Euler Cycle Algorithm

מעגל אוילר

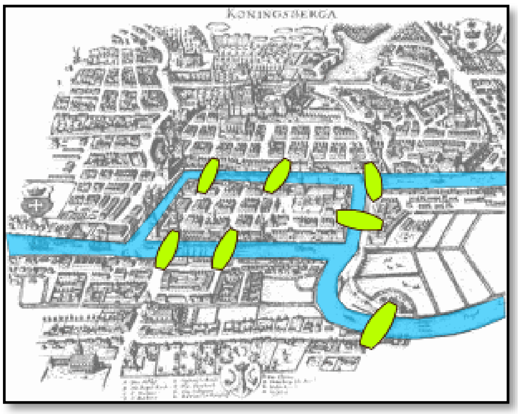
בתורת הגרפים,

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.

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 השערה זו, אך ללא הצלחה.



מפת קנינסברג, הנהר והגשרים מודגשים בצבע

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

הוכחה. היה v קדקוד כלשהו, 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 של x, פרט לצעד האחרון שתרם 1 לדרגה של x. לכן דרגה של x אי-זוגית בסתירה לנתון. לכן x=v וקבלנו מעגל. מש"ל.

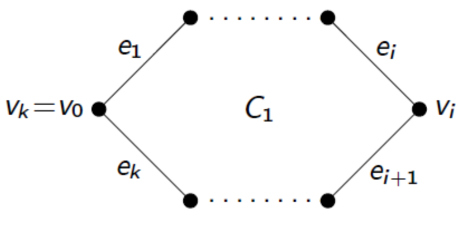
<u>**משפט 2**</u> יהי G גרף ללא קדקודים מבודדים. בגרף G יש מעגל אוילר אם ורק אם הגרף קשיר וכל הדרגות בגרף זוגיות.

הוכחה. הכרחיות. יהי G גרף ללא קדקודים מבודדים שיש בו מעגל אוילר. 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 של מעגל אוילר תורם שתי דרגות (צלע אחת בכניסה וצלע אחת ביציאה). בנוסף, מכוון שזה מעגל אוילר והוא מבקר בכל הצלעות פעם אחת בדיוק, ייספרו כל הצלעות שחלות ב-v בדיוק פעם אחת, לכן דרגה של v היא זוגית.

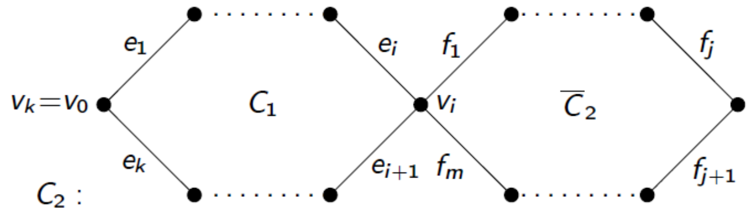
מספקיות. יהי G גרף קשיר ולכל הקדקודים דרגה זוגית. נראה כי בגרף קיים מעגל אוילר.

הוכחה. יהי v_0 קדקוד כלשהו בגרף. לפי משפט 1 קיים מגל $C_1= (e_1,e_2,\ldots,e_k)$. אם C_1 מעגל אוילר, סיימנו את ההוכחה. נניח שב- G יש צלעות שלא שייכות ל- C_1, כלומר $G - C_1 \neq \varnothing$.



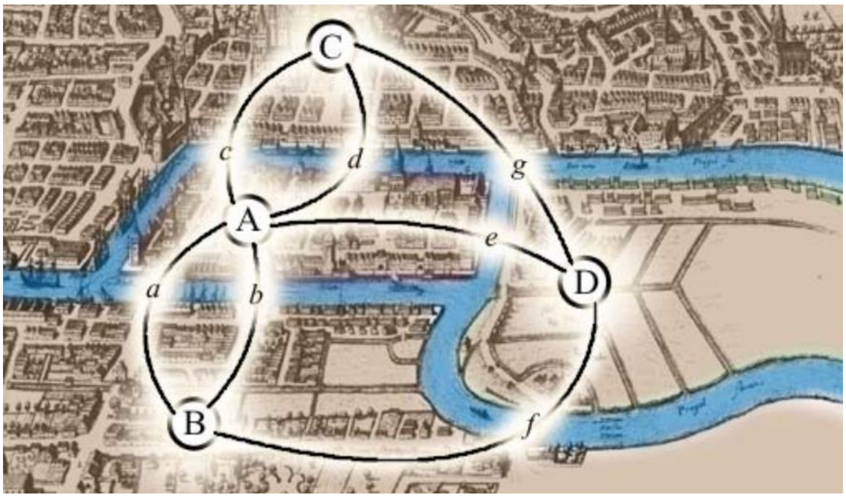
אז, בגלל שהגרף קשיר, ב- $G - C_1$ קיימת צלע $f_1 \in G - C_1$ סמוכה לקדקוד v_i כלשהו של C_1. נעיר שבגרף $G - C_1$ כל הדרגות זוגיות. אכן, כאשר מוחקים כל הצלעות $e_j \in C_1$ מגרף G דרגה של כל קדקוד v_j ירדה ב-2. לכן בגרף $G_1 = G - C_1$ ניתן לבנות מעגל באותו אופן כמו שבנינו C_1. אז סדרת הצלעות

$C_2 = (e_1, e_2, \dots, e_i, f_1, \dots, f_m, e_{i+1}, \dots e_k)$ מהווה מעגל שמספר צלעות שלו גדול ממספר צלעות שב-C_1.



