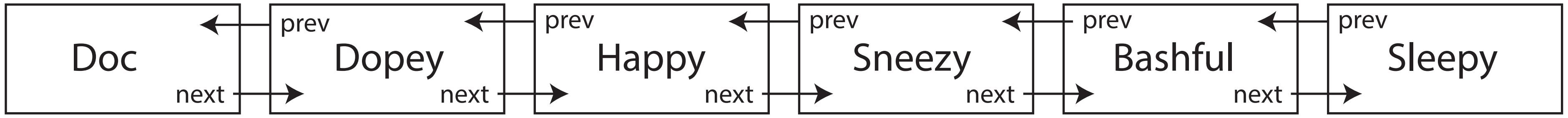
graphs and graph algorithms

main ideas from yesterday

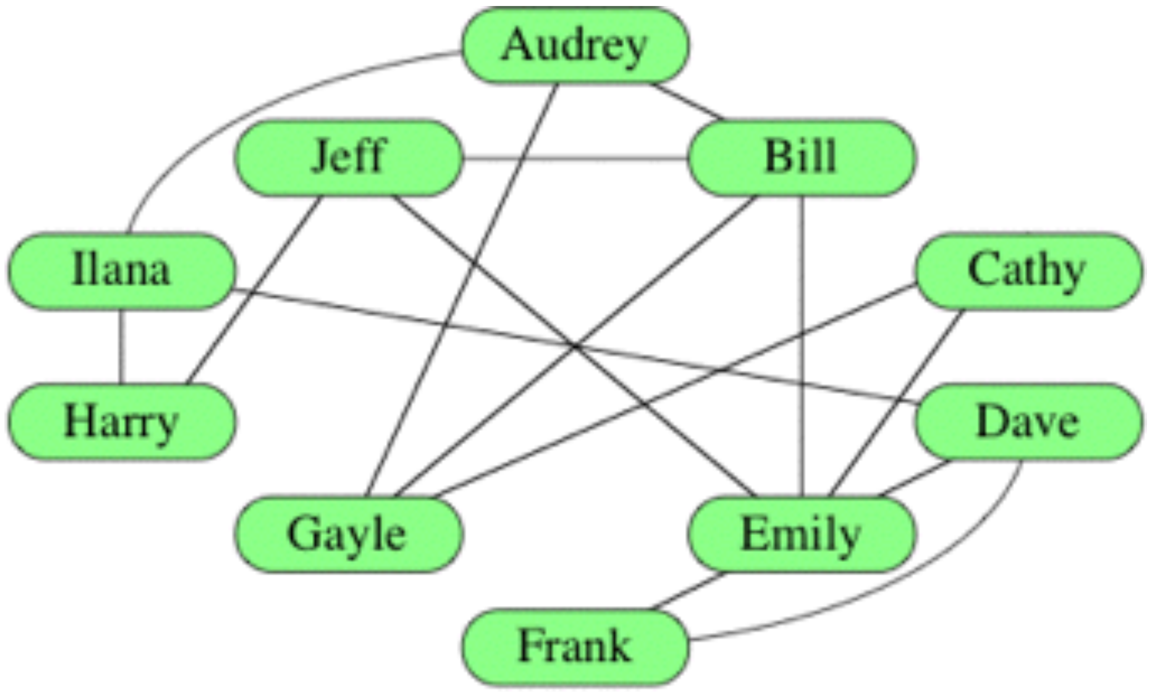
1. Objects can be used to store data items and represent connections between the items.



2. The queue ADT models a first-in-first-out process.

graphs express connections between data

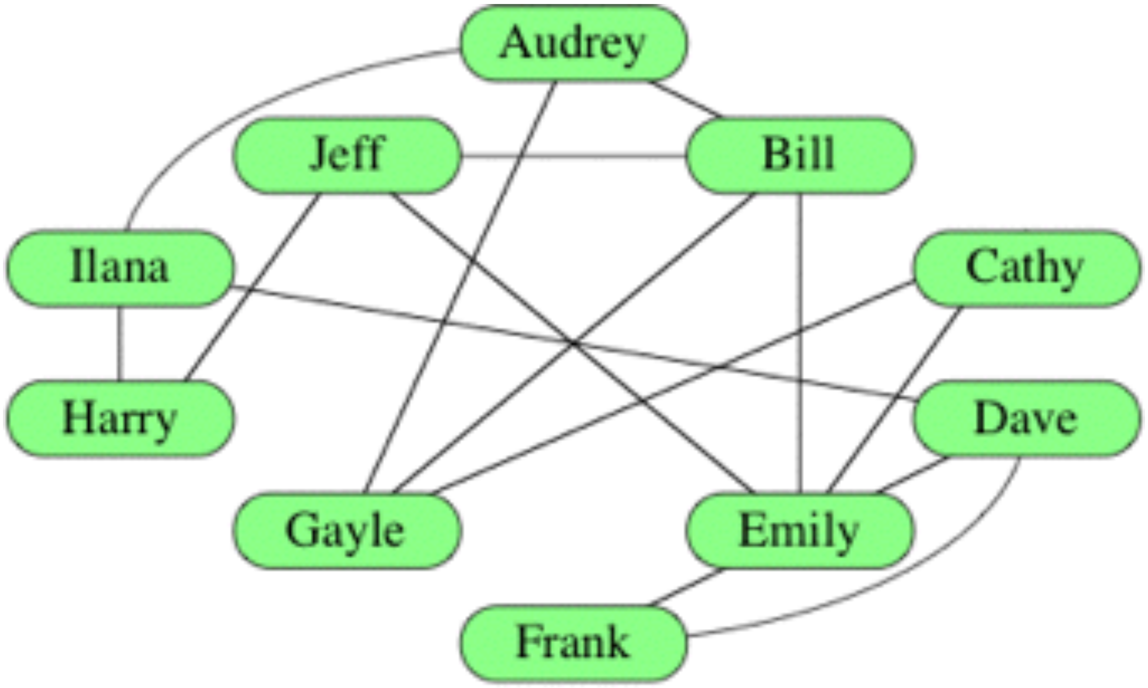
A social network:



Each **vertex** stores some data. Each **edge** connects a pair of vertices. (The words **node** and **vertex** are used interchangeably.)

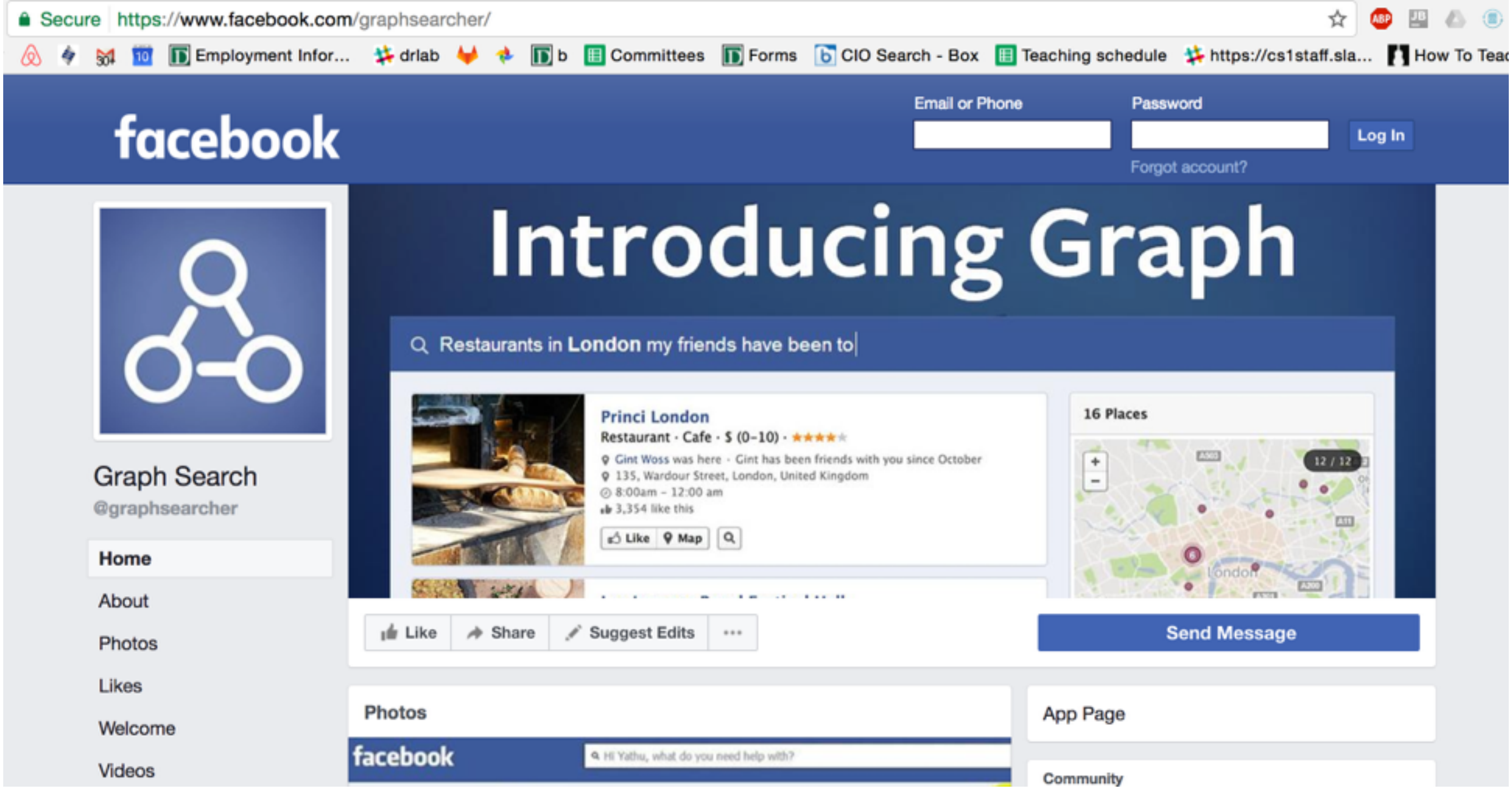
If there are *n* vertices, there may be up to *n (n - 1)* edges.

questions we could ask



- Does Cathy know Gayle? (Yes, there is an edge.)
- Is there a pathway between Harry and Emily? (Same component.)
- What is the shortest path between Harry and Emily? (H to J to E)
- Who is the most well-connected person? (Emily, vertex degree 5.)
- Largest group in which each knows everyone else (clique)?

graph models: social networks



Graph Search
@graphsearcher

Introducing Graph

Restaurants in London my friends have been to

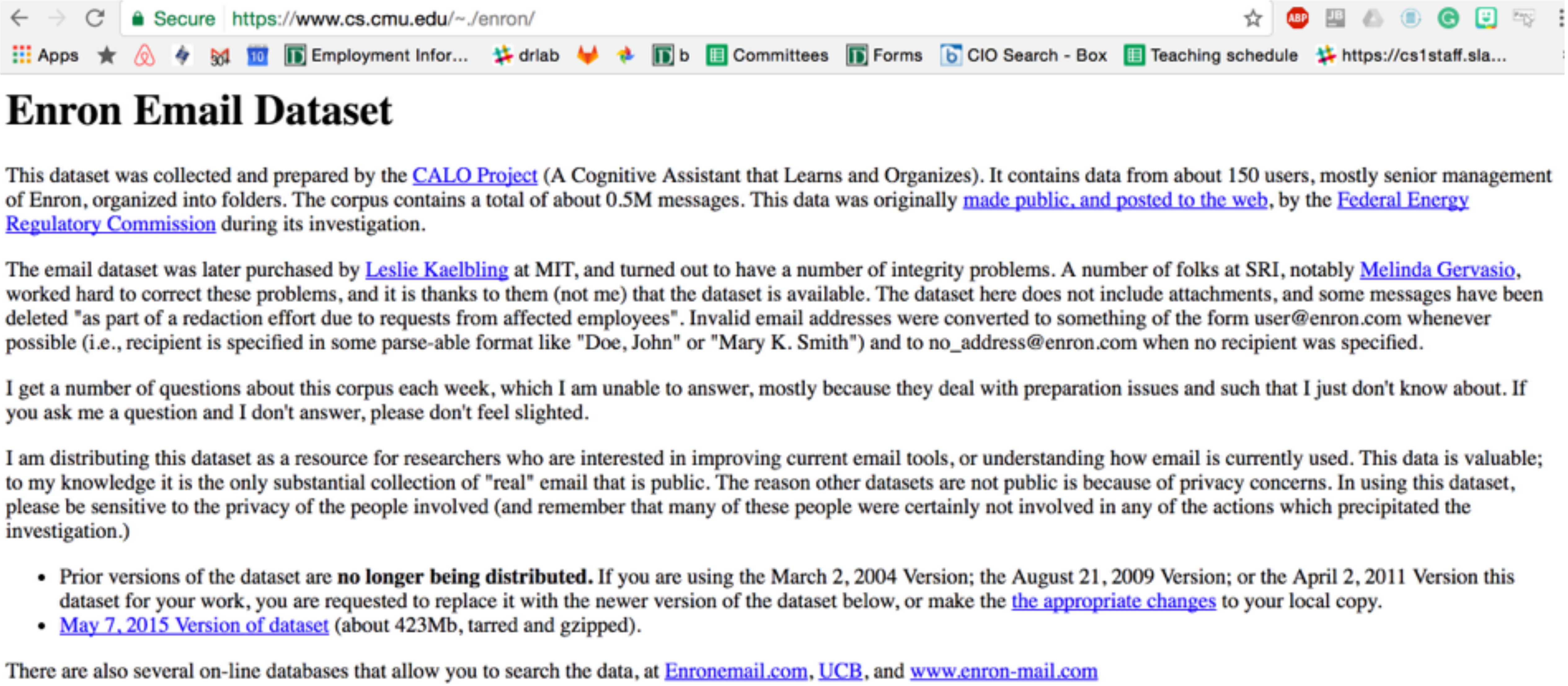
Princi London
Restaurant · Cafe · $ (0-10) · ★★★★☆
Gint Woss was here · Gint has been friends with you since October
135, Wardour Street, London, United Kingdom
8:00am – 12:00 am
3,354 like this
Like Map

16 Places

12 / 12

Hi Yathu, what do you need help with?

graph models: social networks



Enron Email Dataset

This dataset was collected and prepared by the CALO Project (A Cognitive Assistant that Learns and Organizes). It contains data from about 150 users, mostly senior management of Enron, organized into folders. The corpus contains a total of about 0.5M messages. This data was originally made public, and posted to the web, by the Federal Energy Regulatory Commission during its investigation.

The email dataset was later purchased by Leslie Kaelbling at MIT, and turned out to have a number of integrity problems. A number of folks at SRI, notably Melinda Gervasio, worked hard to correct these problems, and it is thanks to them (not me) that the dataset is available. The dataset here does not include attachments, and some messages have been deleted "as part of a redaction effort due to requests from affected employees". Invalid email addresses were converted to something of the form user@enron.com whenever possible (i.e., recipient is specified in some parse-able format like "Doe, John" or "Mary K. Smith") and to no_address@enron.com when no recipient was specified.

