CSE 100: DISJOINT SET, MST, NP-COMPLETENESS

Announcements

- PA3
  - Final submission deadline 11:59pm on Thursday, December 6 (slip days allowed)
- HW5
  - Due next Wednesday!

Dijkstra's Algorithm

- Priority Queue ordering is based on:

g(n): the distance (cost) from start vertex to vertex n

A* Algorithm

- Priority Queue ordering is based on:

g(n): the distance (cost) from start vertex to vertex n

AND

h(n): the **heuristic estimated cost** from vertex n to goal vertex

f(n) = g(n) + h(n)

A* Algorithm

- Priority Queue ordering is based on:

g(n): the distance (cost) from start vertex to vertex n

AND

h(n): the **heuristic estimated cost** from vertex n to goal vertex

f(n) = g(n) + h(n)

Dijkstra can be seen as a special case where h(n)=0

A* Algorithm

- Priority Queue ordering is based on:

g(n): the distance (cost) from start vertex to vertex n

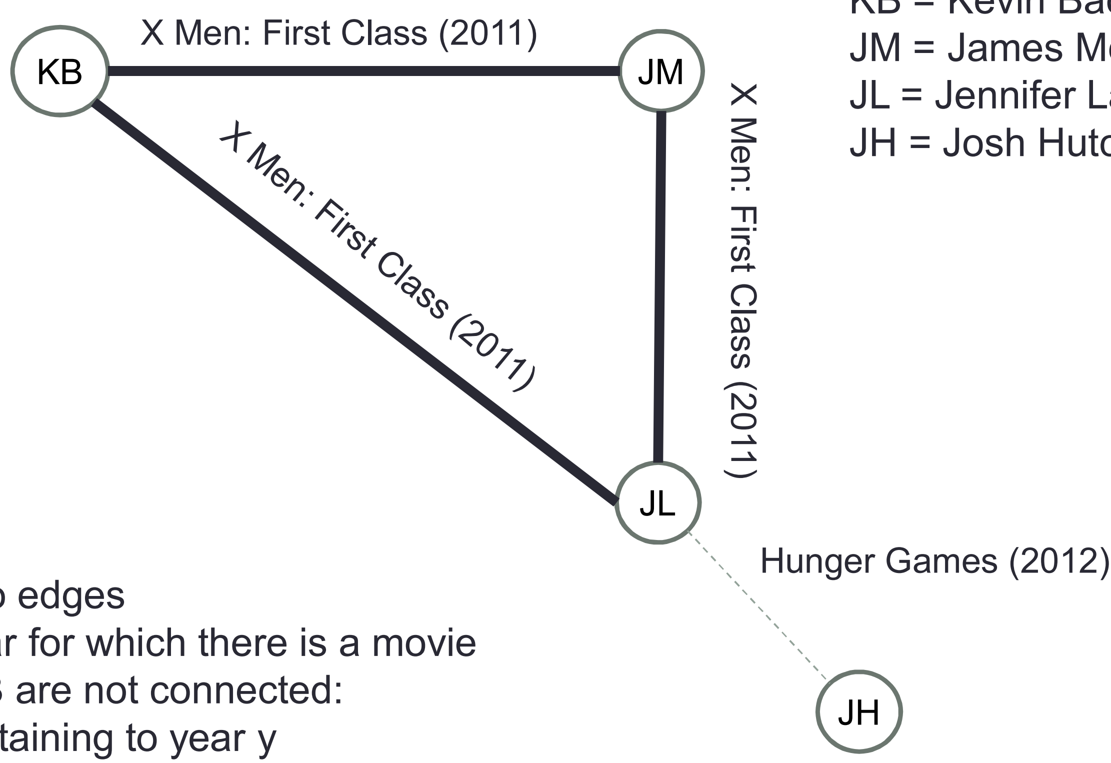
AND

h(n): the **heuristic estimated cost** from vertex n to goal vertex

f(n) = g(n) + h(n)

Guaranteed to find shortest path IF estimate is never an overestimate

The Actor Connections problem



KB = Kevin Bacon
JM = James McAvoy
JL = Jennifer Lawrence
JH = Josh Hutcherson

In 2011

Start with a graph with no edges
Let year y be the first year for which there is a movie
While actor A and actor B are not connected:
    Add all the edges pertaining to year y
    increment y

How do we know if actor A and actor B are connected?

Disjoint Set ADT

- Disjoint Set ADT supports the following operations:
  - Union: Merge two sets
  - Find: Given an element *e*, return its set
- An efficient implementation of a Disjoint Set is to use an Up-tree implementation, where one element is the representative of the set, and other elements point upwards towards it
  - Up-trees can be represented as arrays or as linked structures

Disjoint Set ADT using up-trees

Black: A, B, C, F
Light Blue: A, B, G
Purple: A, C, D
Gold: A, E, F

{A, B}
{C}
{D}
{E}
{F}
{G}

Each tree is a set. Red nodes are sentinels (representatives of the set)

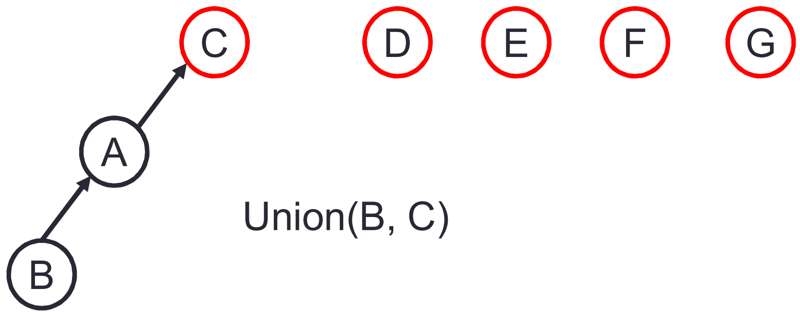
Union(B, A)

Disjoint Set ADT using up-trees

Black: A, B, C, F
Light Blue: A, B, G
Purple: A, C, D
Gold: A, E, F

{C, A, B}
{D}
{E}
{F}
{G}

Each tree is a set. Red nodes are sentinels (representatives of the set)



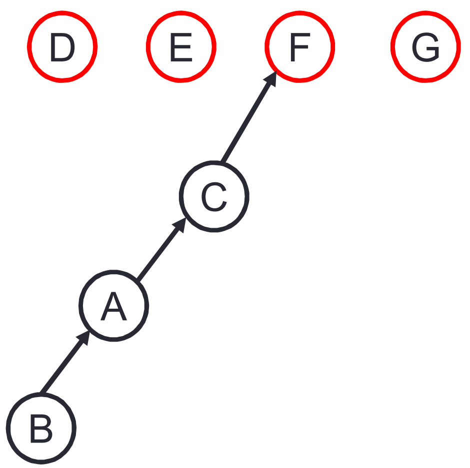
Union(B, C)

Disjoint Set ADT using up-trees

Black: A, B, C, F
Light Blue: A, B, G
Purple: A, C, D
Gold: A, E, F

{D}
{E}
{F, C, A, B}
{G}

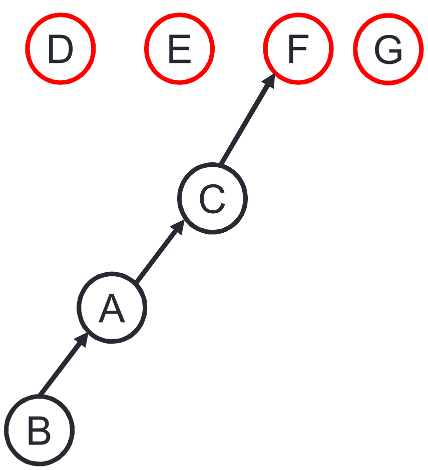
Each tree is a set. Red nodes are sentinels (representatives of the set)



Union(C, F)

