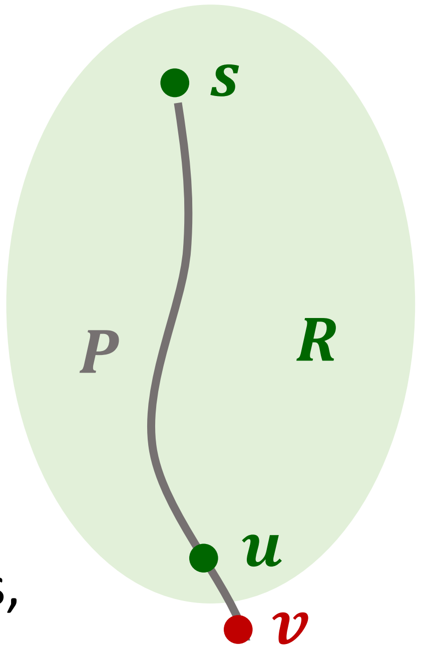
**Proof**

$\subseteq$: For every node $\boldsymbol{v} \in \boldsymbol{R}$ there is a path from $\boldsymbol{s}$ to $\boldsymbol{v}$

- Induction based on edges found.
    - Base case: $\boldsymbol{s}$ is reachable from $\boldsymbol{s}$
    - Inductive step: If there is a path to every member of $\boldsymbol{R}$ after $i$ iterations, then there is a path to every member of $\boldsymbol{R}$ after $i + 1$ iterations

$\supseteq$: Suppose there is a node $\boldsymbol{w} \notin \boldsymbol{R}$ reachable from $\boldsymbol{s}$ via a path $\boldsymbol{P}$

- Take first node $\boldsymbol{v}$ on $\boldsymbol{P}$ such that $\boldsymbol{v} \notin \boldsymbol{R}$
- Predecessor $\boldsymbol{u}$ of $\boldsymbol{v}$ in $\boldsymbol{P}$ satisfies
    - $\boldsymbol{u} \in \boldsymbol{R}$
    - $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$
- But this contradicts the fact that the algorithm exited the while loop.



## Graph Traversal

Learn the basic structure of a graph

Walk from a fixed starting vertex $s$ to find all vertices reachable from $s$

Three states of vertices

- **unvisited**
- **visited/discovered** (in $R$)
- **fully-explored** (in $R$ and all neighbors have been visited)

## Breadth-First Search

Completely explore the vertices in order of their distance from $s$

Naturally implemented using a queue

## BFS($s$)

Global initialization: mark all vertices "unvisited"

```
BFS($s$)
    Mark $s$ "visited"
    Add $s$ to $Q$
    $i$ = 0
    Mark $s$ as "layer $i$"
    while $Q$ not empty
        $u$ = next item removed from $Q$
        $i$ = "layer of $u$"
        for each edge $(u, v)$
            if ($v$ is "unvisited")
                mark $v$ "visited"
                mark $s$ as "layer $i + 1$"
        mark $u$ "fully-explored"
```

# Properties of BFS

BFS($\boldsymbol{s}$) visits $\boldsymbol{x}$ iff there is a path in $\boldsymbol{G}$ from $\boldsymbol{s}$ to $\boldsymbol{x}$.

Edges followed to undiscovered vertices define a
breadth first spanning tree of $\boldsymbol{G}$

Layer $\boldsymbol{i}$ in this tree:

$\boldsymbol{L_i}$ = set of vertices $\boldsymbol{u}$ with shortest path in $\boldsymbol{G}$ from root $\boldsymbol{s}$ of length $\boldsymbol{i}$.

