Principles of Computer Science II

Introduction to Graph Theory

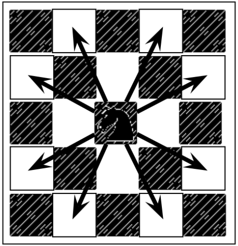
Marco Zecchini

Sapienza University of Rome

Lecture 6

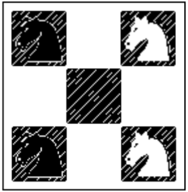
A little bit of Chess

- Knights move using a particular pattern.
- Knights can move two steps in any of four directions (left, right, up, and down) followed by one step in a perpendicular direction,
- Two points are connected by a line if moving from one point to another is a valid knight move.



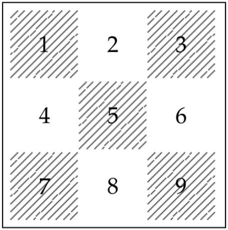
A Chess Puzzle

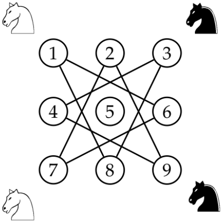
- Two white and two black knights on a 3×3 chessboard.
- Two Knights cannot occupy the same square.
- Starting from the top configuration,
- Can they move, using the usual chess knight's moves, to occupy the bottom configuration?

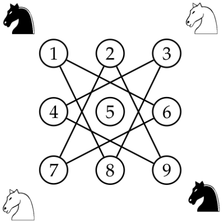


Chess Diagrams

- A Chess Diagram is used to represent movements of chess pieces on the board.
- Example of a 3×3 chessboard.
- Two points are connected by a line if moving from one point to another is a valid knight move.

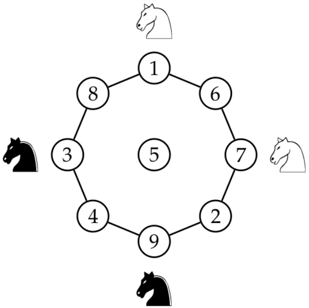


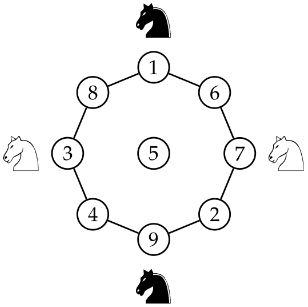




Chess Diagrams – Equivalent Representations

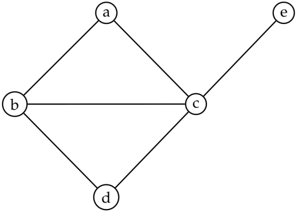
- An equivalent representation of the resulting diagram.
- Now it is easy to see that knights move around a "cycle".
- Every knight's move corresponds to moving to a neighboring point in the diagram – clockwise or counterclockwise
- white-white-black-black cannot be transformed into white-black-white-black

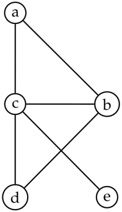




Chess Diagrams & Graphs

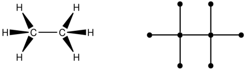
- Chess Diagrams are examples of *graphs*.
- The points are called vertices and lines are called edges.
- A simple graph of five vertices and six edges.
- We denote a graph by $G = G(V, E)$, where
  - V represents the set of vertices
    $V = \{a, b, c, d, e\}$
  - E represents the set of edges
    $E = \{(a, b), (a, c), (b, c), (b, d), (c, d), (c, e)\}$



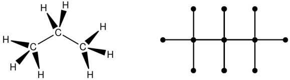


Hydrocarbons as Graphs and Structural Isomers

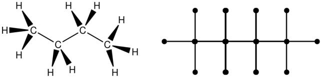
Methane



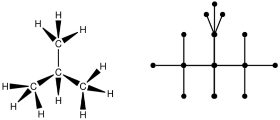
Ethane



Propane



Butane



Isobutane

Basic Definitions

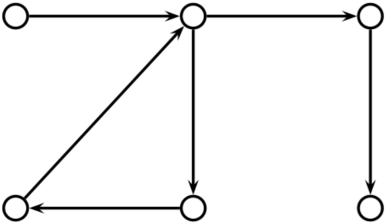
- We denote $|V| = n$ – the number of vertices.
- We denote $|E| = m$ – the number of edges.
- Two vertices u, v are called adjacent or neighboring vertices if there exists an edge $e = (u, v)$.
- We say that edge e is incident to vertices u and v.
- We say that vertices u and v are incident to edge e.
- A loop is an edge from a node to itself: (u, u).
- Two or more edges that have the same endpoints (u, v) are called multiple edges.
- The graph is called simple if it does not have any loops or multiple edges.

