Algorithms at Scale

(Week 2)

Puzzle of the Day:

A bag contains a collection of blue and red balls. Repeat:

- Take two balls from the bag.
- If they are the same color, discard them both and add a blue ball.
- If they are different colors, discard the blue ball and put the red ball back.

What do you know about the color of the final ball?

Summary

Last Week:

Toy example 1: array all 0's?

- Gap-style question:
  All 0's or far from all 0's?

Toy example 2: Faction of 1's?

- Additive $\pm\,\varepsilon$ approximation
- Hoeffding Bound

Is the graph connected?

- Gap-style question.
- O(1) time algorithm.
- Correct with probability 2/3.

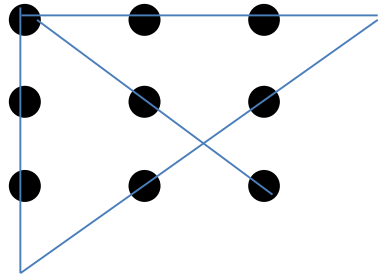
Today:

Number of connected components in a graph.

- *Additive* approximation algorithm.

Weight of MST

- *Multiplicative* approximation algorithm.



9 dots
4 lines

Announcements / Reminders

Problem sets:

Problem Set 1 was due today.

Problem Set 2 will be released tonight.

Announcements / Reminders

Next Week: Guest Lecture

Arnab Bhattacharyya

Arnab's research:
"My research area is theoretical computer science, in a broad sense. More specifically, I am interested in algorithms for big data, computational complexity, analysis and extremal combinatorics on finite fields, and algorithmic models for natural systems."

Today's Problem: Connected Components

Assumptions:

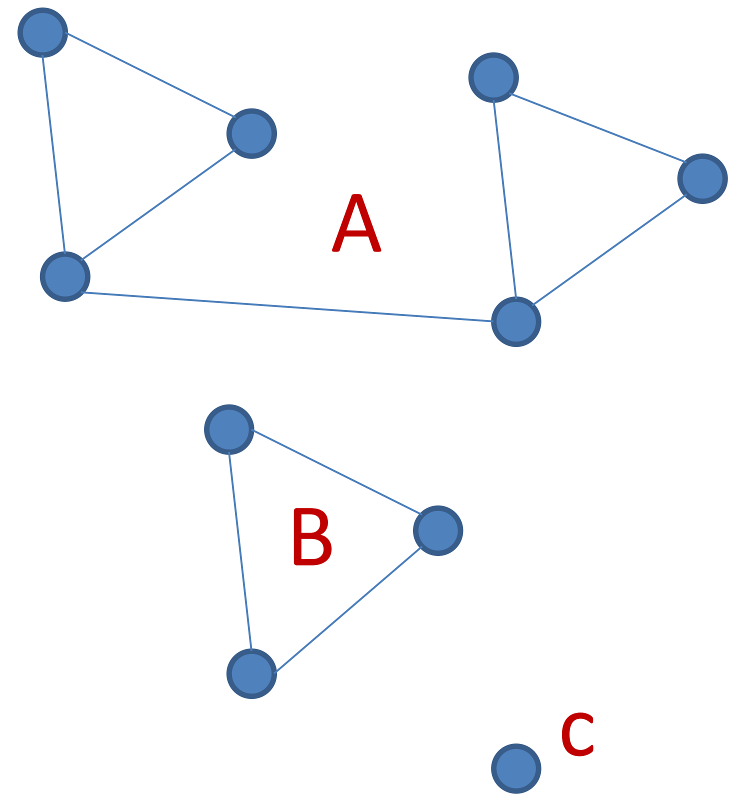
Graph $G = (V,E)$

- Undirected
- n nodes
- m edges
- maximum degree d

Error term: ε

Output:

Number of connected components.



Example: output 3

Today's Problem: Connected Components

Approximation:

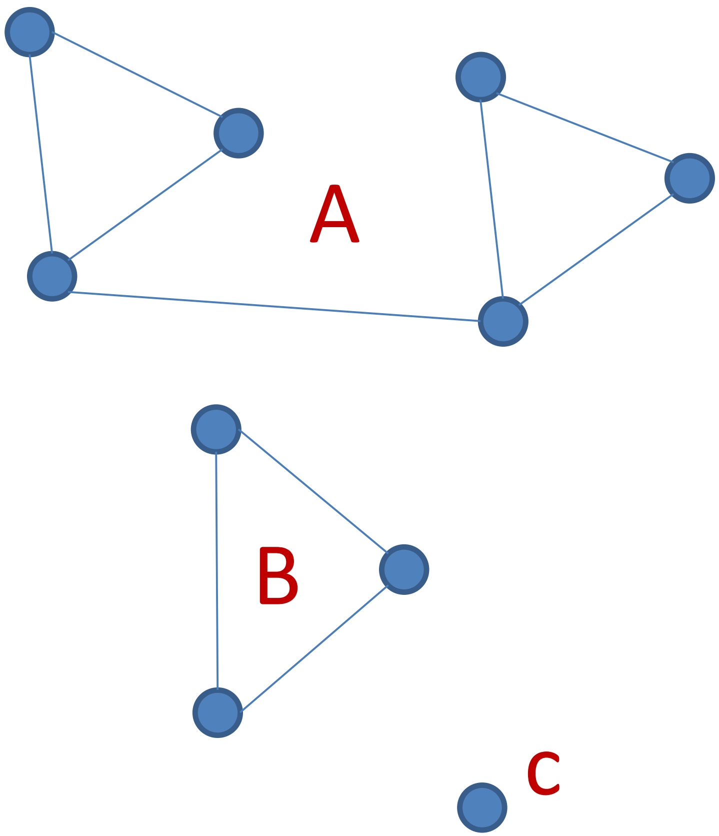
Output C such that:

$$\mathrm{CC}(G) - \epsilon n \le C \le \mathrm{CC}(G) + \epsilon n$$

Alternate form:

$$|\mathrm{CC}(G) - C| \le \epsilon n$$

Correct output: w.p. > 2/3



Example:

ε = 1/10

Output ∈ {2,3,4}

Today's Problem: Connected Components

When is this useful?

What are trivial values of ε?

What are hard values of ε?

What sort of applications is this useful for?

Approximate Connected Components

When is this useful?

What are interesting values of ε?

- What happens when $\varepsilon = 1$?
- What happens when $\varepsilon = 1/(2n)$?

What sort of applications is this useful for?

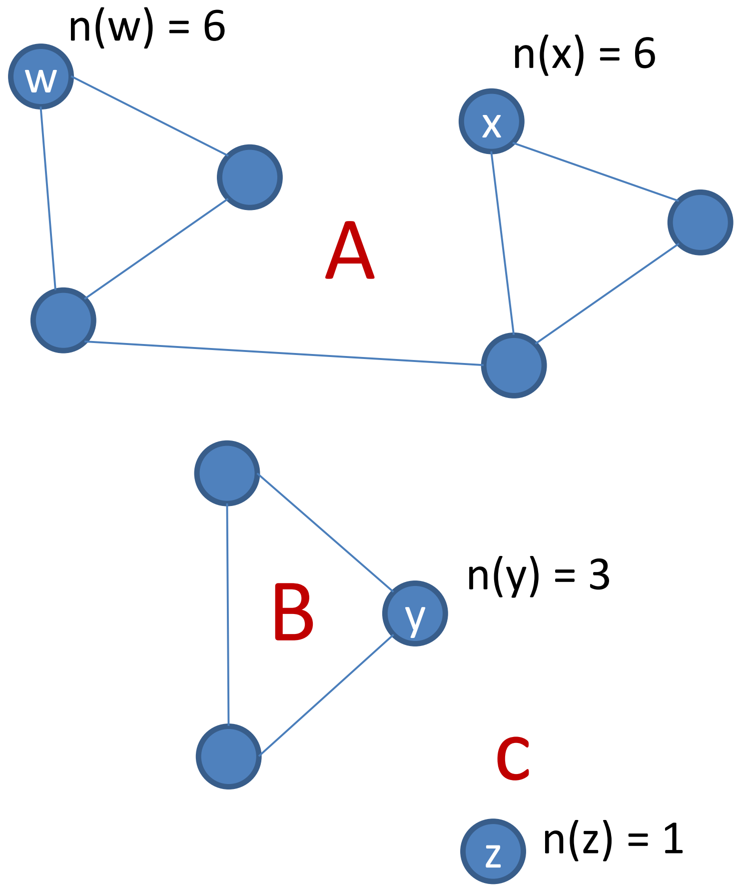
- Large graphs?
- Large social networks?
- The internet?
- Networks with many connected components?
- Number of components follows a heavy tail distribution?

Approximate Connected Components

Key Idea 1:

Define: per-node cost

Let n(u) = number of nodes in the connected component containing node u.



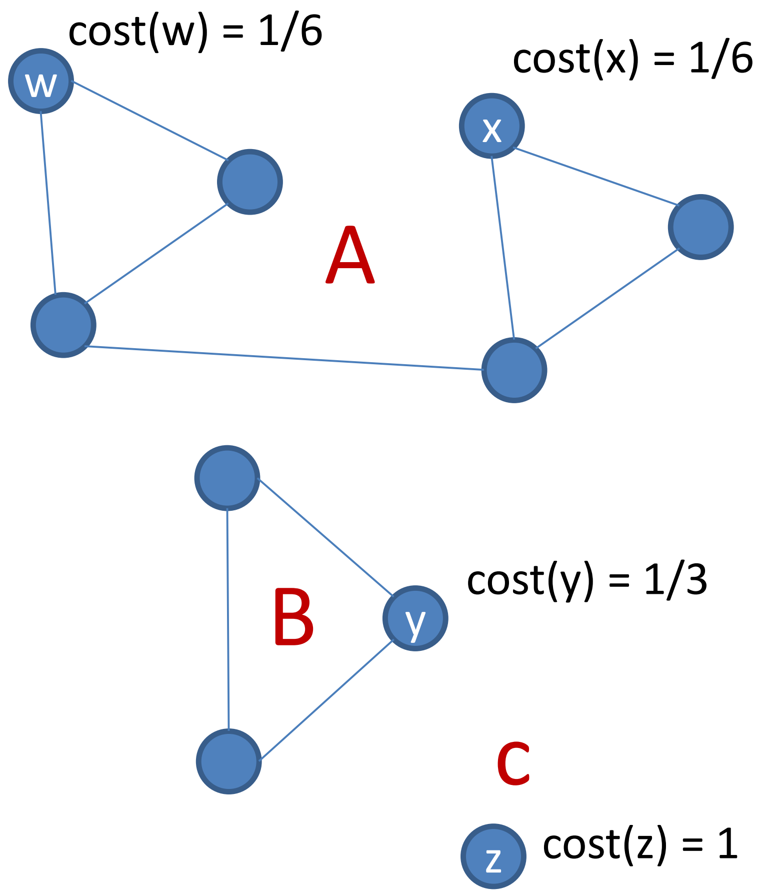
Approximate Connected Components

Key Idea 1:

Define: per-node cost

Let n(u) = number of nodes in the connected component containing node u.

Let cost(u) = 1/n(u).

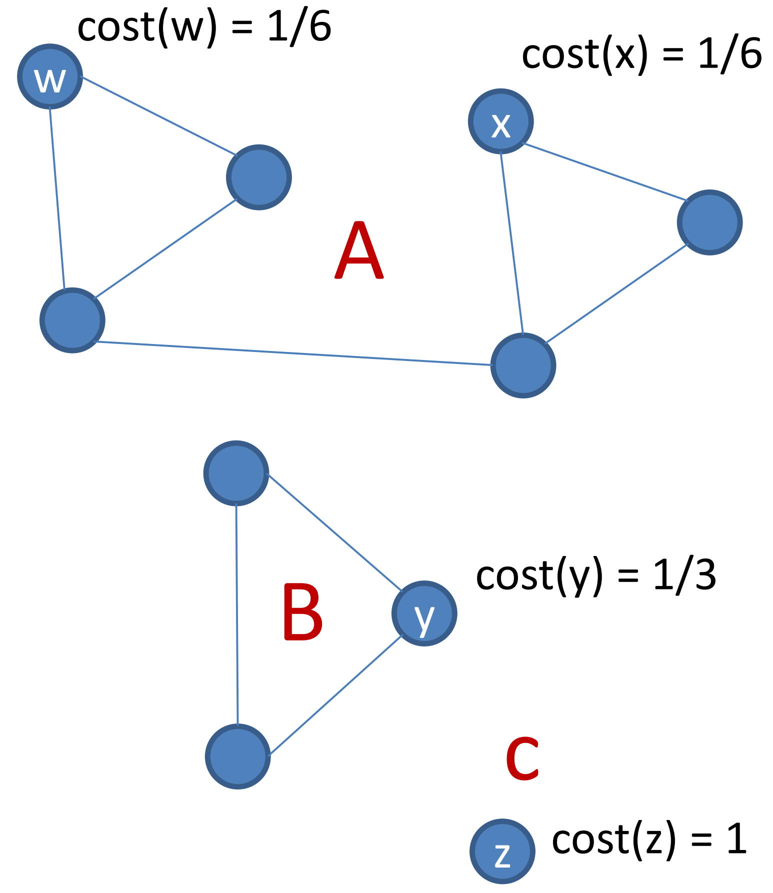


Approximate Connected Components

Key Idea 1:

Why is this useful?

$$\sum_{u \in A} \operatorname{cost}(u) = ??$$

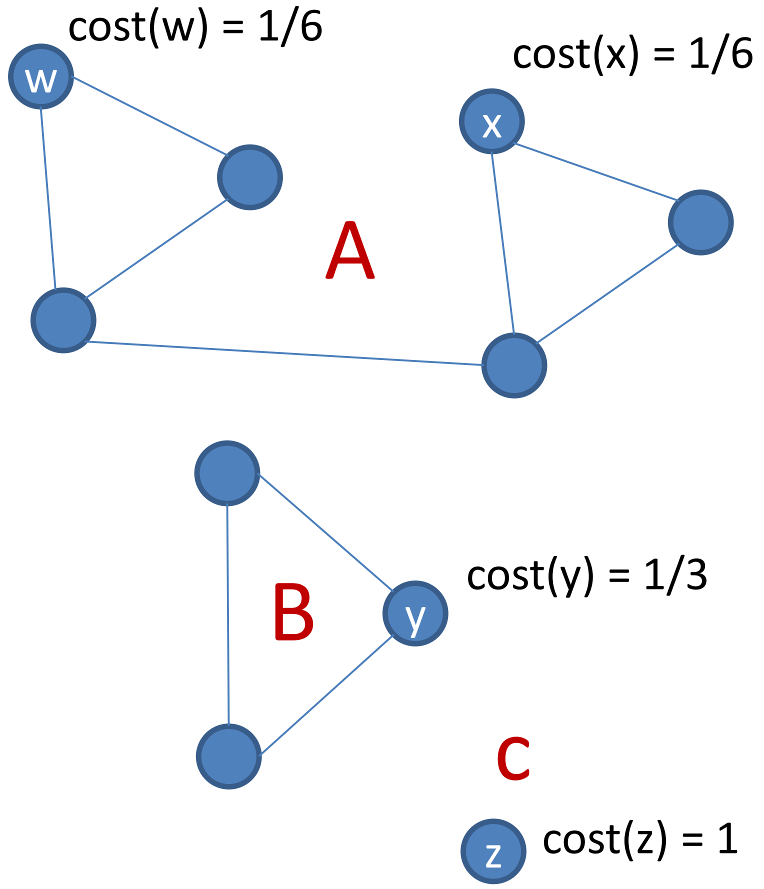


Approximate Connected Components

Key Idea 1:

Why is this useful?

$$\sum_{u \in A} \mathrm{cost}(u) = 1$$



Approximate Connected Components

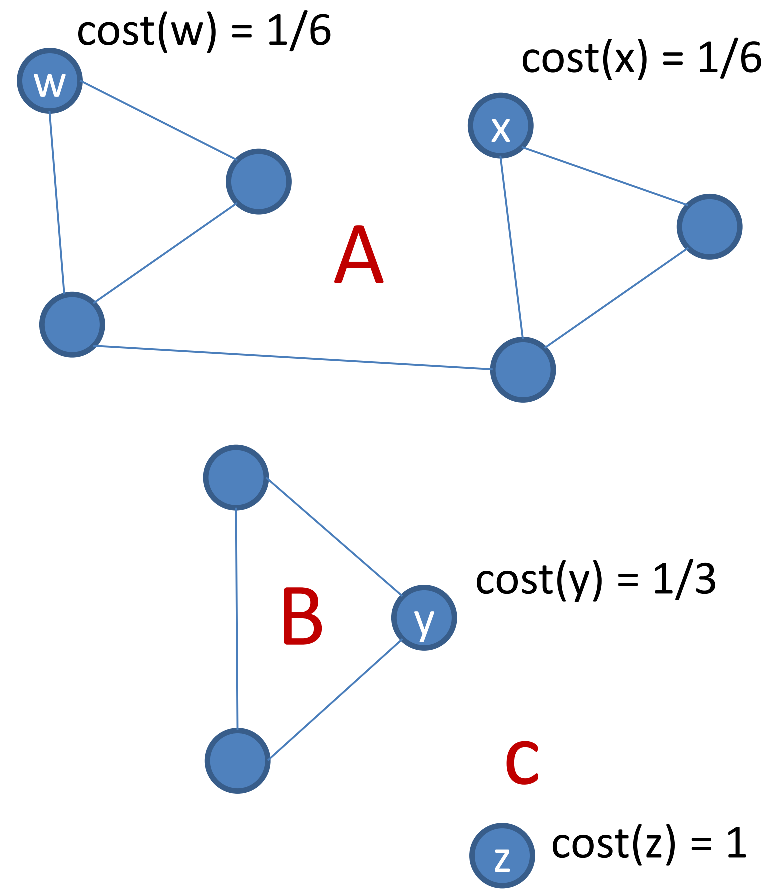
Key Idea 1:

Why is this useful?

$$\sum_{u \in A} \text{cost}(u) = 1$$

$$\sum_{u \in B} \text{cost}(u) = 1$$

$$\sum_{u \in C} \text{cost}(u) = 1$$



Approximate Connected Components

Key Idea 1:

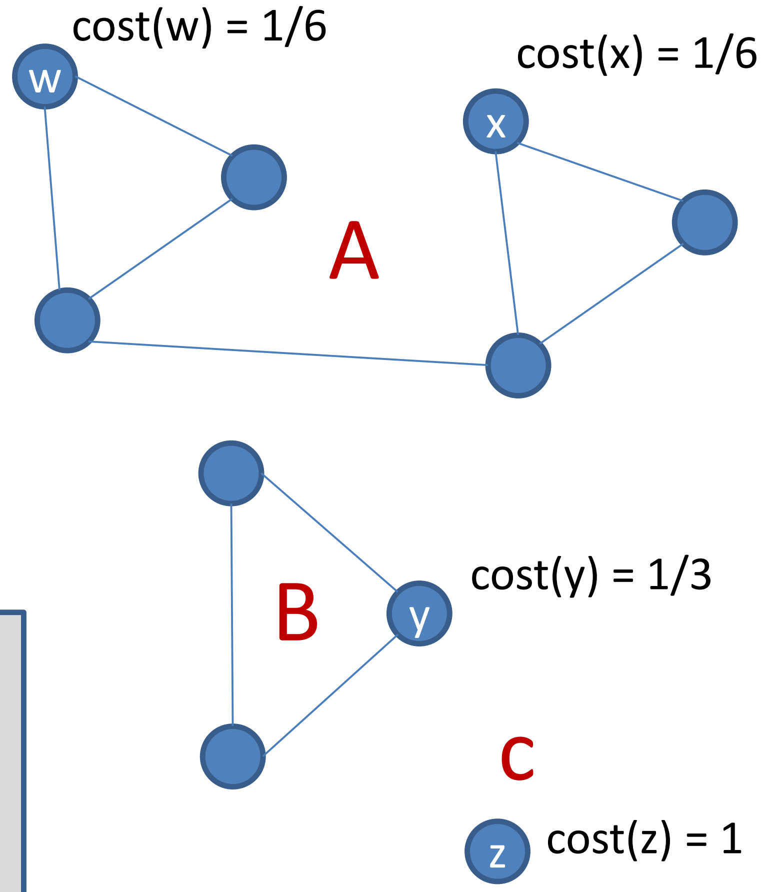
Why is this useful?

$$\sum_{u \in A} \text{cost}(u) = 1$$

$$\sum_{u \in B} \text{cost}(u) = 1$$

$$\sum_{u \in C} \text{cost}(u) = 1$$

$$\sum_{u \in V} \text{cost}(v) = \text{CC}(G)$$

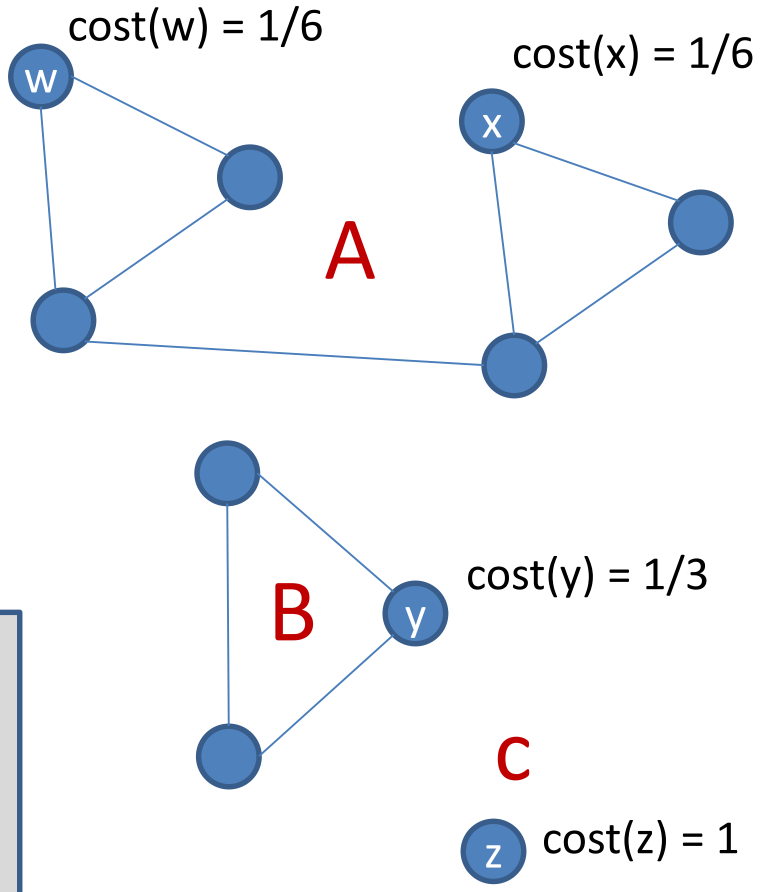


Approximate Connected Components

Algorithm 1

```
sum = 0
for each u in V:
      sum = sum + cost(u)
return sum
```

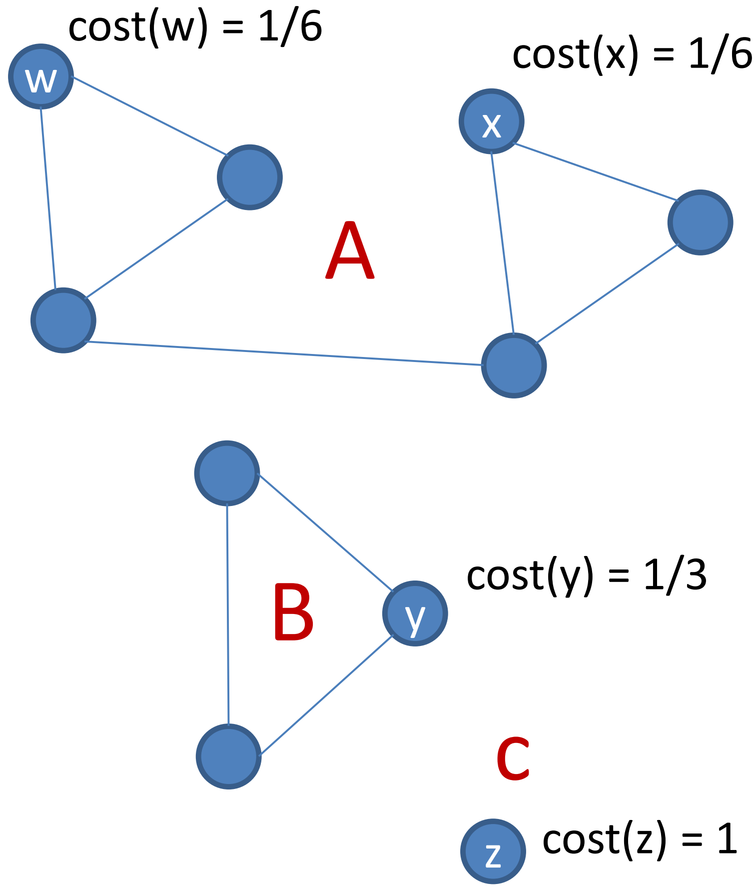
$$\sum_{u \in V} \mathrm{cost}(v) = \mathrm{CC}(G)$$



Approximate Connected Components

Algorithm 1

```
sum = 0
for each u in V:
      sum = sum + cost(u)
return sum
```



Comments:

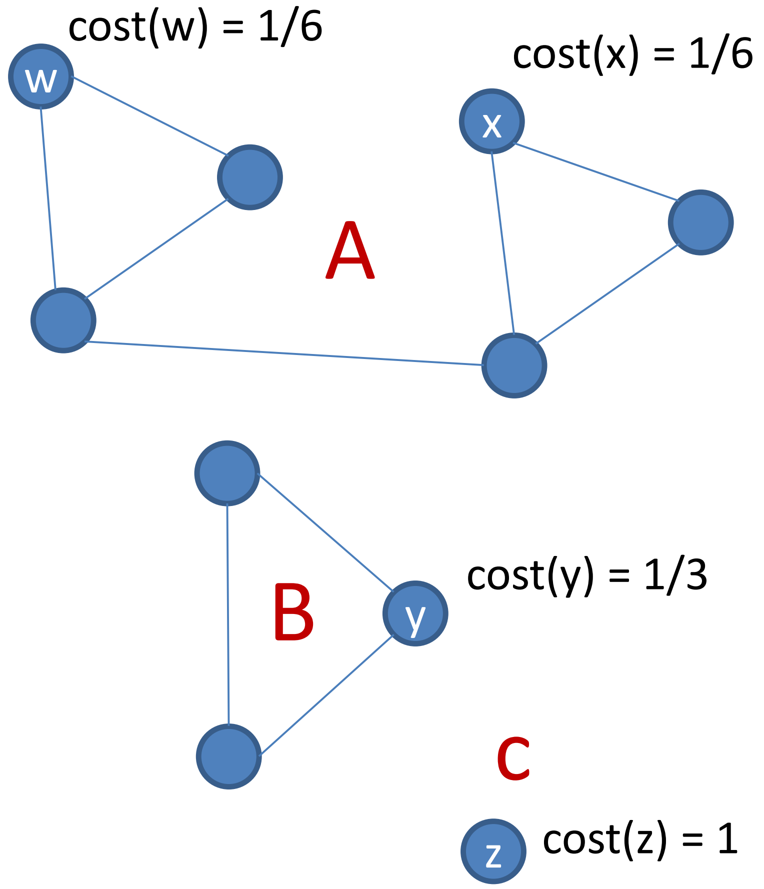
- Need a way to *efficiently* compute cost(u).
- Runs in O(n) time.

