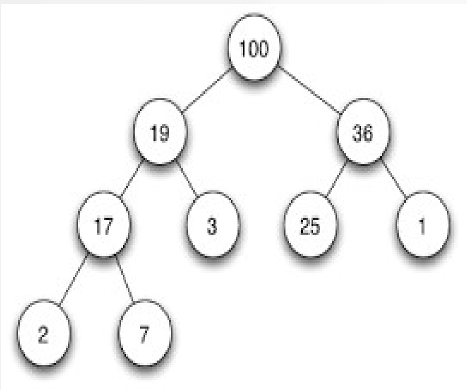
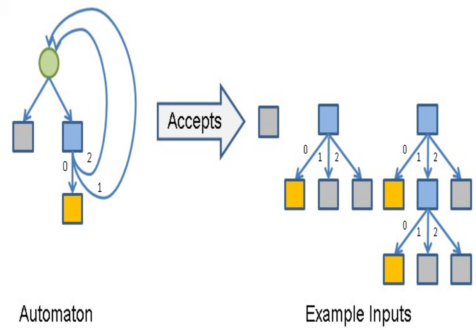
Trees

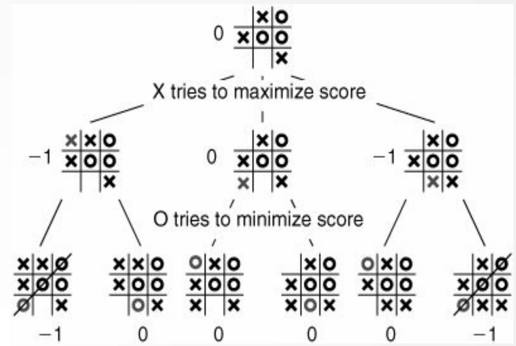
	Miles Jones	MTThF 8:30-9:50am	CSE 4140

August 18, 2016

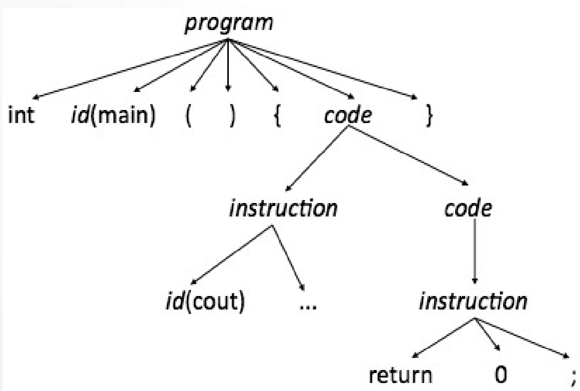
Another Special Type of Graph: Trees

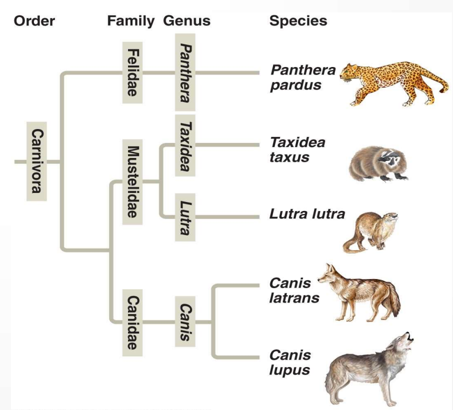






things outside





Trees

1. Definitions of trees
2. Properties of trees
3. Revisiting uses of trees

rooted tree (directed graph)

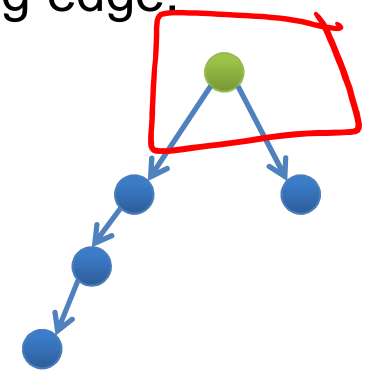
un-rooted tree (undirected graph).

(Rooted) Trees: definitions

D A G

Rosen p. 747-749

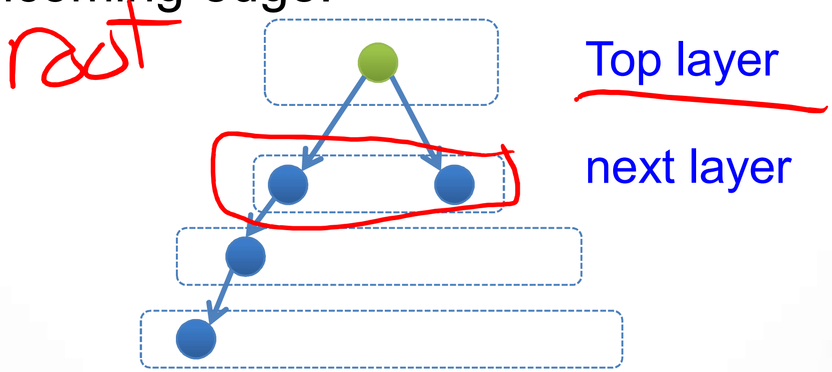
A **rooted tree** is a connected directed acyclic graph in which one vertex has been designated the root, which has no incoming edges, and every other vertex has exactly one incoming edge.



(Rooted) Trees: definitions

Rosen p. 747-749

A **rooted tree** is a connected directed acyclic graph in which one vertex has been designated the root, which has no incoming edges, and every other vertex has exactly one incoming edge.

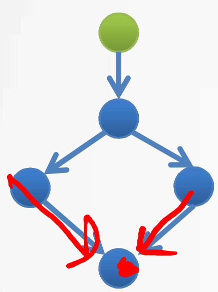


Special case of DAGs from last class.
Note that each vertex in middle has *exactly one* incoming edge from layer above.
Edges are directed *away from* the root.

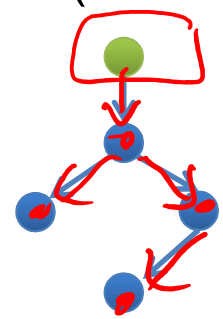
Trees?

Which of the following directed graphs are trees (with root indicated in green)?

A.

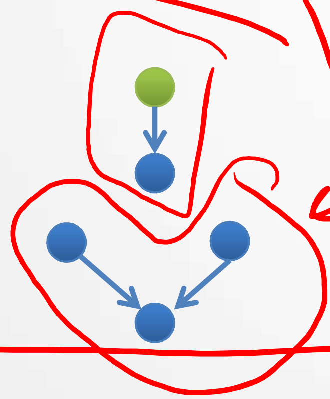


C.



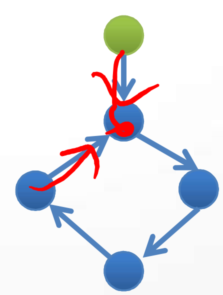
root

B.



not a rooted tree.

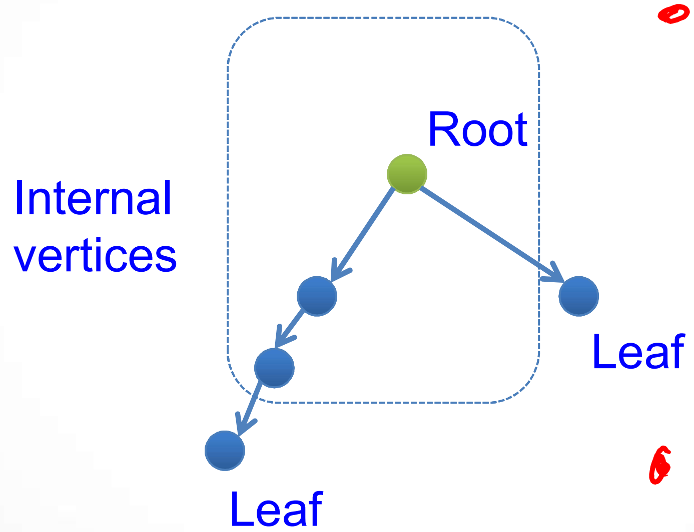
D.



E. wrong answer

(Rooted) Trees: definitions

Rosen p. 747-749

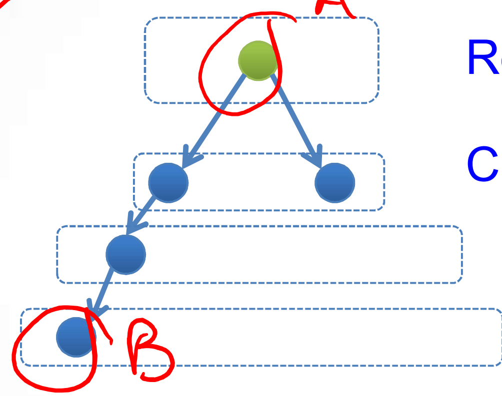


- internal vertex
  a vertex that has an out-going edge
- Leaf: not an internal vertex
  a vertex with no out-edges.

(Rooted) Trees: definitions

height 0

Rosen p. 747-749, 753



Root — Height 0

Children of root — Height 1

A is an ancestor of B

B is a descendant of A

If vertex v is not the root, it has exactly one incoming edge, which is from its parent, p(v).

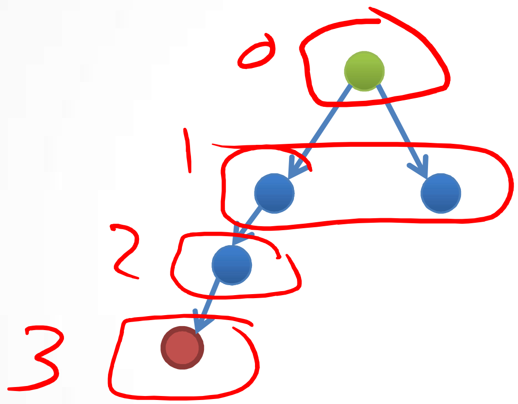
p(root) = root

Height of vertex v is given by the recurrence:

h(v) = h(p(v)) + 1 *if v is not the root,and* h(r) = 0

(Rooted) Trees: definitions

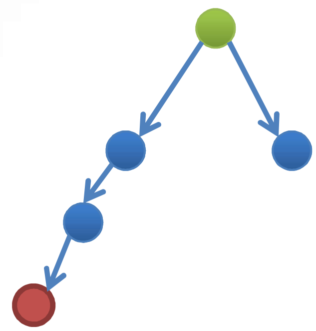
Height of vertex v: h(v) = h(p(v)) + 1 *if v is not the root,and* h(r) = 0



What is the height of the red vertex?
A. 0
B. 1
C. 2
D. 3
E. None of the above.

(Rooted) Trees: definitions

Height of vertex v: h(v) = h(p(v)) + 1 *if v is not the root,and* h(r) = 0



Height of tree is maximum height of a vertex in the tree.

Rosen p. 753

What is the height of the **tree**?

