Quest Journals
Journal of Research in Applied Mathematics
Volume 8 ~ Issue 10 (2022) pp: 17-23
ISSN(Online) : 2394-0743 ISSN (Print): 2394-0735
www.questjournals.org

Research Paper

Three Applications of Graph Theory

Eric Choi, Anthony Delgado, Marty Lewinter

Abstract
Graph theory became a very important part of mathematical modeling in the 20th century with the advent of operations research and computer science. See [1-32]. We examine three real-world problems, two of them involving searches for spanning trees, and one involvinghypercubes.

***R**eceived 22 Sep., 2022; **R**evised 03 Oct., 2022; **A**ccepted 05 Oct., 2022 © The author(s) 2022.*
P**ublished with open access at **www.questjournals.org

I. Orienting directed graphs

A *directed graph*, abbreviated *digraph*, has the property that every edge has a direction. A directed edge *xy* goes from *x* to *y* and is represented by an arrow toward *y*, as shown in Figure 1, where *x* is adjacent to *z* while *z* is not adjacent to *x*.

Given a digraph, the graph with arrows removed is called the *underlying graph*. For the graph of Figure 1, the underlying graph is a two-dimensional mesh, or grid, with dimensions 3 by 2, denoted M(3, 2). A two-dimensional mesh where one of the dimensions is 2, is called a ladder,for obvious reasons.

Notice that the digraph of Figure 1, looks connected. One can find a directed path from *y* to *x*. There is, however, no directed path from *x* to *y*. Since the underlying graph is connected, we call the digraph *weakly connected*. On the other hand, a digraph with the property that any pair of vertices *u* and *v* have a directed *u*-*v* path is called *strongly connected*. Given a connected graph *G*, we describe the act of assigning a direction to each edge as *orienting* the graph. If the resulting graph is strongly connected, we call the orientation *strong*.

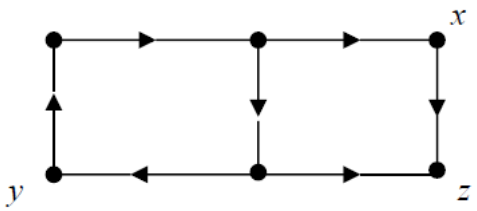


Figure 1

Reversing the direction of the horizontal edge incident with *z* in the digraph of Figure 1, yields a strong orientation. See Figure 2.

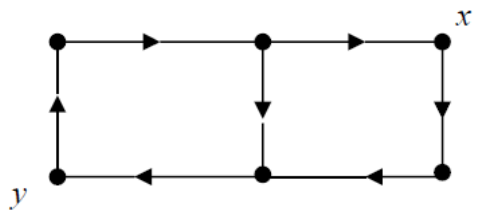


Figure 2

Obtaining a strong orientation for a connected graph can be used to streamline traffic flow in a congested city by making all streets one-way.

Of course, a graph with a bridge cannot be strongly oriented. On the other hand, if a graph, G, on *n* vertices, contains no bridges, it is always possible to obtain a strong orientation.

To show this, we use the *depth first search* (DFS) algorithm for finding a *spanning tree* for G, that is, an acyclic connected graph that contains all of the vertices.

*Corresponding Author: Eric Choi.

(1) Select a vertex of *G* and number it 0. This is the root.
(2) Select any new neighbor of the root and number it 1. Keep edge 01 and orient it from 0, the smaller number, to 1, the larger number.
(3) Select any new neighbor of 1 and call it 2. Keep edge 12, and orient it from 1 to 2.
(4) Continue in this manner until we either exhaust the vertex set of *G*, in which case the algorithm terminates with an oriented spanning tree T, or we reach a vertex numbered$m < n$ with no new neighbors.
(5) If this is the case, we must 'backtrack,' that is, we must find a previously visited vertex *k* with *highest* number such that this vertex has a new neighbor. Call the new vertex $m + 1$, and add the directed edge from *k* to $m + 1$ to the tree and continue the algorithm till it terminates or once again reaches a vertex numbered $p < n$ with no new neighbors. In this case, repeat step 5, and so on, till all vertices have been numbered and we have a spanningtree consisting of directed edges.