Approximate Connected Components

Key Idea 2: Sampling

Sample

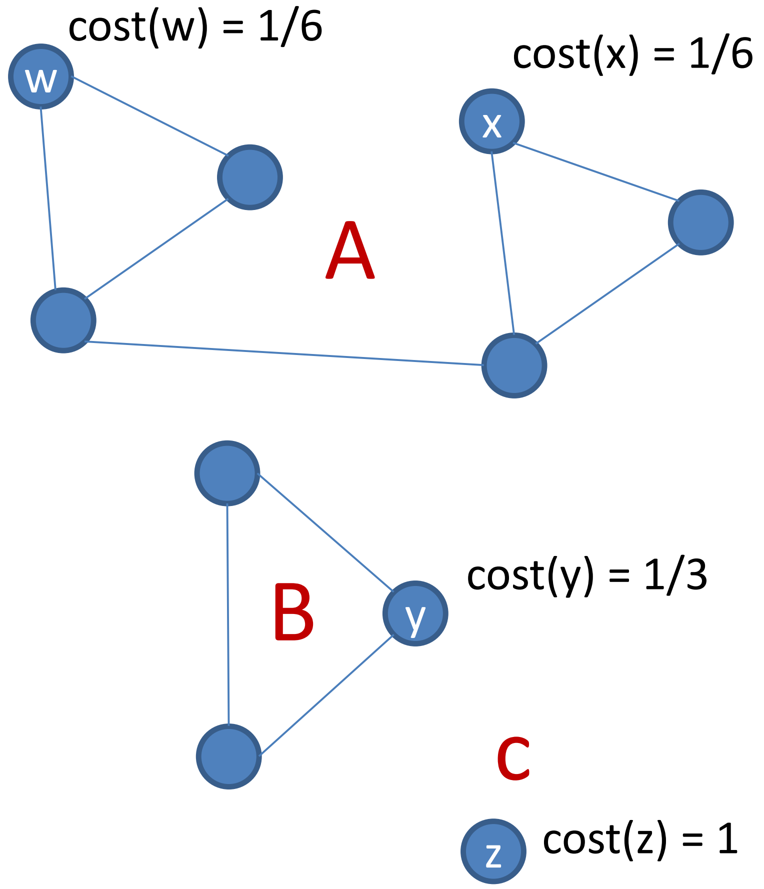
- Choose a small random subset S of V.
- For each node u in S, compute cost(u).
- Use the sample to estimate the *average* cost of all the nodes.



Approximate Connected Components

Key Idea 2: Sampling

Worries?



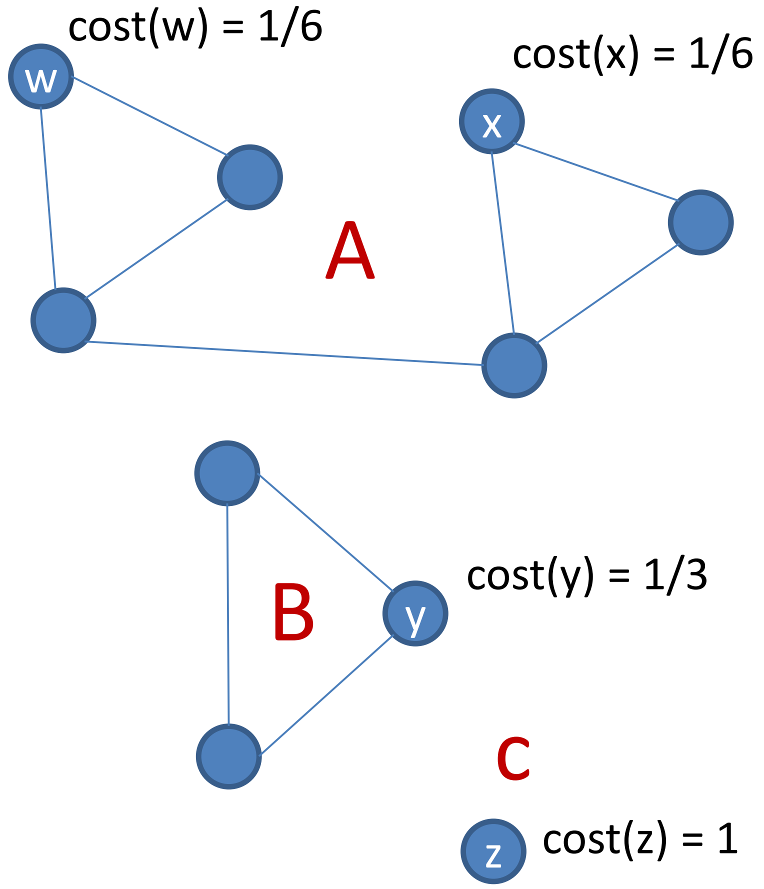
Approximate Connected Components

Key Idea 2: Sampling

Worries?

- Big components are sampled more often than small components?
- Small components may never be sampled?
- Bad examples?

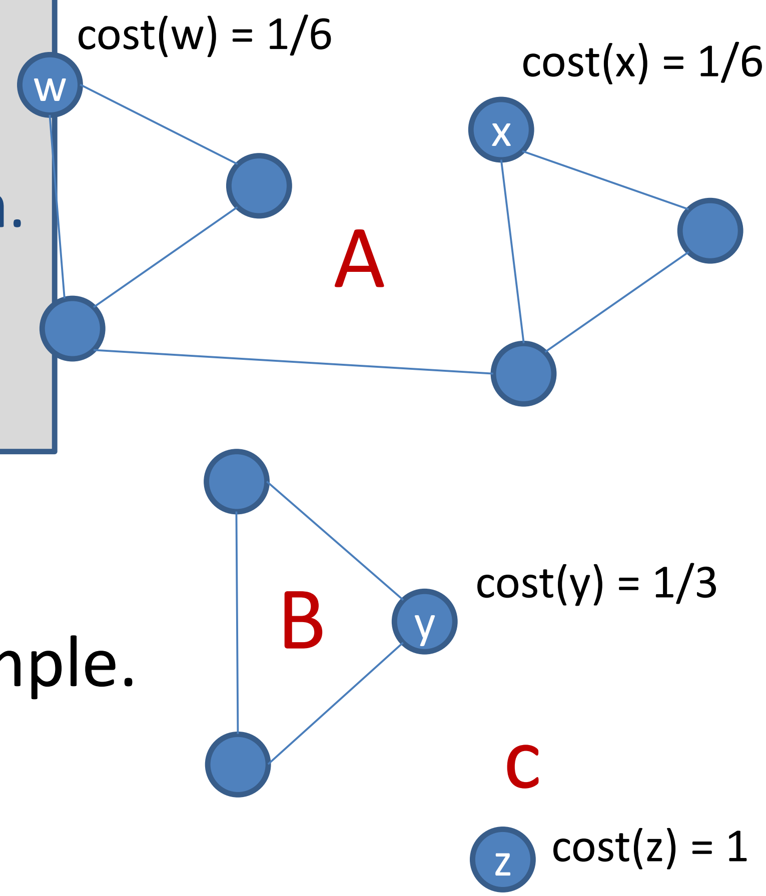
  1 component of size 90,
  10 components of size 1



Approximate Connected Components

Algorithm 2

```
sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    sum = sum + cost(u)
return n·(sum/s)
```



Comments:

- (sum/s) is average cost of sample.
- E*fficiently* compute cost(u)?
- Runs in O(s) time.

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    sum = sum + cost(u)
return n·(sum/s)
```

Define random variables: $Y_1, Y_2, \ldots, Y_s$

$$\begin{aligned} u_j &= \text{node chosen in } j^{\text{th}} \text{ iteration} \\ Y_j &= cost(u_j) \end{aligned}$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
        Choose u uniformly at random.
        sum = sum + cost(u)
return n·(sum/s)
```

$$Y_j = \mathrm{cost}(u_j)$$

$$\mathrm{E}\left[Y_j\right] = \sum_{i=1}^{n} \frac{1}{n} \mathrm{cost}(u_i)$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    sum = sum + cost(u)
return n·(sum/s)
```

$$Y_j = \mathrm{cost}(u_j)$$

$$\mathrm{E}\left[Y_j\right] = \sum_{i=1}^{n} \frac{1}{n} \mathrm{cost}(u_i) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{cost}(u_i)$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    sum = sum + cost(u)
return n·(sum/s)
```

$$Y_j = \mathrm{cost}(u_j)$$

$$\begin{aligned} \mathrm{E}\left[Y_j\right] &= \sum_{i=1}^{n} \frac{1}{n}\mathrm{cost}(u_i) = \frac{1}{n}\sum_{i=1}^{n}\mathrm{cost}(u_i) \\ &= \frac{1}{n}\mathrm{CC}(G) \end{aligned}$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
      Choose u uniformly at random.
      sum = sum + cost(u)
return n·(sum/s)
```

$$Y_j = \mathrm{cost}(u_j)$$

$$\mathrm{E}\left[Y_j\right] = \frac{1}{n}\mathrm{CC}(G)$$

$$\begin{aligned} \mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] &= s\mathrm{E}\left[Y_j\right] \\ &= \frac{s}{n}\mathrm{CC}(G) \end{aligned}$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
      Choose u uniformly at random.
      sum = sum + cost(u)
return n·(sum/s)
```

$$Y_j = \mathrm{cost}(u_j)$$

$$\mathrm{E}\left[Y_j\right] = \frac{1}{n}\mathrm{CC}(G)$$

$$\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] = \frac{s}{n}\mathrm{CC}(G)$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
      Choose u uniformly at random.
      sum = sum + cost(u)
return n·(sum/s)
```

$$Y_j = \mathrm{cost}(u_j)$$

$$\mathrm{E}\left[Y_j\right] = \frac{1}{n}\mathrm{CC}(G)$$

$$\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] = \frac{s}{n}\mathrm{CC}(G)$$

Notice:

Output of algorithm is: $\frac{n}{s}\sum_{j=1}^{s} Y_j$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
      Choose u uniformly at random.
      sum = sum + cost(u)
return n·(sum/s)
```

$$Y_j = \mathrm{cost}(u_j)$$

$$\mathrm{E}\left[Y_j\right] = \frac{1}{n}\mathrm{CC}(G)$$

$$\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] = \frac{s}{n}\mathrm{CC}(G)$$

Notice:

Expected output of algorithm is:

$$\mathrm{E}\left[n \cdot (sum/s)\right] = \frac{n}{s}\left(\frac{s}{n}\mathrm{CC}(G)\right) = \mathrm{CC}(G)$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
      Choose u uniformly at random.
      sum = sum + cost(u)
return n·(sum/s)
```

Important step:

Expected out is number of connected components!

(Algorithm is an unbiased estimator.)

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
        Choose u uniformly at random.
        sum = sum + cost(u)
return n·(sum/s)
```

Notice:

Goal:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n\right\} \leq 1/3$$

Approximate Connected Components

Algorithm 2 Analysis

```
sum = 0
for j = 1 to s:
        Choose u uniformly at random.
        sum = sum + cost(u)
return n·(sum/s)
```

Notice:

Goal:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \leq 1/3$$

Approximate Connected Components

Reminder: Hoeffding Bound

Given: independent random variables $Y_1, Y_2, \ldots, Y_s$

Assume: each $Y_j \in [0,1]$

Then:

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] - \sum_{j=1}^{s} Y_j\right| > t\right\} \le 2e^{-2t^2/s}$$

Approximate Connected Components

Reminder: Hoeffding Bound

Given: independent random variables $Y_1, Y_2, \ldots, Y_s$

Assume: each $Y_j \in [0,1]$

Then:

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] - \sum_{j=1}^{s} Y_j\right| > t\right\} \leq 2e^{-2t^2/s}$$

Goal:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \leq 1/3$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \quad =$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \quad = \quad \Pr\left\{\left|\mathrm{E}\left[\frac{n}{s}\sum_{i=1}^{s} Y_i\right] - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\}$$

$$\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] = \frac{s}{n}\mathrm{CC}(G)$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} = \Pr\left\{\left|\mathrm{E}\left[\frac{n}{s}\sum_{i=1}^{s} Y_i\right] - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\}$$

$$= \Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \frac{s}{n}\epsilon n/2\right\}$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\begin{aligned}
\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} &= \Pr\left\{\left|\mathrm{E}\left[\frac{n}{s}\sum_{i=1}^{s} Y_i\right] - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \\
&= \Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \frac{s}{n}\epsilon n/2\right\} \\
&= \Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\}
\end{aligned}$$

Approximate Connected Components

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] - \sum_{j=1}^{s} Y_j\right| > t\right\} \leq 2e^{-2t^2/s}$$

$$\begin{aligned}
\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} &= \Pr\left\{\left|\mathrm{E}\left[\frac{n}{s}\sum_{i=1}^{s} Y_i\right] - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \\
&= \Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \frac{s}{n}\epsilon n/2\right\} \\
&= \Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\}
\end{aligned}$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \quad =$$

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\} \quad \leq \quad 2e^{-2(\epsilon s/2)^2/s}$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \quad =$$

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\} \quad \leq \quad 2e^{-2(\epsilon s/2)^2/s}$$

$$\leq \quad 2e^{-2\epsilon^2 s/4}$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \quad =$$

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\} \quad \leq \quad 2e^{-2(\epsilon s/2)^2/s}$$

$$\leq \quad 2e^{-2\epsilon^2 s/4}$$

$$\boxed{s = \frac{4}{\epsilon^2}}$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} =$$

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\} \le 2e^{-2(\epsilon s/2)^2/s}$$

$$\le 2e^{-2\epsilon^2 s/4}$$

$$\le 2e^{-\epsilon^2(4/\epsilon^2)/2}$$

$$\boxed{s = \frac{4}{\epsilon^2}}$$

Approximate Connected Components

Algorithm 2 Analysis

Derivation:

$$\Pr\left\{\left|\mathrm{CC}(G) - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} =$$

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\} \leq 2e^{-2(\epsilon s/2)^2/s}$$

$$\leq 2e^{-2\epsilon^2 s/4}$$

$$\leq 2e^{-\epsilon^2(4/\epsilon^2)/2}$$

$$\leq 2e^{-2}$$

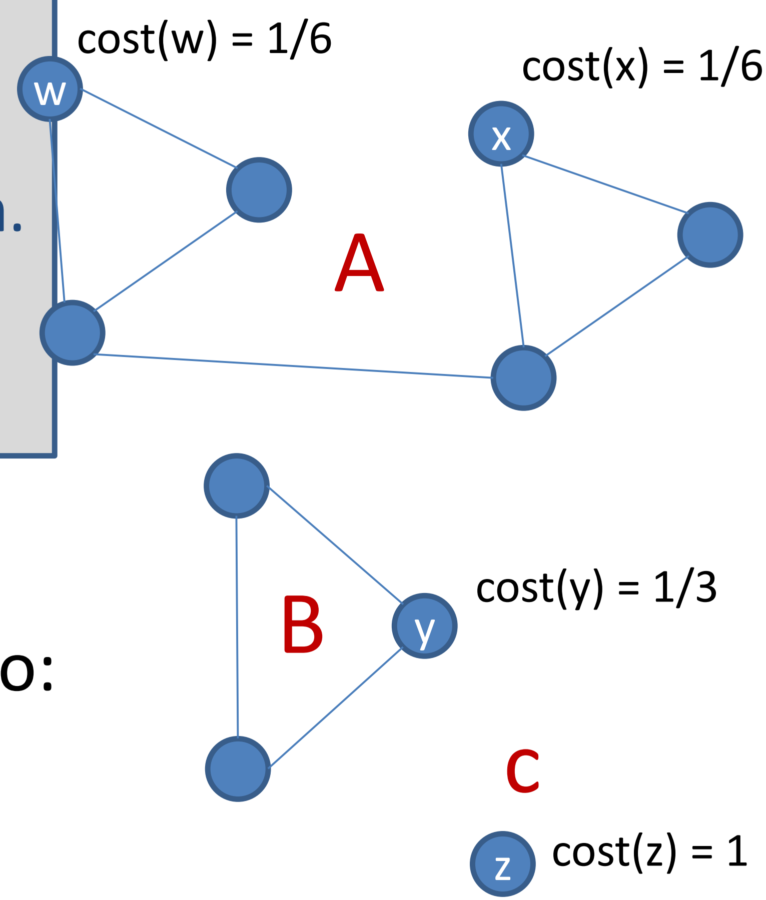
$$< 1/3$$

$$\boxed{s = \frac{4}{\epsilon^2}}$$

Approximate Connected Components

Algorithm 2

```
sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    sum = sum + cost(u)
return n·(sum/s)
```



We have shown:

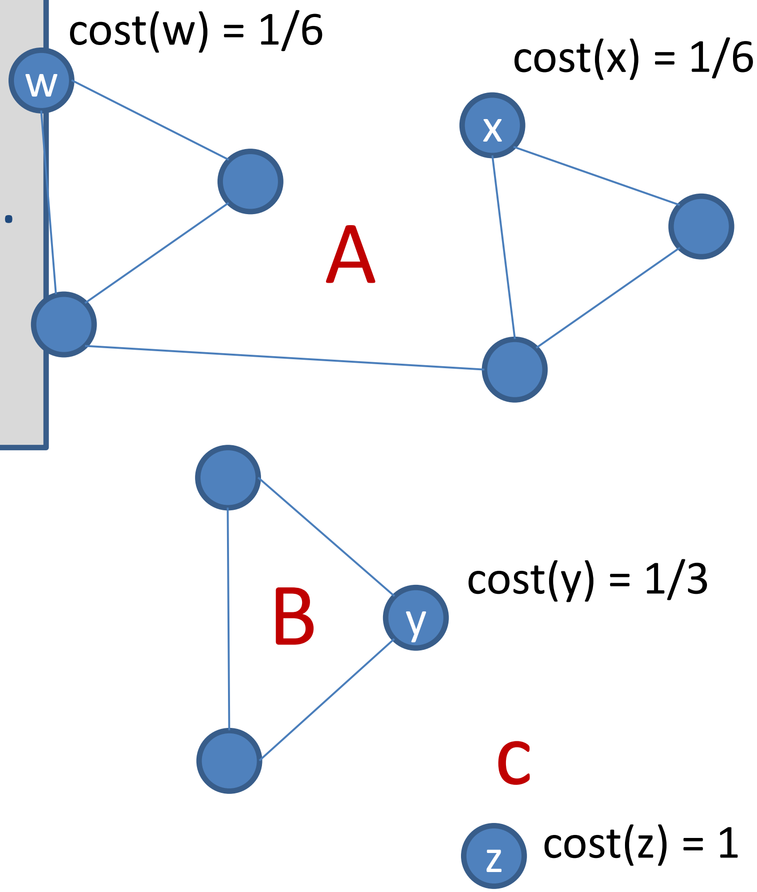
W.p. > 2/3, output is equal to:

$CC(G) \pm \varepsilon n/2$

Approximate Connected Components

Algorithm 2

```
sum = 0
for j = 1 to s:
      Choose u uniformly at random.
      sum = sum + cost(u)
return n·(sum/s)
```



We have shown:

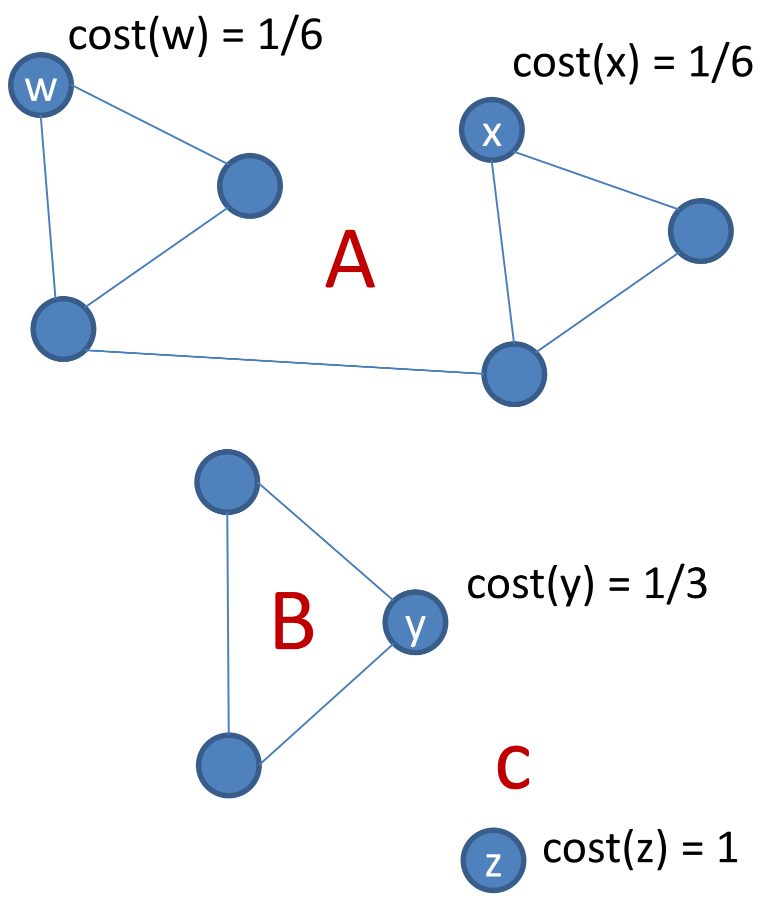
Time: $O(1/\varepsilon^2)$

Approximate Connected Components

Key Idea 2: Sampling

Key problem:

How to efficiently compute cost(u).

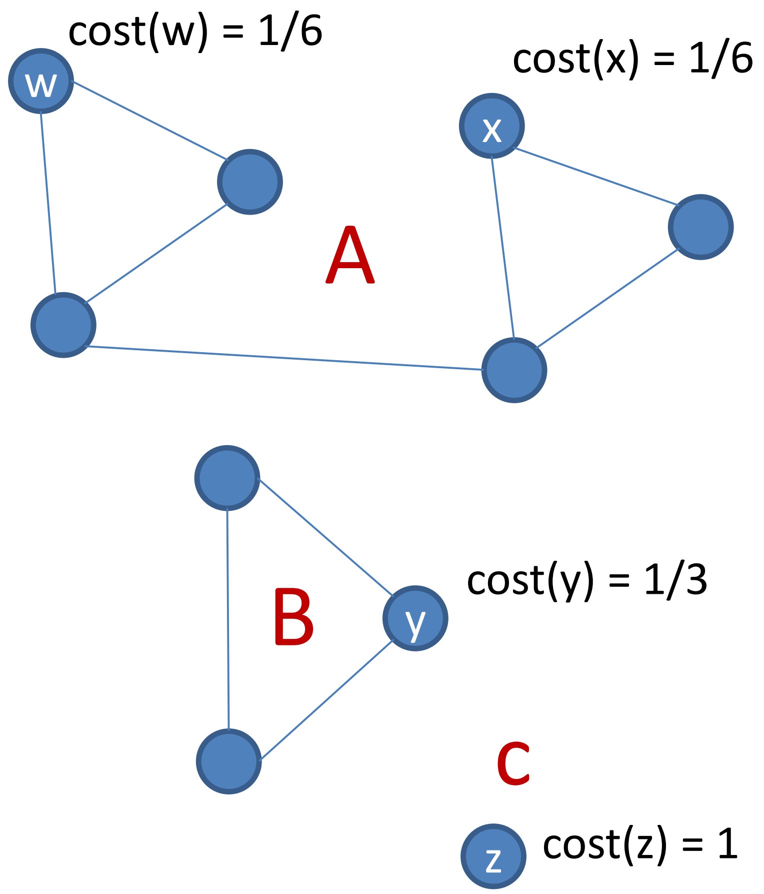


Approximate Connected Components

Key Idea 2: Sampling

Key problem:

How to efficiently compute cost(u).



Key idea 3:

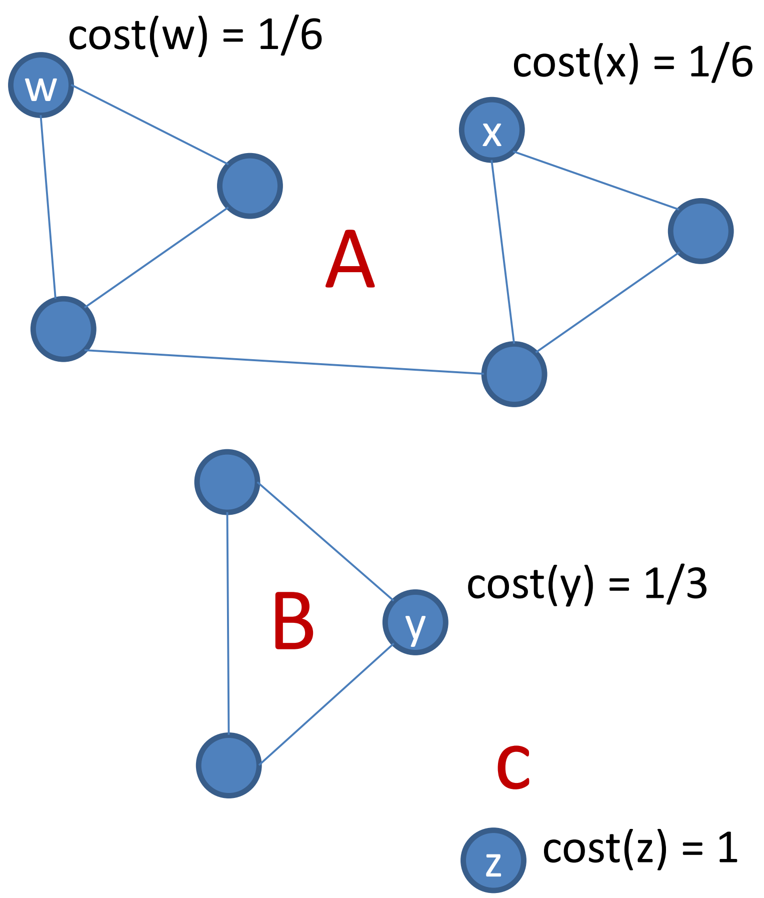
Approximate cost(u).

Approximate Connected Components

Key Idea 3: Approximate Cost

Approximate low cost components:

If cost(u) is small, round up.



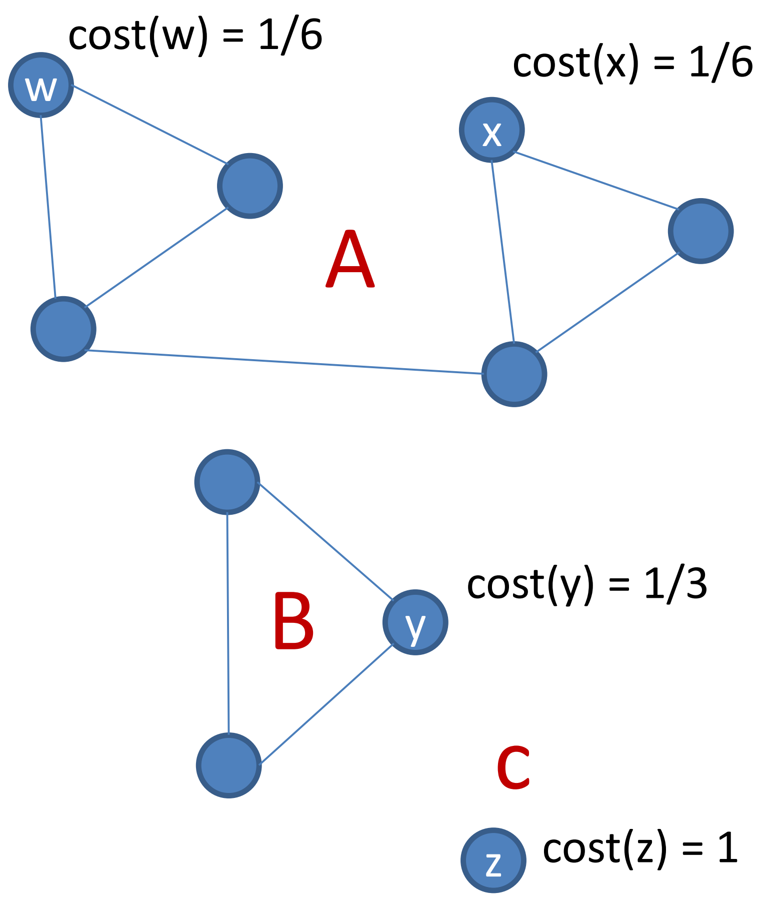
How small is small enough?

Approximate Connected Components

Key Idea 3: Approximate Cost

Approximate low cost components:

If cost(u) < $\varepsilon/2$, round up.