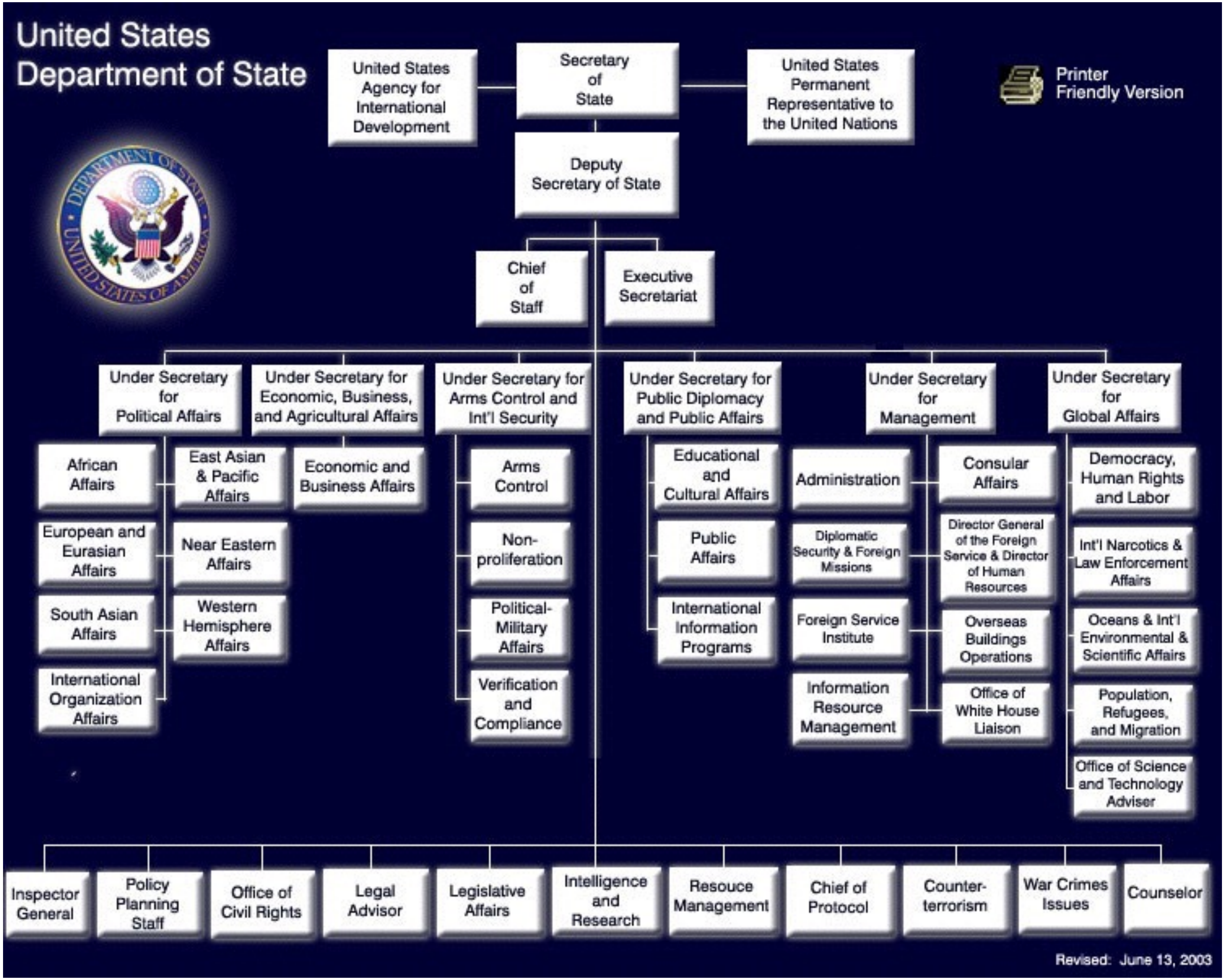
I get a number of questions about this corpus each week, which I am unable to answer, mostly because they deal with preparation issues and such that I just don't know about. If you ask me a question and I don't answer, please don't feel slighted.

I am distributing this dataset as a resource for researchers who are interested in improving current email tools, or understanding how email is currently used. This data is valuable; to my knowledge it is the only substantial collection of "real" email that is public. The reason other datasets are not public is because of privacy concerns. In using this dataset, please be sensitive to the privacy of the people involved (and remember that many of these people were certainly not involved in any of the actions which precipitated the investigation.)

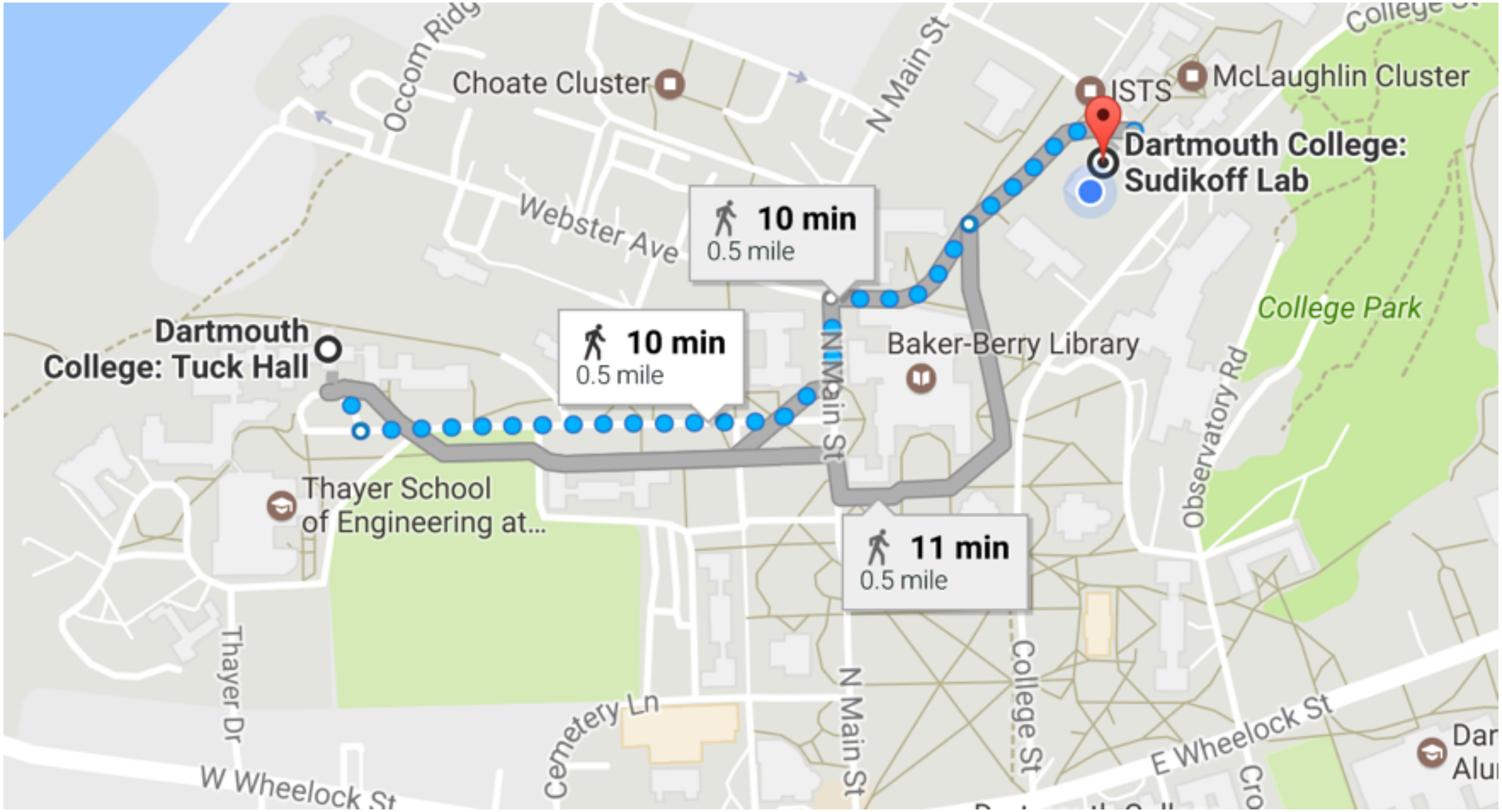
- Prior versions of the dataset are **no longer being distributed.** If you are using the March 2, 2004 Version; the August 21, 2009 Version; or the April 2, 2011 Version this dataset for your work, you are requested to replace it with the newer version of the dataset below, or make the the appropriate changes to your local copy.
- May 7, 2015 Version of dataset (about 423Mb, tarred and gzipped).

There are also several on-line databases that allow you to search the data, at Enronemail.com, UCB, and www.enron-mail.com

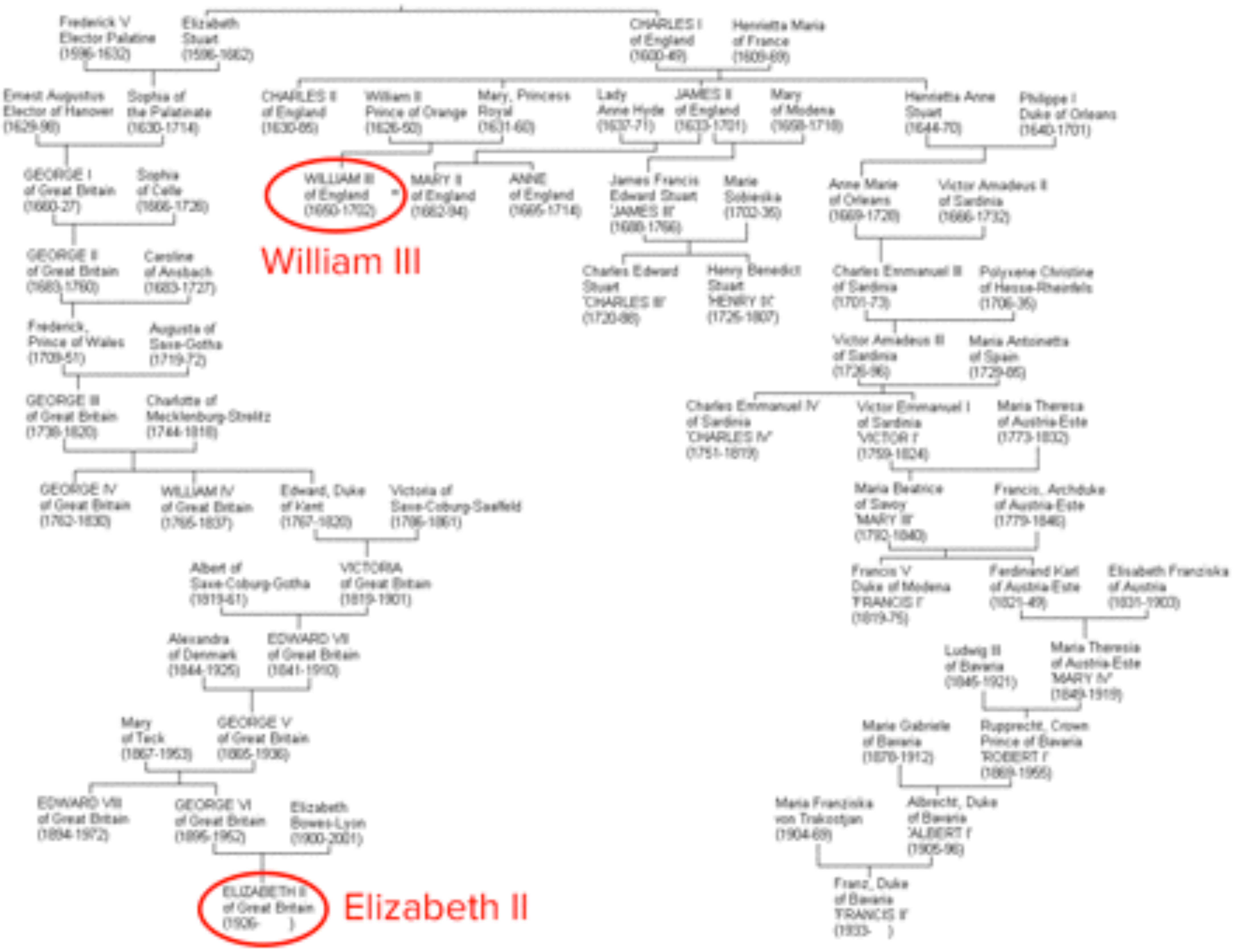
graph models: hierarchies


United States Department of State
Printer Friendly Version
United States Agency for International Development
Secretary of State
United States Permanent Representative to the United Nations
Deputy Secretary of State
Chief of Staff
Executive Secretariat
Under Secretary for Political Affairs
Under Secretary for Economic, Business, and Agricultural Affairs
Under Secretary for Arms Control and Int'l Security
Under Secretary for Public Diplomacy and Public Affairs
Under Secretary for Management
Under Secretary for Global Affairs
African Affairs
East Asian & Pacific Affairs
Economic and Business Affairs
Arms Control
Educational and Cultural Affairs
Administration
Consular Affairs
Democracy, Human Rights and Labor
European and Eurasian Affairs
Near Eastern Affairs
Non-proliferation
Public Affairs
Diplomatic Security & Foreign Missions
Director General of the Foreign Service & Director of Human Resources
Int'l Narcotics & Law Enforcement Affairs
South Asian Affairs
Western Hemisphere Affairs
Political-Military Affairs
International Information Programs
Foreign Service Institute
Overseas Buildings Operations
Oceans & Int'l Environmental & Scientific Affairs
International Organization Affairs
Verification and Compliance
Information Resource Management
Office of White House Liaison
Population, Refugees, and Migration
Office of Science and Technology Adviser
Inspector General
Policy Planning Staff
Office of Civil Rights
Legal Advisor
Legislative Affairs
Intelligence and Research
Resouce Management
Chief of Protocol
Counter-terrorism
War Crimes Issues
Counselor
Revised: June 13, 2003

