L20 – Graphs (finish MST, Euler, Hamilton)

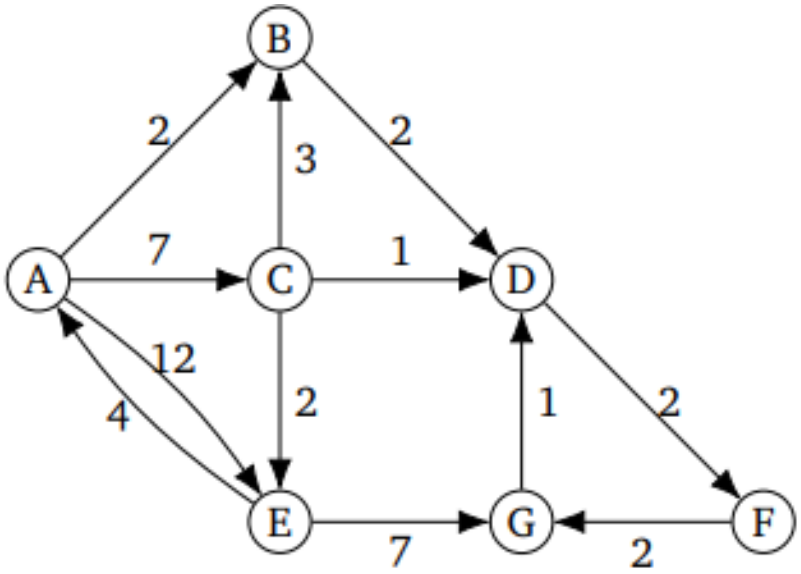
7/24/24

Announcements

- QZ06 tomorrow
  - All graph content L18-L20 (everything)
- EX11 due tomorrow
- Final exam Tuesday, 7/30, 11:30 AM - 2:30 PM in this classroom

Dijkstra's algorithm simulation

- Run Dijkstra's algorithm on this graph starting from A
- Use our column names from yesterday



V	known	dist	prev
A	~~F~~ T	~~∞~~ 0	–
B	~~F~~ T	~~∞~~ 2	A
C	~~F~~ T	~~∞~~ 3	D
D	~~F~~ T	~~∞~~ 1	A
E	~~F~~ T	~~∞~~ ~~4~~ 3	~~∅~~ B
F	~~F~~ T	~~∞~~ ~~9~~ ~~8~~ 6	~~∅~~ ~~C~~ G
G	~~F~~ T	~~∞~~ 5	D

Graph from here

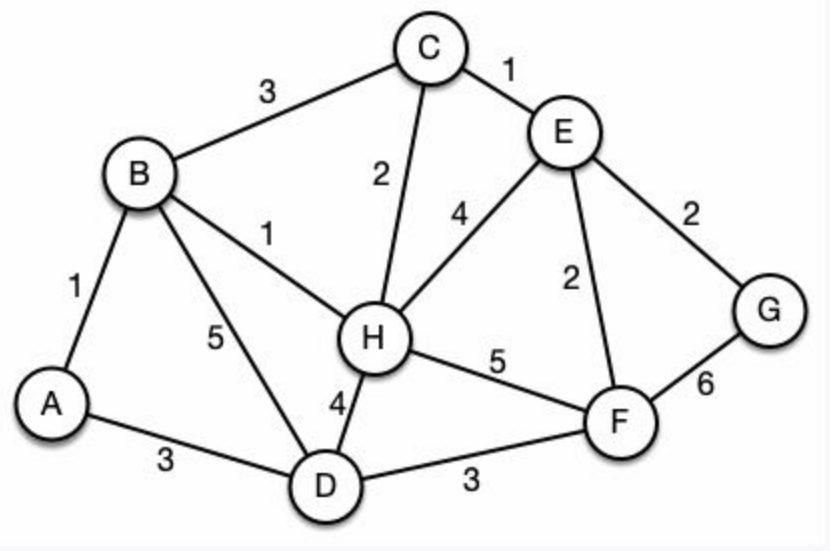
Solution

- PQ
  - (0, A)
  - (2, B) (7, C) (12, E)
  - (4, D) (7, C) (12, E)
  - (6, F) (7, C) (12, E)
  - (7, C) (8, G) (12, E)
  - (8, G) (9, E) (12, E)
  - (9, E) (12, E)
  - (12, E)
  - Empty
- Output: (0, A) (2, B) (4, D) (6, F) (7, C) (8, G) (9, E)

Vertex	Known	Dist	Prev
A	T	0	-
B	T	2	A
C	T	7	A
D	T	4	B
E	T	~~12~~ 9	~~A~~ C
F	T	6	D
G	T	8	F

