Matrix Representation of Graph and Random Walks on Graphs

Report submitted in fulfillment of the requirements

for the Exploratory Project (3rd Semester)

Second Year IDD

by

Kartik Malik

20124026

Under the guidance of

Prof. S. K. Pandey

Aug 2021

Dedicated to

My parents and my teachers.

Declaration

I certify that

1. The work contained in this report is original and has been done by myself and the general supervision of my supervisor.

2. The work has not been submitted for any project.

3. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

4. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: 29th November, 2021

Kartik Malik
IDD (Part II) Student
Department of Mathematical Science
Indian Institute of Technology (BHU)
Varanasi, INDIA 221005.

Certificate

*This is to certify that the work contained in this report entitled "**Matrix Representation of Graph and Random Walks on Graphs**" being submitted by **Kartik Malik** (Roll No. **20124026**), carried out in the Department of Mathematical Science, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: 29th November, 2021

Prof. S. K. Pandey
Professor
Department of Mathematical Science
Indian Institute of Technology (BHU)
Varanasi, INDIA 221005.

Acknowledgments

I would like to express my sincere gratitude to my supervising Professor S.K. Pandey for his constant guidance and support during the whole project work.

Place: IIT (BHU) Varanasi
Date: 29th November **Kartik Malik**

Abstract

Pictorial representation of a graph is very convenient for visual study, other models are better for computer processing. Thus, we need some data structure that can easily represent a graph, which is easy to use and implement for the computations. A matrix turns out to be a convenient and useful way of representing a graph to a computer. The incidence matrix and the adjacency matrix are the two most commonly used matrices for representing a graph, which describes a simple graph completely, up to isomorphism. The adjacency matrix can be modified slightly to the adjacency list, proving to be more efficient than the matrix when dealing with sparse graphs.

Graphs with trillions of edges occur in various fields, such as machine learning, social network analysis, etc. In such cases also, the adjacency matrix can be processed in specific ways to increase efficiency and reduce the I/O and memory requirements.

Markov Chains has a wide range of applications in the topics ranging from physics, chemistry, biology, medicine, music, game theory to sports. Markovian systems appear extensively in thermodynamics and statistical mechanics, whenever probabilities are used to represent unknown or unmodelled details of the system, if it can be assumed that the dynamics are time-invariant, and that no relevant history need be considered which is not already included in the state description. We will first discuss the random walks on Markov Chains and then will follow the same steps on an undirected unweighted network, leading to some important properties and results of random walks on graphs.

Contents

Abstract **7**

Chapters

1 Introduction **10**

1.1 Overview

1.1.1 Graphs

1.1.2 Basic Terminologies

1.2 Motivation

1.3 Contribution

1.4 Organization of Report

2 Matrix Representation of graphs **15**

2.1 Basic Definitions

2.2 Incidence Matrix

2.2.1 Properties

2.3 Adjacency Matrix

2.3.1 Properties

2.3.2 As Data Structures

3 Random Walks **20**

3.1 Random Walks

3.1.1 Markov Chains

3.1.2 Transition Matrix and Properties

3.1.3 Random Walks on undirected unweighted networks

3.2 Methodology

3.2.1 Naive Matrix Exponentiation

3.2.2 Efficient Matrix Exponentiation

3.3 Paths on chessboard

4 Conclusion and Discussion 28

4.1 Result

4.2 Conclusion

Bibliography 30

Chapter 1

Introduction

1.1 Overview

1.1.1 Graphs

A graph is a structure comprising a set of objects (which corresponds to the mathematical abstractions called vertices, or sometimes nodes or points) in which some pairs of the objects are in some sense related, (each of the related pairs of vertices is called an edge). Typically, a graph is depicted in diagrammatic form as a set of dots or circles for the vertices, joined by lines or curves for the edges, which may be directed or undirected.

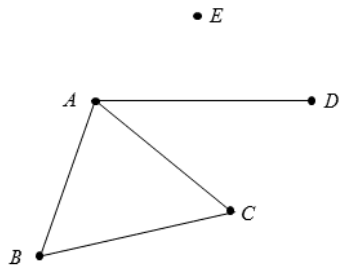


Fig 1.1 Diagrammatic representation of a graph

Formally, a graph G can be defined as a pair (V, E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq \{(u, v) \mid u, v \in V\}$.

Multiple edges (two or more edges that join the same two vertices) can not be shown using the above definition. In more general sense of the term allowing multiple edges, a graph is an ordered triple $\{G = (V, E, \phi)\}$ where V is a set of vertices, E is the set of edges and $\phi: E \longrightarrow \{(u, v) \mid u, v \in V\}$ is an incidence function mapping every edge to a pair of vertices.