Degree of the Vertex

- The number of edges incident to a given vertex v is called the degree of the vertex and is denoted $d(v)$.
- For every graph $G = G(V, E)$, $\sum_{u \in V} d(u) = 2 \cdot |m|$.
- Notice that an edge connecting vertices v and w is counted in the sum twice: first in the term $d(v)$ and again in the term $d(w)$.

Directed & Undirected Graphs

- Many Bioinformatics problems make use of directed graphs.
- An edge can be undirected or directed.
- An undirected edge e is considered an unordered pair, in other words we assume that (u, v) and (v, u) are the same edge.
- A directed edge $e = (u, v)$ and $e' = (v, u)$ are different edges.
- If the edges have a direction, the graph is directed (digraph).
- If a graph has no direction, it is referred as undirected.



Directed Graphs

- In directed graphs, each vertex u has:
  - $indegree(u)$ – the number of incoming edges,
  - $outdegree(u)$ – the number of outgoing edges.
- For every directed graph $G = G(V, E)$,

$$\sum_{u \in V} indegree(u) = \sum_{u \in V} outdegree(u)$$

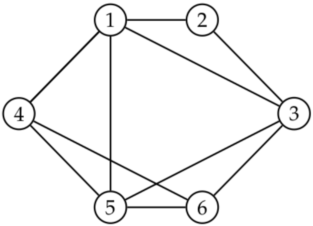
.

Subgraphs & Complete Graphs

- A subgraph G' of G consists of a subset of V and E. That is, $G' = (V', E')$ where $V' \subset V$ and $E' \subset E$.
- A spanning subgraph contains all the nodes of the original graph.
- If all the nodes in a graph are pairwise adjacent, the graph is called complete.

Triangles, Walks, Trails, Paths & Cycles

- A triangle in an undirected graph is a triplet (u, v, w), where $u, v, w \in V$ such that $(u, v), (v, w), (w, u) \in E$.
- A walk is a sequence of vertices and edges of a graph – Vertex can be repeated. Edges can be repeated.
- Trail is a walk in which no edge is repeated.
- Path is a trail in which no vertex is repeated.
- Paths that start and end at the same vertex are referred to as cycles.

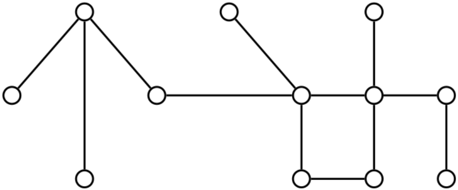


Paths

- A path of length k is a sequence of nodes $(v_0, v_1, \ldots, v_k)$, where we have $(v_i, v_{i+1}) \in E$.
- If $v_i \neq v_j$ for all $0 \leq i < j \leq k$ we call the path simple.
- If $v_0 = v_k$ for all $0 \leq i < j \leq k$ and $v_0 = v_k$ the path is a cycle.
- A path from node u to node v is a path $(v_0, v_1, \ldots, v_k)$ such that $v_0 = u$ and $v_k = v$.

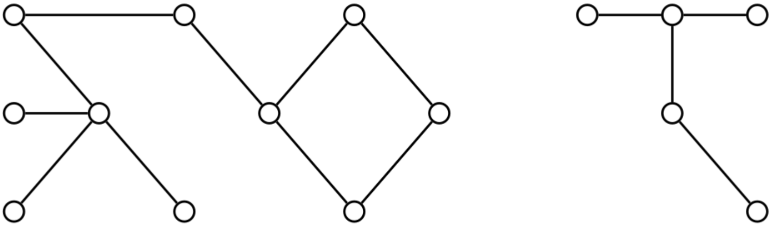
Graph Connectivity

- Two nodes u and v are connected if there is a path from u to v.
- A graph is called connected if all pairs of vertices can be connected by a path, otherwise we say that the graph is disconnected.
- A graph is called complete if there is an edge between every two vertices.