Kruskal's alg practice problem

1. Given this graph, construct an MST using Kruskal's algorithm.
2. What is the sum of its edge weights?

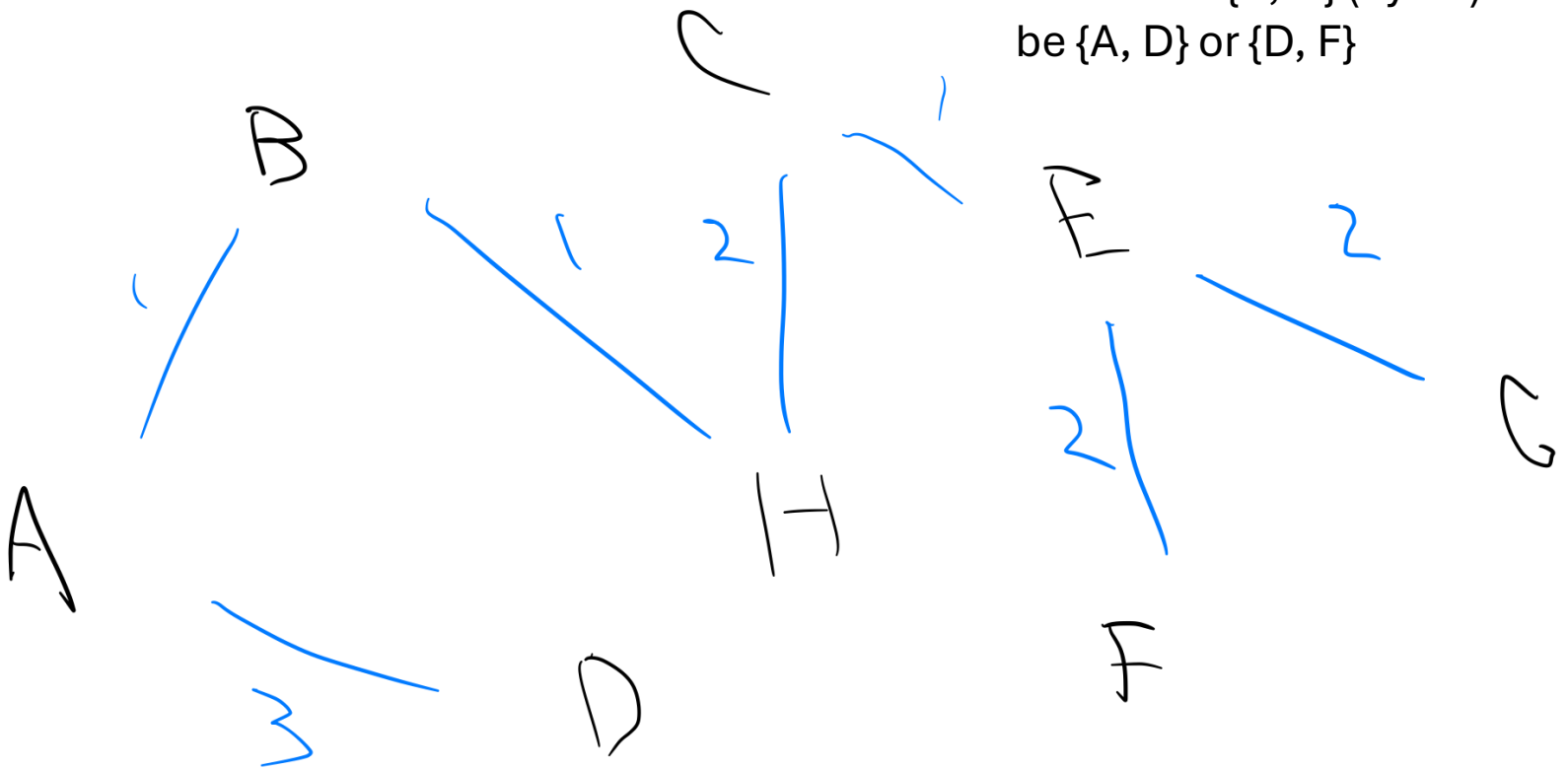


How many MST(s) could Kruskal's algorithm generate for this graph?

(One) solution

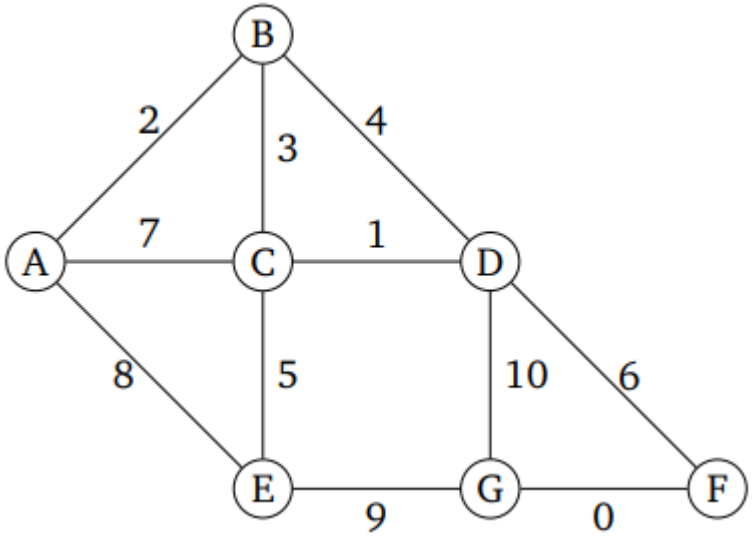
2

We used all the 1 and 2 edges, so the final edge must be a 3-edge. It can't be {B, C} (cycle) but could be {A, D} or {D, F}



Prim's alg practice problem

- Given this graph, construct an MST using Prim's algorithm starting on node A



Edge
(A, B)
(B, C)
(C, D)
(C, E)
(D, F)
(F, G)