אם C_2 הוא מעגל אוילר סיימנו את ההוכחה. אם C_2 אינו מעגל אוילר , נבנה באותו אופן מעגל C_3, וכו'. בגלל שמספר צלעות בגרף G סופי, באיזשהו שלב אנו נקבל מעגל אוילר. מ"של.

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



<u>משפט 3.</u> בגרף קשיר לא מכוון יש מסלול אוילר אם ורק אם יש 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, ואפשר לבנות מעגל אוילר. נוריד ממעגל אוילר את הצלע שהוספנו אותה, נקבל מסלול אוילר. מש"ל

אלגוריתם למציאת מעגל אוילר

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

מקבלים גרף כרשימת שכנות.
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 ממחסנית ומוסיפים לרשימה הנקראת circle שתכיל את מעגל אוילר.
4) אחרת (דרגה של u היא לא 0): מוצאים בגרף את השכן הקרוב שלו w.
5) מוסיפים את w למחסנית.
6) מורידים דרגות של u, w ב-1.
7) מותקים ב-graph את הקשרים v-u && u-v.
8) חוזרים ל-1.

Euler Cycle in Graph

```
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()
      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 - $|E|$ פעמים.

מחיקת צלע אחת מגרף במקרה הגרוע: $deg[u] + deg[v] \le 2(|V|-1)$

סה"כ סיבוכיות: $O(|E|\cdot|V|)$

אלגוריתם מהיר יותר משתמש בתכונות של JAVA, שרץ בסיבוכיות $O(|E|+|V|) = O(|E|)$.

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

```
class Edge
      final int v, w
      boolean isUsed
      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
```

```
Stack<Integer> EulerianCycle (Vector G[]) //O(E+V)=O(E)
// create local view of adjacency lists, to iterate one vertex at a time
// the helper EdgeP data type is used to avoid exploring both copies of an edge v-w
	Stack<Integer> cycle = new Stack<Integer>()
	int nEdges = numOfEdges(G) //O(E)

	// must have at least one edge and the graph must be Eulerian
	if (nEdges == 0 || !isEulerian(G)) return null

	// create local view of adjacency lists, to iterate one vertex at a time
	// the helper EdgeP data type is used to avoid exploring both copies
	// of an edge v-w
	ArrayBlockingQueue<EdgeP>[] adj = buildEdges(G, nEdges) //O(E)

	// initialize stack with any non-isolated vertex
	int s = nonIsolatedVertex(G) //O(V)
	Stack<Integer> stack = new Stack<Integer>()
	stack.push(s)

	// greedily search through edges in iterative DFS style
	while (!stack.isEmpty())  //O(E)
		int v = stack.pop()
		while (!adj[v].isEmpty())
			EdgeP edge = adj[v].dequeue() //remove
			if (!edge.isUsed)
				edge.isUsed = true;
				stack.push(v);
				v = edge.other(v);
			end-if
		end-while
		// push vertex with no more leaving edges to cycle
		cycle.push(v);
	end-while
	// check if all edges are used
	if (cycle.size() != nEdges + 1) cycle = null
	return cycle
end-EulerianCycle
```

פונקציות עזר:

```
// returns any non-isolated vertex; -1 if no such vertex
private static int nonIsolatedVertex(ArrayList[] G)
      for (int v = 0; v < G.length; v++)
            if (G[v].size() > 0)
                  return v
      return -1
end-nonIsolatedVertex

private static int numOfEdges(ArrayList<Integer>[] G){ //O(E)
      int ans = 0
      for(int i=0; i<G.length; i++)
            ans = ans + G[i].size()

      return ans/2
end-numOfEdges

// necessary condition: all vertices have even degree
// (this test is needed or it might find an Eulerian path instead of cycle)
private static boolean isEulerian(ArrayList<Integer>[] G) {
      for (int v = 0; v < G.length; v++) //O(E)
            if (G[v].size() % 2 != 0)
                  System.out.println("vertex "+v+" has odd degree!")
                  return false
            end-if
      end-for
      return true
end-isEulerian

private static ArrayBlockingQueue<EdgeP>[] buildEdges(ArrayList<Integer>[] G,
                                                       int nEdges)
      int nVertices = G.length
      ArrayBlockingQueue<EdgeP>[] adj = new ArrayBlockingQueue[nVertices]

      for (int v = 0; v < nVertices; v++) //O(V)
            adj[v] = new ArrayBlockingQueue<EdgeP>(nEdges+1)

      for (int v = 0; v < nVertices; v++) //O(E)
            for (int w : G[v])
                  if (v < w) {
                        EdgeP e = new EdgeP(v, w)
                        adj[v].enqueue(e);
                        adj[w].enqueue (e);
                  end-if
            end-for
      end-for
      return adj
end-ArrayBlockingQueue
```