Chap 6. Optimal Graph Traversals

Yih-Lang Li (李毅郎)

Computer Science Department

National Chiao Tung University, Taiwan

The sources of most figure images are from the course slides (Graph Theory) of Prof. Gross

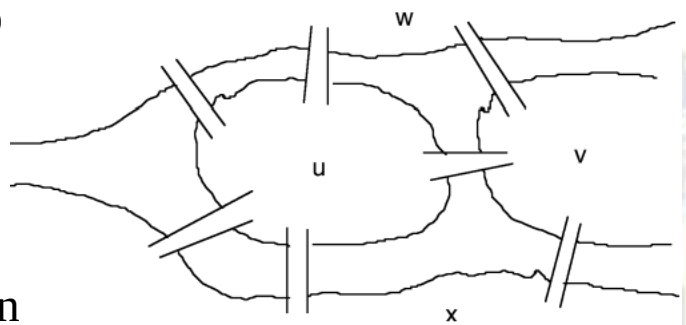
Outline

- Eulerian Trails and Tours
- Debruijn Sequences and Postman Problems
- Hamiltonian Paths and Cycles
- Gray Codes and Traveling Salesman Problems

6.1 Eulerian Trails and Tours

- REVIEW FROM §1.5:
  - ✓ An ***eulerian trail*** in a graph is a trail that contains every edge of that graph.
  - ✓ An ***eulerian tour*** is a closed eulerian trail.
  - ✓ An ***eulerian graph*** is a graph that has an eulerian tour.
- REVIEW FROM §4.5: [**Eulerian-Graph Characterization**] The following statements are equivalent for a connected graph G.

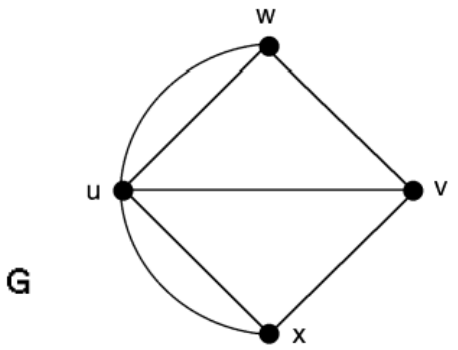
  1. G is eulerian.
  2. The degree of every vertex in G is even.
  3. E_G is the union of the edge-sets of a set of edge-disjoint cycles in G.
- Königsberg was a town of ancient East Prussia. The two branches of the River Pregel converge and flow through to the Baltic Sea. The townspeople wanted to know if it was possible to take a walk that crossed each of the bridges exactly once before returning to the starting point.
  The Prussian emperor, Frederick the Great, brought the problem to the attention of the famous Swiss mathematician Leonhard Euler (1736).



The seven bridges of Königsberg.

Königsberg Bridges Problem

- Königsberg's Graph



Graph representation of Königsberg.

Algo 6.1.1: Eulerian Tour

Input: a conn graph G whose vertices all have even degree.
Output: an eulerian tour T.

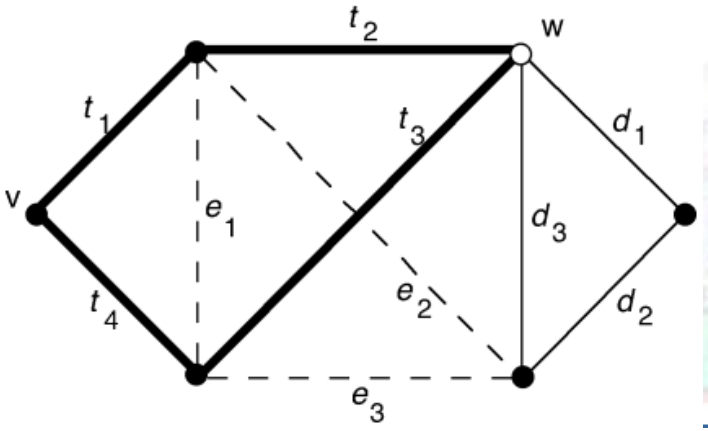
Start at any vertex v, and construct a closed trail T.
While there are edges of G not already in trail T
  Choose a vtx w in T that is endpt of an unused edge.
  Starting at vtx w, construct a closed trail D of unused edges.
  Enlarge trail T by splicing trail D into T at vtx w.
Return T.

By Hierholzer in 1873

- COMPUTATIONAL NOTE: A modified *depth-first search* (§4.2), in which every unused edge remains in the stack, can be used to construct the closed trails.
- **Example 6.1.1:**

$T=<t_1, t_2, t_3, t_4>$, $D=<d_1, d_2, d_3>$

$T'=<t_1, t_2, d_1, d_2, d_3, t_3, t_4>$, $D=<e_1, e_2, e_3>$



Open Eulerian Trails

- **Theorem 6.1.1** A connected graph G has an open eulerian trail if and only if it has exactly two vertices of odd degree. Furthermore, the initial and final vertices of an open eulerian trail must be the two vertices of odd degree.
  - ✓ The concept of the proof is to add a new edge to fulfill the characterization of a Eulerian graph.
  - ✓ ($\Rightarrow$)
    - ➢ Suppose that $\langle x, e_1, v_1, e_2, \ldots, e_m, y \rangle$ is an open eulerian trail in G. Adding a new edge e joining vertices x and y creates a new graph $G^* = G + e$ with an eulerian tour $\langle x, e_1, v_1, e_2, \ldots, e_m, y, e, x \rangle$.
    - ➢ By the eulerian-graph characterization, the degree in the eulerian graph G^* of every vertex must be even, and thus, the degree in graph G of every vertex except x and y is even.
  - ✓ ($\Leftarrow$)
    - ➢ Suppose that x and y are the only two vertices of graph G with odd degree. If e is a new edge joining x and y, then all the vertices of the resulting graph G^* have even degree.
    - ➢ It follows from the eulerian-graph characterization that graph G^* has an eulerian tour T. Hence the trail $T - e$ obtained by deleting edge e from tour T is an open eulerian trail of $G^* - e = G$.

Eulerian Trails in Digraphs

- **Theorem 6.1.2** [**Eulerian-Digraph Characterization**]**.** The following statements are equivalent for a connected digraph *D*.