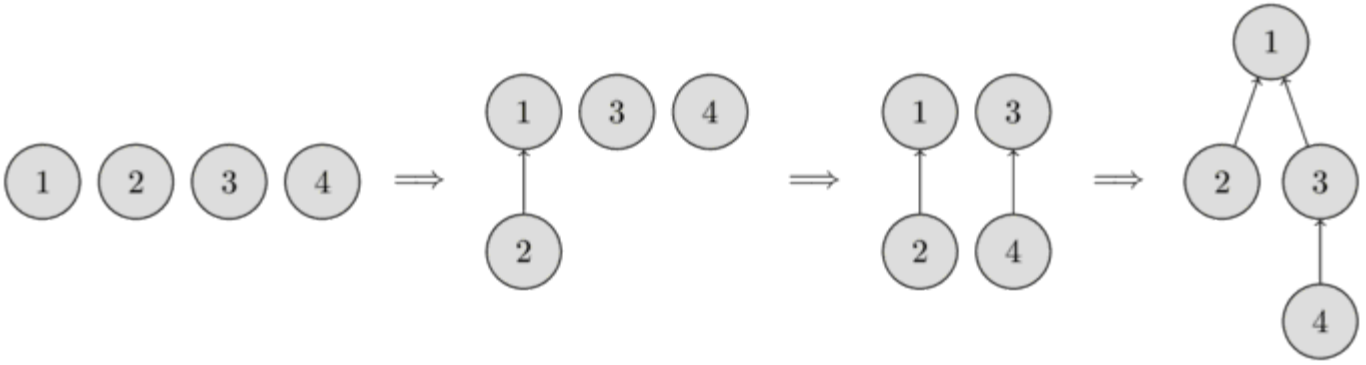
Knowledge questions

1. For this graph, if we had started on a different node with Prim's or if we had used Kruskal's algorithm, would we have gotten the same result?
2. In which algorithm is it simpler (i.e., simpler implementation, not necessarily more efficient) to detect cycles: Kruskal's or Prim's?

    1. Yes, note from yesterday that a graph with distinct weights has a unique MST
    2. Prim's. Note from yesterday that the algorithm simply requires that the destination node not be in the MST already. Implement simply with HashSet/HashMap contains method. Since Kruskal's starts with all nodes, this approach is not possible (use disjoint set)

Disjoint set

- 1, 2, 3, 4 are first each in different disjoint sets
- Then merge 1 and 2 into {1, 2}
  - That set's representative element is 1
- Then merge 3 and 4
  - Leader 3
- Then merge {1, 2} and {3, 4}
  - Leader 1



Disjoint set interface

- `make_set(v)`
  - Create new set consisting of v
- `find_set(v)`
  - Returns representative (or leader) of the set containing v
  - Used to find if two elements are in the same set or not (`find_set(a) == find_set(b)`)
  - Representative can change after `union_sets` call
- `union_sets(a, b)`
  - Merge two sets, the set in which a is located and the set in which b is located
  - If a and b are in the same set, do nothing
- All operations can be done in O(1)* amortized, implemented by array (which can be seen as tree, similar to heap)

Actually O(α(n)), where α(n) is the inverse Ackermann function
α(n) <= 4 for n < 10^{600}, here are the first few entries of F(4, n)

How is disjoint set used in Kruskal's for cycle detection?

Fill in the blanks using find_set(v) and/or union_sets(a, b)

```
algorithm Kruskal(G) is
    F := ∅
    for each v in G.V do
        MAKE-SET(v)
    for each {u, v} in G.E by increasing weight do
        if ________ ≠ ________ then
            F := F ∪ { {u, v} }

            ____________________________
    return F
```

Pseudocode from Wikipedia

Kruskal's alg disjoint set

- Consider the MST made with Kruskal's earlier
- Suppose at this point we are considering the edge (B, C)
- We cannot add it because B and C are in the same set (would create cycle)
- Suppose (C, H) were not in our MST
- Then B and C would be in disjoint sets, so edge (B, C) would be put in the MST

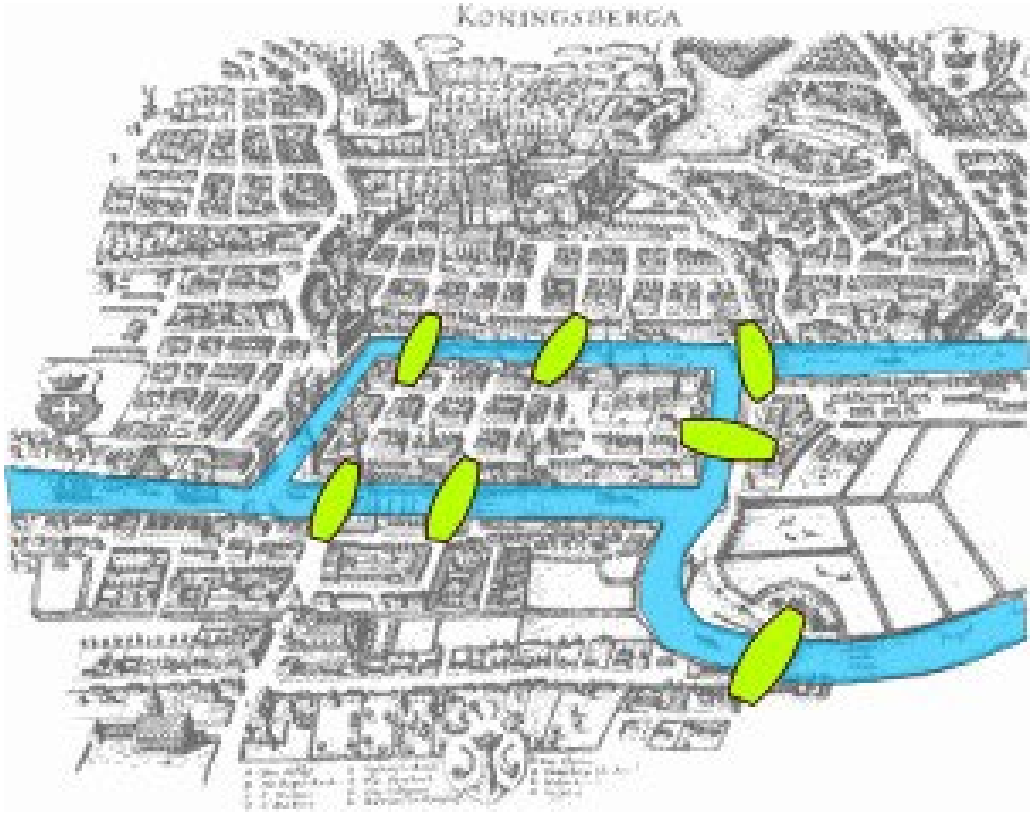
Solution

```
algorithm Kruskal(G) is
    F := ∅
    for each v in G.V do
        MAKE-SET(v)
    for each {u, v} in G.E by increasing weight do
        if FIND-SET(u) ≠ FIND-SET(v) then
            F := F ∪ { {u, v} }
            UNION(FIND-SET(u), FIND-SET(v))
    return F
```