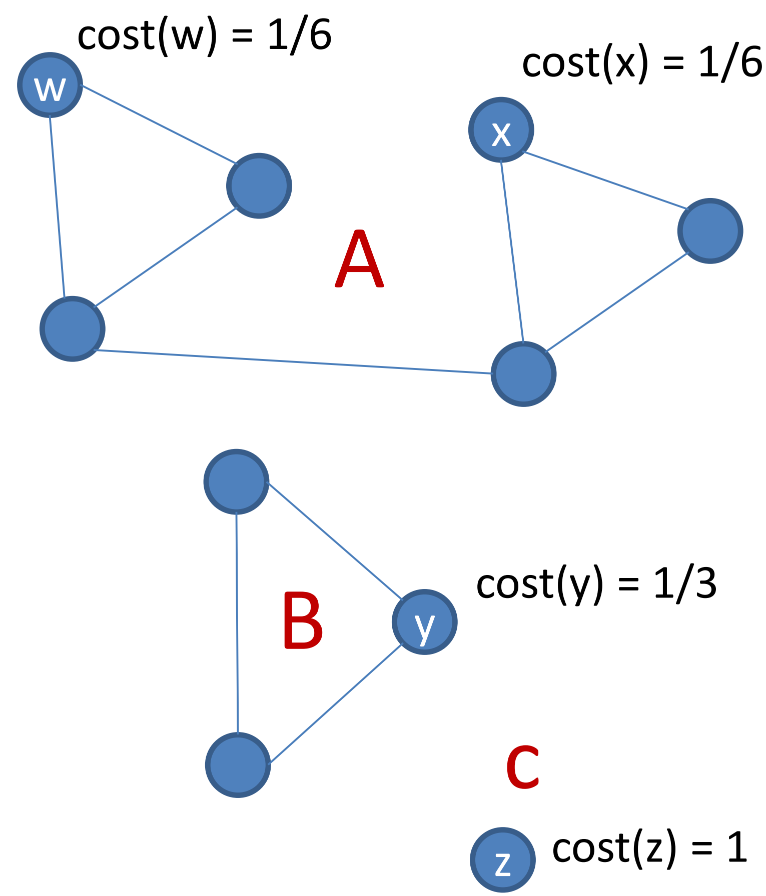
Approximate Connected Components

Key Idea 3: Approximate Cost

Ignore low cost components:

If cost(u) < $\varepsilon/2$, round up.

Total added cost $\leq \varepsilon n/2$.



Approximate Connected Components

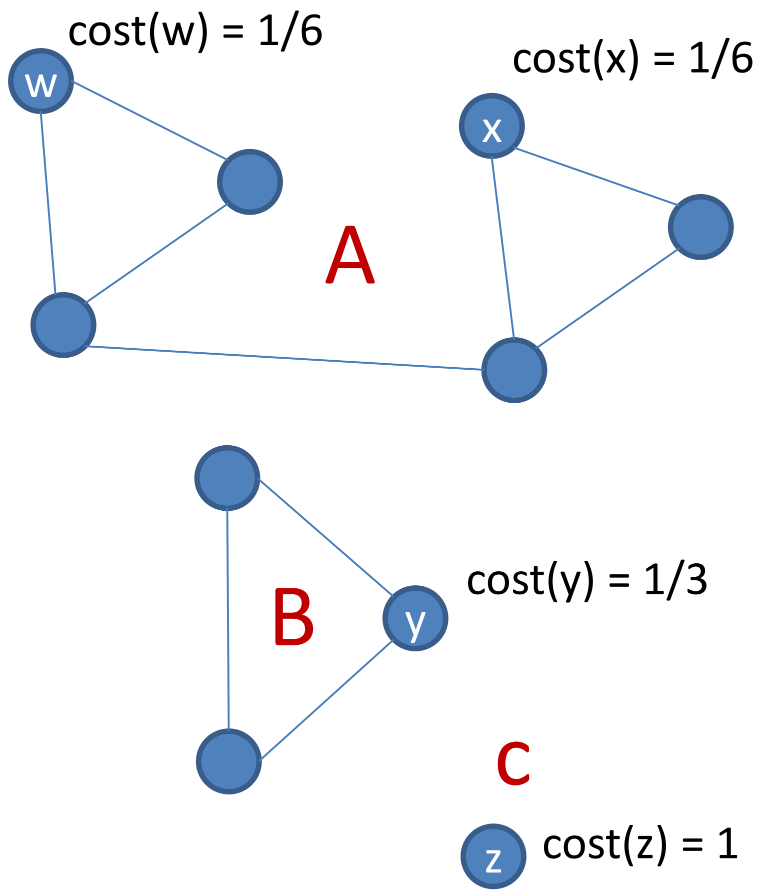
Key Idea 3: Approximate Cost

Define: per-node cost

Let $n(u)$ = number of nodes in the connected component containing node u.

Let $\tilde{n}(u) = \min(n(u), 2/\varepsilon)$.

Let $cost(u) = \max(1/n(u), \varepsilon/2)$
$= 1/\tilde{n}(u)$.



Approximate Connected Components

Key Idea 3: Approximate Cost

Define: per-node cost

Let n(u) = number of nodes in the connected component containing node u.

Let ñ(u) = min(n(u), 2/ε).

Let cost(u) = max(1/n(u), ε/2).
= 1/ñ(u).

Define:

$$\overline{C} = \sum_{u \in V} \mathrm{cost}(u)$$

Note:

$$\begin{aligned} n(u) &\geq \overline{n}(u) \\ 1/n(u) &\leq 1/\overline{n}(u) \end{aligned}$$

Approximate Connected Components

Key Idea 3: Approximate Cost

Define: per-node cost

Let n(u) = number of nodes in the connected component containing node u.

Let ñ(u) = min(n(u), 2/ε).

Let cost(u) = max(1/n(u), ε/2).
= 1/ñ(u).

Define:

$$\overline{C} = \sum_{u \in V} \mathrm{cost}(u)$$

Note:

$$\begin{aligned} n(u) &\geq \overline{n}(u) \\ 1/n(u) &\leq 1/\overline{n}(u) \end{aligned}$$

Approximate Connected Components

Close enough approximation:

$$\left|\mathrm{CC}(G) - \overline{C}\right| \quad = \quad \overline{C} - \mathrm{CC}(G)$$

$$\begin{aligned} n(u) &\geq \overline{n}(u) \\ 1/n(u) &\leq 1/\overline{n}(u) \end{aligned}$$

<u>Intuition:</u>
By rounding cost(u) up to $\varepsilon/2$, we increase the error at most $\varepsilon n/2$.

Approximate Connected Components

Close enough approximation:

$$
\begin{aligned}
\left|\mathrm{CC}(G) - \overline{C}\right| &= \overline{C} - \mathrm{CC}(G) \\
&= \sum_{j=1}^{n} 1/\overline{n}(u) - \sum_{j=1}^{n} 1/n(u)
\end{aligned}
$$

<u>Intuition:</u>
By rounding cost(u) up to $\varepsilon/2$, we increase the error at most $\varepsilon n/2$.

Approximate Connected Components

Close enough approximation:

$$
\begin{aligned}
\left|\mathrm{CC}(G) - \overline{C}\right| &= \overline{C} - \mathrm{CC}(G) \\
&= \sum_{j=1}^{n} 1/\overline{n}(u) - \sum_{j=1}^{n} 1/n(u) \\
&= \sum_{j=1}^{n} \left(1/\overline{n}(j) - 1/n(j)\right)
\end{aligned}
$$

Intuition:
By rounding cost(u) up to $\varepsilon/2$, we increase the error at most $\varepsilon n/2$.

Approximate Connected Components

Close enough approximation:

$$
\begin{aligned}
\left|\mathrm{CC}(G) - \overline{C}\right| &= \overline{C} - \mathrm{CC}(G) \\
&= \sum_{j=1}^{n} 1/\overline{n}(u) - \sum_{j=1}^{n} 1/n(u) \\
&= \sum_{j=1}^{n} \left(1/\overline{n}(j) - 1/n(j)\right) \\
&\leq \sum_{j=1}^{n} 1/\overline{n}(j)
\end{aligned}
$$

<u>Intuition:</u>
By rounding cost(u) up to ε/2, we increase the error at most εn/2.

Approximate Connected Components

Close enough approximation:

$$
\begin{aligned}
|\mathrm{CC}(G) - \overline{C}| &= \overline{C} - \mathrm{CC}(G) \\
&= \sum_{j=1}^{n} 1/\overline{n}(u) - \sum_{j=1}^{n} 1/n(u) \\
&= \sum_{j=1}^{n} (1/\overline{n}(j) - 1/n(j)) \\
&\leq \sum_{j=1}^{n} 1/\overline{n}(j) \\
&\leq \sum_{j=1}^{n} \epsilon/2
\end{aligned}
$$

<u>Intuition:</u>
By rounding cost(u) up to ε/2, we increase the error at most εn/2.

Approximate Connected Components

Close enough approximation:

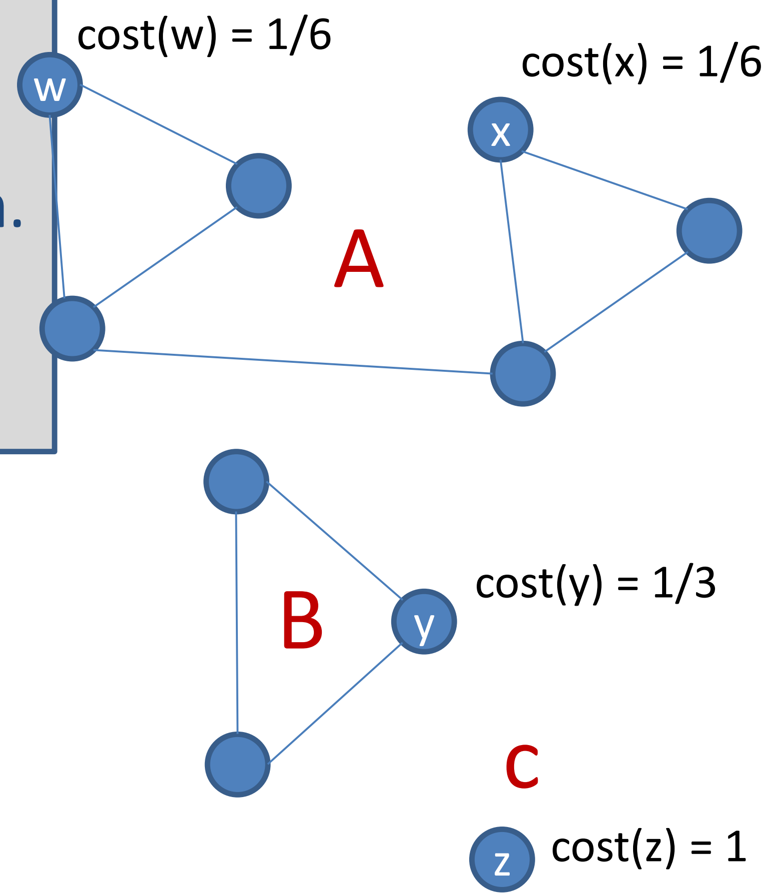
$$
\begin{aligned}
\left|\mathrm{CC}(G) - \overline{C}\right| &= \overline{C} - \mathrm{CC}(G) \\
&= \sum_{j=1}^{n} 1/\overline{n}(u) - \sum_{j=1}^{n} 1/n(u) \\
&= \sum_{j=1}^{n} \left(1/\overline{n}(j) - 1/n(j)\right) \\
&\leq \sum_{j=1}^{n} 1/\overline{n}(j) \\
&\leq \sum_{j=1}^{n} \epsilon/2 \\
&\leq \epsilon n/2
\end{aligned}
$$

<u>Intuition:</u>
By rounding cost(u) up to $\varepsilon/2$, we increase the error at most $\varepsilon n/2$.

Approximate Connected Components

Algorithm 3

sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    sum = sum + cost(u)
return n·(sum/s)



We have shown:

Sufficient to approximate cost(u) by rounding up.

Approximate Connected Components

Algorithm 3

Define: per-node cost

Let $n(u)$ = number of nodes in the connected component containing node u.

Let $\tilde{n}(u) = \min(n(u), 2/\varepsilon)$.

Let $cost(u) = \max(1/n(u), \varepsilon/2)$.
$= 1/\tilde{n}(u)$.

How to efficiently compute $cost(u)$?

Approximate Connected Components

Algorithm 3

Define: per-node cost

Let $n(u)$ = number of nodes in the connected component containing node u.

Let $\tilde{n}(u) = \min(n(u), 2/\varepsilon)$.

Let $cost(u) = \max(1/n(u), \varepsilon/2)$.
$= 1/\tilde{n}(u)$.

How to efficiently compute $cost(u)$?

Approximate Connected Components

Algorithm 3

```
sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    Perform a BFS from u; stop after seeing 2/ε nodes.
    if BFS found > 2/ε nodes:
        sum = sum + ε/2
    else if BFS found n(u) nodes:
        sum = sum + 1/n(u)
return n·(sum/s)
```

Approximate Connected Components

Analysis

Goal:

$$\left|\frac{n}{s} \cdot sum - \overline{C}\right| \leq \epsilon n/2$$

Approximate Connected Components

Analysis

Goal:

$$\left|\frac{n}{s}\cdot sum - \overline{C}\right| \leq \epsilon n/2$$

Implies:

$$\begin{aligned}\left|\frac{n}{s}\cdot sum - \mathrm{CC}(G)\right| &\leq \left|\frac{n}{s}\cdot sum - \overline{C}\right| + \left|\overline{C} - \mathrm{CC}(G)\right| \\ &\leq \epsilon n/2 + \epsilon n/2 \\ &\leq \epsilon n\end{aligned}$$

Approximate Connected Components

Algorithm 3 Analysis

Define random variables: $Y_1, Y_2, \ldots, Y_s$

$$
\begin{aligned}
u_j &= \text{node chosen in } j^{\text{th}} \text{ iteration} \\
Y_j &= cost(u_j)
\end{aligned}
$$

Rounded up cost

Approximate Connected Components

Algorithm 3 Analysis

Define random variables: Y_1, Y_2, …, Y_s

$$
\begin{aligned}
\mathrm{E}\left[Y_j\right] &= \sum_{i=1}^{n} \frac{1}{n} \mathrm{cost}(u_i) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{cost}(u_i) \\
&= \frac{1}{n} \overline{C}
\end{aligned}
$$

Approximate Connected Components

Algorithm 3 Analysis

Unbiased estimator:

$$\mathrm{E}\left[\sum_{j=1}^{s} Y_j\right] = s\mathrm{E}\left[Y_j\right]$$

$$= \frac{s}{n}\overline{C}$$

Approximate Connected Components

Algorithm 3 Analysis

Notice:

Expected output of algorithm is:

$$\mathrm{E}\left[n \cdot (sum/s)\right] = \frac{n}{s}\left(\frac{s}{n}\overline{C}\right) = \overline{C}$$

Approximate Connected Components

Algorithm 3 Analysis

Goal:

$$\Pr\left\{\left|\overline{C} - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \leq 1/3$$

Approximate Connected Components

Algorithm 3 Analysis

Derivation:

$$
\begin{aligned}
\Pr\left\{\left|\overline{C} - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} &= \Pr\left\{\left|\mathrm{E}\left[\frac{n}{s}\sum_{i=1}^{s} Y_i\right] - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \\
&= \Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \frac{s}{n}\epsilon n/2\right\} \\
&= \Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\}
\end{aligned}
$$

Approximate Connected Components

Algorithm 3 Analysis

Derivation:

$$\Pr\left\{\left|\overline{C} - \frac{n}{s}\sum_{j=1}^{s} Y_j\right| > \epsilon n/2\right\} \quad =$$

$$\Pr\left\{\left|\mathrm{E}\left[\sum_{i=1}^{s} Y_i\right] - \sum_{j=1}^{s} Y_j\right| > \epsilon s/2\right\} \quad \leq \quad 2e^{-2(\epsilon s/2)^2/s}$$

$$\leq \quad 2e^{-2\epsilon^2 s/4}$$

$$\leq \quad 2e^{-\epsilon^2(4/epsilon^2)/2}$$

$$\leq \quad 2e^{-2}$$

$$< \quad 1/3$$

$$\boxed{s = \frac{4}{\epsilon^2}}$$

Approximate Connected Components

Analysis

Goal:

$$\left|\frac{n}{s} \cdot sum - \overline{C}\right| \leq \epsilon n/2$$

Implies:

$$\begin{aligned}
\left|\frac{n}{s} \cdot sum - \mathrm{CC}(G)\right| &\leq \left|\frac{n}{s} \cdot sum - \overline{C}\right| + \left|\overline{C} - \mathrm{CC}(G)\right| \\
&\leq \epsilon n/2 + \epsilon n/2 \\
&\leq \epsilon n
\end{aligned}$$

Approximate Connected Components

Algorithm 3

```
sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    Perform a BFS from u; stop after seeing 2/ε nodes.
    if BFS found > 2/ε nodes:
        sum = sum + ε/2
    else if BFS found n(u) nodes:
        sum = sum + 1/n(u)
return n·(sum/s)
```

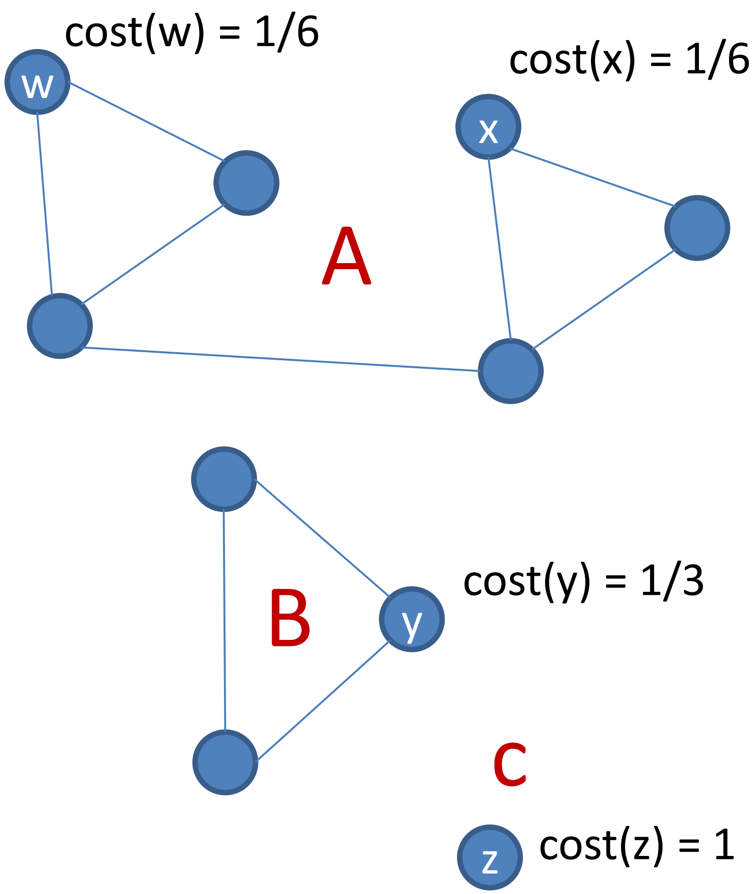
Approximate Connected Components

Algorithm 3

We have shown:

With probability > 2/3, output is equal to:

$CC(G) \pm \varepsilon n$



Approximate Connected Components

Algorithm 3

Cost of BFS: $O((2 / \varepsilon) \cdot d)$

sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    Perform a BFS from u; stop after seeing $2/\varepsilon$ nodes.
    if BFS found > $2/\varepsilon$ nodes:
        sum = sum + $\varepsilon/2$
    else if BFS found n(u) nodes:
        sum = sum + 1/n(u)
return $n \cdot (\text{sum}/s)$

Approximate Connected Components

Algorithm 3

sum = 0
for j = 1 to s:
    Choose u uniformly at random.
    Perform a BFS from u; stop after seeing $2/\varepsilon$ nodes.
    if BFS found > $2/\varepsilon$ nodes:
        sum = sum + $\varepsilon/2$
    else if BFS found n(u) nodes:
        sum = sum + 1/n(u)
return $n \cdot (sum/s)$

Cost of BFS: $O((2 / \varepsilon)\cdot d)$

Total cost:

$O(s(2/\varepsilon)\cdot d) =$

$O((1/\varepsilon^2)(2/\varepsilon)d) =$

$O(d/\varepsilon^3)$

Approximate Connected Components

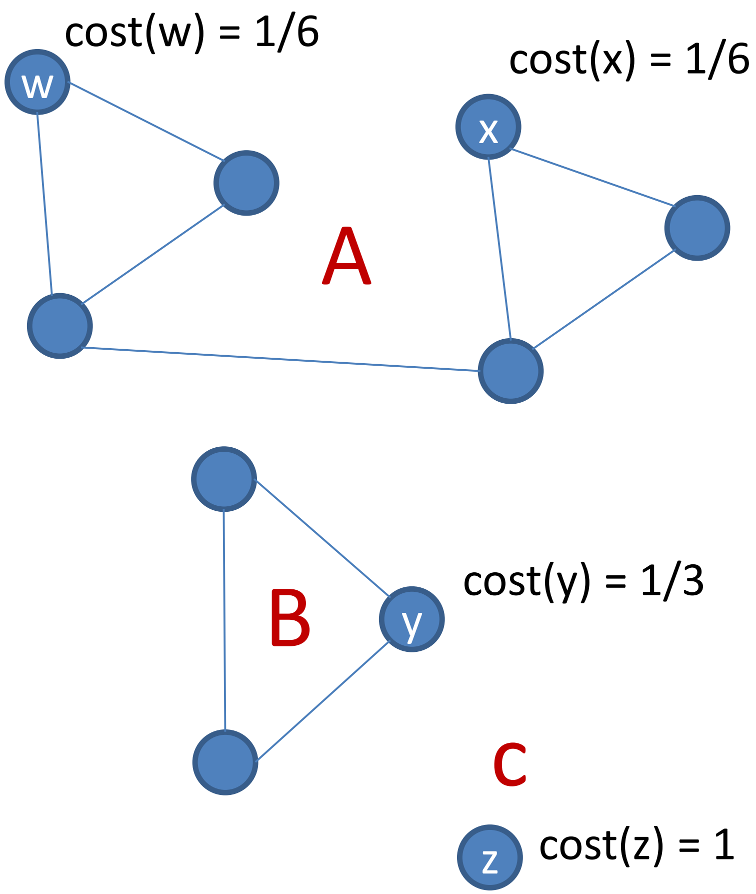
Algorithm 3

We have shown:

With probability > 2/3, output is equal to:

CC(G) ± εn

Running time: $O\left(\frac{d}{\epsilon^3}\right)$



Approximate Connected Components

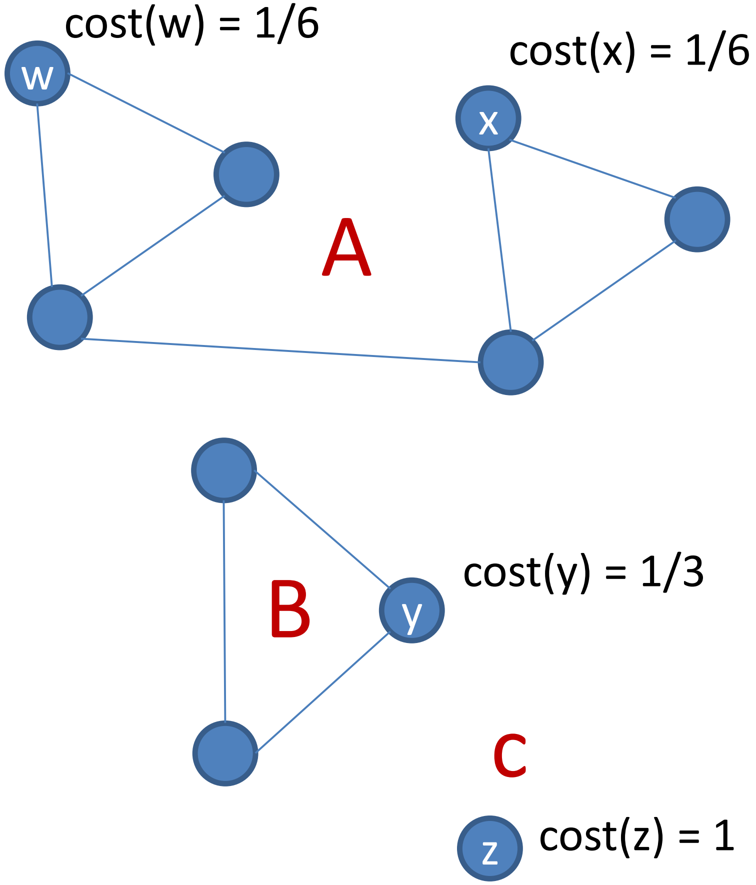
Algorithm 3

We have shown:

With probability > 1 - 1/δ, output is equal to:

CC(G) ± εn

Running time: $O\left(\frac{d \ln \delta}{\epsilon^3}\right)$



Summary

Last Week:

Toy example 1: array all 0's?

- Gap-style question:
  All 0's or far from all 0's?

Toy example 2: Faction of 1's?

- Additive $\pm\,\varepsilon$ approximation
- Hoeffding Bound

Is the graph connected?

- Gap-style question.
- O(1) time algorithm.
- Correct with probability 2/3.

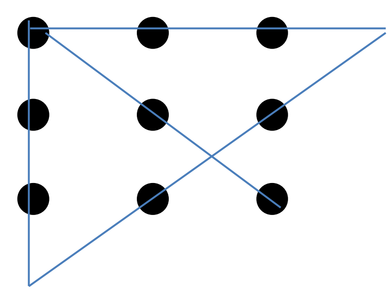
Today:

Number of connected components in a graph.

- Approximation algorithm.

Weight of MST

- Approximation algorithm.



9 dots
4 lines

Today's Problem: Minimum Spanning Tree

Assumptions:

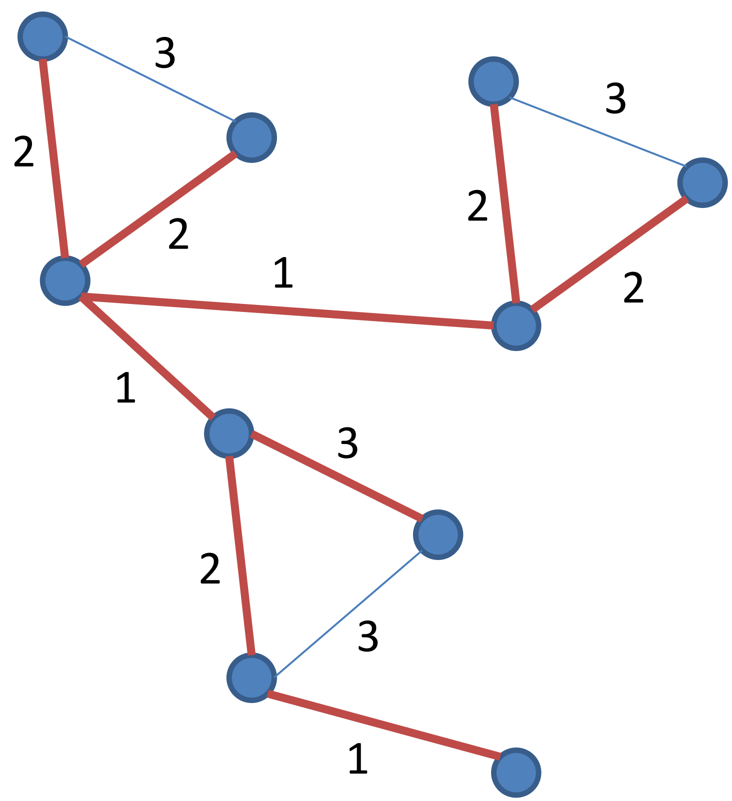
Graph G = (V,E)

- Undirected
- Weighted, max weight W
- Connected
- n nodes
- m edges
- maximum degree d

Error term: $\varepsilon < 1/2$

Output:

Weight of MST.