  1. *Digraph D* is eulerian.

  2. For every vertex *v* in *D*, *indegree*(*v*) = *outdegree*(*v*).

  3. The edge set E_D is the union of the edge-sets of a set of edge-disjoint directed cycles in digraph *D*.

- **Theorem 6.1.3** A connected digraph has an open eulerian trail from vertex *x* to vertex *y* if and only if *indegree*(*x*) + 1 = *outdegree*(*x*), *indegree*(*y*) = *outdegree*(*y*) + 1, and all vertices except *x* and *y* have equal indegree and outdegree.

Tarry's Maze-Searching Algorithm Revisited

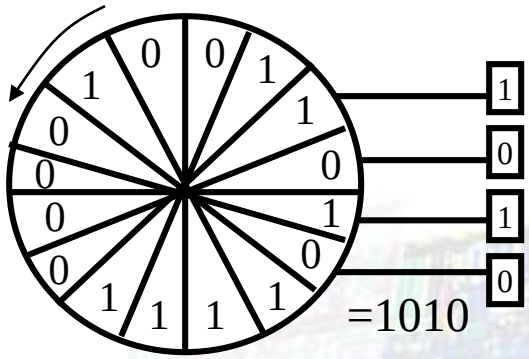
- DEFINITION: A ***double tracing*** is a closed walk that traverses every edge exactly twice. A double tracing is ***bidirectional*** if every edge is used once in each of its two directions.

- **Proposition 6.1.4** Every connected graph has a bidirectional double tracing. In a tree, every double tracing is bidirectional.

6.2 DeBruijn Sequences and Postman Problems

- DEFINITION: A bitstring of length 2^n is called a (2,*n*)-***deBruijn sequence*** if each of the 2^n possible bitstrings of length *n* occurs *exactly once* as a substring, where wraparound is allowed.
- **Example 6.2.1:** (2,1)-deBruijn, (2,2)-deBruijn, (2,3)-deBruijn, (2,4)-deBruijn.

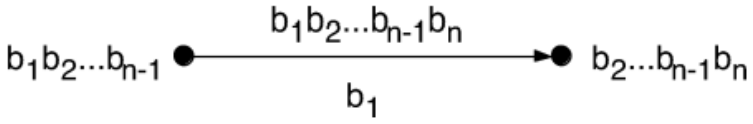
  01 **0110** **01110100** **0000100110101111**
- **Application 6.2.1.** *Identifying the Position of a Rotating Drum*: Suppose that a rotating drum has 16 different sectors. We can assign a 0 or 1 to each of the 16 sectors by putting conducting material in some of the sectors and nonconducting material in others. Sensors can then be placed in a fixed position to "read" a 4-bit string corresponding to the four sectors that are positioned there. For example, in Figure 6.2.1, the sensors read 1010.
  - ✓ It's more space efficient than writing a 4-bit identifying code in each sector.
  - ✓ Generally, (2,*n*)-deBruijn sequences can be constructed for any positive integers.
  - ✓ Also can be extended to (*p*,*n*)-sequences for any positive integers *p* and *n* by finding an Eulerian tour on a digraph.



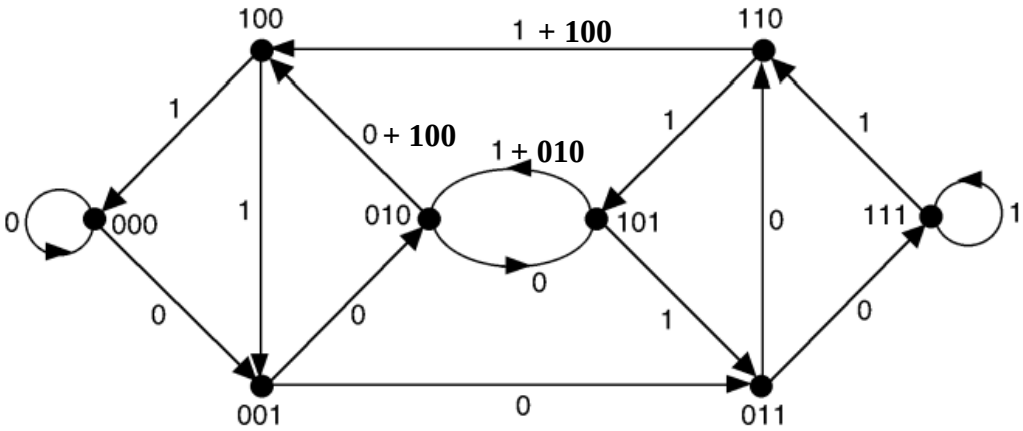
- DEFINITION: The (2,*n*)-***deBruijn sequence*** $D_{2,n}$ consists of 2^{n-1} vertices, labeled by the bitstrings of length *n*-1, and 2^n arcs, labeled by the bitstrings of length *n*. The arc from vertex $b_1b_2\ldots b_{n-1}$ to vertex $b_2\ldots b_{n-1}b_n$ is labeled $b_1b_2\ldots b_n$.

DeBruijn Sequences and Postman Problems

- **Example 6.2.2:**



edge rule: $b_1b_2\ldots b_{n-1} \rightarrow b_2\ldots b_{n-1}0/1$



The $(2,4)$-deBruijn digraph $D_{2,4}$.

- **Proposition 6.2.1.** *The* (2,*n*)*-deBruijn digraph* $D_{2,n}$ *is eulerian.*
  - ✓ **We have to prove (1) $D_{2,n}$ is connected and (2) $\forall v$, *indegree*(v)=*outdegree*(v).**
  - ✓ The deBruijn graph $D_{2,n}$ is strongly connected, since if $a_1a_2\ldots a_{n-1}$ and $b_1b_2\ldots b_{n-1}$ are any two vertices of $D_{2,n}$, then the vertex sequence:

    $$a_1a_2\ldots a_{n-1};\ a_2\ldots a_{n-1}b_1;\ a_3\ldots a_{n-1}b_1b_2;\ \ldots;\ b_1b_2\ldots b_{n-1}$$

    defines a directed trail from $a_1a_2\ldots a_{n-1}$ to $b_1b_2\ldots b_{n-1}$.
  - ✓ Moreover, for every vertex $b_1b_2\ldots b_{n-1}$ of $D_{2,n}$, the only outgoing arcs from $b_1b_2\ldots b_{n-1}$ are $b_1b_2\ldots b_{n-1}0$ and $b_1b_2\ldots b_{n-1}1$, and the only incoming arcs to $b_1b_2\ldots b_{n-1}$ are $0b_1b_2\ldots b_{n-1}$ and $1b_1b_2\ldots b_{n-1}$ .Thus, *indegree*($b_1b_2\ldots b_{n-1}$) = *outdegree*($b_1b_2\ldots b_{n-1}$) = 2, which implies, by Theorem 6.1.2, that $D_{2,n}$ *is eulerian*.