To obtain a strong orientation for G, after constructing the spanning tree T (using a DFS), orient each of the edges of G not in T, toward the vertex with lower number.

Let's see how this process can be used to give the graph G = M(3,3) a strong orientation using the DFS algorithm. In Figure 3, we labeled a vertex with a 0 and chose a neighbor, which we labeled with a 1. The edge between them is bold. It is the first edge of the desired spanning tree, T.

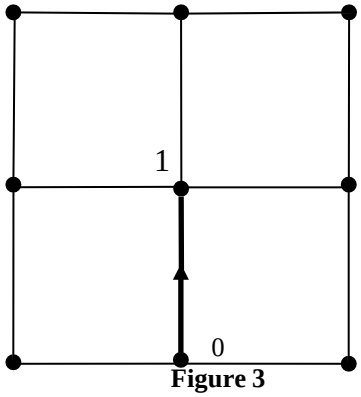


Figure 3

Now select a neighbor of 1 and call it 2. Orient the edge from 1 to 2 and bolden it. See Figure 4.

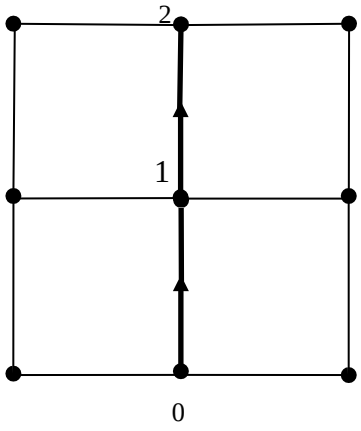


Figure 4

We continue this process till we obtain 5, which has no new neighbors. See Figure 5.

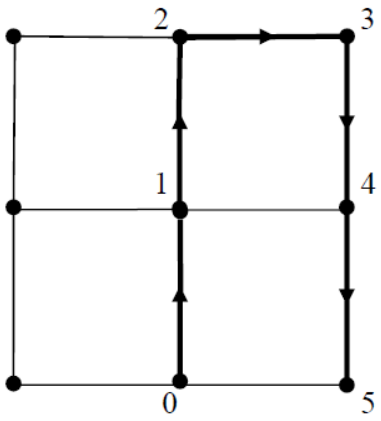


Figure 5

We then must backtrack and search for the highest number that has a new neighbor. This is 2.Label its neighbor, 6, and continue the algorithm till the spanning tree T is finished. See Figure 6.

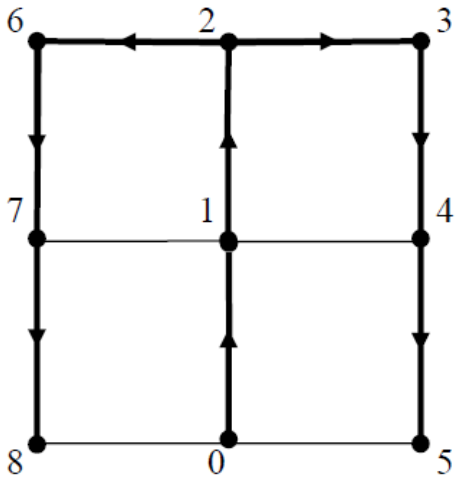


Figure 6

Finally, we orient each of the edges of G not in T, toward the vertex with lower number,obtaining the strongly connected digraph of Figure 7.