Pseudocode from Wikipedia

7 Bridges of Konigsberg

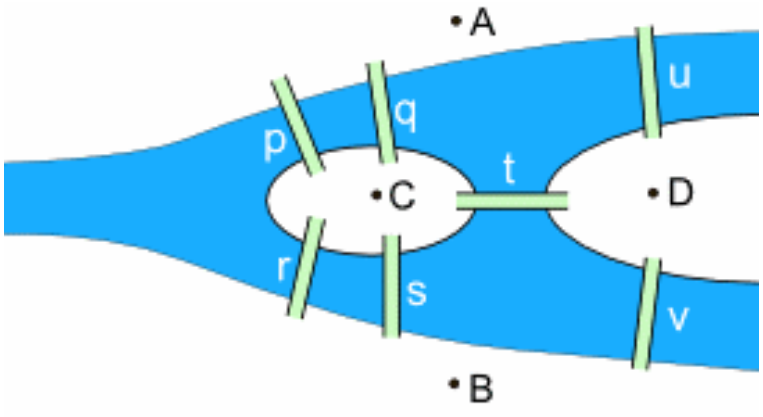
- Can you walk through the city and cross each bridge exactly once (either direction)?



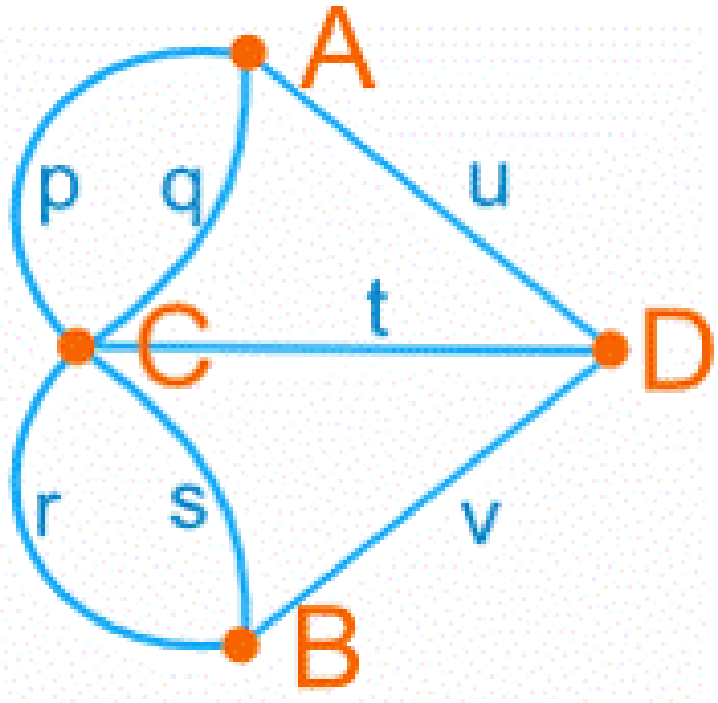
Leonard Euler

- e Euler's number, invented notation $f(x)$, $\sum$, $i = \sqrt{-1}$, etc.
  - People had to stop naming things after him
- Abstracted the 7 Bridges problem and laid the groundwork for graph theory
- Represent land masses as vertices and bridges as edges

Simplified representation



Can a person execute a walk that crosses each bridge once and only once? Can start anywhere and end anywhere

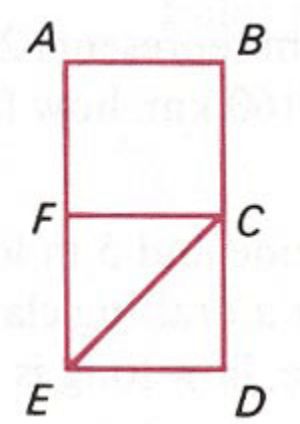


Put a pen on any vertex. Can you trace the diagram without picking up the pen and without redrawing an edge?

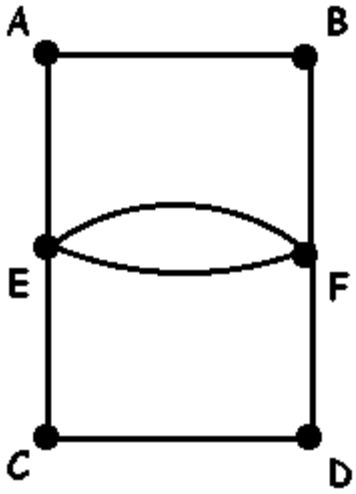
Euler path and circuit

- Euler path (EP)
  - Path in the graph that passes through each edge once
- Euler circuit (EC)
  - EP that begins and ends on the same vertex

