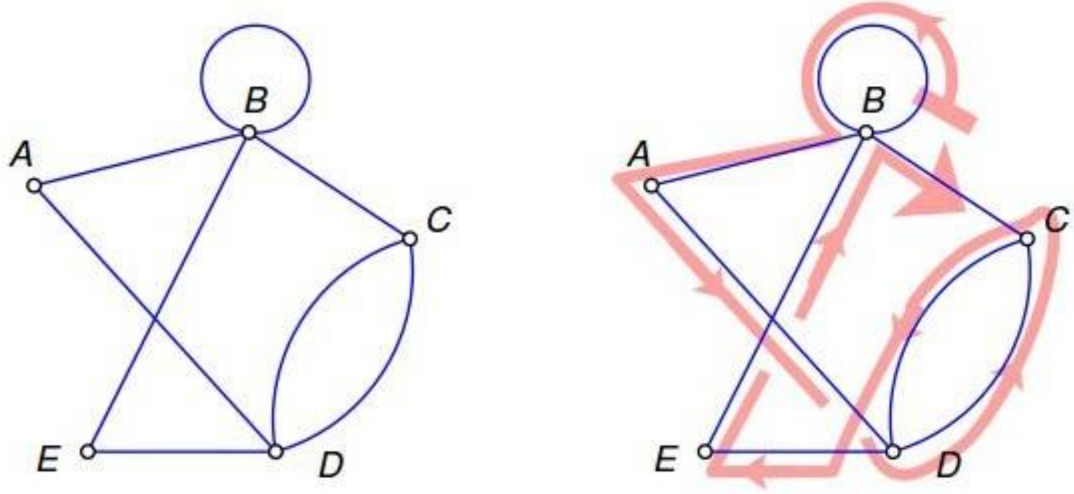
Euler Path

- A path that uses **every** edge of exactly **once**.
  - Path: different start and end
  - Graph can have multiple edges between nodes / self edges

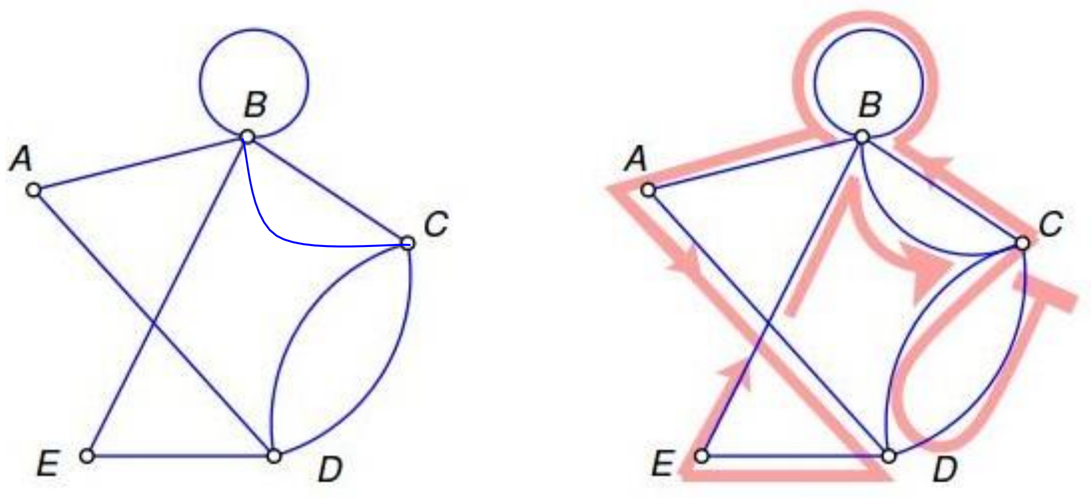


An Euler path: BBADCDEBC

Src: https://www.math.ku.edu/~jmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

Euler Cycle

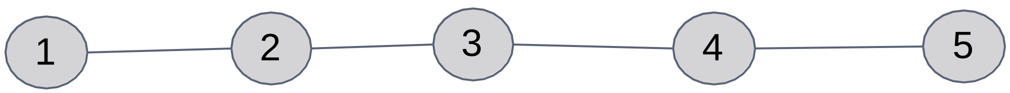
- A cycle that uses **every** edge of exactly **once**.
  - Cycle: start node = end node



