Warm-up

Primality Testing

for $i = 2$ to $n/2$ **do**
  if i divides n evenly **then** } $O(1)$ } $O(n)$
    return false
return true

pseudo-polynomial

1. What is the runtime of this algorithm in terms of n, the input value?
   function of input size
2. Is this a polynomial time algorithm, as per the definition from lecture and from the reading? Why or why not?

No: if input is b-bit int,
time is $O(2^b)$

Total probability question

Give an $O(n^3)$ time algorithm to compute a 2D-array X, where $X[i,j]$ is the probability that some integer in the range $[i,j]$ (inclusive) is chosen. You may assume that arithmetic operations take $O(1)$ time each.

for i = 1 to n
  for j = i+1 to n
    X[i,j] = P_i
    for k = i+1 to j
      X[i,j] += P_k

$\sum_{i=1}^{n}\left(\sum_{j=i+1}^{n} j-i\right)$

$j-i$

Q 2(b):

// $X[i,j] = X[i,j-1] + P_j$

$O(n^2)$
for i = 1 to n

$O(n)$
X[i,i] = P_i
for j = i+1 to n
X[i,j] = X[i,j-1] + P_j

Hotels cost/distance question

Devise an algorithm which takes as input A and n, and outputs the resulting set. Your algorithm does not need to be particularly efficient.

$O(n \lg n)$ sort by cost

$O(n^2)$
for i := 1 to n — $\binom{n}{2}$ or ${}_nC_2$ or $C(n,2)$
  $O(n)$ for j := i+1 to n
    $O(1)$ if $dist_j > dist_i$
      remove hotel j

return all non-removed

Hotels cost/distance question

Analyze the worse-case runtime of your algorithm using Θ-notation.

Hotels cost/distance question

Assuming each operation takes 10^{-11} seconds, what is the computing time used by the algorithm, if $n = 250$?. *You may define what counts as a bit operation if it seems ambiguous. Clearly state all assumptions made.* $O(n^2) + n\lg n$

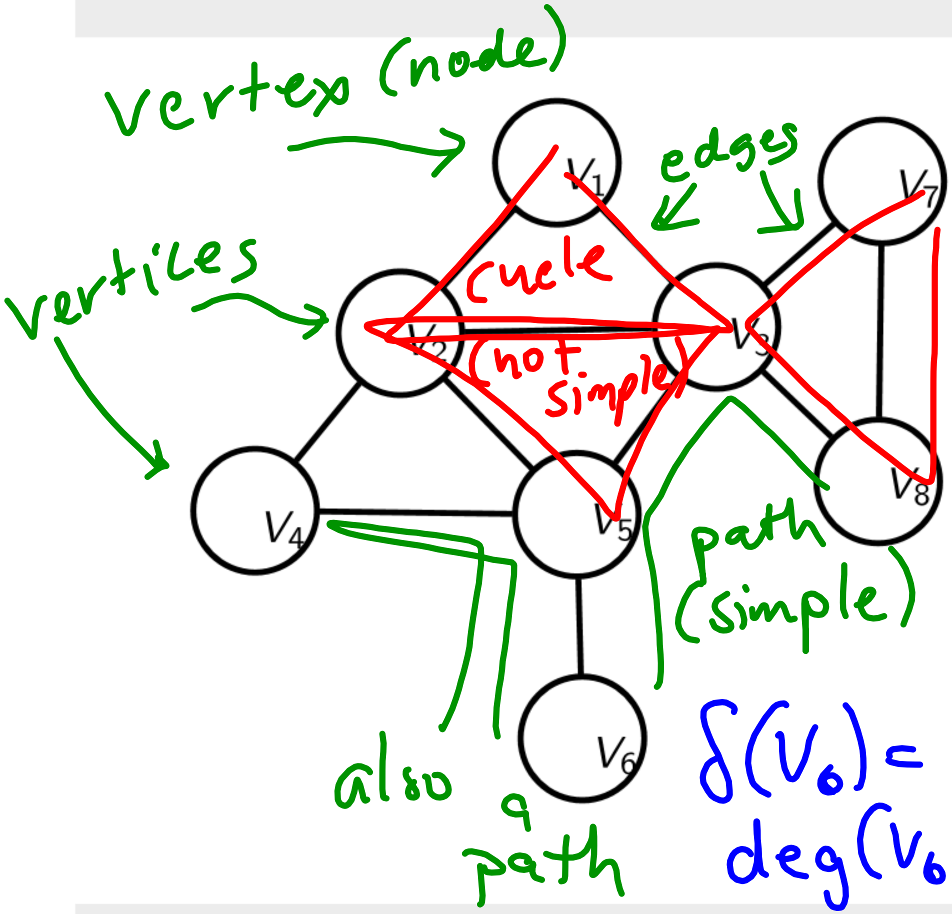
lead const estimate?

