Graph Connectivity

- An undirected graph is connected if there is a path between every pair of vertices $x$ and $y$
- A connected component is a maximal connected subset of vertices

Connected Components

- Undirected Graphs

Computing Connected Components in $O(n+m)$ time

- A search algorithm from a vertex $v$ can find all vertices in $v$'s component
- While there is an unvisited vertex $v$, search from $v$ to find a new component

Directed Graphs

- A directed graph is strongly connected if for every pair of vertices $x$ and $y$, there is a path from $x$ to $y$, and there is a path from $y$ to $x$

Testing if a graph is strongly connected

- Pick a vertex $x$
  - $S_1 = \{ y \mid \text{path from } x \text{ to } y \}$
  - $S_2 = \{ y \mid \text{path from } y \text{ to } x \}$
  - If $|S_1| = n$ and $|S_2| = n$ then strongly connected
- Compute $S_2$ with a “Backwards BFS”
  - Reverse edges and compute a BFS
Strongly Connected Components

A set of vertices C is a strongly connected component if C is a maximal strongly connected subgraph.

Strongly connected components can be found in $O(n+m)$ time

- But it’s tricky!
- Simpler problem: given a vertex v, compute the vertices in v’s SCC in $O(n+m)$ time
  - $S_1 = \{ y \mid \text{path from v to y} \}$
  - $S_2 = \{ y \mid \text{path from y to v} \}$
  - SCC containing v is $S_1$ intersect $S_2$

Topological Sort

- Given a set of tasks with precedence constraints, find a linear order of the tasks

Find a topological order for the following graph

If a graph has a cycle, there is no topological sort

- Consider the first vertex on the cycle in the topological sort
- It must have an incoming edge

Definition: A graph is Acyclic if it has no cycles.

Lemma: If a (finite) graph is acyclic, it has a vertex with in-degree 0

- Proof:
  - Pick a vertex $v_1$, if it has in-degree 0 then done
  - If not, let $(v_2, v_1)$ be an edge, if $v_2$ has in-degree 0 then done
  - If not, let $(v_3, v_2)$ be an edge .
  - If this process continues for more than n steps, we have a repeated vertex, so we have a cycle
Topological Sort Algorithm

While there exists a vertex v with in-degree 0
Output vertex v
Delete the vertex v and all outgoing edges

Details for O(n+m) implementation

• Maintain a list of vertices of in-degree 0
• Each vertex keeps track of its in-degree
• Update in-degrees and list when edges are removed
• m edge removals at O(1) cost each

Random Graphs

• What is a random graph?
• Choose edges at random
• Interesting model of certain phenomena
• Mathematical study
• Useful inputs for graph algorithms

Model of Random Graphs

• Undirected Graphs
  – Random Graph with n vertices and m edges, G_m
  – Random Graph with n vertices where each edge has probability p, G_p
  – Models are similar when \( p = \frac{2m}{n(n-1)} \)

```csharp
for (int i = 0; i < n - 1; i++)
  for (int j = i + 1; j < n; j++)
    if (random.NextDouble() < p)
      AddEdge(i, j);
```
Stable Matching Results

• Averages of 5 runs
• Much better for M than W
• Why is it better for M?
• What is the growth of m-rank and w-rank as a function of n?

<table>
<thead>
<tr>
<th>n</th>
<th>m-rank</th>
<th>w-rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>7.02</td>
<td>94.95</td>
</tr>
<tr>
<td>500</td>
<td>8.57</td>
<td>58.28</td>
</tr>
<tr>
<td>500</td>
<td>6.42</td>
<td>75.87</td>
</tr>
<tr>
<td>500</td>
<td>8.06</td>
<td>87.73</td>
</tr>
<tr>
<td>1000</td>
<td>9.65</td>
<td>75.74</td>
</tr>
<tr>
<td>1000</td>
<td>6.00</td>
<td>146.93</td>
</tr>
<tr>
<td>1000</td>
<td>6.00</td>
<td>133.53</td>
</tr>
<tr>
<td>1000</td>
<td>7.54</td>
<td>58.94</td>
</tr>
<tr>
<td>1000</td>
<td>7.38</td>
<td>137.40</td>
</tr>
<tr>
<td>2000</td>
<td>7.24</td>
<td>204.01</td>
</tr>
<tr>
<td>2000</td>
<td>7.92</td>
<td>267.79</td>
</tr>
<tr>
<td>2000</td>
<td>7.92</td>
<td>267.76</td>
</tr>
<tr>
<td>2000</td>
<td>7.92</td>
<td>267.76</td>
</tr>
<tr>
<td>2000</td>
<td>7.92</td>
<td>267.76</td>
</tr>
</tbody>
</table>

Coupon Collector Problem

• n types of coupons
• Each round you receive a random coupon
• How many rounds until you have received all types of coupons
• \( p_i \) is the probability of getting a new coupon after \( i-1 \) have been collected
• \( t_i \) is the time to receive the \( i \)-th type of coupon after \( i-1 \) have been received

\[
p_i = \frac{n - (i - 1)}{n} = \frac{n - i + 1}{n}
\]

\( t_i \) has geometric distribution with expectation

\[
p = \frac{n}{n - i + 1}
\]

\[
E(T) = E(t_1 + t_2 + \cdots + t_n)
\]

where

\[
\begin{align*}
\frac{1}{p_1} &= n - 1 + 1 \\
\frac{1}{p_2} &= n - 2 + 1 + 1 \\
&\vdots \\
\frac{1}{p_n} &= n - n + 1 + \cdots + 1
\end{align*}
\]

\[
= n \cdot H_n
\]

\[
E(T) = n \cdot H_n = n \log n + \gamma n + \frac{1}{2} + O(1/n)
\]

Stable Matching and Coupon Collecting

• Assume random preference lists
• Runtime of algorithm determined by number of proposals until all w’s are matched
• Each proposal can be viewed as asking a random w
• Number of proposals corresponds to number of steps in coupon collector problem

1There are some technicalities here that are being ignored