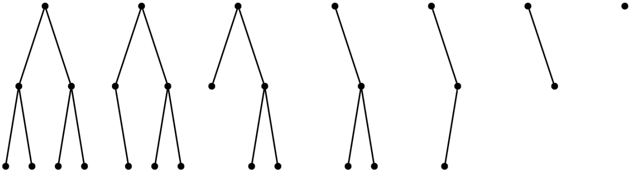
Graph Connectivity

- Disconnected graphs can be decomposed into a set of one or more connected components.



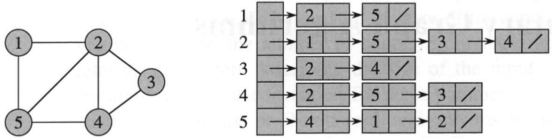
Forests & Trees

- A simple graph that does not contain any cycles is called a forest.
- A forest that is connected is called a tree.
- A tree has $n-1$ edges.
- Any two of the following three statements imply that a graph is a tree (and thus they also imply the third one):
  1. The graph has $n-1$ edges.
  2. The graph does not contain any cycles.
  3. The graph is connected.



Representation of Graphs

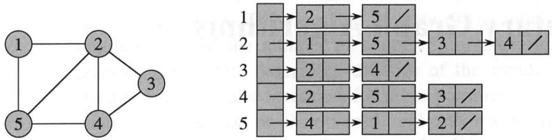
- Two standard ways to represent a graph $G(V, E)$:
  1. A collection of adjacency lists.
     - Usually prefered for sparse graphs.
     - Sparse graph: $|E|$ is much less than $|V|^2$.
  2. An adjacency matrix.
     - Usually prefered for dense graphs.
     - Dense graph: $|E|$ is close to $|V|^2$.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacency List

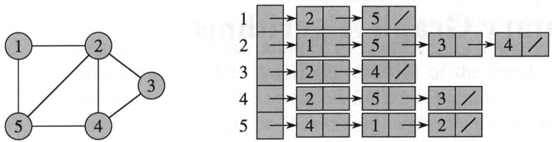
- Adjacency List Representation
- Consists of an array *Adj* of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices adjacent to u in G.
- The vertices are stored in arbitrary order.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacency List

- Adjacency List Representation
- Consists of an array *Adj* of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices adjacent to u in G.
- The vertices are stored in arbitrary order.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

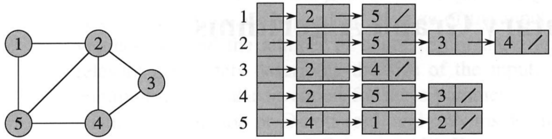
Question?

Does this remind you a data structure we saw last week?

Adjacency Matrix

- Adjacency Matrix Representation of $G(V, E)$
- We assume that vertices are numbered $1, 2, \ldots |V|$.
- The matrix $|V| \times |V|$ matrix.
- $A = (a_{i,j})$, where

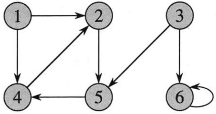
$$a_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E. \\ 0, & \text{otherwise.} \end{cases}$$

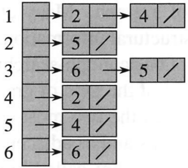


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacency List and Adjacency Matrix Examples

- Adjacency Matrix Representation





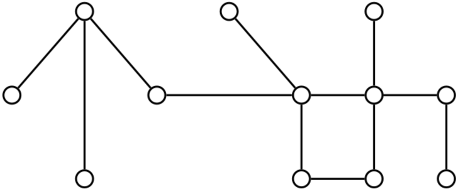
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Paths

- A path is a sequence of vertices and edges of a graph – Vertices cannot be repeated. Edges cannot be repeated.
- A path of length k is a sequence of vertices $(v_0, v_1, \ldots, v_k)$, where we have $(v_i, v_{i+1}) \in E$.
- If $v_i \neq v_j$ for all $0 \leq i < j \leq k$ we call the path simple.
- If $v_0 = v_k$ for all $0 \leq i < j \leq k$ and $v_0 = v_k$ the path is a cycle.
- A path from vertex u to vertex v is a path $(v_0, v_1, \ldots, v_k)$ such that $v_0 = u$ and $v_k = v$.