$\approx 3n^2 + n\log n$

Hotels cost/distance question

Do you believe your algorithm has obtained the best possible asymptotic runtime? Explain your reasoning.

CSCI 570 Summer 2016
Graph Fundamentals



Euler cycle

$\delta(v_6) =$

$\deg(v_6) =$

Tree

- n-1 edges
- connected
- no cycles

Question

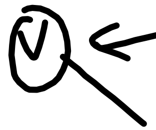
30

You have a graph with 10 vertices, and each node has degree 6.
How many edges are there?

"count":

```
count = 0
for each vertex v
    count += deg(v)
return count/2
```

Question



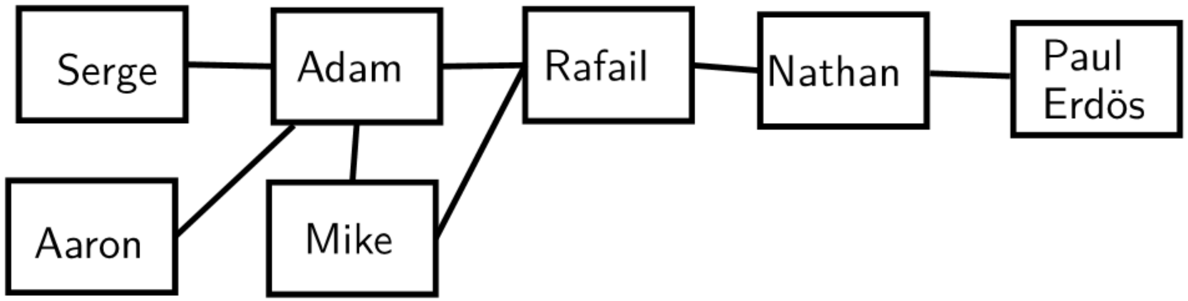
not ok

Can you draw a graph with 5 vertices, each with degree 1?

$$\sum_{v \in V} \delta(v) \text{ must be even}$$

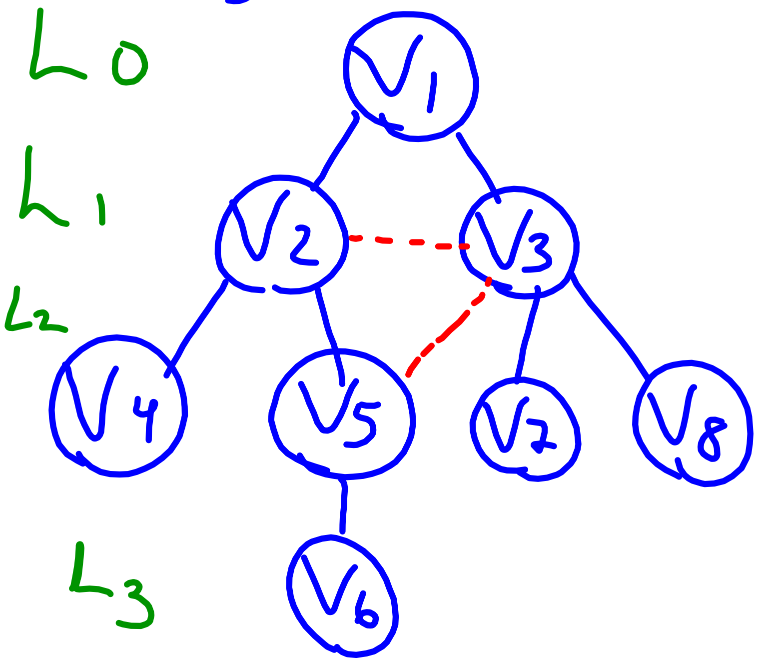
$$= 2 \cdot |E| = 2m$$

Collaboration Graph



Breadth-First Search

BFS from V_1 (graph on handout front)



BFS

Implementing Breadth-First Search

BFS(G, s) : $\Theta(n+m)$

Assumes graph is in memory

Set discovered[s] = true

Set discovered[v] = false for all other v

L[0] ← {s} ; i ← 0

while L[i] is not empty **do**

Make L[$i+1$] as empty list

for all vertices $u \in$ L[i] **do**

for all edges (u, v) **do**

if discovered[v] = false **then**

discovered[v] ← true

Add v to list L[$i+1$]

$i \leftarrow i+1$

$O(n)$ times

each edge twice

$O(m)$ TOTAL

$O(1)$

$\cancel{O(m)}$

$O(n \cdot m)$

Representing a Graph

Adjacency List.