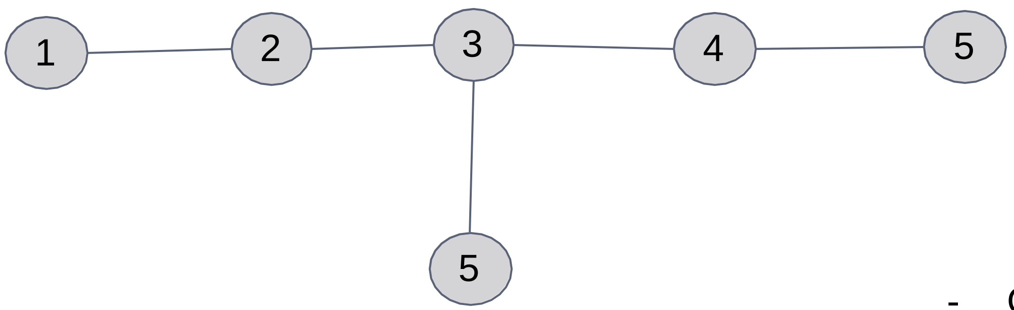
An Euler circuit: CDCBBADEBC

Src: https://www.math.ku.edu/~jmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

Analyzing Euler tour



EulerPath(1, 5) = [1, 2, 3, 4, 5]

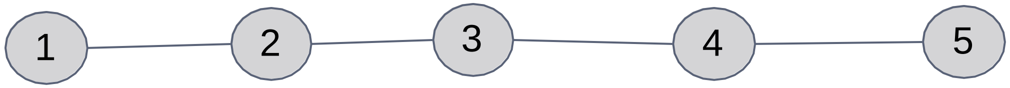


EulerPath(1, 5) = NA..why?

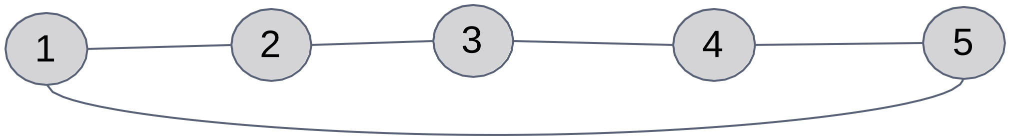
- Go 1, 2, 3
- If we go to 5, we can’t go back to 4
- if we go to 4, 5..we can’t go back to 3

Observation:
Intermediate nodes must have even degrees

Analyzing Euler tour



EulerPath(1, 5) = [1, 2, 3, 4, 5]



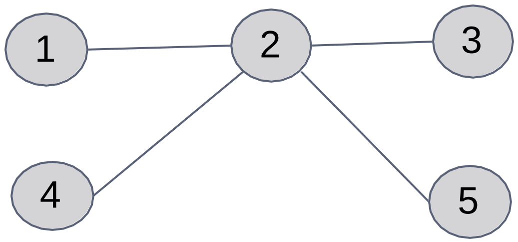
EulerPath(1, 5) = NA..why?

- Go 1, 2, 3, 4, 5
- If you go back to 1, cycle..not path

Observation:

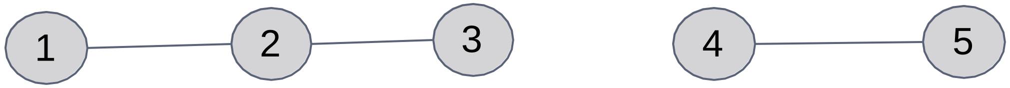
- Euler path: Start/End nodes must have odd degrees, others even degree
- Euler cycle: All nodes must have even degrees

Analyzing Euler tour



EulerPath(????, ???) = NA

We can’t identify start/end



Disconnected graph = NA

Euler Cheat Sheet

# odd vertices	Euler path?	Euler circuit?
0	No	**Yes***
2	**Yes***	No
4, 6, 8, ...	No	No
1, 3, 5,	No such graphs exist	

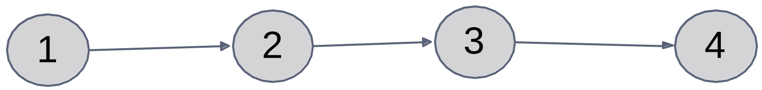
* *Provided the graph is connected.*

Other Facts:

- Every graph has an even number of odd vertices
- 2 * Edges = ∑degree[vi] = Sum of nodes degrees
- We can know if there is a tour without finding it...based only on nodes degrees
- **Coding** Concerns: Multiple Edges - Self Loops - Disconnected Graphs

Src: https://www.math.ku.edu/~jmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

Euler in directed graphs

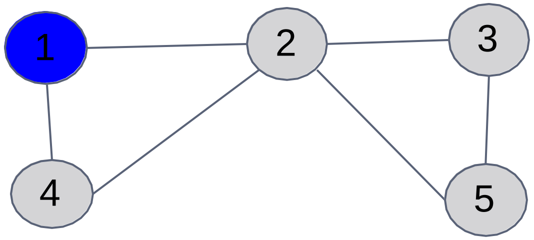


- In Directed Graph
  - node has in-degree and out degree
  - in[1] = 0, out[1] = 1   in[2] = 1, out[2] = 1
- Euler cycle: In-Deg == Out-Deg for all nodes
- Euler path
  - start node: Indeg[i] == outdeg[i]-1
  - end node: Indeg[i] == outdeg[i]+1
  - others: Indeg[i] == outdeg[i]
- Euler in Mixed Graph is more challenging