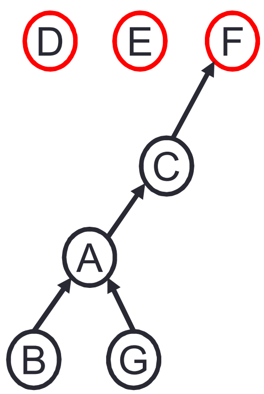
Disjoint Set ADT using up-trees

Each tree is a set. Red nodes are sentinels (representatives of the set)

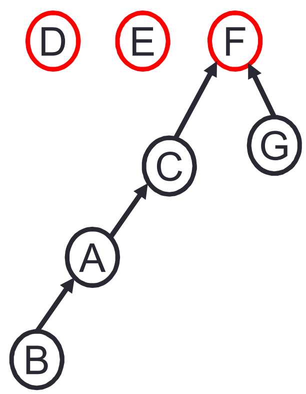


E: More than one of these

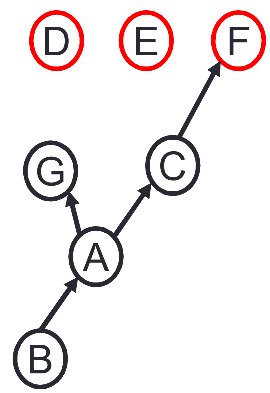
Which of these could be the result of union(A, G)?



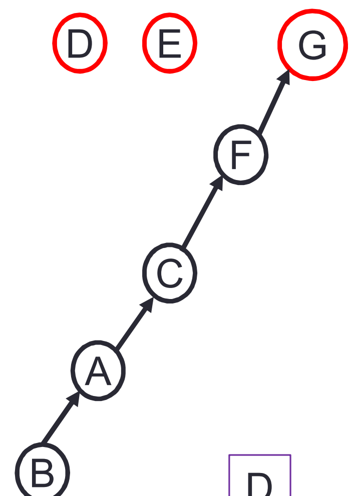
A



B



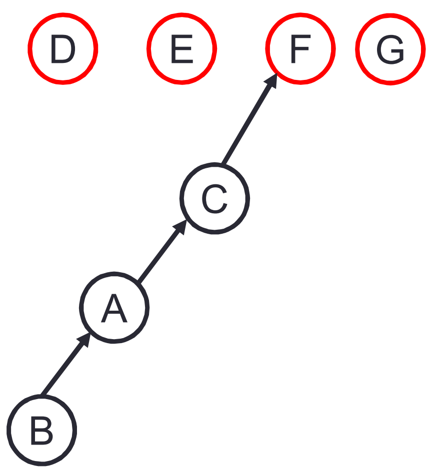
C



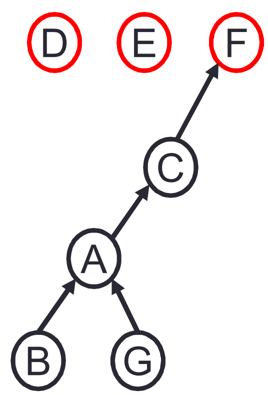
D

Disjoint Set ADT using up-trees

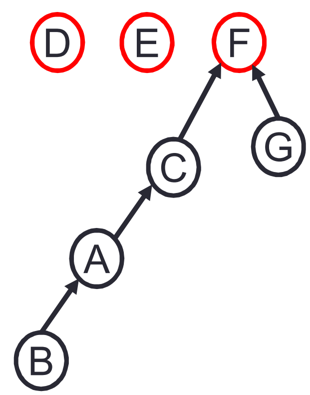
Each tree is a set. Red nodes are sentinels (representatives of the set)



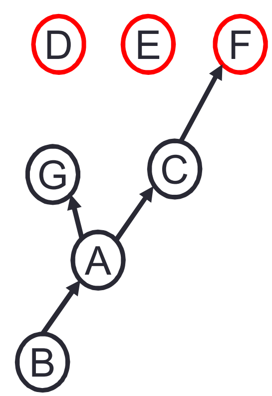
Which of these is the BEST result of union(A, G)?



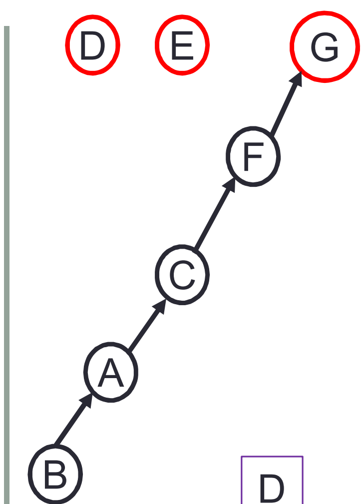
A



B



C



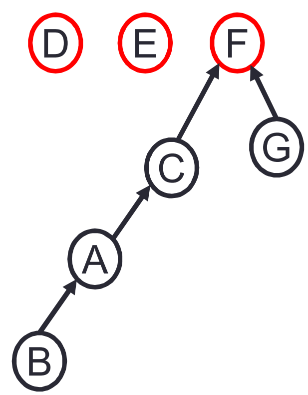
D

Disjoint Set ADT using up-trees

Each tree is a set. Red nodes are sentinels (representatives of the set)

Union(x, y): Make the sentinel of x point to the sentinel of y
Find: Trace up pointers until you reach the sentinel (root)

N is the # of elements in a particular tree



What is the worst case running time of find?

A. O(1)
B. O(logN)
C. O(N)
D. More than O(N)

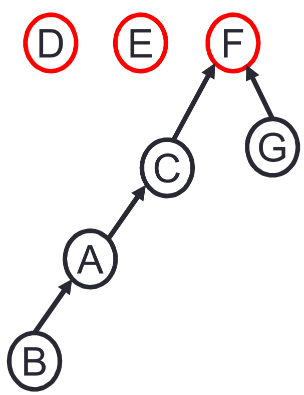
Disjoint Set ADT using up-trees

Each tree is a set. Red nodes are sentinels (representatives of the set)

Union(x, y): Make the sentinel of x point to the sentinel of y
Find: Trace up pointers until you reach the sentinel (root)

Assuming you have already found the sentinel,
What is the running time of union?

A. O(1)
B. O(logN)
C. O(N)
D. More than O(N)

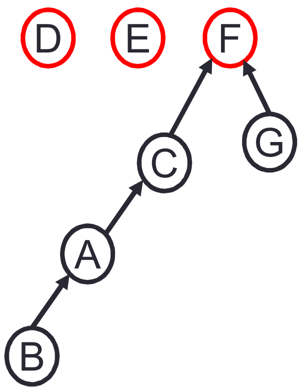


Optimizations: Weighted union

Each tree is a set. Red nodes are sentinels (representatives of the set)

Union(x, y): Make the sentinel of x point to the sentinel of y
Find: Trace up pointers until you reach the sentinel (root)