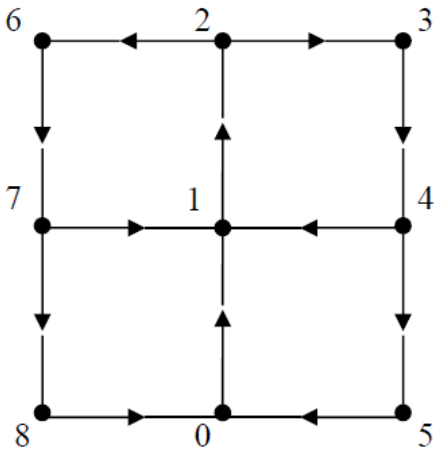
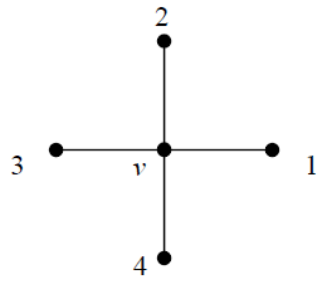


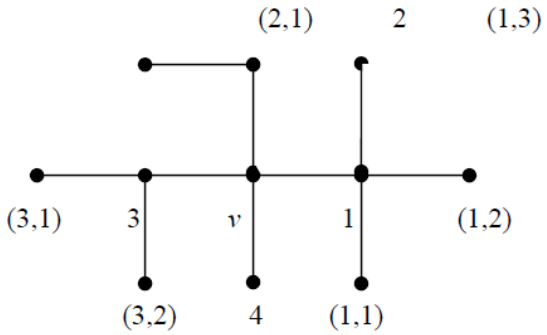
Figure 7

II. Reach-preserving vertices of spanning trees

A vertex, *v*, in a graph G with a spanning tree T, is called *reach-preserving*, if its distance from any vertex, *x*, in T is the same as its distance from *x* in G. In symbols, $d_T(v, x) = d_G(v, x)$, for all *x* in G. Now imagine a network of roads in a financially strapped community that must scrap some of them to yield a connected network with the minimum number of highways. Furthermore, we wish to preserve the distances from the local hospital to all the road intersections. Modeling this situation with a graph, G, in which each edge represents a road, and a particular vertex, *v*, represents an important facility such as a hospital or police station, we must produce a spanning tree, T, in which *v* is reach-preserved. Fortunately, this can be done using an algorithm called a *Breadth First Search* (BFS), which we will now describe. *v* will be the called the root.

(i) Keep all edges incident with *v*, and number its neighbors consecutively in any order starting with 1.
(ii) Number the neighbors of 1 that we haven't numbered yet by (1,*j*) where *j* increases consecutively starting at 1. Keep all edges from 1 to the (1,*j*)'s.
(iii) Repeat this for the remaining vertices with one coordinate. We now have a list of vertices {*v*, 1, 2, ... , (1,1), (1,2), ... , (2,1), (2,2), ... , (3,1), (3,2), ... } and a list of edges {(*v*)(1), (*v*)(2), ... , (1)(1,1), (1)(1,2), ... , (2)(2,1), (2)(2,2), ... , (3)(3,1), (3)(3,2), ...}. The list of vertices is in *lexicographic order* –like the order of words in a dictionary. Thus in our vertex list, for example, (3,1) precedes (3,2). We denote by D_r all vertices with *r* coordinates. At this stage of the algorithm, our list of vertices is $\{v\} \cup D_1 \cup D_2$. The distance from *v* to any vertex on our list is the same in *T* as it is in *G*, that is, *v* is reach-preserving so far.
(iv) Label the neighbors of (1,1) which are not yet on our list by (1,1,*j*), with *j* increasing consecutively, starting with 1, and add them to our list of vertices. Add all edges of the form (1,1)(1,1, *j*) to our list of edges. Visit the remaining vertices of D_2 in lexicographic order and do the same as was done with (1,1). Our list of vertices has now acquired D_3.
(v) Continue the algorithm till each vertex has been added to our list. Done. The figures below illustrate the BFS algorithm in stages for the 2-mesh M(5,3). The root is the center vertex.





The graph of Figure 8 yields the final stage, namely, a BFS spanning tree for M(5,3) in which *v* is reach-preserved.

