Randomized Algorithm
(Lecture 2: Randomized Min_Cut)

Instructor: Dr. Guatam Das
Lecture note by Xin Jin
**Min_Cut Problem**

**Definition:**
Min_cut Problem is to find the minimum edge set C such that removing C disconnects the graph.

**Traditional Solution:**
Max-flow: The maximum amount of flow is equal to the capacity of a minimum cut.
Example of Min_Cut

e.g. Min_Cut = 2
**Intuition**

- Let a graph $G$ has $n$ nodes and size of $\text{min\_cut} = k$, that is $|C| = k$
  then :
  - degree for each node $\geq k$
  - total number of edges in $G \geq nk/2$
Randomized Min_Cut

Input: a graph G(V, E), |V| = n
Output: min_cut C

Repeat:
  Pick any edge uniformly at random, collapse it and remove self-loops

Until:
  |V| down to 2

*Running time is O(n-2)
Example of Randomized Min_Cut

min_cut = 2

min_cut = 4

Or maybe…
**Las Vegas VS Monte Carlo**

- **Las Vegas Algorithm:** It always produces the correct answer and the expected running time is finite (e.s.p. randomized quick sort)

- **Monte Carlo Algorithm:** It may produce incorrect answer but with bounded error probability (e.s.p. randomized min_cut)
Analysis

- Probability of the first edge $\notin C$
  \[ \text{Prob} = \frac{(kn/2 - k)}{(kn/2)} = \frac{(n-2)}{n} \]

- Probability of the second edge $\notin C$
  \[ \text{Prob} = \frac{(k(n-1)/2 - k)}{(k(n-1)/2)} = \frac{(n-3)}{(n-1)} \]

min\_cut
### Analysis

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Probability of avoiding C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \frac{n - 2}{n} )</td>
</tr>
<tr>
<td>2</td>
<td>( \frac{n - 3}{n - 1} )</td>
</tr>
<tr>
<td>3</td>
<td>( \frac{n - 4}{n - 2} )</td>
</tr>
<tr>
<td>4</td>
<td>( \frac{n - 5}{n - 3} )</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>n - 2</td>
<td>( \frac{1}{3} )</td>
</tr>
</tbody>
</table>

**Prob. Of outputting C:**

\[
Pr >= \frac{n - 2 \cdot n - 3 \cdot \ldots \cdot 2 \cdot 1}{n \cdot n - 1 \cdot 4 \cdot 3} = \frac{2}{n \cdot (n - 1)}
\]
Analysis

- Probability of getting a min_cut is at least \( \frac{2}{n(n-1)} \)

  Might look like small, but gets bigger after repeating the algorithm
e.s.p. If algorithm is running twice, probability of outputting C would be:

\[
Pr = 1 - \left( 1 - \frac{2}{n(n-1)} \right)^2
\]
Analysis

- Let $r$ be the number of running times of algorithm
  
  Total running time = $O(n \cdot r)$

  Probability of getting C:
  
  $Pr \approx 1 - \left(1 - \frac{2}{n^2}\right)^r$
Analysis

If \( r = \frac{n^2}{2} \)
then

\[
T(n) = O(n^*n^2 / 2) = O(n^3)
\]

\[
Pr = 1 - \left( 1 - \frac{2}{n^2} \right)^\frac{n^2}{2}
\]

\[
\approx 1 - \frac{1}{e}
\]