Introduction to Computer Science

CSCI 109

Readings

St. Amant, Ch. 4

“An **algorithm** (pronounced AL-go-rith-um) is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850.”

Andrew Goodney

Fall 2019

Reminders

- HW #1 due tomorrow.
- Grading: After HW#1 is graded, if you feel there has been a grading error, you have two options: #1 (best option) go to TA office hours to discuss the problem. This will give you a chance to get feedback on your answer while also resolving the dispute. Option #2: post a private note on Piazza and the graders will look into the issue.
- HW#2 out later today

Where are we?

Date	Topic		Assigned	Due	Quizzes/Midterm/Final	Slide Deck
26-Aug	Introduction	What is computing, how did computers come to be?				1
2-Sep	Labor day					
9-Sep	Computer architecture	How is a modern computer built? Basic architecture and assembly	HW1			2
16-Sep	Data structures	Why organize data? Basic structures for organizing data			**Quiz 1** on material taught in class 8/26 and 9/9	3
23-Sep	Data structures	Trees, Graphs and Traversals	HW2	HW1		4
30-Sep	More Algorithms/Data Structures	Recursion and run-time				5
7-Oct	Complexity and combinatorics	How "long" does it take to run an algorithm. P vs NP			**Quiz 2** on material taught in class 9/16 and 9/23	5
14-Oct	Algorithms and programming	Programming, languages and compilers		HW2	**Quiz 3** on material taught in class 9/30	7
21-Oct	Operating systems	What is an OS? Why do you need one?	HW3		**Quiz 4** on material taught in class 10/7	8
28-Oct	Midterm	Midterm			**Midterm** on all material taught so far.	
4-Nov	Computer networks	How are networks organized? How is the Internet organized?		HW3		9
11-Nov	Artificial intelligence	What is AI? Search, plannning and a quick introduction to machine learning			**Quiz 5** on material taught in class 9/4	10
18-Nov	The limits of computation	What can (and can't) be computed?	HW4		**Quiz 6** on material taught in class 11/11	11
25-Nov	Robotics	Robotics: background and modern systems (e.g., self-driving cars)			**Quiz 7** on material taught in class 11/18	12
2-Dec	Summary, recap, review	Summary, recap, review for final		HW4	**Quiz 8** on material taught in class 11/25	13
13-Dec	Final exam 11 am - 1 pm in SGM 123				**Final** on all material covered in the semester	

Problem Solving

- Architecture puts the computer under the microscope
- Computers are used to solve problems
- Abstraction for problems
  - How to represent a problem ?
  - How to break down a problem into smaller parts ?
  - What does a solution look like ?
- Two key building blocks
  - Algorithms
  - Abstract data types

Algorithms

- Algorithm: a step by step description of actions to solve a problem
- Typically at an abstract level
- Analogy: clearly written recipe for preparing a meal

"Algorithms are models of procedures at an abstract level we decided is appropriate." [St. Amant, pp. 53]

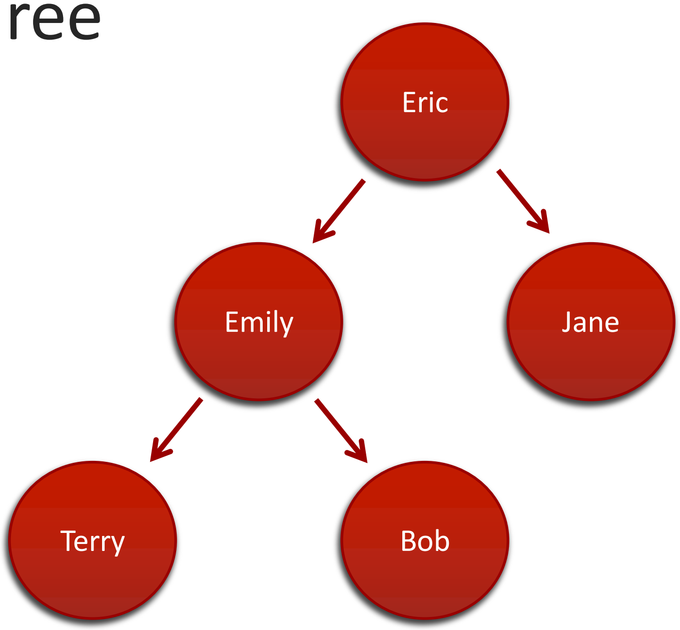
Abstract Data Types

- Models of collections of information
- Typically at an abstract level

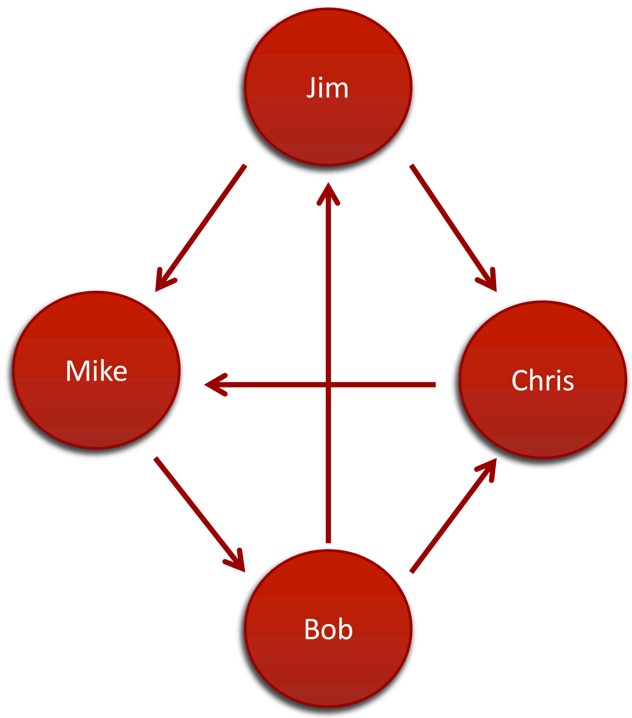
"… describes what can be done with a collection of information, without going down to the level of computer storage." [St. Amant, pp. 53]

Sequences, Trees and Graphs

- Sequence: a list
  - Items are called elements
  - Item number is called the index
- Tree

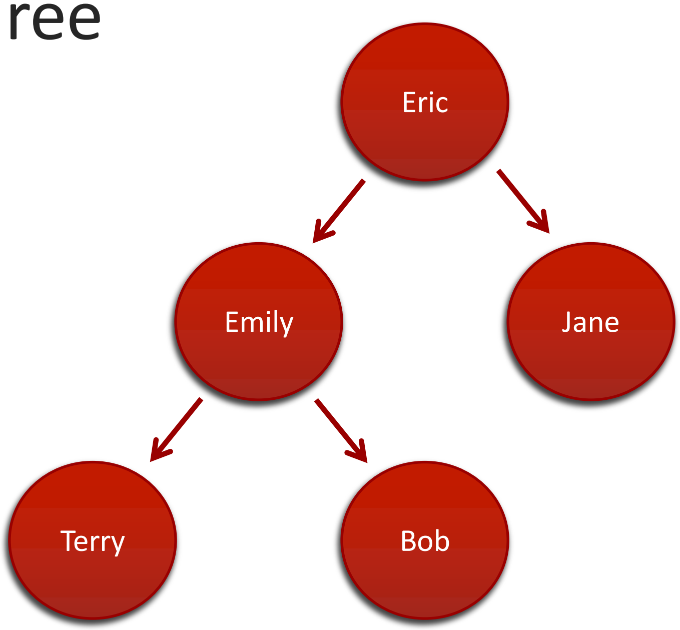


- Graph



Sequences, Trees and Graphs

- Sequence: a list
  - Items are called elements
  - Item number is called the index
- Tree



- Lists
  - Searching
    - Unsorted list
    - Sorted list
  - Sorting
    - Selection sort
    - Quicksort
- The notion of a brute force algorithm
- The divide and conquer strategy

Motivation for Abstract Data Structures (Graphs, Trees)

- The nature of some data, and the way we need to accesses it often requires some structure, or organization to make things efficient (or even possible)
- Data: large set of people and their family relationship used for genetic research
- Problems: two people share a rare genetic trait, how closely are the related? (motivates for a tree)

Motivation for Abstract Data Structures (Graphs, Trees)

- Data set: roads and intersections.
- Problem: how to travel from A to B @5pm on a Friday? How to avoid traffic vs. prefer freeways? (motivates a weighted graph)
- Data set: freight enters country at big port (LA/Long Beach).
- Problem: How to route freight given train lines/connections?
  - Route fastest, vs. lowest cost?
- Data set: airport locations
- Problem: how to route and deliver a package to any address in the US with minimum cost? Think UPS, FedEx

Motivation for Abstract Data Structures (Graphs, Trees)

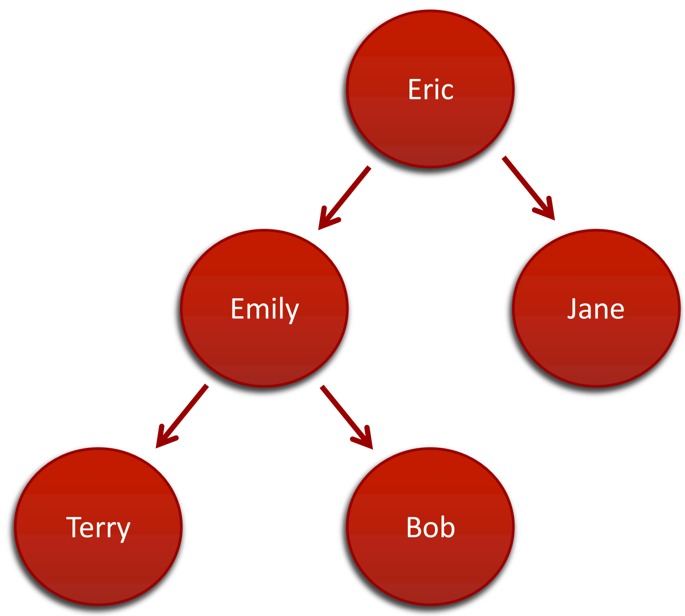
- Data set: network switches and their connectivity (network links)
- Problem: Chose a subset of network links that connect all switches without loops (networks don’t like loops). Motivates graphs, and graph -> tree algorithm

Motivation for Abstract Data Structures (Graphs, Trees)

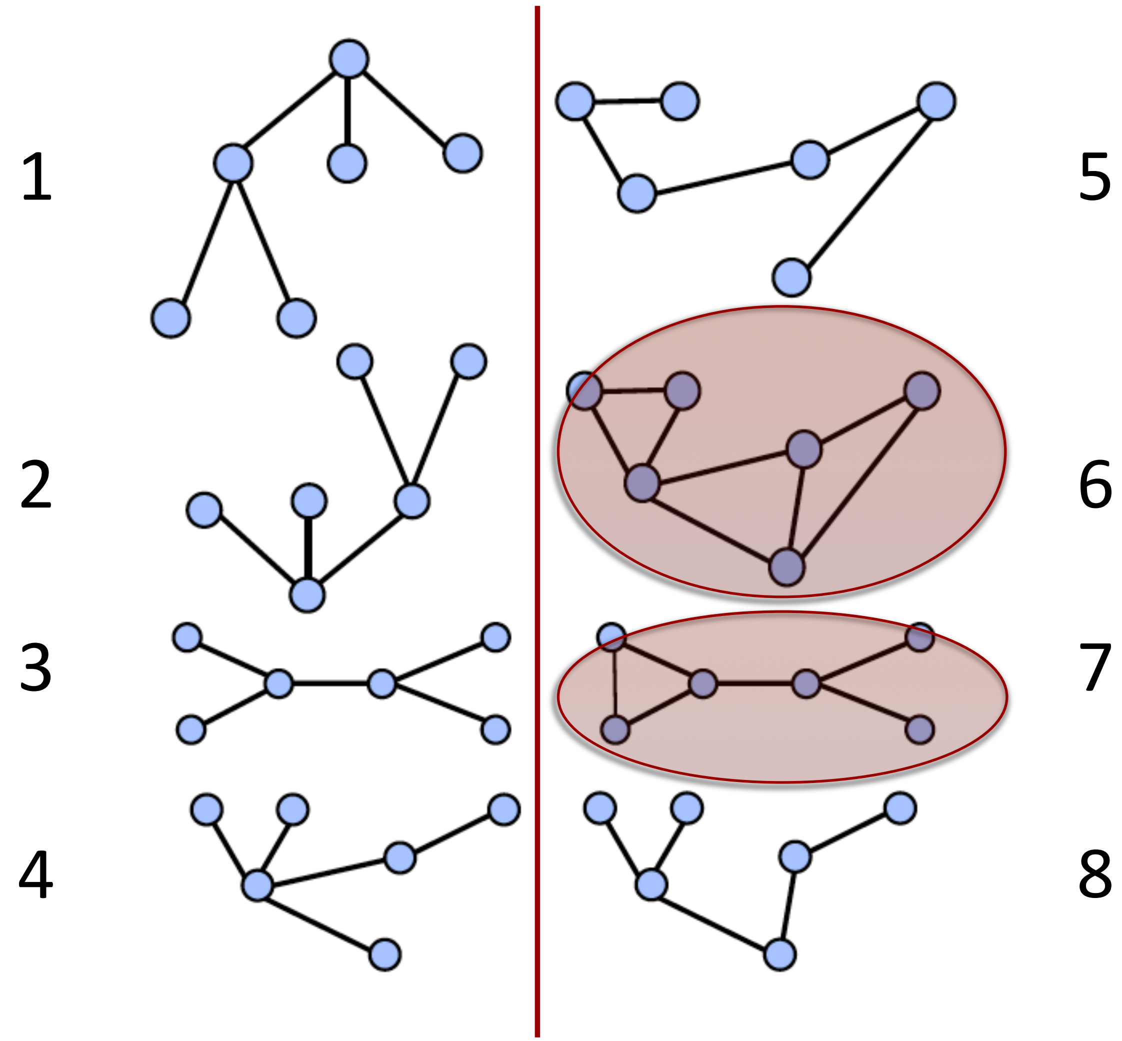
- Data set: potential solutions to a big problem
- Problem: how to find an optimal solution to the problem, without searching every possibility (solution space too big). Motivates graphs and graph search to solve problems.
- Other data/problems that motivate graphs/trees:
  - Financial networks and money flows, social networks, rendering HTML code, compilers, 3D graphics and game engines… and more

Trees

- Each node/vertex has exactly one parent node/vertex
- No loops
- Directed (links/edges point in a particular direction)
- Undirected (links/edges don't have a direction)
- Weighted (links/edges have weights)
- Unweighted (links/edges don't have weights)



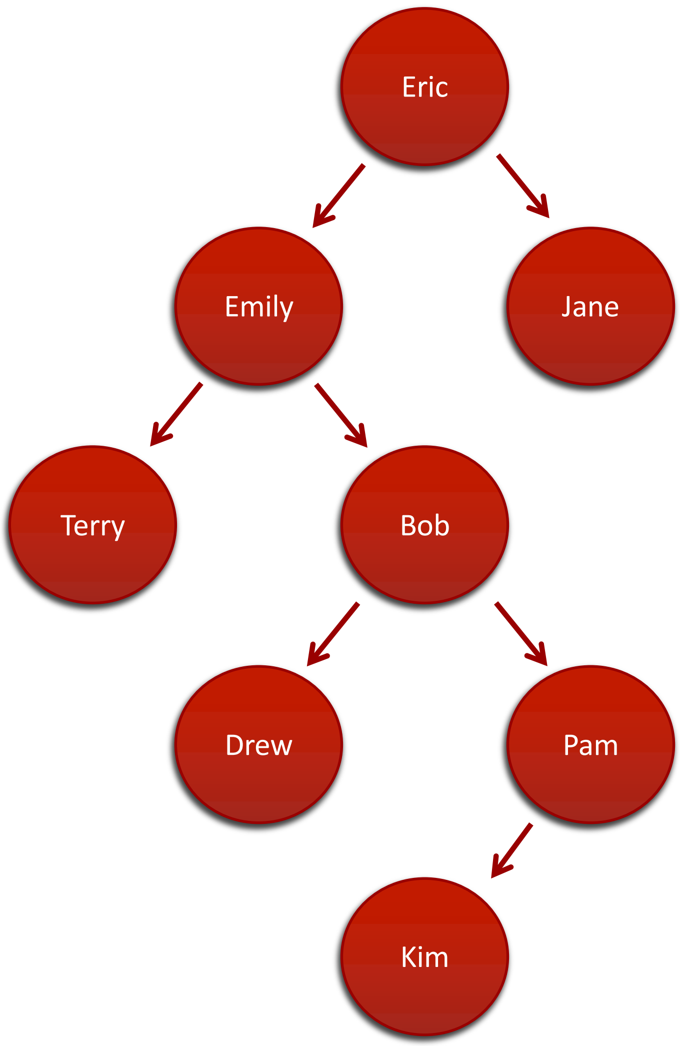
Which of these are NOT trees?