# Properties of BFS

**Claim:** For undirected graphs:

All edges join vertices on the same or adjacent layers of BFS tree

**Proof:** Suppose not...

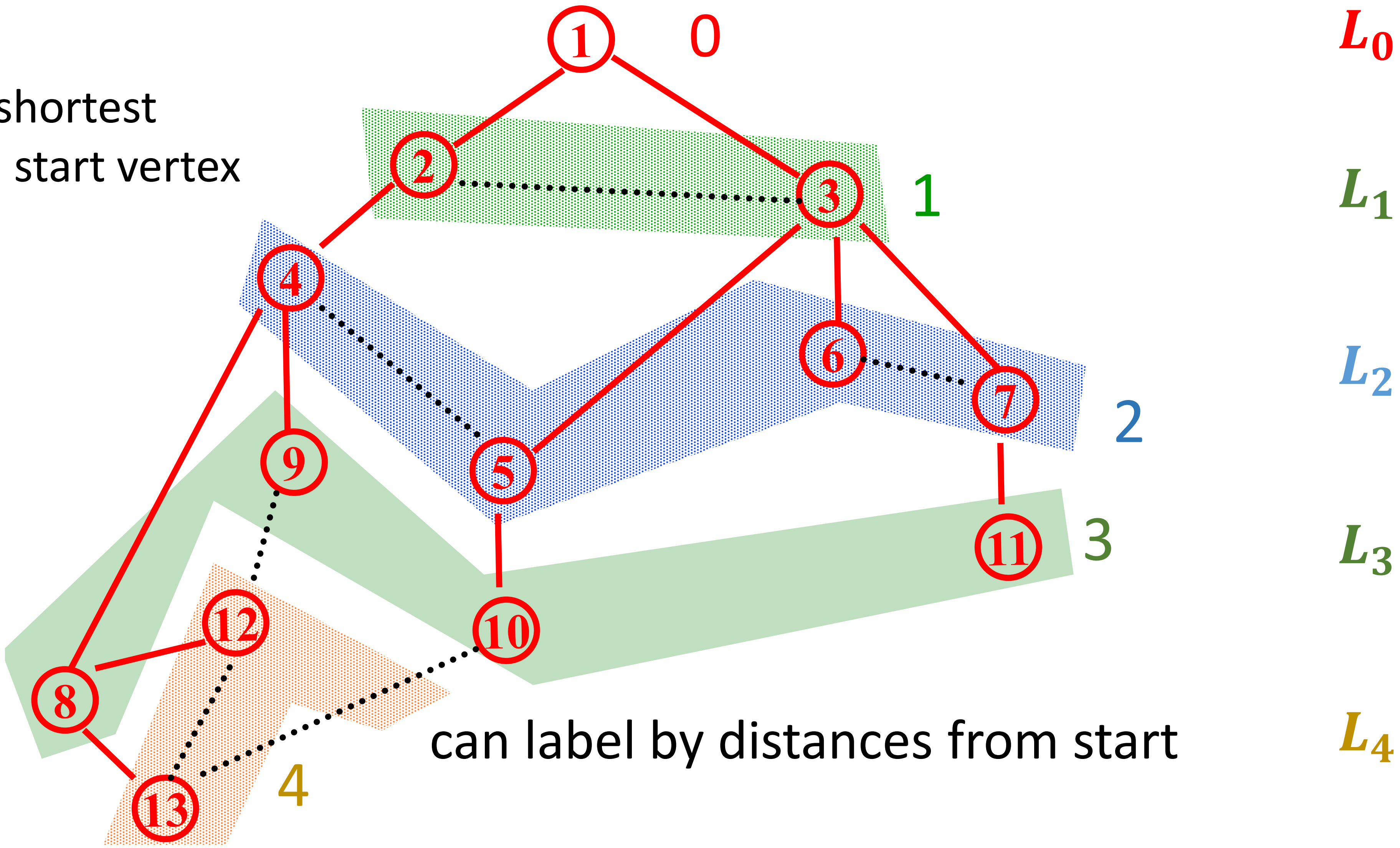
Then there would be vertices $(x, y)$ s.t. $x \in L_i$ and $y \in L_j$ and $j > i + 1$.

Then, when vertices adjacent to $x$ are considered in BFS,
$y$ would be added with layer $i + 1$ and not layer $j$.

Contradiction.

# BFS Application: Shortest Paths

Tree gives shortest paths from start vertex



can label by distances from start