Graph has a list of vertices

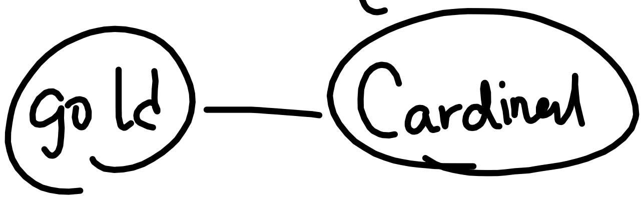
Each vertex has a list of adj. vertices

Adjacency Matrix boolean 2D array.

$$M[v_i, v_j] = ?\ \exists (v_i, v_j)$$

Bipartite Graphs

2-colorable
assign ea. vertex one of 2 colors
so each edge is dichromatic
(both ends diff color)



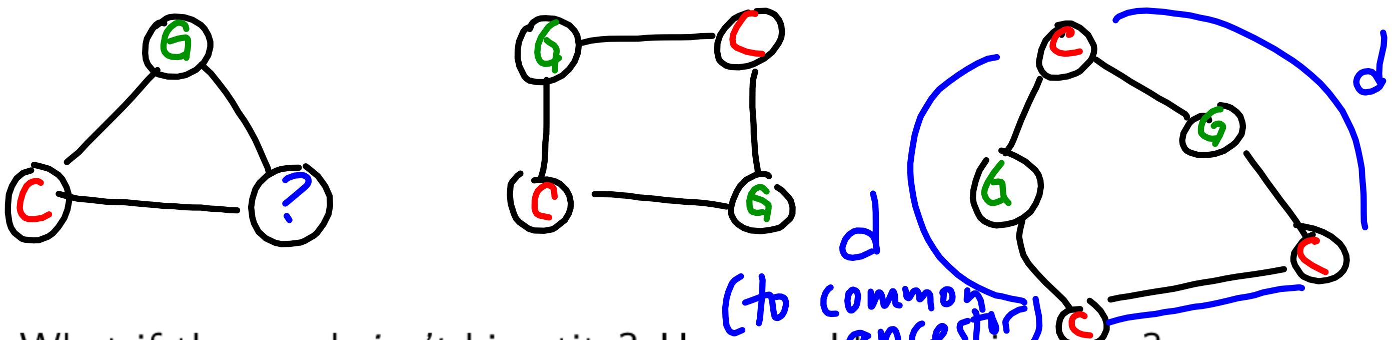
"Certificate" : coloring of vertices

Suppose a graph is bipartite: how can I offer proof to you that it is?

Suppose I offer that proof. How can you check that my proof is valid?

1. Check only 2 colors
2. each vertex exactly one color
3. for each $e=(u,v)$
   if $color(u) == color(v)$
   reject
4. accept

Bipartite Graphs



What if the graph *isn't* bipartite? How would I convince you?

G is bipartite iff G has no odd len cycles

Pf. ← pretty clear?

Bipartite Graphs

True or False: Every tree is bipartite.

→ no odd len cycles
(no cycles at all)

True or False: Every graph with a cycle in it is *not* bipartite.