Graph/Tree Traversal

- Traversing a graph or a tree: “moving” and examining the nodes to enumerate the nodes or look for solutions
- Example: find all living descendants of X in our genetic database.
- For traversing a graph we pick a starting node, then two methods are obvious:
  - Depth first
    - Go as deep (far away from starting node) as possible before backtracking
  - Breadth first
    - Examine one layer at a time

Tree Traversal



- Depth first traversal

  Eric, Emily, Terry, Bob, Drew, Pam, Kim, Jane

- Breadth first traversal

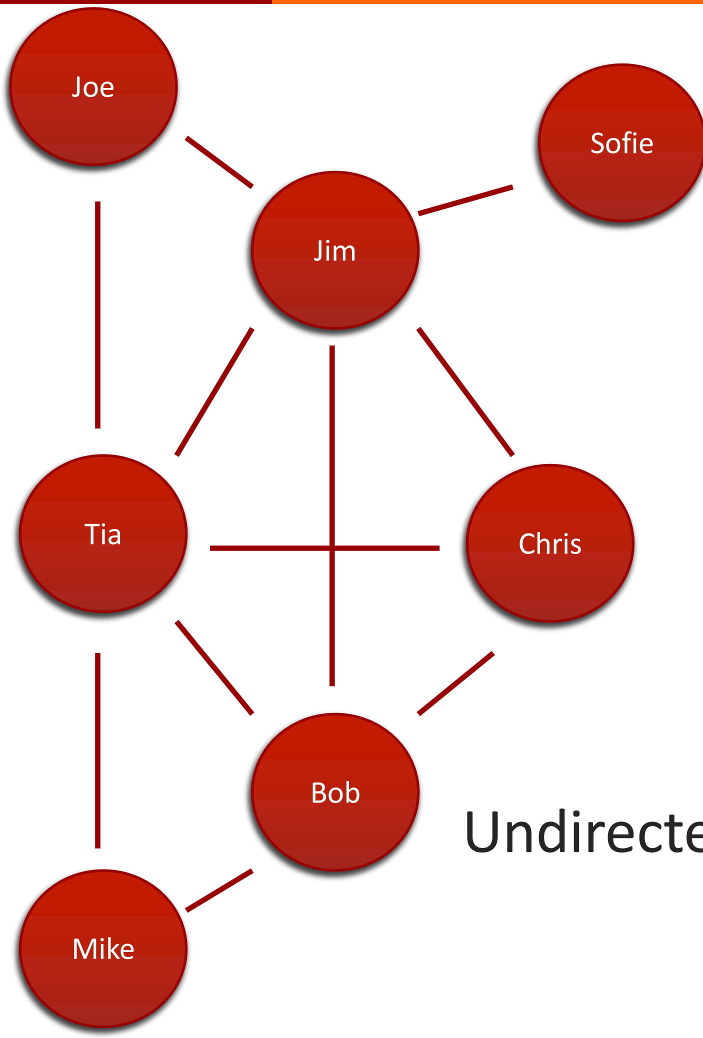
  Eric, Emily, Jane, Terry, Bob, Drew, Pam, Kim

  Eric, Jane, Emily, Bob, Terry, Pam, Drew, Kim

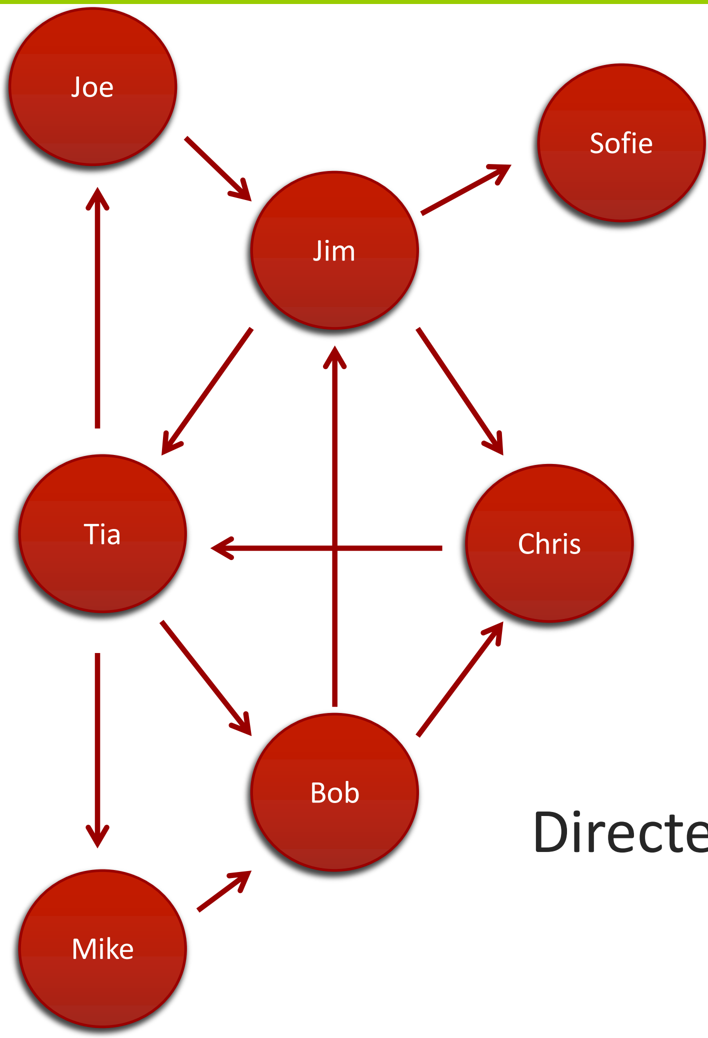
Tree Traversal

- Depth first vs. Breadth first eventually visit all nodes, but do so in a different order
- Used to answer different questions
  - Depth first: good for game trees, evaluating down a certain path
  - Breadth first: look for shortest path between two nodes (e.g for computer networks)
- Roughly:
  - Depth first: find ‘a’ solution to the problem
  - Breadth first: find ‘the’ solution to the problem

Graphs: Directed and Undirected



Undirected



Directed

Graph to Tree Conversion Algorithms

- Sometimes the question is best answered by a tree, but we have a graph
- Need to convert graph to tree (by deleting edges)
- Usually want to create a “spanning tree”

Spanning Trees

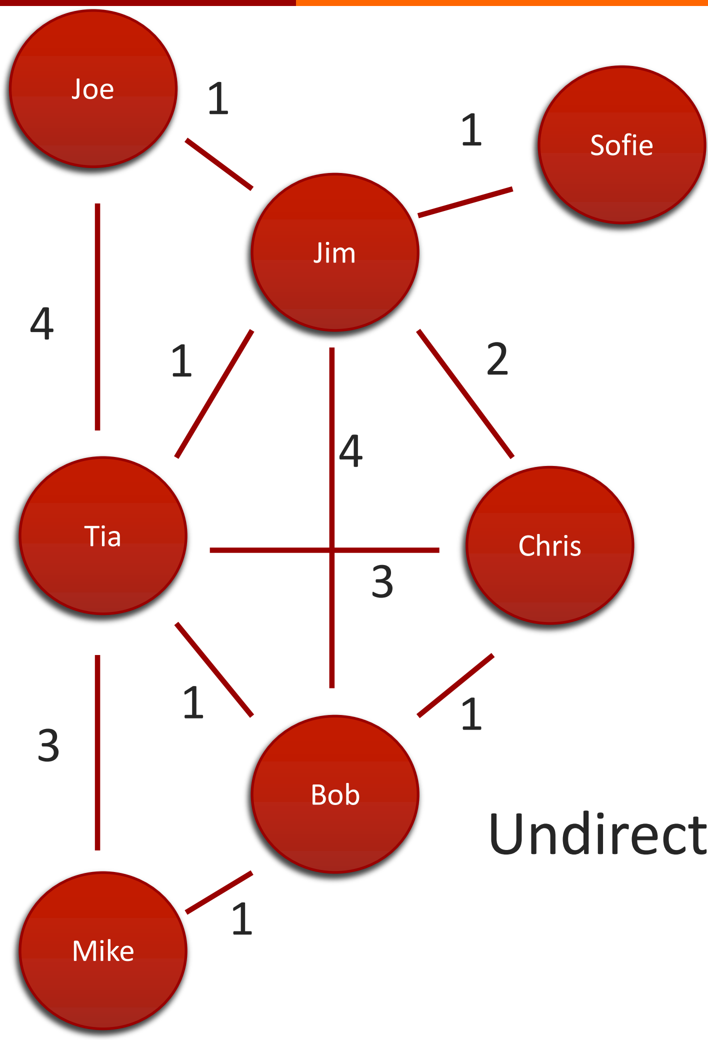
- Spanning tree: Any tree that covers all vertices
  - "Cover" = "include" in graph-speak
- Example: graph of social network connections. Want to create a "phone tree" to disseminate information in the event of an emergency
- Example: network of switches with redundant links and multiple paths between switches (there are loops aka cycles in the graph). Need to chose a set of links that connects all switches with no loops.

Minimum Spanning trees

- Spanning tree: Any tree that covers all vertices, not as common as the MST
- *Minimum* spanning tree (MST): Tree of minimal total edge cost
- If you have a graph with weighted edges, a MST is the tree where the sum of the weights of the edges is minimum
- There is at least one MST, could be more than one
- If you have unweighted edges any spanning tree is a MST

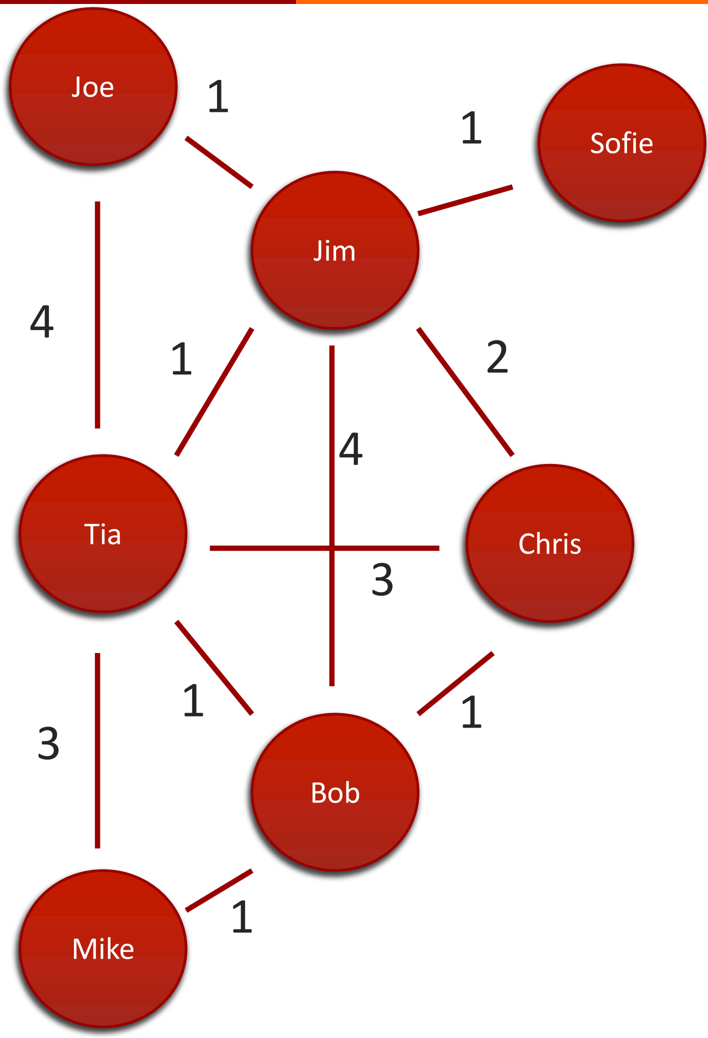
- Why compute the minimum spanning tree?
  - Minimize the cost of connections between cities (logistics/shipping)
  - Minimize of cost of wires in a layout (printed circuit, integrated circuit design)