DeBruijn Sequences and Postman Problems

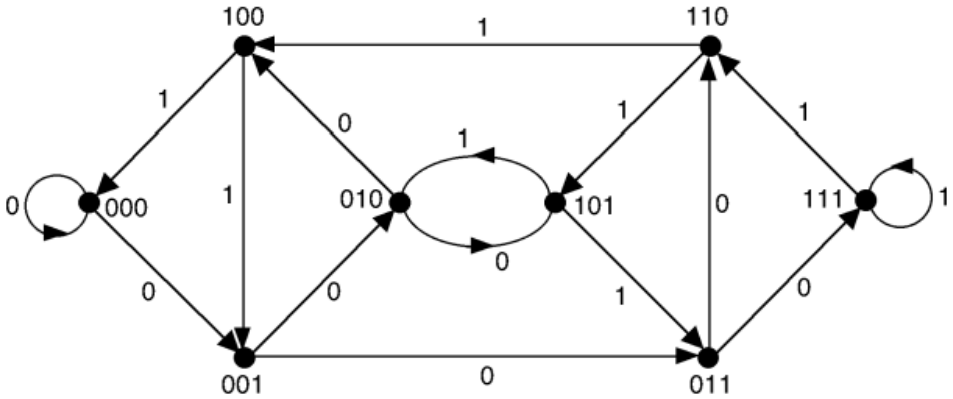
- **Example 6.2.3:**

Algo 6.2.1: Constructing a $(2,n)$-deBruijn Sequence

Input: a positive integer n.
Output: a $(2,n)$-deBruijn sequence S.

Construct the $(2,n)$-deBruijn digraph $D_{2,n}$.
Choose a vertex v.
Construct a directed eulerian tour T of $D_{2,n}$ starting at v.
Initialize sequence S as the empty sequence $\langle\rangle$.
For each arc e in tour T (taken in order of the tour sequence)
  Append the single-bit label of arc e to the right of seq S.

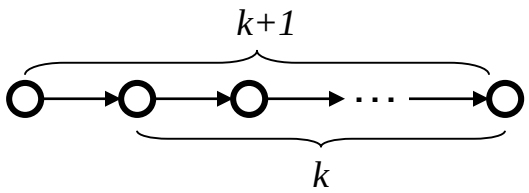
$$000 \xrightarrow{0} 000 \xrightarrow{0} 001 \xrightarrow{0} 010 \xrightarrow{0} 100 \xrightarrow{1} 001$$
$$\xrightarrow{0} 011 \xrightarrow{0} 110 \xrightarrow{1} 101 \xrightarrow{1} 010 \xrightarrow{0} 101 \xrightarrow{1} 011$$
$$\xrightarrow{0} 111 \xrightarrow{1} 111 \xrightarrow{1} 110 \xrightarrow{1} 100 \xrightarrow{1} 000$$
$$\Rightarrow 0000100110101111$$



- **Remark:** Traversing the tour starting from a vertex different from 000 would result in a (2,4)-deBruijn sequence that is merely a *cyclic shift* of the sequence above. A different (2,4)-deBruijn sequence (i.e., one that is not a cyclic shift) can be obtained by constructing a different eulerian tour in $D_{2,4}$ (see Exercises).

DeBruijn Sequences and Postman Problems

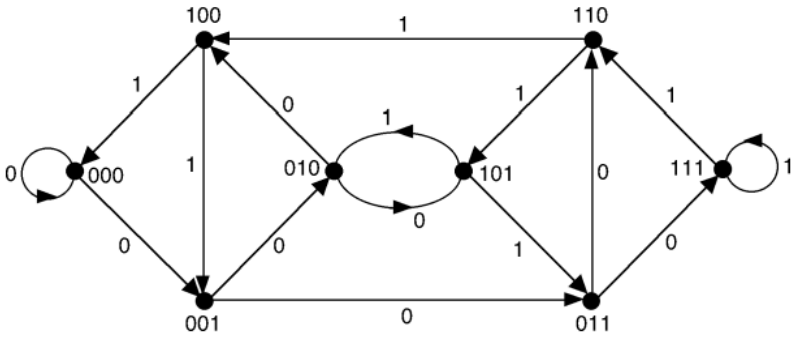
- **Proposition 6.2.2.** *The sequence of single-bit arc labels of any directed trail of length k,* $1 \le k \le n$*, in the* (2,*n*)*-deBruijn digraph, is precisely the string of the leftmost k bits of the full label of the initial arc of that trail.*
  - ✓ The concept of the proof.



  - ✓ The definition of $D_{2,n}$ implies the assertion for $k = 1$. Assume for some k, $1 \le k \le n$-1, that the assertion is true for any directed trail of length k.
  - ✓ let $\langle v_0, v_1, \ldots, v_{k+1} \rangle$ be any directed trail of length k+1. By the induction hypothesis, the single-bit arc labels associated with the subtrail $\langle v_1, v_2, \ldots, v_{k+1} \rangle$ match the first k bits of the full label of the arc from v_1 to v_2.
  - ✓ By the definition of the deBruijn digraph, these k bits are the first k bits of the label in vertex v_1, which means that they are also bits 2 through k+1 of the full label of the arc from v_0 to v_1. But the single-bit label on that arc is the leftmost bit of its full label, which completes the induction step.