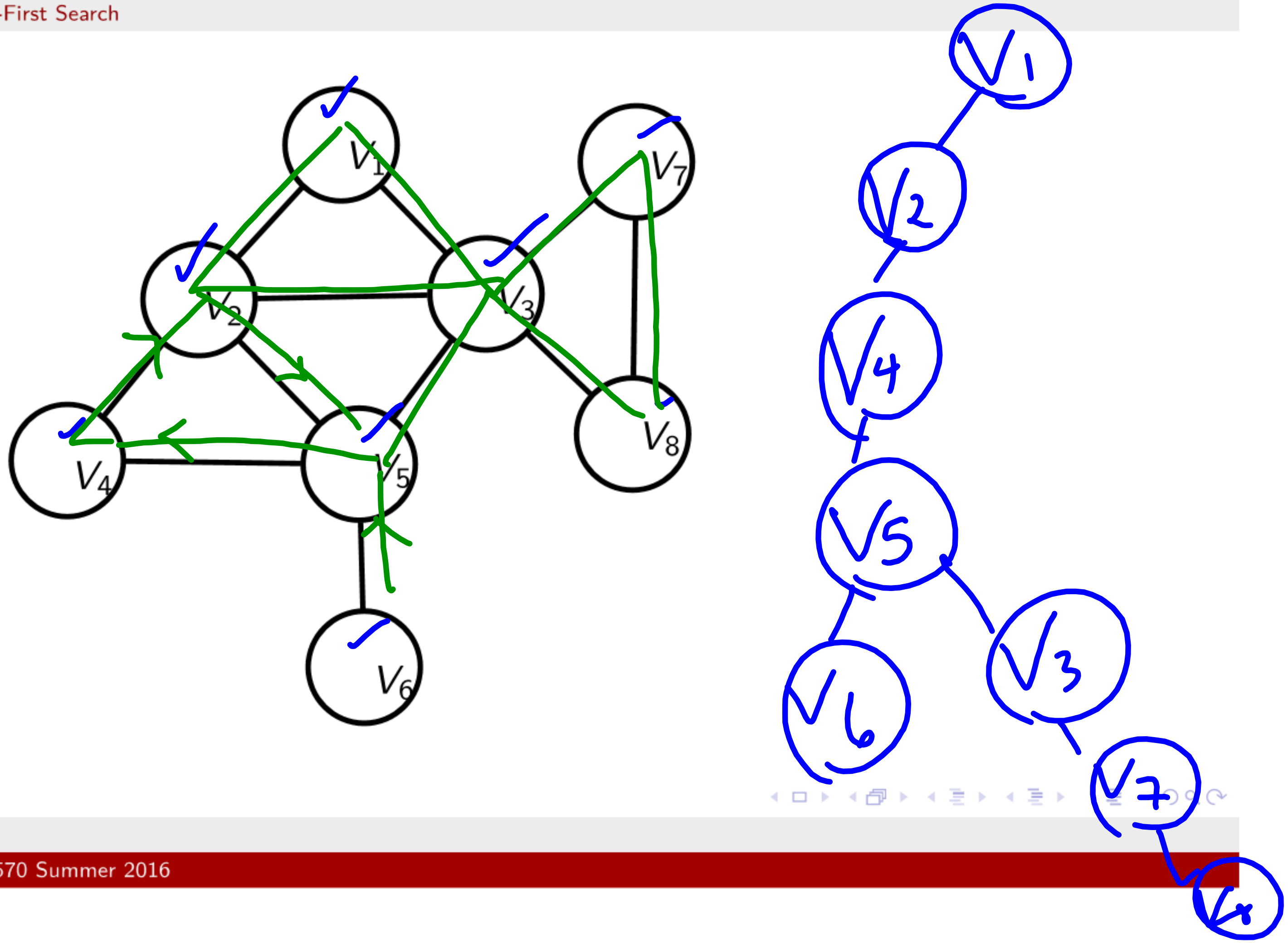
even length okay

How can we check if a graph is bipartite?

- BFS from arbitrary vertex
- partition odd | even layers
- for each edge $e = (u,v)$
  if layer(u) == layer(v)
  reject

```
DFS-recursive(u)
  Mark u as "discovered"
  for each edge (u, v) do
    if v is not marked "discovered" then
      DFS-recursive(v)
```

Depth-First Search



DFS-iterative(s)

$\forall_v$ discovered[v] = `false`

Initialize S to be a stack with s as its only element

while S is not empty **do**

$u \leftarrow$ pop(S)

if discovered[u] = `false` **then**

discovered[u] = `true`

for all edges (u, v) **do**

push(S, v)

once for each vertex

ea. edge once

How long does a depth-first search take?