For example in the Fig. 1.1, V = {A, B, C, D, E} and E = {{A, B}, {B, C}, {A, C}, {A, D}}, and the graph is simply represented as G(V, E).

1.1.2 Basic Terminologies

- An **undirected graph** is a graph in which each edge symbolizes an unordered, transitive relationship between two nodes. Such edges are shown by plain lines or arcs. Thus in such graphs, the adjacency relation defined by the edges is symmetric, or $E \subseteq \{\{u, v\} \mid u, v \in V\}$ (sets of vertices, rather than ordered pairs).
- A **directed graph** is a graph in which each edge symbolizes an ordered, non-transitive relationship between two nodes. Such edges are shown by lines or arcs with an arrowhead at one end.
- An **unweighted graph** is a graph in which all the relationships symbolized by the edges are considered equivalent. Such edges are rendered as plain lines or arcs.
- A **weighted graph** is a graph in which each edge has a weight (some real value) which symbolizes the relationship between the nodes, such as the cost, capacity, distance, etc. Such edges are usually annotated by a number or letter placed beside the edge.
- A **simple graph** is an unweighted, undirected graph containing no self-loops or parallel edges. Since the graph does not allow self-loops, the

adjacency is irreflexive, i.e., $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$.

- The **degree** or *valency* of a vertex is the number of edges that are incident to it, with self-loop counted twice. The degree of vertex v_i is generally represented as $d(v_i)$. The degree of a graph is the maximum of the degrees of its vertices.
    - Indegree of a vertex is the number of edges *pointing* to a node, i.e., the number of head ends adjacent to a vertex.
    - Outdegree of a vertex is the number of edges going out of a node, i.e., the number of tail ends adjacent to a vertex.
- A **walk** is defined as a finite alternating sequence of vertices and edges, beginning and ending with vertices, such that each edge is incident with the vertices preceding and following it. No edge appears (or is traversed) more than once in a walk. A vertex, however, may appear more than once.
    - A *closed walk* is a walk whose first vertex is the same as the last.
    - An *open walk* is a walk which ends on a different vertex from the one where it starts.
    - A *path* is an open walk in which no vertex appears more than once.

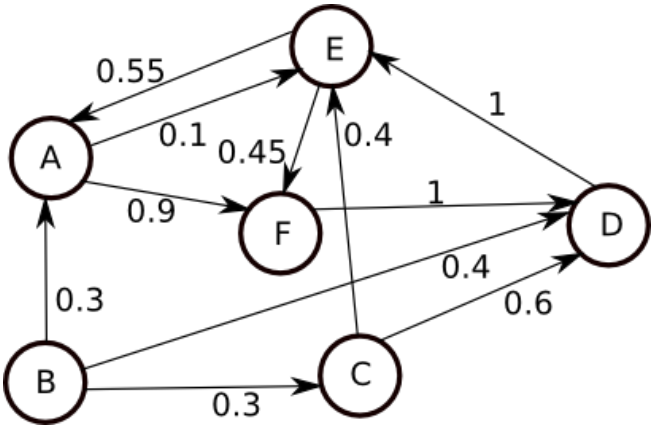


Fig 1.2 Diagrammatic representation of a weighted, directed graph. The graph is not simple, since there is a parallel edge between A and E. In the graph, the indegree of F is 2 whereas it's outdegree is 1.

1.2 Motivation

Although a diagrammatic representation of a graph is very convenient for visual study, other representations are better for computer processing. Thus, we need some data structure that can easily represent a graph, which is easy to use and implement for the computations. That data structure must be able to do the modifications, such as adding or removing the edge or vertex, giving weights to the edge in the graph easily, or telling if two vertices are connected directly or not. A matrix turns out to be a just, convenient and useful way of representing a graph to a computer. Matrices lend themselves easily to mechanical manipulations. Besides, many known results of matrix algebra can be readily applied to study the structural properties of graphs from an algebraic point of view. In many applications of graph theory, such as in electrical network analysis and operations research, matrices also turn out to be the natural way of expressing the problem. The incidence matrix and the adjacency matrix are the two most commonly used matrices for representing a graph.

Random walks on graphs are sometimes of great interest, they are of great use in modeling scientific theories whenever probabilities are used to represent unknown or unmodelled details of the system. Markov Chains become a lot more useful too if it can be assumed that the dynamics are time-invariant, and that no relevant history need be considered which is not already included in the state description. Thus, we will explore random walks on the graphs.