graph models: physical maps



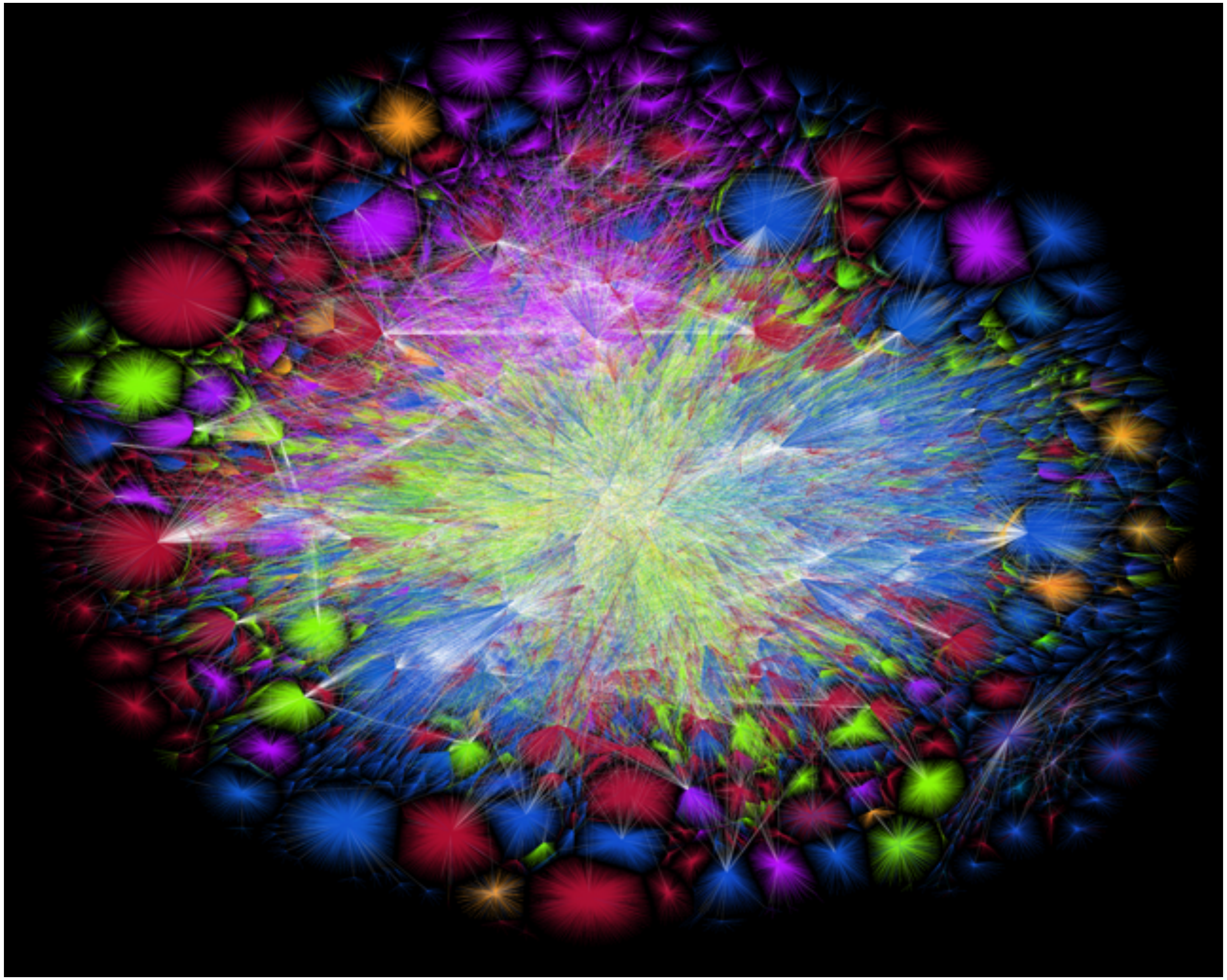
graph models: genealogy


Frederick V Elector Palatine (1596-1632)
Elizabeth Stuart (1596-1662)
CHARLES I of England (1600-49)
Henrietta Maria of France (1609-69)
Ernest Augustus Elector of Hanover (1629-98)
Sophia of the Palatinate (1630-1714)
CHARLES II of England (1630-85)
William II Prince of Orange (1626-50)
Mary, Princess Royal (1631-60)
Lady Anne Hyde (1637-71)
JAMES II of England (1633-1701)
Mary of Modena (1658-1718)
Henrietta Anne Stuart (1644-70)
Philippe I Duke of Orleans (1640-1701)
GEORGE I of Great Britain (1660-27)
Sophia of Celle (1666-1726)
WILLIAM III of England (1650-1702)
MARY II of England (1662-94)
ANNE of England (1665-1714)
James Francis Edward Stuart 'JAMES III' (1688-1766)
Marie Sobieska (1702-35)
Anne Marie of Orleans (1669-1728)
Victor Amadeus II of Sardinia (1666-1732)
GEORGE II of Great Britain (1683-1760)
Caroline of Ansbach (1683-1727)
Charles Edward Stuart 'CHARLES III' (1720-88)
Henry Benedict Stuart 'HENRY IX' (1725-1807)
Charles Emmanuel III of Sardinia (1701-73)
Polyxene Christine of Hesse-Rheinfels (1706-35)
Frederick, Prince of Wales (1709-51)
Augusta of Saxe-Gotha (1719-72)
Victor Amadeus III of Sardinia (1726-96)
Maria Antonietta of Spain (1729-85)
GEORGE III of Great Britain (1738-1820)
Charlotte of Mecklenburg-Strelitz (1744-1818)
Charles Emmanuel IV of Sardinia 'CHARLES IV' (1751-1819)
Victor Emmanuel I of Sardinia 'VICTOR I' (1759-1824)
Maria Theresa of Austria-Este (1773-1832)
GEORGE IV of Great Britain (1762-1830)
WILLIAM IV of Great Britain (1765-1837)
Edward, Duke of Kent (1767-1820)
Victoria of Saxe-Coburg-Saalfeld (1786-1861)
Maria Beatrice of Savoy 'MARY III' (1792-1840)
Francis, Archduke of Austria-Este (1779-1846)
Albert of Saxe-Coburg-Gotha (1819-61)
VICTORIA of Great Britain (1819-1901)
Francis V Duke of Modena 'FRANCIS I' (1819-75)
Ferdinand Karl of Austria-Este (1821-49)
Elisabeth Franziska of Austria (1831-1903)
Alexandra of Denmark (1844-1925)
EDWARD VII of Great Britain (1841-1910)
Ludwig III of Bavaria (1845-1921)
Maria Theresia of Austria-Este 'MARY IV' (1849-1919)
Mary of Teck (1867-1953)
GEORGE V of Great Britain (1865-1936)
Marie Gabriele of Bavaria (1878-1912)
Rupprecht, Crown Prince of Bavaria 'ROBERT I' (1869-1955)
EDWARD VIII of Great Britain (1894-1972)
GEORGE VI of Great Britain (1895-1952)
Elizabeth Bowes-Lyon (1900-2001)
Maria Franziska von Trakostjan (1904-69)
Albrecht, Duke of Bavaria 'ALBERT I' (1905-96)
ELIZABETH II of Great Britain (1926-)
Franz, Duke of Bavaria 'FRANCIS II' (1933-)

William III

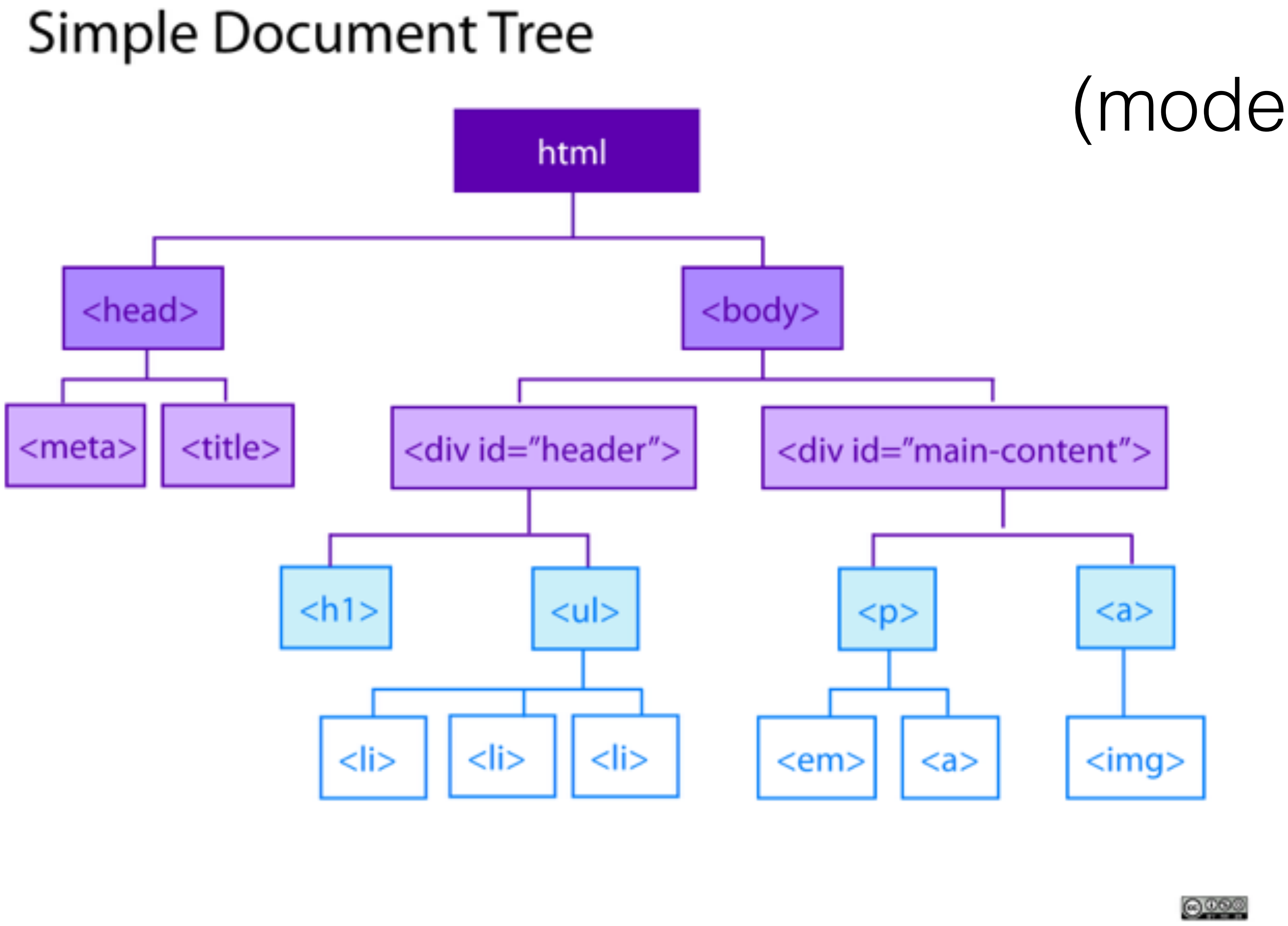
Elizabeth II

graph models: the web



(Image from the Opte project.)

graph models: document structure

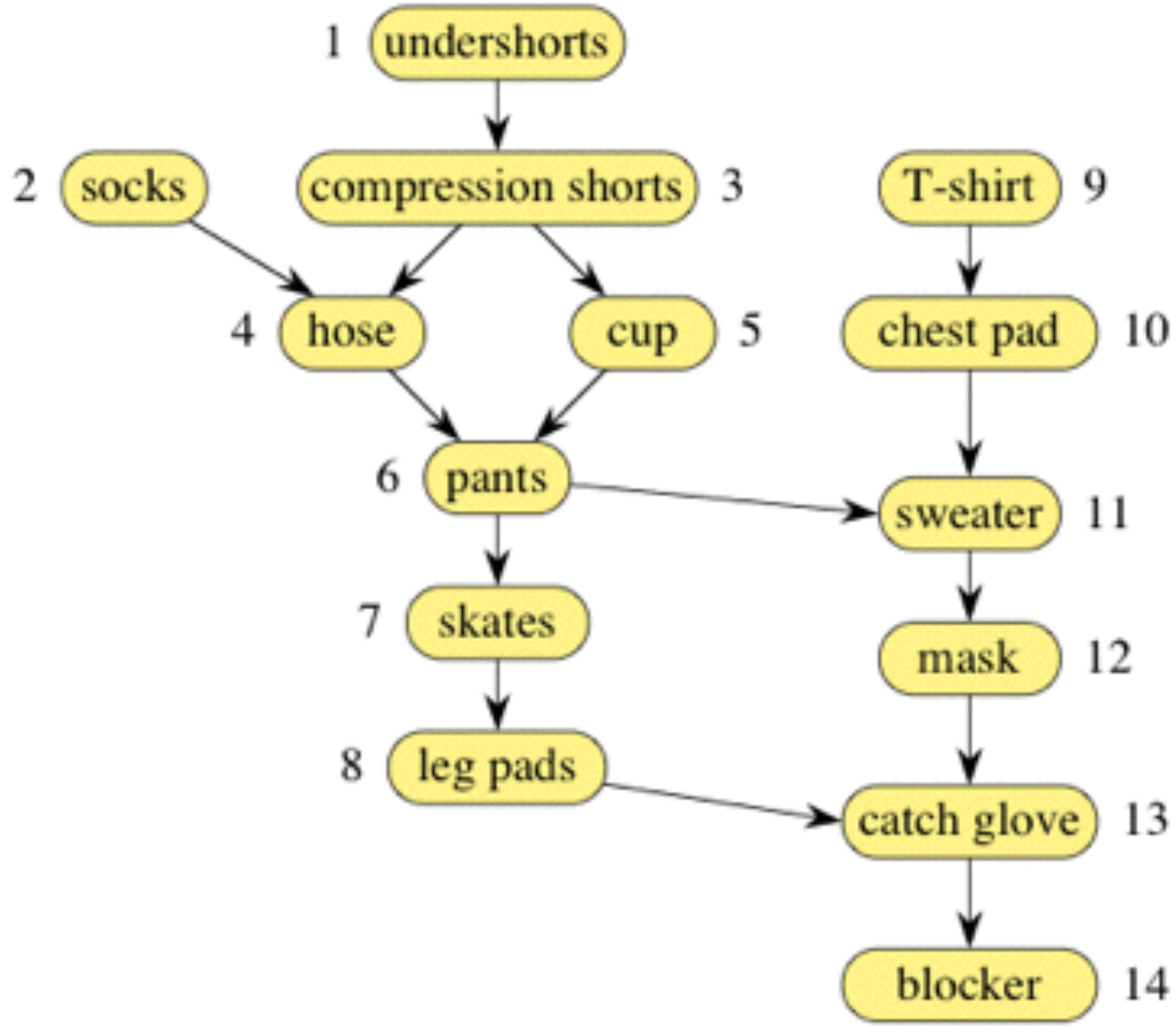


(models containership)

(Image from dabrook.org.)

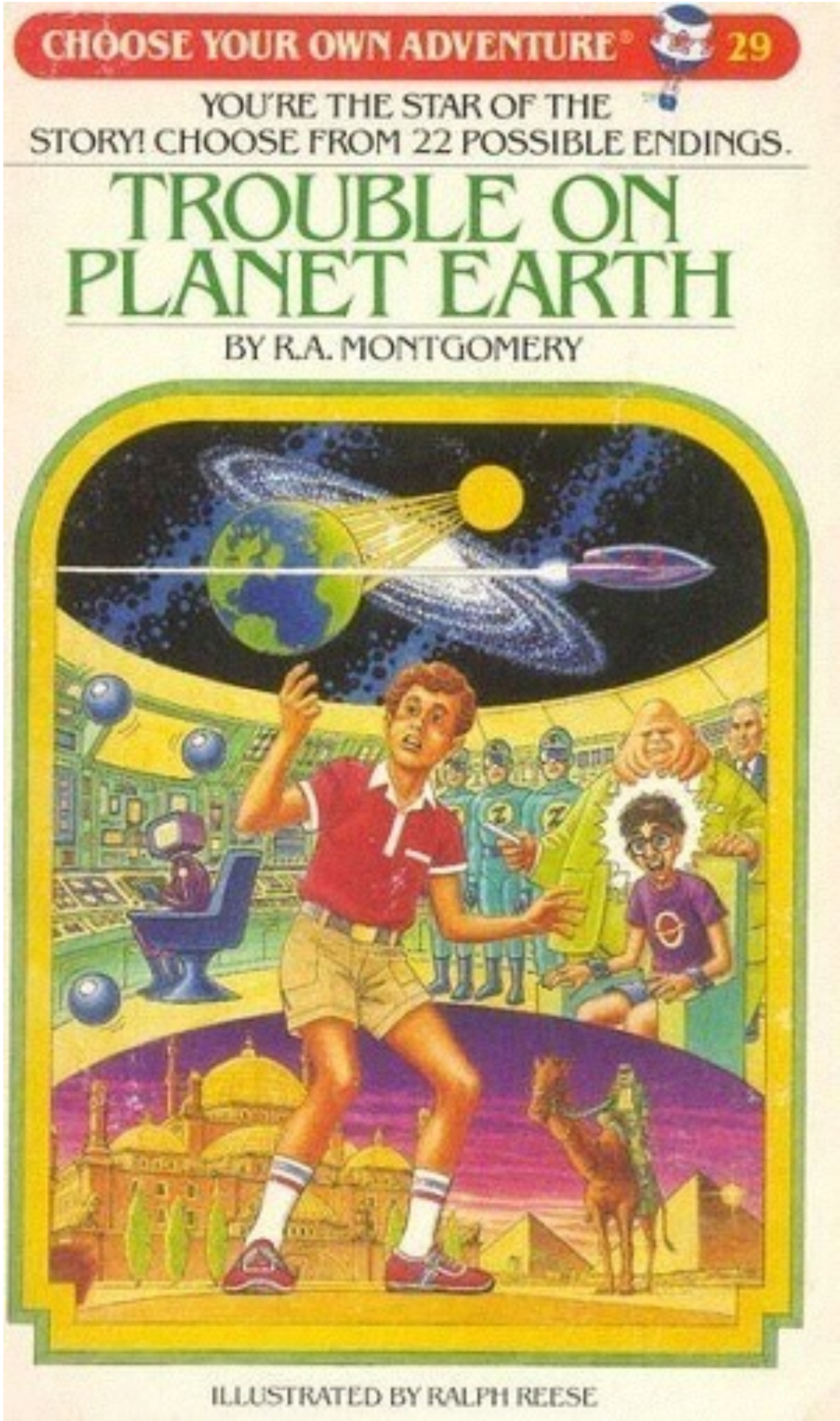
graph models: ordering constraints

Restrictions on the order in which a hockey goalie can get dressed:

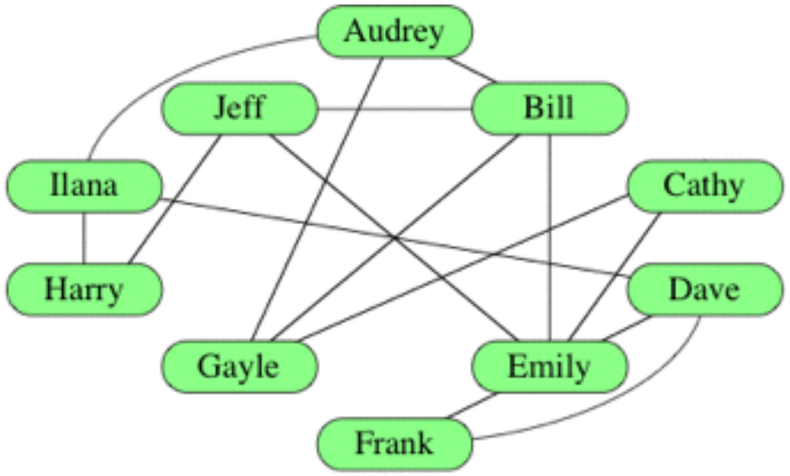


(Note: **directed** graph. Example by Tom Cormen.)

graph models: decisions and AI


CHOOSE YOUR OWN ADVENTURE® 29
YOU'RE THE STAR OF THE STORY! CHOOSE FROM 22 POSSIBLE ENDINGS.
TROUBLE ON PLANET EARTH
BY R.A. MONTGOMERY
ILLUSTRATED BY RALPH REESE

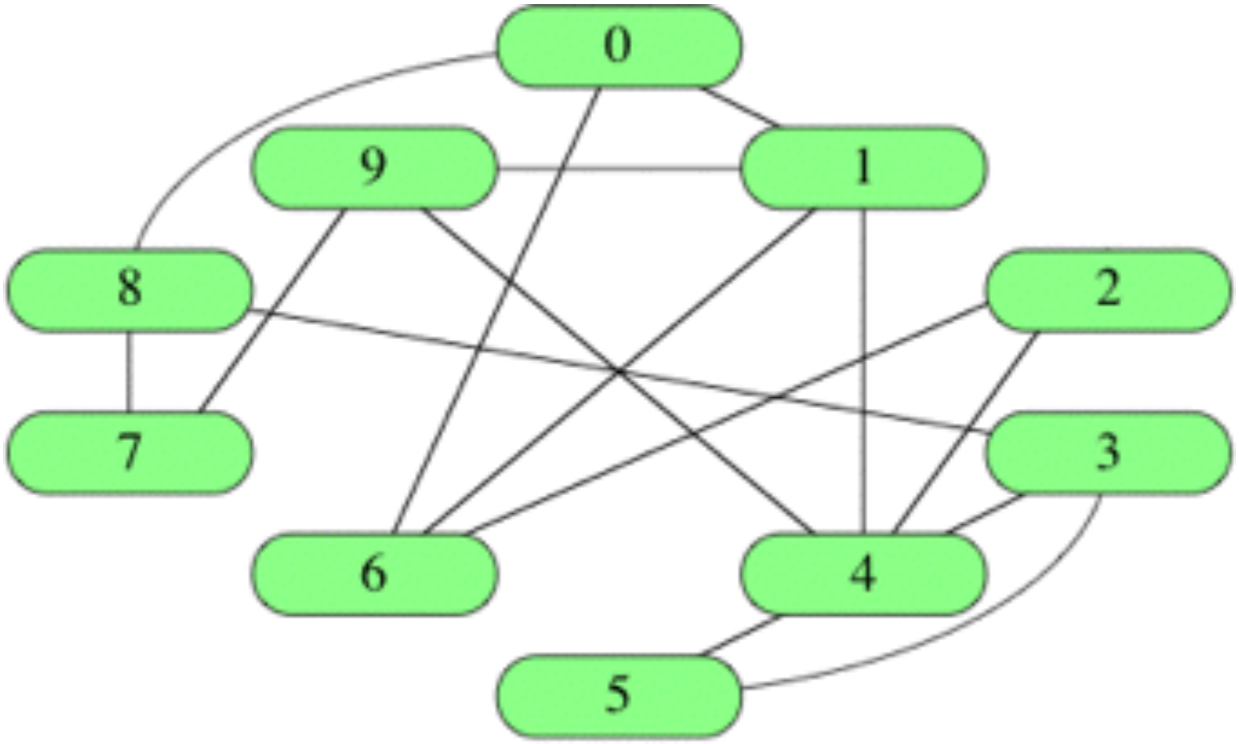
questions we could ask



- Does Cathy know Gayle? (Yes, there is an edge.)
- Is there a pathway between Harry and Emily? (Same component.)
- What is the shortest path between Harry and Emily? (H to J to E)
- Who is the most well-connected person? (Emily, vertex degree 5.)
- Largest group in which each knows everyone else (clique)?

Food for thought: what are analogous questions for each of the previous applications?

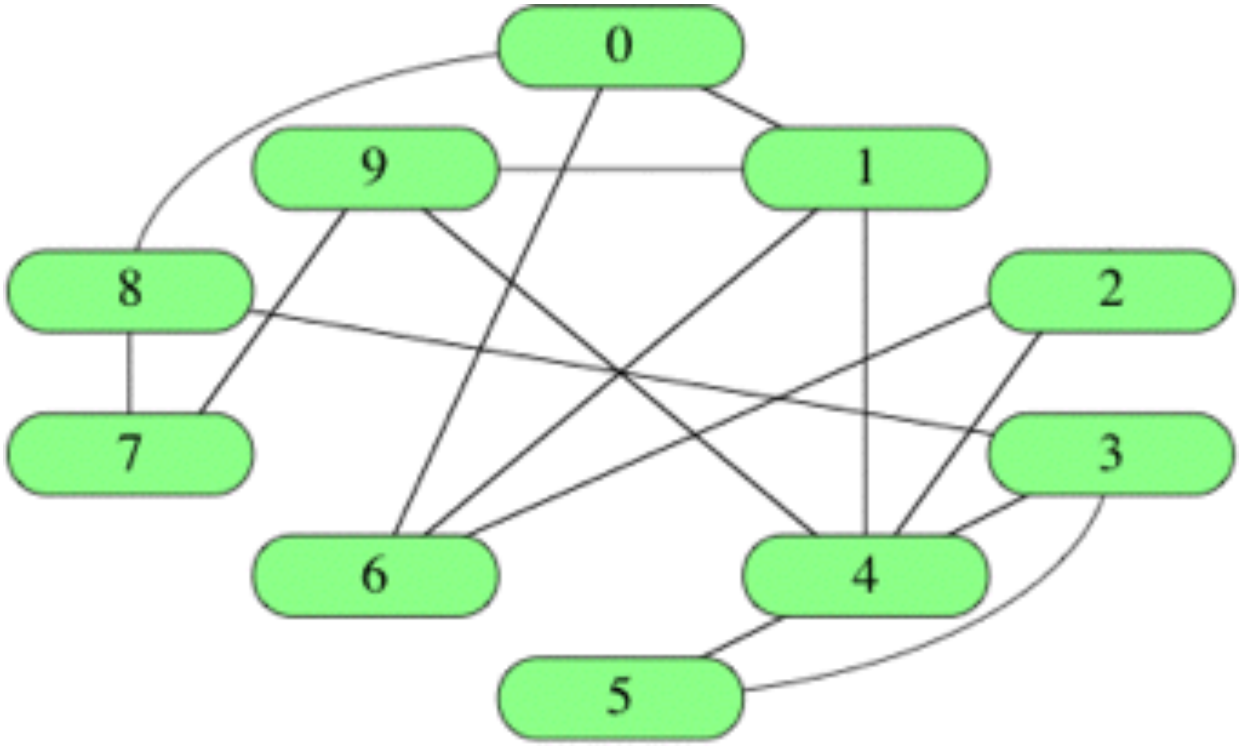
representing a graph: edge list

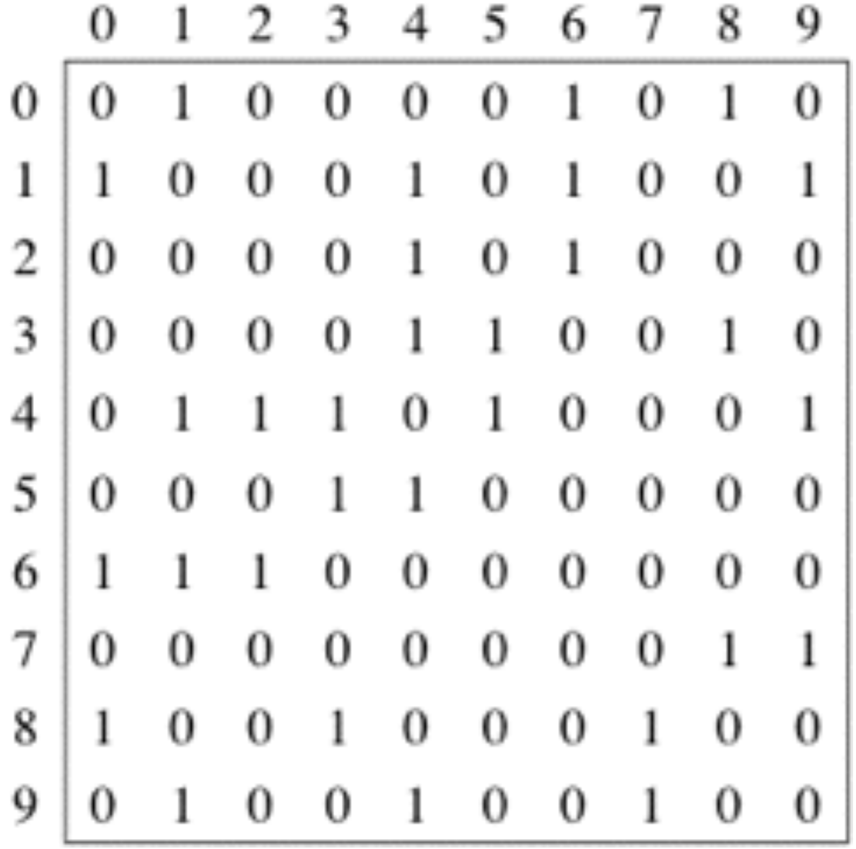


[[0,1], [0,6], [0,8], [1,4], [1,6], [1,9], [2,4], [2,6], [3,4], [3,5], [3,8], [4,5], [4,9], [7,8], [7,9]]

- How long does it take to answer whether two vertices are connected?
- How much memory is required?

representing a graph: adjacency matrix

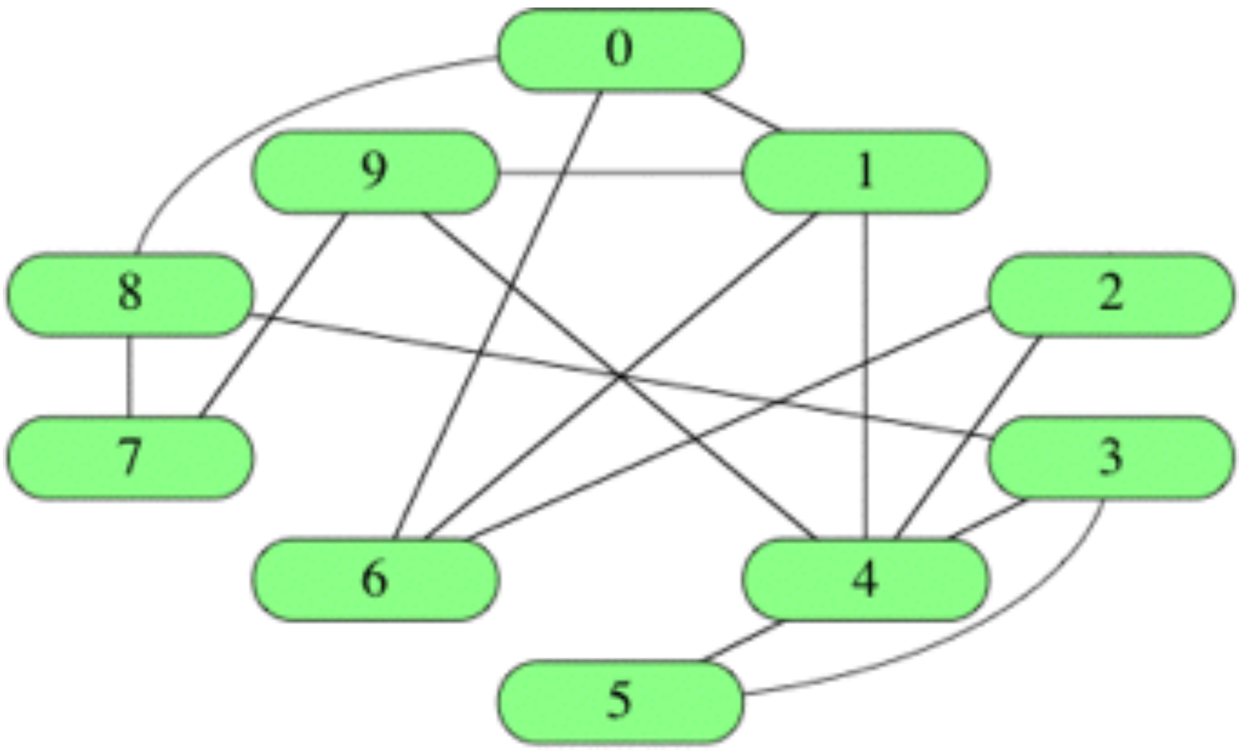


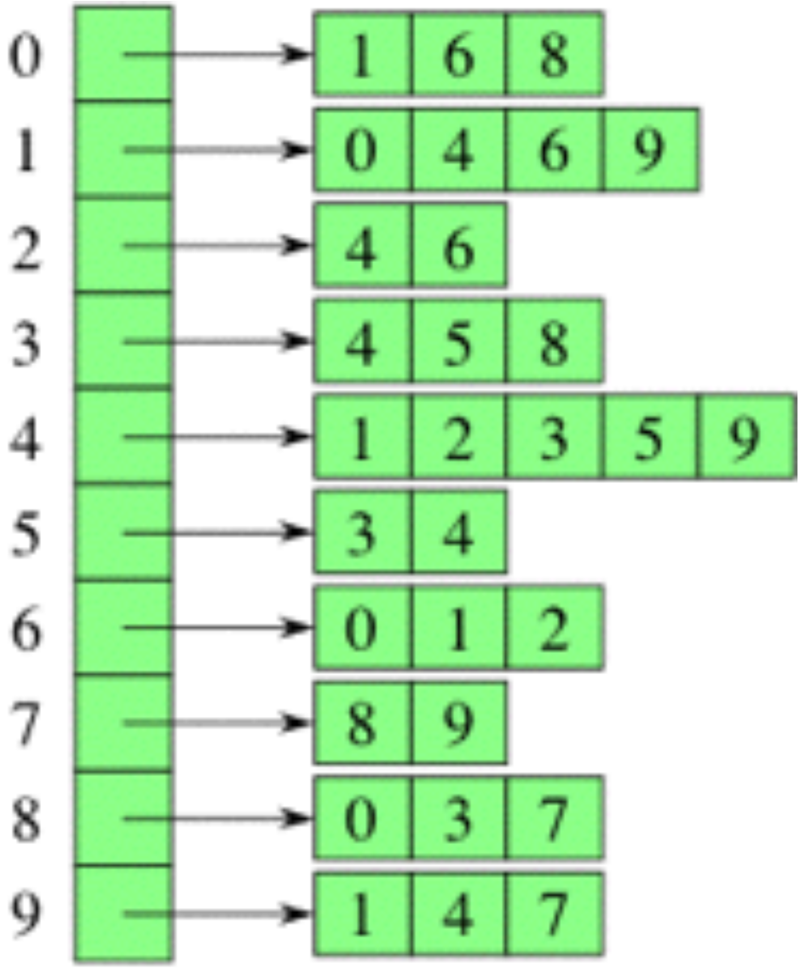


	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	1	0	1	0
1	1	0	0	0	1	0	1	0	0	1
2	0	0	0	0	1	0	1	0	0	0
3	0	0	0	0	1	1	0	0	1	0
4	0	1	1	1	0	1	0	0	0	1
5	0	0	0	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1
8	1	0	0	1	0	0	0	1	0	0
9	0	1	0	0	1	0	0	1	0	0