O(m) total

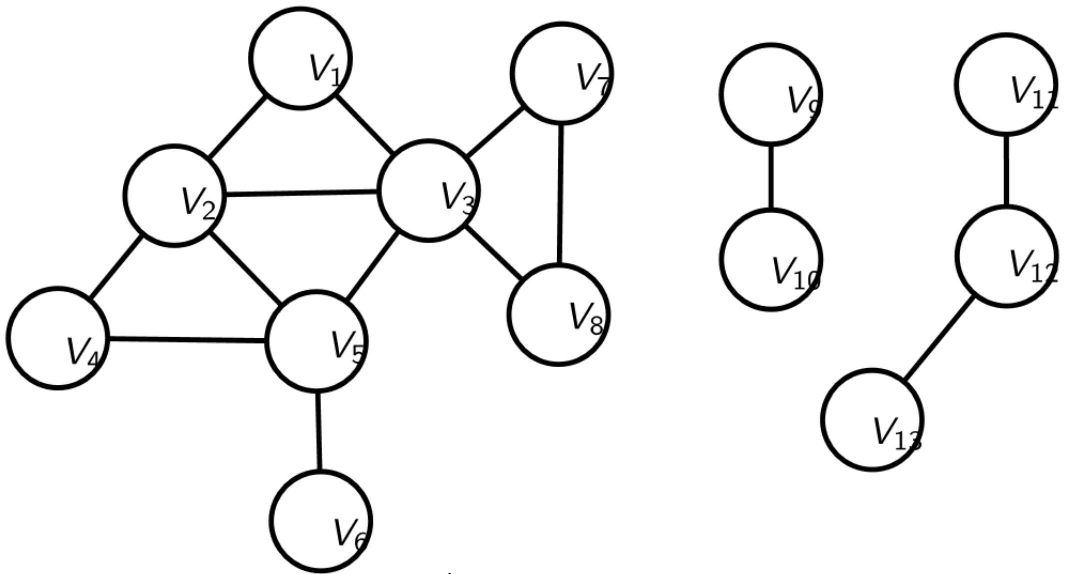
Tot: O(m+n)

Depth-First Search

Suppose I do a BFS and a DFS, separately, on the same graph starting at the same vertex.

- Do I visit the same vertices in both searches? Yes
- For the vertices that are visited in both searches, are they visited in the same order? No

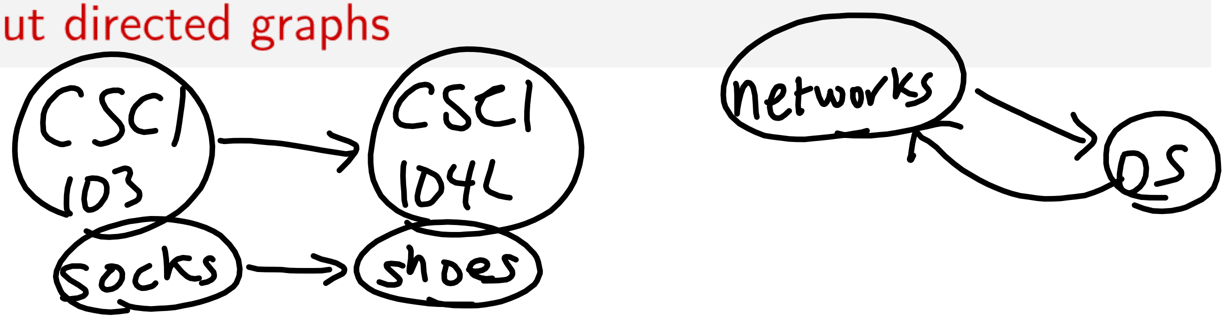
Application: Listing all connected components



$\forall v$ discovered[v] = false

for each $v \in V$
  if ! discovered [v]
    DFS (v) //or BFS(v)

About directed graphs



- What can we represent with a directed graph?
- What does it mean to be connected?

? if underlying undirected is connected

? old def: $\forall u,v\ \exists$ path $u \leadsto v$ and $v \leadsto v$ } strongly connected

? $\forall u,v\ \exists$ path $u \leadsto v$ or $v \leadsto u$ } weakly connected

Directed Acyclic Graphs

MergeSort (etc)
required total ordering

What type of problem might we represent with a directed graph such that it would have no cycles?

- prerequisites
- clothing

Directed Acyclic Graphs

directed acyclic graph

We define a *topological order* in a DAG as an ordering $v_1, v_2, \ldots, v_n$ such that if v_i appears earlier in the order than v_j, there is no path in G from v_j to v_i.

- Does every DAG have a topological order?
- Is it the case that every graph with a topological order is a DAG?

If DAG, $\exists v$ $\delta^-(v) = 0$

in-degree

```
Topological-Sort(G)
  Compute incoming[v] for each vertex
  Create B, an empty bag data structure
  B ← all v with incoming[v] = 0.
  while B ≠ ∅ do
    Remove v from B
    Output v
    for each w ∈ adj[v] do
      subtract one from incoming[w]
      if incoming[w] is now zero then
        add w to B
```

$O(m)$ or $O(n)$ (depending)

$O(n)$