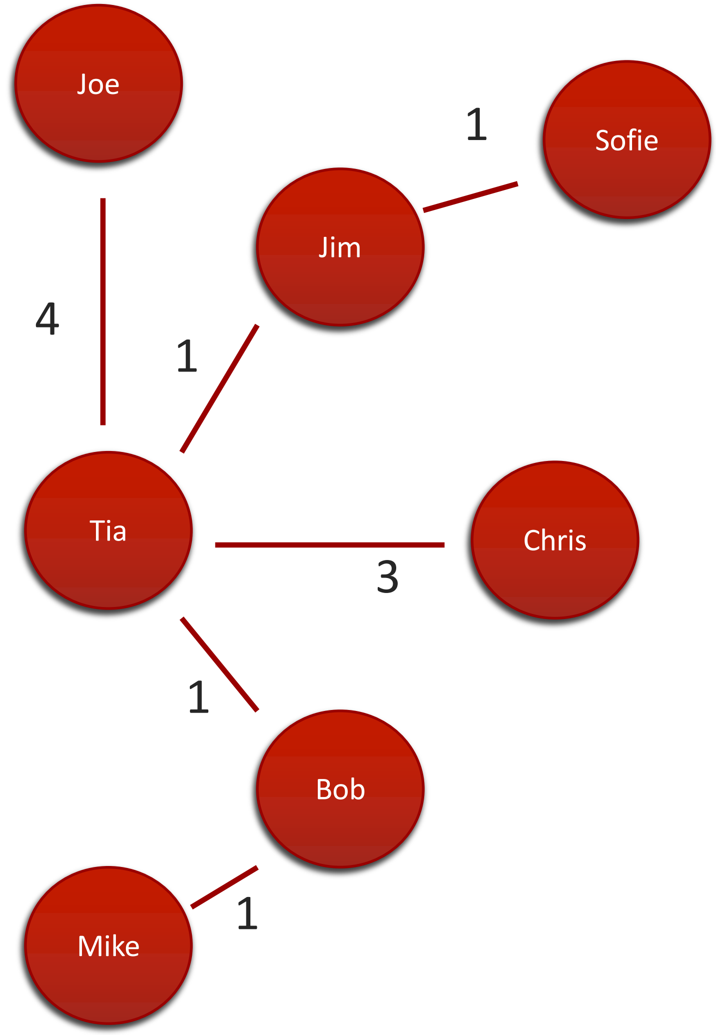
Edge costs, minimum spanning tree



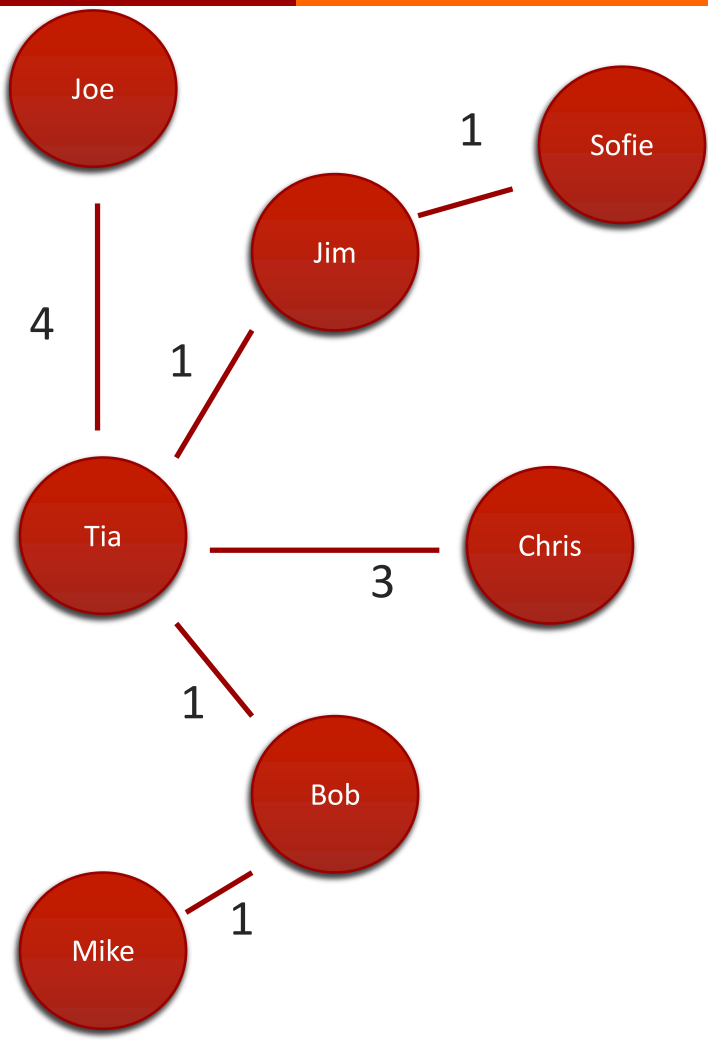
Undirected

Edge costs, minimum spanning tree

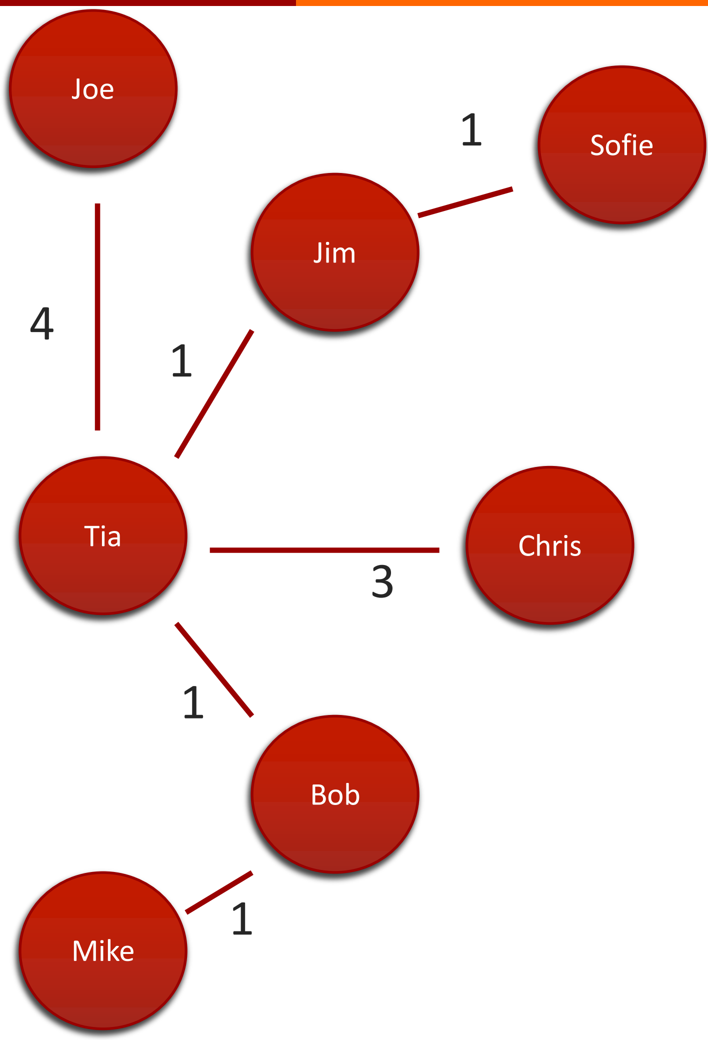


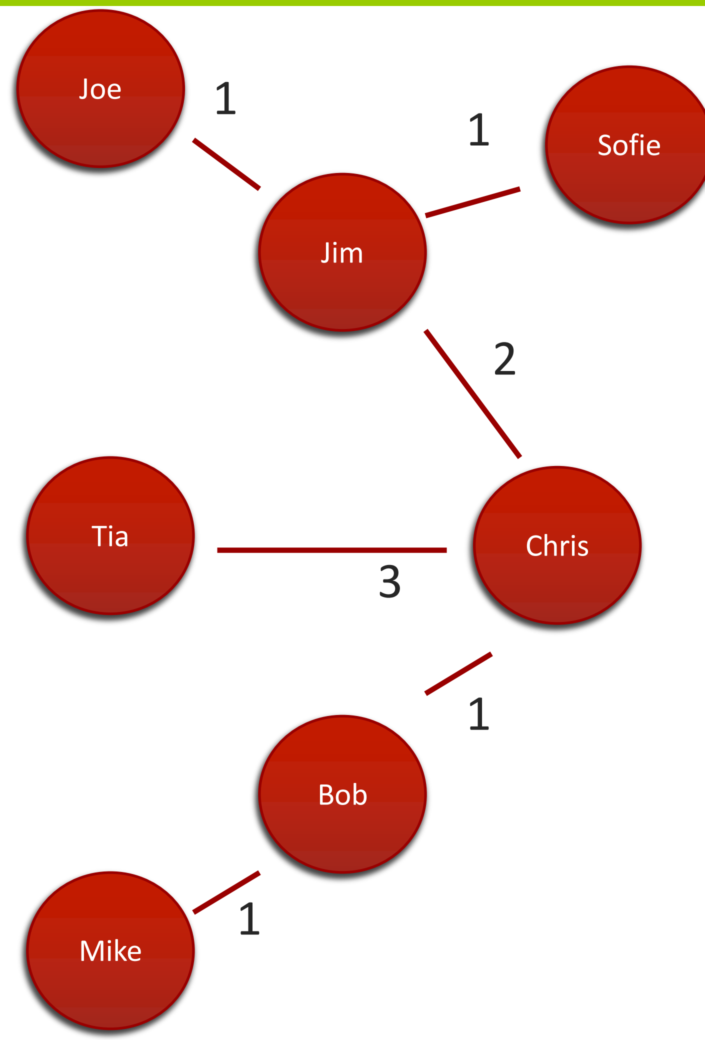


Spanning Trees



Spanning Trees





Computing the MST

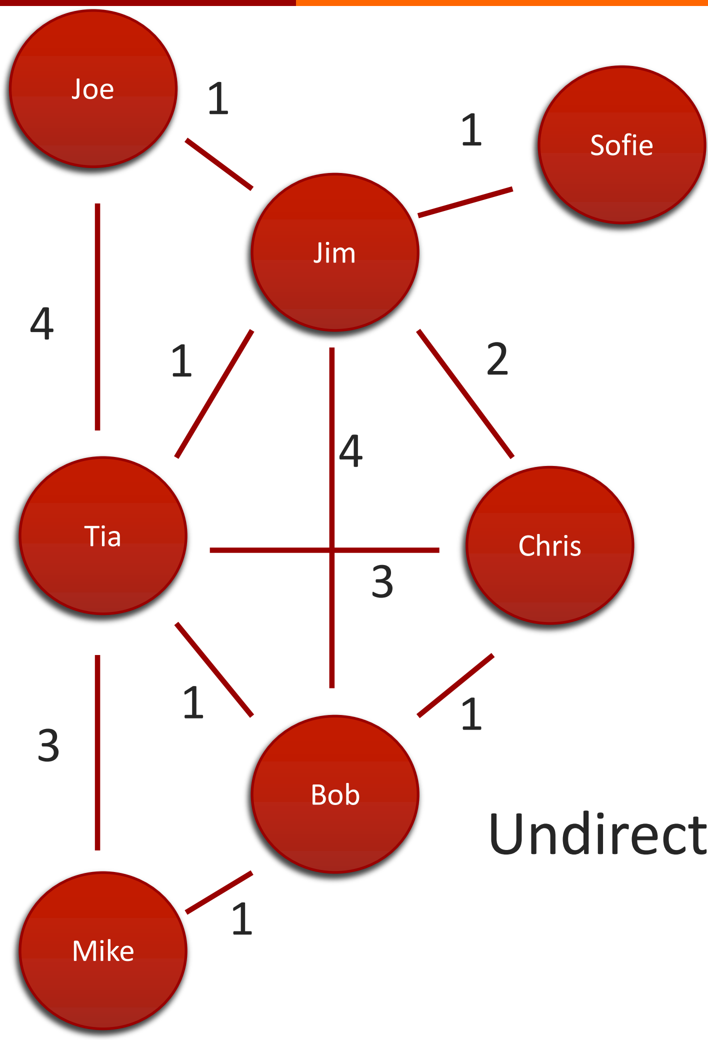
- Two greedy algorithms to compute the MST
  - Prim's algorithm: Start with any node and greedily grow the tree from there
  - Kruskal's algorithm: Order edges in ascending order of cost. Add next edge to the tree without creating a cycle.
- 'Greedy' means solution is refined at each step using the most obvious next step, with the hope that eventual solution is globally optimal

Prim's algorithm

- Initialize the minimum spanning tree with a vertex chosen at random.
- Find all the edges that connect the tree to new vertices (i.e uncovered, or disconnected), find the minimum and add it to the tree
- Keep repeating step 2 until all vertices are added to the MST

(adapted from: https://www.programiz.com/dsa)