Example: output 16

Today's Problem: Minimum Spanning Tree

Approximation:

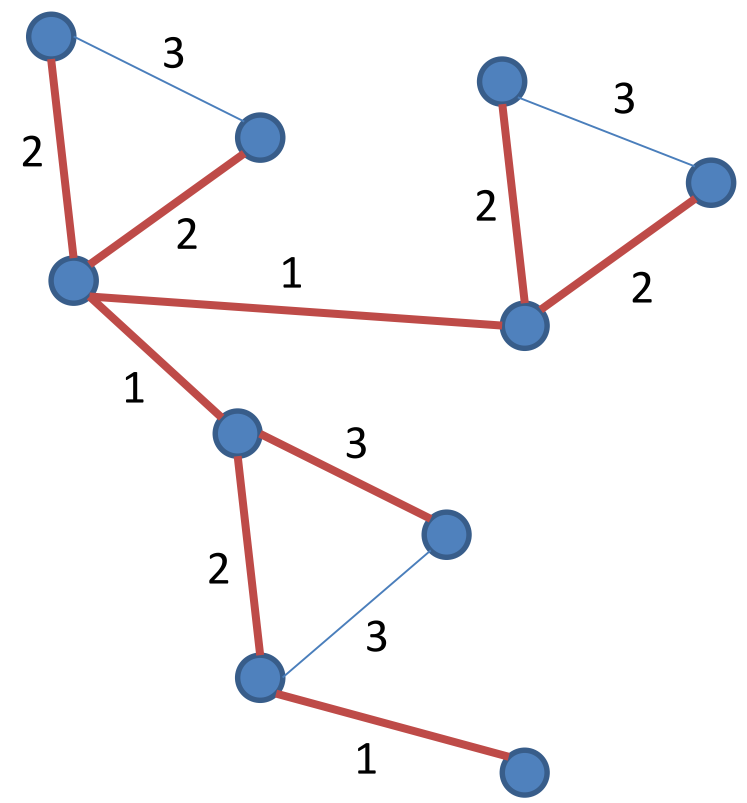
Output M such that:

$$\mathrm{MST}(G)(1-\epsilon) \leq M \leq \mathrm{MST}(1+\epsilon)$$

Alternate form:

$$M = \mathrm{MST}(G)(1 \pm \epsilon)$$

Correct output: w.p. > 2/3



Example:

ε = 1/4

Output ∈ [12,20]

Today's Problem: Minimum Spanning Tree

When is this useful?

What are trivial values of ε?

What are hard values of ε?

What sort of applications is this useful for?

Why multiplicative approximation for MST and additive approximation for connected components?

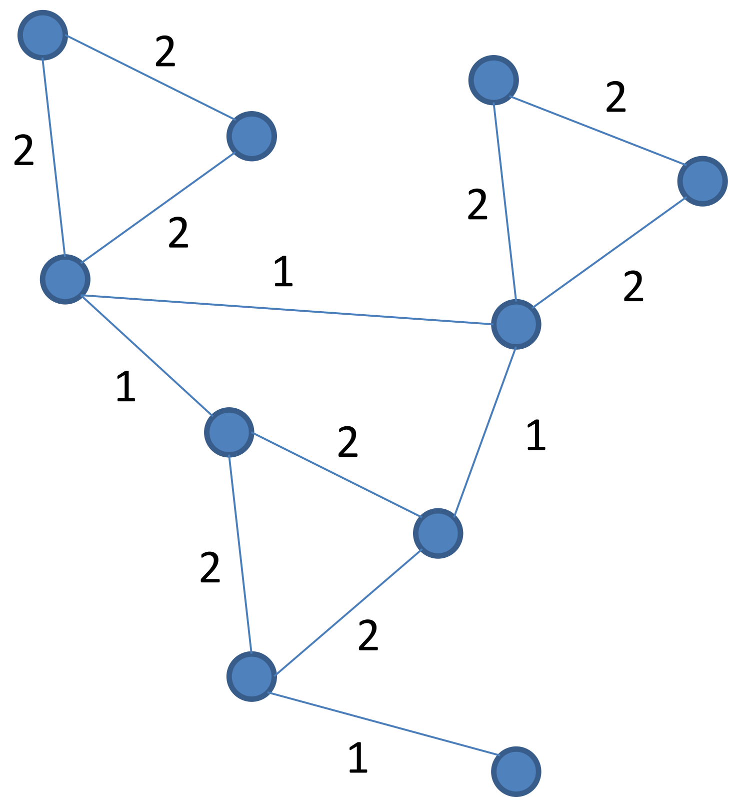
Simple Minimum Spanning Tree

Assume all weights 1 or 2

Which edges must be in MST?

How many weight-2 edges in MST?

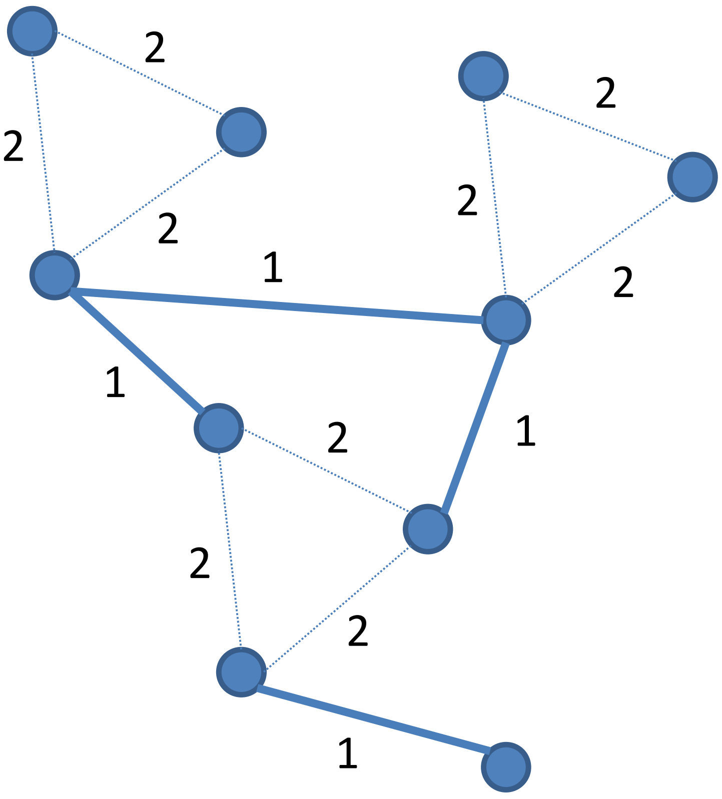
Best (exact) algorithm?



Simple Minimum Spanning Tree

Assume all weights 1 or 2

Let G_1 = graph containing only edges of weight 1.

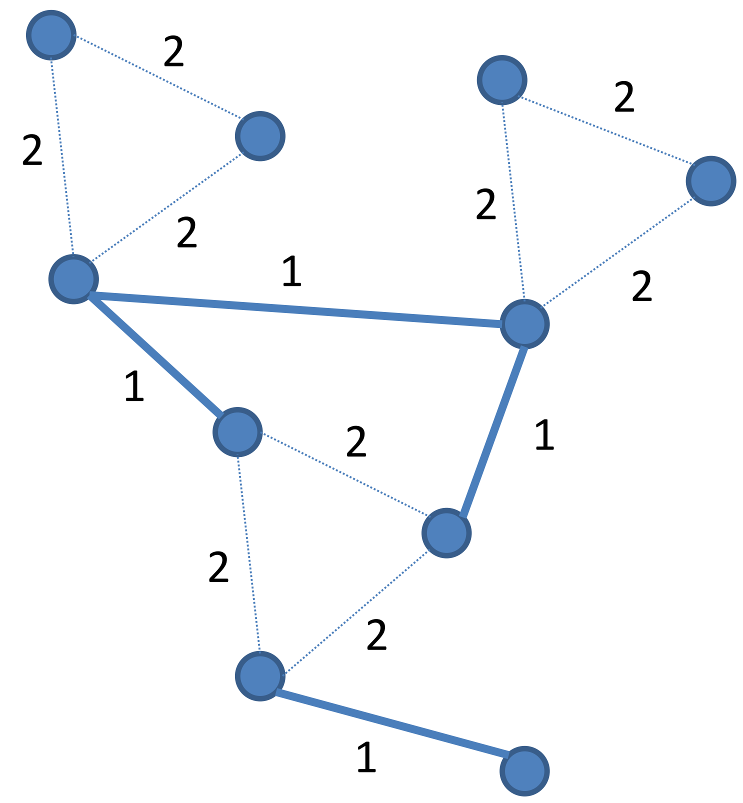


Simple Minimum Spanning Tree

Assume all weights 1 or 2

Let G_1 = graph containing only edges of weight 1.

Let C_1 = number of connected components in G_1.



Ex: $C_1 = 6$

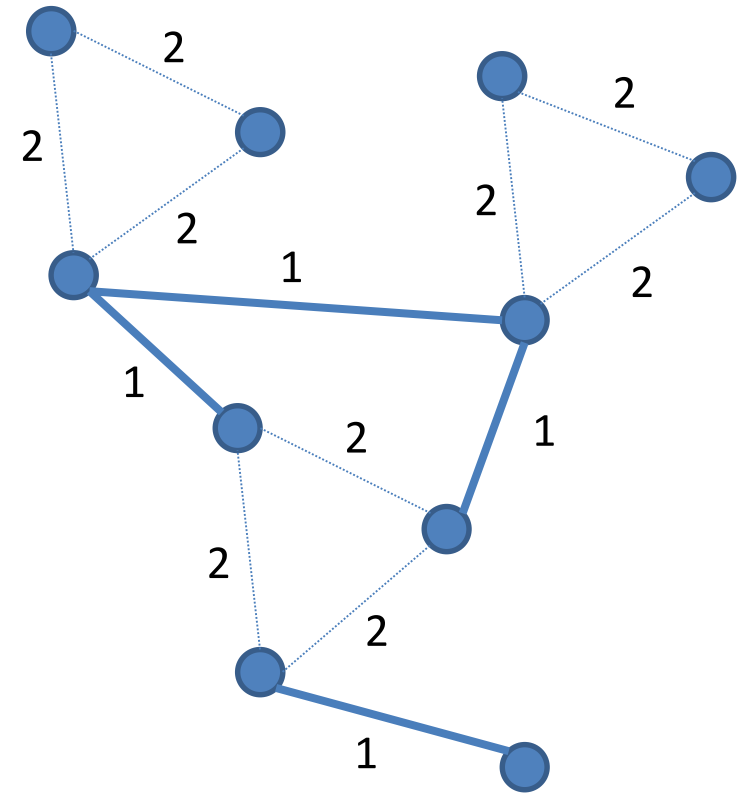
Simple Minimum Spanning Tree

Assume all weights 1 or 2

Let G_1 = graph containing only edges of weight 1.

Let C_1 = number of connected components in G_1.

Claim: MST contains example C_1-1 edges of weight 2.



Ex: $C_1 = 6$

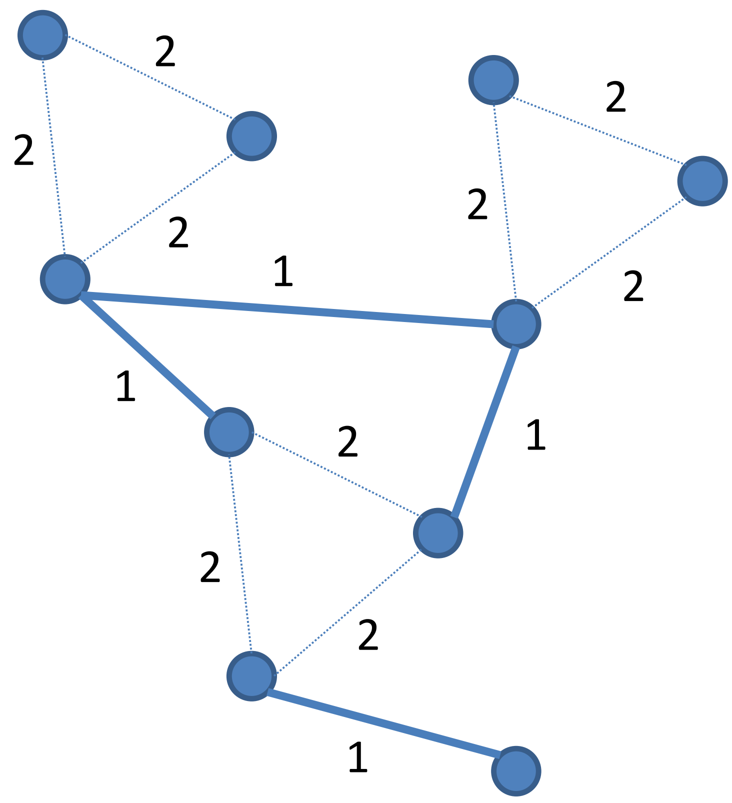
Simple Minimum Spanning Tree

Assume all weights 1 or 2

Claim: MST contains example
C_1-1 edges of weight 2.

Basic MST Property:

For any cut, minimum weight edge across cut is in MST.



Ex: $C_1 = 6$

Simple Minimum Spanning Tree

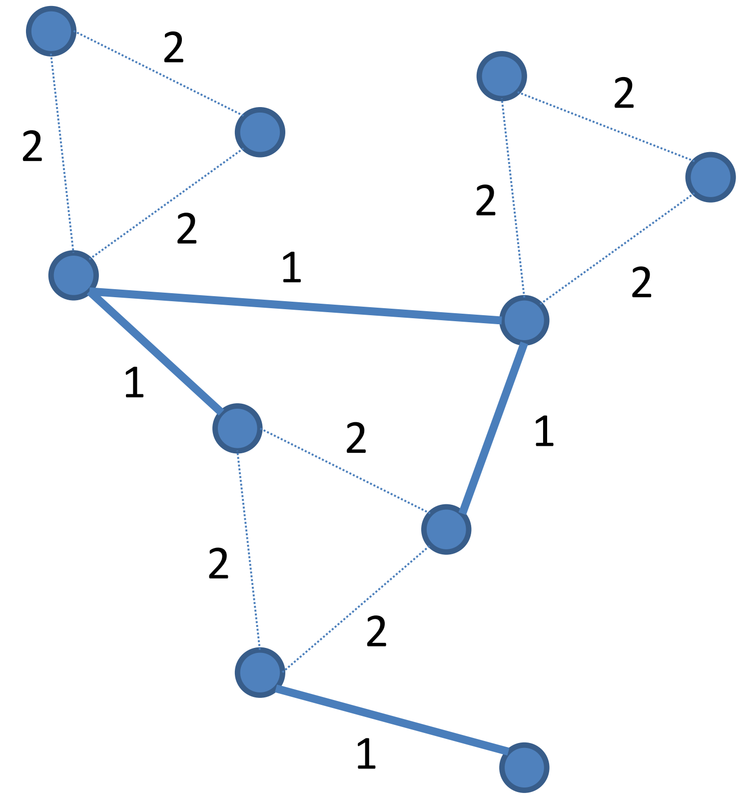
Assume all weights 1 or 2

Claim: MST contains example C_1-1 edges of weight 2.

Algorithm:

For any connected component, add minimum weight outgoing edge.

Here all the edges have weight 2, so add C_1-1 edges of weight 2.



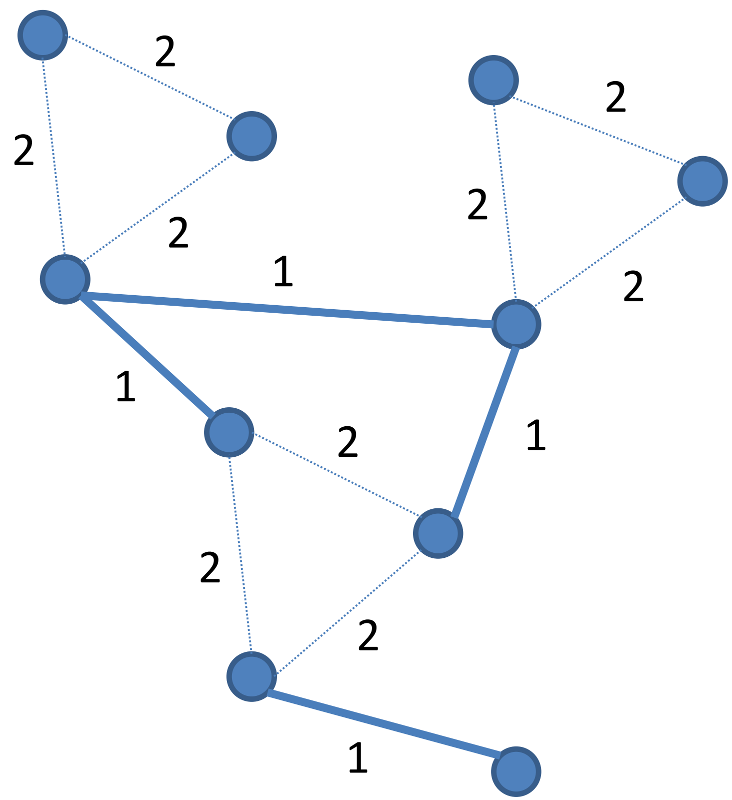
Ex: $C_1 = 6$

Simple Minimum Spanning Tree

Assume all weights 1 or 2

Claim: MST contains example C_1-1 edges of weight 2.

Weight of MST?

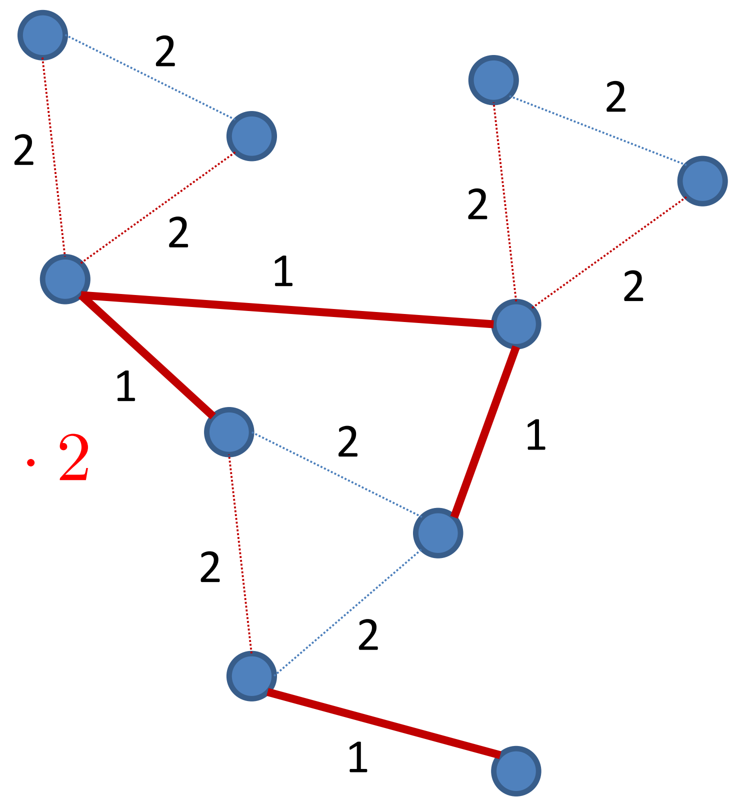


Ex: C_1 = 6

Simple Minimum Spanning Tree

Assume all weights 1 or 2

Claim: MST contains example
C_1-1 edges of weight 2.



Weight of MST?

$$(n-(C_1-1)-1)\cdot 1+(C_1-1)\cdot 2$$

$$=n+C_1-2$$

Ex: 10 + 6 – 2 = 14

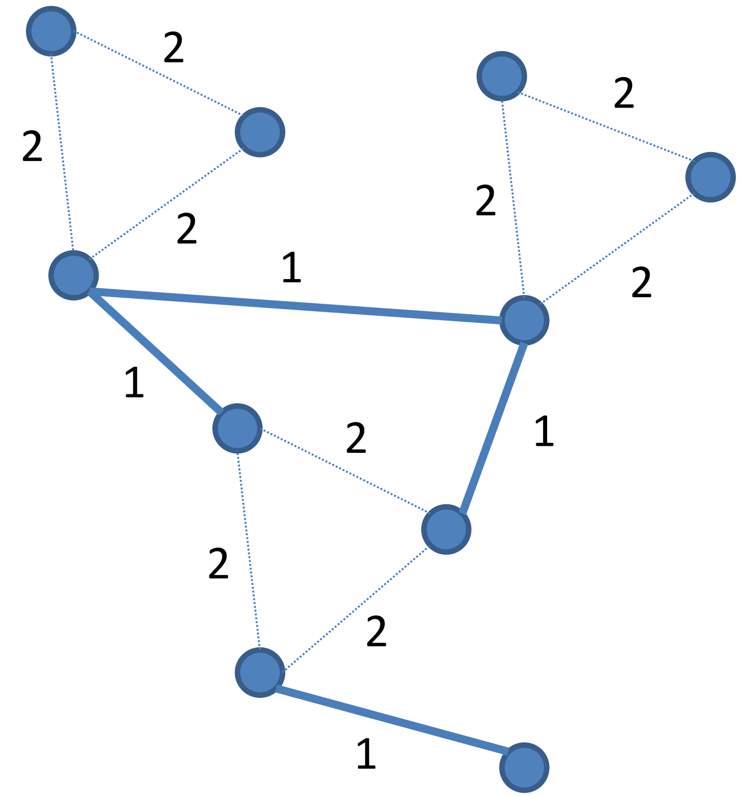
Ex: C_1 = 6

Simple Minimum Spanning Tree

Assume all weights 1 or 2

Weight of MST: $n + C_1 - 2$

Algorithm idea?



Ex: $C_1 = 6$

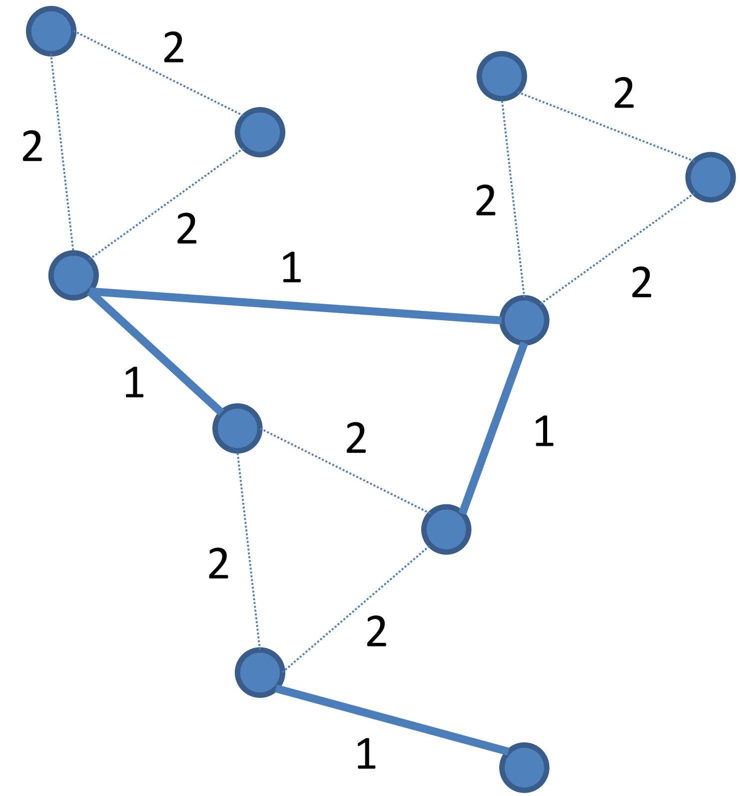
Simple Minimum Spanning Tree

Assume all weights 1 or 2

Weight of MST: $n + C_1 - 2$

Algorithm idea:

Approximate connected components of G_1.



Ex: $C_1 = 6$

Approximate Minimum Spanning Tree

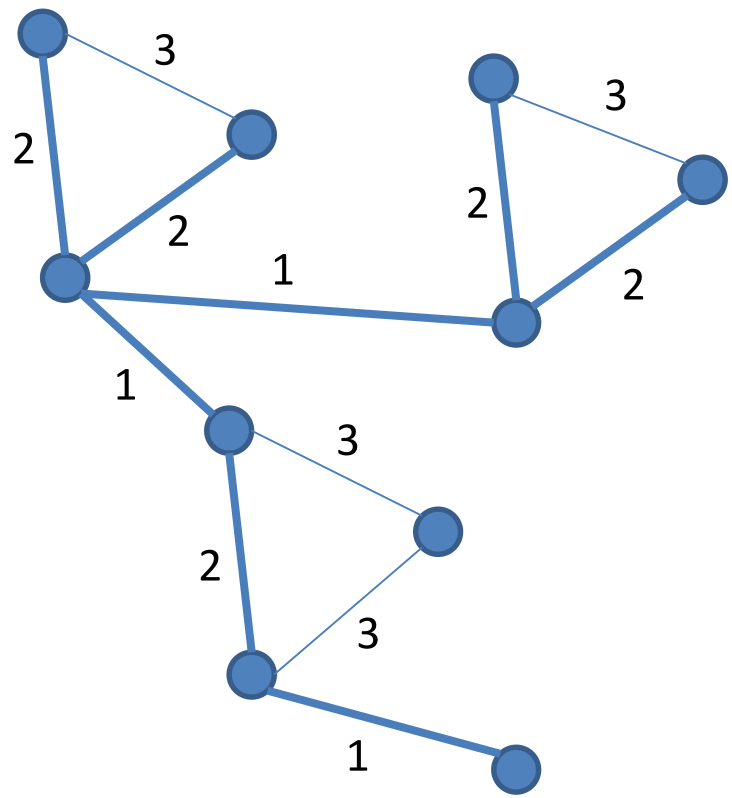
Weights {1, 2, …, W}

Let G_1 = graph containing only edges of weight 1.

Let G_2 = graph containing only edges of weight {1, 2}.

…

Let G_j = graph containing only edges of weights {1, 2, …, j}.



Ex: G_2

Approximate Minimum Spanning Tree

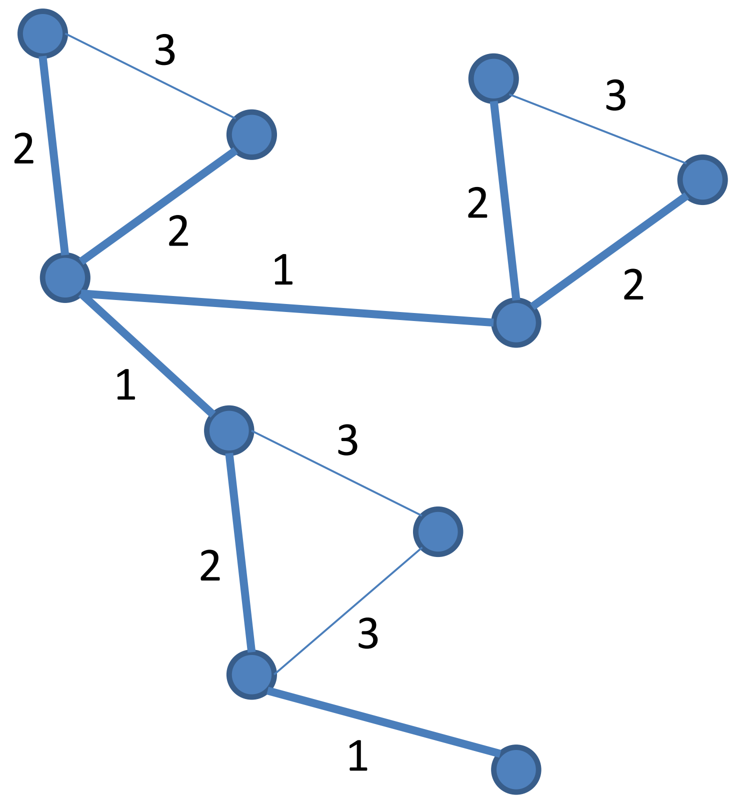
Weights {1, 2, …, W}

Let C_1 = number CC in G_1.

Let C_2 = number CC in G_2.

…

Let C_j = number CC in G_j.



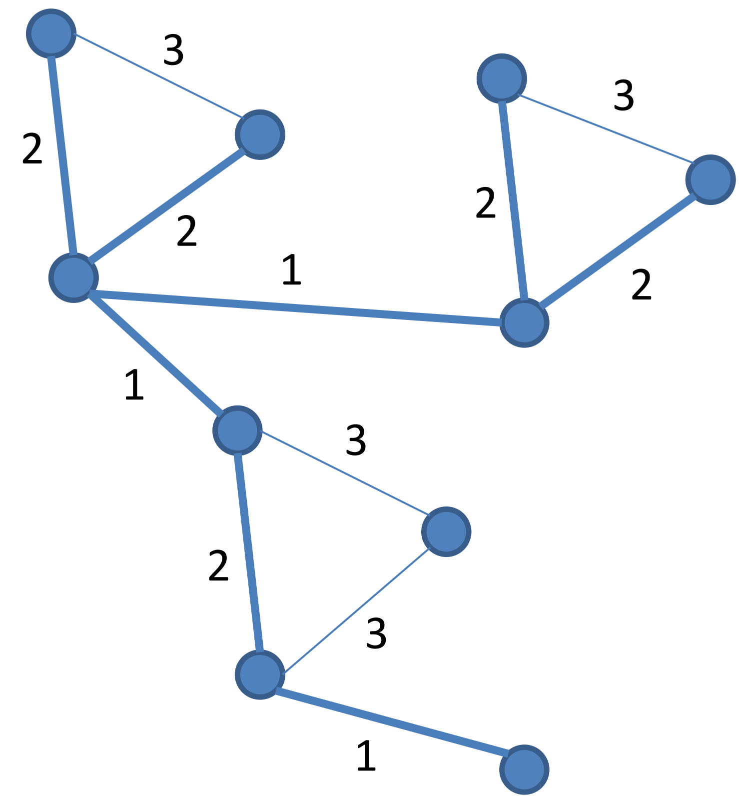
Ex: G_2

Approximate Minimum Spanning Tree

Weights {1, 2, …, W}

Claim:

MST(G) contains $C_j - 1$ edges of weight $> j$.



