CSE 417 Algorithms and Complexity

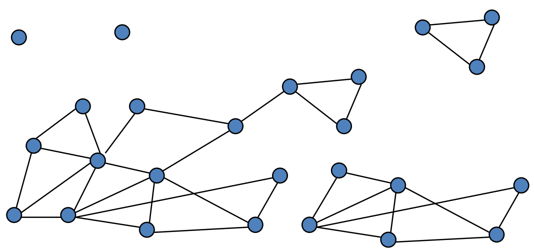
Graph Algorithms
Autumn 2020
Lecture 7

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

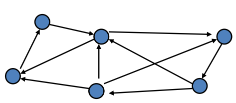


Computing Connected Components in O(n+m) time

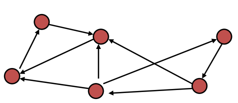
- A search algorithm from a vertex v can find all vertices in v's component
- While there is an unvisited vertex v, search from v to find a new component

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



Strongly Connected



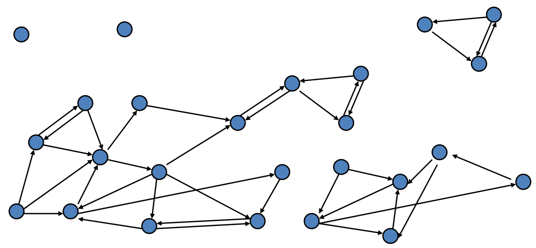
Not Strongly Connected

Testing if a graph is strongly connected

- Pick a vertex x
  - S_1 = { y | path from x to y }
  - S_2 = { y | path from y 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



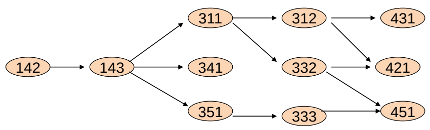
Strongly connected components can be found in O(n+m) time

- But it's tricky!
- Simpler problem: given a vertex v, compute the vertices in v's scc in O(n+m) time

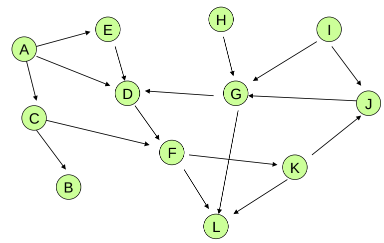
- S_1 = { y | path from v to y }
- S_2 = { y | path from y to v}
- Scc containing v is S_1 Intersect S_2

Topological Sort

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

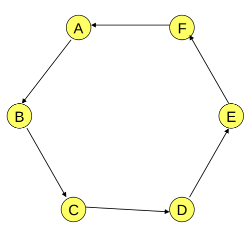


Find a topological order for the following graph



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

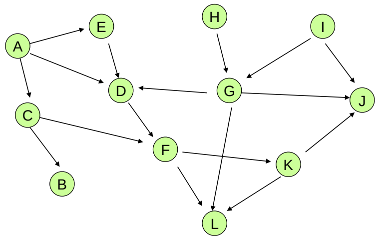
- Proof:
  – Pick a vertex v_1, if it has in-degree 0 then done
  – If not, let (v_2, v_1) be an edge, if v_2 has in-degree 0 then done
  – If not, let (v_3, v_2) be an edge . . .
  – If this process continues for more than n steps, we have a repeated vertex, so we have a cycle

Topological Sort Algorithm

While there exists a vertex v with in-degree 0

Output vertex v

Delete the vertex v and all out going edges

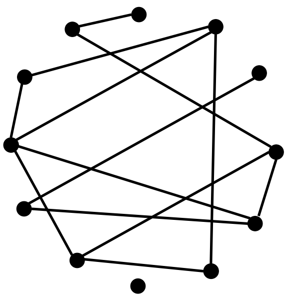


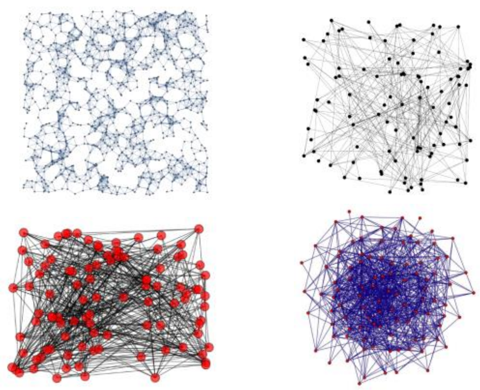
Details for O(n+m) implementation

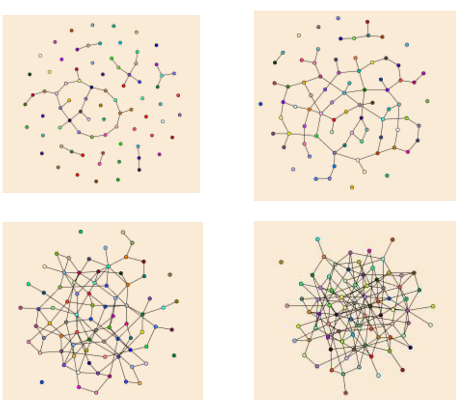
- Maintain a list of vertices of in-degree 0
- Each vertex keeps track of its in-degree
- Update in-degrees and list when edges are removed
- m edge removals at O(1) cost each

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 = 2m / (n * (n – 1))

```
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?

n	m-rank	w-rank
500	5.10	98.05
500	7.52	66.95
500	8.57	58.18
500	6.32	75.87
500	5.25	90.73
500	6.55	77.95
1000	6.80	146.93
1000	6.50	154.71
1000	7.14	133.53
1000	7.44	128.96
1000	7.36	137.85
1000	7.04	140.40
2000	7.83	257.79
2000	7.50	263.78
2000	11.42	175.17
2000	7.16	274.76
2000	7.54	261.60
2000	8.29	246.62

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

$$\frac{1}{p_i} = \frac{n}{n-i+1}$$

$$\begin{aligned} \mathrm{E}(T) &= \mathrm{E}(t_1 + t_2 + \cdots + t_n) \\ &= \mathrm{E}(t_1) + \mathrm{E}(t_2) + \cdots + \mathrm{E}(t_n) \\ &= \frac{1}{p_1} + \frac{1}{p_2} + \cdots + \frac{1}{p_n} \\ &= \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} \\ &= n \cdot \left(\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}\right) \\ &= n \cdot H_n. \end{aligned}$$

$$\mathrm{E}(T) = n \cdot H_n = n \log n + \gamma n + \frac{1}{2} + O(1/n).$$

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[1] as asking a random w
- Number of proposals corresponds to number of steps in coupon collector problem

[1]There are some technicalities here that are being ignored