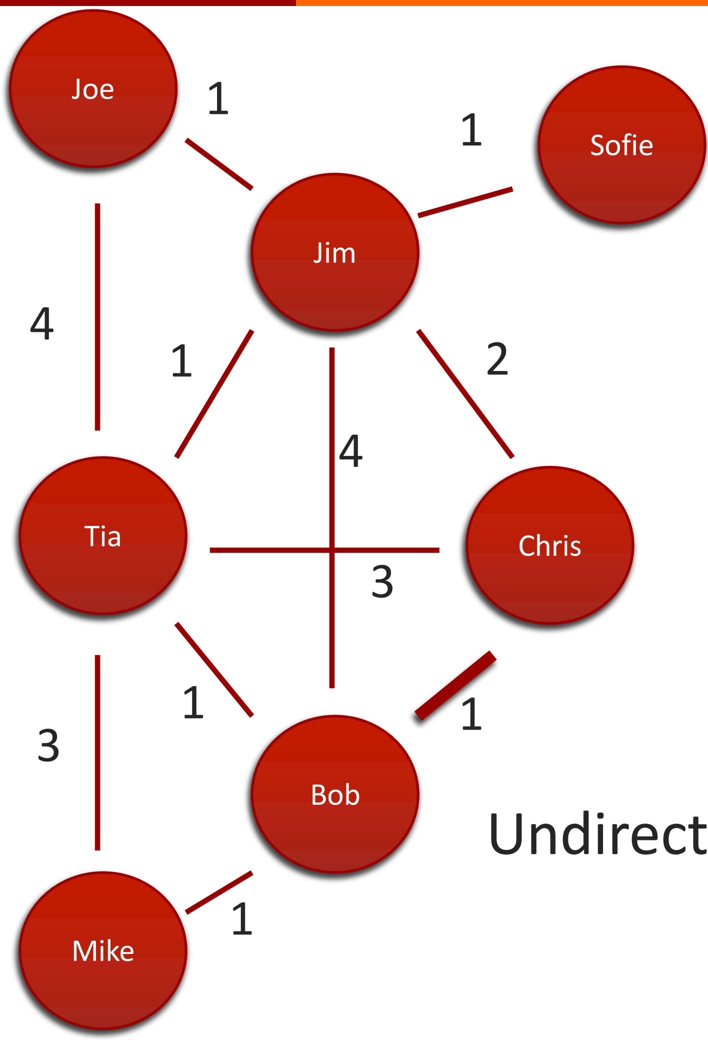
Prim's algorithm



Undirected

1	Joe-Jim
1	Jim-Sofie
1	Jim-Tia
1	Tia-Bob
1	Chris-Bob
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

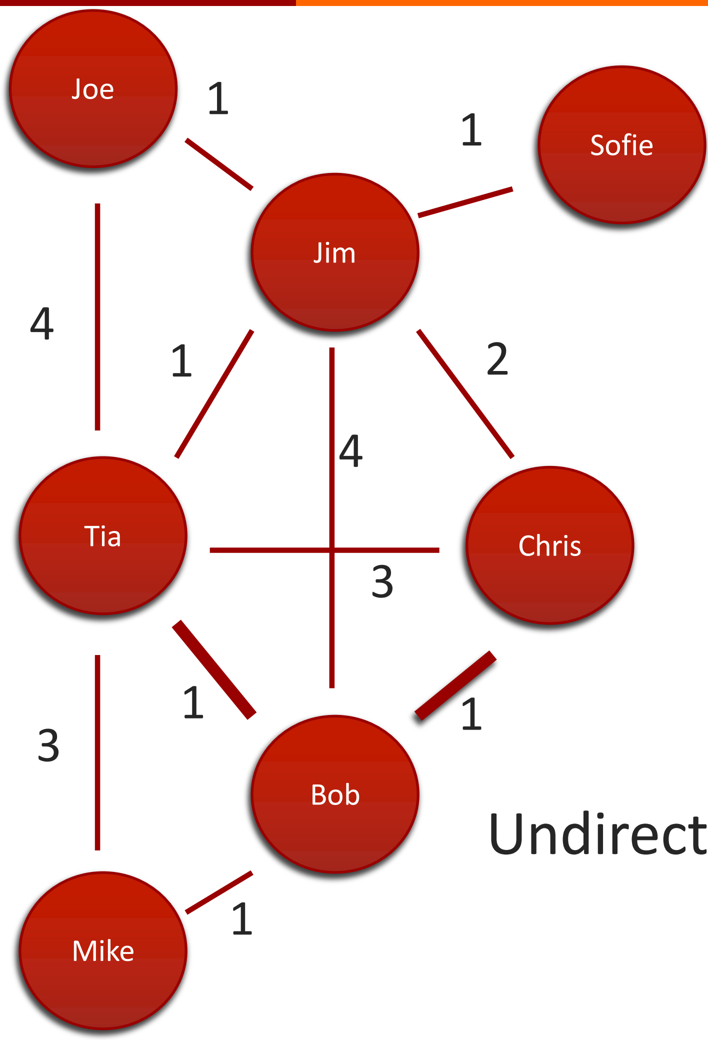
Prim's algorithm



Undirected

1	Joe-Jim
1	Jim-Sofie
1	Jim-Tia
1	Tia-Bob
1	**Chris-Bob**
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

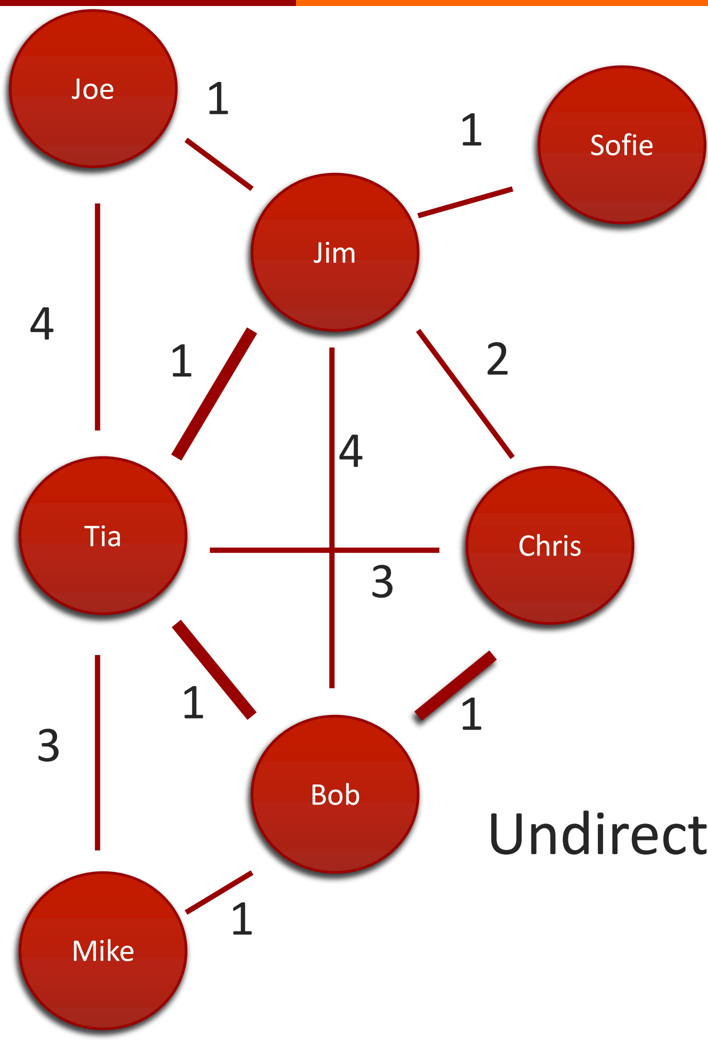
Prim's algorithm



Undirected

1	Joe-Jim
1	Jim-Sofie
1	Jim-Tia
1	**Tia-Bob**
1	**Chris-Bob**
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

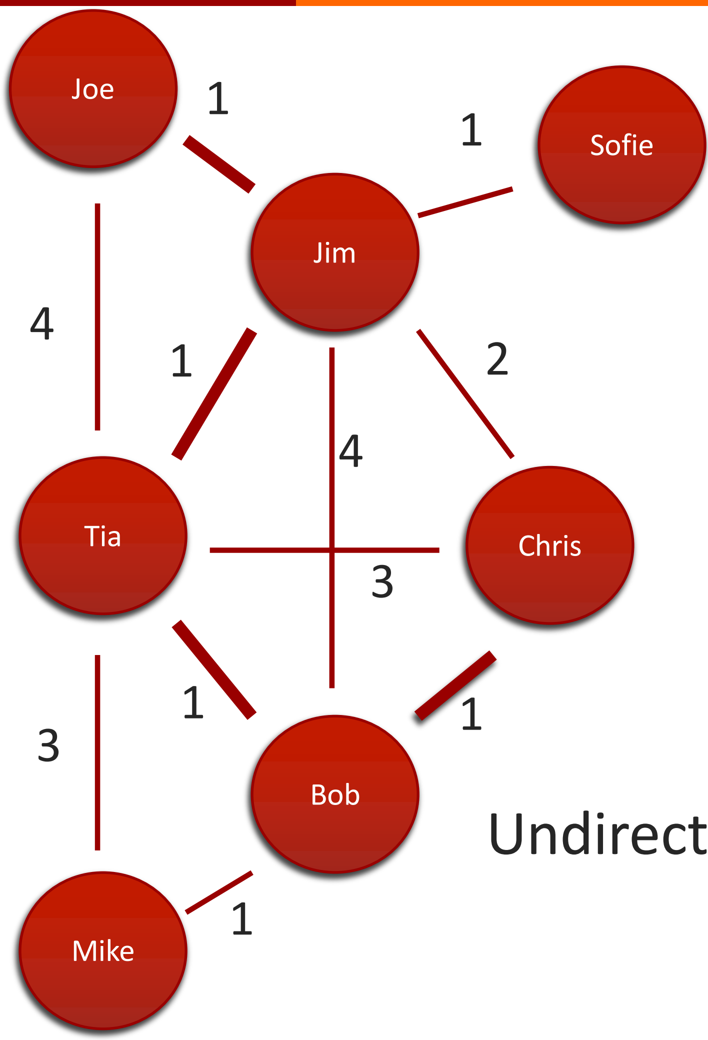
Prim's algorithm



Undirected

1	Joe-Jim
1	Jim-Sofie
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

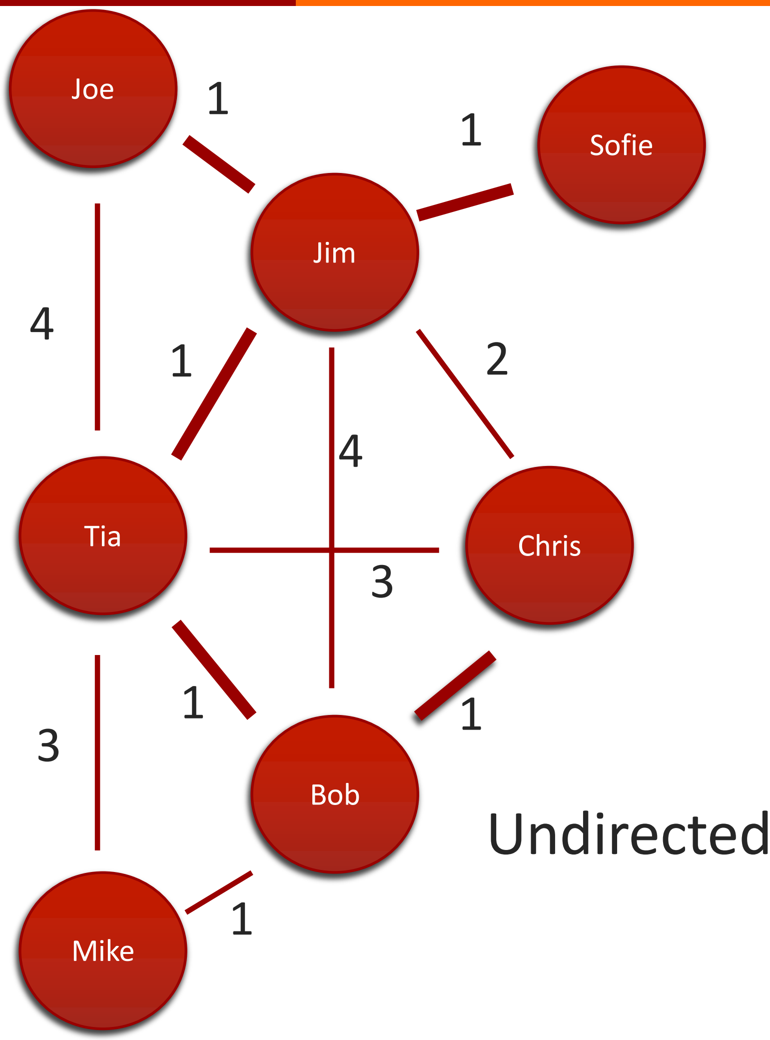
Prim's algorithm



Undirected

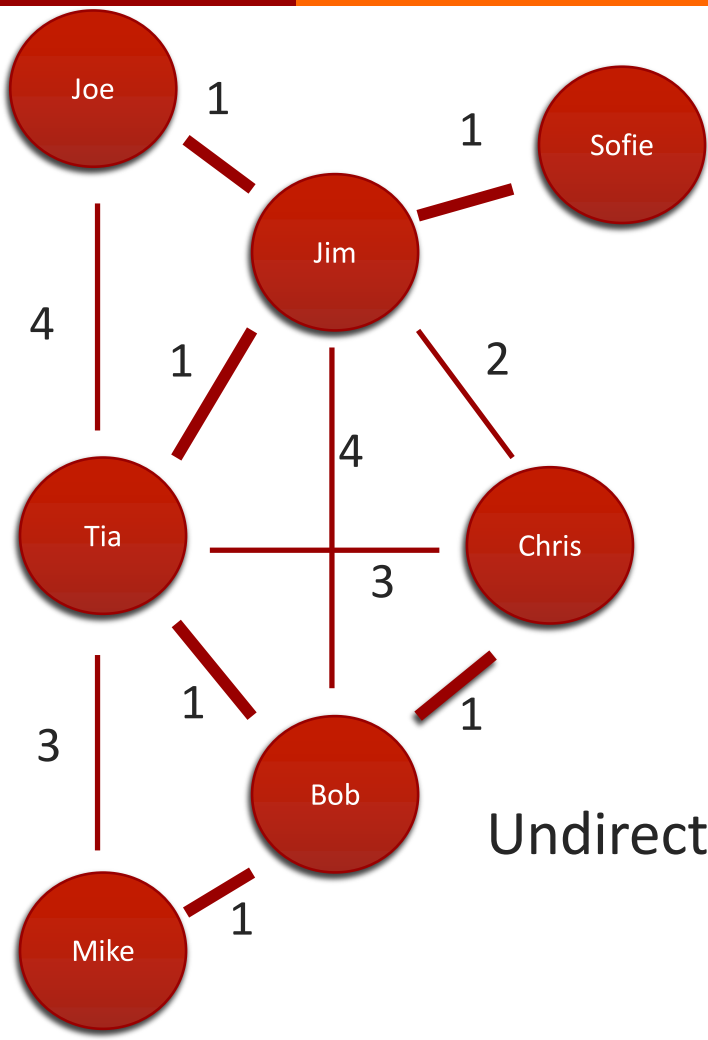
1	**Joe-Jim**
1	Jim-Sofie
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Prim's algorithm



1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Prim's algorithm



Undirected

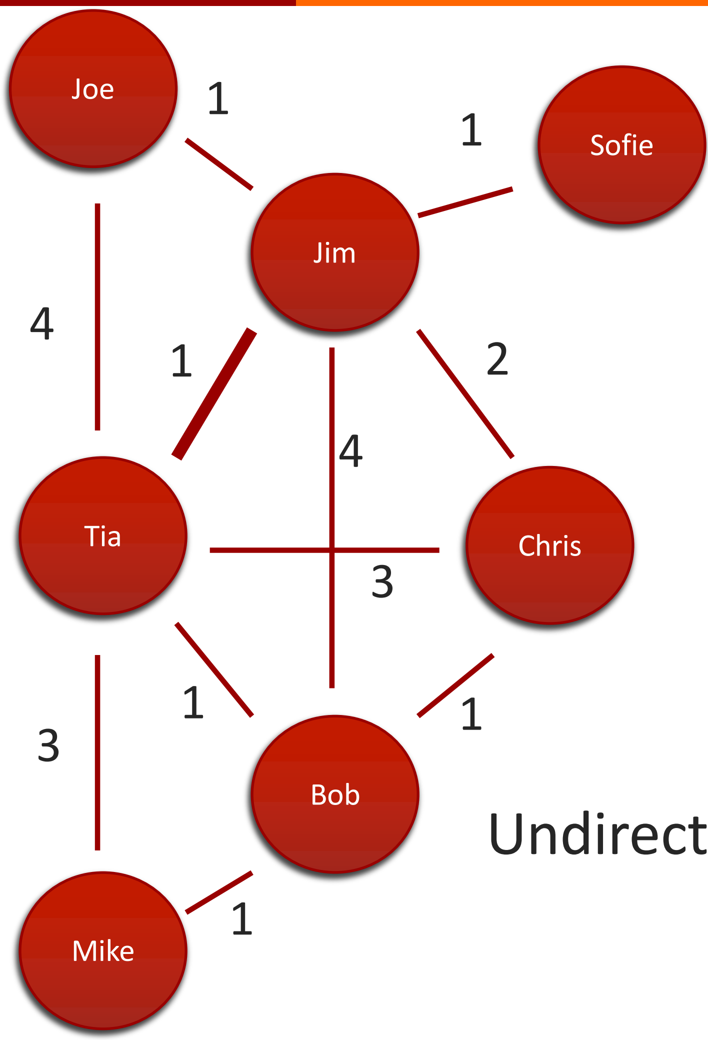
1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Kruskal's algorithm

- Sort all the edges from low weight to high
- Take the edge with the lowest weight, if adding the edge would create a cycle, then reject this edge and select the edge with the next lowest weight
- Keep adding edges until we reach all vertices.

(adapted from: https://www.programiz.com/dsa)

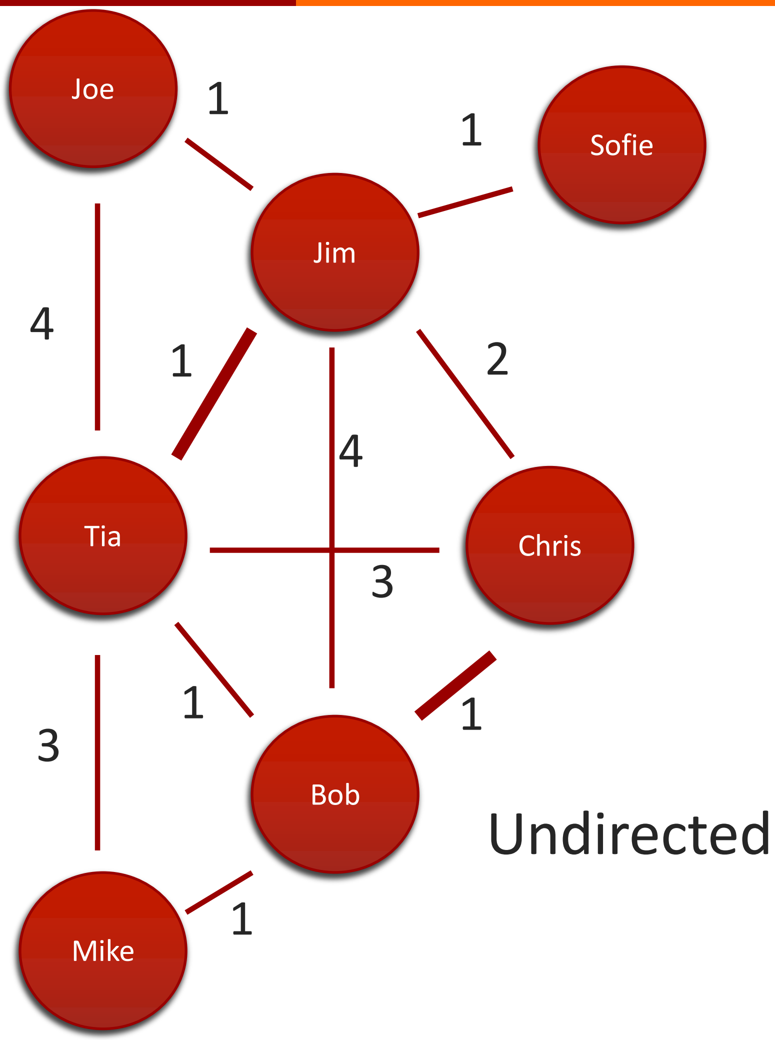
Kruskal's algorithm



Undirected

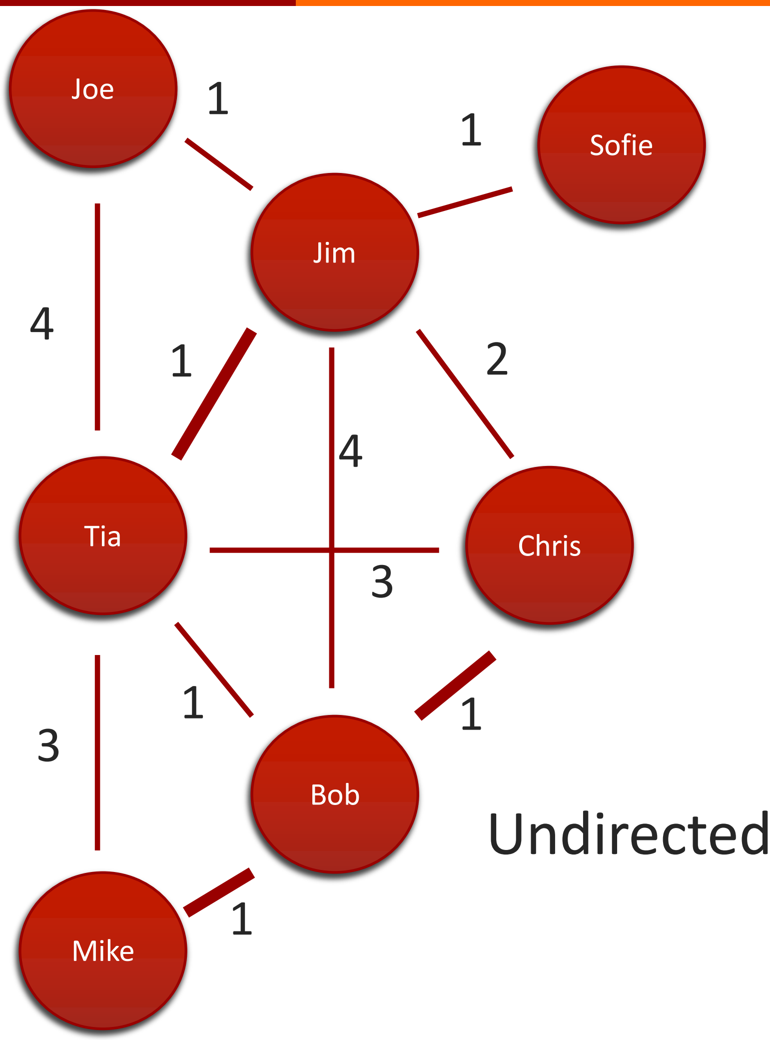
1	Joe-Jim
1	Jim-Sofie
1	**Jim-Tia**
1	Tia-Bob
1	Chris-Bob
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Kruskal's algorithm



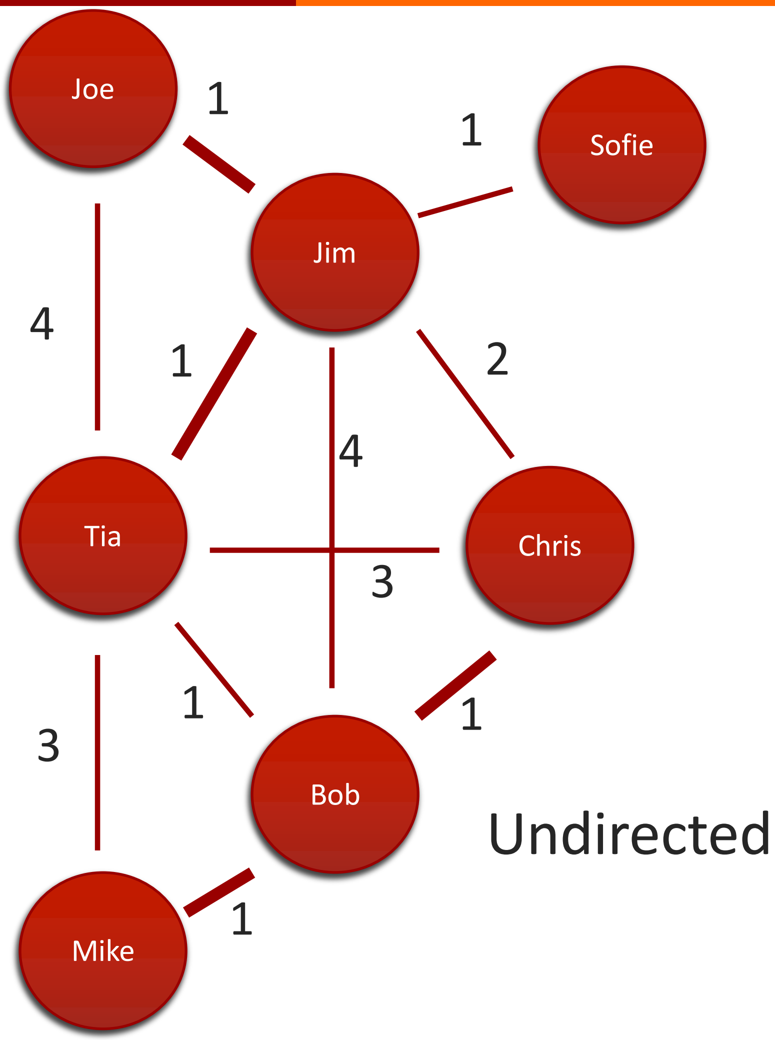
1	Joe-Jim
1	Jim-Sofie
1	**Jim-Tia**
1	Tia-Bob
1	**Chris-Bob**
1	Mike-Bob
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Kruskal’s algorithm



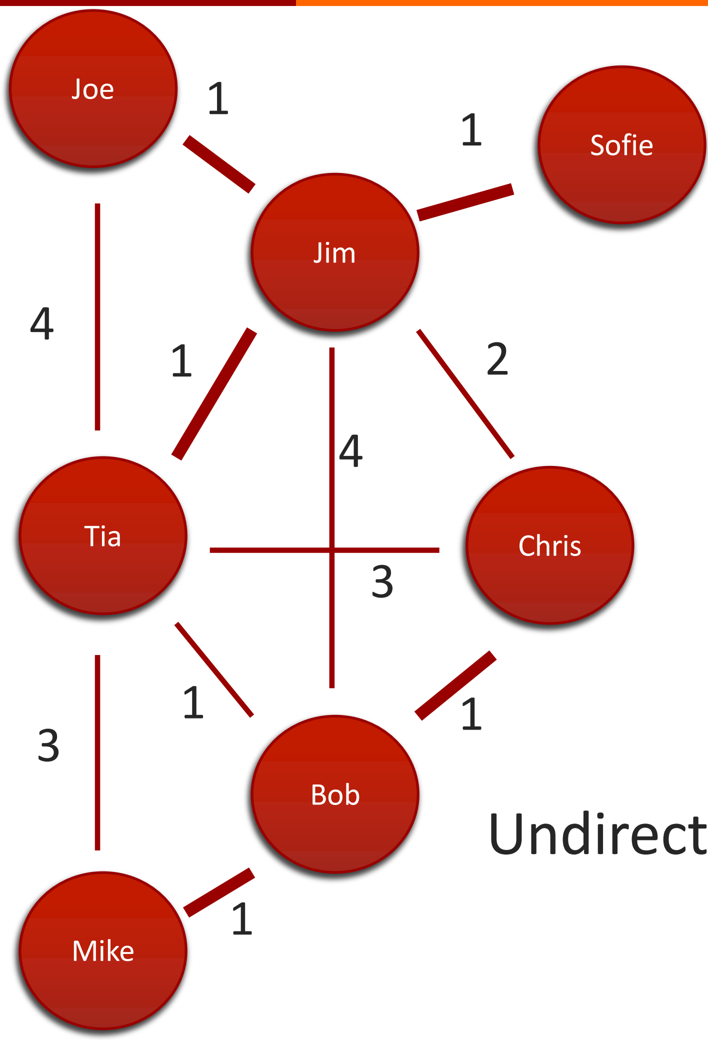
1	Joe-Jim
1	Jim-Sofie
1	**Jim-Tia**
1	Tia-Bob
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Kruskal’s algorithm