Hierholzer's algorithm

- Target: Find cycles, and combine them
- Assume there is an euler cycle in G
- What happens if we started from node v, kept following edges from it?
  - We must return to v again, as there is a cycle
  - But not necessarily whole graph is covered in this cycle
- Assume you have graph G with 2 cycles
  - {1, 2, 1} and {2, 3, 2}
  - Can we get the whole graph cycle
  - Yes embed one cycle in the other: {1, **2, 3, 2**, 1}

Hierholzer's algorithm



If started from 1, and keep following edges

- we must return to 1 again
- we can have 2 different cycles:

Possibilities:

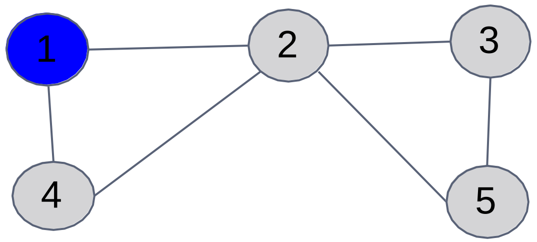
1) {1, 2, 4, 1}
2) {1, 2, 3, 5, 2, 1}

if graph has tour - We must have cycle, but it may not cover whole graph

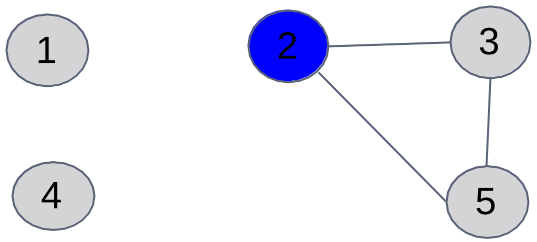
Hierholzer's algorithm

- Start from whatever node v
  - Keep following edges, till back to v
  - Now you have closed tour T
  - T = {v, a, b, c, d, ….v}....remove cycle edges
- Find any node c in T with edge to some node NOT in T
  - Then c has a closed tour starts and ends on it
  - Find c tour…...embed c inside T...do again
  - e.g. {c, q, w, ….c} => T = {v, a, b, **c, q, w, ….c**, d, ….v}
- We can implement it efficiently in O(E)

Hierholzer's algorithm



- Start from 1, find closed tour
- E.g. T = {1, 2, 4, 1}
- Remove T edges
- Find nodes C with edge for node not in T
- Only C = 2



- Start from 2, find closed tour
- E.g. T’ = {2, 3, 5, 2}
- Remove T edges
- Embed in T
- T = {1, **2, 3, 5, 2**, 4}
- Find new C. None. DONE

Hierholzer's algorithm: impl

- Algorithm can be implemented directly
- Find First cycle T
- Identify possible new cycles from T
- Find new cycle and embed in T..etc
- It will be a bit long simple code
  - Iterative Full Code example: method EulerTour

Hierholzer's algorithm: impl

```
// undirected graph: adjMax[i][j] = how many edges between i and j
// adjMax[i][j] = adjMax[j][i]
vector< vector<int> > adjMax;
vector<int> tour;
int n, m;
int start_node;

void find_cycle(int i)
{
    tour.push_back( i );

    if (i == start_node && tour.size() > 1)
        return; // 2nd time..we are done

    lp(j, n)
    {
        if(adjMax[i][j])
        {
            adjMax[i][j]--, adjMax[j][i]--;
            find_cycle(j);
            break;
        }
    }
}
```

Hierholzer's algorithm: impl

- Can we utilize recursion mechanism more:
  - Instead of finding 1 cycle
  - Recursively, Find other cycles and embed them
  - Tricky to think/code...need much tracings
- Define Euler(int i)
  - Either start new cycle and embed it {**i**, a, b, c…..**i**}
  - Or complete previous started cycle
  - It will work for Euler cycle/path

Hierholzer's algorithm: impl

```
// undirected graph: adjMax[i][j] = how many edges between i and j
// adjMax[i][j] = adjMax[j][i]
vector< vector<int> > adjMax;
vector<int> tour;
int n, m;

void euler(int i)
{
    lp(j, n)
    {
        if(adjMax[i][j])
        {
            adjMax[i][j]--, adjMax[j][i]--;
            euler(j);
        }
    }
    tour.push_back( i );
}
```