Weighted union: Make the root the larger tree

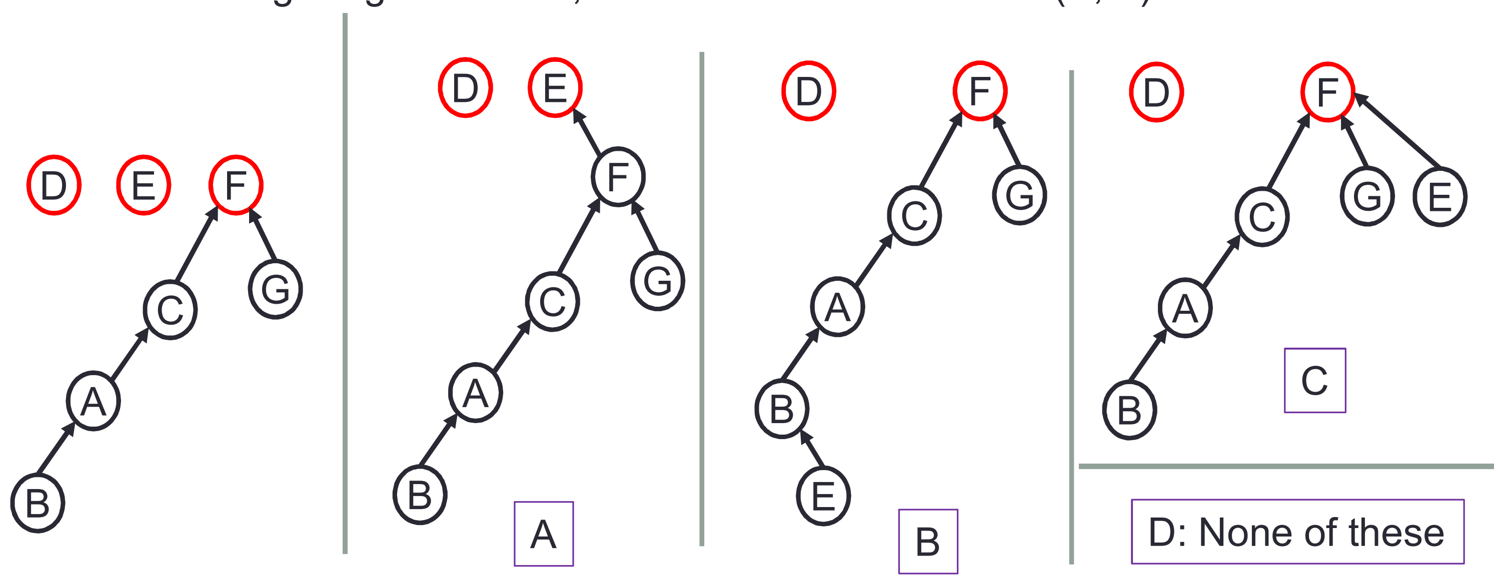


Optimizations: Weighted union

Union(x, y): Make the sentinel of x point to the sentinel of y
Find: Trace up pointers until you reach the sentinel (root)

Weighted union: Make the root the larger tree

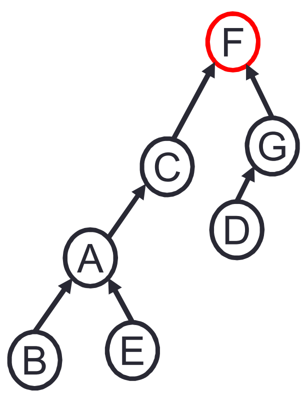
Using weighted union, what is the result of union (E, F)

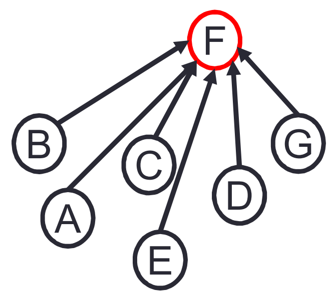


Optimizations: Path Compression

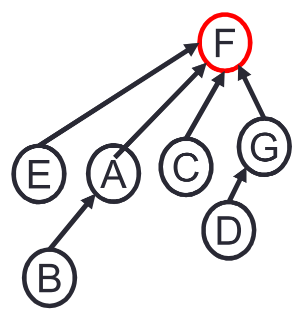
Union(x, y): Make the sentinel of x point to the sentinel of y
Find: Trace up pointers until you reach the sentinel (root)

Path compression: When you do a find, point all nodes on the find path to the root
If you do find(E) on the tree on the left using path compression, what is the result?

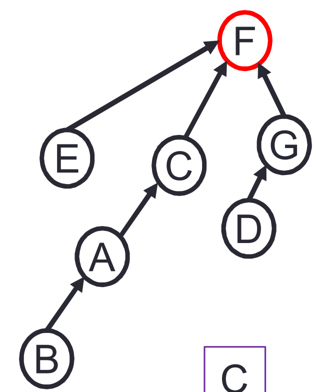




A



B

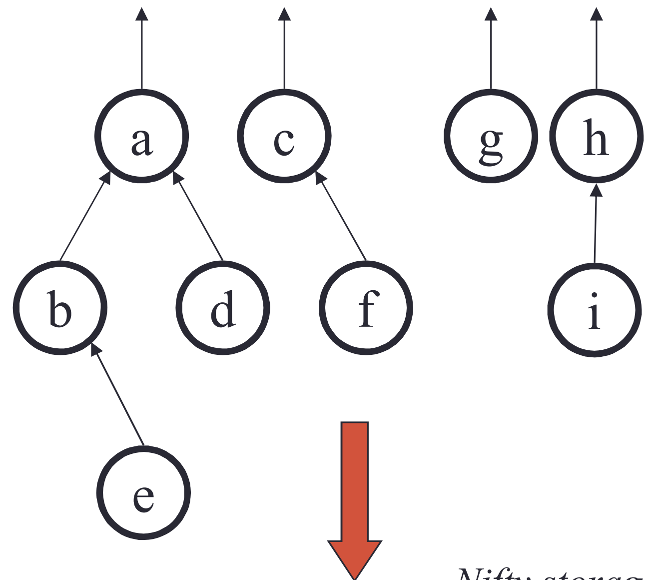


C

D: None of these

Disjoint set data structure using arrays

- A forest of up-trees can easily be stored in an array.
- Also, if the node names are integers or characters, we can use a very simple, perfect hash. Otherwise use a hashmap (C++ unordered map)



Nifty storage trick!

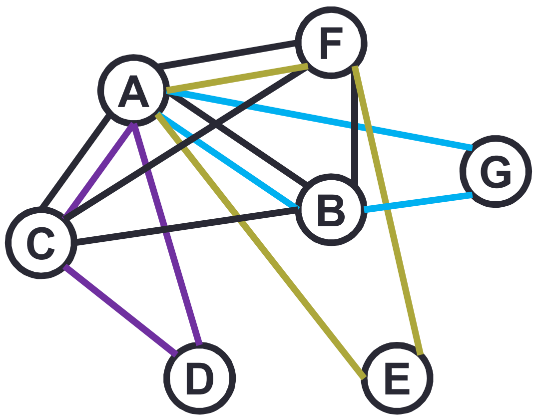
	0 (a)	1 (b)	2 (c)	3 (d)	4 (e)	5 (f)	6 (g)	7 (h)	8 (i)
up-index:	-1	0	-1	0	1	2	-1	-1	7

Slide credit: David Kaplan

Actor Connections, revisited

Develop your algorithm for solving the actor connection question, using disjoint sets. Remember, you have 2 operations: Union and Find

Black (2011): A, B, C, F
Light Blue (2004): A, B, G
Purple (2000): A, C, D
Gold (2013): A, E, F

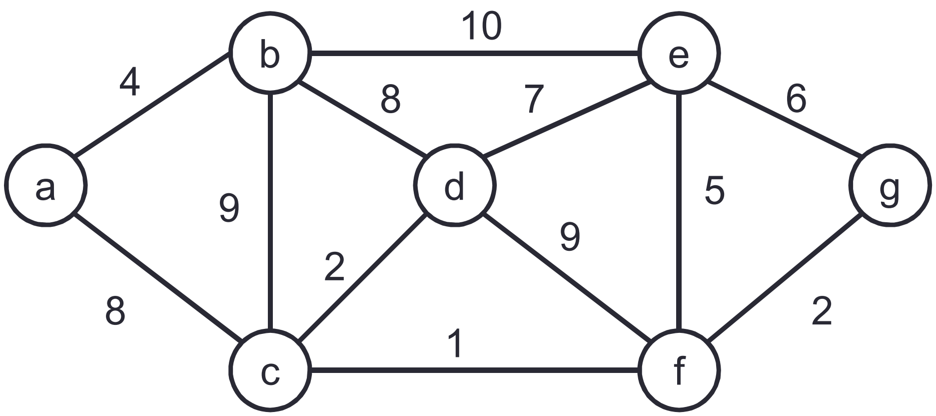


Learning Goals

- Find the minimum spanning tree(s) in a graph
- Analyze Kruskal's algorithm
- Explain and analyze the Traveling Salesperson problem
- Explain the idea of an NP-Complete problem