1	**Joe-Jim**
1	Jim-Sofie
1	**Jim-Tia**
1	Tia-Bob
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

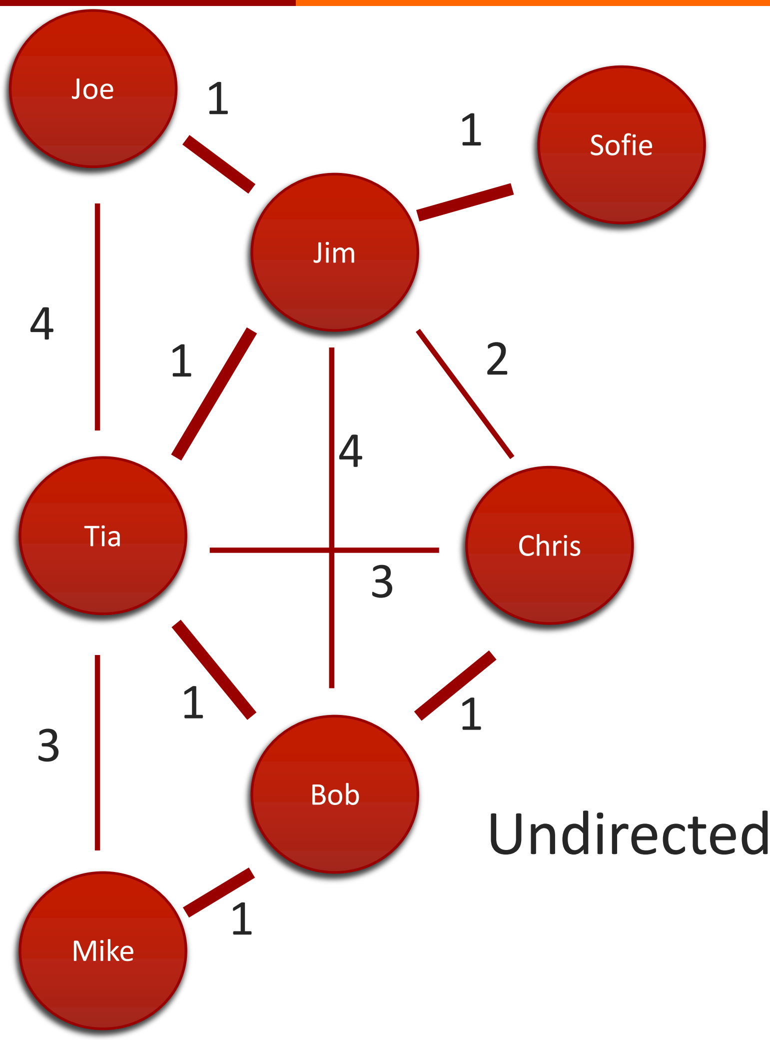
Kruskal's algorithm



Undirected

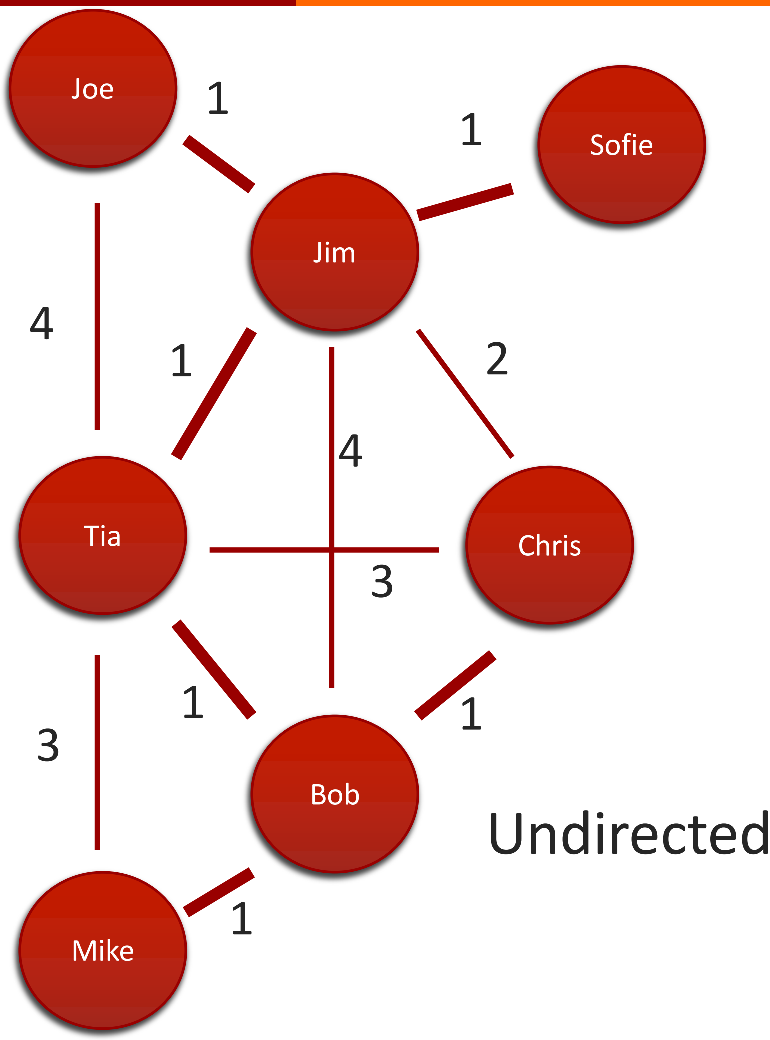
1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	Tia-Bob
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Kruskal's algorithm



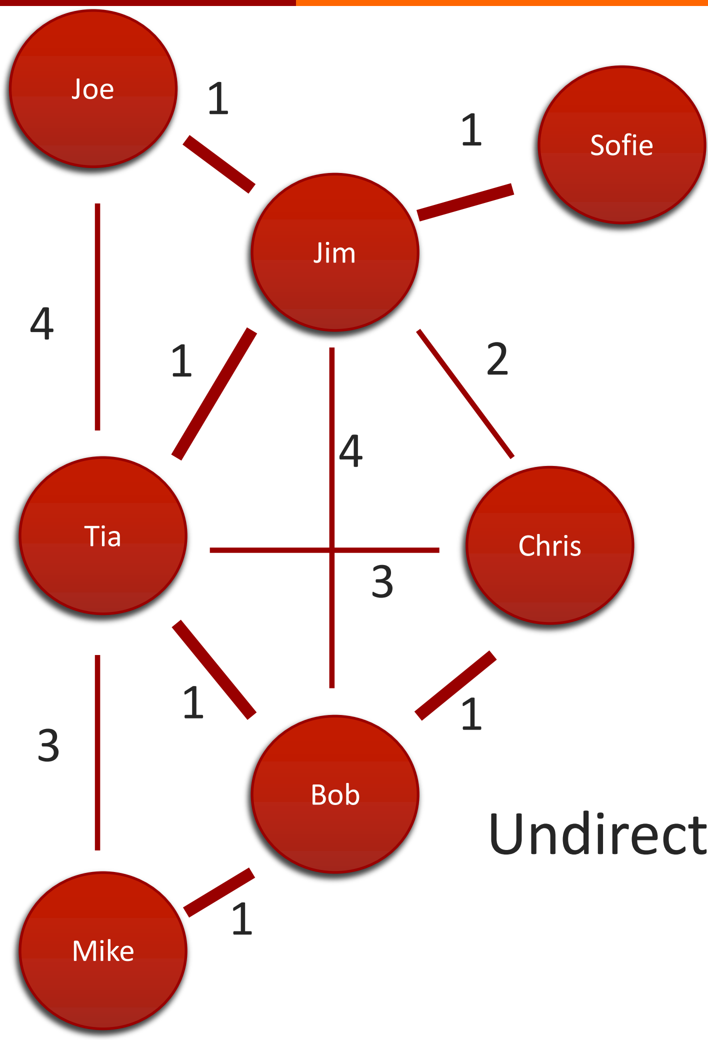
1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Kruskal's algorithm



1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

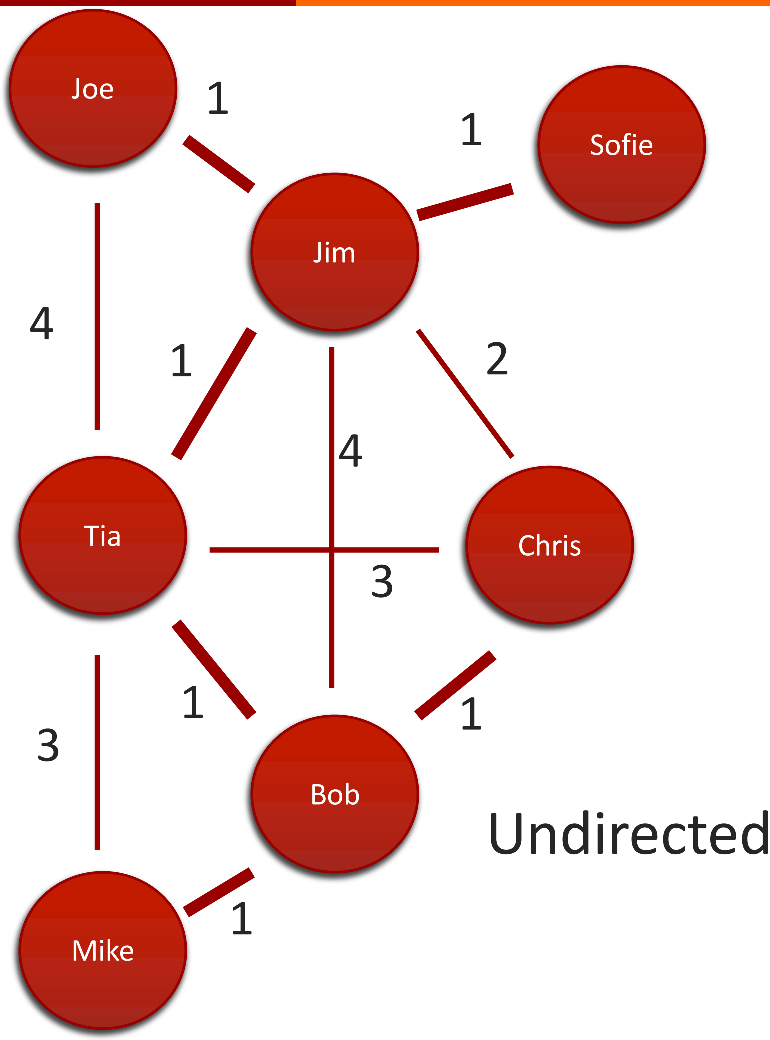
Kruskal's algorithm



Undirected

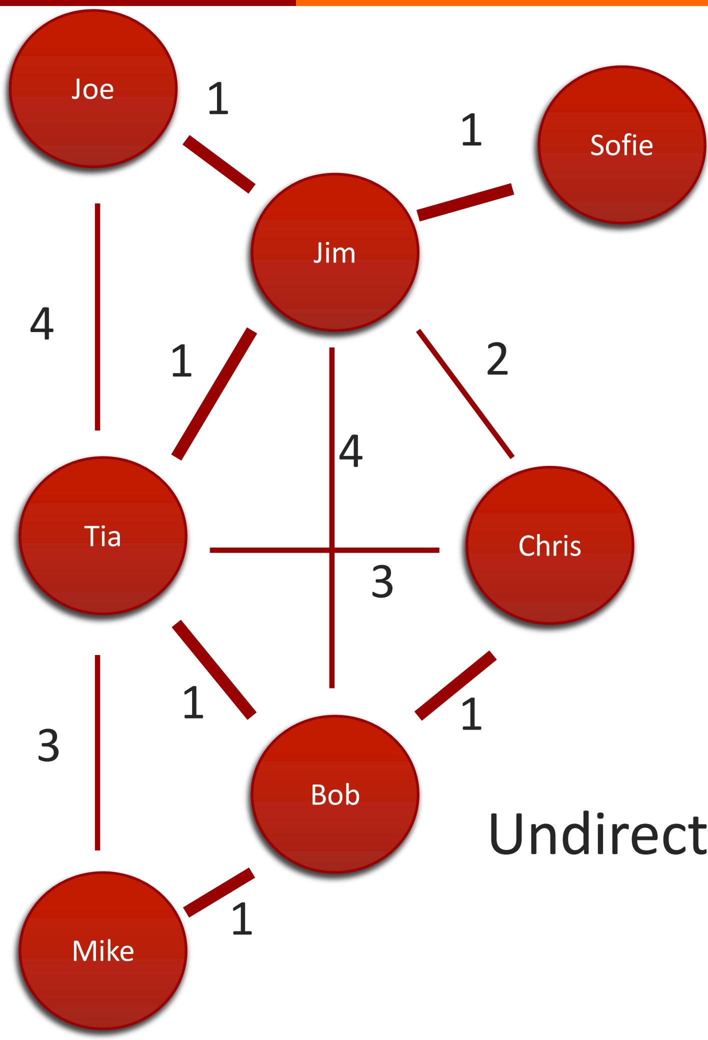
1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