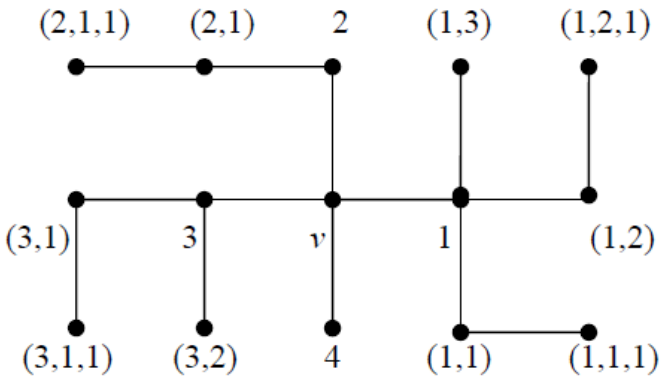


Figure 8

A BFS search is also useful in a news broadcasting system. Suppose it takes a certain unit of time to notify a single vertex (the root) about a news story. Then it takes one unit of time for the root to notify all of its neighbors. These new neighbors, in turn, require one unit of time to notify their neighbors, and so on, until all the vertices have been notified.

Select a center vertex, that is, a vertex with the smallest maximum distance to the rest of the vertices, as the root. Then continue with a BFS to find the rest of the notification scheme. This will minimize the total notification time.

III. Error-correcting codes using hypercubes

We wish to model the search for a one digit error correcting code using the hypercube, Q_3. To begin, here are a few definitions.

One digit error correcting code: A scheme that corrects a transmission error of one binary digit. For example, if the text is 000011110 and it erroneously arrives as 100011110, that is, the first digit is incorrect, we want the code to automatically correct it.

Hypercubes: A family of graphs defined recursively as follows: The first hypercube, Q_1, consists of two adjacent vertices labeled 0 and 1. The second hypercube, Q_2, is formed by taking two labeled copies of Q_1 and doing two things. (1) Render corresponding vertices adjacent. (2) Relabel the vertices of the left copy of Q_1 with 00 and 01, and the vertices of the right copy with 10 and 11. That is, we placed a 0 in front of the labels of the left copy of Q_1 (in bold) and a 1 in front of the labels of the right copy (in bold). See Figure 9.

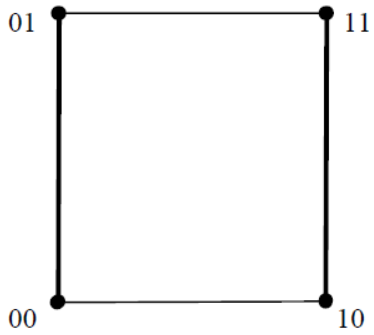


Figure 9

The third hypercube, Q_3, is formed by taking two labeled copies of Q_2, one inside the other for visual clarity (no edge crossings), and doing two things. (1) Render corresponding vertices adjacent. (2) Relabel the vertices of the inner copy of Q_2 with 100, 101, 110, and 111. Then relabel the vertices of the outer copy with 000, 001, 010, and 011. That is, we placed a 0 in front of the labels of one copy of Q_2 and a 1 in front of the labels of the other copy. The two copies of Q_2 are in bold. See Figure 10.

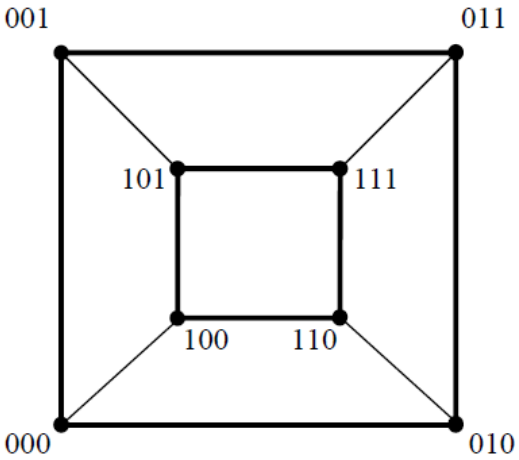


Figure 10

Note that two vertices in Q_1, Q_2, or Q_3 are adjacent when their labels differ in exactly one place.
In Q_3, consider the two vertices $x = 000$ and $y = 111$. The rest of the vertices are adjacent to exactly one of x and y. That is, each of the remaining vertices are closer to one of x and y. For example, 010 is adjacent to x and two edges away from y.
Now take a text such as 110001 and replace each 0 by 000 and each 1 by 111. This unfortunately makes the text three times longer. We then have 111 111 000 000 000 111. If one digit is erroneously switched in transmission, we will have a sequence such as 101 instead of 111.