- How long does it take to answer whether two vertices are connected?
- How much memory is required?

representing a graph: adjacency lists



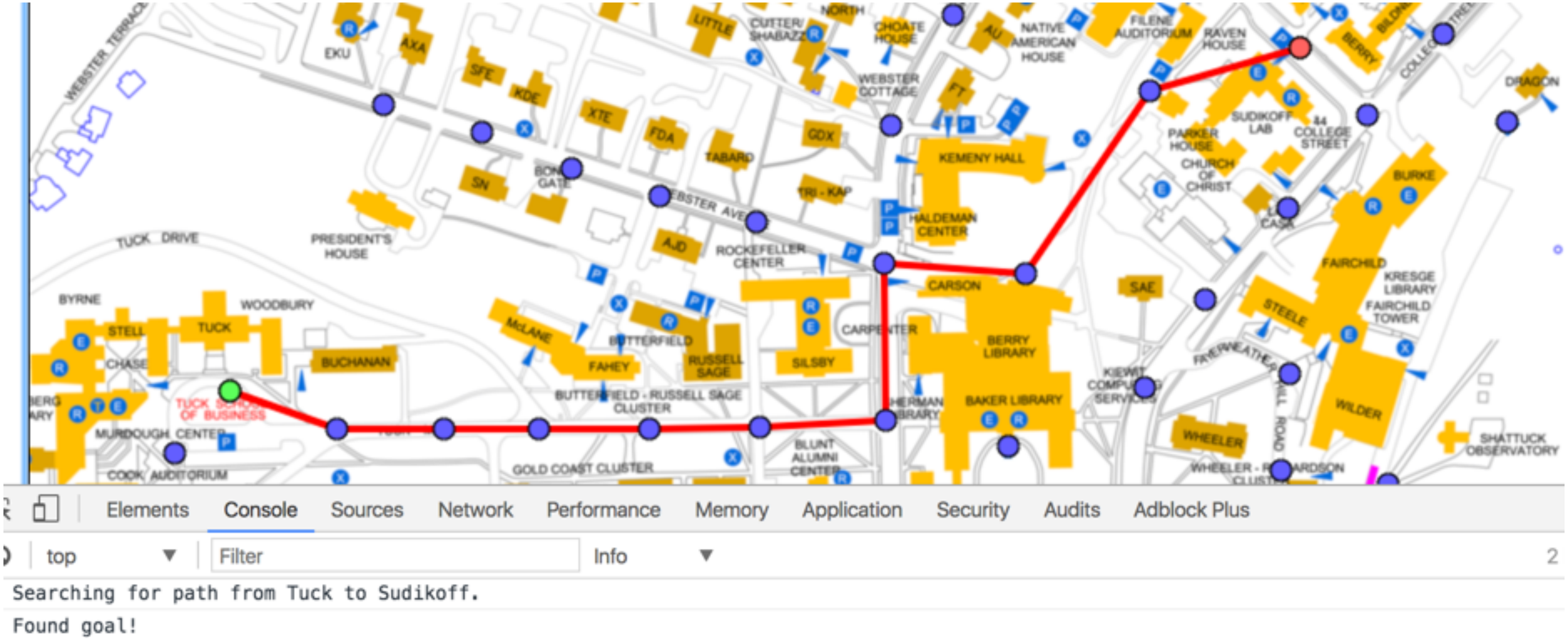


0 → 1 6 8
1 → 0 4 6 9
2 → 4 6
3 → 4 5 8
4 → 1 2 3 5 9
5 → 3 4
6 → 0 1 2
7 → 8 9
8 → 0 3 7
9 → 1 4 7

- How long does it take to answer whether two vertices are connected?
- How much memory is required?

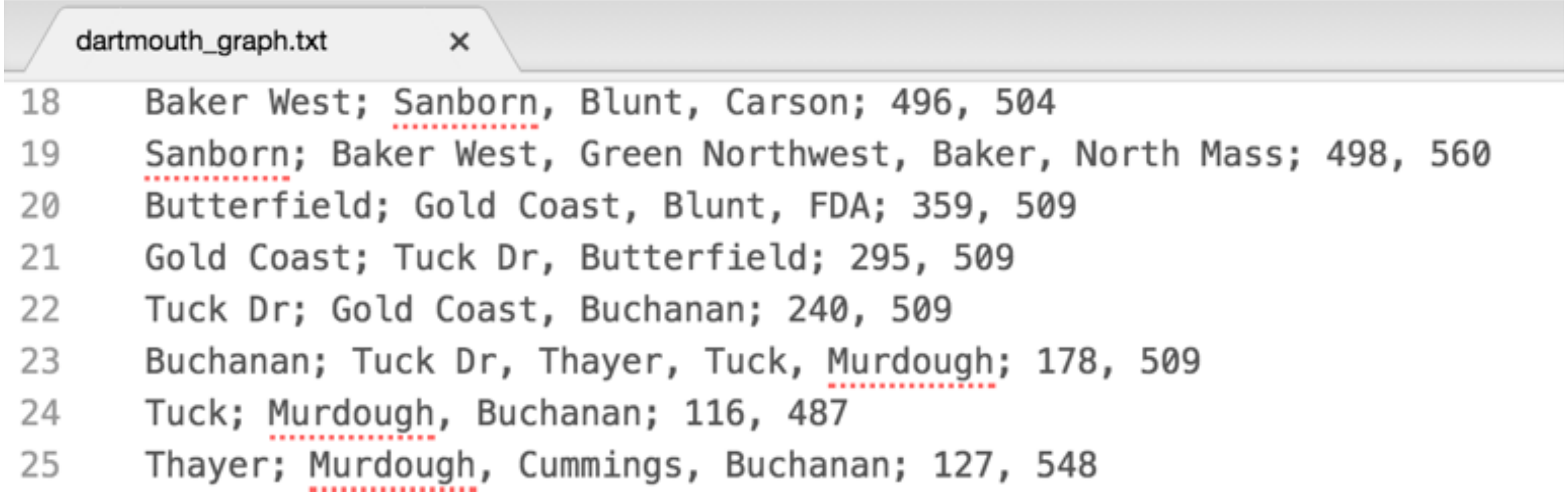
(Our preferred method)

representing a graph: example



```
Searching for path from Tuck to Sudikoff.
Found goal!
```

what's in a node?



```
Baker West; Sanborn, Blunt, Carson; 496, 504
Sanborn; Baker West, Green Northwest, Baker, North Mass; 498, 560
Butterfield; Gold Coast, Blunt, FDA; 359, 509
Gold Coast; Tuck Dr, Butterfield; 295, 509
Tuck Dr; Gold Coast, Buchanan; 240, 509
Buchanan; Tuck Dr, Thayer, Tuck, Murdough; 178, 509
Tuck; Murdough, Buchanan; 116, 487
Thayer; Murdough, Cummings, Buchanan; 127, 548
```

- some data: name, pixel coordinates:

```
tuckNode.name = “Tuck”;
tuckNode.x = 116;
tuckNode.y = 487;
```

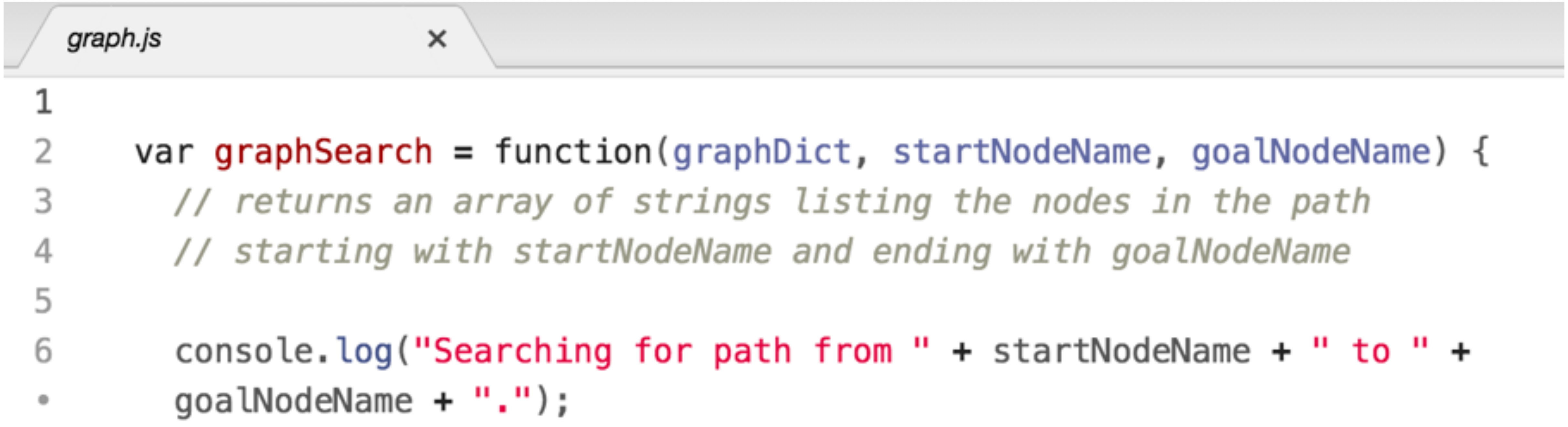
- an adjacency list:

```
tuckNode.adjacent =
  [“Murdough”, “Buchanan”];
```

given the name of a node, how do you get the node?

graphDict dictionary indexes nodes using names (strings):

```
var myNode = graphDict[“Tuck”];
console.log(myNode.name);
console.log(myNode.x);
console.log(myNode.y);
```



```

var graphSearch = function(graphDict, startNodeName, goalNodeName) {
  // returns an array of strings listing the nodes in the path
  // starting with startNodeName and ending with goalNodeName

  console.log("Searching for path from " + startNodeName + " to " +
  goalNodeName + ".");
```

start by experimenting with fetching nodes from graphDict.

given node name, how do you get names of adjacent nodes?

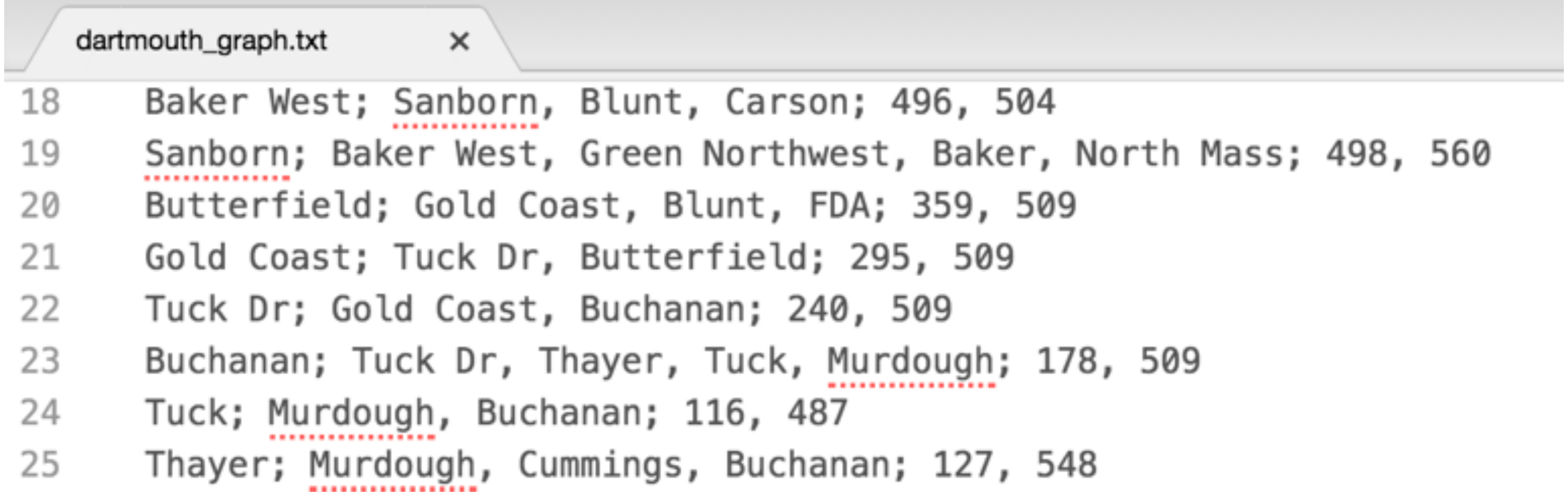
`graphDict` dictionary indexes nodes using names (strings):

```
var currentNodeName = “Tuck”;

// Grab the node from the dictionary
var currentNode = graphDict[currentNodeName];

// The node contains the adjacency list:
console.log(currentNode.adjacent);
```

In this example, `currentNode.adjacent` contains an array of strings.

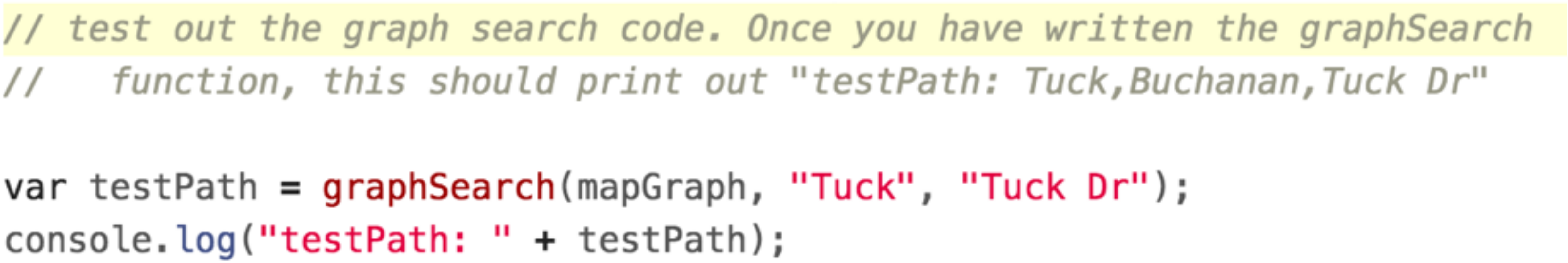
breadth-first search on a graph

given two strings representing the start and goal locations, what is the shortest connecting sequence of node names?



```
Baker West; Sanborn, Blunt, Carson; 496, 504
Sanborn; Baker West, Green Northwest, Baker, North Mass; 498, 560
Butterfield; Gold Coast, Blunt, FDA; 359, 509
Gold Coast; Tuck Dr, Butterfield; 295, 509
Tuck Dr; Gold Coast, Buchanan; 240, 509
Buchanan; Tuck Dr, Thayer, Tuck, Murdough; 178, 509
Tuck; Murdough, Buchanan; 116, 487
Thayer; Murdough, Cummings, Buchanan; 127, 548
```

