**Euler Cycle and Euler Path**

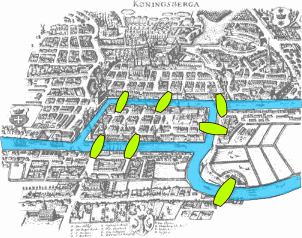
**מעגל ומסלול אוילר**

בתורת הגרפים,

**מסלול אוילר** הוא מסלול (לא בהכרח פשוט) בגרף העובר בכל צלע בדיוק פעם אחת.

**מעגל אוילר** הוא מסלול (לא בהכרח פשוט) אוילר מעגלי (מתחיל ונגמר באותו צומת). גרף המכיל מעגל אוילר נקרא **גרף אוילר או גרף אוילריאני**.

המסלול והמעגל נקראים על שם **לאונרד אוילר** שעסק בהם לראשונה כאשר פתר ב-1735 את בעיית הגשרים של קניגסברג.

[](https://he.wikipedia.org/wiki/%D7%A7%D7%95%D7%91%D7%A5:Konigsberg_bridges.png)העיר קניגסברג (Königsberg) שבפרוסיה המזרחית (כיום קלינינגרד שברוסיה) הייתה מחולקת לארבעה חלקים על ידי הנהר פרגוליה (Pregel). שבעה גשרים חיברו בין ארבעת חלקי העיר. בין תושבי העיר התפתחה מסורת לפיה לא ניתן להלך בעיר ולחצות את כל שבעת הגשרים מבלי לעבור על גשר אחד לפחות יותר מפעם אחת. תושבי העיר ניסו להוכיח או להפריך את השערה זו, אך ללא הצלחה.

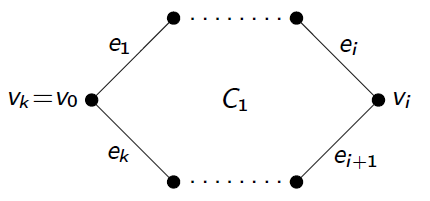
מפת קנינסברג, הנהר והגשרים מודגשים בצבע

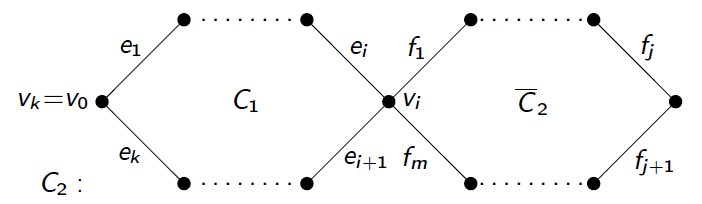
**משפט 1:** יהי G גרף קשיר, לא מכוון שכל דרגותיו זוגיות, ללא קדקודים מבודדים. אז כל קדקוד הגרף שייך למעגל כלשהו (לאו דווקא פשוט).

*הוכחה*: ניקח קדקוד כלשהו v, נצא מהקדקוד v ונטייל בגרף באופן כלשהו, תוך כדי שמירת הכלל שלא לחזור על אותה צלע פעמיים. בגלל שמספר הצלעות בגרף סופי התהליך חייב להסתיים. אם התהליך הסתיים ב- v – קבלנו מעגל. נניח שהתהליך הסתיים בקדקוד x≠v ואיננו יכולים להמשיך. בגלל ש- x≠v, כל מעבר ב- x תורם 2 לדרגה שלx (כניסה ויציאה מהצומת), פרט לצעד האחרון שתרם 1 לדרגה של x. לכן דרגה של x אי-זוגית בסתירה לנתון. לכן x=v וקבלנו מעגל. מש"ל.