Kruskal's algorithm



1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

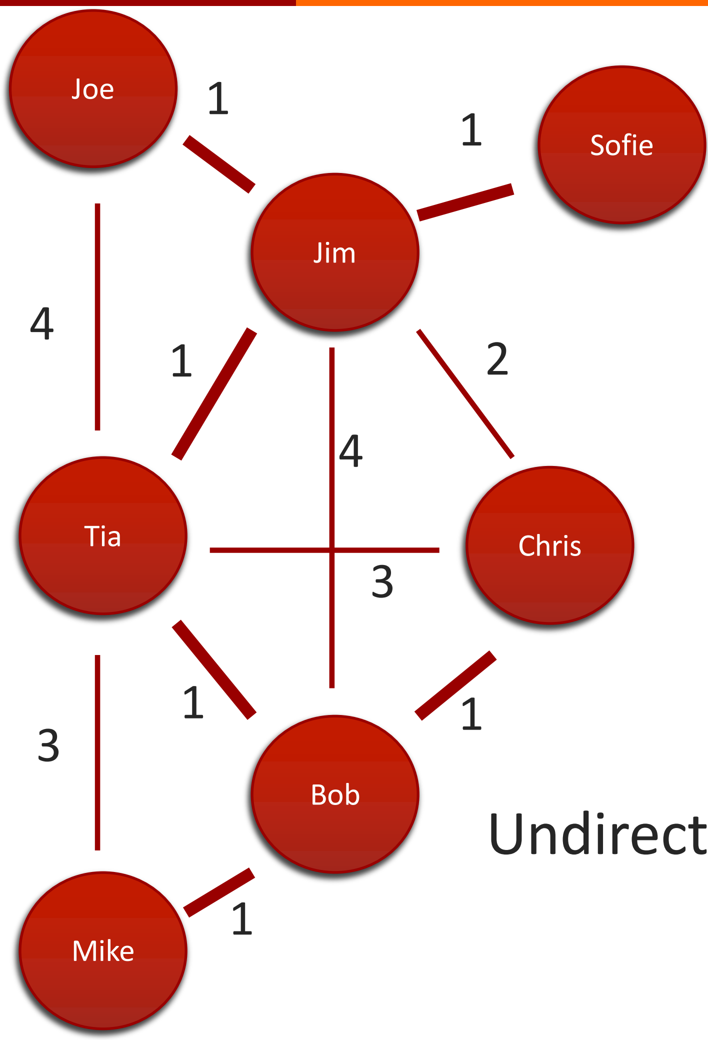
Kruskal's algorithm



Undirected

1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

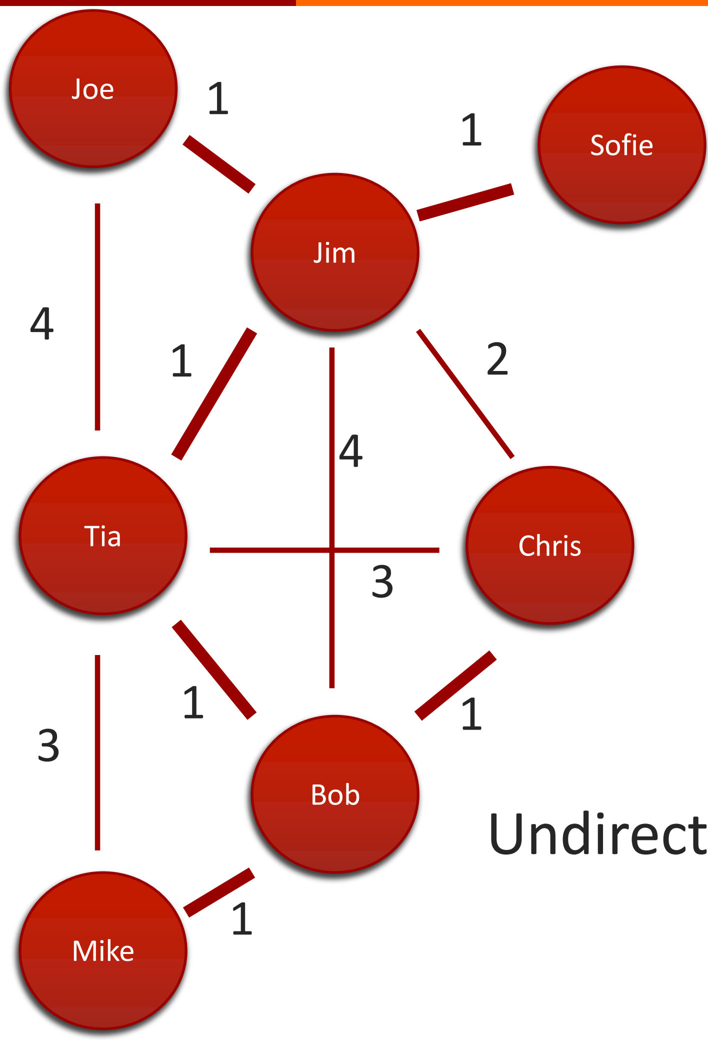
Kruskal's algorithm



Undirected

1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

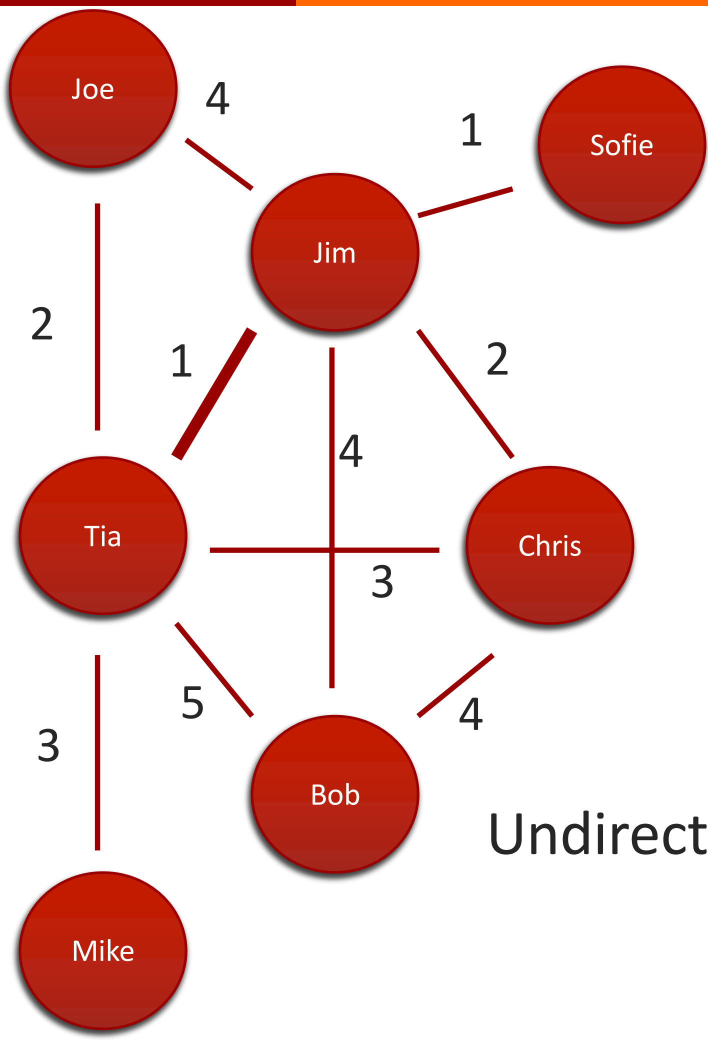
Kruskal’s algorithm



Undirected

1	**Joe-Jim**
1	**Jim-Sofie**
1	**Jim-Tia**
1	**Tia-Bob**
1	**Chris-Bob**
1	**Mike-Bob**
2	Chris-Jim
3	Tia-Chris
3	Mike-Tia
4	Joe-Tia
4	Jim-Bob

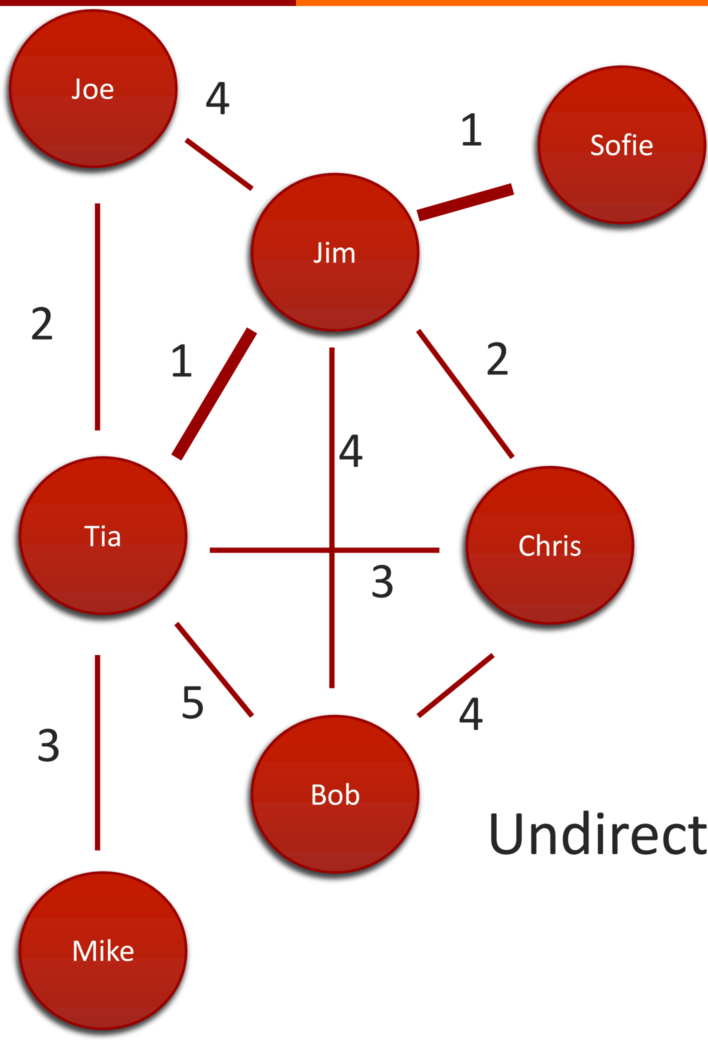
Kruskal's algorithm example #2



Undirected

1	Jim-Sofie
1	**Jim-Tia**
2	Chris-Jim
2	Joe-Tia
3	Tia-Chris
3	Mike-Tia
4	Jim-Bob
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

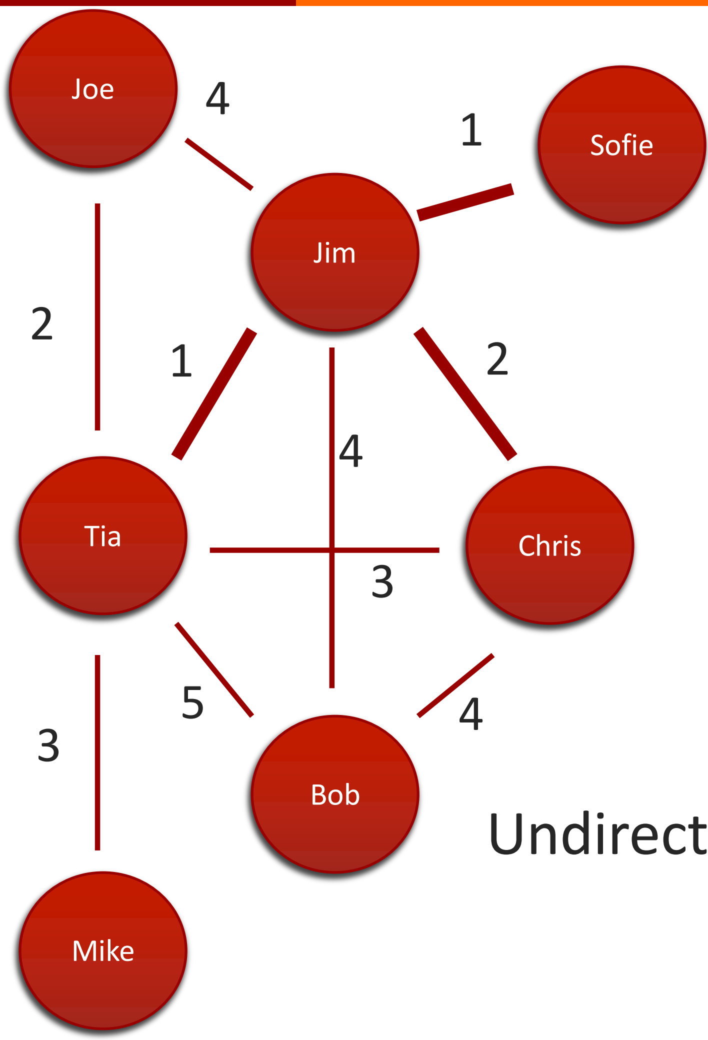
Kruskal's algorithm example #2



Undirected

1	**Jim-Sofie**
1	**Jim-Tia**
2	Chris-Jim
2	Joe-Tia
3	Tia-Chris
3	Mike-Tia
4	Jim-Bob
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

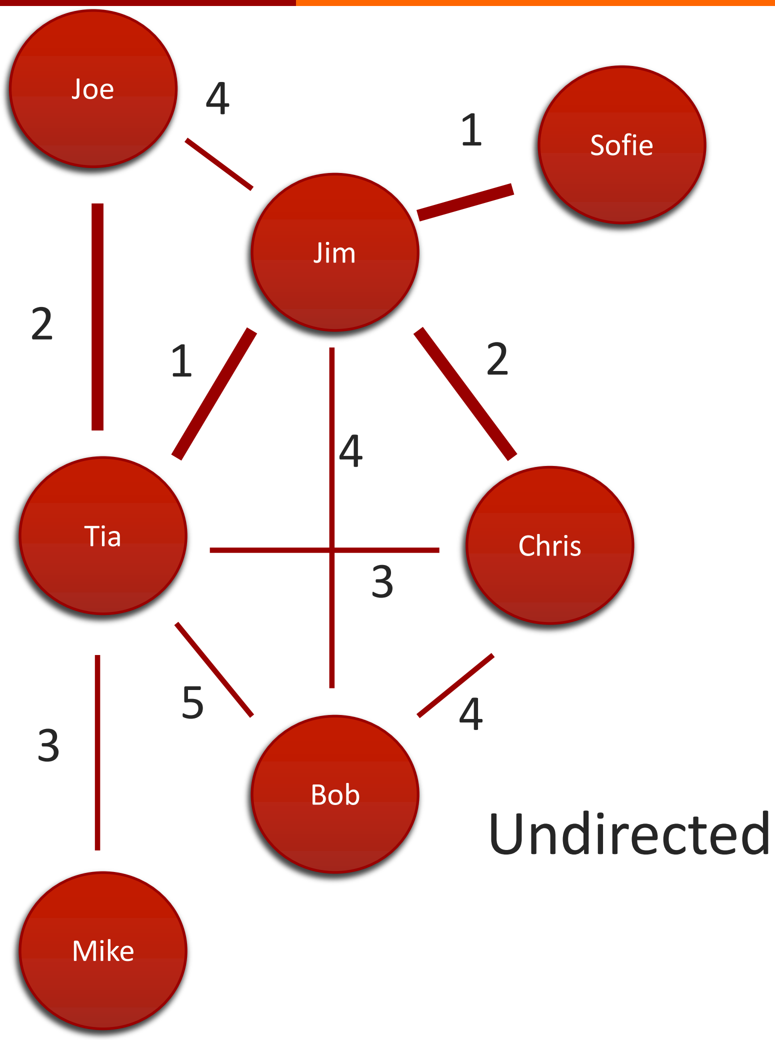
Kruskal's algorithm example #2



Undirected

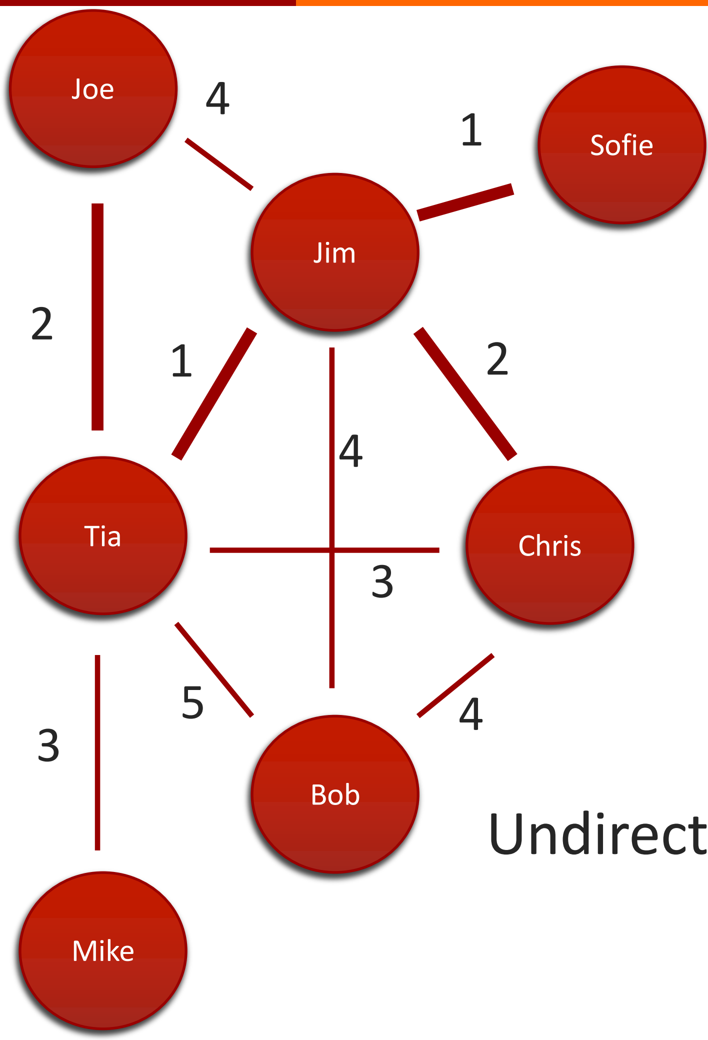
1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	Joe-Tia
3	Tia-Chris
3	Mike-Tia
4	Jim-Bob
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

Kruskal’s algorithm example #2



1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	Mike-Tia
4	Jim-Bob
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

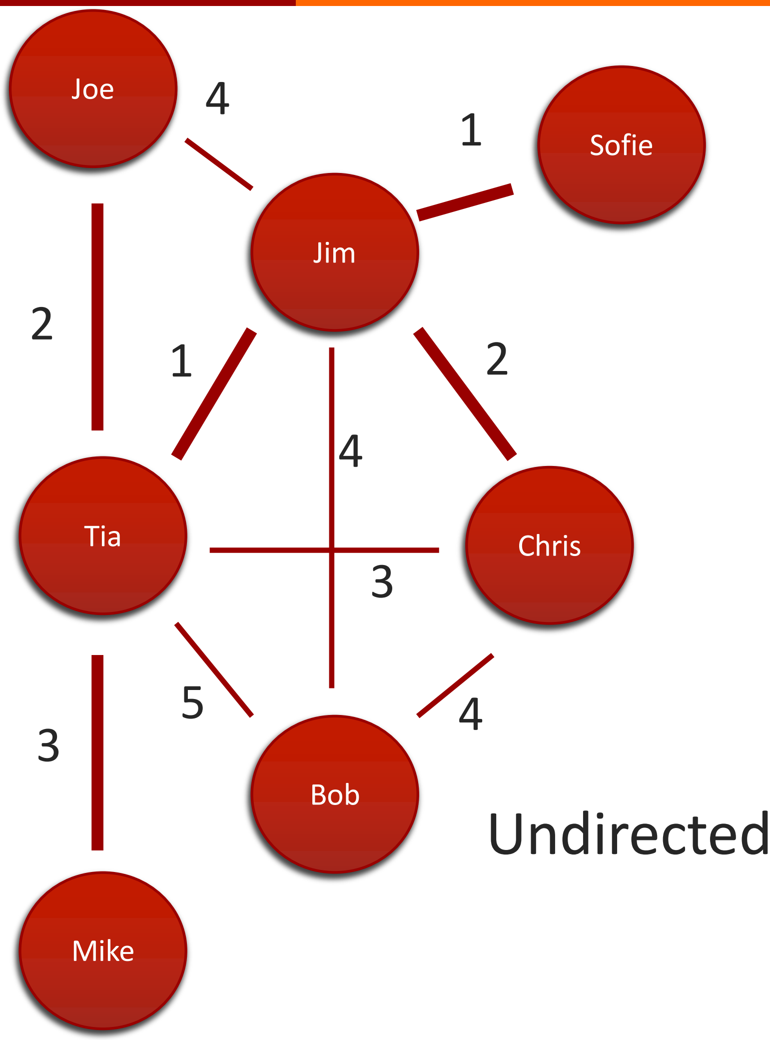
Kruskal’s algorithm example #2



Undirected

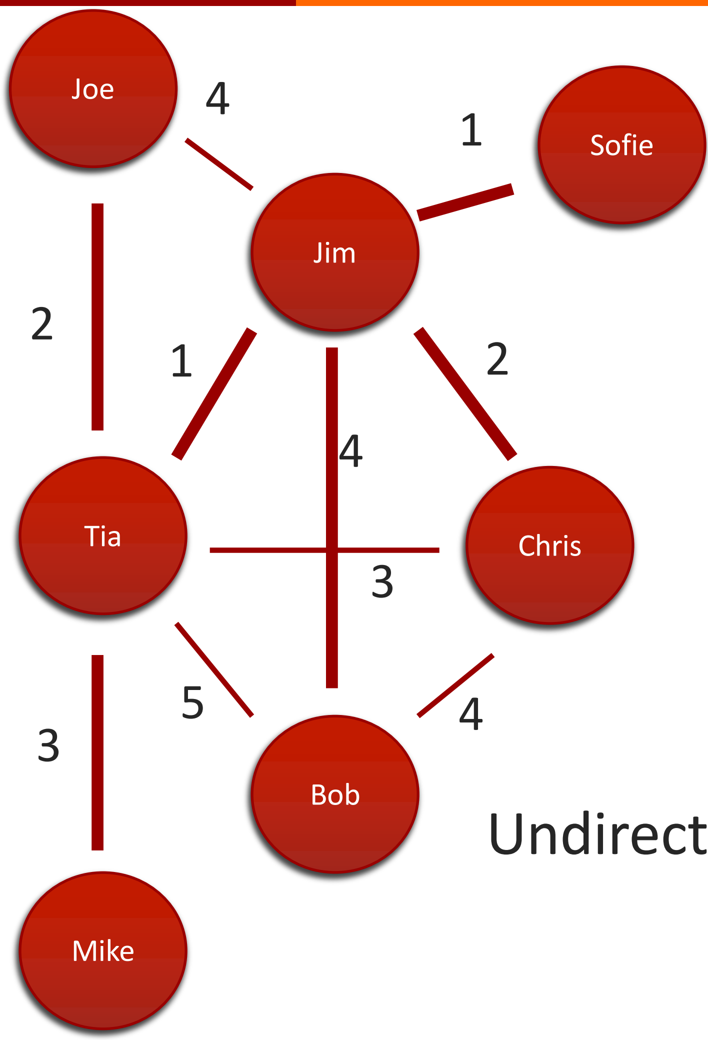
1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	Mike-Tia
4	Jim-Bob
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