1.3 Contribution

In this work, we explore the adjacency matrix for representation of a graph, a simple data structure which can be used for computer processing. We will also explore the random walks on the graphs, starting from the Markov Chains.

1.4 Organization of Report

The second chapter contains some vital background information about matrix representation of graphs and definitions used throughout the report.

The third chapter describes some algorithms exploiting the properties of transition and adjacency matrices. The detailed algorithms, programs (in c++) and other important representations are layed out that would help us to interpret the results described.

In the last chapter we conclude by discussing the results and discussing the probable effects of the results in the field of efficient representation of graphs.

Chapter 2

2.1 Basic Definitions

- In the matroid theory of graphs the **rank** of an undirected graph is defined as the number n − c, where c is the number of connected components of the graph.
- Analogously, the **nullity** of an undirected graph is defined as the number e - (n − c), where c is the number of connected components of the graph.

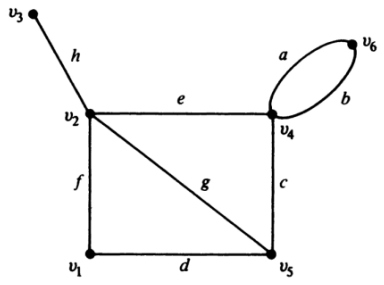
(Rank, Nullity)

- The **rank** of the matrix refers to the number of linearly independent rows or columns in the matrix. Thus, the dimension of the vector space spanned by its rows (or columns, will be the same) is the rank of the matrix.

2.2 Incidence Matrix

Incidence matrix is a two-dimensional matrix in which the rows represent the vertices and columns represent the edges. The entries indicate the incidence relation between the vertex at a row and edge at a column, i.e., in a graph of n vertices and e edges, incidence matrix is a n × e matrix, $A = [a_{ij}]$, where:

$$a_{ij} = \begin{cases} 1, & \text{if } j^{th} \text{ edge } e_j \text{ is incident on } i^{th} \text{ vertex } v_i \\ 0, & \text{otherwise} \end{cases}$$



(a)

	a	b	c	d	e	f	g	h
v_1	0	0	0	1	0	1	0	0
v_2	0	0	0	0	1	1	1	1
v_3	0	0	0	0	0	0	0	1
v_4	1	1	1	0	1	0	0	0
v_5	0	0	1	1	0	0	1	0
v_6	1	1	0	0	0	0	0	0

Fig 2.1 Graph and its incidence matrix

Image source: Graph Theory with Applications to Engineering and Computer Science by Narsingh Deo

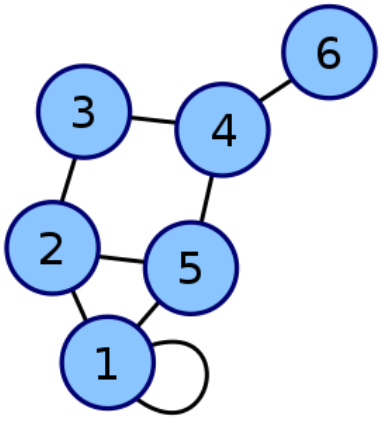
2.2.1 Properties

- Since every edge is incident on precisely two vertices, each column of A has exactly two 1′s. Thus, parallel edges will produce identical columns in the matrix.
- The number of 1′s in each row equals the degree of the corresponding vertex. Thus, a row with all 0's is an isolated vertex.
- Rank of the incidence matrix is the same as the rank of the graph. Thus for a connected graph of n vertices, the rank of the incidence matrix is n - 1, thus we need only n – 1 rows of an incidence matrix to specify the corresponding graph completely. Such an (n – 1) × e submatrix A_f of A is called a ***reduced incidence matrix.***
- Reduced incidence matrix for a tree will be (n – 1) × (n - 1) matrix, and since its rank is (n - 1), it will be non-singular.

2.3 Adjacency Matrix

Adjacency matrix is a two-dimensional matrix, in which the rows represent source vertices and columns represent destination vertices. Only the cost for one edge can be stored between each pair of vertices, i.e., information about the parallel edges can not be stored. Thus, the adjacency matrix of an undirected graph G with n vertices and no parallel edges is an n × n symmetric binary matrix A = $[a_{ij}]$ defined over the ring of integers such that:

$$a_{ij} = \begin{cases} 1, & \text{if there is an edge between the i}^{th} \text{ and j}^{th} \text{ vertices} \\ 0, & \text{otherwise} \end{cases}$$



$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Coordinates are 1–6.