Ex: G_2

Approximate Minimum Spanning Tree

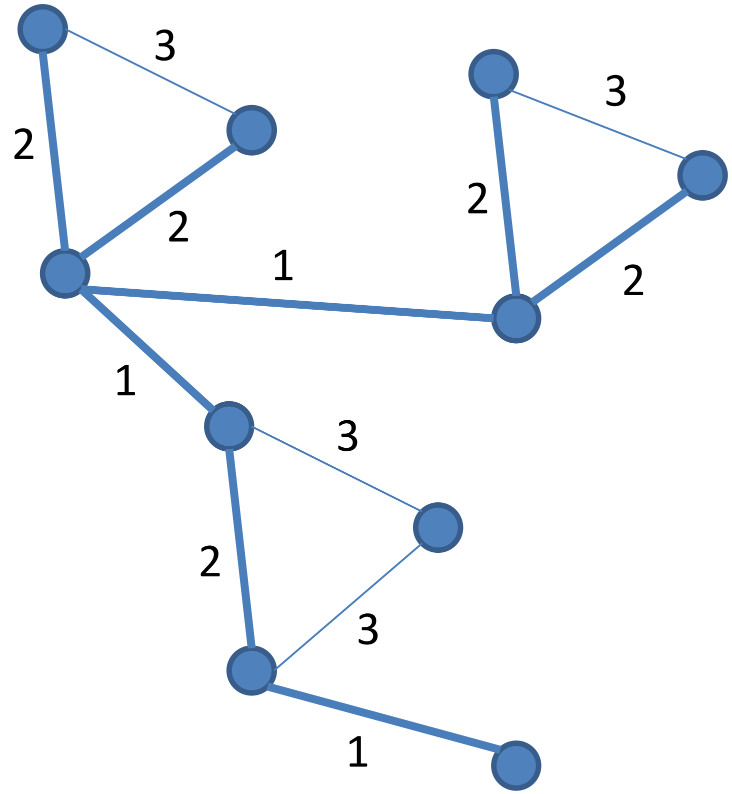
Weights {1, 2, …, W}

Claim:

MST(G) contains $C_j - 1$ edges of weight $> j$.

Why?

There are C_j connected components in G_j. There much be $C_j - 1$ edges connecting them, and each must have weight $> j$.



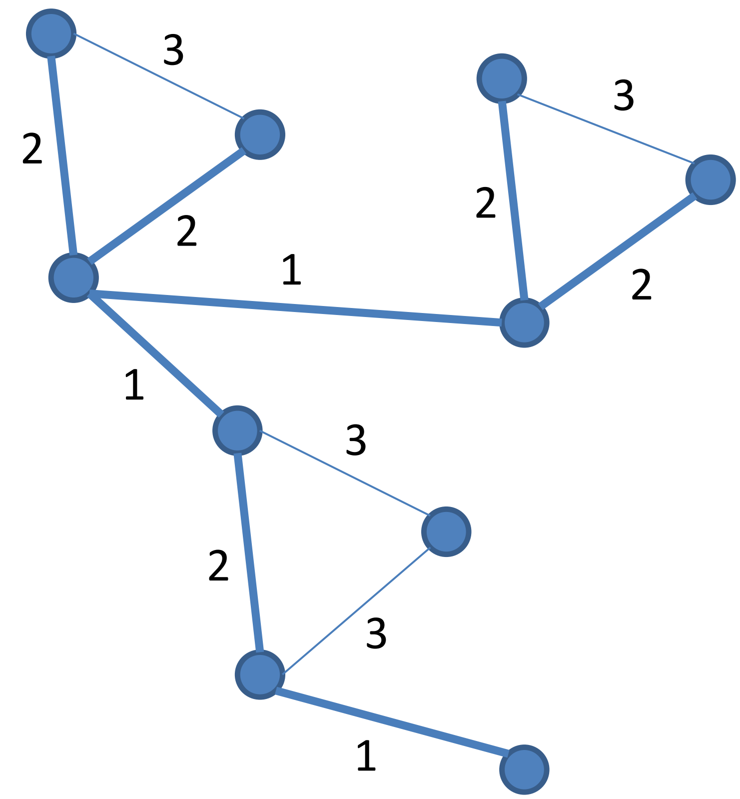
Ex: G_2

Approximate Minimum Spanning Tree

Weights {1, 2, …, W}

Lemma:

$$\mathrm{MST}(G) = n - W + \sum_{j=1}^{W-1} C_j$$



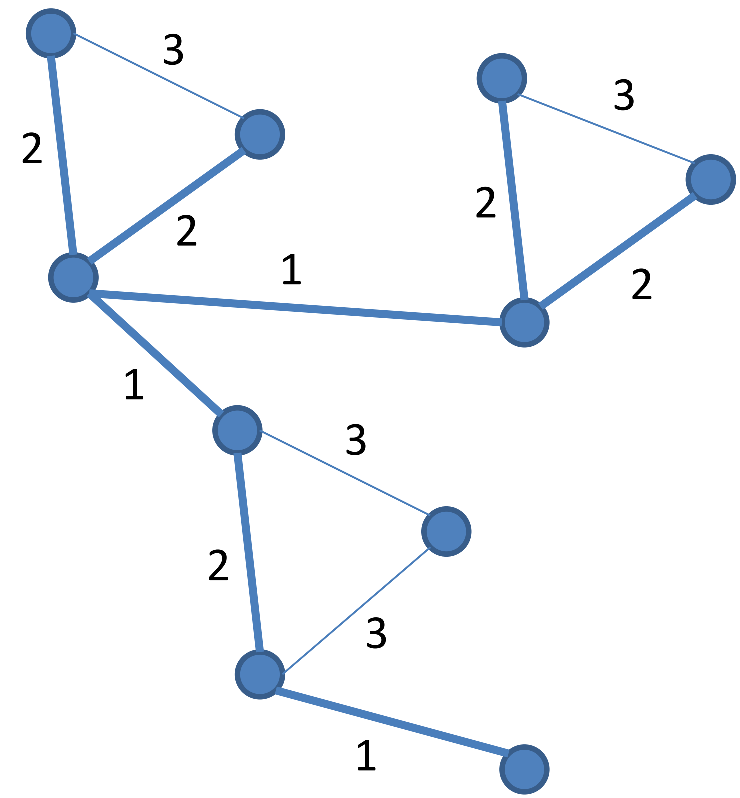
Ex: G_2

Approximate Minimum Spanning Tree

Weights {1, 2, …, W}

Edges of weight 1:

$n - 1$ edges total in MST
$C_1 - 1$ edges of weight > 1
➔
$(n - 1) - (C_1 - 1)$ edges of weight 1.
➔
$(n - C_1)$ edges of weight 1.



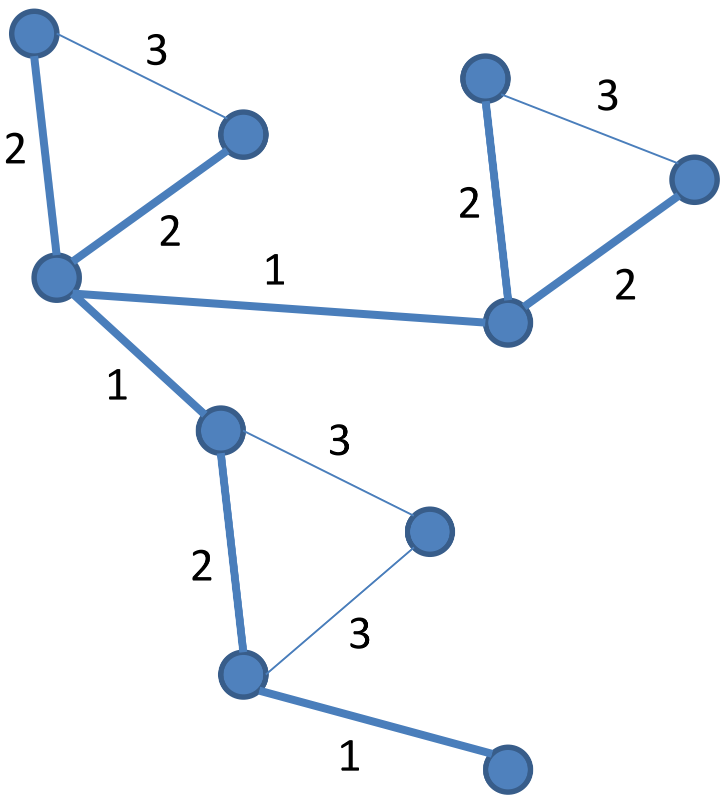
Ex: G_2

Approximate Minimum Spanning Tree

Weights {1, 2, …, W}

Edges of weight $j+1$:

$C_j - 1$ edges of weight $> j$
$C_{j+1} - 1$ edges of weight $> j+1$
➔
$(C_j - 1) - (C_{j+1} - 1)$ edges of weight $j+1$.
➔
$(C_j - C_{j+1})$ edges of weight $j+1$.



Ex: G_2

Note: $C_j \geq C_{j+1}$

Approximate Minimum Spanning Tree

Weights {1, 2, …, W}

Sum the weights:

$$\mathrm{MST}(G) \quad = \quad (n - C_1) + \sum_{j=1}^{W-1} (j+1)(C_j - C_{j+1})$$

number of edges of weight 1

weight of edge of weight j+1

number of edges of weight j+1

Note: sum is from j = 1 to W-1.

Approximate Minimum Spanning Tree

Weights {1, 2, ..., W}

Sum the weights:

$$\begin{aligned}\mathrm{MST}(G) &= (n - C_1) + \sum_{j=1}^{W-1} (j+1)(C_j - C_{j+1}) \\ &= (n - C_1) + (2C_1 - 2C_2) + (3C_2 - 3C_3) \\ &\quad +(4C_3 - 4C_4) + \ldots \\ &\quad +(WC_{W_1} - WC_W)\end{aligned}$$

Approximate Minimum Spanning Tree

Weights {1, 2, ..., W}

Sum the weights:

$$
\begin{aligned}
\mathrm{MST}(G) &= (n - C_1) + \sum_{j=1}^{W-1} (j+1)(C_j - C_{j+1}) \\
&= (n - C_1) + (2C_1 - 2C_2) + (3C_2 - 3C_3) \\
&\quad +(4C_3 - 4C_4) + \ldots \\
&\quad +(WC_{W_1} - WC_W) \\
&= \boxed{n + C_1 + C_2 + \ldots + C_{W-1} - WC_W}
\end{aligned}
$$

Approximate Minimum Spanning Tree

Weights {1, 2, ..., W}

$$C_W = 1$$

Sum the weights:

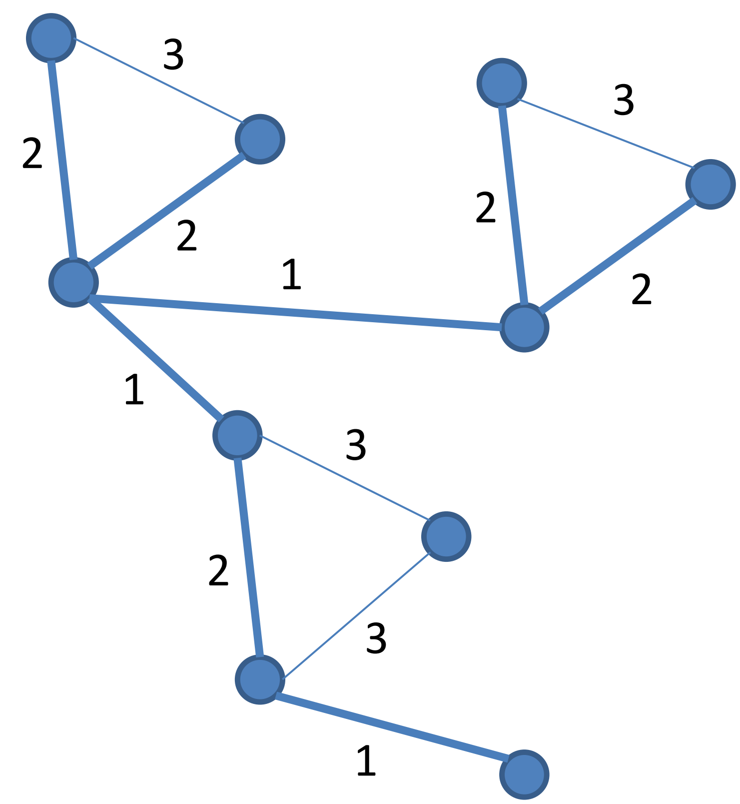
$$\begin{aligned} \mathrm{MST}(G) &= n + C_1 + C_2 + \ldots + C_{W-1} - W C_W \\ &= n + C_1 + C_2 + \ldots + C_{W-1} - W \\ &= \boxed{n - W + \sum_{j=1}^{W-1} C_j} \end{aligned}$$

Approximate Minimum Spanning Tree

Weights {1, 2, …, W}

Lemma:

$$\mathrm{MST}(G) = n - W + \sum_{j=1}^{W-1} C_j$$



Ex: G_2

Approximate Minimum Spanning Tree

Algorithm ApproxMST

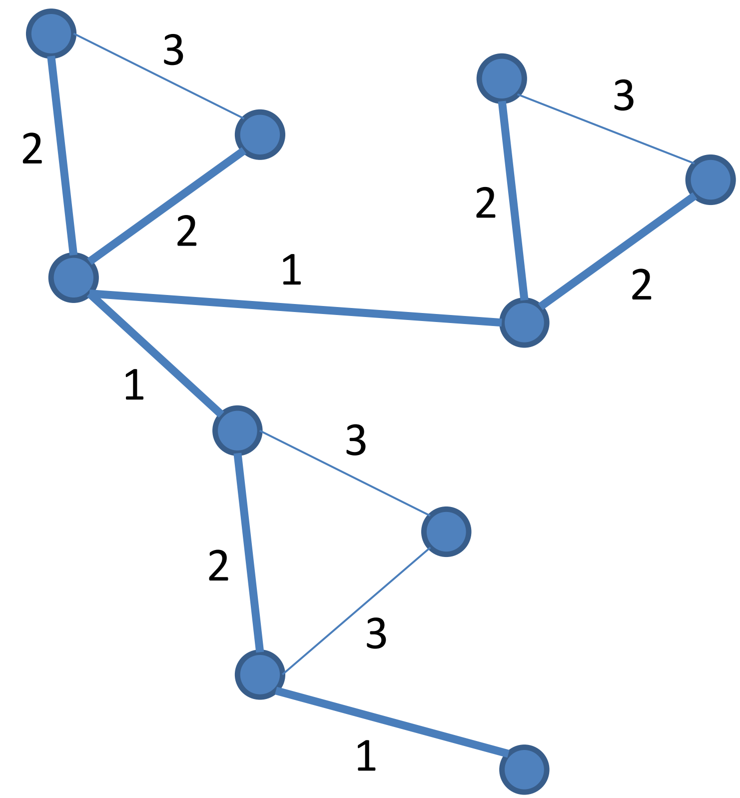
sum = n – W

for j = 1 to W – 1:

    X_j = AproxCC(G_j, d, ε', δ)

    sum = sum + X_j

return sum



Ex: G_2

Approximate Minimum Spanning Tree

Error Calculation

```
sum = n – W
for j = 1 to W – 1:
    X_j = AproxCC(G_j, d, ε', δ)
    sum = sum + X_j
return sum
```

Set: $\varepsilon' = \varepsilon/W$

Sum of errors: $\leq W(\varepsilon n/W) \leq \varepsilon n$

Approximate Minimum Spanning Tree

Error Calculation

```
sum = n – W
for j = 1 to W – 1:
    X_j = AproxCC(G_j, d, ε', δ)
    sum = sum + X_j
return sum
```

Guarantee for each AproxCC:

$$\Pr\{|X_j - C_j| > \epsilon n/W\} < 1/3$$

Approximate Minimum Spanning Tree

Error Calculation

```
sum = n – W
for j = 1 to W – 1:
     X_j = AproxCC(G_j, d, ε', δ)
     sum = sum + X_j
return sum
```

Guarantee for each AproxCC:

$$\Pr\{|X_j - C_j| > \epsilon n/W\} < 1/3$$

Not good enough: $\Pr\{\text{all correct}\} \cong (2/3)^W$

Approximate Minimum Spanning Tree

Error Calculation

```
sum = n – W
for j = 1 to W – 1:
    Xj = AproxCC(Gj, d, ε', δ)
    sum = sum + Xj
return sum
```

Set $\varepsilon' = \varepsilon/W$, $\delta = 1/(3W)$

Error probability: $$\Pr\{\text{any fails}\} \le \sum_{j=1}^{W-1} \frac{1}{3W} \le \frac{W-1}{3W} < 1/3$$

Approximate Minimum Spanning Tree

Error Calculation

sum = n – W

for j = 1 to W – 1:

    X_j = AproxCC(G_j, d, ε', δ)

    sum = sum + X_j

return sum

Set $\varepsilon' = \varepsilon/W$, $\delta = 1/(3W)$

Guarantee for each AproxCC:

$$\Pr\{|X_j - C_j| > \epsilon n/W\} < \frac{1}{3W}$$

Approximate Minimum Spanning Tree

Error Calculation

```
sum = n – W
for j = 1 to W – 1:
    X_j = AproxCC(G_j, d, ε', δ)
    sum = sum + X_j
return sum
```

Set: $\varepsilon' = \varepsilon/W$, $\delta = 1/(3W)$

Sum of errors: $\leq W(\varepsilon n/W) \leq \varepsilon n$

➔ $\mathrm{MST}(G) - \epsilon n \leq sum \leq \mathrm{MST}(G) + \epsilon n$

Approximate Minimum Spanning Tree

Error Calculation

$$\boxed{\mathrm{MST}(G) \geq n - 1 \geq n/2}$$

Approximate Minimum Spanning Tree

Error Calculation

$$\mathrm{MST}(G) \geq n - 1 \geq n/2$$

$$\mathrm{MST}(G) - \epsilon n \leq sum \leq \mathrm{MST}(G) + \epsilon n$$

Approximate Minimum Spanning Tree

Error Calculation

$$\boxed{\mathrm{MST}(G) \geq n - 1 \geq n/2}$$

$$\boxed{\mathrm{MST}(G) - \epsilon n \leq sum \leq \mathrm{MST}(G) + \epsilon n}$$

$$\begin{aligned} \mathrm{MST}(G) + \epsilon n &\leq \mathrm{MST}(G) + \epsilon(2\mathrm{MST}(G)) \\ &\leq \mathrm{MST}(G)(1 + 2\epsilon) \end{aligned}$$

Approximate Minimum Spanning Tree

Error Calculation

$$\mathrm{MST}(G) \geq n - 1 \geq n/2$$

$$\boxed{\mathrm{MST}(G) - \epsilon n \leq sum \leq \mathrm{MST}(G) + \epsilon n}$$

$$\begin{aligned} \mathrm{MST}(G) + \epsilon n &\leq \mathrm{MST}(G) + \epsilon(2\mathrm{MST}(G)) \\ &\leq \mathrm{MST}(G)(1 + 2\epsilon) \end{aligned}$$

$$\begin{aligned} \mathrm{MST}(G) - \epsilon n &\geq \mathrm{MST}(G) - \epsilon(2\mathrm{MST}(G)) \\ &\geq \mathrm{MST}(G)(1 - 2\epsilon) \end{aligned}$$

Approximate Minimum Spanning Tree

Error Calculation

$$\boxed{\mathrm{MST}(G) \geq n-1 \geq n/2}$$

$$\boxed{\mathrm{MST}(G) - \epsilon n \leq sum \leq \mathrm{MST}(G) + \epsilon n}$$

$$\begin{aligned}
\mathrm{MST}(G) + \epsilon n &\leq \mathrm{MST}(G) + \epsilon(2\mathrm{MST}(G)) \\
&\leq \mathrm{MST}(G)(1+2\epsilon)
\end{aligned}$$

$$\begin{aligned}
\mathrm{MST}(G) - \epsilon n &\geq \mathrm{MST}(G) - \epsilon(2\mathrm{MST}(G)) \\
&\geq \mathrm{MST}(G)(1-2\epsilon)
\end{aligned}$$

$$\boxed{\mathrm{MST}(G)(1-2\epsilon) \leq \mathrm{MST}(G) \leq \mathrm{MST}(G)(1+2\epsilon)}$$

Approximate Minimum Spanning Tree

Running time

```
sum = n – W
for j = 1 to W – 1:
    X_j = AproxCC(G_j, d, ε', δ)
    sum = sum + X_j
return sum
```

Set $\varepsilon' = \varepsilon/W$, $\delta = 1/(3W)$

Running time: $O\left(W \cdot \frac{d \ln\left(1/(1/3W)\right)}{(\epsilon/W)^3}\right)$

Approximate Minimum Spanning Tree

Running Time

```
sum = n – W
for j = 1 to W – 1:
    X_j = AproxCC(G_j, d, ε', δ)
    sum = sum + X_j
return sum
```

Set $\varepsilon' = \varepsilon/W$, $\delta = 1/(3W)$

Running time: $O\left(W \cdot \frac{d \ln(1/(1/3W))}{(\epsilon/W)^3}\right) = \boxed{O\left(\frac{dW^4 \log W}{\epsilon^3}\right)}$

Approximate MST

Summary

We have shown:

With probability > 2/3, output is equal to:

MST(G)(1 ± εn)

Running time: $$O\left(\frac{dW^4 \log W}{\epsilon^3}\right)$$

Approximate MST

Summary

Note:

See: Chazelle, Rubinfeld, Trevisan

Impossible to do better than: $$\Omega\left(\frac{dW}{\epsilon^2}\right)$$

Best known: $$O\left(\frac{dW}{\epsilon^2}\log\frac{dW}{\epsilon}\right)$$

Summary

Last Week:

Toy example 1: array all 0's?

- Gap-style question:
  All 0's or far from all 0's?

Toy example 2: Faction of 1's?

- Additive $\pm\,\varepsilon$ approximation
- Hoeffding Bound

Is the graph connected?

- Gap-style question.
- O(1) time algorithm.
- Correct with probability 2/3.

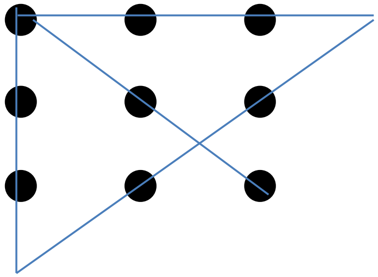
Today:

Number of connected components in a graph.

- Approximation algorithm.

Weight of MST

- Approximation algorithm.



9 dots
4 lines

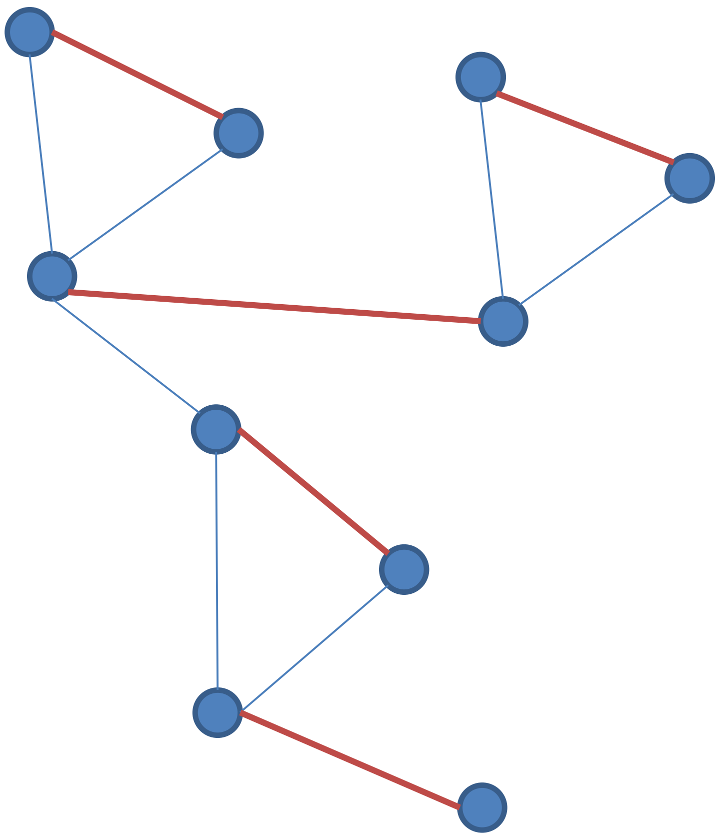
Today's Problem: Maximum Matching

Matching:

Output set of edges M such that no two edges in M are adjacent.

Size of Maximum Matching:

Output the largest value v where there is a matching M of size v.



Example:
Size of matching: 5

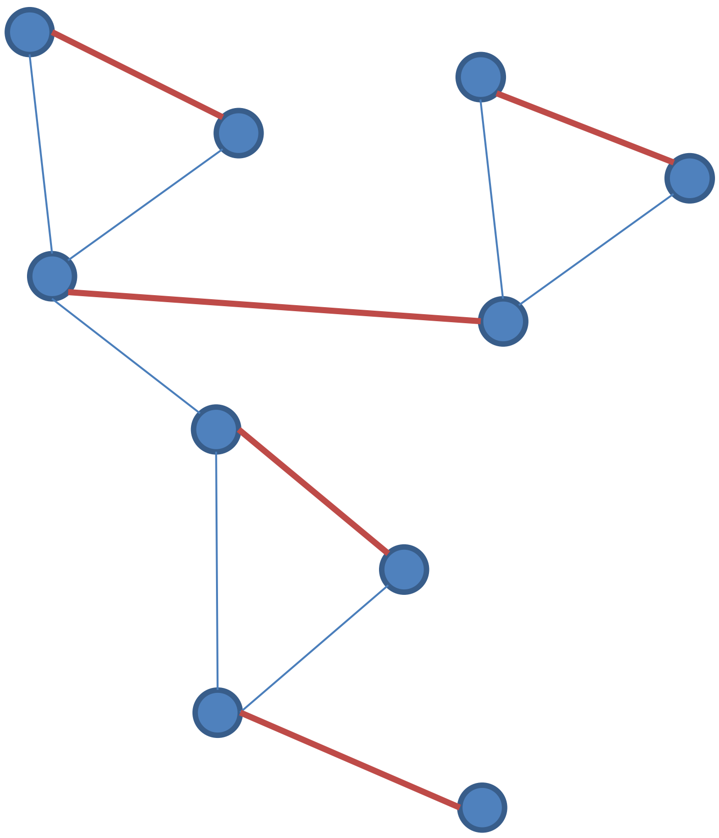
Today's Problem: Maximal Matching

Maximal Matching:

Output set of edges M such that no two edges in M are adjacent, and no more edges can be added to M.

Size of Maximal Matching:

Output the largest value v where there is a maximal matching M of size v.