Shortest Paths

- A shortest path between vertices u and v is a path from u to v of minimum length.
- The distance $d(u, v)$ between vertices u and v is the length of a shortest path between u and v.
- If u and v are in different connected component then $d(u, v) = \infty$.

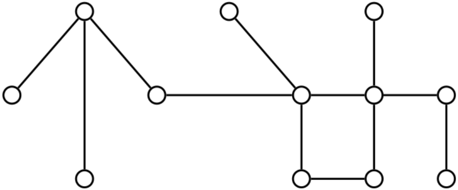


Graph Diameter

- The diameter D of a connected graph is the maximum (over all pairs of vertices in the graph) distance.

$$D = \max_{(u,v):u,v \; connected} d(u,v)$$

- If a graph is disconnected then we define the diameter to be the maximum of the diameters of the connected components.



Breadth-first Search

- Given a graph $G(V, E)$ and a distinguished source vertex u,
- breadth-first search systematically explores the edges of G to "discover" every vertex that is reachable from u.
- It computes the distance from u to each reachable vertex.
- It computes a spanning subgraph of G, the "breadth-first tree", with root u that contains all reachable vertices.
- **It computes all possible shortest paths starting from u:** for any vertex v reachable from u, the path in the breadth-first tree from u to v corresponds to a "shortest path" from u to v in G.
- BFS works with unweighted graphs or graphs where all edges have the same costs.

Example of Execution of Breadth-First Search Algorithm

Initial Graph

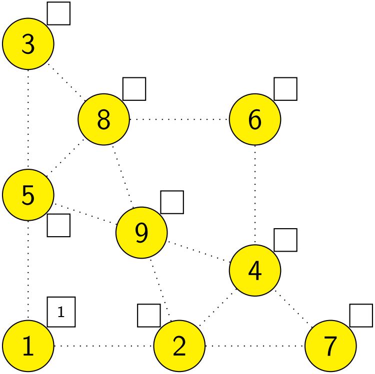
The graph contains 9 vertices, 14 edges

Vertex **1** is the source node.

Vertex **1** marked as discovered.

Vertices **2,5** marked as frontier.

All other vertices are not discovered.



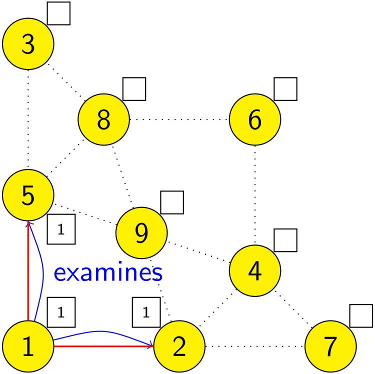
Example of Execution of Breadth-First Search Algorithm

1^{st} Round

Vertex **1** examines adjacent vertices.

Vertice **2,5** marked as discovered.

Vertices **3,4,7,8,9** marked as the frontier.

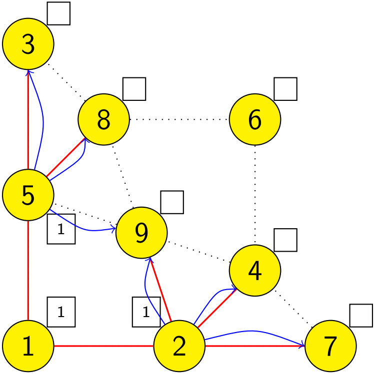


Example of Execution of Breadth-First Search Algorithm

2^{nd} Round

Vertices **3,4,7,8,9** marked as discovered.

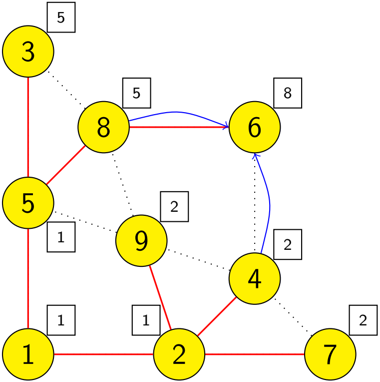
Vertex **6** marked as frontier.