- **Example 6.2.4:**

100 $\xrightarrow[1001]{1}$ **001** $\xrightarrow{0}$ **011** $\xrightarrow{0}$ **110** $\xrightarrow{1}$ **101**



Guan's Postman Problem

- Meigu Guan (管梅谷) envisioned a letter carrier who wants to deliver the mail through a network of streets and return to the post office as quickly as possible – ***Chinese Postman Problem***.
- DEFINITION: A ***postman tour*** in a graph G is a closed walk that uses each edge of G at least once.
- DEFINITION: In a weighted graph, an ***optimal postman tour*** is a postman tour whose total edge-weight is a minimum.
- **If every vertex of the graph has even degree, then any eulerian tour is an optimal postman tour.**
- REVIEW FROM §4.7: A ***matching*** in a graph G is a subset M of E_G such that no two edges in M have an endpoint in common.
- DEFINITION: A ***perfect matching*** in a graph is a matching in which every vertex is an endpoint of one of the edges.

Guan's Postman Problem

Algorithm 6.2.2: Optimal Postman Tour

Input: a connected weighted graph G.

Output: an optimal postman tour W.

Find the set S of odd-degree vertices of G.

For each pair of odd-degree vertices u and v in S

  Find a shortest (minimum cost) path P in G between verices u and v.

  Let d_{uv} be the length of path P.

Form a complete graph K on the vertices of S.

For each edge e of the complete graph K

  Assign to edge e the weight d_{uv}, where u and v are the endpoints of e.

Find a perfect matching M in K whose total edge-weight is minimum.

For each edge e in the perfect matching M

  Let P be the corresponding shortest path in G between the endpoints of edge e.

  For each edge f on path P

    Add to graph G a duplicate copy of edge f, including its edge-weight.

Let G^* be the eulerian graph formed by adding to graph G the edge duplications from the previous step.

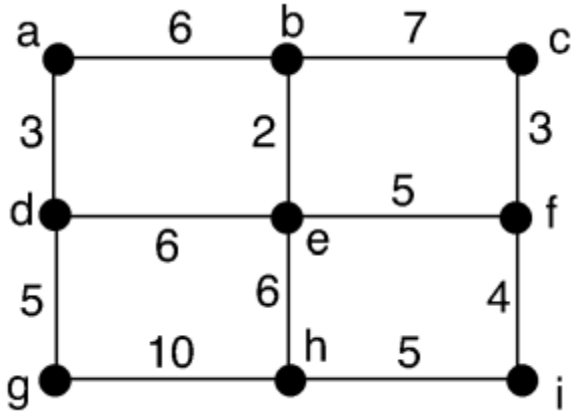
Construct an eulerian tour W in G^*.

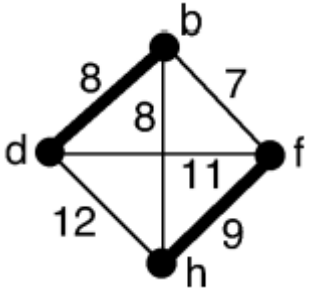
The eulerian tour W corresponds to an optimal postman tour of the original graph G.

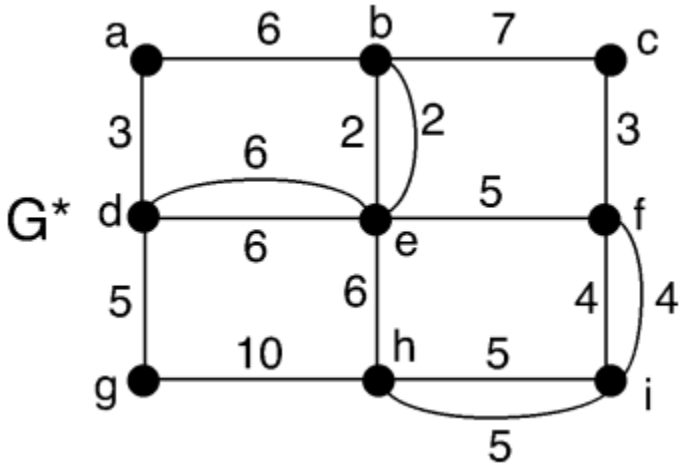
1. Build a complete graph for all odd-degree vertices.
2. Find a minimum edge-weight perfect matching.
3. Duplicate all edges in the shortest path between the endpoints of edges.

Guan's Postman Problem

- **Example 6.2.5:**







$\langle a,\ b,\ c,\ f,\ e,\ b,\ e,\ d,\ e,\ h,\ i,\ f,\ i,\ h,\ g,\ d,\ a \rangle$

Guan's Postman Problem

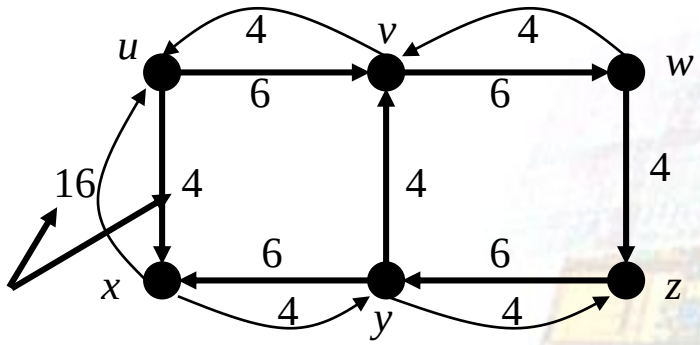
- **Remark:** Edmonds and Johnson's polynomial-time solution applies equally well to the digraph version of the Chinese Postman Problem, but for *mixed graph* (having both undirected and directed edges), the problem becomes NP-hard ([Pa76]). In their same paper, Edmonds and Johnson gave an approximate algorithm for the mixed-graph problem, and G. Frederickson [Fr79] showed that the solution obtained is never worse than twice the optimum. He also proposed some modifications that improved the worst-case performance.

Other Eulerian-Type Applications

- **Application 6.2.2.** *Street Sweeping*: Suppose that the digraph in Figure 6.2.6 below represents a network of one-way streets, with the bold arcs representing streets to be swept. Each edge-weight gives the time required to traverse that street *without* sweeping (i.e., deadheading the street). The time required to sweep a street is estimated as twice the deadheading time. What route minimizes the total time to sweep all the required streets, starting and ending at vertex *z*?