EP and EC examples



EP: F, A, B, C, F, E, C, D, E

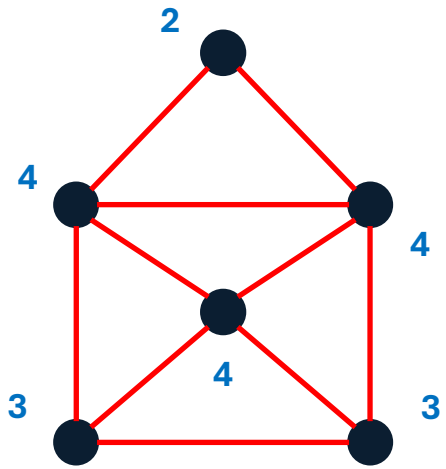


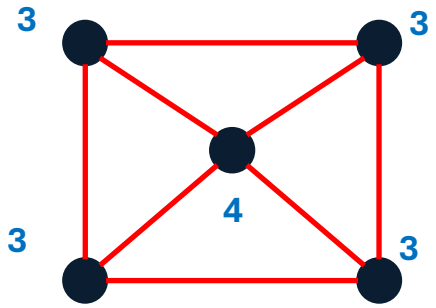
EC: E, A, B, F, E, F, D, C, E

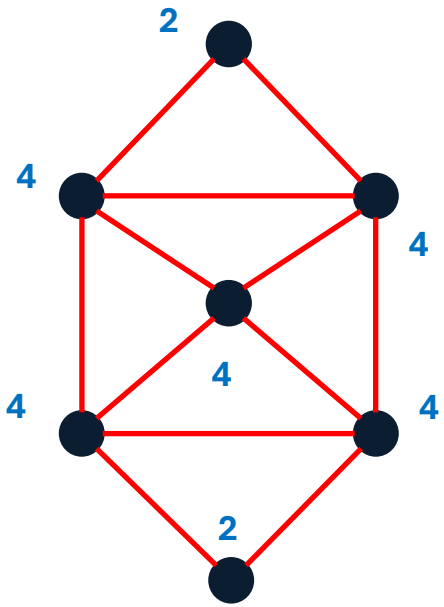
Does G have EP/EC?

- To answer “Does G have any EP or EC?”, first compute degree of every vertex in G
- Degree of vertex v
  - #edges involving v
  - We are talking about undirected graph with unordered edges {a, b}, so deg(v) = number of edges {v, u} ∈ E

Degree examples







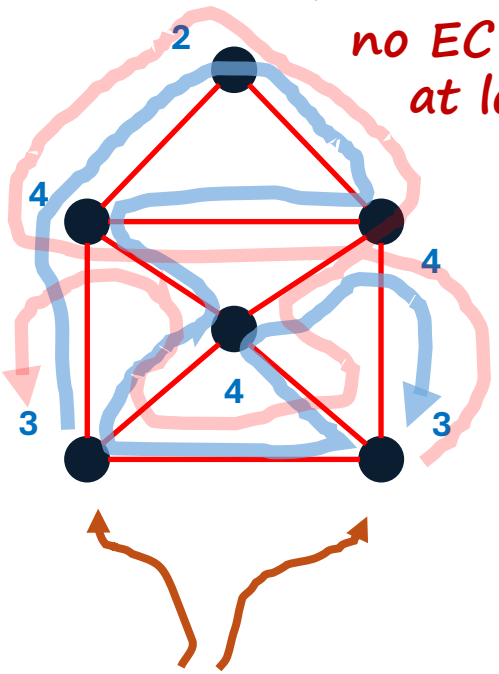
Euler theorems

- Given connected undirected G = (V, E)
- Theorem 1: General
  - The sum of the degrees of all vertices in G exactly 2*|E| (thus, even)
  - Thus, the number of odd vertices is even
- Theorem 2: EP
  - If G has exactly 2 odd vertices, then it has at least one EP, which must start at one odd vertex and end at the other
  - If G has more than 2 odd vertices, then it has no EP
- Theorem 3: EC
  - If G has any odd vertices, then it has no EC
  - If G has all even vertices, then there is at least one EC
- Key idea for theorems 2 and 3
  - Odd vertex degree 1 can be exited but not entered (start node) or entered but not exited (end node)
  - Odd degree ≥ 3 (2n+1), above will hold after you enter and exit that node 2n times
  - 2 odd vertices: one is start and other is end
  - >2 odd: doesn’t work, can’t have more than 1 start and 1 end

odd vertices

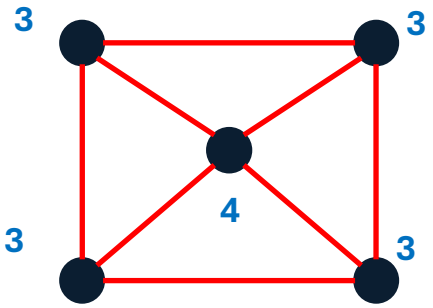
# odd vertices	implication
0	≥1 EC (no EP that is not EC)
1	Impossible (th. 1)
2	No EC, ≥1 EP where every EP starts on one odd and ends on the other
>2	No EC, no EP

Euler theorem examples

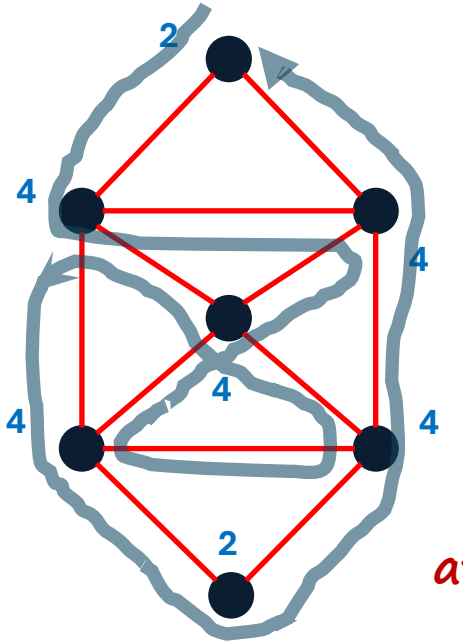


2 odd
no EC
at least 1 EP

they start, end here



4 odd
no EP (so no EC)