Example:
Size of matching: 5

Today's Problem: Maximal Matching

Size of Maximal Matching:

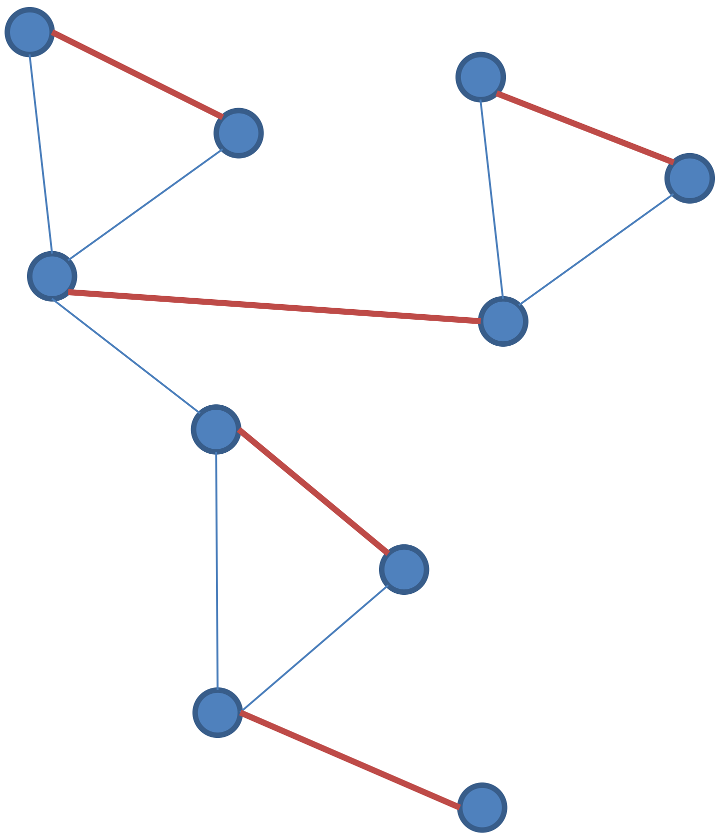
Output the largest value v where there is a maximal matching M of size v.

Note:

The maximum matching is at most twice as big as the maximal matching.

➔

Maximal is a 2-approximation of maximum.



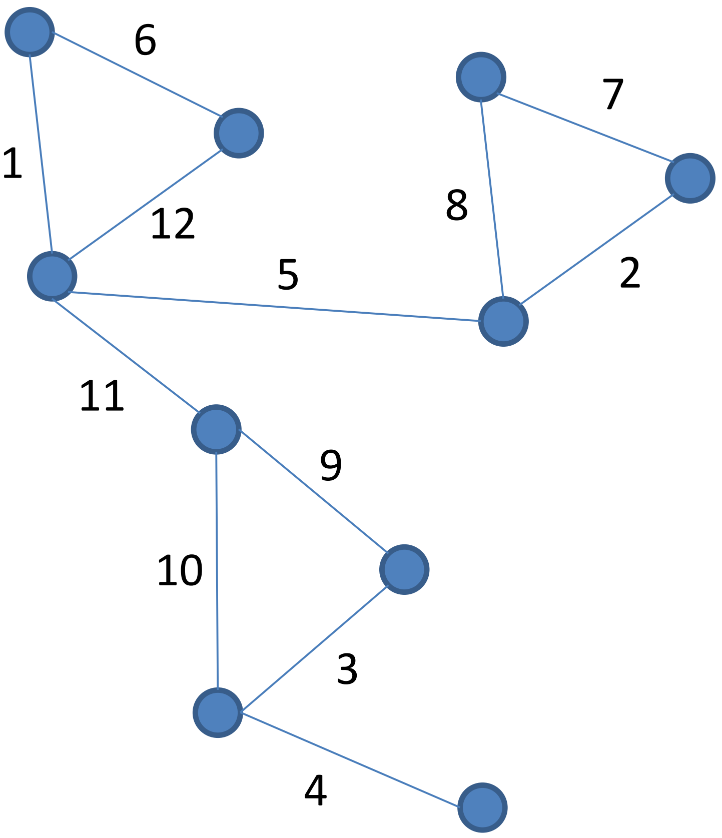
Example:

Size of matching: 5

Today’s Problem: Maximal Matching

Algorithm for maximal matching:

1) Assign each edge a random number. (Equivalent: choose a random permutation of the edges.)

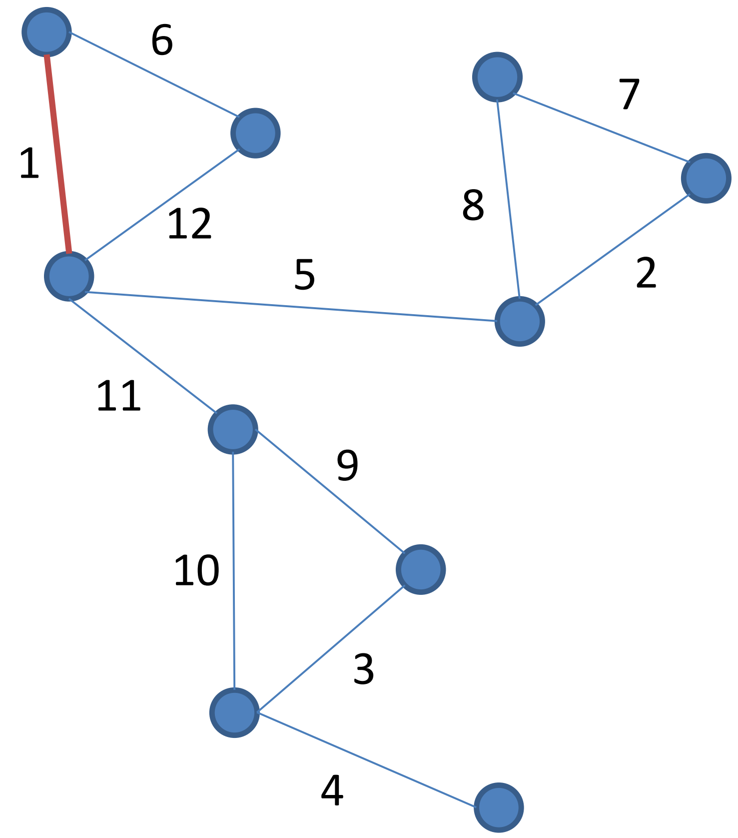


Today's Problem: Maximal Matching

Algorithm for maximal matching:

1) Assign each edge a random number. (Equivalent: choose a random permutation of the edges.)

2) Greedily, in order, try to add each edge to the matching.

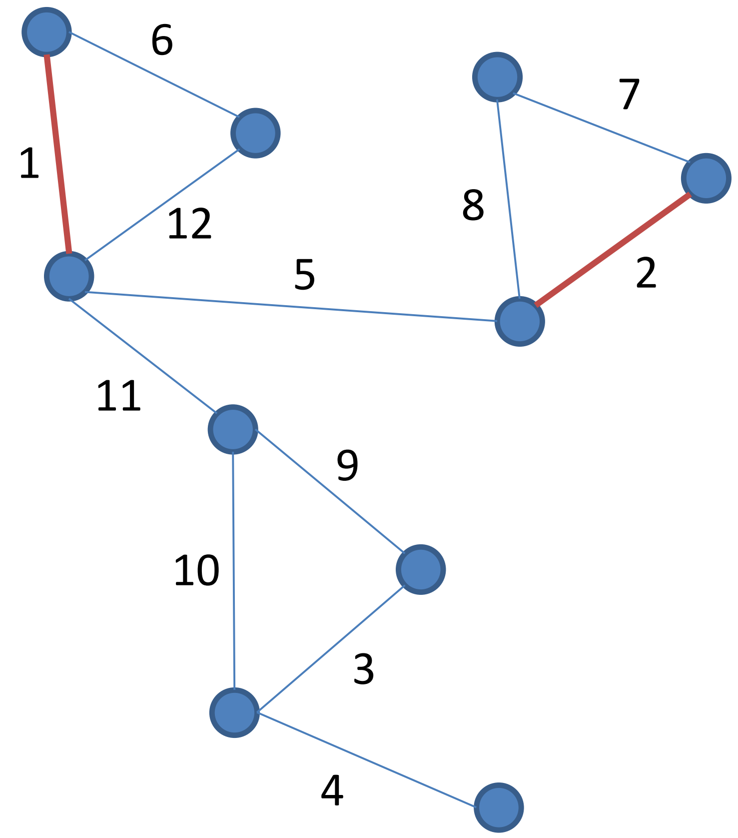


Today’s Problem: Maximal Matching

Algorithm for maximal matching:

1) Assign each edge a random number. (Equivalent: choose a random permutation of the edges.)

2) Greedily, in order, try to add each edge to the matching.



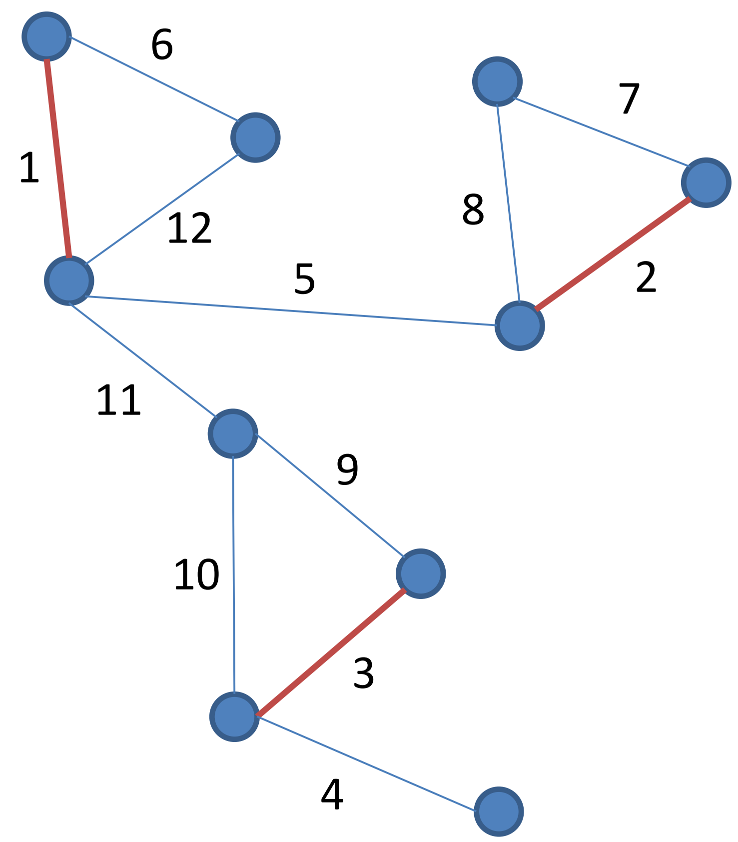
Today's Problem: Maximal Matching

Algorithm for maximal matching:

1) Assign each edge a random number. (Equivalent: choose a random permutation of the edges.)

2) Greedily, in order, try to add each edge to the matching.

➔ Each random permutation defines a unique maximal matching.



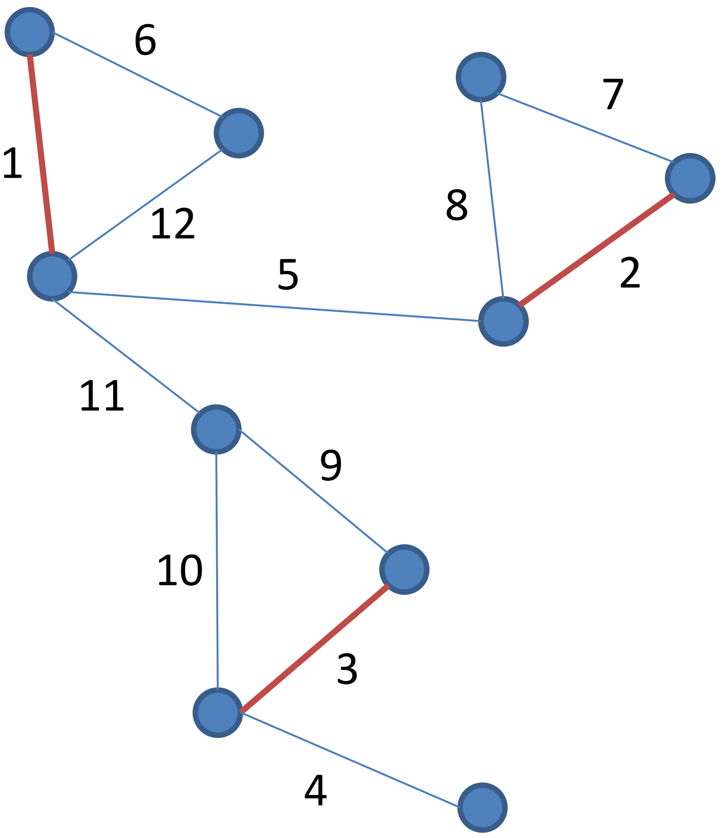
Today's Problem: Maximal Matching

To solve via sampling:

1) Choose a random permutation for the edges (e.g., a hash function).

2) Choose s edges at random.

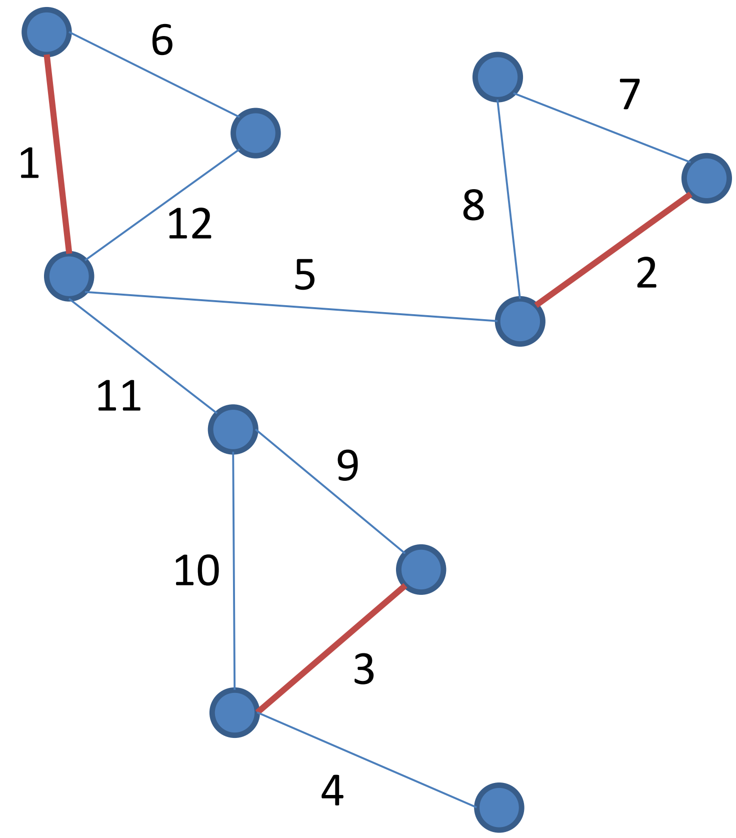
3) Decide if they are in the matching for the chosen permutation.



Today's Problem: Maximal Matching

To decide if an edge is in the matching:

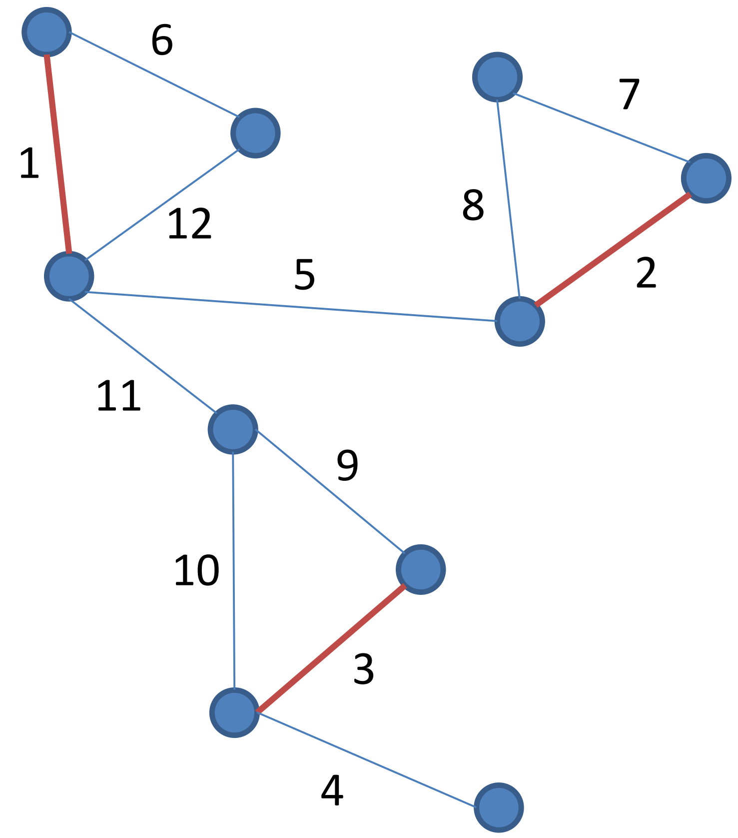
```
query(e):
    for all neighbors e' of e:
        if query(e') = true
            return false
    return true
```



Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if query(e') = true
            return false
    return true
```

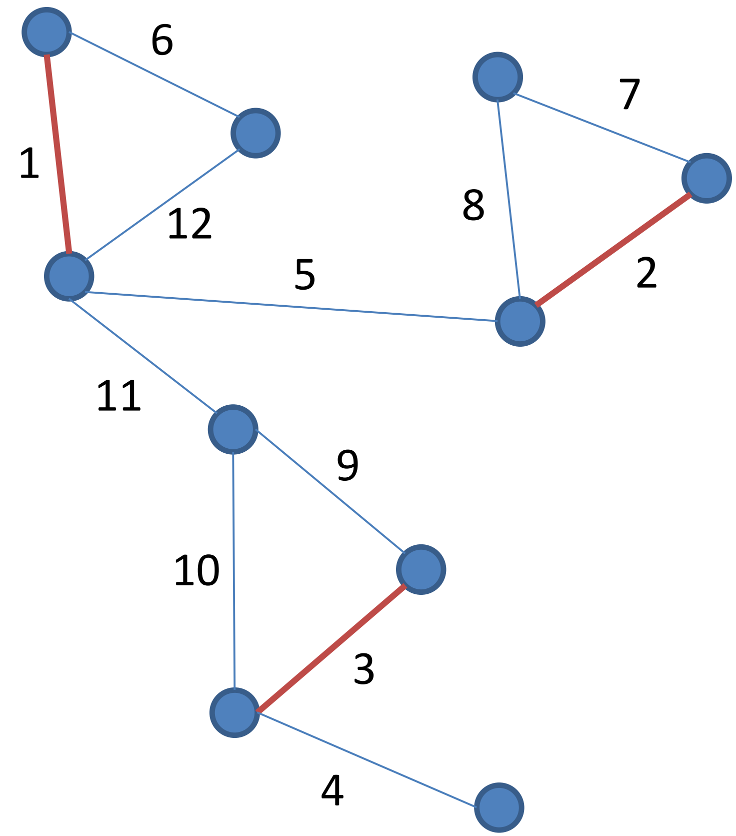


Oops… That doesn't exactly work!

Today’s Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e’ of e:
        if hash(e’) < hash(e)
            if query(e’) = true
                return false
    return true
```



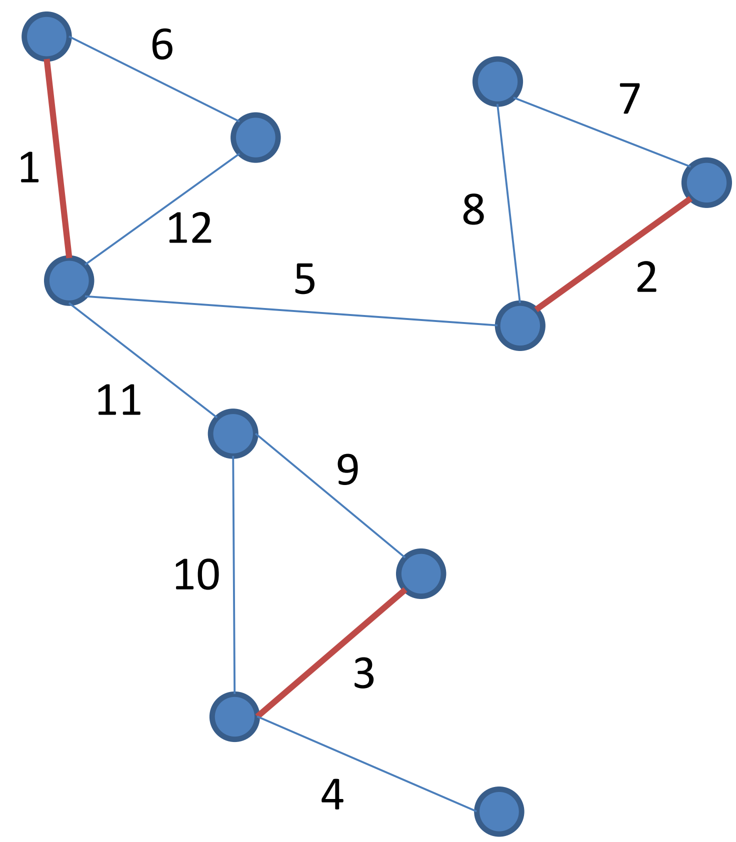
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today’s Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e’ of e:
        if hash(e’) < hash(e)
            if query(e’) = true
                return false
    return true
```



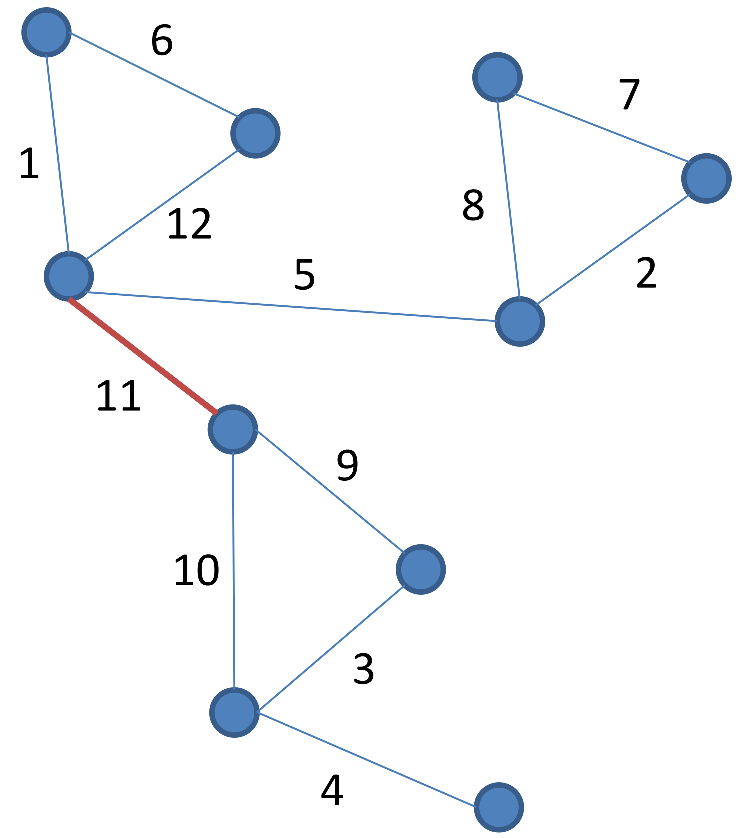
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



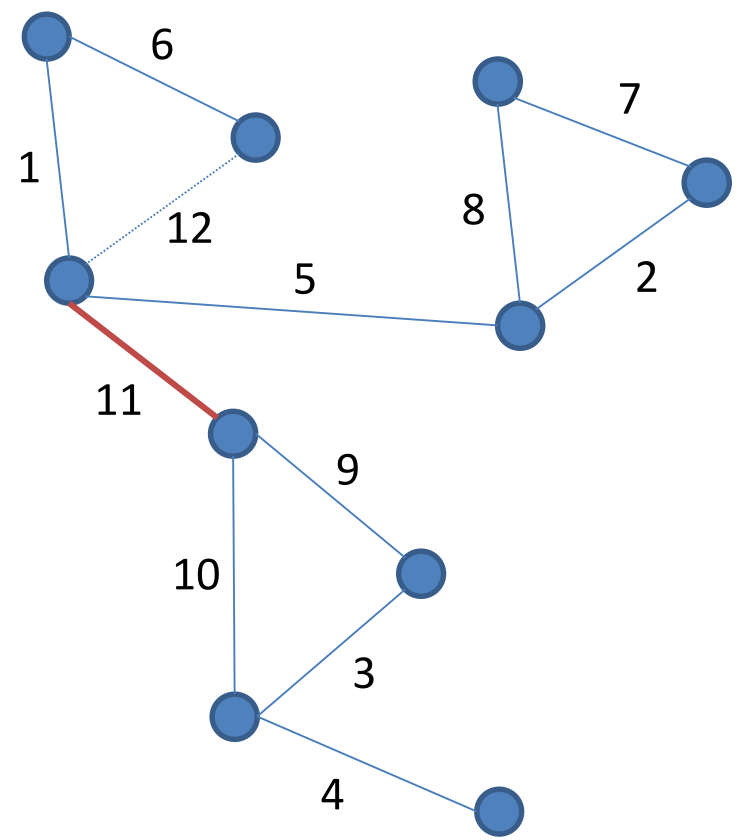
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



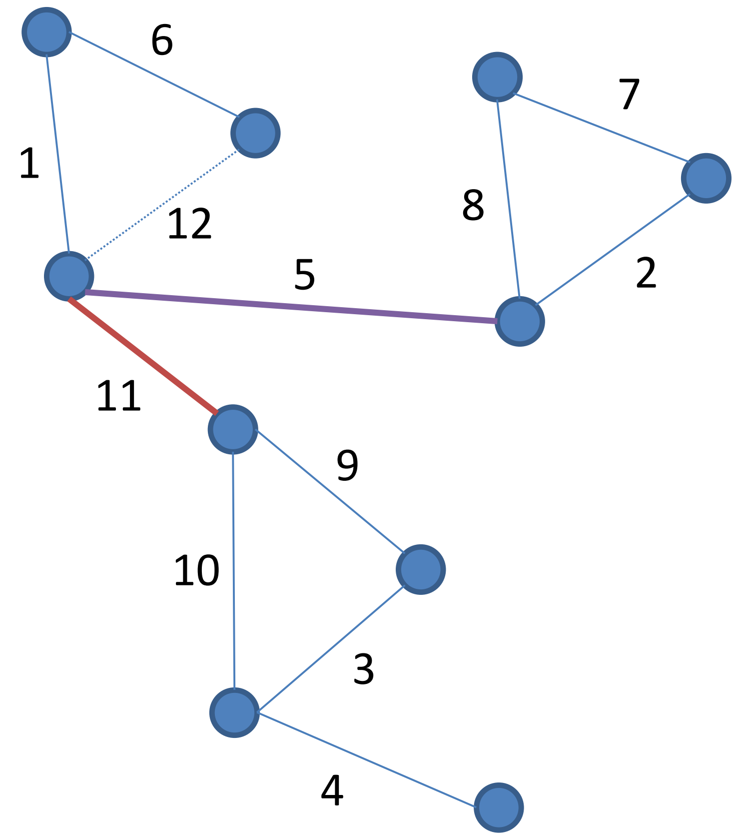
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



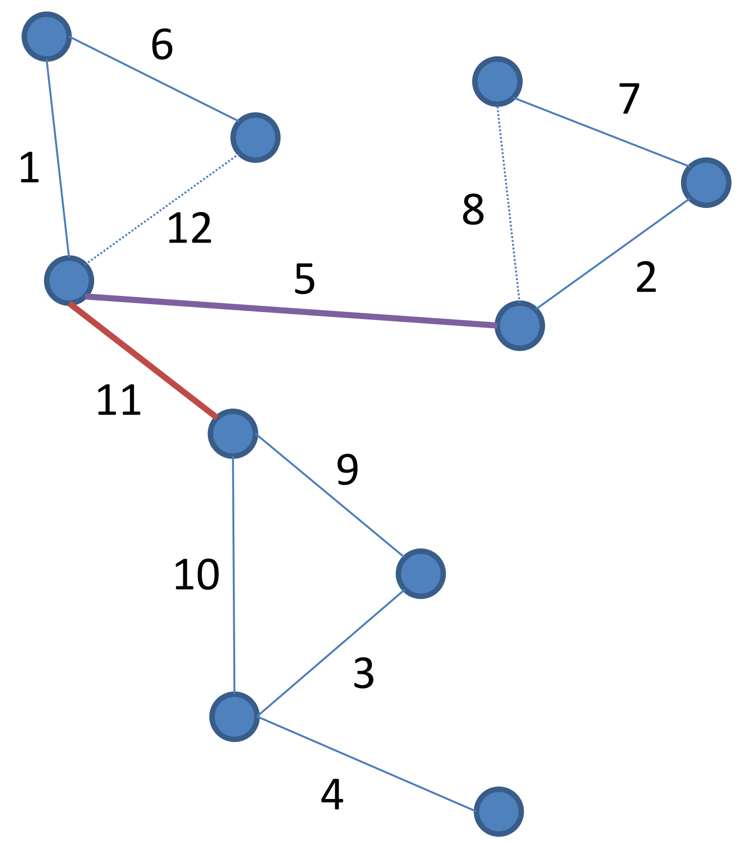
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



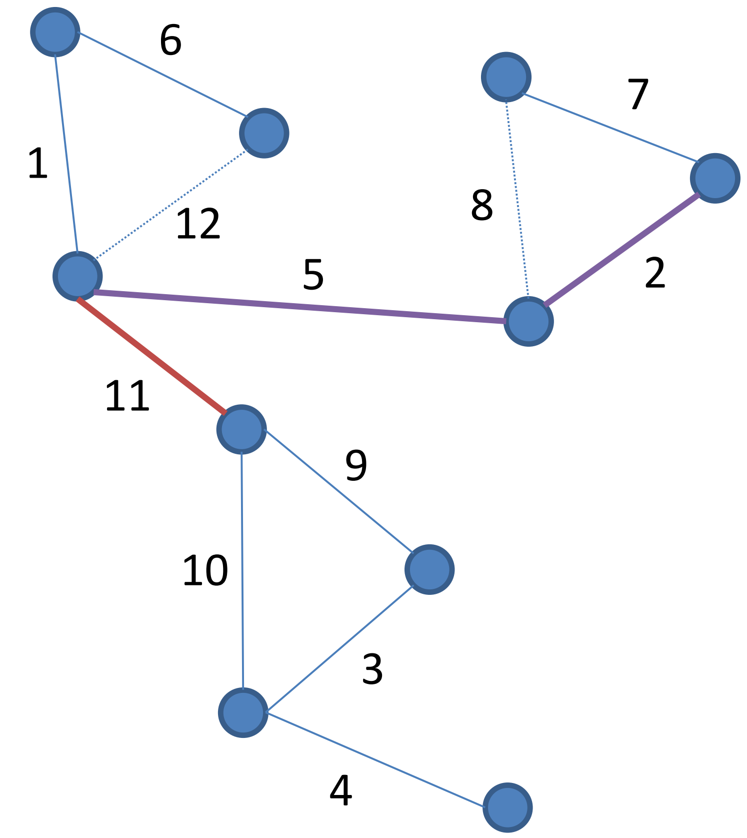
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



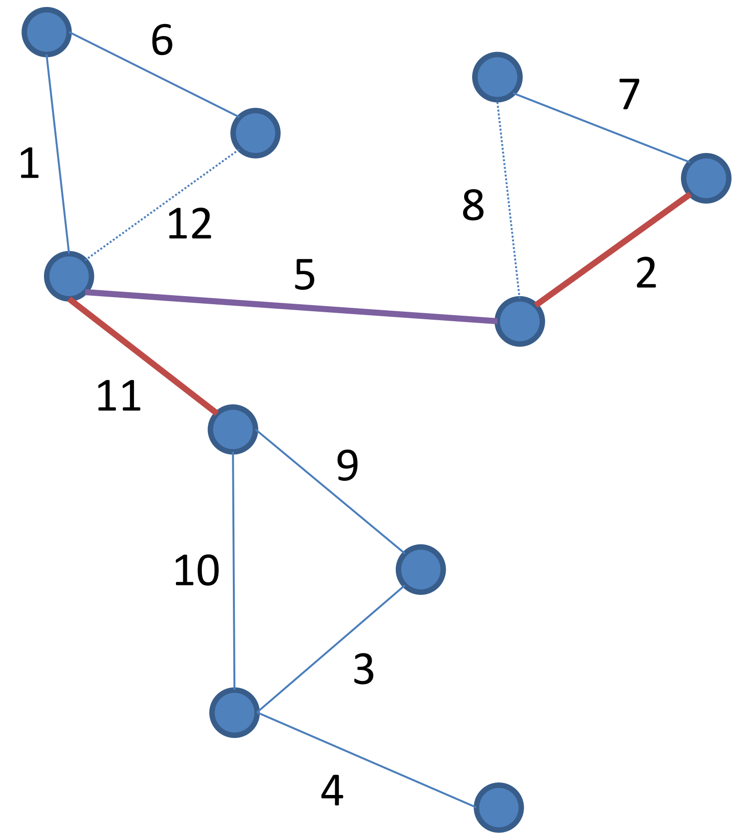
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today’s Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e’ of e:
        if hash(e’) < hash(e)
            if query(e’) = true
                return false
    return true
```