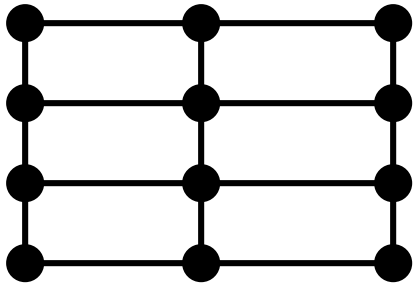
  - ✓ $< z, y, x, y, v, u, v, u, x, y, v, w, z >$
  - ✓ Total sweeping time = 72
  - ✓ Deadheading time = 20

Deadheading time

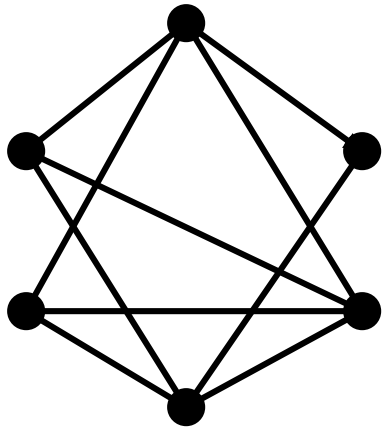


Other Eulerian-Type Applications

- **Application 6.2.3.** *Mechanical Plotters*: Suppose that a Mechanical Plotter is to plot several thousand copies of the grid shown in Figure 6.2.7, and suppose that it takes twice as long to plot a horizontal edge as it takes to plot a vertical edge. The problem of routing the plotter so that the total time is minimized can be modeled as a postman problem [ReTa81]. (See Exercises).



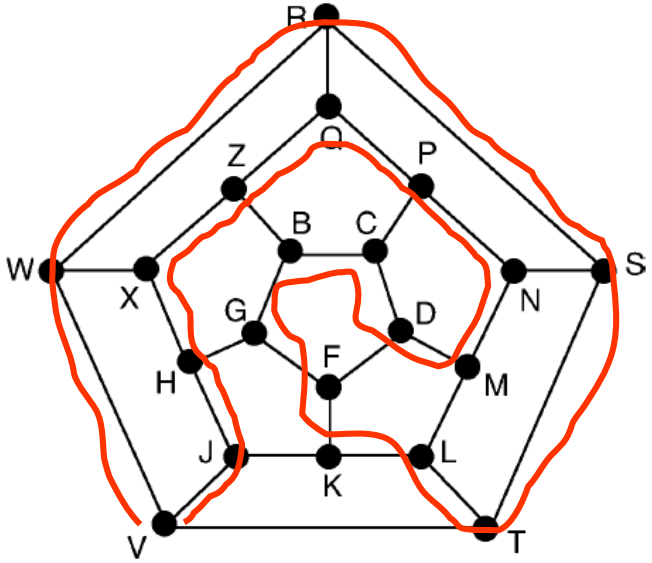
$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} - & 1 & 0 & 1 & 1 & 0 \\ 1 & - & 1 & 1 & 0 & 1 \\ 0 & 1 & - & 0 & 1 & 0 \\ 1 & 1 & 0 & - & 1 & 1 \\ 1 & 0 & 1 & 1 & - & 1 \\ 0 & 1 & 0 & 1 & 1 & - \end{pmatrix} \end{array}$$



- **Application 6.2.4.** *Sequencing Two-Person Conferences*: Suppose certain pairs in a department of six people, $\{A,B,C,D,E,F\}$, must meet privately in a single available conference room. The matrix above indicates with a 1 each pair that must meet. Is it possible to sequence the two-person conferences so that one of the participants in each conference (except the last one) also participates in the next conference, but no one participates in three consecutive conferences?

6.3 Hamiltonian Paths and Cycles

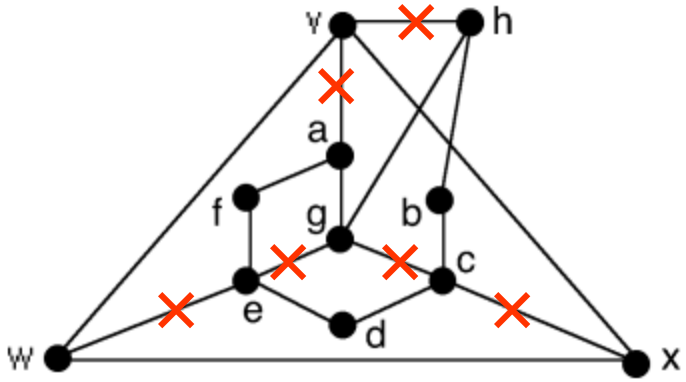
- **DEFINITION**: A ***hamiltonian path*** **(*cycle*)** of a graph (cycle) that contains all the vertices. For digraphs, the hamiltonian path or cycle is directed.
- **DEFINITION**: A ***hamiltonian graph*** is a graph that has a hamiltonian cycle.
- Irish mathematician Sir William Rowan Hamilton was some of the earliest study of hamiltonian graphs.
- A byproduct of his work was the invention of a puzzle known as the *Icosian Game*.
- No characterization such as even degree chcek for Eulerian graph is known, nor is there a quick way of determining whether a given graph is hamiltonian. Actually, it is NP-complete.



Dodecahedral graph for the Icosian Game.

Showing That a Graph Is Not Hamiltonian

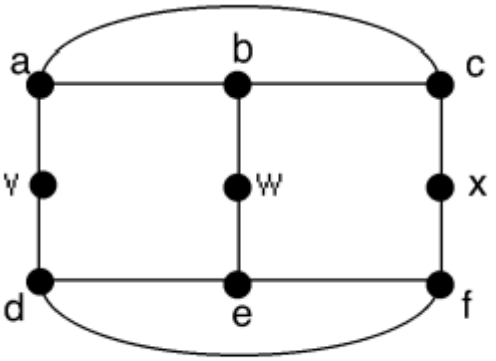
- The following rules are based on the simple observation that any hamiltonian cycle must contain exactly two edges incident on each vertex.
- The strategy for applying these rules is to begin a construction of a hamiltonian cycle and show at some point during the construction that it is impossible to proceed any further.
  - ✓ If a vertex *v* has degree 2, then both of its incident edges must be part of any hamiltonian cycle.
  - ✓ During the construction of a hamiltonian cycle, no cycle can be formed until all the vertices have been visited.
  - ✓ If during the construction of a hamiltonian cycle two of the edges incident on a vertex *v* are shown to be required, then all other incident edges can be deleted.
- **Example 6.3.1:**
  - ✓ Apply rule 1 to *b*, *d*, *f*
  - ✓ Apply rule 3 to *c*, *e* – remove *gc*, *cx*, *ge*, and *ew*.
  - ✓ Form a cycle, *wv*, *vx*, *xw*, before ending – rule 2



A non-hamiltonian graph.