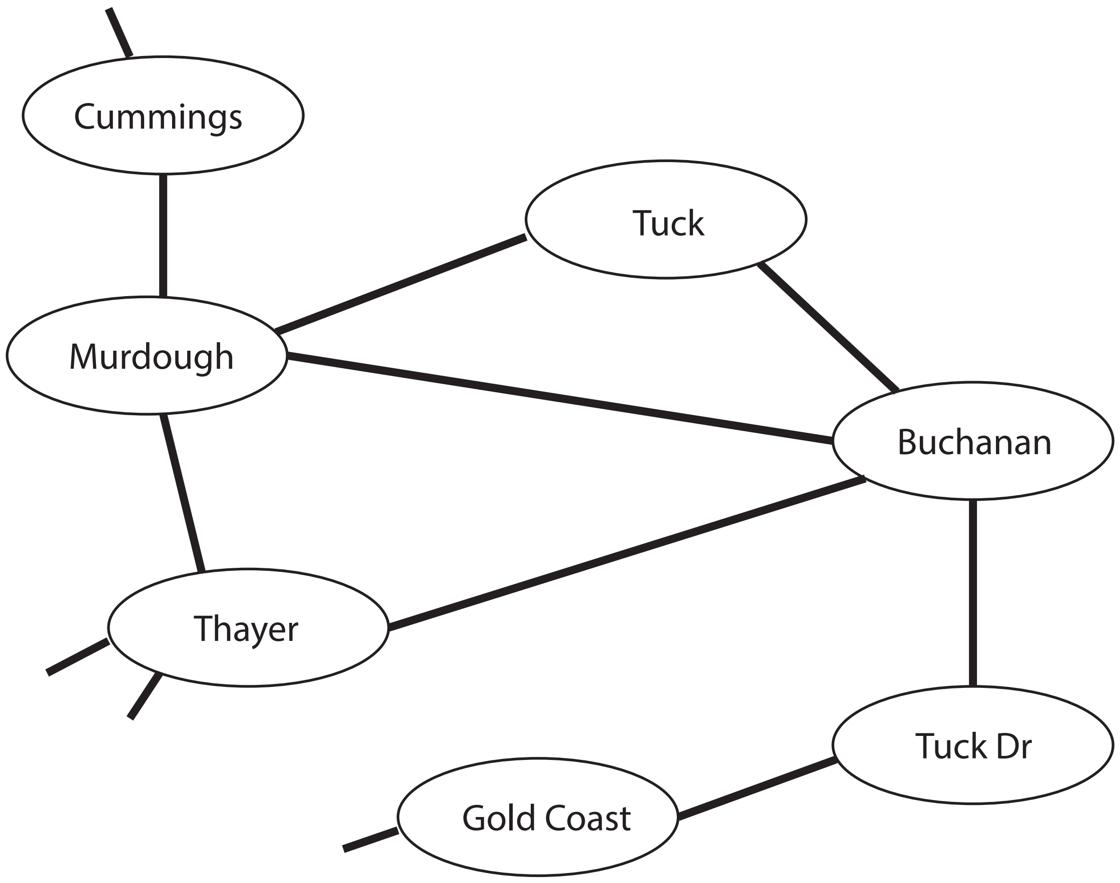
Example:



```
// test out the graph search code. Once you have written the graphSearch
//   function, this should print out "testPath: Tuck,Buchanan,Tuck Dr"

var testPath = graphSearch(mapGraph, "Tuck", "Tuck Dr");
console.log("testPath: " + testPath);
```

a ‘harder’ problem that is easier to solve

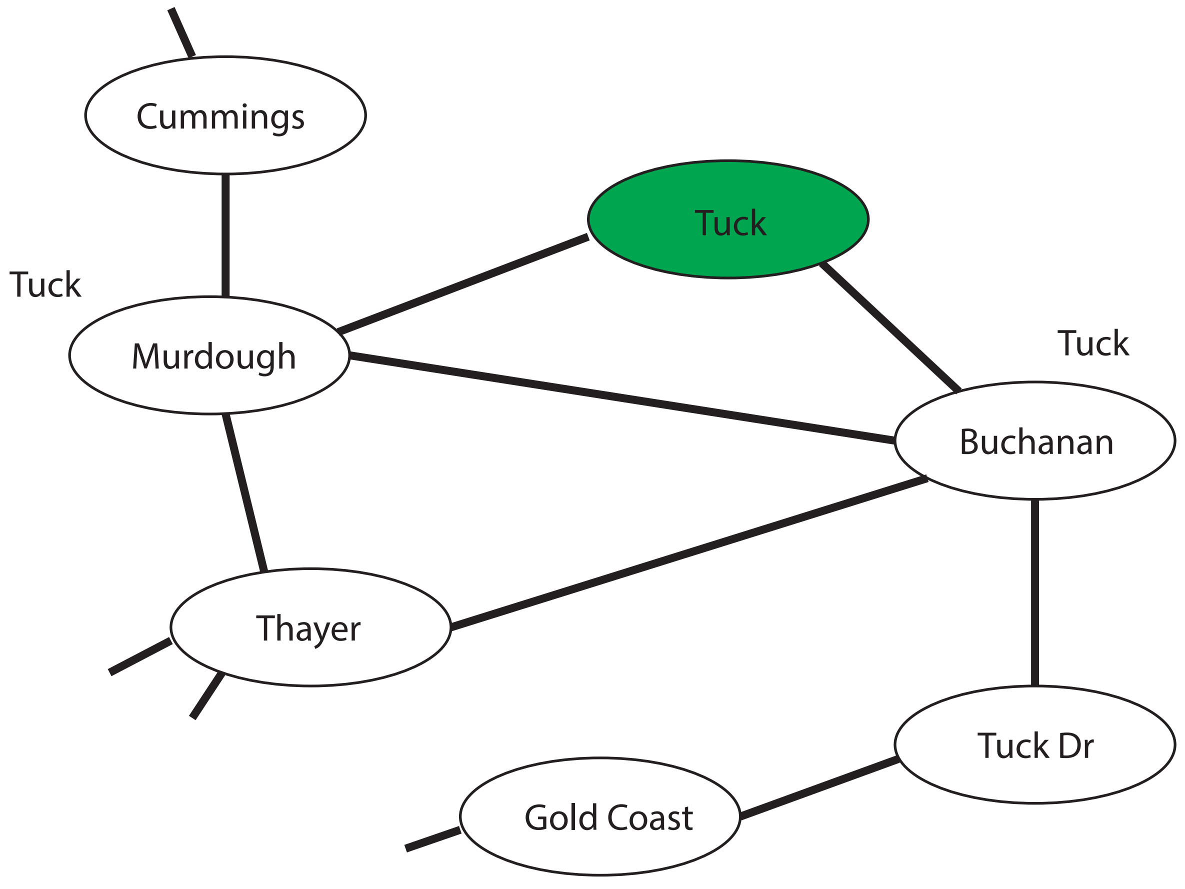
given a string for the start, what is the shortest connecting sequence to every other node?



(Note — geometry does not matter.)

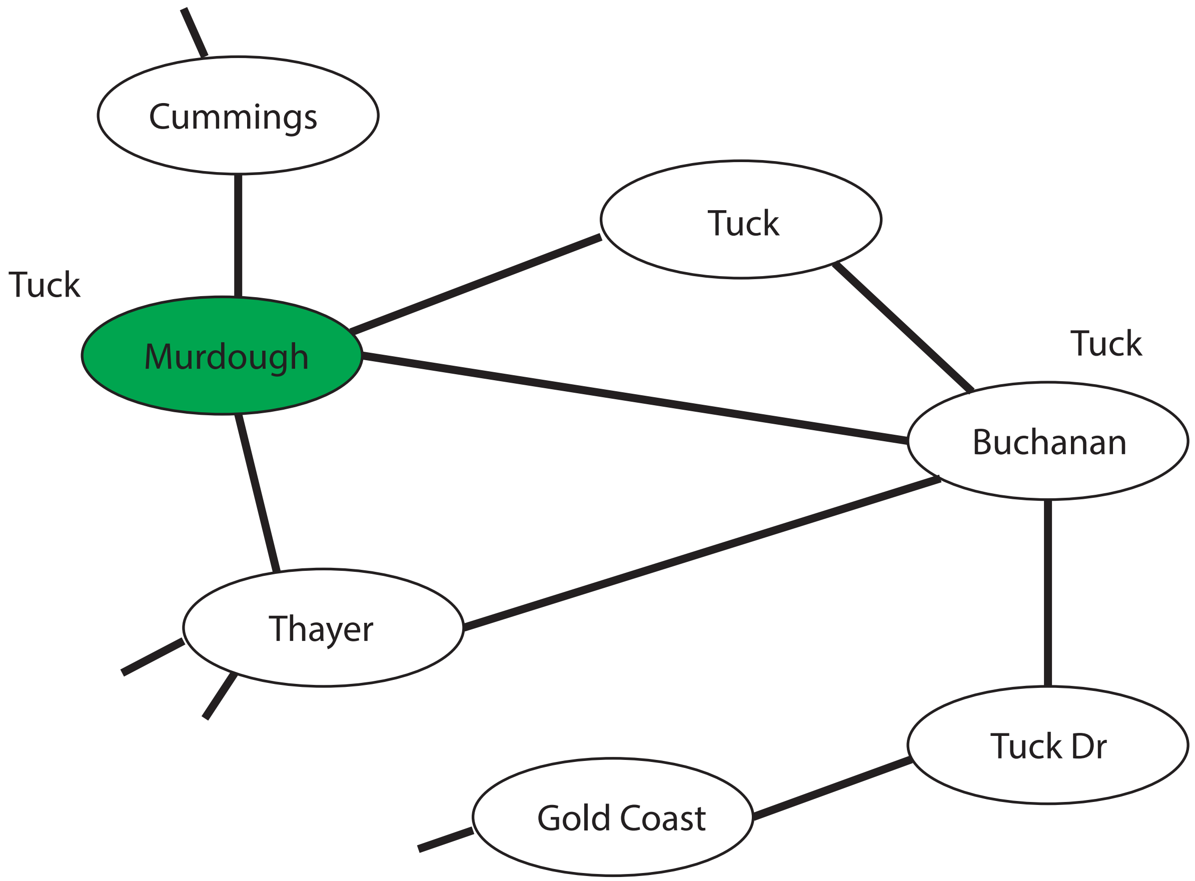
breadth-first exploration from Tuck

Start at Tuck. Send minions to claim adjacent nodes.



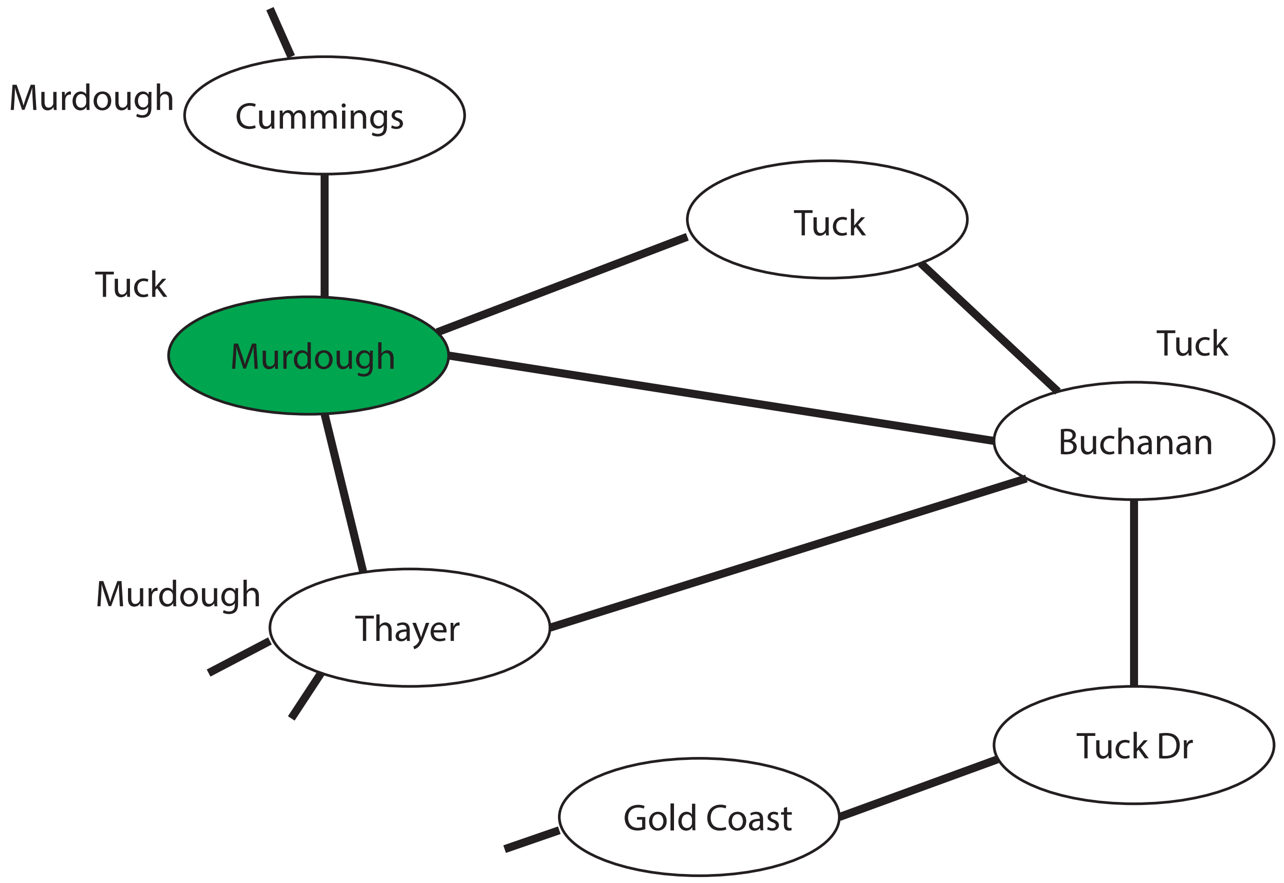
breadth-first exploration from Tuck

Now that Murdough has been claimed, it starts producing minions of its own:



breadth-first exploration from Tuck

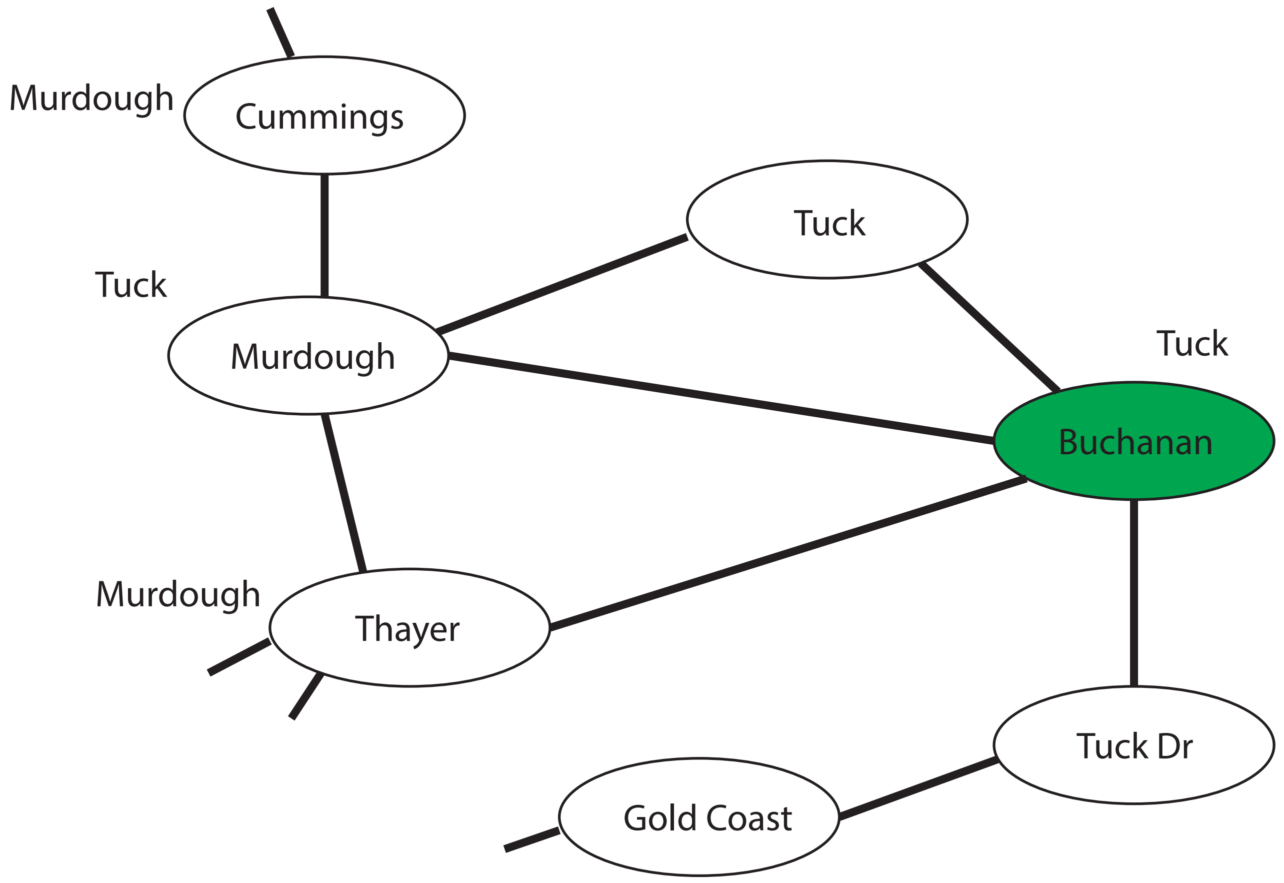
Now that Murdough has been claimed, it starts producing minions of its own:



Notice: Murdough-ians do not reclaim Tuck.