Fig 2.2 Graph and its adjacency matrix

Source: Wikipedia

2.3.1 Properties

- The entries along the principal diagonal of A are all 0's if and only if the graph has no self-loops. A self-loop at the i^{th} vertex corresponds to $a_{ii} = 1$.
- The adjacency matrix A of a bipartite graph whose two parts have r and s vertices can be written in the form

$$A = \begin{pmatrix} 0_{r,r} & B \\ B^{\mathsf{T}} & 0_{s,s} \end{pmatrix},$$

where B is an r × s matrix, and $0_{r,r}$ and $0_{s,s}$ represent the r × r and s × s zero matrices. In this case, the smaller matrix B uniquely represents the graph, and the remaining parts of A can be discarded as redundant. B is known as the ***biadjacency matrix***.

Formally, let G = (U, V, E) be a bipartite graph with parts $U = \{u_1, \ldots, u_r\}$, $V = \{v_1, \ldots, v_s\}$ and edges E. The biadjacency matrix is the r × s 0–1 matrix B in which $b_{i,j} = 1$ if and only if $(u_i, v_j) \in E$.

- **Isomorphism**: Suppose two directed or undirected graphs G_1 and G_2 with adjacency matrices A_1 and A_2 are given. G_1 and G_2 are isomorphic if and only if there exists a permutation matrix P such that $PA_1P^{-1} = A_2$.

In particular, A1 and A2 are similar and therefore have the same characteristic polynomial, eigenvalues, determinants, and trace. These can therefore serve as isomorphism invariants of graphs. However, two graphs may possess the same set of eigenvalues but not be isomorphic.

2.3.2 As Data Structures

The adjacency matrix may be used as a data structure for the representation of graphs in computer programs for manipulating graphs. The main alternative data structure, also in use for this application, is the *adjacency list*. Because each entry in the adjacency matrix requires only one bit, it can be represented in a very compact way, occupying only $|V|^2/8$ bytes to represent a directed graph, or (by using a packed triangular format and only storing the lower triangular part of the matrix) approximately $|V|^2/16$ bytes to represent an undirected graph.

Chapter 3

3.1 Random Walks

Let G = (V, E, w) be a weighted graph. A random walk on a graph is a process that begins at some vertex, and at each time step moves to another vertex randomly with the probability proportional to the weight of the corresponding edge. When the graph is unweighted, the vertex the walk moves to is chosen uniformly at random among the neighbors of the present vertex. *Drunkard's walk* is an example of a random walk.
When the graph is allowed to be directed and weighted, such a walk is also called a Markov chain.

3.1.1 Markov Chain

A Markov chain is a sequence of possible events in which the probability of each event depends only on the present state, i.e., the past affects the future only through the present. Markov chains are often described by a sequence of directed graphs, where the edges of graph *n* are labeled by the probabilities of going from one state at time n to the other states at time n + 1, $P(X_{n+1} = x \mid X_n = x_n)$. The same information is represented by the transition matrix from time n to time n + 1.

Absorbing Markov Chain

In the mathematical theory of probability, an absorbing Markov chain is a Markov chain in which every state can reach an absorbing state. An absorbing state is a state that, once entered, cannot be left.

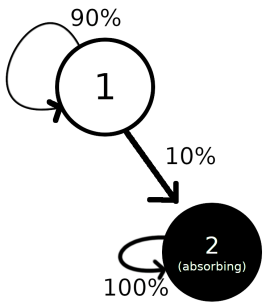


Fig 3.1 A Markov chain with two states, 1 and 2, with 2 completely absorbing.
Source: Dev.to

A state that is not absorbing is called a transient state.

3.1.2 Transition Matrix

Transition Matrix is a square matrix used to describe transitions of a Markov chain, whose each entry is a non-negative real number representing a probability. Thus, a transition matrix is the same as an adjacency matrix of a directed graph with values in each row summing to 1. If the probability of moving from i to j in one time step is $P(j \mid i) = P_{i,j}$, the transition matrix P is:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \dots & P_{1,j} & \dots & P_{1,S} \\ P_{2,1} & P_{2,2} & \dots & P_{2,j} & \dots & P_{2,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{i,1} & P_{i,2} & \dots & P_{i,j} & \dots & P_{i,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{S,1} & P_{S,2} & \dots & P_{S,j} & \dots & P_{S,S} \end{bmatrix}.$$

Properties

- Since the total of transition probability from a state i to all other states must be 1, i.e., $\sum_{j=1}^{S} P_{i,j} = 1.$
- Let X_n has the distribution **Q,** where **Q** is the 1 × S row vector, then

  $P(X_{n+1} = j) = \sum_i P(X_{n+1} = j \mid X_n = i)$, i.e., $\sum_i Q_i P_{ij}$ which is jth entry of $Q.P.$