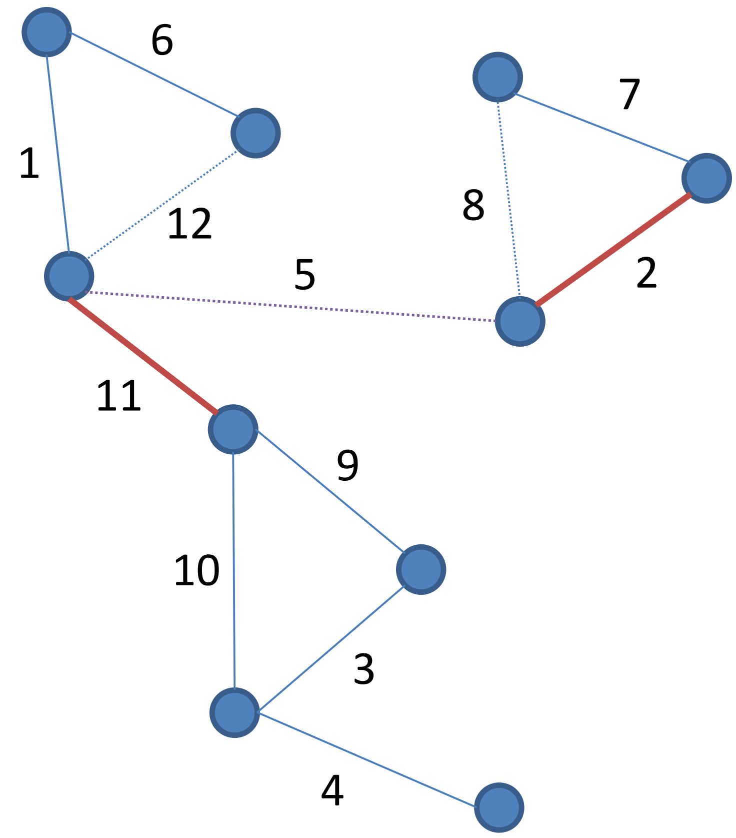


hash(e) returns the number chosen for edge e.
Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



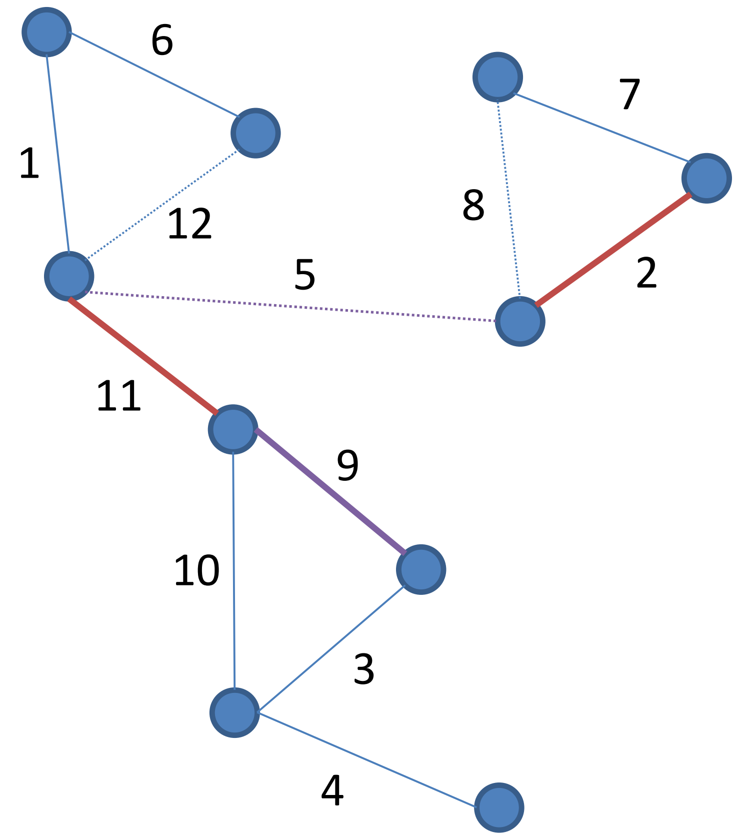
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



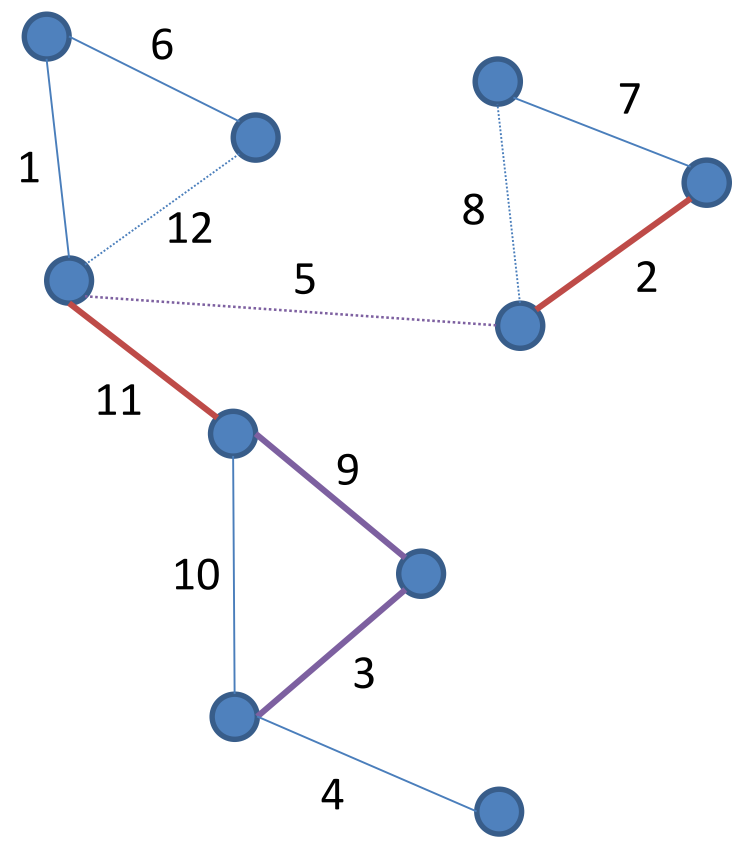
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



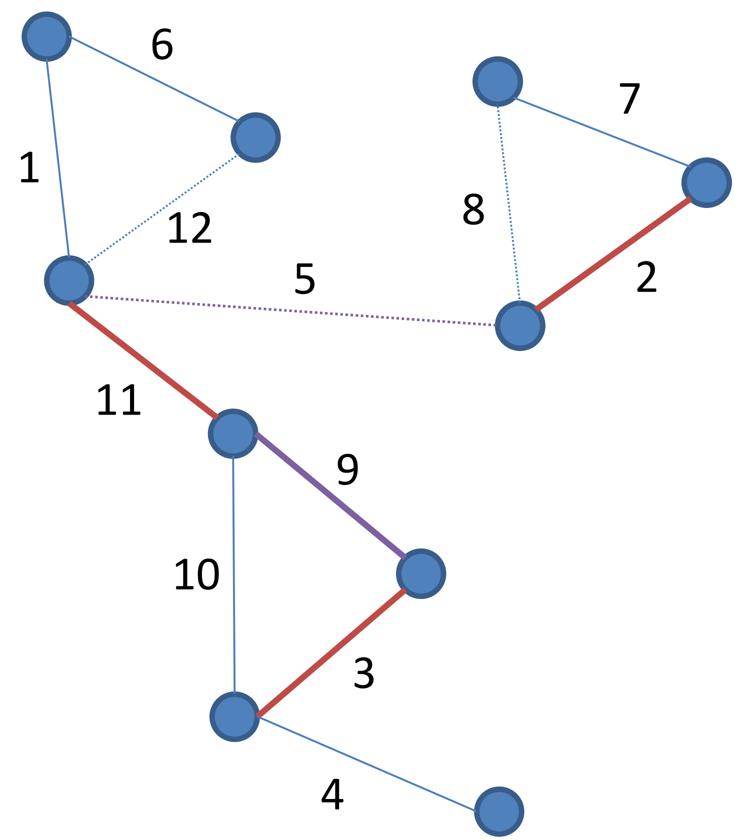
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



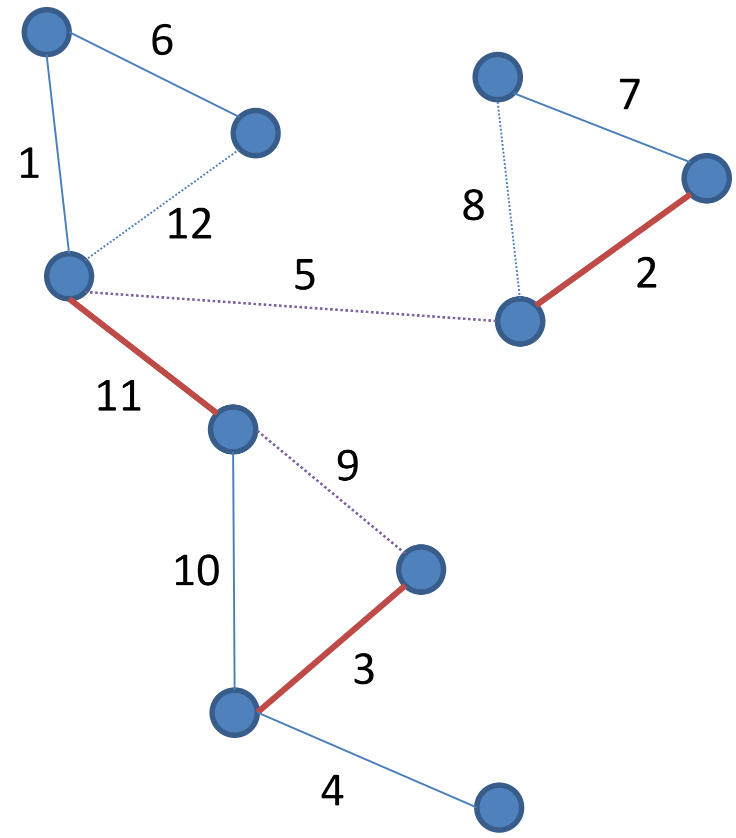
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



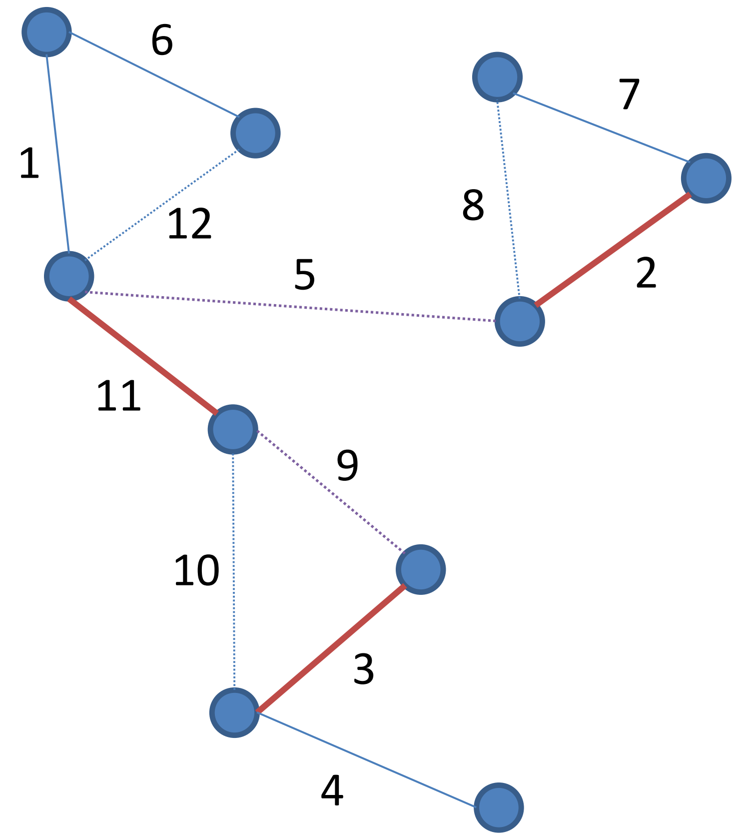
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



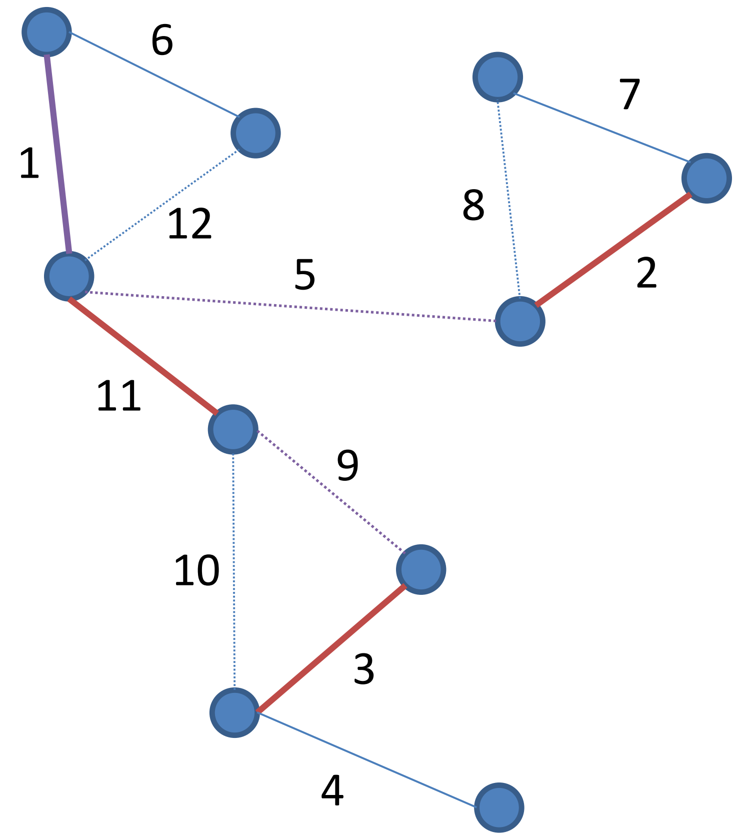
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



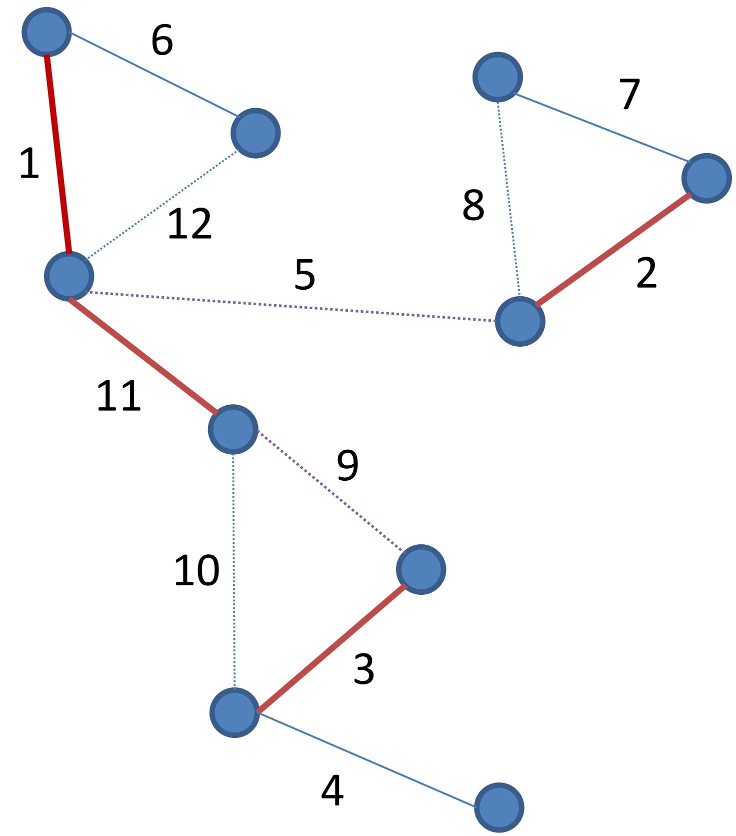
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



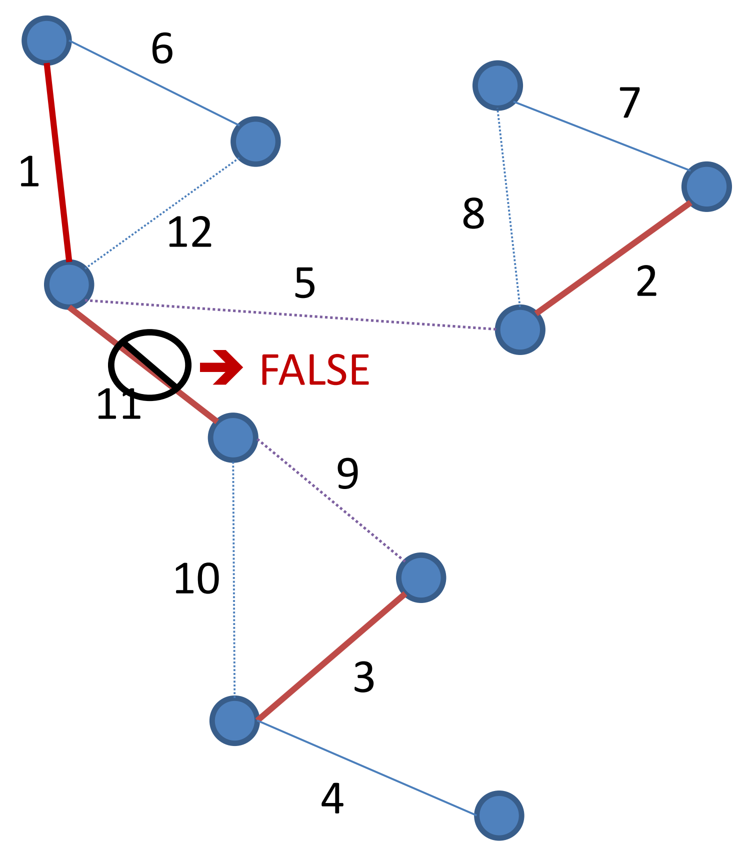
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

To decide if an edge is in the matching:

```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```



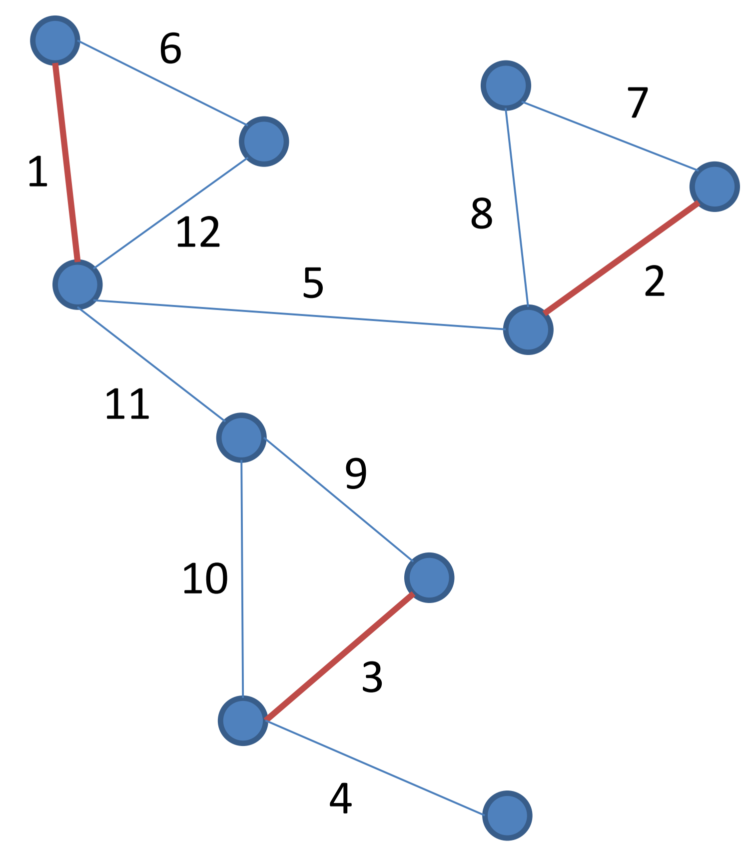
hash(e) returns the number chosen for edge e.

Only query *smaller* edges. *Larger* edges do not matter.

Today's Problem: Maximal Matching

Key question:

How expensive is a query?

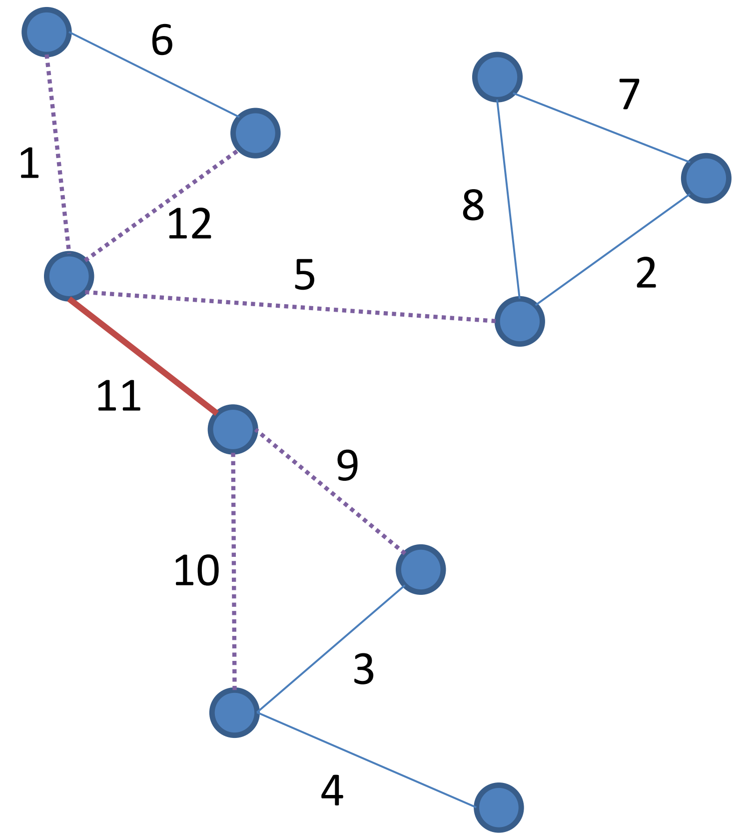