0 odd
at least 1 EC

Algorithm for finding EC in O(|E| + |V|)

Won't have you find EP/EC on assessments, will ask conceptual questions like previous slides

Algorithm here for your curiosity

Given undirected unweighted G=(V,E)

1) *Start with empty tmp stack, empty EC stack*

If 0 odd nodes pick any as cn, the curr node

else if 2 odd nodes pick one odd as cn, the curr node

else no EP/EC: end

2) *If cn has no neighbors:*

push cn to EC stack
cn = pop tmp stack

Else // cn has neighbors

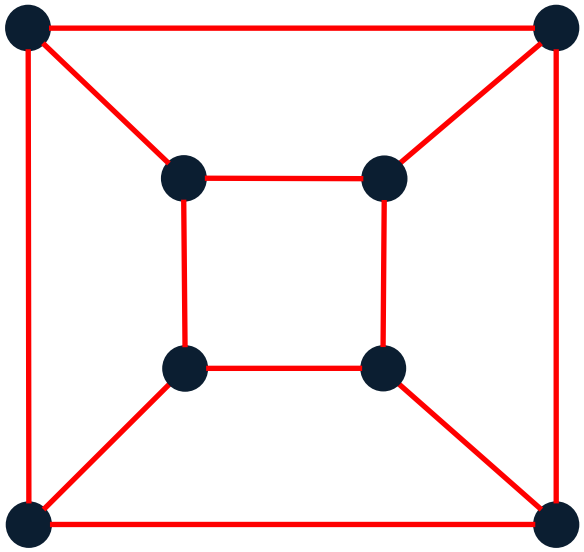
push cn to tmp stack
pick any neighbor nb of cn
remove edge (cn,nb)
make nb the new cn

3) *Repeat step 2 until*

cn has no neighbors and tmp stack is empty

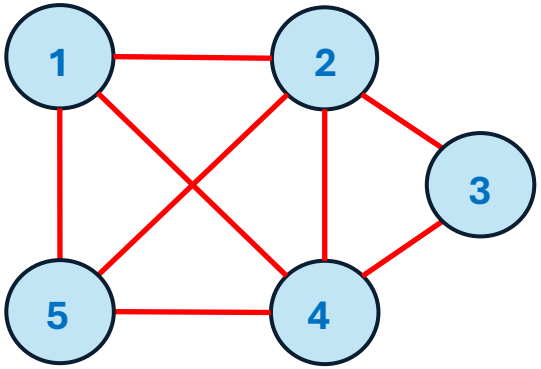
3D (beginning of topology)

- This is a cube “stretched” or flattened to 2D
  - Every convex polyhedron can be projected into 2D as a connected planar graph
- We can use the graph to study some properties of the cube
- E.g., can a bug crawl along all edges of the cube without crossing any twice?
- No
  - #odd vertices = 8, no EP



Hamiltonian path and cycle

- Given undirected connected graph G = (V, E)
- Hamiltonian path (HP)
  - Path in G that contains each **vertex** once
- Hamiltonian circuit (HC)
  - HP with the same start and end vertex