A. 0
B. 1
C. 2
D. 3
E. None of the above.

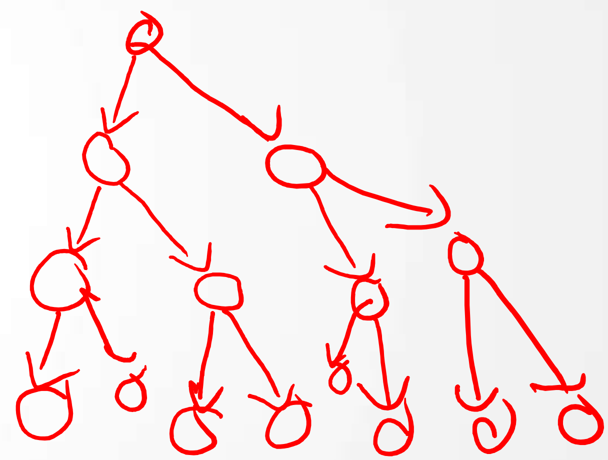
Binary tree

Rosen p. 749, 754

A **binary tree** is a rooted tree where every (internal) vertex has no more than 2 children.

How many leaves does a binary tree of height 3 have?

A. 2
B. 3
C. 6
D. 8
E. any of the above.



Binary tree

Rosen p. 749, 754

A **binary tree** is a rooted tree where every (internal) vertex has no more than 2 children.

How many leaves does a binary tree of height 3 have?

A. 2
B. 3
C. 6
D. 8
E. any of the above.

See Theorem 5 for proof of upper bound

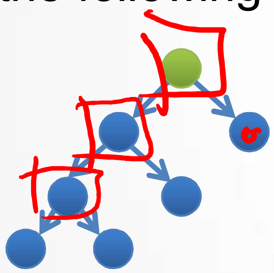
Binary tree

Rosen p. 749

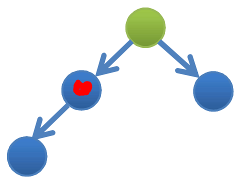
A **full** binary tree is a rooted tree where every internal vertex has exactly 2 children.

Which of the following are full binary trees?

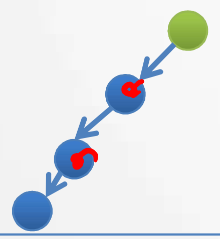
A.



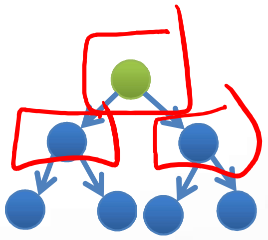
C.



B.



D.



Binary tree

Rosen p. 749

A **full** binary tree is a rooted tree where every internal vertex has exactly 2 children.

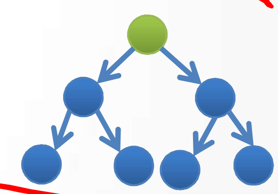
At most how many vertices are there in a full binary tree of height h?

A. $\Theta(h)$

C. $\Theta(h^2)$

B. $\Theta(2^h)$

D. $\Theta(\log h)$



Max number of vertices when tree is balanced

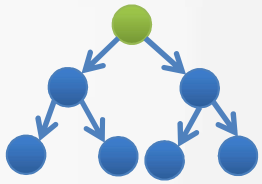
Binary tree

Rosen p. 749

A **full** binary tree is a rooted tree where every internal vertex has exactly 2 children.

Key insight: number of vertices doubles on each level.

$1 + 2 + 4 + 8 + \ldots + 2^h$
$= 2^{h+1}-1$ i.e. $\Theta(2^h)$



Max number of vertices when tree is balanced

If n is number of vertices:

$n = 2^{h+1}-1$

so

$h = \log(n+1) - 1$ i.e. $\Theta(\log n)$

Binary tree

Rosen p. 749

Relating height and number of vertices:

$\boxed{\log(n+1) - 1 \;<=\; h} \;<=\; \_\_\_$

This is what we just proved.

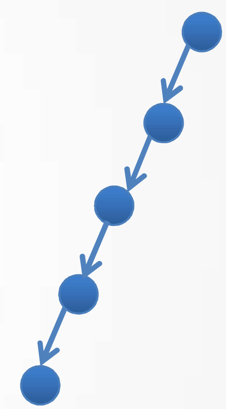
How do we prove?

What tree with n vertices has the greatest possible height?

Binary tree

Rosen p. 749

Relating height and number of vertices:



$$\log(n+1) - 1 <= h <= n-1$$

This is what we just proved.

How do we prove?

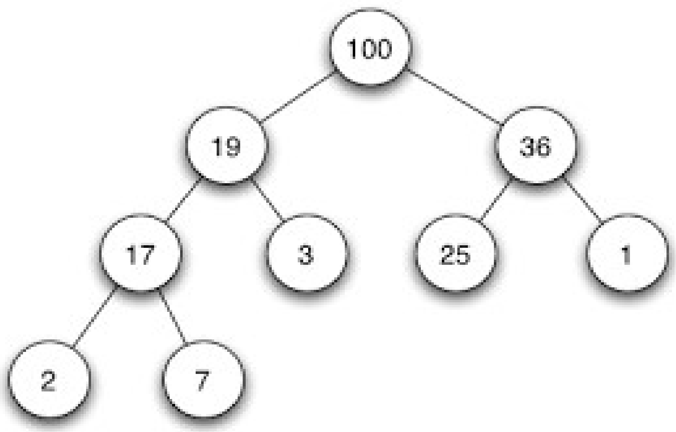
What tree with n vertices has the greatest possible height?

Trees

1. Definitions of trees ✓

2. Properties of trees ✓

3. Revisiting uses of trees



In data structures:
Max heap

Binary Search Trees

- Facilitate binary search (must maintain sorted order of data)
- Dynamic

Implementation

Each vertex is an object with the fields

p = parent
lc = left child
rc = right child
value

When is **p null**?

A. If we have an error in our implementation.
B. When the value is 0.
C. When the vertex is a leaf node.
D. When the vertex is the root node.
E. None of the above.

Binary Search Trees

- Facilitate binary search (must maintain sorted order of data)
- Dynamic

Implementation

Each vertex is an object with the fields

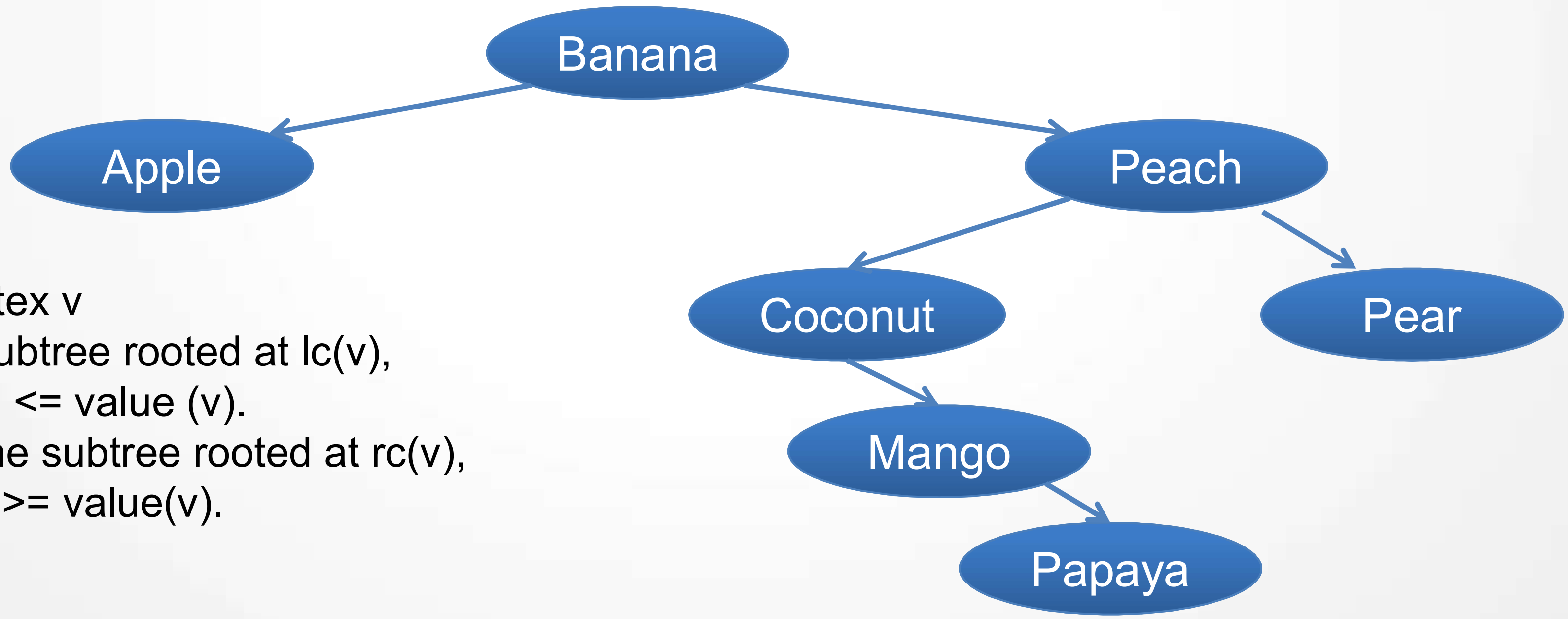
p = parent
lc = left child
rc = right child
value

Under which of these conditions is **lc always null**?

A. If we have an error in our implementation.
B. When the value is 0.
C. When the vertex is a leaf node.
D. When the vertex is the root node.
E. None of the above.