Showing That a Graph Is Not Hamiltonian

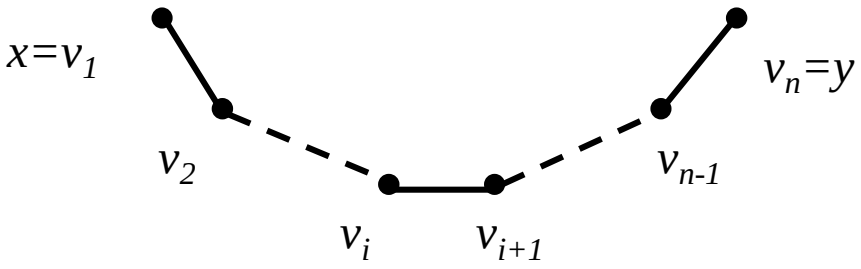
- **Example 6.3.2:**
  - ✓ Apply rule 1 to *v*, *w*, *x*.
  - ✓ We have to select *ab* or *bc* to fulfill the the need of *b*.
  - ✓ But if we select *ab*, then *c* can not be selected in hamiltonian cycle.
  - ✓ Similarly for selecting *bc*.

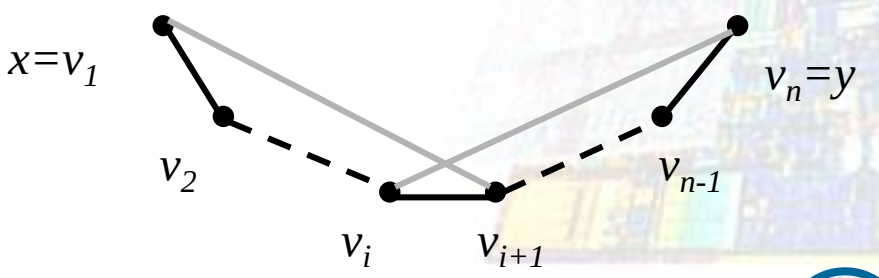


Another non-hamiltonian graph.

Sufficient Conditions for a Graph to be Hamiltonian

- **Theorem 6.3.1 [Ore, 1960].** *Let G be a simple n-vertex graph, where $n \geq 3$, such that $deg(x) + deg(y) \geq n$ for each pair of non-adjacent vertices x and y. Then G is hamiltonian.*
  - ✓ By way of contradiction, assume that the theorem is false, and let G be a maximal counterexample. That is, G is non-hamiltonian and satisfies the conditions of the theorem, and the addition of any edge joining two non-adjacent vertices of G results in a hamiltonian graph.
  - ✓ Let x and y be two non-adjacent vertices of G. To reach a contradiction, it suffices to show that $deg(x) + deg(y) \leq n - 1$.
  - ✓ Since graph $G + xy$ contains a hamiltonian cycle, G contains a hamiltonian path whose endpoints are x and y. Let $< x = v_1, v_2, \ldots, v_n = y >$ be such a path.
  - ✓ For each $i = 2, \ldots, n-1$, at least one of the pairs v_1, v_{i+1} and v_i, v_n is non-adjacent, since otherwise, $< v_1, v_2, \ldots, v_i, v_n, v_{n-1}, \ldots, v_{i+1}, v_1 >$ would be a hamiltonian cycle in G . This means that if $(a_{i,j})$ is the adjacent matrix for G, then $a_{1,i+1} + a_{i,n} \leq 1$, for $i = 2, \ldots, n-2$.





Sufficient Conditions for a Graph to be Hamiltonian

Thus,

$$deg(x)+deg(y)=\sum_{i=2}^{n-1}a_{1,i}+\sum_{i=2}^{n-1}a_{i,n}=a_{1,2}+\sum_{i=3}^{n-1}a_{1,i}+\sum_{i=2}^{n-2}a_{i,n}+a_{n-1,n}=1+\sum_{i=2}^{n-2}a_{1,i+1}+\sum_{i=2}^{n-2}a_{i,n}+1$$

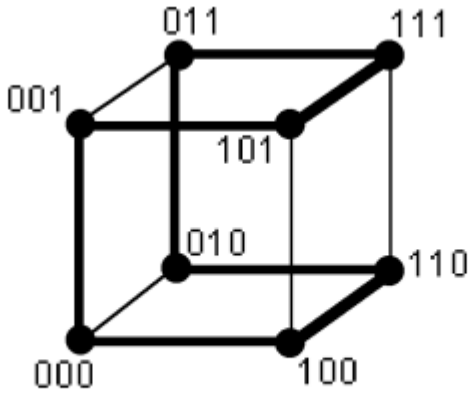
$$=2+\sum_{i=2}^{n-2}(a_{1,i+1}+a_{i,n})\le 2+n-3=n-1$$

which establishes the desired contradiction.