Example of Execution of Breadth-First Search Algorithm

3rd Round

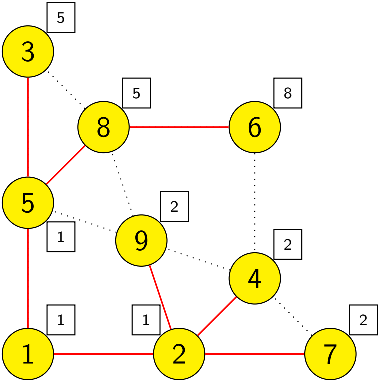
All vertices are discovered.



Example of Execution of Breadth-First Search Algorithm

Final Graph

Breadth-first search tree constructed.



Dijkstra's Algorithm

Goal: Find the shortest paths from a source node to all other nodes in a graph with **non-negative weights**.
Input: A weighted graph $G = (V, E)$ and a source node s.
Output: The minimum distances from the source node to every other node and predecessors to reconstruct the paths.

Main Idea: Iteratively expand nodes based on the currently known minimum distance.

Intuition of the Algorithm

We iteratively execute the following steps:

1. From our current position, we identify all adjacent nodes.
2. Keeping track in a list of the distance to reach each node, we update the distance to reach each node the (the intuition of how to update it will be more clear in few minutes).
3. We move towards the node that has the minimum value in the list of distances.

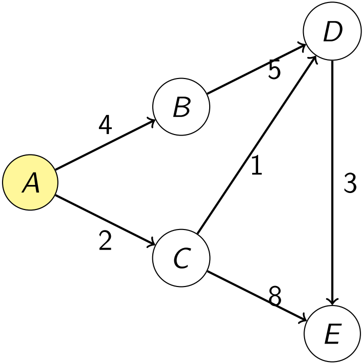
Dijkstra's Algorithm: Initialization

Initialization:

All nodes set to ∞, except the source ($A = 0$).

Initial Distances:

Node	Distance	Predecessor
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-

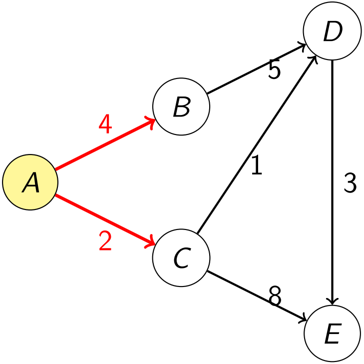


Round 1: Process Node A

Current Node: A (distance 0).
Update distances for neighbors B and C: $d[B] = 4, d[C] = 2$

Predecessors:
$pred[B] = A, pred[C] = A$

Node	Distance	Predecessor
A	0	-
B	4	A
C	2	A
D	∞	-
E	∞	-

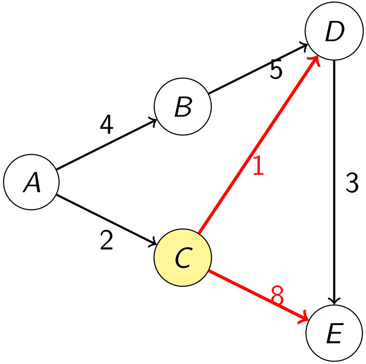


Round 2: Process Node C

Current Node: C (distance 2).
Update distances for neighbors D and E:

$d[D] = 3, d[E] = 10$
Predecessors:
$pred[D] = C, pred[E] = C$

Node	Distance	Predecessor
A	0	-
B	4	A
C	2	A
D	3	C
E	10	C

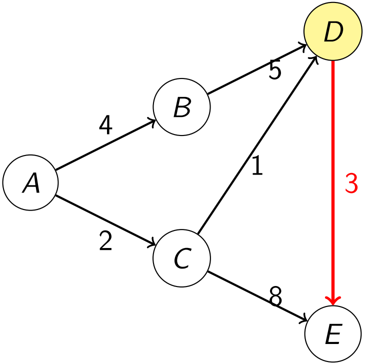


Round 3: Process Node D

Current Node: D (distance 3).
Update distance for neighbor E:

$d[E] = 6$ (updated via D)
Predecessor: $pred[E] = D$

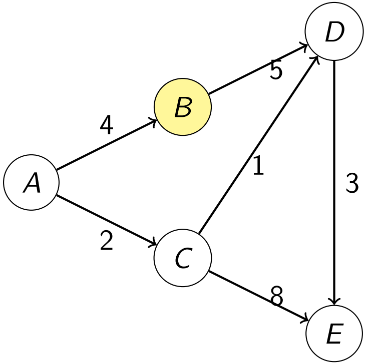
Node	Distance	Predecessor
A	0	-
B	4	A
C	2	A
D	3	C
E	6	D



Round 4: Process Node *B*

Current Node: *B* (distance 4).
No updates are made, as all reachable nodes have shorter paths.