HC: 1, 2, 3, 4, 5, 1

HP: 1, 2, 3, 4, 5

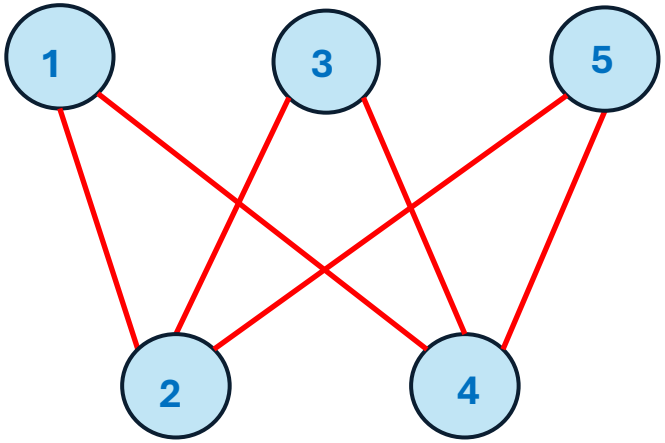
HC: 2, 3, 4, 5, 1, 2

HC: 2, 3, 4, 1, 5, 2

HP: 1, 5, 2, 4, 3

HP: 2, 5, 1, 4, 3

Another example



HP: 1, 2, 3, 4, 5

HP: 1, 4, 3, 2, 5

HP: 1, 4, 5, 2, 3

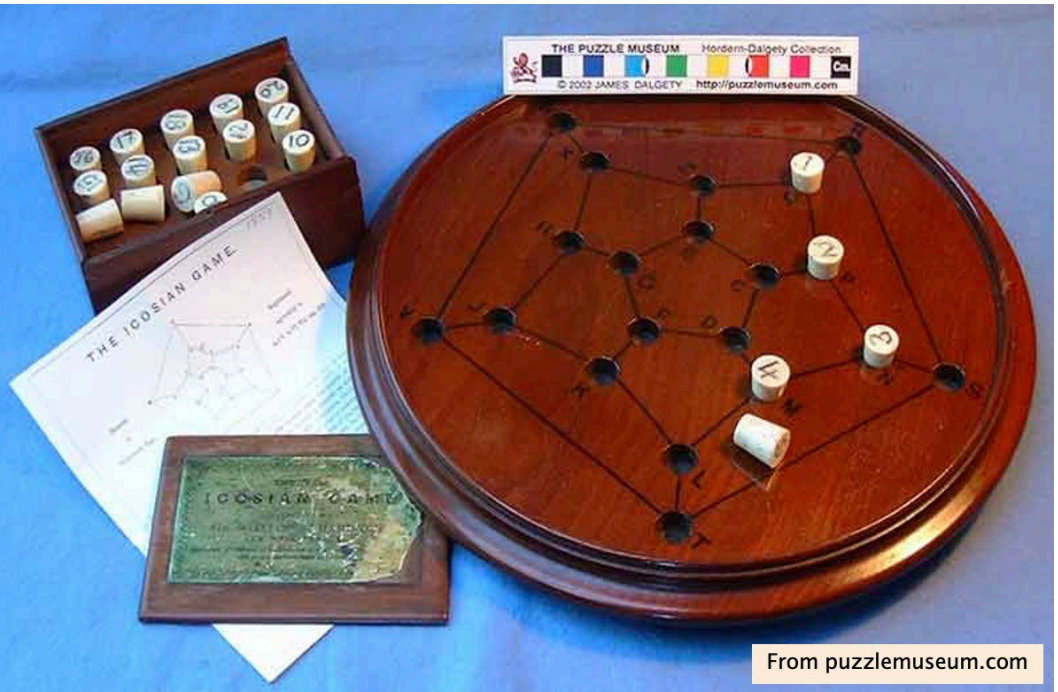
HP: 3, 2, 5, 4, 1

etc.

HC: none

Icosian Game

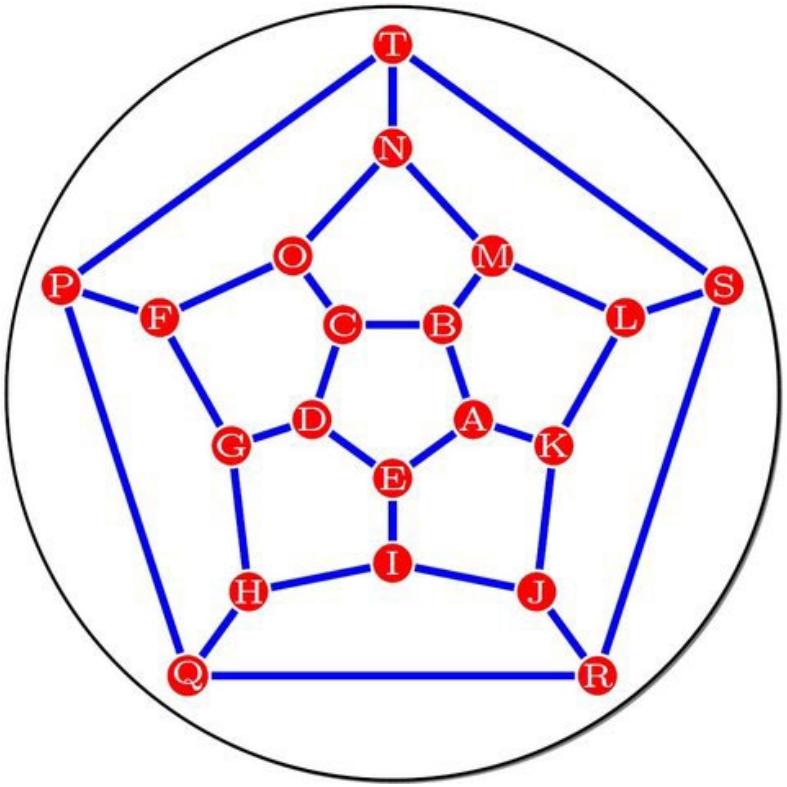
- Board game invented by Irish mathematician William Rowan Hamilton in 1857
- Put pegs in holes to define a path that uses each hole once and ends where it starts (i.e., HC)