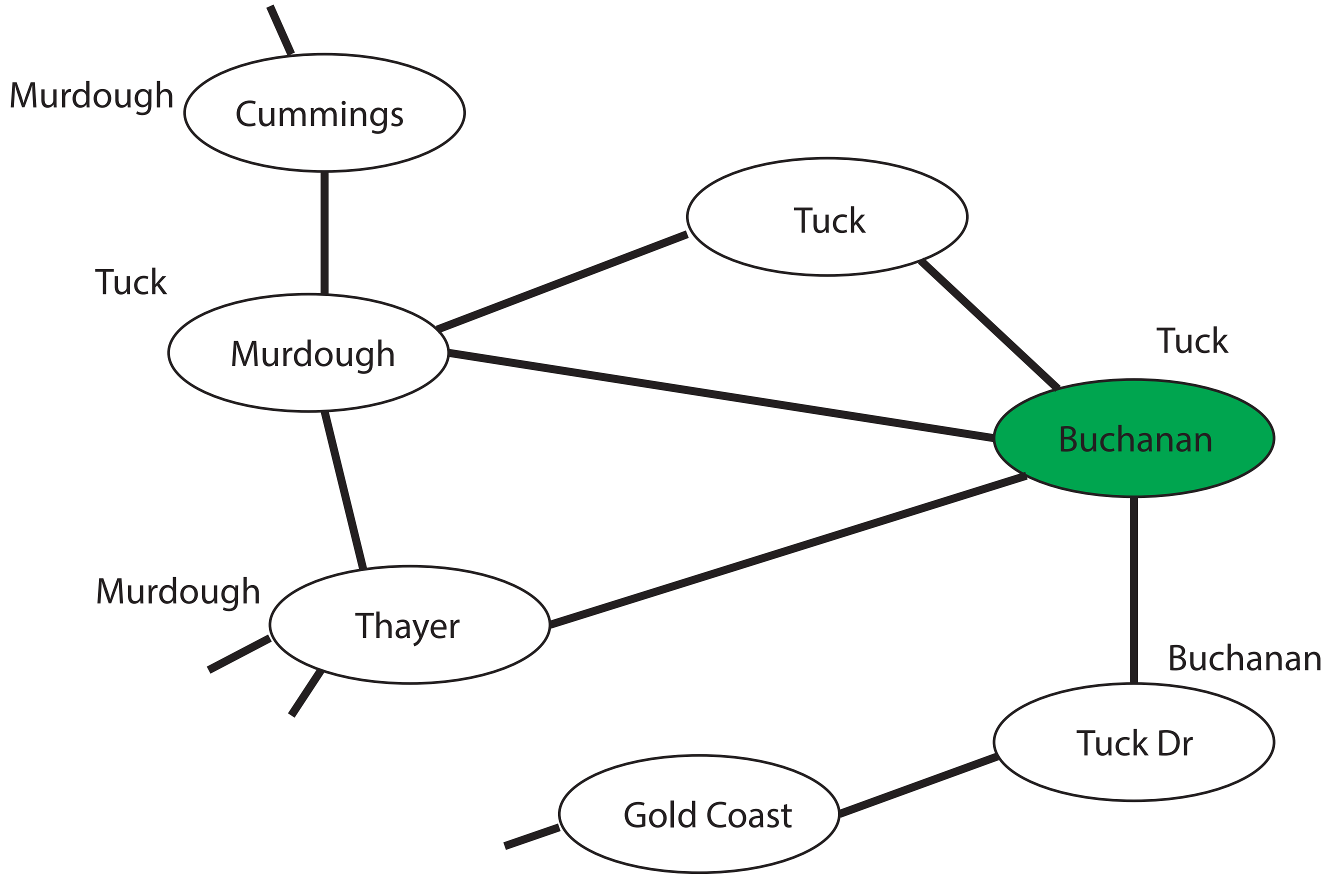
breadth-first exploration from Tuck

Buchanan was also claimed by Tuck, and starts producing minions of its own:



breadth-first exploration from Tuck

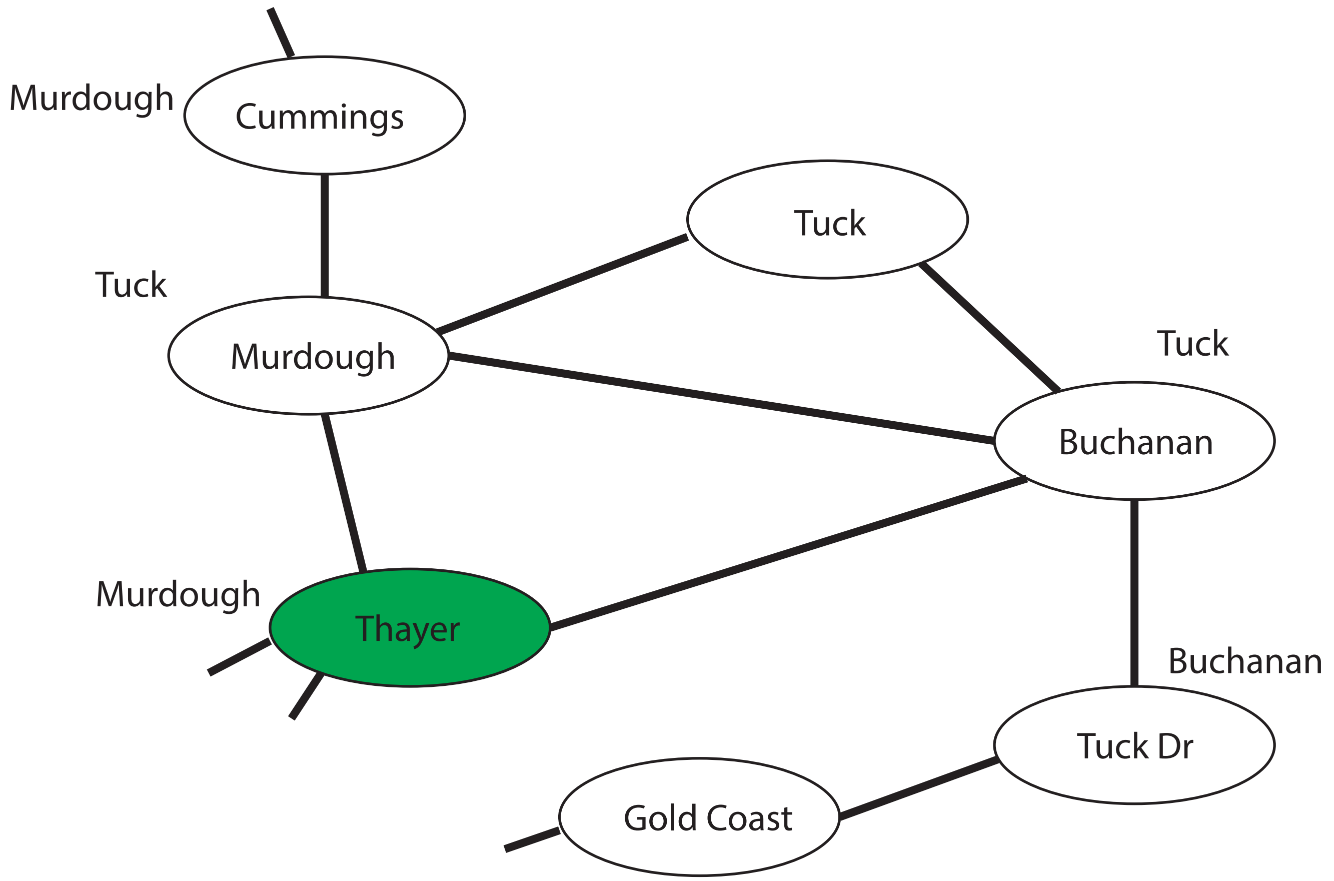
Buchanan was also claimed by Tuck, and starts producing minions of its own:



Notice: Buchanites do not claim Tuck, Murdough, or Thayer (already claimed). They do claim Tuck Dr.

breadth-first exploration from Tuck

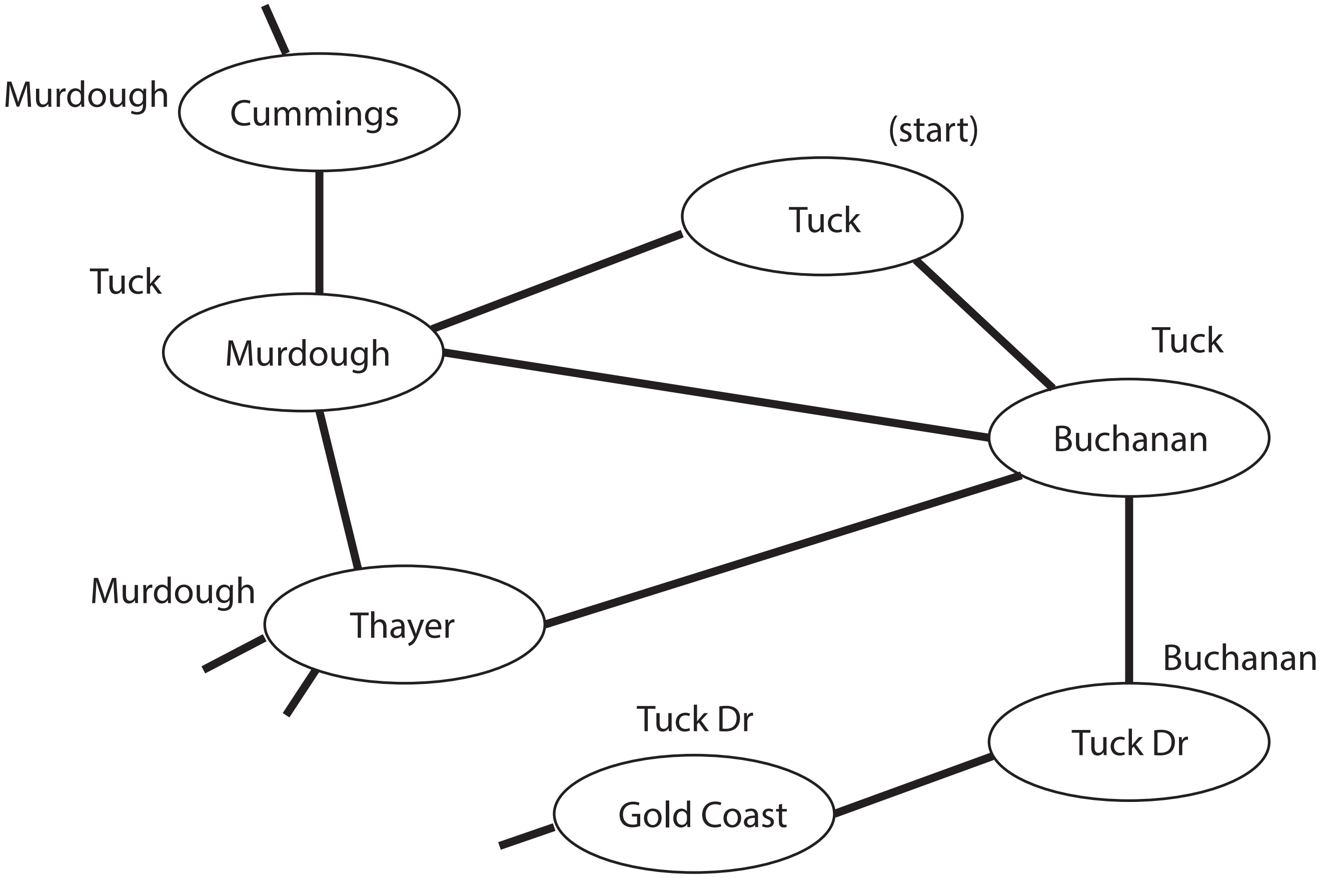
Thayer starts producing minions:



Continue this process until all nodes have been claimed.

finding the path with backchaining

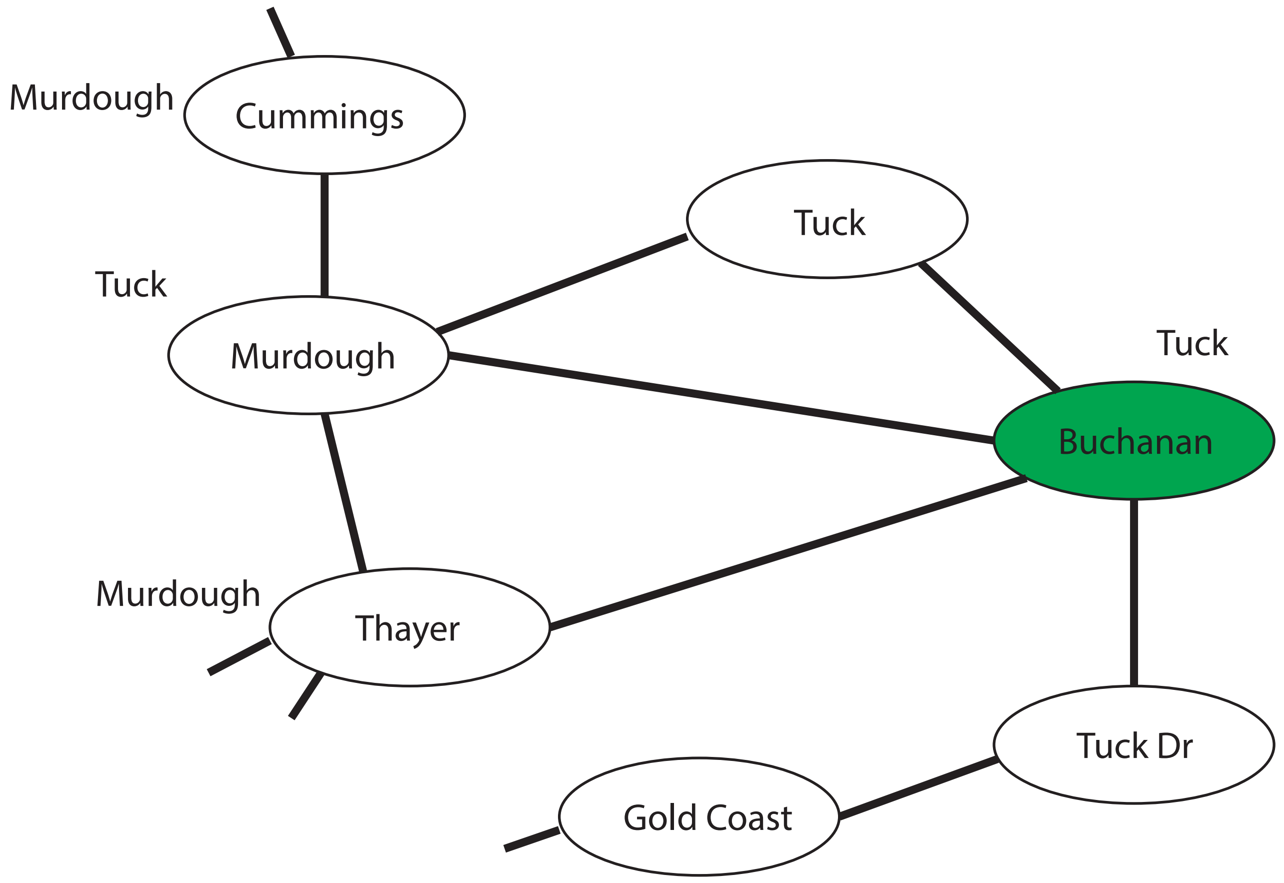
What is a fastest way from Goal Coast to Tuck?



Gold Coast was first claimed from Tuck Dr. Tuck Dr was first claimed by Buchanan. Buchanan was first claimed from Tuck.