Binary Search Trees

- Facilitate binary search (must **maintain sorted order** of data)
- Dynamic

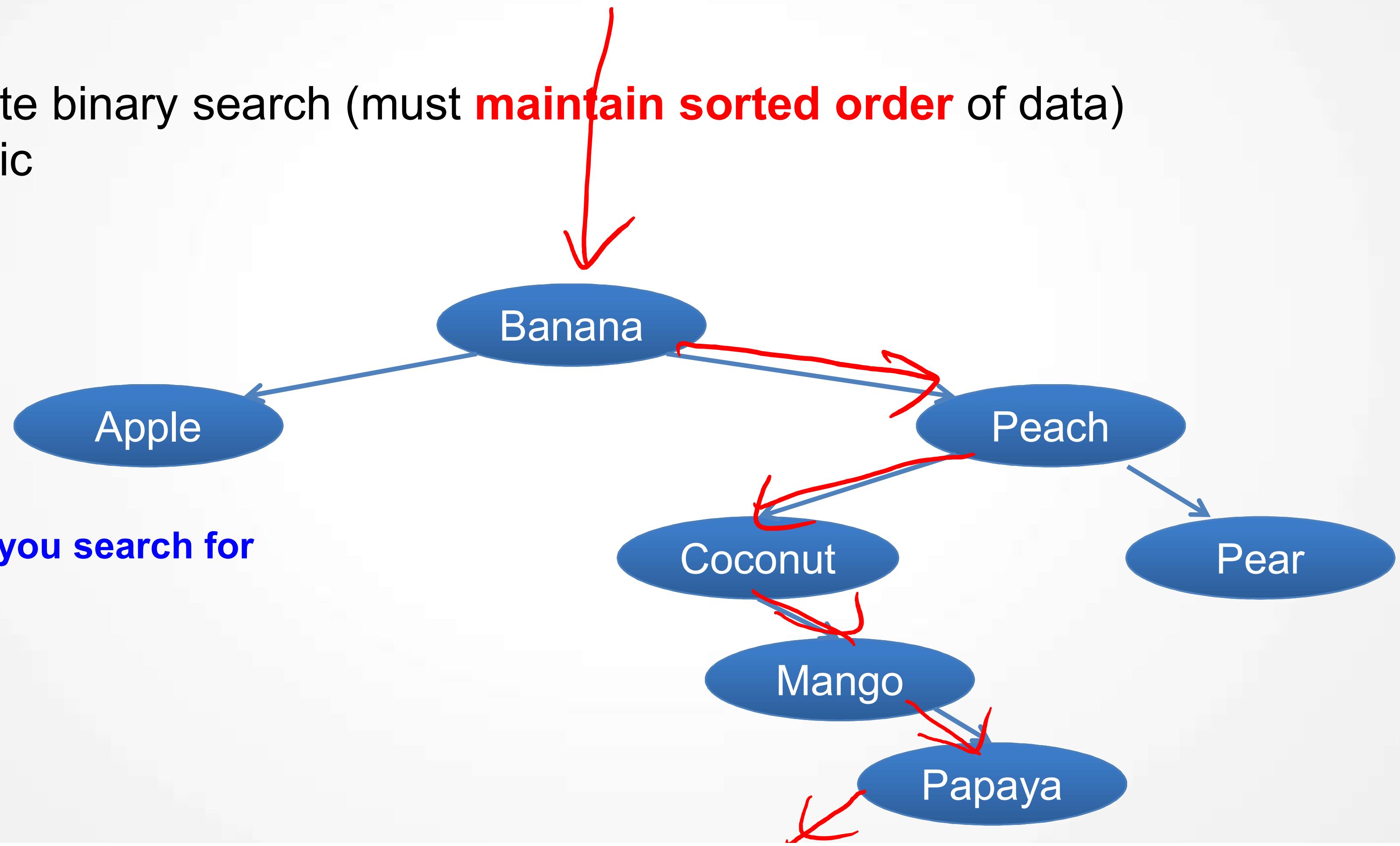


For each vertex v

- If x is in subtree rooted at lc(v), value(x) <= value (v).
- If x is in the subtree rooted at rc(v), value(x)>= value(v).

Binary Search Trees

- Facilitate binary search (must **maintain sorted order** of data)
- Dynamic



How would you search for "orange?

Binary Search Trees

- Facilitate binary search (must **maintain sorted order** of data)
- Dynamic

To search for target T in a binary search tree.

1. Compare T to value(r) where r is the root.
2. If T = value(r), done ☺.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).

Binary Search Trees

- Facilitate binary search (must **maintain sorted order** of data)
- Dynamic

To search for target T in a binary search tree.

1. Compare T to value(r) where r is the root.
2. If T = value(r), done ☺.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).

How long does this take?

Binary Search Trees

- Facilitate binary search (must **maintain sorted order** of data)
- Dynamic

To search for target T in a binary search tree.

1. Compare T to value(r) where r is the root.
2. If T = value(r), done ☺.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).

How long does this take?

Constant time at each level, number of levels is height+1.

Binary Search Trees

- Facilitate binary search (must **maintain sorted order** of data)
- Dynamic

To search for target T in a binary search tree.

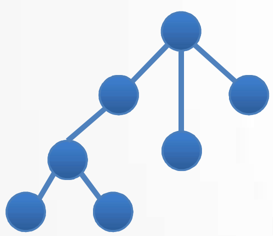
1. Compare T to value(r) where r is the root.
2. If T = value(r), done ☺.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).

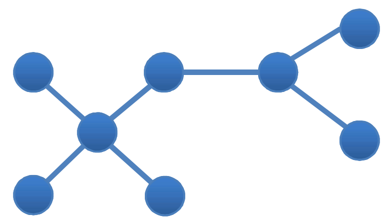
How long does this take? Time proportional to height!

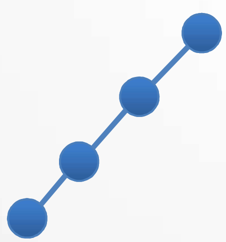
Unrooted trees

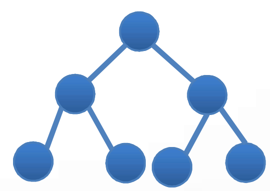
Rosen p. 746

An **unrooted tree** is a connected undirected graph with no cycles.









Equivalence between rooted and unrooted trees

Theorem: An undirected graph is an unrooted tree if and only if it contains all the edges of some rooted tree.

What does this mean?

(1) If we replace all directed edges in a rooted tree with undirected edges, the result will be an unrooted tree.

(2) There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

Equivalence between rooted and unrooted trees

Goal (1): If we replace all directed edges in a rooted tree with undirected edges, the result will be an **unrooted tree**.

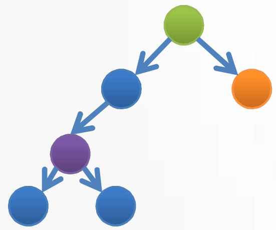
What do we need to prove?

A. The resulting undirected graph will be connected.
B. The resulting undirected graph will be undirected.
C. The resulting undirected graph will not have cycles.
D. All of the above.

Equivalence between rooted and unrooted trees

Goal (1): If we replace all directed edges in a rooted tree with undirected edges, the result will be an **unrooted tree**.

SubGoal (1a): this resulting graph is connected, i.e. between any two vertices u and v there is a path in the graph.



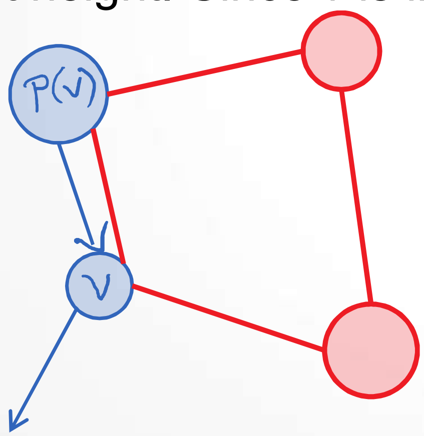
Idea: To find path between purple and orange, follow parents of purple all the way to root, then follow its children down to orange.

Equivalence between rooted and unrooted trees

Goal (1): If we replace all directed edges in a rooted tree with undirected edges, the result will be an **unrooted tree**.

SubGoal (1b): this resulting graph has no cycles.

Assume by contradiction that there exists a cycle and let v be the vertex with the deepest height. Since v is in a cycle, there are two edges in the cycle incident with v.



There is one edge from the parent. All other edges go to deeper vertices contradicting that v is the deepest.

Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

Idea: finding right directions for edges will be similar to finding topological sort last class.

Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

SubGoal (2a): Any unrooted tree with at least two vertices has a vertex of degree exactly 1. dependent on the graph being finite

Proof: Towards a contradiction, assume that all vertices have degree 0 or >=2.
Since a tree is connected, eliminate the case of degree-0 vertices.
Goal: construct a cycle to arrive at a contradiction.

Start at any vertex u_0.
Pick u_{i+1} so that it is adjacent to u_i but is **not** u_{i-1}. Why?

Get $u_0, u_1, \ldots, u_n$. By Pigeonhole Principle, must repeat. Cycle!

Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

SubGoal (2b): If T is unrooted tree and v has degree 1 in T, then T-{v} is unrooted tree.

Proof: To check that T-{v} is unrooted tree,

* confirm T-{v} is connected and

* T-{v} does not have a cycle.

Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

SubGoal (2b): If T is unrooted tree and v has degree 1 in T, then T-{v} is unrooted tree.

Proof: To check that T-{v} is unrooted tree,

* confirm T-{v} is connected and

* T-{v} does not have a cycle.

Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

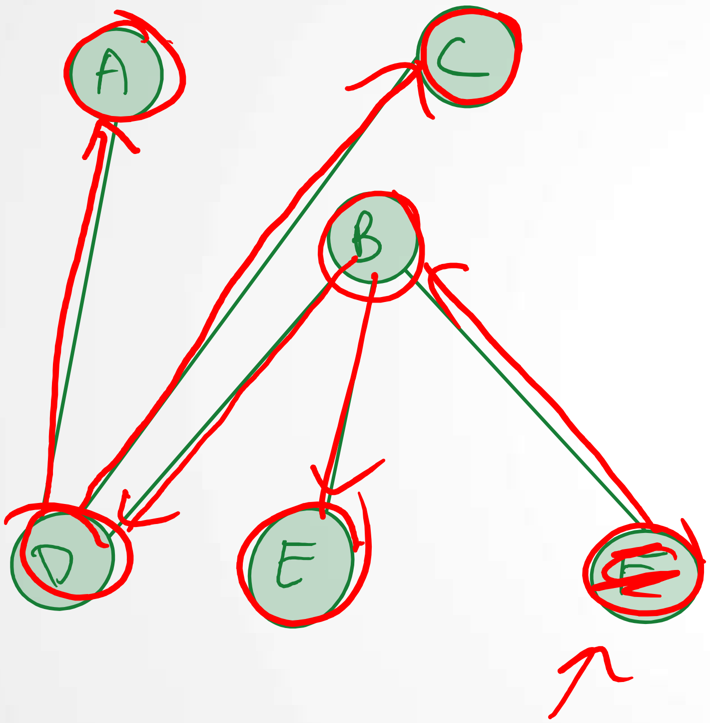
Using the subgoals to achieve the goal:

Root(T: unrooted tree with n nodes)

1. If $n=1$, let the only vertex v be the root, set $h(v):=0$, and return.
2. Find a vertex v of degree 1 in T, and let u be its only neighbor.
3. Root(T-$\{v\}$).
4. Set $p(v):= u$ and $h(v):=h(u)+1$.

Recursion!

Example



vertex	A	B	C	D	E	F
degree	1	3	1	3	1	1
	0	3	1	2	1	1
	0	3	0	1	1	1
	0	2	0	0	1	1
	0	1	0	0	0	1
	0	0	0	0	0	0

$h(A) = h(D) + 1$

$h(C) = h(D) + 1$

$h(D) = h(B) + 1$

$h(E) = h(B) + 1$

$h(B) = h(F) + 1$

$h(F) = 0$

$h(B) = 1$

$h(E) = 2$

$h(D) = 2$

$h(C) = 3$

$h(A) = 3$

Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

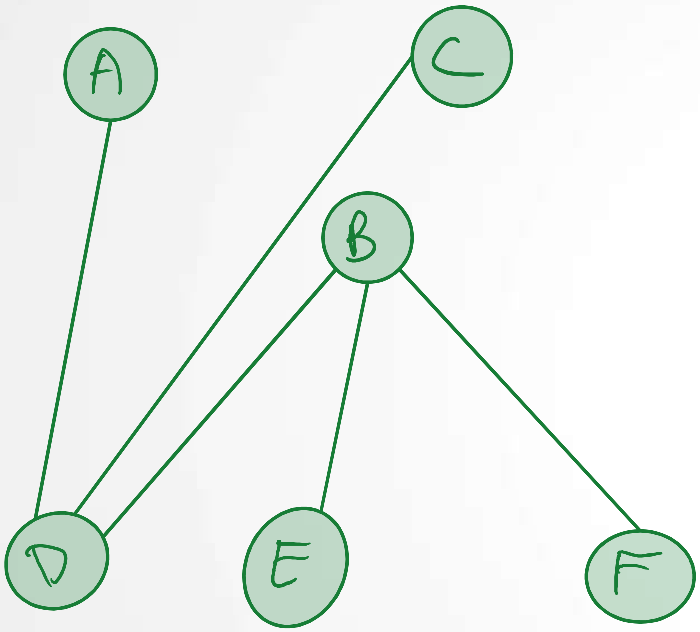
Using the subgoals to achieve the goal:

Root(T: unrooted tree with n nodes)

1. If $n=1$, let the only vertex v be the root, set $h(v):=0$, and return.
2. Find a vertex v of degree 1 in T, and let u be its only neighbor.
3. Root(T-$\{v\}$).
4. Set $p(v):= u$ and $h(v):=h(u)+1$.