Minimum spanning trees

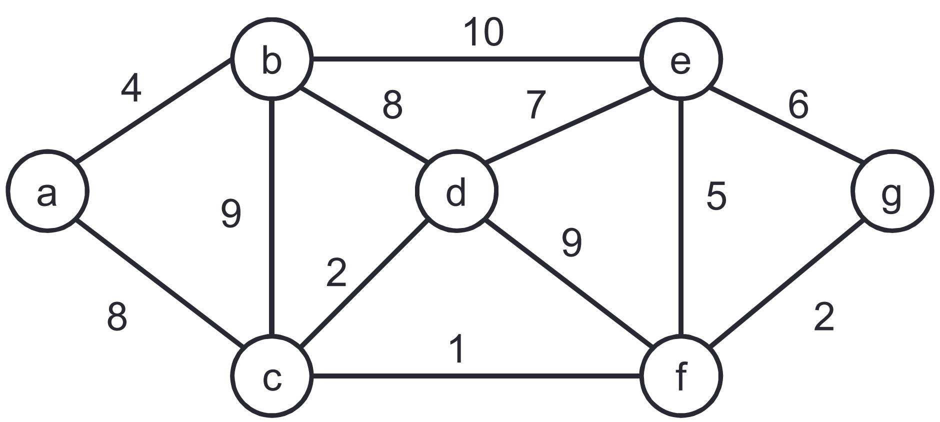
- What is the weight of the minimum spanning tree in this graph? (And what is it)



A. 14
B. 22
C. 28
D. 38
E. Other

Minimum spanning trees: Kruskal's algorithm

- Sort edges from smallest to largest
- Initially place each node into its own subset
- Repeat until all nodes are connected:
  - Select the smallest edge where the endpoints are in different subsets and include that edge in the MST

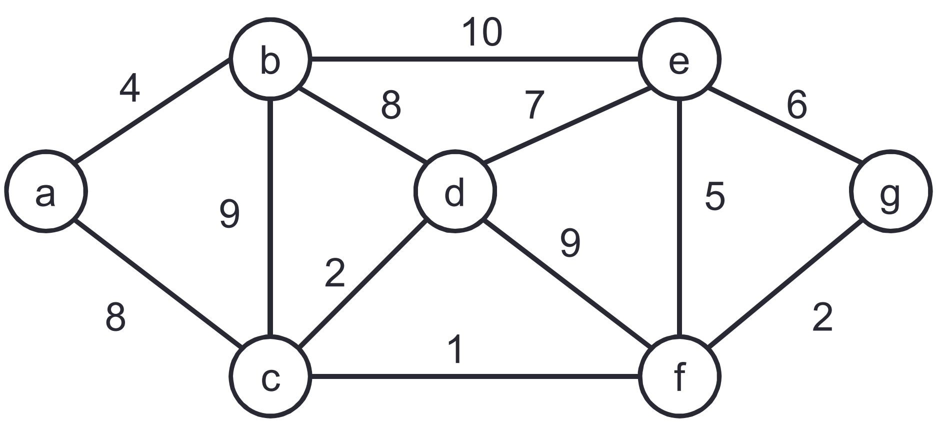


What data structure should you use in this algorithm?

A. A heap
B. A balanced binary search tree
C. A disjoint set
D. More than one of the above

Minimum spanning trees: Kruskal's algorithm

- Sort edges from smallest to largest
- Initially place each node into its own subset
- Repeat until all nodes are connected:
  - Select the smallest edge where the endpoints are in different subsets and include that edge in the MST



What is the worst case running time of Kruskal's algorithm?

A. O(|E|)
B. O(|E| log(|E|)
C. O(|V| * |E|)
D. Other

Assume the graph is *connected* (i.e. no nodes are "floating")

Greedy Algorithm

- A "greedy algorithm" is one that always selects the locally largest step toward the goal.

Greedy Algorithm

- A "greedy algorithm" is one that always selects the locally largest step toward the goal.

<u>Kruskal's algorithm</u>
Sort edges from smallest to largest
Initially place each node into its own subset
Repeat until all nodes are connected:
    Select the smallest edge where the endpoints are in different subsets
        and include that edge in the MST

Is Kruskal's algorithm a greedy algorithm?
A. Yes
B. No
C. Sometimes

Greedy Algorithm

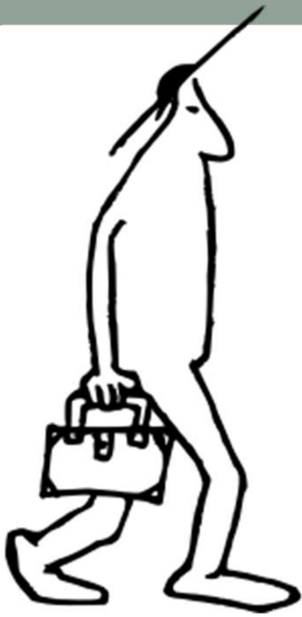
- A "greedy algorithm" is one that always selects the locally largest step toward the goal.

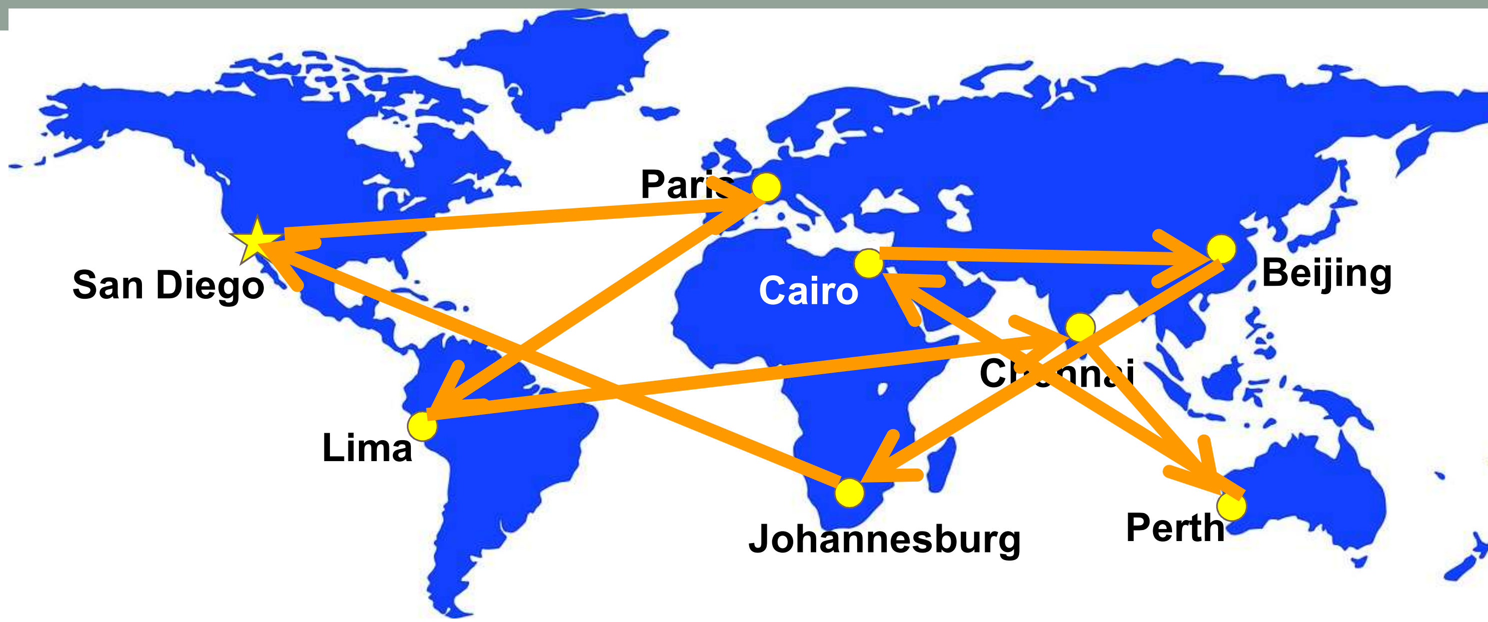
Kruskal's algorithm
Sort edges from smallest to largest
Initially place each node into its own subset
Repeat until all nodes are connected:
    Select the smallest edge where the endpoints are in different subsets
        and include that edge in the MST