Reverse this sequence: Tuck, Buchanan, Tuck Dr, Gold Coast.

breadth-first search: data structures



- Which node should produce minions next? We keep a **queue**.
- Which nodes have been reached first (claimed) from where? We keep a **dictionary**, `visitedFrom`.

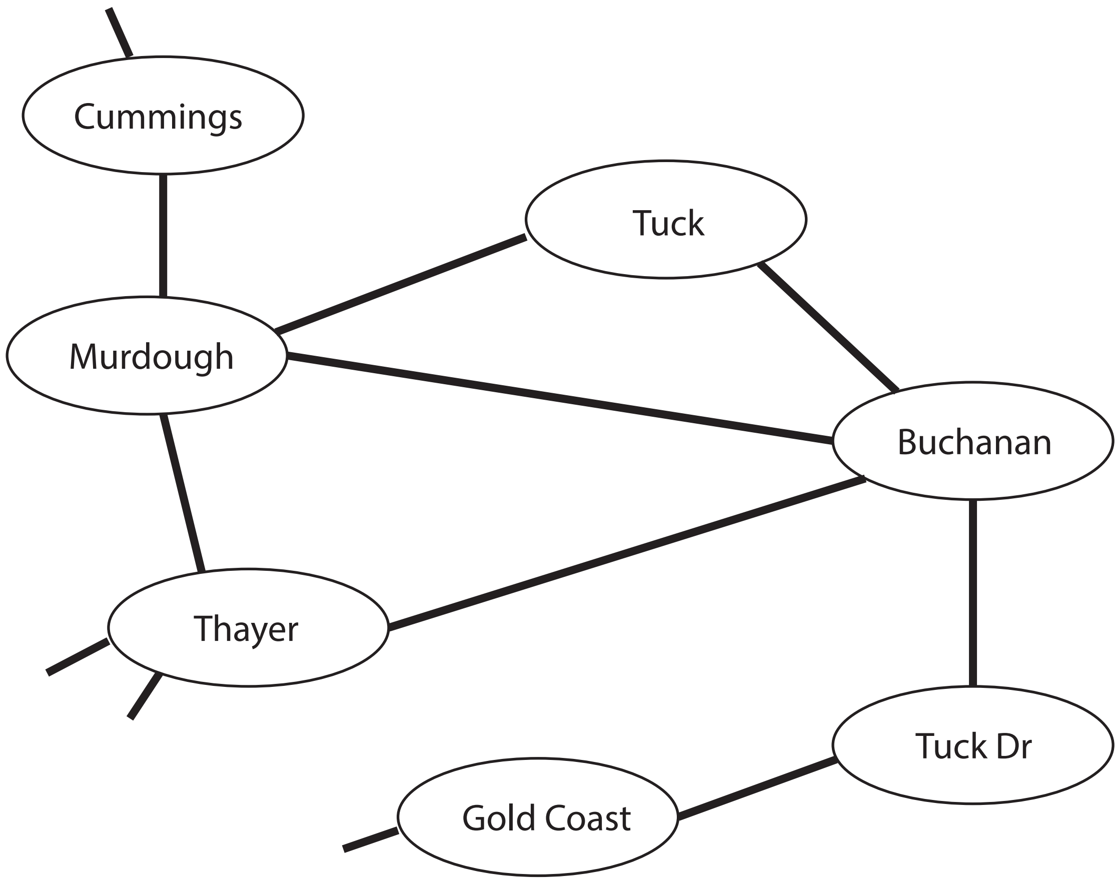
`visitedFrom[“Thayer”]` is `“Murdough”`

breadth-first search: pseudo-code

```
add starting node name to new queue (e.g., (”Tuck”))
create dictionary visitedFrom and add entry for starting name

while queue is not empty:
    dequeue current node name from the queue
    get the corresponding node from graphDict

    if the current node is the goal, success,
        backchain

    for each adjacent node name that is not in visitedFrom:
        add node name to queue for future exploration
        mark where node name was reached from in visitedFrom
```

bfs: data structures example

add start to queue and visitedFrom

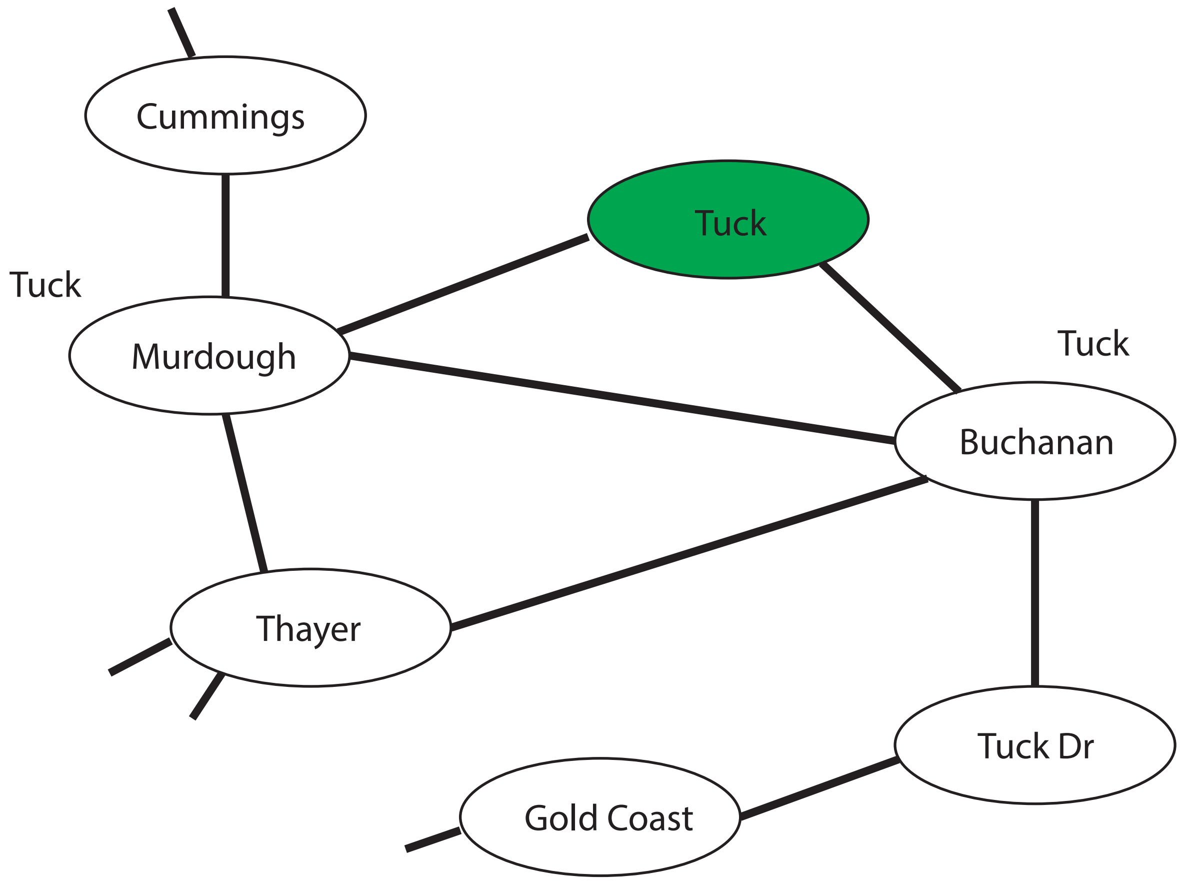


queue: “Tuck”

visitedFrom: {“Tuck”: “start”}

breadth-first exploration from Tuck

Start at Tuck. Send minions to claim adjacent nodes.

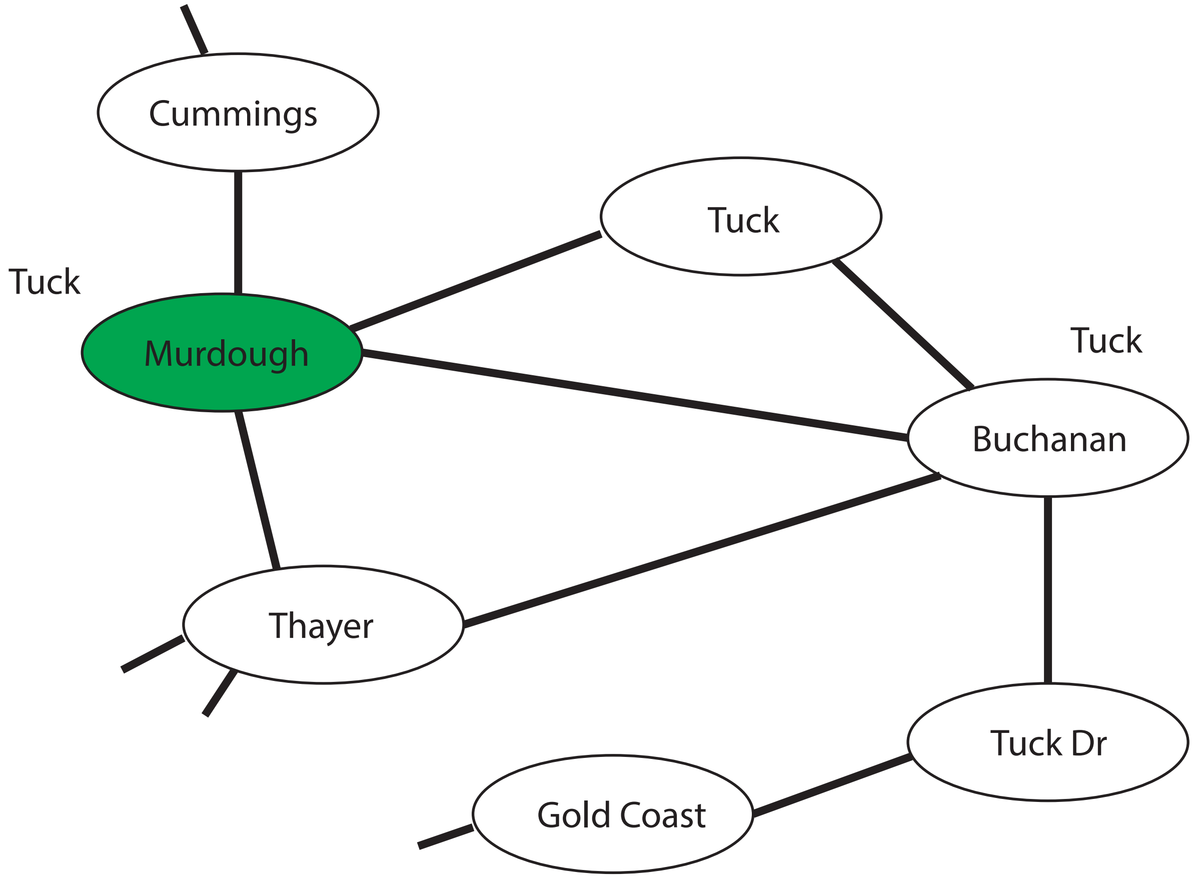


queue: “Murdough”, “Buchanan”

visitedFrom: {“Tuck”: “start”, “Murdough”: “Tuck”, “Buchanan”: “Tuck”}

breadth-first exploration from Tuck

Next, dequeue “Murdough”. Its adjacent node names are “Tuck”, “Cummings”, and “Thayer”. Since “Tuck” is already in visitedFrom, just add “Cummings” and “Thayer” to queue and visitedFrom.

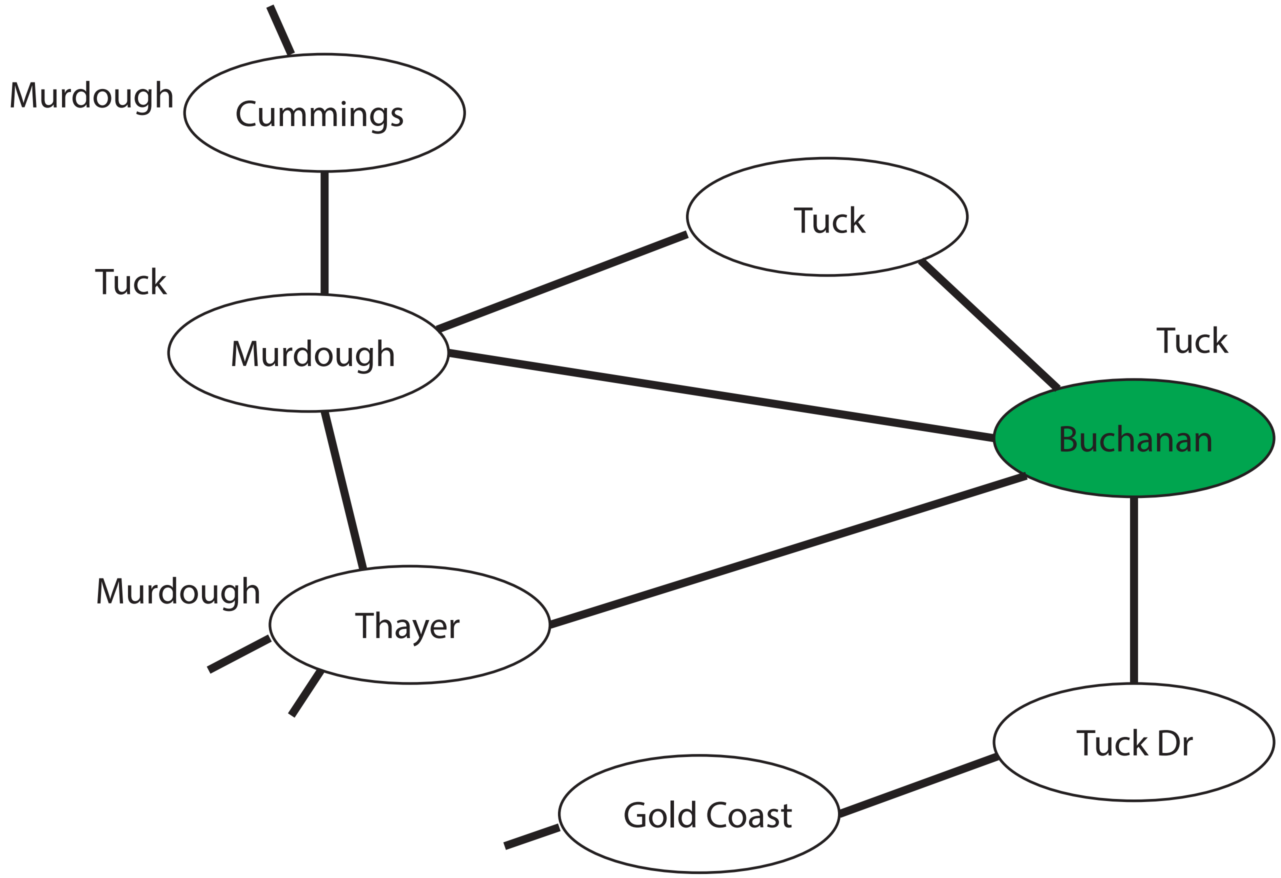


queue: “Buchanan”, “Cummings”, “Thayer”

visitedFrom: {“Tuck”: “start”, “Murdough”: “Tuck”, “Buchanan”: “Tuck”, “Cummings”: “Murdough”, “Thayer”: “Murdough”}

breadth-first exploration from Tuck

Buchanan is next in the queue. It will add Tuck Dr. to queue and visitedFrom.



queue: “Cummings”, “Thayer”, “Buchanan”

visitedFrom: {“Tuck”: “start”, “Murdough”: “Tuck”, “Buchanan”: “Tuck”, “Cummings”: “Murdough”, “Thayer”: “Murdough”, “Tuck Dr.”: “Buchanan”}

assignment suggestions

To get started:

1. Ignore the fancy graphics, use a test case starting at Tuck.
2. Make sure you understand graphDict and why you need it.
3. Terminate the while loop after a fixed number of steps (1? 2?)
4. Print out the contents of the queue and visitedFrom each time you modify one of them. Do they both contain strings? Are the strings what you expect?
5. Once things are working, terminate the while loop using the empty queue condition (to handle when goal is not in the graph) an if/return pair (to handle when the goal was found).
6. Don’t worry about backchaining until you are absolutely certain that visitedFrom is correct when the goal is found.