Recursion!

Example



vertex	A	B	C	D	E	F
degree	1	3	1	3	1	1

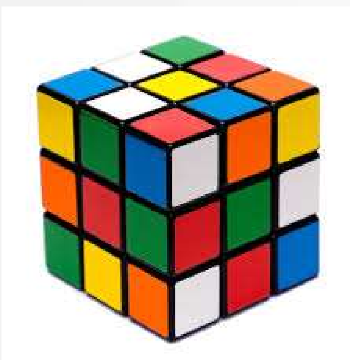
Counting

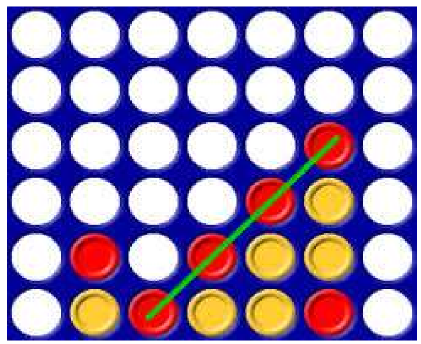
1, 2, 3, 4, …

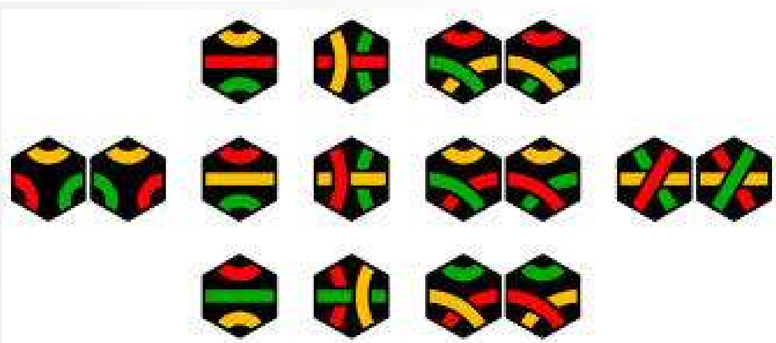
What do we mean by counting?

How many arrangements or combinations of objects are there of a given form?

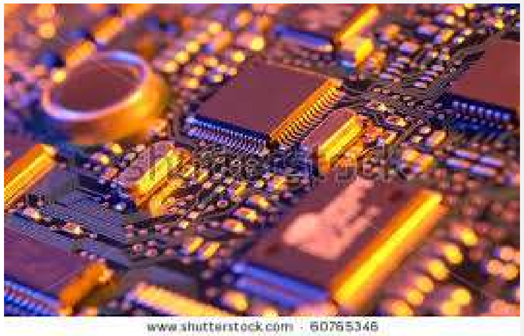
How many of these have a certain property?







Why is counting important?



For computer scientists:

- **Hardware**: How many ways are there to arrange components on a chip?
- **Algorithms**: How long is this loop going to take? How many times does it run?
- **Security**: How many passwords are there?
- **Memory**: How many bits of memory should be allocated to store an object?

Miis

In some video games, each player can create a character with custom facial features.

How many distinct characters are possible?

Miis

In some video games, each player can create a character with custom facial features. How many distinct characters are possible?

Considering only these 12 hairstyles and 8 hair colors, how many different characters are possible?

A. $8+12 = 20$
B. $8*12 = 96$
C. $8^{12} = 68719476736$
D. $12^{8} = 429981696$
E. None of the above

Product rule

Rosen p. 386

For any sets, A and B: |A x B| = |A| |B|

In our example:

A = { hair styles } |A| = 12
B = { hair colors } |B| = 8

A x B = { (s, c) : s is a hair style and c is a hair color }

|A x B| = the number of possible pairs of hair styles & hair colors
= the number of different ways to specify a character

Product rule

Rosen p. 386

For any sets, A and B: |A x B| = |A| |B|

More generally:

Suppose that a procedure can be broken down into a sequence of two tasks. If there are n_1 ways to do the first task and for each of these ways of doing the first task, there are n_2 ways to do the second task, then there are $\mathbf{n_1 n_2}$ ways to do the procedure.

Product rule

Rosen p. 386

For any sets, A and B: $|A \times B| = |A|\ |B|$

More generally:

To count the number of pairs of objects:
* Count the number of choices for selecting the first object.
* Count the number of choices for selecting the second object.
* Multiply these two counts.

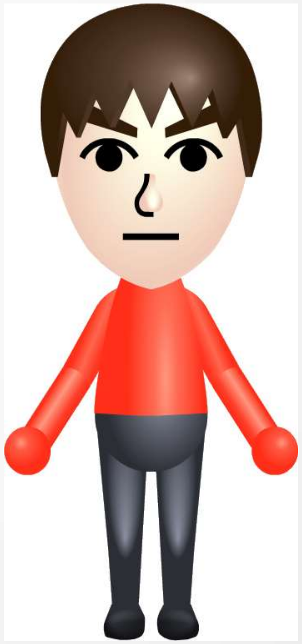
CAUTION: this will only work if the *number* of choices for the second object doesn't depend on which first object we choose.

Miis: preset characters

Other than the 96 possible custom Miis,
a player can choose one of 10 preset characters.

How many different characters can be chosen?

A. 96
B. 10
C. 106
D. 960
E. None of the above.



Sum rule

Rosen p. 389

For any disjoint sets, A and B: |A U B| = |A| + |B|

In our example:

A = { custom characters} |A| = 96

B = { preset characters } |B| = 10

A U B = { m : m is a character that is either custom or preset }

|A U B| = the number of possible characters

Sum rule

Rosen p. 389

For any disjoint sets, A and B: |A U B| = |A| + |B|

More generally:

If a task can be done either in one of n_1 ways or in one of n_2 ways, where none of the set of n_1 ways is the same as any of the set of n_2 ways, then there are $n_1 + n_2$ ways to do the task.

Sum rule

Rosen p. 389

For any disjoint sets, A and B: |A U B| = |A| + |B|

More generally:

To count the number of objects with a given property:
* Divide the set of objects into mutually exclusive (disjoint/nonoverlapping) groups.
* Count each group separately.
* Add up these counts.

Length n binary strings

Select which method lets us count the ~~number of~~ length n binary strings.

number of possible

A. The product rule.
B. The sum rule.
C. Either rule works.
D. Neither rule works.

Length n binary strings

Select which method lets us count the number of length n binary strings.

A. The product rule.
B. The sum rule.
C. Either rule works.
D. Neither rule works.

Select first bit, then second, then third …

{0…} U {1…} gives recurrence $N(n) = 2N(n-1)$, $N(0)=1$

Memory: storing length n binary strings

How many binary strings of length n are there?

How many bits does it take to store a length n binary string?

Memory: storing length *n* binary strings

How many binary strings of length *n* are there? **2^n**

How many bits does it take to store a length *n* binary string? **n**

General principle: number of bits to store an object is

$$\lceil \log_2(\text{number of objects}) \rceil$$

Why the ceiling function?

Memory: storing integers

Scenario: We want to store a non-negative integer that has at most n digits.
How many bits of memory do we need to allocate?

A. n
B. 2^n
C. 10^n
D. $n*\log_2 10$
E. $n*\log_{10} 2$

Ice cream!

At an ice cream parlor, you can choose to have your ice cream in a bowl, cake cone, or sugar cone. There are 20 different flavors available.

How many single-scoop creations are possible?

A. 20
B. 23
C. 60
D. 120
E. None of the above.

Ice cream!

At an ice cream parlor, you can choose to have your ice cream in a bowl, cake cone, or sugar cone. There are 20 different flavors available.

You can convert your single-scoop of ice cream to a sundae. Sundaes come with your choice of caramel or hot-fudge. Whipped cream and a cherry are options. How many desserts are possible?

A. 20*3*2*2
B. 20*3*2*2*2
C. 20*3 + 20*3*2*2
D. 20*3 + 20*3*2*2*2
E. None of the above.

A scheduling problem

In one request, four jobs arrive to a server: J1, J2, J3, J4.

The server starts each job right away, splitting resources among all active ones.

Different jobs take different amounts of time to finish.

How many possible finishing orders are there?

A. 4^4
B. 4+4+4+4
C. 4 * 4
D. None of the above.

A scheduling problem

In one request, four jobs arrive to a server: J1, J2, J3, J4.

The server starts each job right away, splitting resources among all active ones.

Different jobs take different amounts of time to finish.

How many possible finishing orders are there?

Which options are available will depend on first choice; but the **number** of options will be the same.

Product rule analysis

- 4 options for which job finishes first.
- Once pick that job, 3 options for which job finishes second.
- Once pick those two, 2 options for which job finishes third.
- Once pick first three jobs, only 1 remains.

$(4)(3)(2)(1) = 4! = 24$

Permutations

Permutation:

Rosen p. 407

rearrangement / ordering of n distinct objects so that each object appears exactly once

Theorem 1: The number of permutations of n objects is

$$\mathbf{n! = n(n-1)(n-2) \ldots (3)(2)(1)}$$

Convention: $0! = 1$

Traveling salesperson

Planning a trip to

New York
Chicago
Baltimore
Los Angeles
San Diego
Minneapolis
Seattle

Must **start in New York** and **end in Seattle**.

How many ways can the trip be arranged?

A. 7!
B. 2^7
C. None of the above.

Traveling salesperson

Planning a trip to

New York
Chicago
Baltimore
Los Angeles
San Diego
Minneapolis
Seattle

Must **start in New York** and **end in Seattle**.
Must also **visit Los Angeles immediately after San Diego**.

How many ways can the trip be arranged now?

Traveling salesperson

Planning a trip to

New York
Chicago
Baltimore
Los Angeles
San Diego
Minneapolis
Seattle

Treat LA & SD as a single stop.

(1)(4!)(1) = 24 arrangements.

Must **start in New York** and **end in Seattle**.
Must also **visit Los Angeles immediately after San Diego**.

How many ways can the trip be arranged now?

Traveling salesperson

Planning a trip to

New York
Chicago
Baltimore
Los Angeles
San Diego
Minneapolis
Seattle

Must **start in New York** and **end in Seattle**.
Must also **visit Los Angeles and San Diego immediately after each other (in any order).**

How many ways can the trip be arranged now?