Then since 101 is closer to *y*, the code can be programmed to replace it by 111. This is the essence of error correcting codes using hypercubes.

Two questions occur to us at this point. (1) Can we develop a *two* digit error correcting code? We can but not in Q_3. (2) Can we deal with texts with more than two characters, not just 0 and 1. Yes we can, but not in Q_3.

Drawing higher order hypercubes with the method we used to obtain Q_2 and Q_3 gets quite difficult to visualize, so we regard Q_n as a collection of vertices labeled by each of the 2^n binary strings of length n. Two vertices are adjacent in Q_n if their labels differ in exactly one place. Thus 11011 is adjacent to 11111 in Q_5, since their labels differ in the third place.

Now suppose we are sending texts requiring four symbols, say {0, 1, ?, .}. Consider the following four vertices in Q_9. We added two spaces to make them easier to compare.

000 000 000 to replace '0'
111 111 000 to replace '1'
000 111 111 to replace '?'
111 000 111 to replace '.'

The distance between any two of them is six, since any two labels differ in exactly six places. Doing the indicated replacing makes the binary transmission nine times as long. If one or two digits are switched in transmission, the error correcting program substitutes the closest of theabove four vertices, and we are done.

References

[1]. M.Lewinter, Spanning trees of product graphs. *Cong. Num*. 54 (157-161) 1986.
[2]. M.Lewinter, *Graph Theory*. Monographs in Undergrad. Math.,13, JUM, Guilford, NC, 1986.
[3]. F.Harary, M.Lewinter, Hypercubes and other recursively defined hamilton-laceable graphs. *Cong.Num*. 60 (81-84) 1987.
[4]. M.Lewinter, Interpolation theorem for the number of degree--preserving vertices of spanning trees. *IEEE Trans.Circuits and Systems*, CAS-34,(205) 1987.
[5]. M.Lewinter, et al, Which two-legged caterpillars span hypercubes? *GTN of NY* XIV (30-32) 1987.
[6]. M.Lewinter, M.Migdail-Smith, Degree-preserving vertices of spanning trees of the hypercube. *GTNof NY* XIII (26-27) 1987.
[7]. M.Lewinter, Which caterpillars span a hypercube? *GTN of NY* XIII (23-25) 1987.
[8]. F.Buckley, M.Lewinter, On graphs with center-preserving spanning trees. *GTN of NY* XIV (33-35)1987.
[9]. F.Buckley, M.Lewinter, A note on graphs with diameter-preserving spanning trees. *J.Graph Theory*,12 (525-528) 1988.
[10]. F. Harary, M. Lewinter, The starlike trees which span a hypercube. Comput. Math. Appl., 15 (299-302) 1988.
[11]. F.Harary, M.Lewinter, Spanning subgraphs of a hypercube II: Double starlike trees. *Mathl. Comput.Modelling*, 1 (216-217) 1988.
[12]. F.Harary, M.Lewinter, Spanning subgraphs of a hypercube III: Meshes. *Intern.J.Computer Math*.,25(1-4) 1988.
[13]. F.Harary, M.Lewinter, et al, On two-legged caterpillars which span hypercubes. *Cong. Num*. 66(103-108) 1988.
[14]. M.Lewinter, *Graph Theory II*. Monographs in Undergraduate Mathematics, Vol. 13, JUM,Guilford, NC, 1988.
[15]. F.Harary, M.Lewinter, Spanning subgraphs of a hypercube V: Spanned subcubes. *Annals NY Academy of Sciences*, 576 (219-225) 1989.