- **Corollary 6.3.2[Dirac, 1952].** *Let G be a simple n-vertex graph, where $n \ge 3$, such that $deg(v) \ge n/2$ for each vertex v. Then G is hamiltonian.*
- **Theorem 6.3.3.** *Let D be a simple n-vertex digraph. Suppose that for every pair of vertices v and w for which there is no arc from v to w, $outdeg(v)+indeg(w) \ge n$. Then D is hamiltonian.*
- **Corollary 6.3.4.** *Let D be a simple n-vertex digraph such that for every vertex v, $outdeg(v) \ge n/2$ and $index(v) \ge n/2$. Then D is hamiltonian.*

6.4 Gray Codes and Traveling Salesman Problems

- The word "code" is no longer referred to as secrecy as digital techniques have been applied routinely to represent analog signals in digital form.
- **DEFINITION**: A ***Gray code of order*** *n* is an ordering of the 2^n length-*n* bitstrings such that consecutive bitstrings (and the first and last bit string) differ in precisely one bit position.
- **Example 6.4.1:** The sequence <000, 100, 110, 010, 011, 111, 101, 001> is a Gray code of order 3.
- **REVIEW FROM Sec.** 1.2: The ***n-dimensional hypercube*** Q_n is the graph whose vertex set is the set of *length-n* bitstrings, such that there is an edge between two vertices if and only if they differ in exactly one bit.
- A Gray code of order *n* corresponds to a hamiltonian cycle in the hypercube graph Q_n.



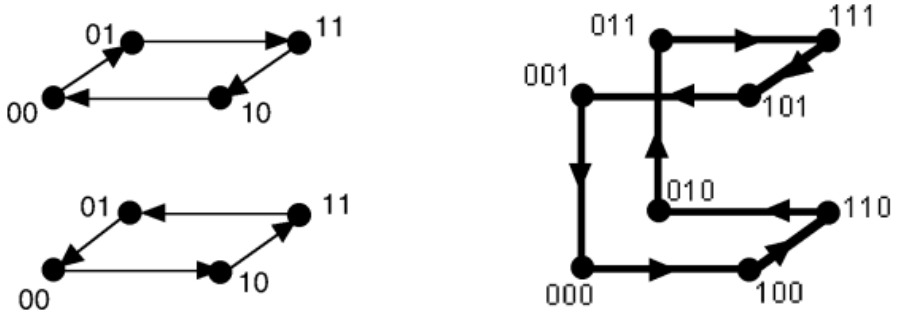
A hamiltonian cycle in the hypercube Q_3.

Gray Codes and Traveling Salesman Problems

- Q_n is hamiltonian for all n, thereby proving that Gray codes of all orders exits.
  - ✓ The $(n+1)$-dimensional hypercube Q_{n+1} can be obtained from two copies of the n-dimensional hypercube Q_n.
  - ✓ Adjoin a 0 to the right/left of each n-bit sequence of one of the copies of Q_n.
  - ✓ Adjoin a 1 to the right/left of each sequence of the other copy.
  - ✓ Join by an edge the two vertices labeled by corresponding sequences.
- *How to obtain a hamiltonian* – traverse one Q_n first, and then go to the second Q_n, traverse the second Q_n, and finally go back to the first Q_n.
- **Theorem 6.4.1.** *The n-dimensional hypercube Q_n is hamiltonian for all $n \geq 2$.*
  - ✓ Q_2 is a 4-cycle and, hence hamiltonian. Assume for some $n \geq 2$ that there exists a hamiltonian cycle $< b_1, b_2, ..., b_{2^n}, b_1 >$ in Q_n. Then

$$< b_1 0, b_2 0, ..., b_{2^n} 0, b_{2^n} 1, b_{2^n - 1} 1, ..., b_1 1, b_1 0 >$$

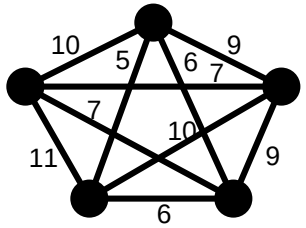
- **Application 6.4.1** *Transmitting Photographs from a Spacecraft – value with single-bit error is very close to true value.*



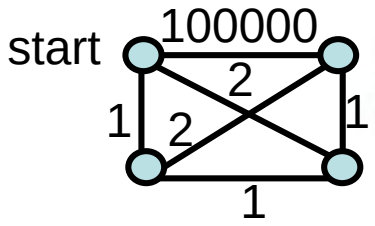
Gray code of order 3 from a Gray code of order 2.

Traveling Salesman Problem

- ***Traveling Salesman Problem* (*TSP*)** is to minimize the total airfare for a traveling salesman who wants to make a tour of *n* cities, visiting each city exactly once before returning home.
- The problem of finding a minimum-weight hamiltonian cycle appears to have been first posed as a TSP by H. Whitney in 1934.
- TSP is a NP-hard problem.
- A provably optimal solution to a 318-city problem.



  - ✓ To evaluate 10^{655} tours at the rate of 1 billion tours per second takes a computer 10^{639} years.
  - ✓ The Crowder-Padberg solution took about 6 minutes by computer.
- **DEFINITION**: A ***heuristic*** is a guideline that helps in choosing from among several possible alternative for a decision step.
- **DEFINITION**: A ***heuristic algorithm*** is an algorithm whose steps are guided by heuristics. In effect, the heuristic algorithm is forfeiting the guarantee of finding the best solution, so that it can terminate quickly.
- ***Algorithm 6.4.1*: Nearest Neighbor**



Traveling Salesman Problem

- **Theorem 6.4.2.** *If there exists a polynomial-time approximate algorithm whose solution to every instance of the general TSP is never worse than some constant r times the optimum, then P = NP.*
- **DEFINITION**: Let G be a weighted simple graph with vertices labeled $1,\ldots,n$, such that edge ij has weight c_{ij}. Then G is said to satisfy the triangle inequality if $c_{ij} \leq c_{ik} + c_{kj}$ for all i, j, and k.
- **Theorem 6.4.3.** *For every $r > 1$, there exists an instance of the TSP, obeying the triangle inequality, for which the solution obtained by the nearest-neighbor algorithm is at least r times the optimal value.*

Algorithm 6.4.2: Double the tree

Input: a weighted complete graph G.

Output: a sequence of vertices and edges that forms a hamiltonian cycle.

Find a minimum spanning tree T^* of G.

Create an eulerian graph H by using two copies of each edge of T^*.