Traveling salesperson

Planning a trip to

New York
Chicago
Baltimore
Los Angeles
San Diego
Minneapolis
Seattle

Break into two disjoint cases:
Case 1: LA before SD 24 arrangements
Case 2: SD before LA 24 arrangements

Must **start in New York** and **end in Seattle**.
Must also **visit Los Angeles and San Diego immediately after each other (in any order).**

How many ways can the trip be arranged now?

Traveling salesperson

Planning a trip to

- New York
- Chicago
- Baltimore
- Los Angeles
- San Diego
- Minneapolis
- Seattle

Realistically, choose order of visiting cities based on distance…
we wouldn't go to Los Angeles, then Minneapolis, then San Diego, then New York, then Seattle, then Chicago, etc.

Must **start in New York** and **end in Seattle**.
Must also **visit Los Angeles and San Diego immediately after each other (in any order).**

How many ways can the trip be arranged now?

Traveling salesperson

Planning a trip to

- New York
- Chicago
- Baltimore
- Los Angeles
- San Diego
- Minneapolis
- Seattle

Is there an order of visiting the cities that stops at each city exactly once and minimizes the distance traveled?

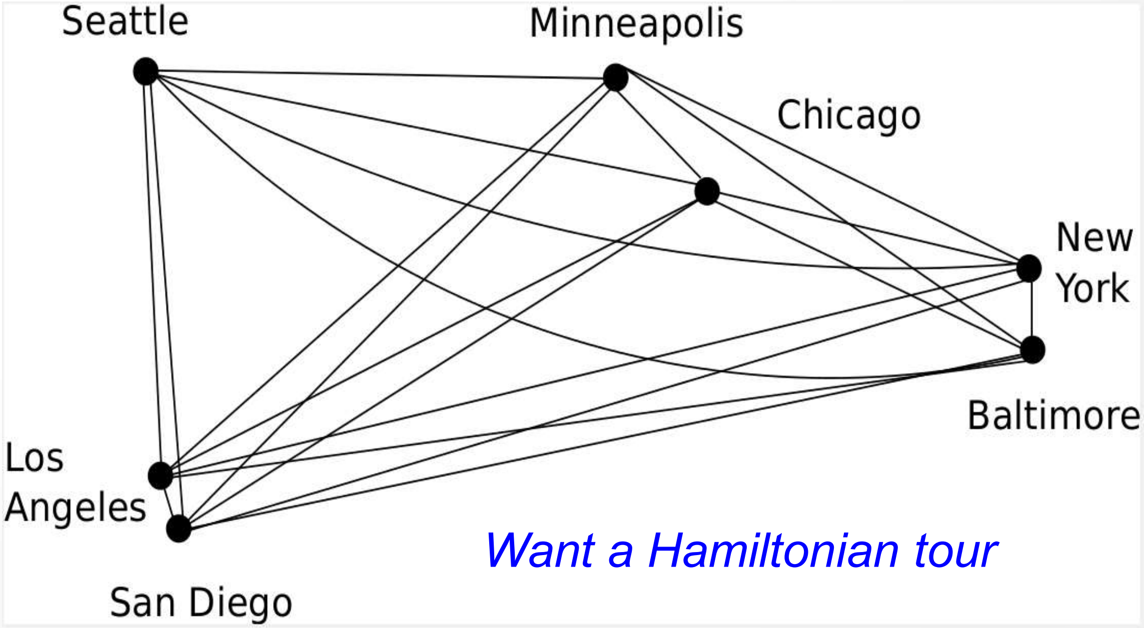
	NY	Chicago	Balt.	LA	SD	Minn.	Seattle
NY	0	800	200	2800	2800	1200	2900
Chicago	800	0	700	2000	2100	400	2000
Balt.	200	700	0	2600	2600	1100	2700
LA	2800	2000	2600	0	100	1900	1100
SD	2800	2100	2600	100	0	2000	1300
Minn.	1200	400	1100	1900	2000	0	1700
Seattle	2900	2000	2700	1100	1300	1700	0

Traveling salesperson

Planning a trip to

- New York
- Chicago
- Baltimore
- Los Angeles
- San Diego
- Minneapolis
- Seattle

Is there an order of visiting the cities that stops at each city exactly once and minimizes the distance traveled?



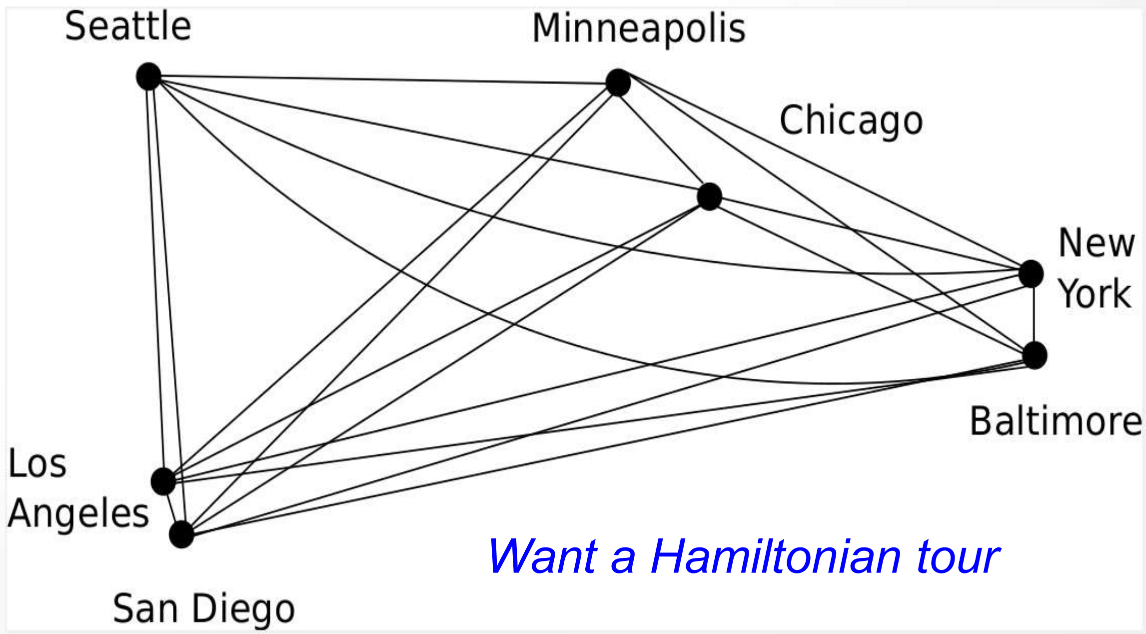
Want a Hamiltonian tour

Traveling salesperson

Developing an algorithm which, given a set of cities and distances between them, computes a shortest distance path between all of them is **NP-hard** (considered intractable, very hard).

Is there **any** algorithm for this question?

A. No, it's not possible.
B. Yes, it's just very slow.
C. ?

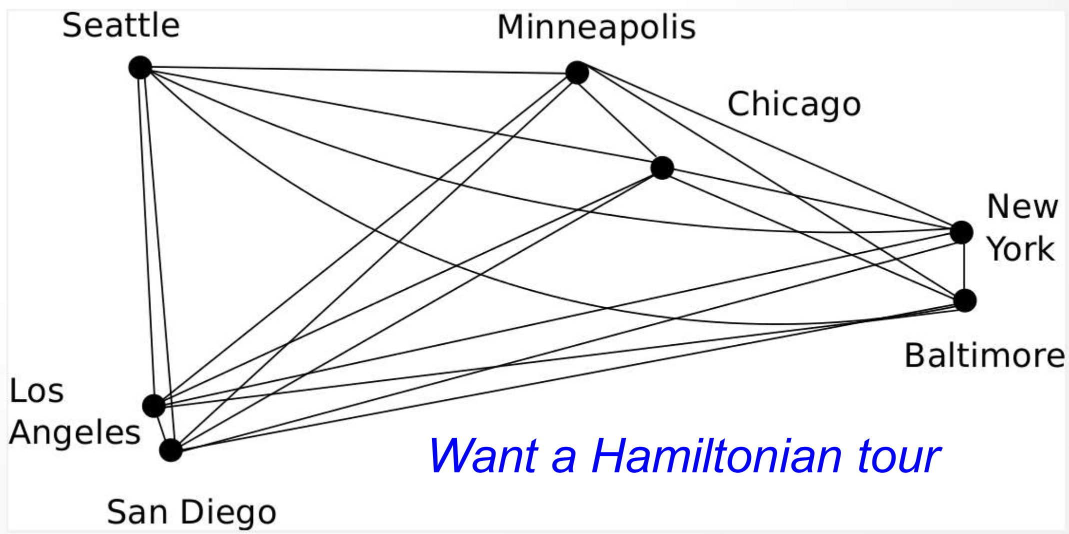


Traveling salesperson

Exhaustive search algorithm

List all possible orderings of the cities.
For each ordering, compute the distance traveled.
Choose the ordering with minimum distance.

How long does this take?



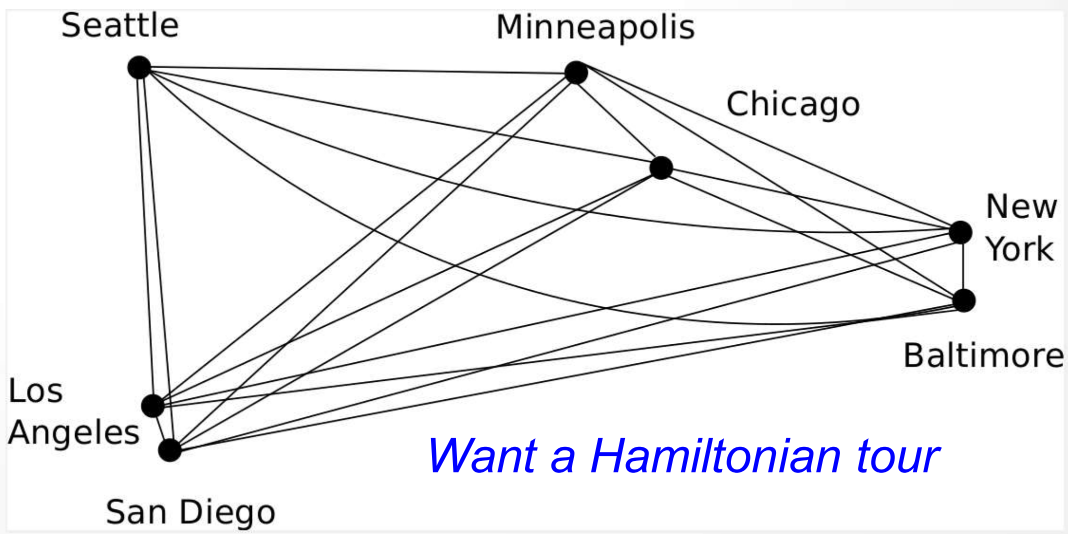
Traveling salesperson

Exhaustive search algorithm: given *n* cities and distances between them.

List all possible orderings of the cities.
For each ordering, compute the distance traveled.
Choose the ordering with minimum distance.

O(number of orderings)

How long does this take?