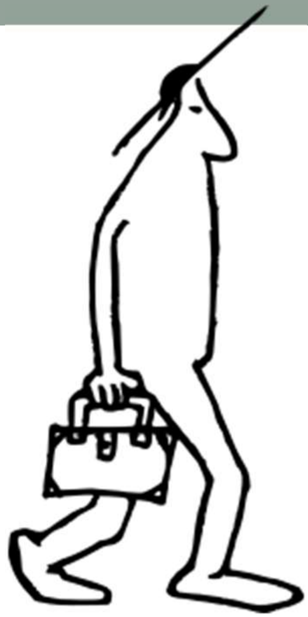
Greedy algorithms are simple and fast, but for MANY problems they do not return the optimal solution.

Is Kruskal's algorithm a greedy algorithm?
A. Yes
B. No
C. Sometimes

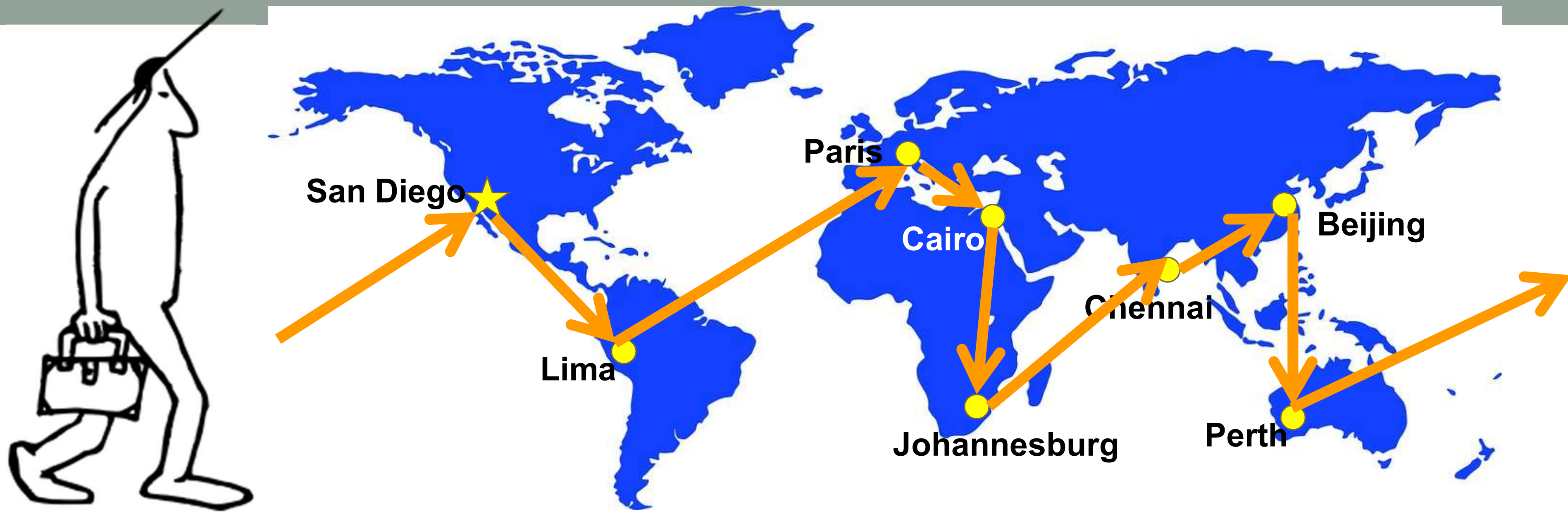

Paris
San Diego
Cairo
Beijing
Chennai
Lima
Johannesburg
Perth

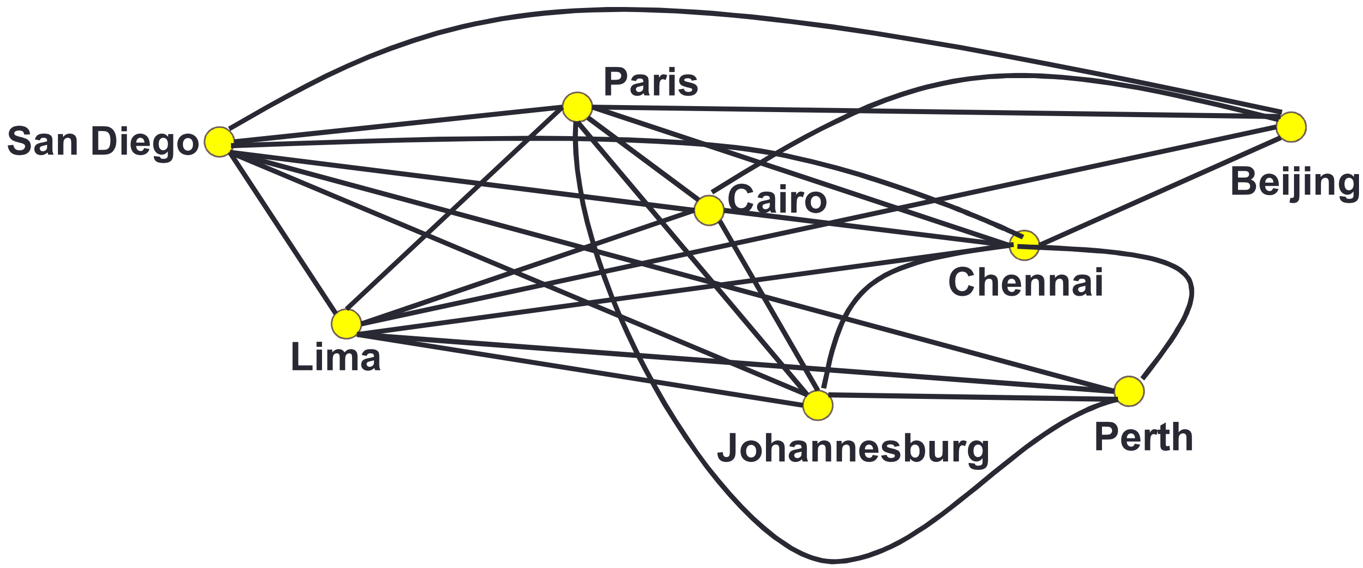



Paris
San Diego
Cairo
Beijing
Lima
Johannesburg
Perth




Paris
San Diego
Cairo
Beijing
Chennai
Lima
Johannesburg
Perth


San Diego
Lima
Paris
Cairo
Johannesburg
Chennai
Beijing
Perth



Is this graph sparse or dense?