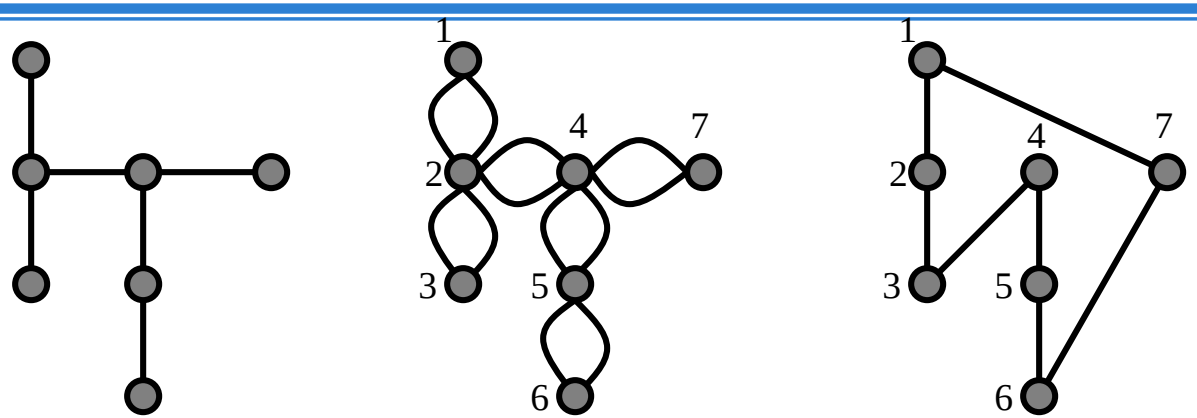
Construct an eulerian tour W of H.

Construct a hamiltonian cycle in G from W as follows:

Follow the sequence of edges and vertices of W until the next edge in the sequence is joined to an already visited vertex. At that point, skip to the next unvisited vertex by taking a shortcut, using an edge that is not part of W. Resume the traversal of W, taking shortcuts whenever necessary, until all the vertices have been visited. Complete the cycle by returning to the starting vertex via the edge joining it to the last vertex.

Two Heuristic Algorithms that Have Performance Guarantees

- **Example 6.4.2:**



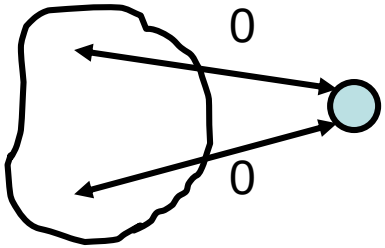
- **Theorem 6.4.4.** *For all instances of the TSP that obey the triangle inequality, the solution produced by Algorithm 6.4.2 is never worse than twice the optimal value.*
  - ✓ Let C be the hamiltonian cycle produced by Algorithm 6.4.2, and let C^* and T^* be a minimum-weight hamiltonian cycle and a minimum-weight spanning tree, respectively.
  - ✓ The total edge-weight of the eulerian tour is $2\times wt(T^*)$, and since each shortcut is an edge that joins the initial and terminal points of a path of length at least 2, the triangle inequality implies that $wt(C)\leq 2\times wt(T^*)$.
  - ✓ But C^* minus one of its edges is a spanning tree, which implies $2\times wt(T^*) \leq 2 \times wt(C^*)$.
- **Algorithm 6.4.3: Tree and Matching – better way to yield eulerian tour**
- **Theorem 6.4.5.** *For all instances of the TSP that obey the traingle inequality, the solution produced by Christofides' algorithm is never worse than* $\frac{3}{2}$ *times the optimal.*

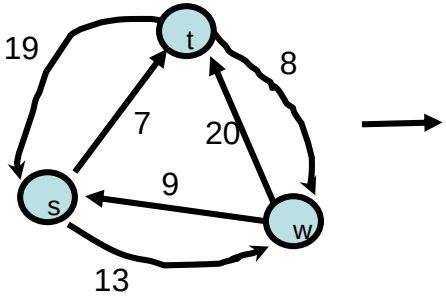
Two Heuristic Algorithms that Have Performance Guarantees

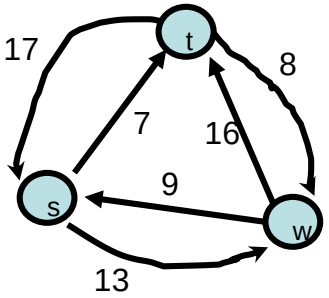
Algorithm 6.4.3: Tree and matching
Input: a weighted complete graph G.
Output: a sequence of vertices and edges that forms a hamiltonian cycle.
Find a minimum spanning tree T^* of G.
Let O be the subgraph of G induced on the odd-degree vertices in T^*.
Find a minimum-weight perfect matching M^* in O.
Create an Eulerian graph H by adding the edges in M^* to T^*.
Construct an Eulerian tour W of H, as in Algorithm 6.4.2.
Construct a Hamiltonian cycle in G from W, as in Algorithm 6.4.2.

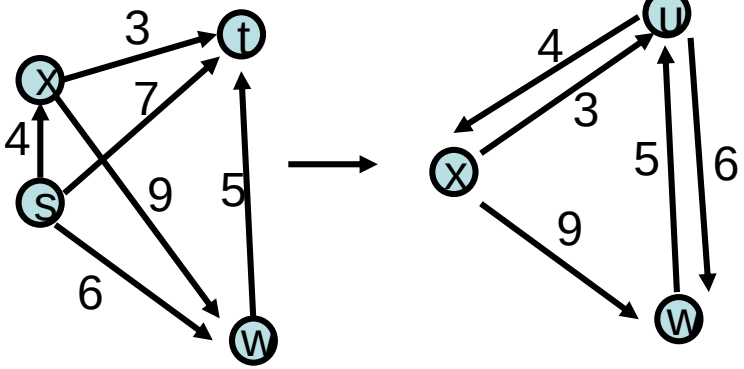
TSP's in Disguise

- **Variation 1.** Find a minimum-weight hamiltonian path.
- **Variation 2.** Find a minimum-weight hamiltonian path in digraph *G*, from a specified vertex *s* to a specified vertex *t*.
- **Variation 3.** Find a minimum-weight closed walk in digraph *G* that visits each vertex at least once.









$$c^*_{i,j} = \begin{cases} c_{s,j}, & \text{if } i = u \\ c_{i,t}, & \text{if } j = u \\ c_{i,j}, & \text{otherwise} \end{cases}$$