Traveling salesperson

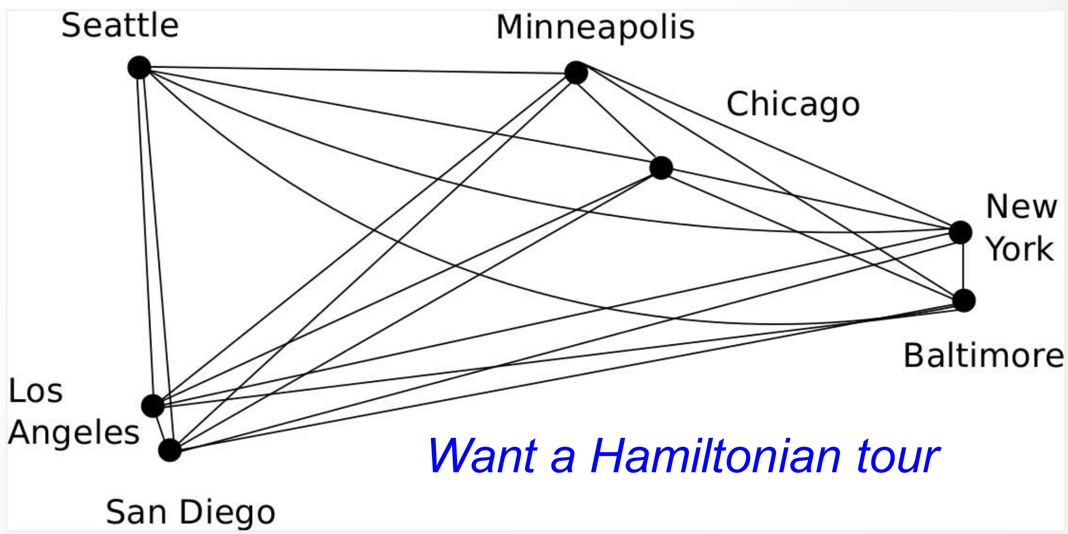
Exhaustive search algorithm: given *n* cities and distances between them.

List all possible orderings of the cities.
For each ordering, compute the distance traveled.
Choose the ordering with minimum distance.

O(number of orderings)

How long does this take?

A. O(n)
B. $O(n^2)$
C. $O(n^n)$
D. O(n!)
E. None of the above.



Traveling salesperson

Exhaustive search algorithm: given *n* cities and distances between them.

List all possible orderings of the cities.
For each ordering, compute the distance traveled.
Choose the ordering with minimum distance.

O(number of orderings)

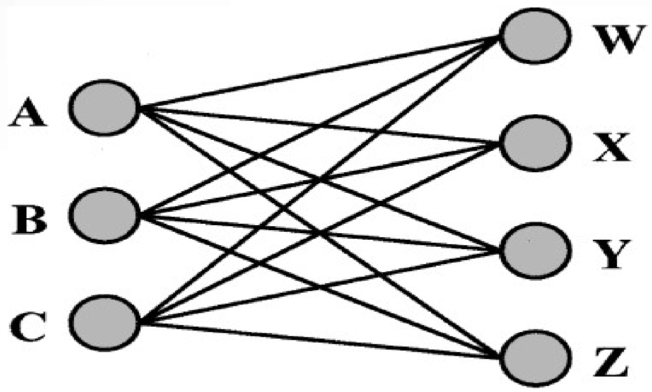
How long does this take?

A. O(n)
B. $O(n^2)$
C. $O(n^n)$
D. O(n!)
E. None of the above.

$2^n < n! < n^n$
for large n

Moral: counting gives upper bound on algorithm runtime.

Bipartite Graphs



Rosen p. 658

A **complete bipartite graph** is an undirected graph whose vertex set is partitioned into two sets V_1, V_2 such that

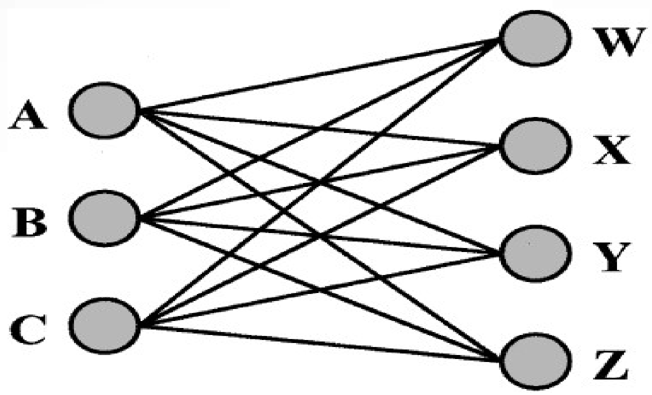
- there is an edge between each vertex in V_1 and each vertex in V_2
- there are no edges both of whose endpoints are in V_1
- there are no edges both of whose endpoints are in V_2

Is this graph Hamiltonian?

A. Yes
B. No

Bipartite Graphs

Rosen p. 658



A **complete bipartite graph** is an undirected graph whose vertex set is partitioned into two sets V_1, V_2 such that

- there is an edge between each vertex in V_1 and each vertex in V_2
- there are no edges both of whose endpoints are in V_1
- there are no edges both of whose endpoints are in V_2

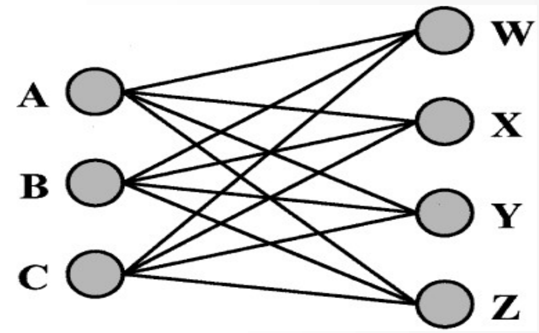
Is every complete bipartite graph Hamiltonian?

A. Yes

B. No

Bipartite Graphs

Rosen p. 658



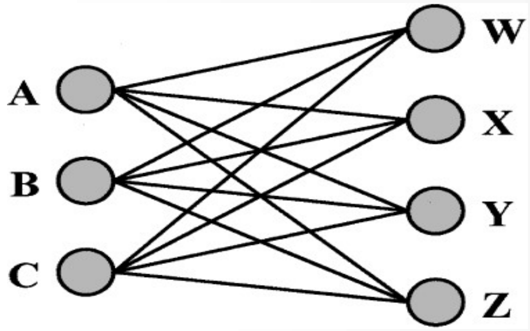
Claim: any **complete bipartite graph** with $|V_1| = k$, $|V_2| = k+1$ is Hamiltonian.

How many Hamiltonian tours can we find?

A. k
B. k(k+1)
C. k!(k+1)!
D. (k+1)!
E. None of the above.

Bipartite Graphs

Rosen p. 658



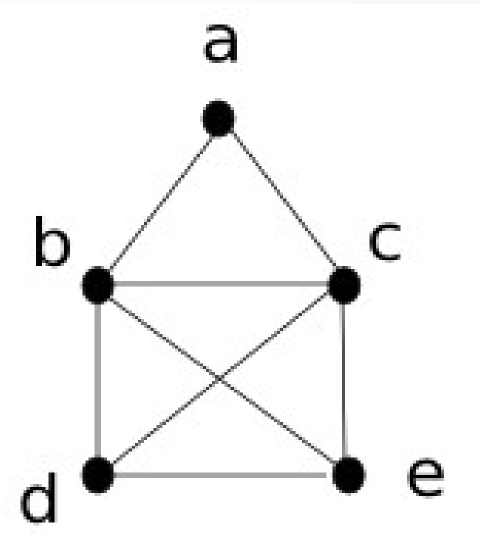
Claim: any **complete bipartite graph** with $|V_1| = k$, $|V_2| = k+1$ is Hamiltonian.

How many Hamiltonian tours can we find?

A. k
B. k(k+1)
C. k!(k+1)!
D. (k+1)!
E. None of the above.

Product rule!

When product rule fails

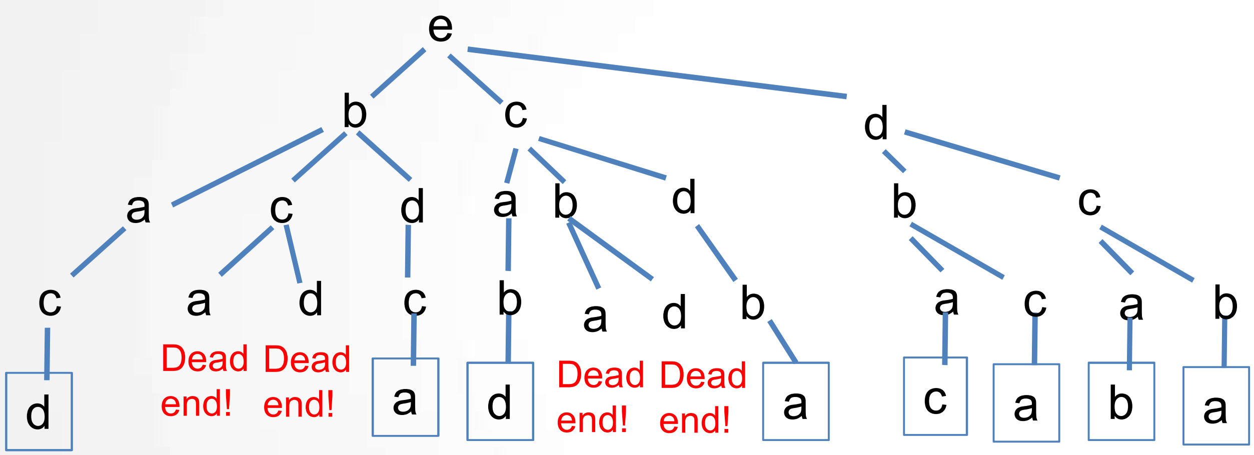


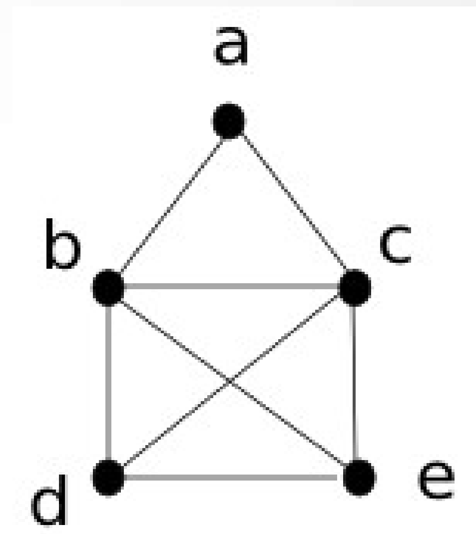
How many Hamiltonian tours can we find?