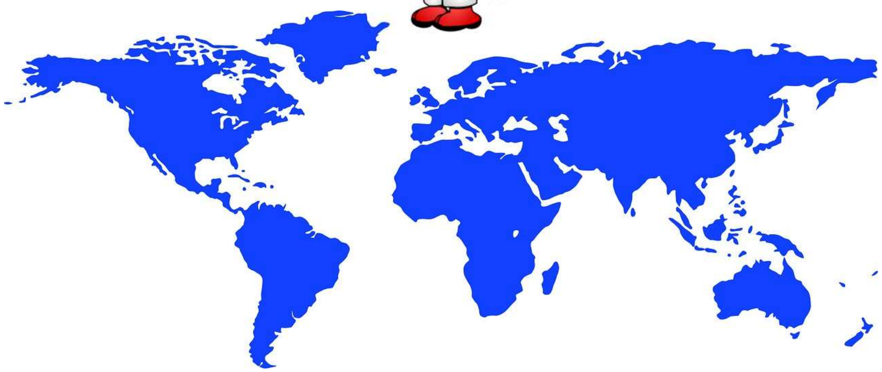
A. Sparse
B. Dense

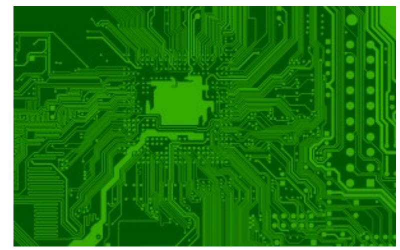
	SD	Lima	Paris	Chen.	Cairo	Perth	Beij.	J'berg
SD	0	6,091	9,144	14,587	12,276	15,078	10,234	16,575
Lima	6,091	0	10,248	17,540	12,414	14,924	16,637	10,872
Paris	9,144	10,248	0	8,031	3210	14,269	8,212	8,295
Chen.	14,587	17,540	8,031	0	5,360	6,276	4,615	7,133
Cairo	12,276	12,414	3210	5,360	0	11,258	7,540	6,260
Perth	15,078	14,924	14,269	6,276	11,258	0	7,985	8,308
Beij.	10,234	16,637	8,212	4,615	7,540	7,985	0	11,699
J'berg	16,575	10,872	8,295	7,133	6,260	8,308	11,699	0

The Traveling Salesperson Problem: Given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Lots of applications!



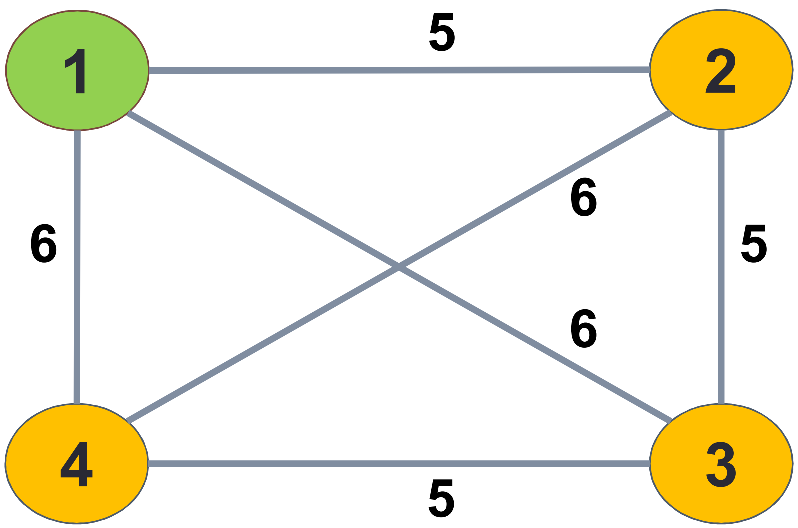


http://www.math.uwaterloo.ca/tsp/index.html

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Greedy algorithm: pick best next choice

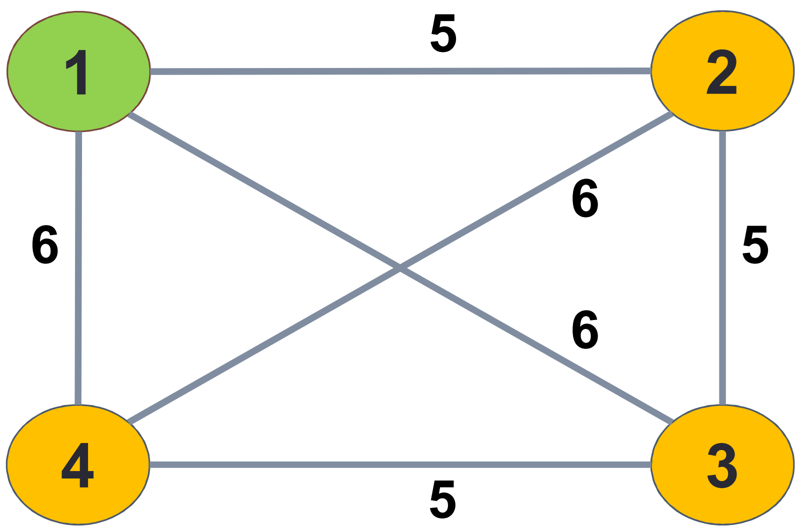
Warmup: What tour does the Greedy algorithm construct for this graph?



A. 1 → 2 → 3 → 4 → 1
B. 1 → 3 → 2 → 4 → 1
C. 1 → 4 → 3 → 2 → 1
D. 1 → 2 → 4 → 3 → 1

Greedy algorithm: pick best next choice

Is this the best possible tour for this graph?



1 → 2 → 3 → 4 → 1

A. Yes
B. No

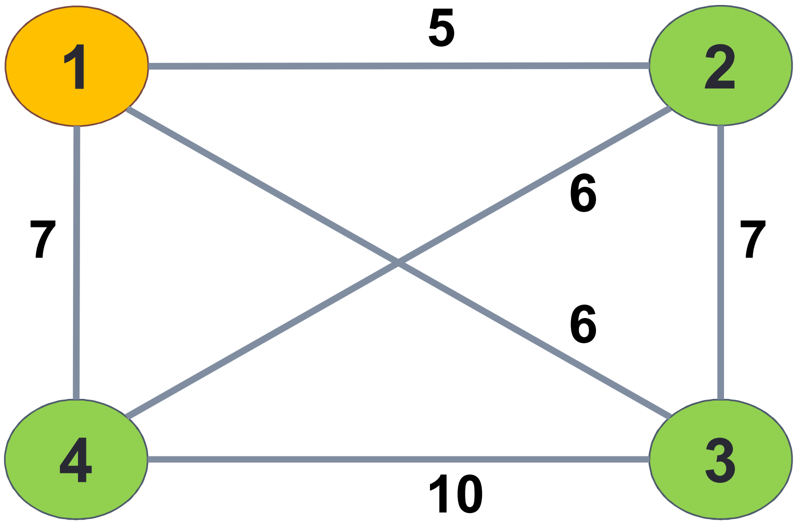
In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

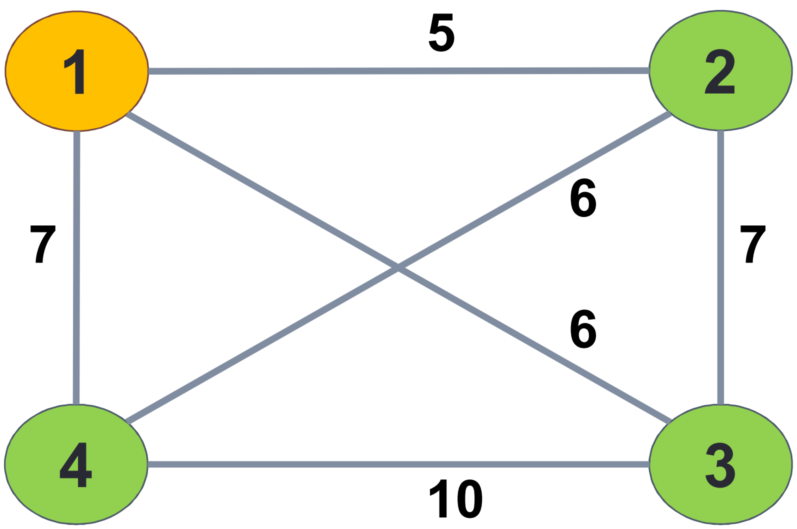
Greedy algorithm: pick best next choice

Will the greedy algorithm always work?