**משפט 2:** יהי G גרף ללא קדקודים מבודדים. בגרף G יש מעגל אוילר אם ורק אם הגרף קשיר וכל הדרגות בגרף זוגיות.

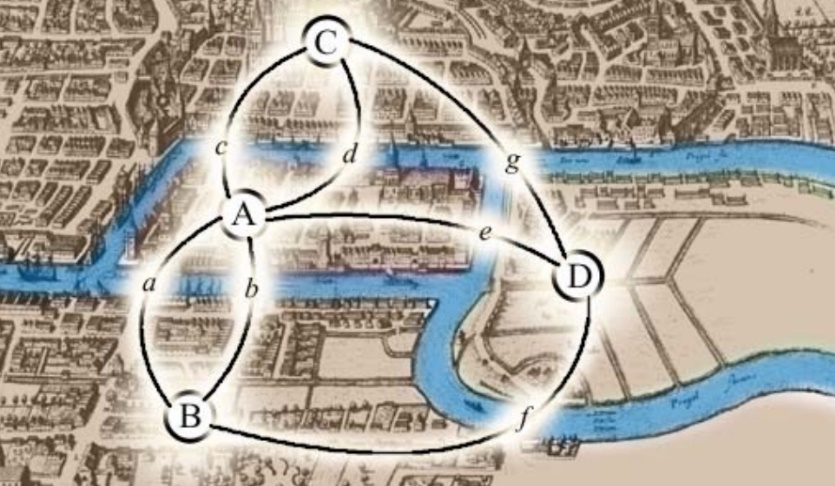
*הוכחה:*

1. *הכרחיות*: יהי G גרף ללא קדקודים מבודדים שיש בו מעגל אוילר. נראה כי גרף קשיר וכל הדרגות בגרף זוגיות. *הוכחה:* כל קדקוד סמוך לפחות לצלע אחת, אז מעגל אוילר עובר על כל קדקודי הגרף. לכן קיים מסלול שמחבר כל שני קדקודי הגרף והמסלול הזה הוא חלק ממעגל אוילר. אזי הגרף קשיר. נוכיח כי כל הדרכות זוגיות . יהי v קדקוד כלשהו בגרף. כל מעבר במעגל אוילר תורם לקודקוד שתי דרגות (צלע אחת בכניסה וצלע אחת ביציאה). בנוסף, מכוון שמדובר על גרף עם מעגל אוילר, אז בכל הצלעות פעם אחת בלבד, לפי כך יספרו כל הצלעות שחלות ב-v בדיוק פעם אחת, לכן דרגה של v היא זוגית.
2. *מספקיות*: יהי G גרף קשיר ולכל הקדקודים דרגה זוגית. נראה כי בגרף קיים מעגל אוילר. *הוכחה*: יהי v0 קדקוד כלשהו בגרף. לפי משפט 1 קיים מעגל ((e1,e2,…,ek C1= כאשר ei – צלע במעגל. אם C1 מעגל אוילר, סיימנו את ההוכחה. 

נניח שב-G יש צלעות שלא שייכות ל- C1, כלומר G – C1≠∅ (הטיול שלנו בגרף הסתיים במעגל שכיסה לפחות חלק מהקשתות, אבל לא בטוח שאת כולן). אז, בגלל שהגרף קשיר, ב- G – C1 קיימת צלע f1∈ G – C1 סמוכה לקדקוד vi כלשהו של C1. נעיר שבגרף G – C1 כל הדרגות זוגיות. אכן, כאשר מוחקים כל הצלעות ej∈C1 מגרף G דרגה של כל קדקוד vj ירדה ב-2. לכן בגרף G1= G – C1 ניתן לבנות מעגל באותו אופן כמו שבנינו ב- C1. אז סדרת הצלעות ((e1,e2,…,ei,f1,…,fm,ei+1,…ek C2= מהווה מעגל כאשר מספר הצלעות שלו גדול ממספר הצלעות שב-C1. 

אם C2 הואמעגל אוילר סיימנו את ההוכחה. אם C2 אינו מעגל אוילר, נבנה באותו אופן מעגל C3, וכו'. בגלל שמספר צלעות בגרף G סופי, באיזשהו שלב אנו נקבל מעגל אוילר. מש"ל.

לפי משפט 2 אנו רואים שלבעיה של גשרים קניגסברג אין פתרון כי כל הדרגות אי-זוגיות:



**משפט 3:** בגרף קשירלא מכוון יש מסלול אוילר אם ורק אםיש בו 0 או 2 קדקודים בעלי דרגה אי-זוגית.

*הוכחה*: נוסיף לגרף עוד צלע שמחברת את הקדקודים בעלי דרגה אי-זוגית. עכשיו כל הדרגות זוגיות, לפי כך אפשר לבנות מעגל אוילר. נוריד ממעגל אוילר את הצלע שהוספנו, נקבל מסלול אוילר. מש"ל

**אלגוריתם למציאת מעגל אוילר**:

בהוכחה של משפט 2 אנו בעצם בנינו אלגוריתם למציאת מעגל אוילר:

מקבלים גרף כרשימת שכנויות.  
לוקחים קדקוד שרירותי v ומוסיפים אותו למחסנית.  
1) כל עוד המחסנית אינה ריקה מבצעים את השלבים 2-8:  
2) בודקים u קדקוד אחרון של מחסנית.  
3) אם הדרגה של u היא 0, מוחקים אותו מהמחסנית ומוסיפים לרשימה הנקראת circle שתכיל את  
 המעגל האוילר.  
4) אחרת (דרגה של u היא לא 0): מוצאים בגרף את השכן הקרוב שלו, w.  
5) מוסיפים את w למחסנית.  
6) מורידים דרגות של w, u ב-1.  
7) מוחקים ב-graph את הקשרים u-v && v-u.  
8) חוזרים לשלב 1.