```
query(e):
    for all neighbors e' of e:
        if hash(e') < hash(e)
            if query(e') = true
                return false
    return true
```

Today's Problem: Maximal Matching

Some simple analysis:

If graph has maximum degree d, then there are at most $2d^k$ paths of length k starting from the query edge.

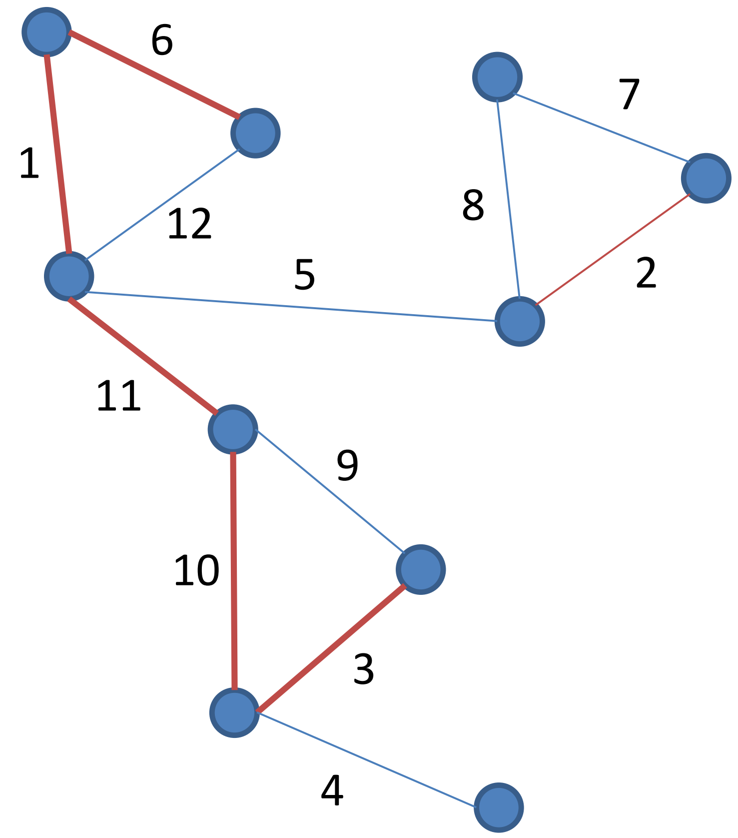


Today's Problem: Maximal Matching

Some simple analysis:

If graph has maximum degree d, then there are at most $2d^k$ paths of length k starting from the query edge.

Each path of length k defines a random permutation of hash values.



Permutation: [6,1,11,10,3]

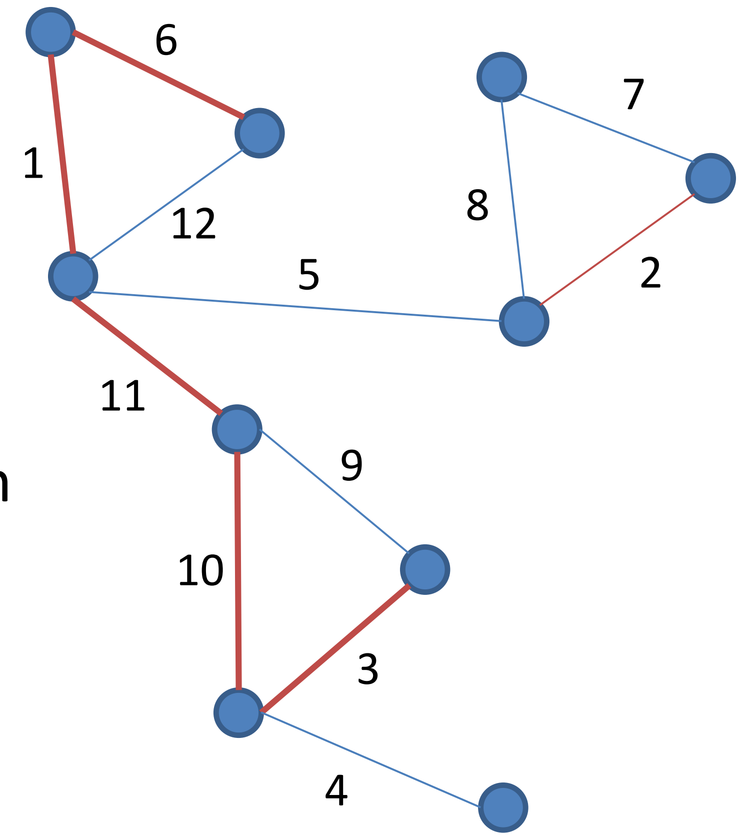
Today's Problem: Maximal Matching

Some simple analysis:

If graph has maximum degree d, then there are at most $2d^k$ paths of length k starting from the query edge.

Each path of length k defines a random permutation of hash values.

There are $k!$ possible permutations.



Permutation: [6,1,11,10,3]

Today's Problem: Maximal Matching

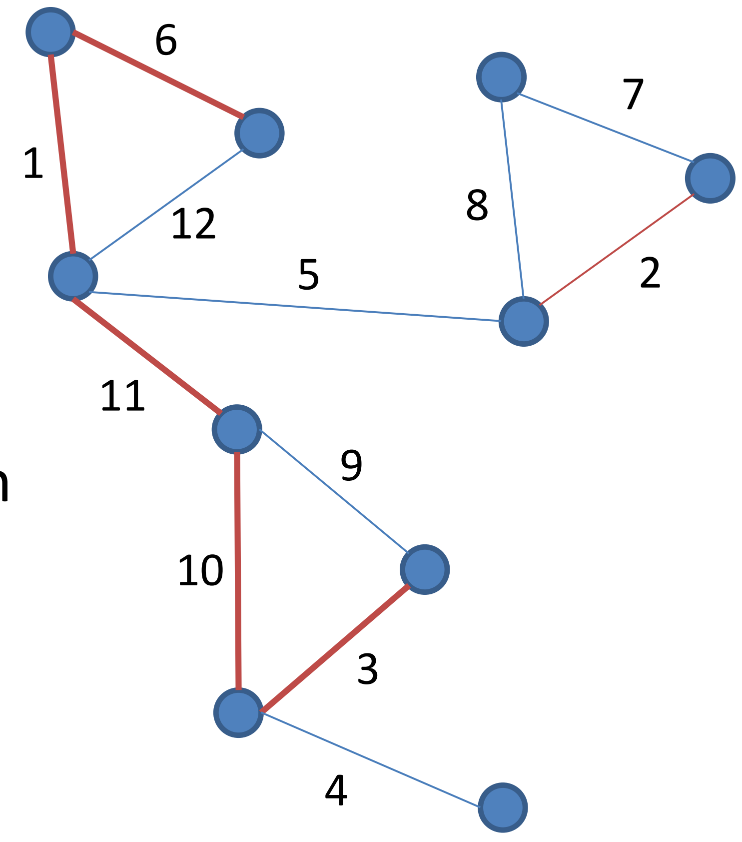
Some simple analysis:

If graph has maximum degree d, then there are at most $2d^k$ paths of length k starting from the query edge.

Each path of length k defines a random permutation of hash values.

There are k! possible permutations.

Pr[path is all decreasing] = 1/k!

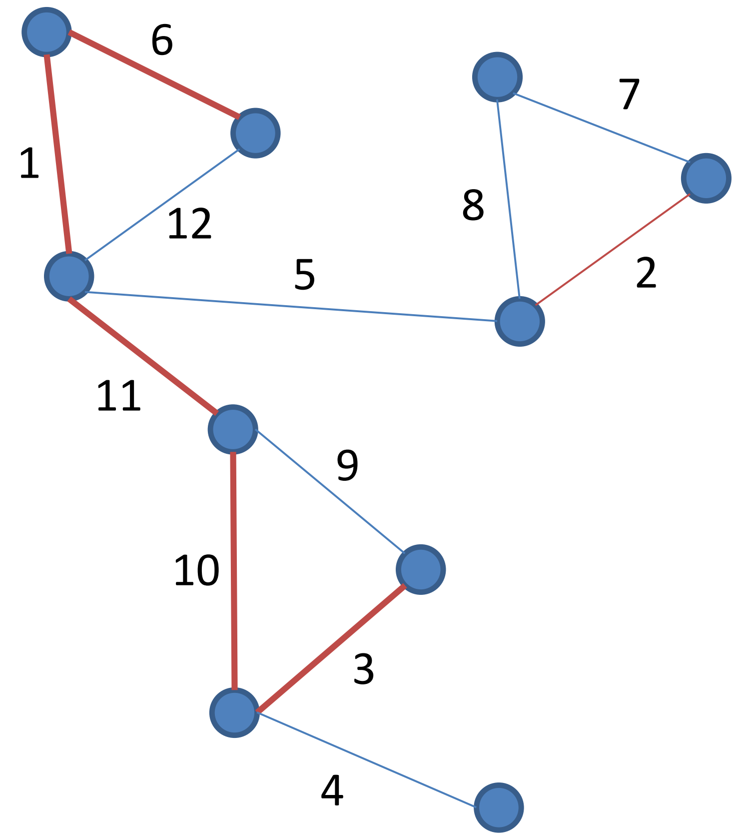


Permutation: [6,1,11,10,3]

Today's Problem: Maximal Matching

Conclusion:

The expected number of paths traversed of length k is at most: $\frac{d^k}{k!}$



Permutation: [6,1,11,10,3]

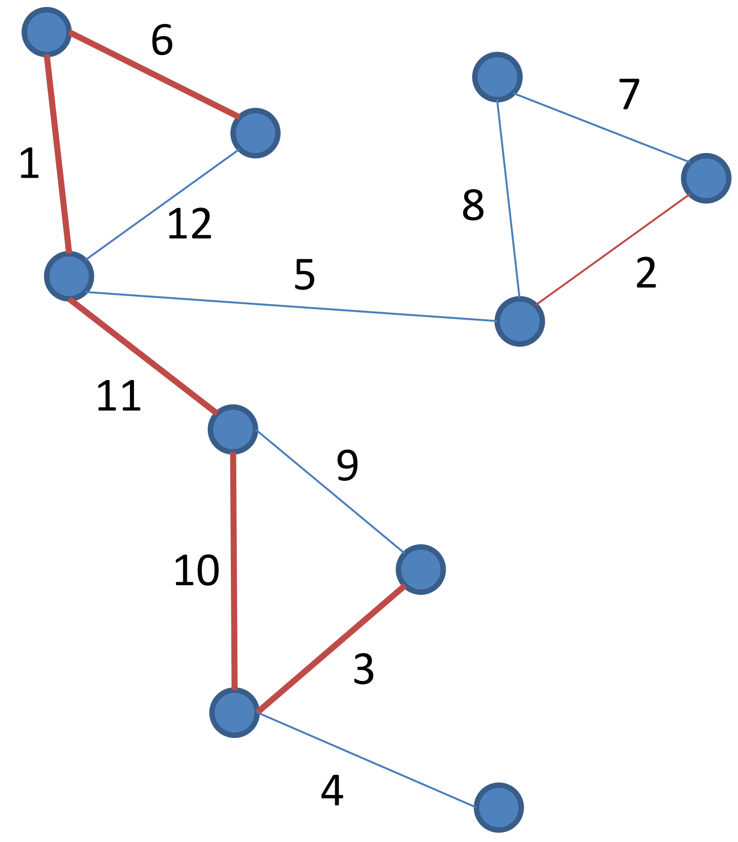
Today's Problem: Maximal Matching

Conclusion:

The expected number of paths traversed of length k is at most: $\frac{d^k}{k!}$

The expected total cost of a query is:

$$\sum_{k=1}^{\infty} \frac{d^k}{k!} = O\left(e^d\right)$$



Permutation: [6,1,11,10,3]

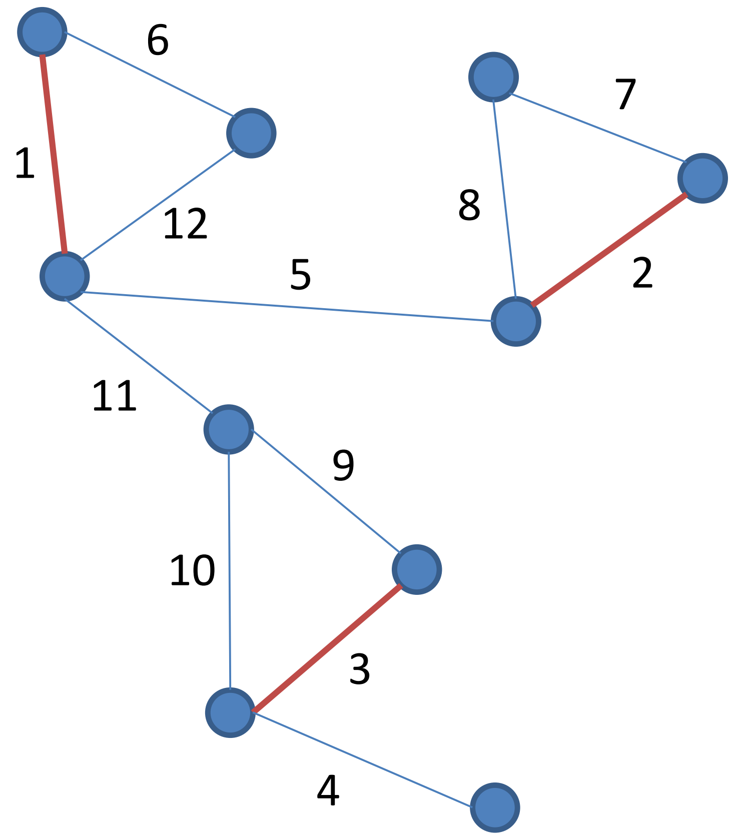
Today’s Problem: Maximal Matching

Key question:

How expensive is a query?

$$E[\text{cost}] = O(e^d)$$

```
query(e):
    for all neighbors e’ of e:
        if hash(e’) < hash(e)
            if query(e’) = true
                return false
    return true
```



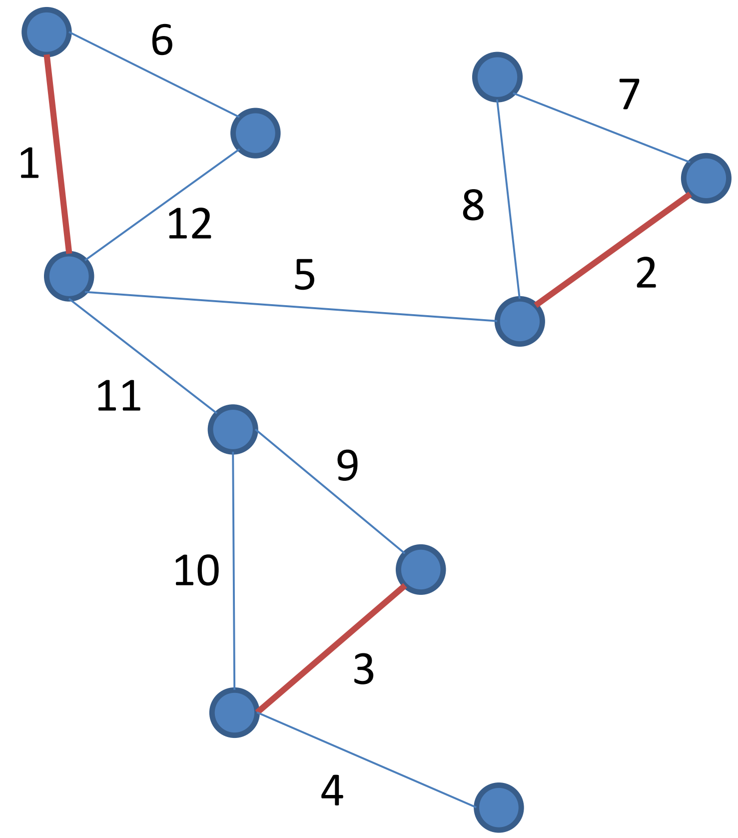
Today's Problem: Maximal Matching

To solve via sampling:

1) Choose a random permutation for the edges (e.g., a hash function).

2) Choose s edges at random.

3) Decide if they are in the matching for the chosen permutation via query operation.



Approximate Maximal Matching

MaxMatch-Sampling

```
sum = 0
for j = 1 to s:
    Choose edge e uniformly at random.
    if (query(e) = true) then
        sum = sum + 1
return m·(sum/s)
```

Approximate Maximal Matching

MaxMatch-Sampling

```
sum = 0
for j = 1 to s:
      Choose edge e uniformly at random.
      if (query(e) = true) then
          sum = sum + 1
return m·(sum/s)
```

Claim: returns size of maximal matching $\pm\ \varepsilon m$

Approximate Maximal Matching

MaxMatch-Sampling

```
sum = 0
for j = 1 to s:
      Choose edge e uniformly at random.
      if (query(e) = true) then
         sum = sum + 1
return m·(sum/s)
```

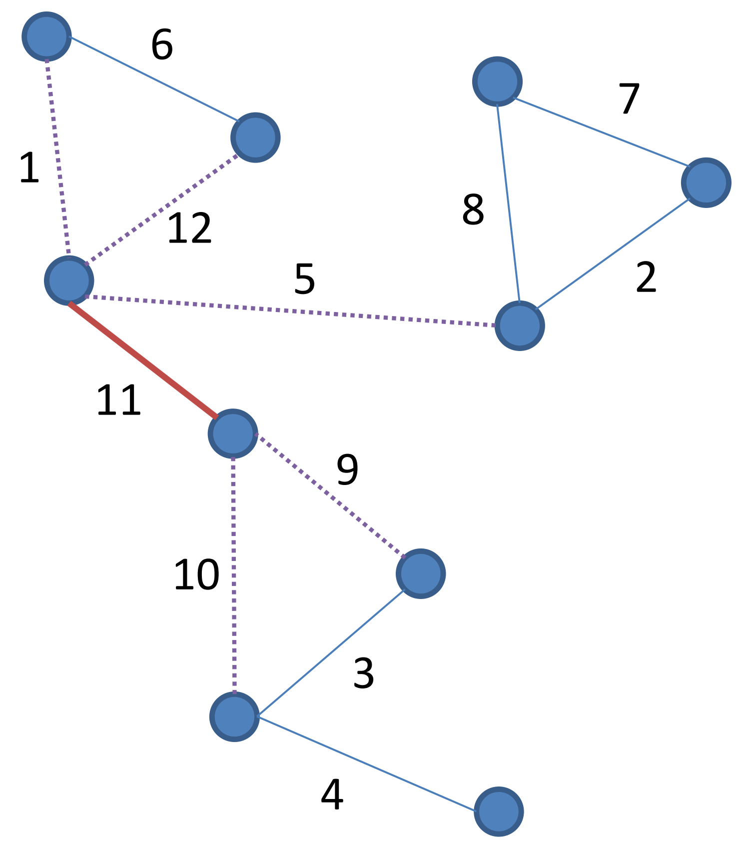
Claim: returns size of maximal matching $\pm\ \varepsilon m$

Claim: Runs in time $O(e^d / \varepsilon^2)$

Today's Problem: Maximal Matching

Two improvements:

1) Reduce error from $\pm\ \varepsilon m$ to $\pm\ \varepsilon n$.

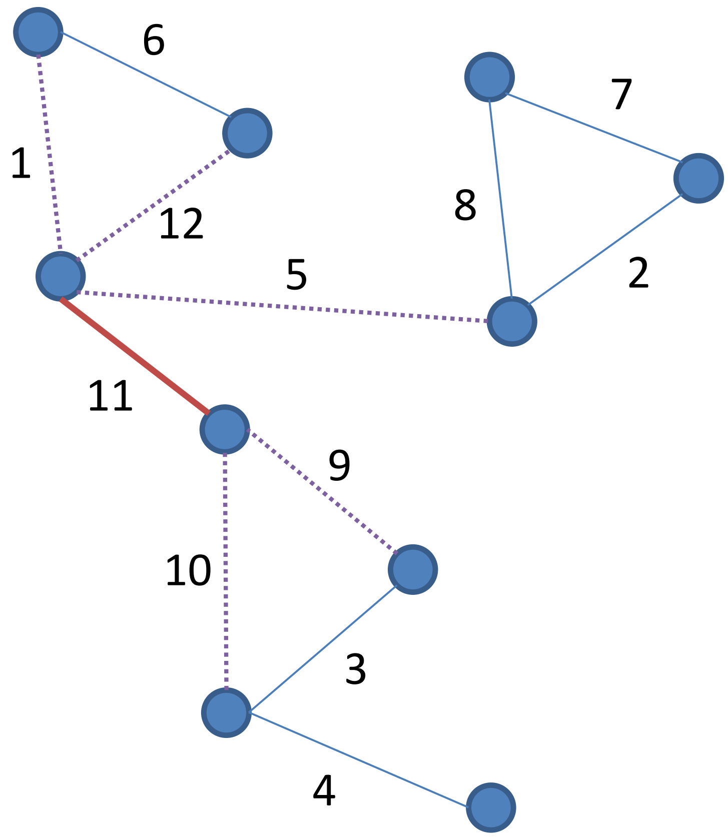


Today’s Problem: Maximal Matching

Two improvements:

1) Reduce error from ± εm to ± εn.

(Hint: each node is either matched or unmatched, and you can compute the size of the matching from the number of matched nodes.)

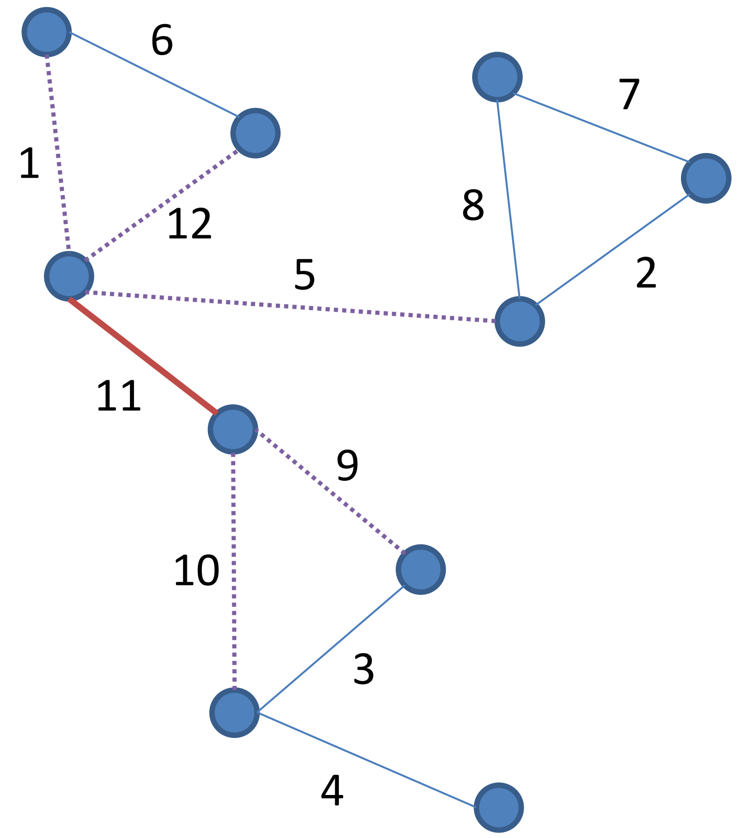


Today’s Problem: Maximal Matching

Two improvements:

1) Reduce error from $\pm\ \varepsilon m$ to $\pm\ \varepsilon n$.
(Hint: each node is either matched or unmatched, and you can compute the size of the matching from the number of matched nodes.)

2) Reduce the running time from exponential to $O(d^4/\ \varepsilon^2)$.



Today's Problem: Maximal Matching

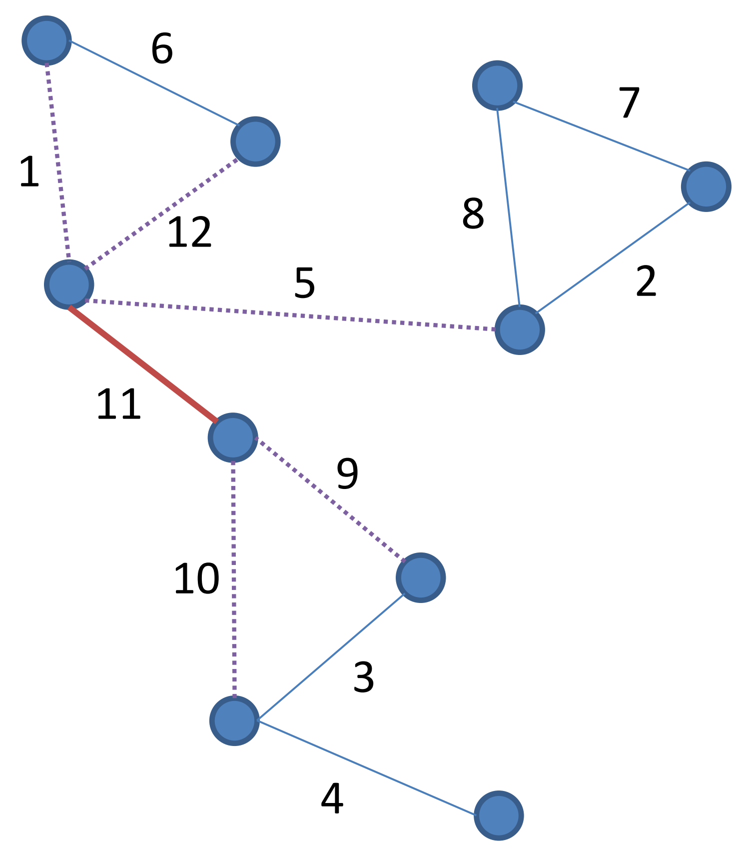
Two improvements:

1) Reduce error from $\pm\, \varepsilon m$ to $\pm\, \varepsilon n$.

(Hint: each node is either matched or unmatched, and you can compute the size of the matching from the number of matched nodes.)

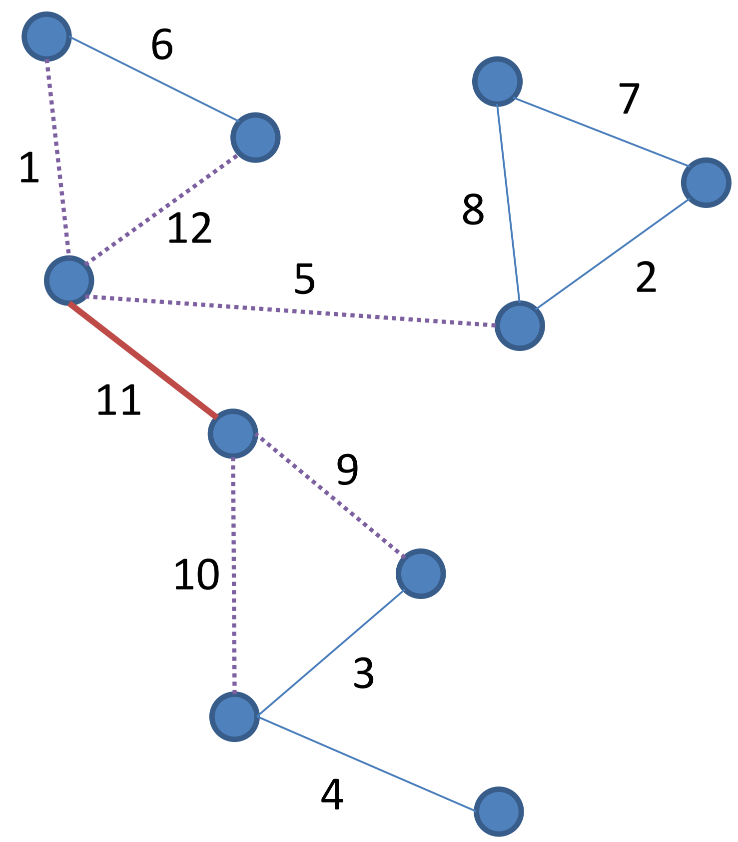
2) Reduce the running time from exponential to $O(d^4/\, \varepsilon^2)$.

(Hint: In query, explore neighboring edges in order of smallest weight first. Analysis is not simple!)



Questions to think about:

1) Show that the sampling algorithm works as claims (if the query operation is correct).



2) Reduce error from ± εm to ± εn.

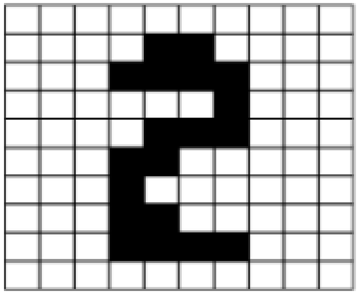
(Hint: each node is either matched or unmatched, and you can compute the size of the matching from the number of matched nodes.)

3) Can you find a multiplicative (instead of additive) approximation? Why not?

(Hint: Think about a graph where the maximal matching is very small.)

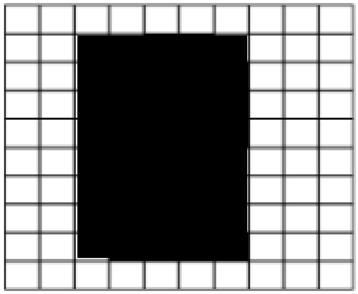
Two more questions:

1) Give an algorithm for deciding if the black pixels are connected or ε-far from connected in an n by n square of pixels.



connected

2) Give an algorithm for deciding if the black pixels are a rectangle or ε-far from a rectangle in an n by n square of pixels.



rectangle

Hint: imagine querying a grid of pixels distance εn apart.

Summary

Last Week:

Toy example 1: array all 0's?

- Gap-style question:
  All 0's or far from all 0's?

Toy example 2: Faction of 1's?

- Additive $\pm\,\varepsilon$ approximation
- Hoeffding Bound

Is the graph connected?

- Gap-style question.
- O(1) time algorithm.
- Correct with probability 2/3.

Today:

Number of connected components in a graph.

- Approximation algorithm.

Weight of MST

- Approximation algorithm.

Size of maximal matching

- Approximation algorithm.