A. Yes
B. No

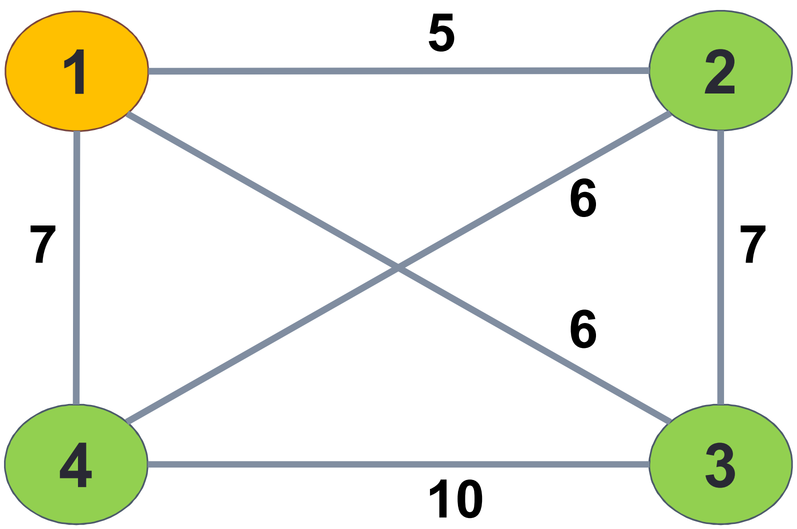
If yes, why? ***If no, find counterexample.***


1
2
3
4
5
6
7
6
7
10



Greedy:

1 → 2 → 4 → 3 → 1



Greedy: **1 → 2 → 4 → 3 → 1** **27**

Optimal: **1 → 3 → 2 → 4 → 1**

6+7+6+7 = 26

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

	SD	**Lima**	**Paris**	**Chen.**	**Cairo**	**Perth**	**Beij.**	**J'berg**
SD	0	6,091	9,144	14,587	12,276	15,078	10,234	16,575
Lima	6,091	0	10,248	17,540	12,414	14,924	16,637	10,872
Paris	9,144	10,248	0	8,031	3210	14,269	8,212	8,295
Chen.	14,587	17,540	8,031	0	5,360	6,276	4,615	7,133
Cairo	12,276	12,414	3210	5,360	0	11,258	7,540	6,260
Perth	15,078	14,924	14,269	6,276	11,258	0	7,985	8,308
Beij.	10,234	16,637	8,212	4,615	7,540	7,985	0	11,699
J'berg	16,575	10,872	8,295	7,133	6,260	8,308	11,699	0

Just try all paths and choose the shortest!

Brute force approach

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

SD → Lima → Paris → Cairo → Perth → Beijing → Chennai → Johannesburg → San Diego

6,091 + 10,248 + 3210 + 11,258 + 7,985 + 4,615 + 7,133 + 16,575 = 67,115km

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

SD → Lima → Paris → Cairo → Perth → Beijing → Chennai → Johannesburg → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 4,615 + 7,133 + 16,575 = 67,115km

SD → Lima → Paris → Cairo → Perth → Johannesburg → Beijing → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 8,308 + 11,699 + 4,615 + 14,587 = 70,016km

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

SD → Lima → Paris → Cairo → Perth → Beijing → Chennai → Johannesburg → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 4,615 + 7,133 + 16,575 = 67,115km

SD → Lima → Paris → Cairo → Perth → Johannesburg → Beijing → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 8,308 + 11,699 + 4,615 + 14,587 = 70,016km

. . .

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 11,699 + [illegible] 7,133 + 14,587 = 72,211km

SD → Lima → Paris → Cairo → Perth → Beiji[illegible]ai → Johannesburg → San Diego

6,091 + 10,248 + 3,210 + 11,25[illegible] 4,615 + 7,133 + 16,575 = 67,115km

It works!!

SD → Lima → Pa[illegible] → Perth → Johannesburg → Beijing → Chennai → San Diego

6,091 + 10,24[illegible] 3,210 + 11,258 + 8,308 + 11,699 + 4,615 + 14,587 = 70,016km

. . .

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

★ **SD → Lima → Paris → Cairo → Johannesburg → Perth → Chennai → Beijing → San Diego**

6,091 + 10,248 + 3,210 + 6,260 + 8,308 + 6,276 + 4,615 + 10,234 = 55,242km

But how long does it take…?

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

Permutations coming next

```
bestPath = null, bestDist = +Infinity
for each permutation of cities, starting and ending in Hometown:
    calculate distance of current permutation
    if (distance < bestDist)
        bestPath = current permutation, bestDist = distance

return bestPath
```

O(n)

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

```
bestPath = null, bestDist = +Infinity
for each permutation of cities, starting and ending in Hometown:
    calculate distance of current permutation
    if (distance < bestDist)
        bestPath = current permutation, bestDist = distance

return bestPath
```

O(n)

But how many permutations?!?

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

How many permutations for a TSP starting with San Diego?

San Diego
Cairo
Johannesburg
Chennai
Lima
Paris
Beijing
Perth

How many permutations are there for the tour?

A. $7!$
B. 7^n
C. 2^7
D. $2*8$

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

How many permutations?

San Diego	**How many choices for the first city? 1 (San Diego)**
Cairo	**How many choices for the next city? 7**
Johannesburg	**How many choices for the next city? 6**
Chennai	**How many choices for the next city? 5**
Lima	**How many choices for the next city? 4**
Paris	**How many choices for the next city? 3**
Beijing	**How many choices for the next city? 2**
Perth	**How many choices for the next city? 1**
	How many choices for the last city? 1 (San Diego)

In general we have (n-1)! permutations to try!

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

(n-1)! permutations

```
bestPath = null, bestDist = +Infinity
for each permutation of cities, starting and ending in Hometown:
    calculate distance of current permutation
    if (distance < bestDist)
        bestPath = current permutation, bestDist = distance

return bestPath
```

O(n)

(n-1)! * n = O(n!)

N	N!
10	~3.6 million
19	1.22 x 10^{17} (the age of the universe)
23	# of stars in the universe
59	# of atoms in the universe

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Greedy algorithm: pick best next choice

```
bestPath = []
current = Hometown
cities to visit = all other cities
while (more cities to visit)
    select city closest to current and add to bestPath
    remove current city from cities to visit
    current = selected city
return bestPath
```

What is the running time of the greedy algorithm?

A. $O(n)$
B. $O(n^2)$
C. $O(n^3)$
D. $O(n!)$

TSP Brute Force

N	N!
10	~3.6 million
19	1.22×10^{17} (the age of the universe)
23	# stars in the universe
59	# of atoms in the universe

Yikes!

What do we do now?

Think really hard about a faster solution?