**Euler Cycle in Graph – Pseudo Code:**

**G – graph as adjacency-list: array of vectors   
deg(v) - degree of vertex v : array of vertex degrees**

**EULER\_CYCLE (G)** v **// arbitrary vertex**  
 Stack st←∅  
 Vector C←∅ **//Euler cycle of vertexes**   
 st.push(v)  
 **while** (st is not empty) **//O(|E|)** v = st.peek() **//returns the object at the top of this stack  
 // without removing it from the stack.**  
 **if** (deg[v]==0) **then**  
 C.add(v)  
 st.pop() **// removes the object at the top of this stack**  
 **else**  
 u = G[v].element[0] **//the first vertex in adjacency-list O(1)** st.push(u)  
 G[v].delete(u) **//delete edge (u,v), O(|V|)**  
 G[u].delete(v) **//delete edge (u,v)**   
 deg[v] = deg[v] - 1  
 deg[u] = deg[u] - 1  
 **end if  
 end while** **return** C  
**end** **EULER\_CYCLE**

**סיבוכיות:** לולאת **while** עוברת על כל צלעות הגרף - פעמים.  
מחיקת צלע אחת מהגרף במקרה הגרוע:   
**סה"כ סיבוכיות:**

**אלגוריתם מהיר יותר שרץ בסיבוכיות :**

<http://algs4.cs.princeton.edu/41graph/EulerianCycle.java.html>

**רעיון האלגוריתם:** למנוע לעבור על צלע כלשהי v-w יותר מפעם אחת כוונה כאשר מתבוננים ברשימת השכניות של קודקוד v, לא לעבור שנית עליה כאשר מתבוננים ברשימת השכניות של קודקוד w.

**מימוש האלגוריתם**: מבוסס על אלגוריתם של חיפוש לעומק (DFS) בגרסה אינטרקטיבית.

// an undirected edge, with a field to indicate whether the edge has already been use*d*

**class** **Edge**

**int** v, w //vertices

**Boolean** isUsed//indicate whether the edge has already been used

**Edge(int v, int w)** // constructor

**this**.v = v

**this**.w = w

isUsed = **false**

**end Edge**

// returns the other vertex of the edge

**public** **int** other(**int** vertex)

**if** (vertex == v) **return** w

**else** **if** (vertex == w) **return** v

**else** **print**("Illegal endpoint")

**end-other**

**end-class-Edge**

**class EulerianCycle**

// Determine whether a graph has an Eulerian cycle

// using necessary and sufficient conditions

// (without computing the cycle itself):

// - at least one edge

// - degree(v) is even for every vertex v

// - the graph is connected (ignoring isolated vertices)

**boolean isEulerian(Vector[] G) //O(V)**

**for** each v in V

**if** (G[v].size() == 0 || G[v].size() % 2 != 0){

System.***out***.println("vertex "+v+" has odd degree!");

**return** **false**

end-if

end-for

**return** **true**

**end-isEulerian**

// create local view of adjacency lists, to iterate one vertex at a time

// the helper Edge data type is used to avoid exploring both copies of an edge *v-w*

**Queue**<Edge>**[] buildEdges(Vector [] G) //O(E)**

nVertices = G.length

nEdges = **numOfEdges(G)**

Queue<Edge>[nVertices] adj

for v = 0 to nVertices-1 **//O(V)**

adj[v] = new Queue(nEdges+1)

end-for

for v = 0 to nVertices-1 **//O(E)**

for (int w : G[v])

if (v < w)

Edge e = new Edge (v, w)

adj[v].add(e);

adj[w].add(e);

end-if

end-for

end-for

return adj

**end-buildEdges**

**numOfEdges(Vector[] G) //O(E)**

numE = 0

**for** i=0 to G.length-1

numE = numE + G[i].size()

end-for

**return** numE/2

**end-numOfEdges**

//Computes an Eulerian cycle in the specified graph, if one exists.

//O(E+V)=O(E)

**Stack<Integer> EulerianCycle (Vector G[], , , s)**

**if (!isEulerian(G)) //O(|V|)**

**print ("not Eulerian")**

**return null**

**end-if**

**nEdges = numOfEdges(G) //O(|E|)**

// adj - local view of adjacency lists, to iterate one vertex at a time

// the helper Edge data type is used to avoid exploring both copies of an //edge v-w

**Queue<Edge>[] adj = buildEdges(G)//O(|E|)**

// cycle is a stack for Eulerian cycle,

// st is a stack for any non-isolated vertex

Stack cycle, st

st.push(s) //s- non-isolated vertex

// greedily search through edges in iterative DFS style

**while** (!st.isEmpty()) **//O(E)**

**int** v = st.pop()

**while** (!adj[v].isEmpty())

Edge edge = adj[v].dequeue() **//O(1) remove**

**if** (!edge.isUsed)

edge.isUsed = **true**

st.push(v) **//O(1)**

v = edge.other(v)

**end-if**

**end-while**

// push vertex with no more leaving edges to cycle

cycle.push(v)

**end-while**

**return cycle**

**end-EulerianCycle**

**end-class-EulerianCycle**