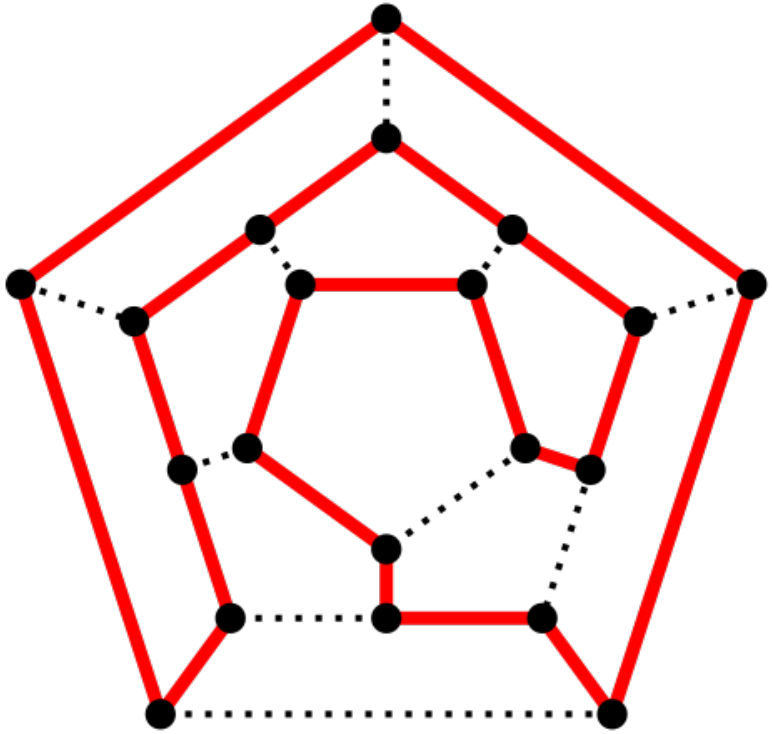


From puzzlemuseum.com

Icosian Game

- Planar graph, flattened dodecahedron
- There is HC for this graph





Hamiltonian harder than Euler

- We saw O(|V| + |E|) algorithm for EP/EC construction
- Hamiltonian not nearly as easy for some strange reason
- Dirac 1952 (sufficient condition)
  - A simple graph G (no loops, no multi-edges) with N vertices is Hamiltonian if degree of each vertex is ≥ N/2
- Ore 1960 (sufficient condition)
  - A graph G with N vertices is Hamiltonian if for every pair of non-adjacent vertices, the sum of their degrees is ≥ N

Efficiency: P vs. NP

- No known efficient (polynomial time) algorithm for finding HP/HC
- For some data, known algorithms for HP/HC take exponential time
- Have not proven that HP/HC is not polynomial, just have not yet found a polynomial time algorithm

NP

- We cannot find HP/HC in polynomial time
- However, given an HP/HC, we can easily (polynomial time) verify that it is an HP/HC for a graph
- See the difference
  - Does this G have an HC?
    - Y/N decision
  - Given this path P in G, is it an HC?
    - Verify
  - Find an HC for G
    - Constructive

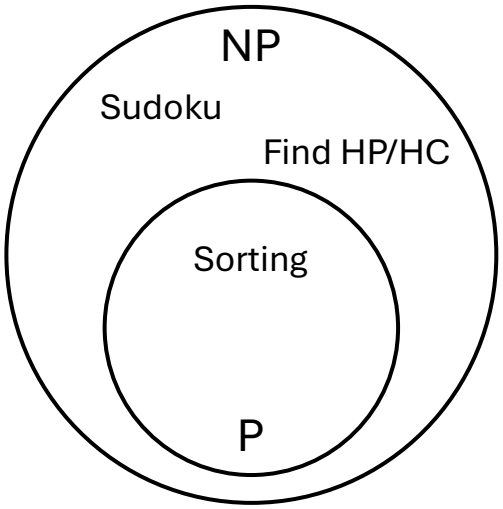
Another example of P and NP

- Sudoku
  - Proposed solutions are easily verified
  - Time to check solution grows polynomially as grid gets bigger
  - Known algorithms for finding solutions take time that grows exponentially as grid gets bigger
  - So, sudoku is in NP (quickly checkable)
  - But seems not to be in P (quickly solvable)

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

$P \stackrel{?}{=} NP$

- Is the set of P questions the same as the set of NP questions?
- If the solution to a problem is easy to check for correctness, must the problem be easy to solve?
- A Millenium Prize Problem ($1,000,000 for solution)
- Proof either way worth way more than $1000000, profound implications for math, cryptography, algorithms, etc.
- We know $P \subseteq NP$ because all problems solvable in P time have solutions verifiable in P time
  - Just solve problem in P and compare it to given solution



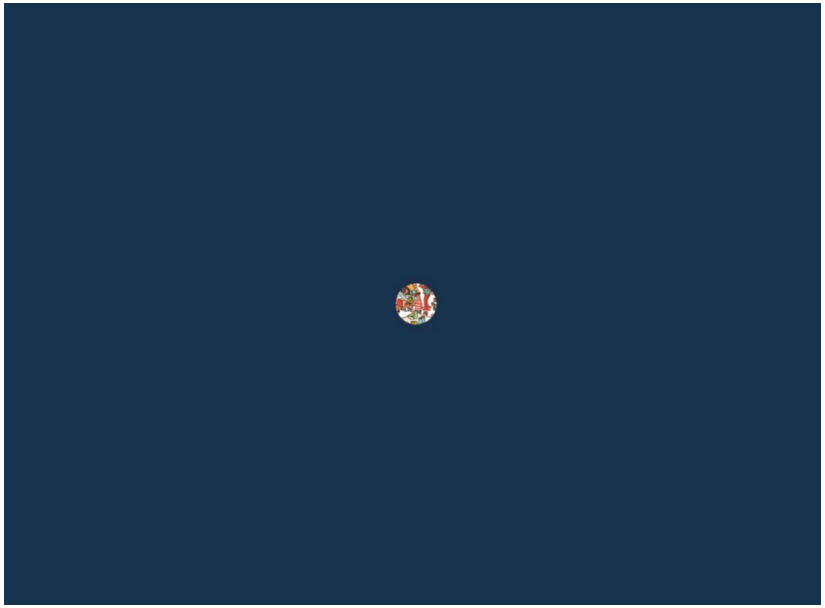
Are P and NP the same circle?

Example crypto application: zero-knowledge (ZK) proof

- Zero-knowledge proof
  - One party (prover) can prove to another party (verifier) that a given statement is true, while avoiding conveying to the verifier any information beyond the mere fact of that statement's truth
- Example usage
  - Prove to a website that you are who you claim to be but without giving them your password

ZK proof example

- How to prove that you know the location of Waldo but without giving away his location?



Shift this part of the board so the verifier can’t see the location

ZK protocol with Hamiltonian

- There is a ZK protocol that involves finding and revealing a Hamiltonian cycle in a graph
- Protocol wouldn't work if we know Hamiltonian is in NP but not P, but it turns out P = NP