Node	Distance	Predecessor
A	0	-
B	4	A
C	2	A
D	3	C
E	6	D



Final Results

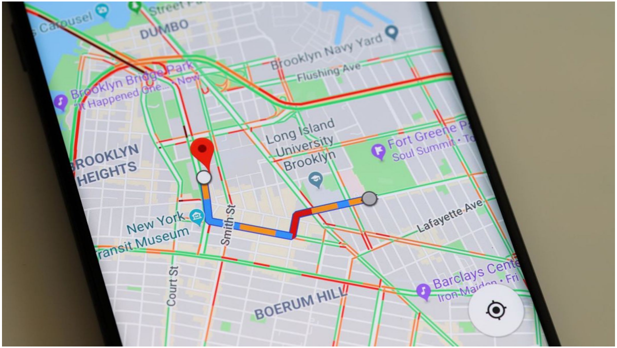
Shortest Paths and Final Distances:

Node	Distance	Predecessor
A	0	-
B	4	A
C	2	A
D	3	C
E	6	D

Pseudocode

1. Initialize the distance of all nodes to ∞, except the source node s (set $d[s] = 0$).
2. Mark all nodes as unvisited.
3. Repeat until all nodes have been visited:
   - Select the unvisited node u with the smallest known distance.
   - Mark u as visited.
   - For each unvisited neighbor v of u:
     - Calculate an alternative distance $alt = d[u] + w(u, v)$.
     - If $alt < d[v]$, update $d[v]$ and set $pred[v] = u$.

Google Maps and Dijkstra



https://youtu.be/Kuyq_HLSPtI?si=wnAliXs3PvO6GytE

Other SP algorithms

There are other algorithms to compute the shortest path in a graph (e.g., Depth First Search).

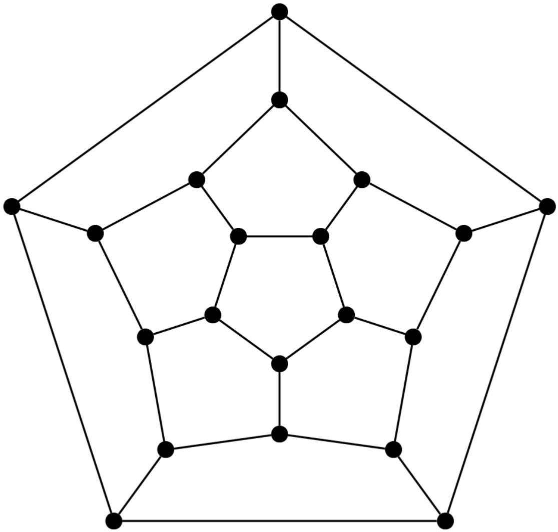
Other SP algorithms

There are other algorithms to compute the shortest path in a graph (e.g., Depth First Search).
And also other problems...

Hamilton's Game

- Sir William Hamilton invented a game corresponding to a graph whose twenty vertices were labeled with the names of twenty famous cities.
- The goal is to visit all twenty cities in such a way that every city is visited exactly once before returning back to the city where the tour started.

Hamilton Path



Hamiltonian Cycle Problem

Hamiltonian Cycle Problem

Find a cycle in a graph that visits every vertex exactly once.

Input: A graph G.

Output: A cycle in G that visits every vertex exactly once.

Algorithms for and Hamiltonian path

- The Hamiltonian path problem is considered an NP-Complete problem (i.e., we don't know an efficient problem to solve it!)

Graphs in Python

Open this Jupyter Notebook and let us see how to create and configure graphs: `https://drive.google.com/file/d/1cRjLIOGOA4nt0ZVwzGhZBJT2yWhXVn9`
`view?usp=sharing`