A. 5!
B. 5!4!
C. ?

When product rule fails

Tree Diagrams



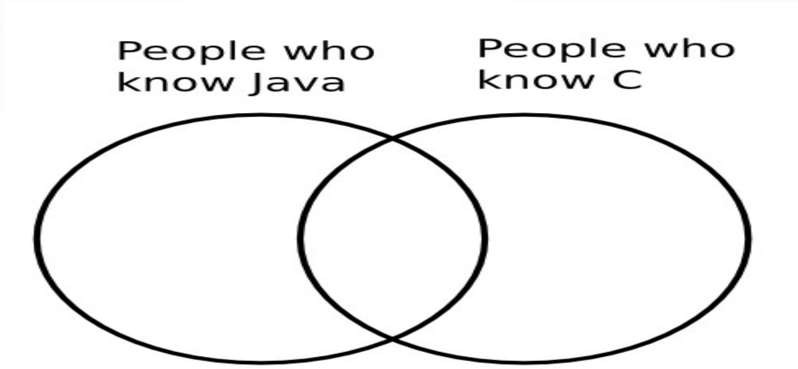


Which Hamiltonian tours start at e?
List all possible next moves.
Then count leaves.

Rosen p.394-395

When sum rule fails

Rosen p. 392-394



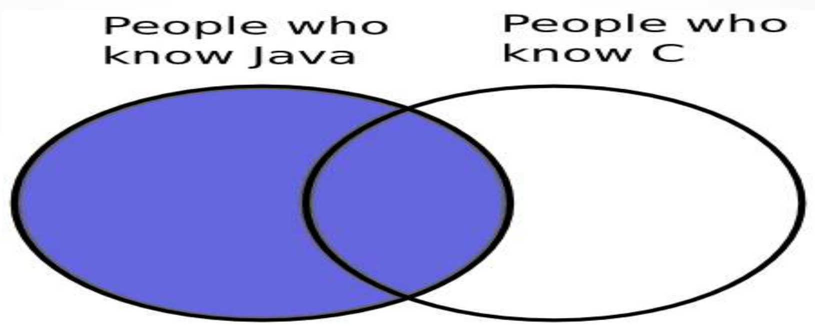
Let A = { people who know Java } and B = { people who know C }

How many people know Java or C (or both)?

A. $|A| + |B|$
B. $|A|\,|B|$
C. $|A|^{|B|}$
D. $|B|^{|A|}$
E. None of the above.

When sum rule fails

Rosen p. 392-394



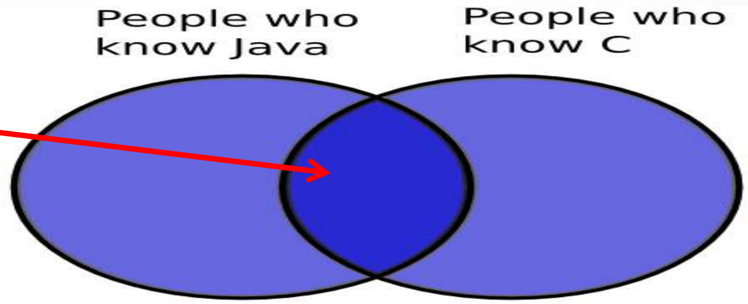
Let A = { people who know Java } and B = { people who know C }

people who know Java or C = # people who know Java

When sum rule fails

Rosen p. 392-394

Double counted!

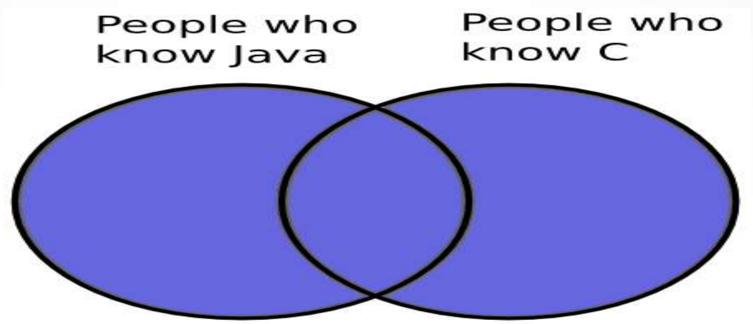


Let A = { people who know Java } and B = { people who know C }

\# people who know Java or C = # people who know Java
+ # people who know C

When sum rule fails

Rosen p. 392-394

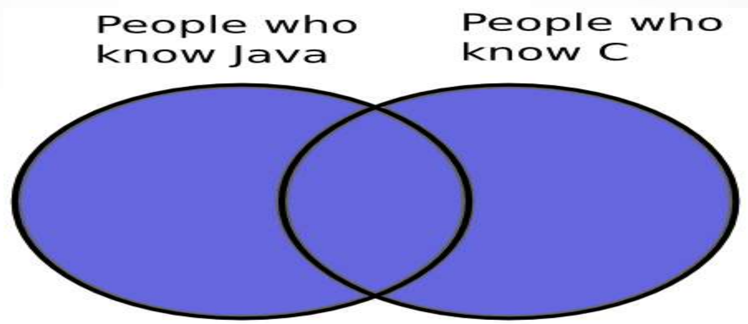


Let A = { people who know Java } and B = { people who know C }

\# people who know Java or C = # people who know Java
+ # people who know C
- # people who know both

Inclusion-Exclusion principle

Rosen p. 392-394

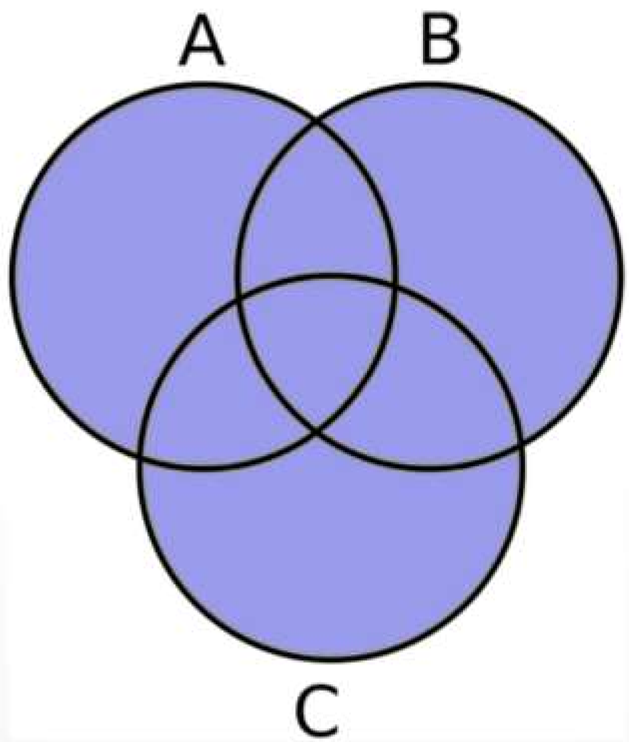


Let A = { people who know Java } and B = { people who know C }

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Inclusion-Exclusion for three sets