Kruskal's algorithm example #2



1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	**Mike-Tia**
4	Jim-Bob
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

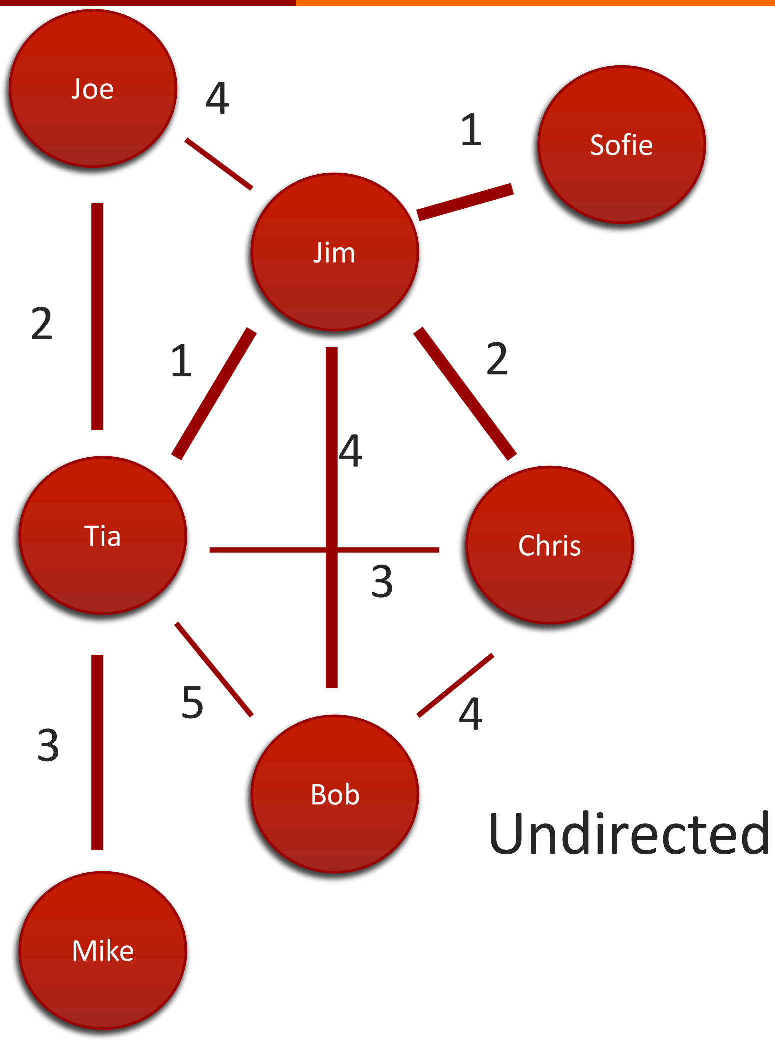
Kruskal’s algorithm example #2



Undirected

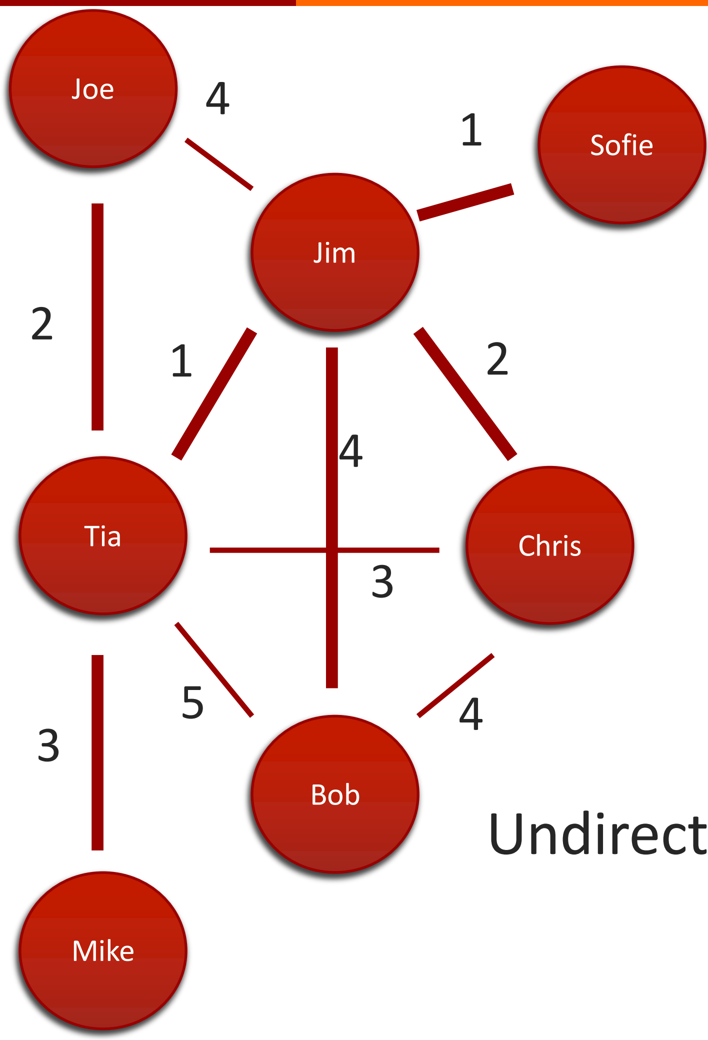
1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	**Mike-Tia**
4	**Jim-Bob**
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

Kruskal’s algorithm example #2



1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	**Mike-Tia**
4	**Jim-Bob**
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

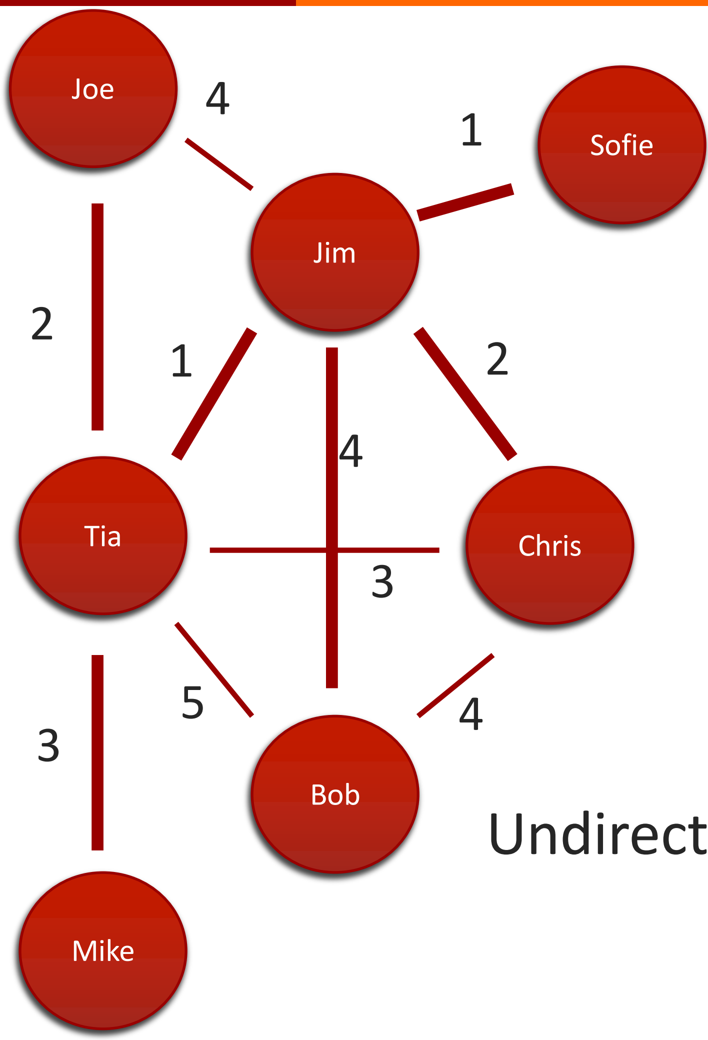
Kruskal's algorithm example #2



Undirected

1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	**Mike-Tia**
4	**Jim-Bob**
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

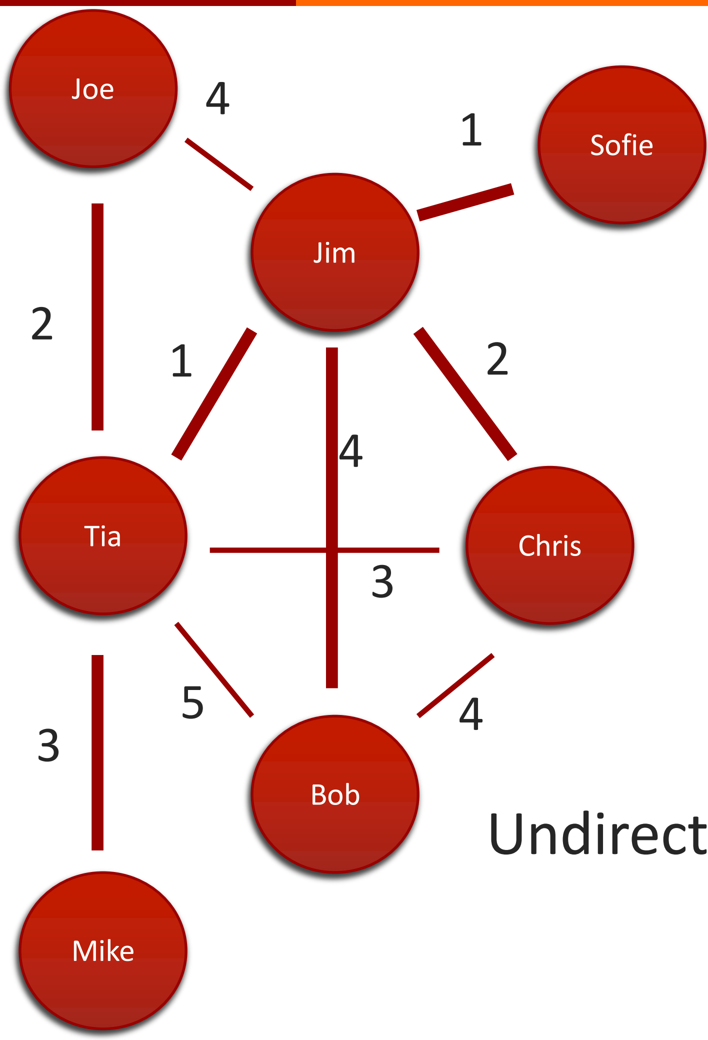
Kruskal's algorithm example #2



Undirected

1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	**Mike-Tia**
4	**Jim-Bob**
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

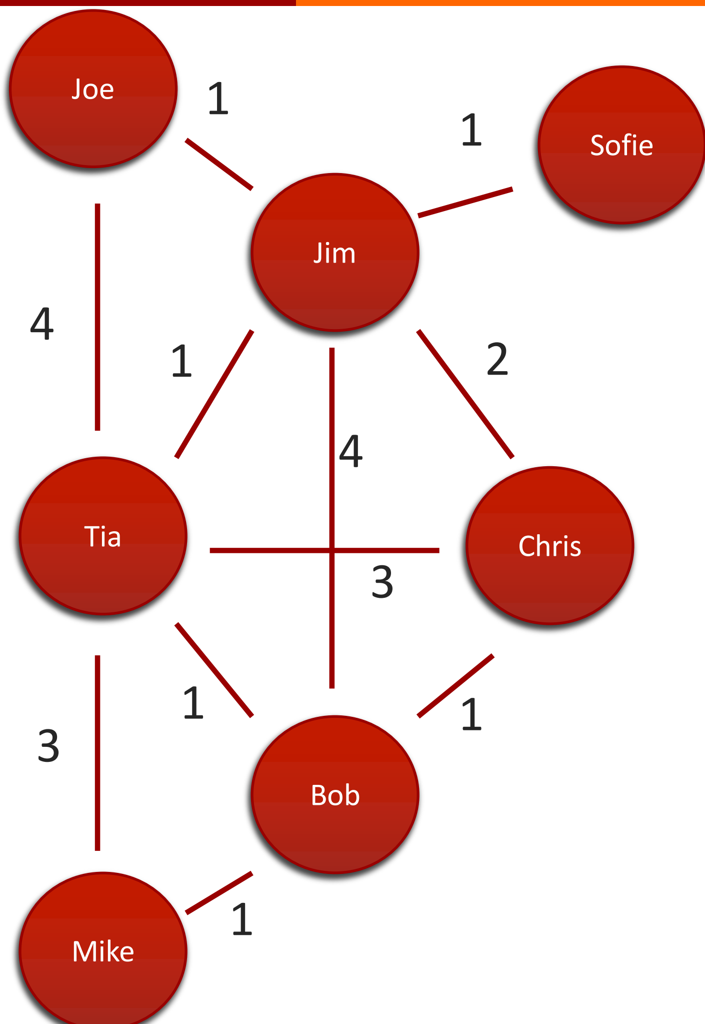
Kruskal's algorithm example #2



Undirected

1	**Jim-Sofie**
1	**Jim-Tia**
2	**Chris-Jim**
2	**Joe-Tia**
3	Tia-Chris
3	**Mike-Tia**
4	**Jim-Bob**
4	Joe-Jim
4	Chris-Bob
5	Tia-Bob

Shortest path

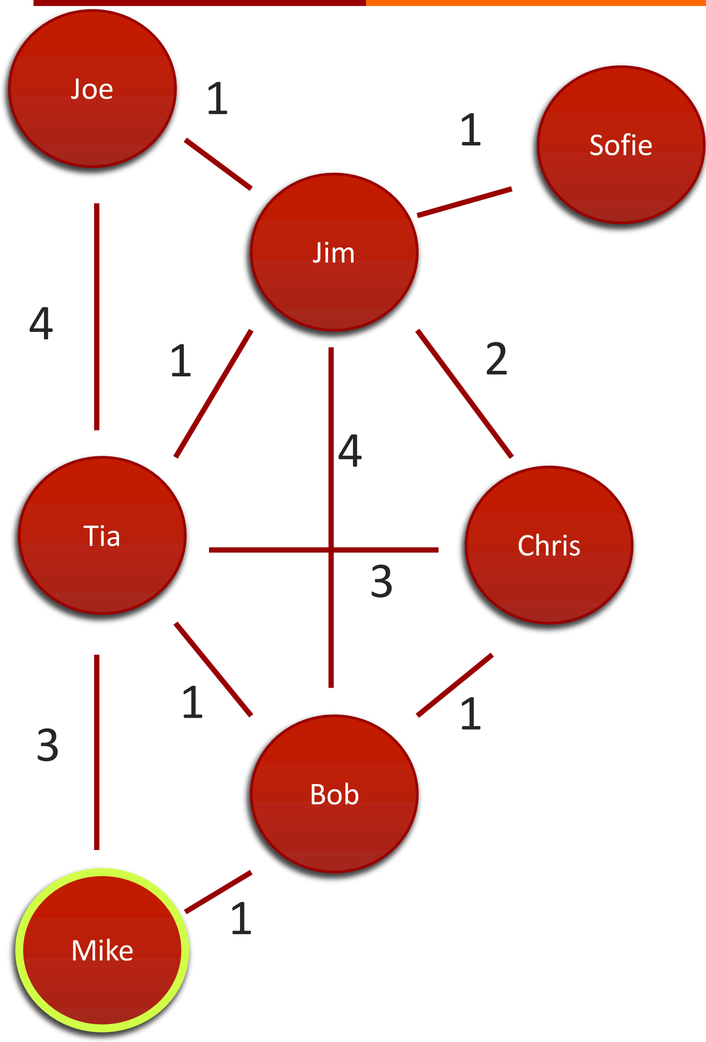


- For a given source vertex (node) in the graph, it finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex.
- Say your source vertex is Mike
  - Lowest cost path from Mike to Jim is Mike – Bob - Tia – Jim (cost 3)
  - Lowest cost path from Mike to Joe is Mike – Bob – Tia – Jim – Joe (cost 4)
- Very important for networking applications!