[16]. J.Gratt, M.Lewinter, The eccentricities of a center-preserving spanning tree. *GTN of NY* XVII (18-19) 1989.
[17]. M.Lewinter, Which starlike trees span *n*-meshes? *GTN of NY* XVII (12-15) 1989.
[18]. M.Lewinter, et al, Which trees span ternary cubes? *GTN of NY* XVII (16-17) 1989.
[19]. M.Haag, M.Lewinter, Double embeddings of a graph in the hypercube. *GTN of NY* XX (50-51)1991.
[20]. F.Harary, M.Lewinter, Spanning subgraphs of a hypercube VI: Survey and unsolved problems.
[21]. *Graph Theory, Combinatorics, and Applications*, edited by Y.Alavi et al. J.Wiley NY (633-637) 1991.
[22]. M.Lewinter, et al, Equipartition sets of hypercubes. *J. Comb. Info. & System Sci.* 16 (19-24) 1991.
[23]. T.Bocchi, M.Lewinter, How many 4-cycles does a spanning tree of $M(m, n)$ span? *GTN of NY* XXIII (33-34) 1992.
[24]. M.Lewinter, et al, Which double starlike trees span ladders? *Annals of Discrete Math*, 55 (327-332)1993.
[25]. F.Buckley, M.Lewinter, Graphs with all diametral paths through distant central nodes. *Mathl. Comput. Modelling*, 11 (35-41) 1993.
[26]. F.Harary, M.Lewinter, Spanning subgraphs of a hypercube IV: Rooted trees. *Mathl. Comput.Modelling*, 17 (85-88) 1993.
[27]. F.Harary, M.Lewinter, All hypercubes are pluperfect. *Bull.Malaysian Math. Soc*. 16 (21-23) 1993.
[28]. M.Aaron, M.Lewinter, 0-deficient vertices of spanning trees. *GTN of NY* XXVII (31-32) 1994.
[29]. T.Bocchi, M.Lewinter, et al, *k* -deficient spanning trees. *GTN of NY* XXIX (42-43) 1995.
[30]. M.Lewinter, et al, An interpolation theorem for the maximum deficiencies of spanning trees. *GTNof NY* XXX (37) 1996.
[31]. M.Lewinter, et al, An infinite class of reach-preservable graphs. *Networks*, 29 (217-218) 1997.
[32]. M.Lewinter, Distance, spanning trees and hypercubes: new and old results. *GTN of NY* XXXII(42-46) 1997.
[33]. D.Aulicino, M.Lewinter, *k*-equidistant sets of Q_n . *GTN of NY* XXXVI (17-19) 1999.
[34]. D.Aulicino, F.Harary, M.Lewinter, Embedding *k*-dimensional meshes in hypercubes. *GTN of NY* XXXVII (51-53) 1999.
[35]. D.Aulicino, M.Lewinter, Pancentral graphs. *Cong. Num*. 150, (69-72) 2001.
[36]. M.Lewinter, et al, *The Saga of Mathematics*: *A Brief History*. Prentice-Hall, 2002.
[37]. F.Buckley, M.Lewinter, *A Friendly Introduction to Graph Theory*. Prentice-Hall, 2003.
[38]. M.Gargano, M.Lewinter, J.Malerba, Hypercubes and Pascal's triangle: A tale of two proofs. *Math. Magazine*, Vol.76, 3 (216-217) 2003.
[39]. M.Lewinter, et al, A survey of undergraduate research in graph theory. *GTN of NY* (8-15)2006.
[40]. *A.Delgado*, M.Gargano, M.Lewinter, L.V.Quintas, Spanning-tree-center vertices of a graph. *Cong.Num.* 189, (43-48) 2008.
[41]. A.Delgado, M.Gargano, M.Lewinter, et al, Subcubes of Hypercubes. *Cong. Num.* 189, (25-32)2008.
[42]. A.Delgado, M.Lewinter, L.Quintas, Equipartition sets of hypercubes II, *Bulletin of the ICA*, 63,(2011), 51-59.
[43]. M.Lewinter, B.Phillips, Spanning *k*-branched starlike trees. *GTN of NY*. LX (29-31) 2011.
[44]. F.Buckley, M.Lewinter, *Introductory Graph Theory with Applications*. Waveland Press. 2014.