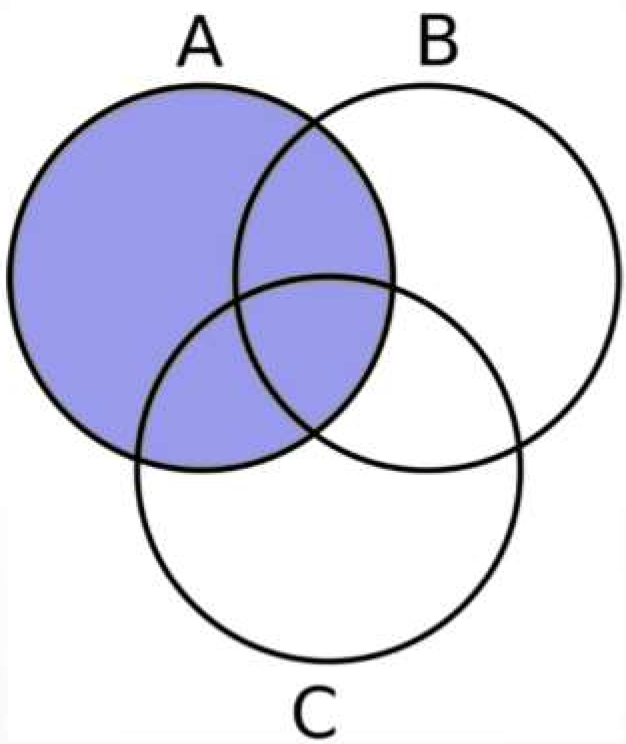
Rosen p. 392-394



$$|A \cup B \cup C| = ?$$

Inclusion-Exclusion for three sets

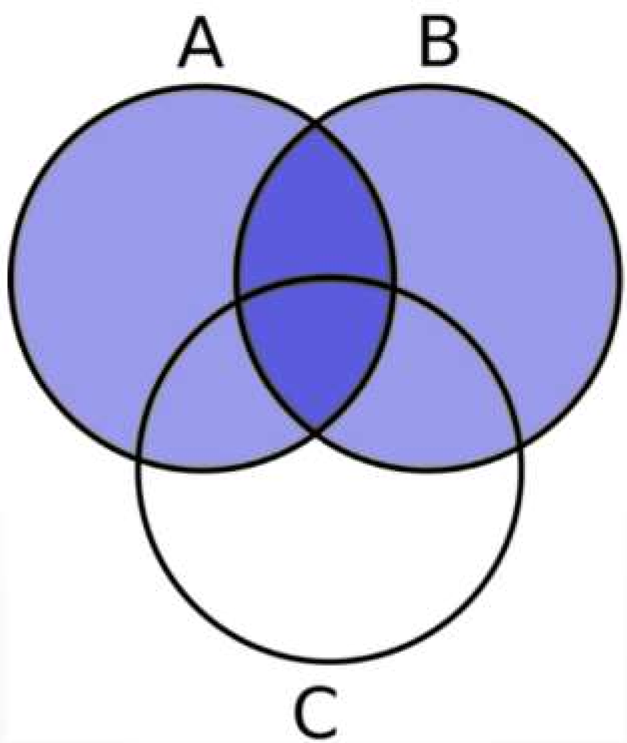
Rosen p. 392-394



$$|A \cup B \cup C| = ?$$

Inclusion-Exclusion for three sets

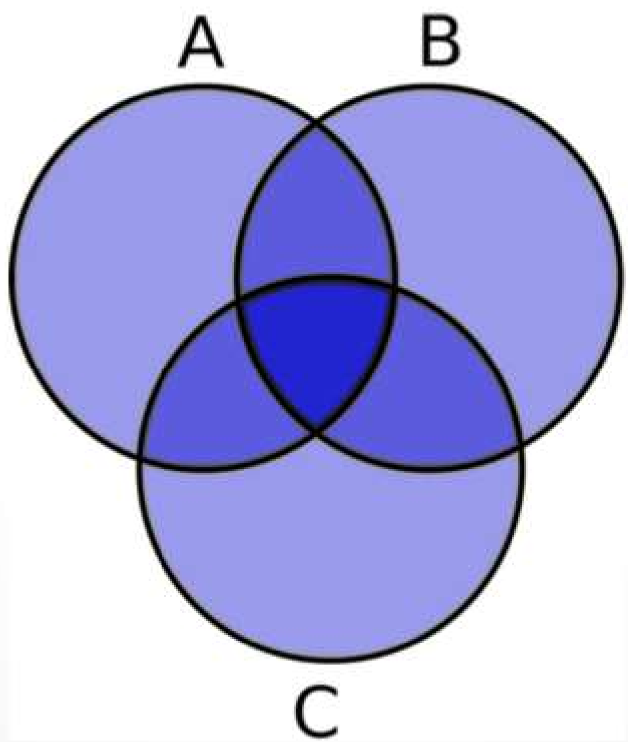
Rosen p. 392-394



$$|A \cup B \cup C| = ?$$

Inclusion-Exclusion for three sets

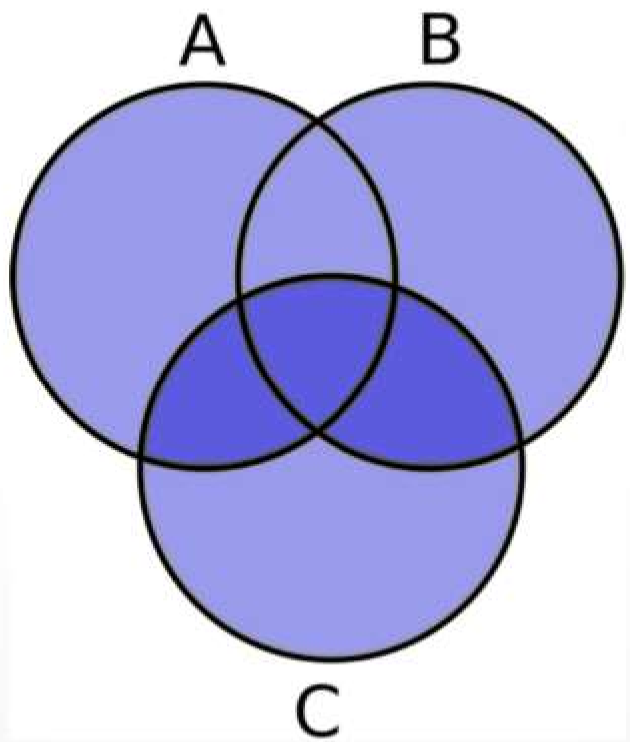
Rosen p. 392-394



$|A \cup B \cup C| = ?$

Inclusion-Exclusion for three sets

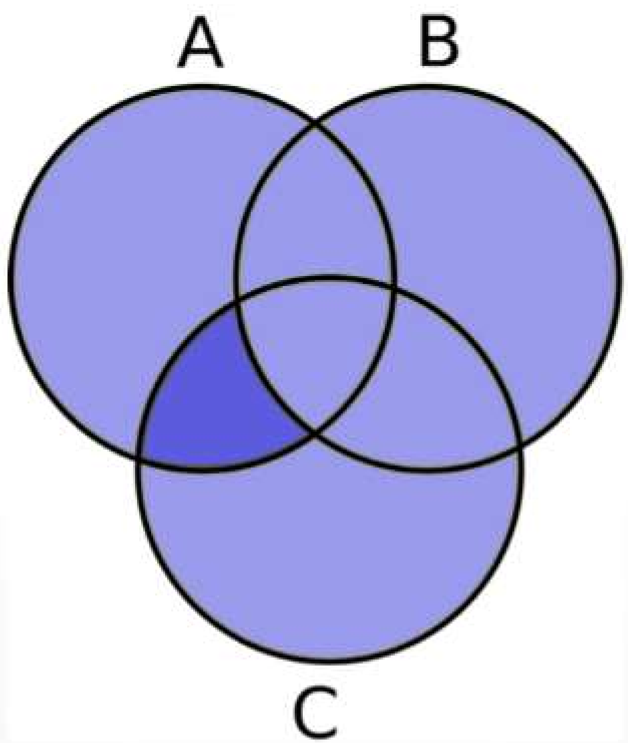
Rosen p. 392-394



$|A \cup B \cup C| = ?$

Inclusion-Exclusion for three sets

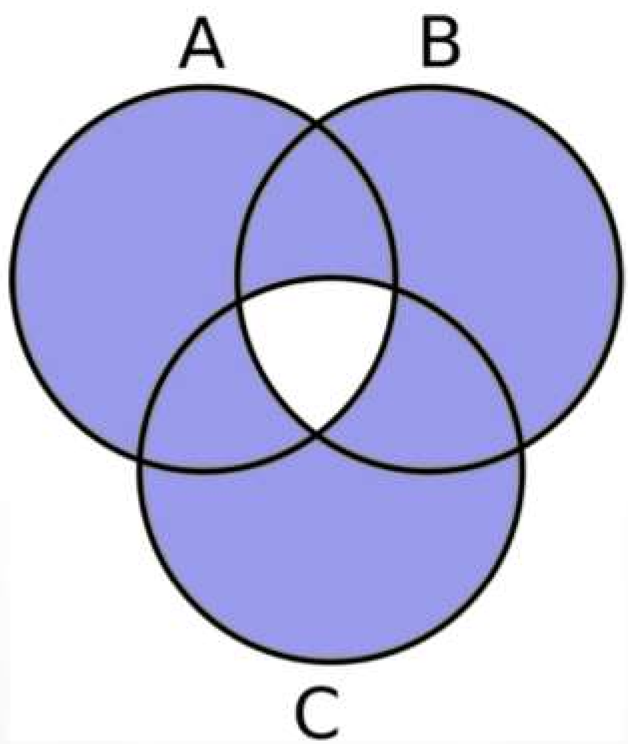
Rosen p. 392-394



$|A \cup B \cup C| = ?$

Inclusion-Exclusion for three sets

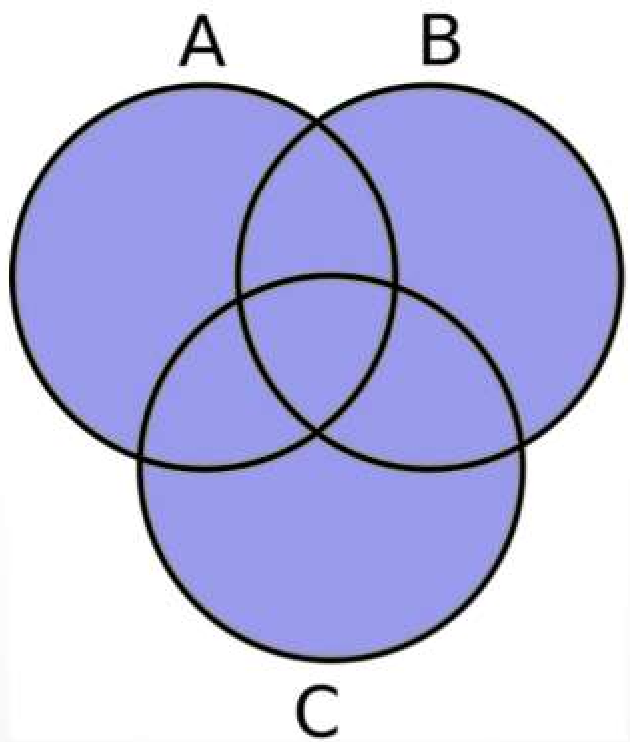
Rosen p. 392-394



$$|A \cup B \cup C| = ?$$

Inclusion-Exclusion for three sets

Rosen p. 392-394



$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

Inclusion-Exclusion principle

Rosen p. 556

If A_1, A_2, …, A_n are finite sets then

$$|A_1 \cup A_2 \cup \cdots A_n| = \sum_{1 \le i \le n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k|$$
$$- \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|$$

Templates

How many four-letter strings have one vowel and three consonants?

There are 5 vowels: AEIOU
and 21 consonants: BCDFGHJKLMNPQRSTVWXYZ.

A. $5*21^3$
B. 26^4
C. 5+52
D. None of the above.

Templates

How many four-letter strings have one vowel and three consonants?

There are 5 vowels: AEIOU
and 21 consonants: BCDFGHJKLMNPQRSTVWXYZ.

Template	*# Matching*
VCCC	5 * 21 * 21 * 21
CVCC	21 * 5 * 21 * 21
CCVC	21 * 21 * 5 * 21
CCCV	21 * 21 * 21 * 5

Total: $4*5*21^3$

Counting with categories

Rosen p. 394

If $A = X_1 \cup X_2 \cup \ldots \cup X_n$ and all X_i, X_j disjoint and all X_i have same size, then

$$|X_i| = |A| / n$$

More generally:

There are n/d ways to do a task if it can be done using a procedure that can be carried out in n ways, and for every way w, d of the n ways give the same result as w did.

Counting with categories

Rosen p. 394

If $A = X_1 \cup X_2 \cup \ldots \cup X_n$ and all X_i, X_j disjoint and all X_i have same size, then

$$|X_i| = |A| / n$$

More generally:

There are n/d ways to do a task if it can be done using a procedure that can be carried out in n ways, and for every way w, d of the n ways give the same result as w did.

Counting with categories

Rosen p. 394

If $A = X_1 \cup X_2 \cup \ldots \cup X_n$ and all X_i, X_j disjoint and all X_i have same size, then

$$|X_i| = |A| / n$$

Or in other words,

If objects are partitioned into categories of equal size, and we want to think of different objects as being the same if they are in the same category, then

categories = (# objects) / (size of each category)

Ice cream!

An ice cream parlor has n different flavors available. How many ways are there to order a two-scoop ice cream cone (where you specify which scoop goes on bottom and which on top, and the two flavors must be different)?

A. n^2
B. $n!$
C. $n(n-1)$
D. $2n$
E. None of the above.

Ice cream!

An ice cream parlor has n different flavors available.
How can we use our earlier answer to decide the number of cones,
if we count two cones as the same if they have the same two flavors
(even if they're in opposite order)?

A. Double the previous answer.
B. Divide the previous answer by 2.
C. Square the previous answer.
D. Keep the previous answer.
E. None of the above.

Ice cream!

An ice cream parlor has n different flavors available.
How can we use our earlier answer to decide the number of cones,
if we count two cones as the same if they have the same two flavors
(even if they're in opposite order)?

Objects:
Categories:
Size of each category:

categories = (# objects) / (size of each category)

Ice cream!

An ice cream parlor has n different flavors available.
How can we use our earlier answer to decide the number of cones,
if we count two cones as the same if they have the same two flavors
(even if they're in opposite order)?

Objects: cones
Categories: flavor pairs (regardless of order)
Size of each category:

categories = (# objects) / (size of each category)

Ice cream!

An ice cream parlor has n different flavors available.
How can we use our earlier answer to decide the number of cones,
if we count two cones as the same if they have the same two flavors
(even if they're in opposite order)?

Objects: cones **n(n-1)**
Categories: flavor pairs (regardless of order)
Size of each category: **2**

categories = (n)(n-1)/ 2

Avoiding double-counting

Object Symmetries

How many different colored triangles can we create by tying these three pipe cleaners end-to-end?

A. 3!
B. 2^3
C. 3^2
D. 1
E. None of the above.

Object Symmetries

How many different colored triangles can we create by tying these three pipe cleaners end-to-end?

Objects: all different colored triangles
Categories: physical colored triangles (two triangles are the same if they can be rotated and/or flipped to look alike)
Size of each category:

categories = (# objects) / (size of each category)

Object Symmetries

How many different colored triangles can we create by tying these three pipe cleaners end-to-end?

Objects: all different colored triangles **3!**
Categories: physical colored triangles (two triangles are the same if they can be rotated and/or flipped to look alike)
Size of each category: **(3)(2)** three possible rotations, two possible flips

categories = (# objects) / (size of each category) = **6/6 = 1**