Dijkstra’s algorithm: Basic idea

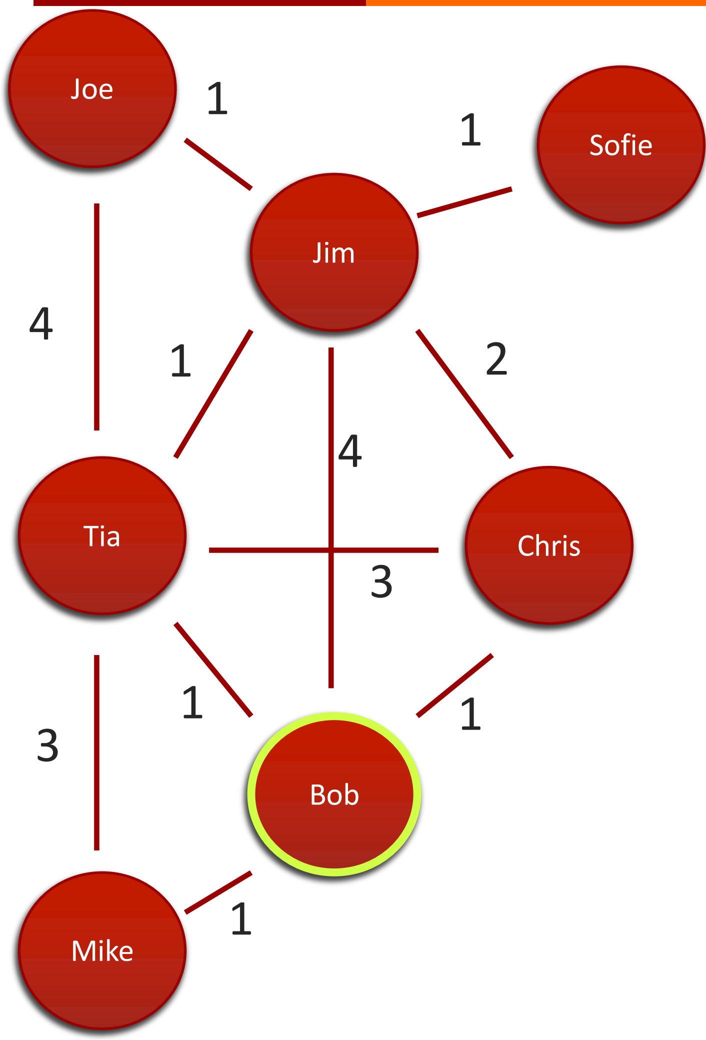
- Fan out from the initial node
- In the beginning the distances to the neighbors of the initial node are known. All other nodes are tentatively infinite distance away.
- The algorithm improves the estimates to the other nodes step by step.
- As you fan out, perform the operation illustrated in this example: if the current node *A* is marked with a distance of 4, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 4 + 2 = 6. If B was previously marked with a distance greater than 6 then change it to 6. Otherwise, keep the current value.

Shortest path from Mike



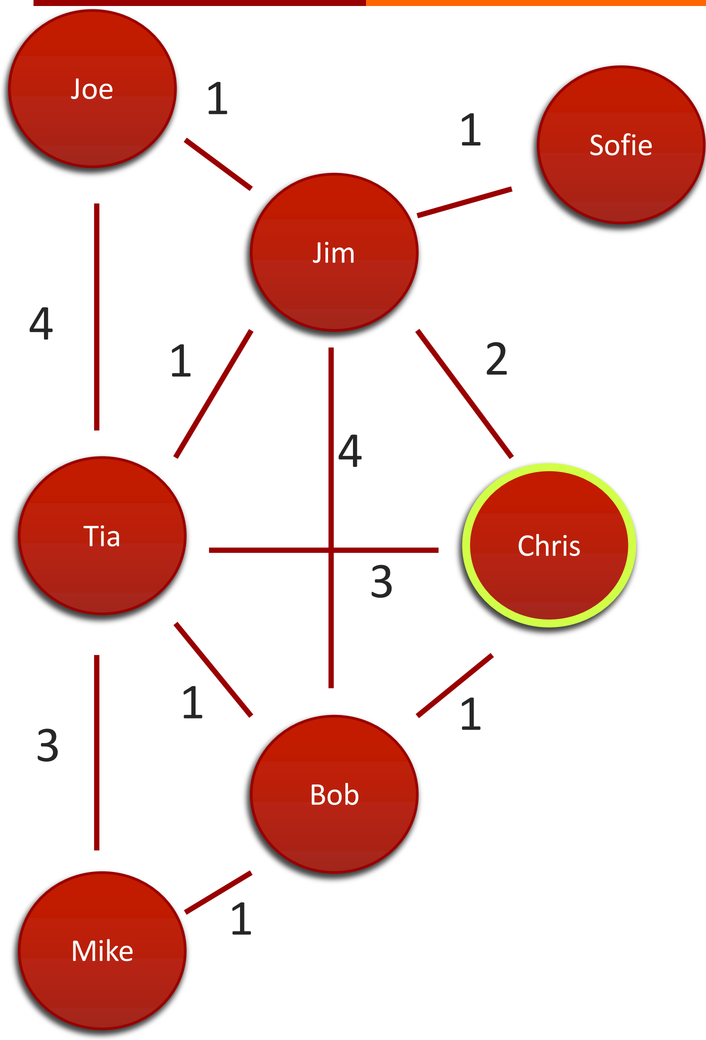
Mike	Bob	Tia	Jim	Chris	Sofie	Joe
0	1	3	∞	∞	∞	∞

Shortest path from Mike



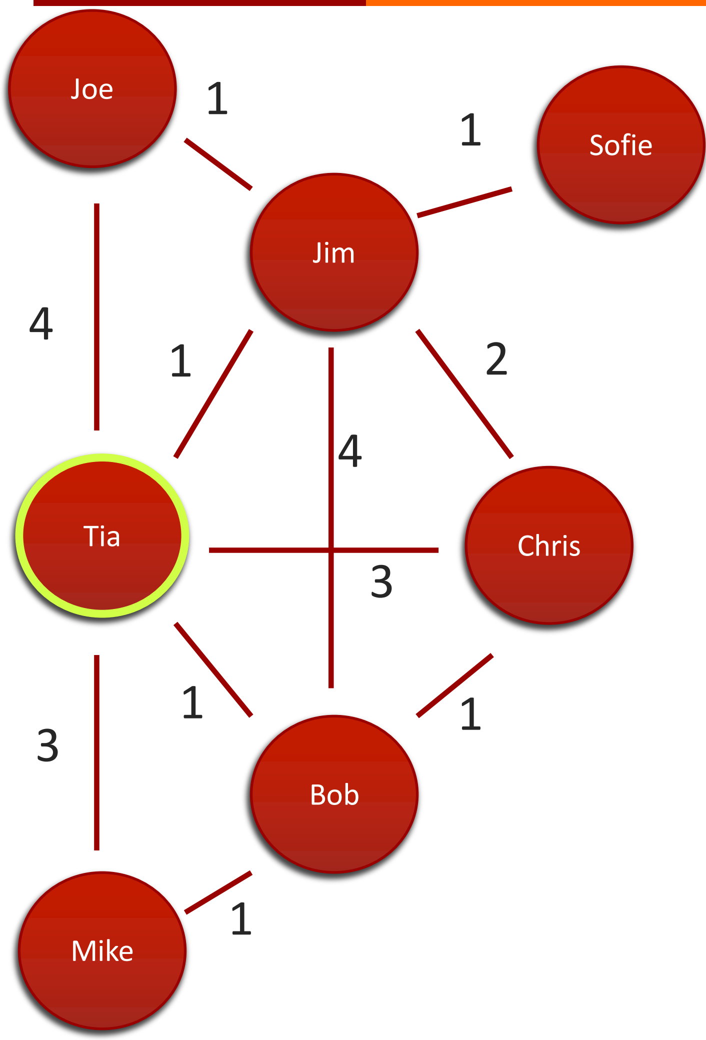
Mike	Bob	Tia	Jim	Chris	Sofie	Joe
0	1	3	∞	∞	∞	∞
0	1	2	5	2	∞	∞

Shortest path from Mike



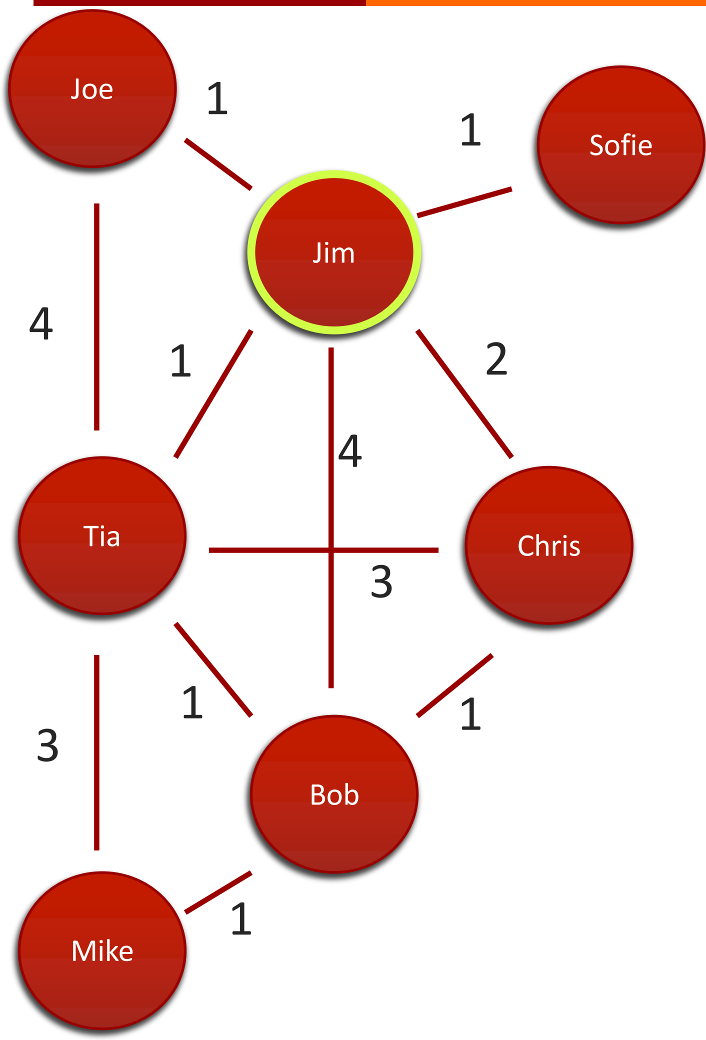
Mike	Bob	Tia	Jim	Chris	Sofie	Joe
0	1	3	∞	∞	∞	∞
0	1	2	5	2	∞	∞
0	1	2	4	2	∞	∞

Shortest path from Mike



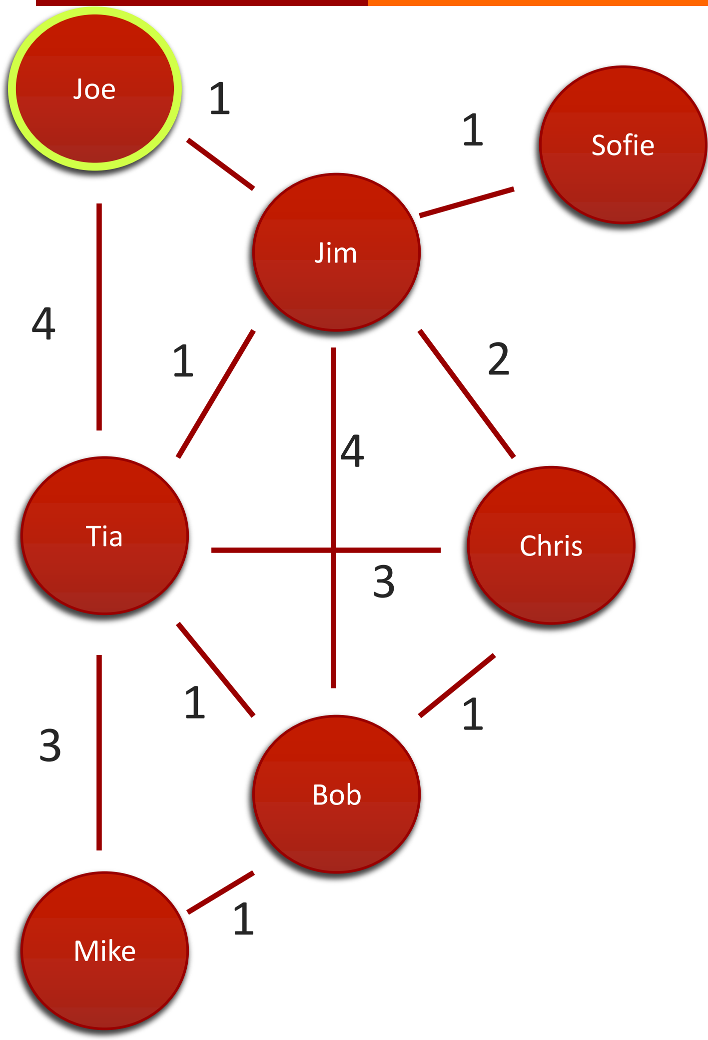
Mike	Bob	Tia	Jim	Chris	Sofie	Joe
0	1	3	∞	∞	∞	∞
0	1	2	5	2	∞	∞
0	1	2	4	2	∞	∞
0	1	2	3	2	∞	6

Shortest path from Mike



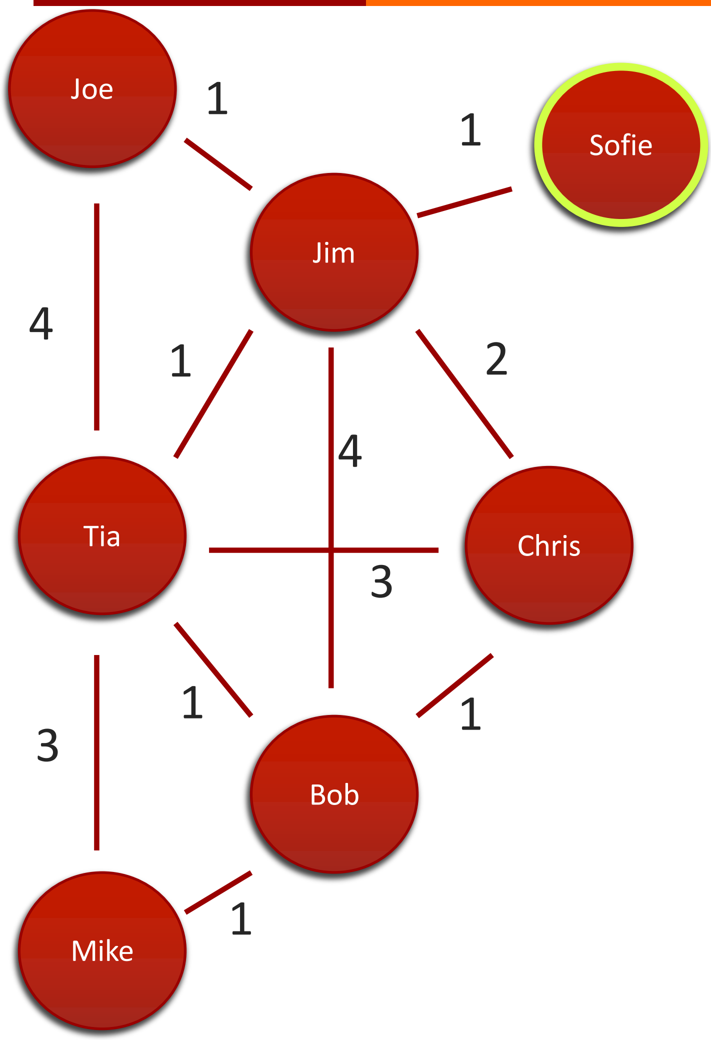
Mike	Bob	Tia	Jim	Chris	Sofie	Joe
0	1	3	∞	∞	∞	∞
0	1	2	5	2	∞	∞
0	1	2	4	2	∞	∞
0	1	2	3	2	∞	6
0	1	2	3	2	4	4

Shortest path from Mike



Mike	Bob	Tia	Jim	Chris	Sofie	Joe
0	1	3	∞	∞	∞	∞
0	1	2	5	2	∞	∞
0	1	2	4	2	∞	∞
0	1	2	3	2	∞	6
0	1	2	3	2	4	4
0	1	2	3	2	4	4

Shortest path from Mike



Mike	Bob	Tia	Jim	Chris	Sofie	Joe
0	1	3	∞	∞	∞	∞
0	1	2	5	2	∞	∞
0	1	2	4	2	∞	∞
0	1	2	3	2	∞	6
0	1	2	3	2	4	4
0	1	2	3	2	4	4
0	1	2	3	2	4	4