Complexity Theory

Classifies problems by their inherent difficulty

Searching a Linked List – O(n)

Sorting an Array – O(n log n)

n x n Matrix-Matrix Multiply– $O(n^{\sim 2.37})$

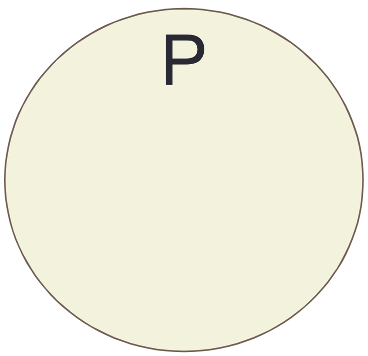
Complexity Theory

Classifies problems by their inherent difficulty

Searching a Linked List – O(n)

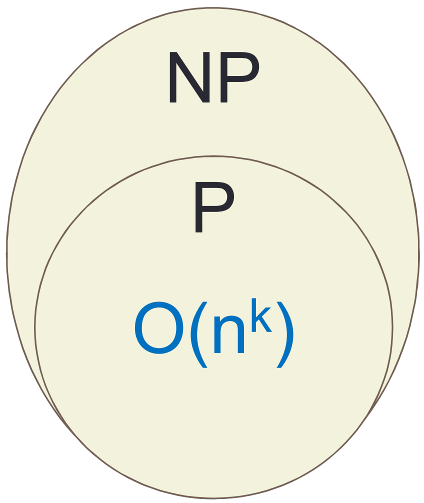
Sorting an Array – O(n log n)

n x n Matrix-Matrix Multiply– O($n^{\sim 2.37}$)



Complexity Theory

Classifies problems by their inherent difficulty



Running Times

(or why people worry about algorithm complexity)

problem size → complexity ↓	n = 10	100	1,000	10,000	100,000	1,000,000
log n	3.3219	6.6438	9.9658	13.287	16.609	19.931
$\log^2 n$	10.361	44.140	99.317	176.54	275.85	397.24
sqrt n	3.162	10	31.622	100	316.22	1000
n	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
n log n	33.219	664.38	9965.8	132877	$1.66*10^6$	$1.99*10^7$
$n^{1.5}$	31.6	10^3	$31.6*10^4$	10^6	$31.6*10^7$	10^9
n^2	100	10^4	10^6	10^8	10^{10}	10^{12}
n^3	1000	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	1024	10^{30}	10^{301}	10^{3010}	10^{30103}	10^{301030}
n!	3 628 800	$9.3*10^{157}$	10^{2567}	10^{35659}	10^{456573}	$10^{5565710}$

O

Running times

Running times of different big-O algorithms for larger and larger inputs.

P ?= NP How to get rich and famous

CMI

ABOUT PROGRAMS MILLENNIUM PROBLEMS

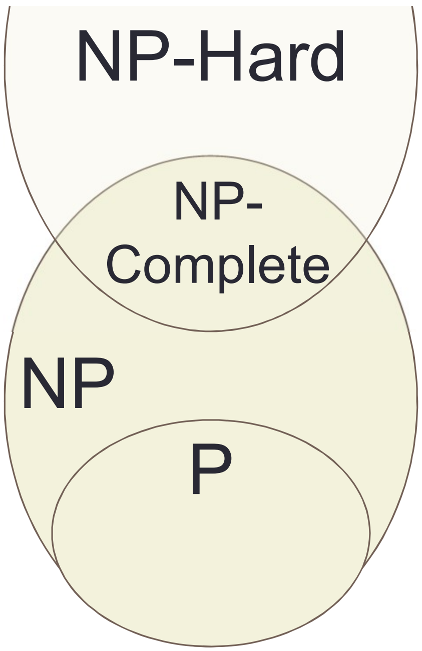
The Millennium Prize Problems

Following the decision of the Scientific Advisory Board, the Board of Directors of CMI designated a $7 million prize fund for the solutions to these problems, with $1 million allocated to the solution of each problem.

, with $1 million allocated to the solution of each problem.

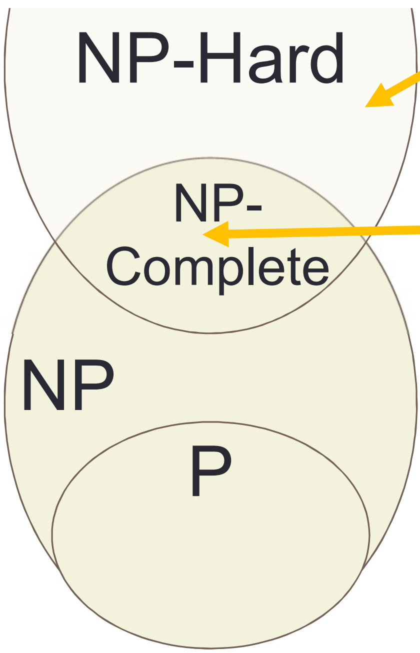
http://www.claymath.org/millennium-problems

Complexity Theory



(Hierarchy if P != NP)

Complexity Theory



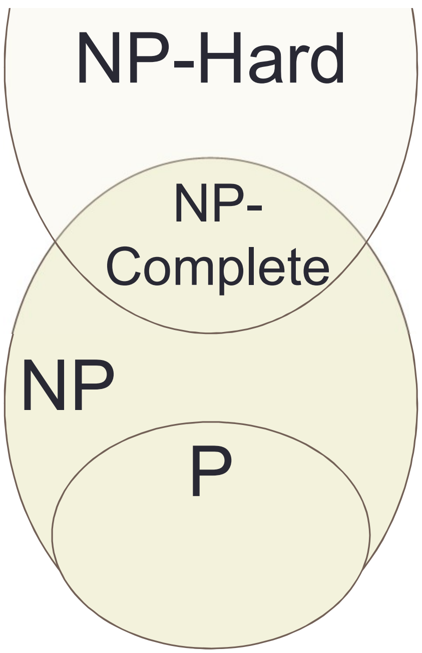
(Hierarchy if P != NP)

NP-Hard: Problems are *at least* as difficult to solve as hardest problems in NP

NP-Complete: No known polynomial time algorithm to find a solution, but can check a solution in polynomial time

A polynomial time solution for *any* NP-Complete problem would solve *all* NP-Complete problems

Complexity Theory

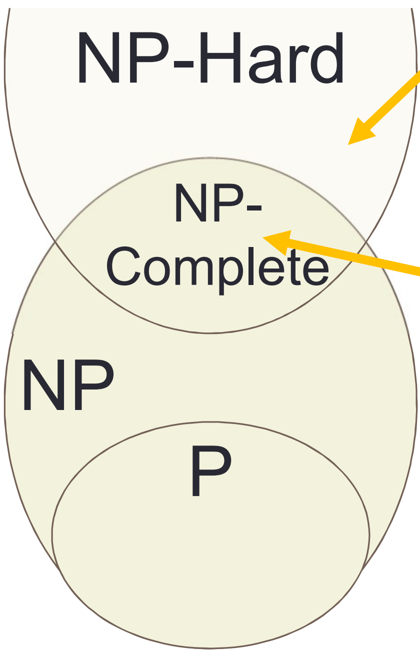


TSP "optimization": given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Where does (do you think) the TSP optimization problem fits into this diagram?

A. In P (there is a polynomial time way to find a solution)
B. In NP/NP-Complete (we might not know a polynomial time way to find a solution, but if someone gives us a proposed solution, we can verify whether or not it's correct)
C. NP-Hard (neither of the above is true)
D. I have no idea! I'm so confused!

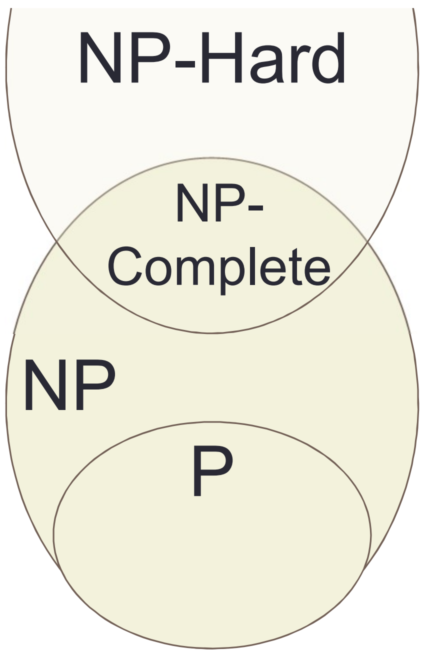
Complexity Theory



TSP "optimization": given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

TSP "decision": given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has a distance less than L.

Complexity Theory



Since TSP (both versions) is NP-Hard, solving it in polynomial time may be difficult (if not impossible)

Next time… how to prove a problem is NP-Hard.