  Thus, the X_{n+1} will have the distribution **Q**.P. Similarly X_{n+2} will have the distribution **Q**.P^2, X_{n+3} = **Q**.P^3, and so on… Similarly, if $P(X_n = i)$, i.e., if we are on a particular state at present, then after m steps, $P(X_{n+m} = j)$ will be the (i, j)th entry of P^m.
- Canonical form of an absorbing matrix

  Let an absorbing Markov chain with transition matrix P have t transient states and r absorbing states. Then

$$P = \begin{pmatrix} Q & R \\ \mathbf{0} & I_r \end{pmatrix},$$

  where Q is a t × t matrix, R is a nonzero t × r matrix, **0** is an r × t zero matrix, and I_r is the r × r identity matrix. Thus, Q describes the probability of transitioning from some transient state to another while R describes the probability of transitioning from some transient state to some absorbing state.

- The expected number of visits to a transient state j starting from a transient state i (before being absorbed) will be inv(I - Q).
  - Proof

    Since the probability of transitioning from i to j in exactly k steps is the entry at (i, j) of Q^k, thus N = I + Q + Q^2 + Q^3 + Q^4 + … , i.e.,

    N = 1 / (I - Q) = inv(I - Q).

- Thus, the expected number of steps before being absorbed when starting at the transient state i will be the sum of N_{ij} for all j, i.e., N**1**, where **1** is a length-t column vector whose entries are all 1.

3.1.3 Random Walks on undirected unweighted networks

The above properties can also be applied to undirected, unweighted graphs since the adjacency matrix of graphs is analogous to the transition matrix of Markov chains.

- If A is the adjacency matrix of the directed or undirected graph G, then the element (i, j) of the matrix A^n gives the number of (directed or undirected) walks of length n from vertex i to vertex j.
  If n is the smallest non-negative integer, such that for some i, j, the element (i, j) of A^n is positive, then n is the distance between vertex i and vertex j.
- Matrix Power A^n when calculated by taking the *distance product* ($c_{ij} = \min(a_{ik} + b_{kj})$) : the element (i, j) gives the shortest walk of n steps from vertex i to vertex j.

Thus, we can now use the above properties to design a simple method for finding the number of paths, minimum path lengths between any two nodes or even answer such queries.

3.2 Methodology

Matrix exponentiation seems to be an excellent tool for answering most of the challenges. Thus, let's see the algorithm to calculate A^m for a graph with n vertices, e edges.

3.2.1 Naive Matrix Exponentiation

Directly applying the mathematical definition of matrix multiplication gives an algorithm that takes time of the order of n^3 field operations to multiply two $n \times n$ matrices over that field ($\Theta(n^3)$ in big O notation). Thus, A^m would have the complexity $\Theta(n^3 m)$.

```
struct Matrix {
	vector<vector<int>> result;
	int n;
	Matrix (int _n) : n(_n) {
		result.resize(n);
		for (int row = 0; row < n; row++)
			result[row].resize(n);
	}

	Matrix operator*(const Matrix& b) const {
		int _n = n;
		Matrix ans(n);
		for (int row = 0; row < n; row++) {
			for (int col = 0; col < n; col++) {
				for (int idx = 0; idx < n; idx++) {
					ans.result[row][col] +=
					 result[row][idx] * b.result[idx][col];
				}
			}
		}
		return ans;
	}
};

Matrix Identity(int n) {
	Matrix Iden(n);
	for (int row = 0; row < n; row++)
		Iden.result[row][row] = 1;
	return Iden;
```

```
}
Matrix power(Matrix mat, ll expo) {
      if (expo == 0)
            return Identity(mat.n);
      if (expo == 1)
            return mat;
      Matrix ans = power(mat, (expo - 1));
      ans = ans * mat;
      return ans;
}
```

However, we can significantly improve the exponentiation part; we don't need to calculate all A's powers from 1, 2, ..., m. Instead, we can exploit the fact that A^4 can be calculated as $A^2 \times A^2$, thus for calculating A^m, we'll calculate it as $A^{m/2} \times A^{m/2}$ if m is even or $A \times A^{(m-1)/2} \times A^{(m-1)/2}$ if m is odd. Thus, significantly improving the time complexity from linear in m to logarithmic in m, i.e., from $\Theta(n^3 m)$ to $\Theta(n^3 \log m)$.

3.2.2 Efficient Matrix Exponentiation

Since we are only optimizing the power function, the rest of the code will remain the same, except for the power function.

```
Matrix power(Matrix mat, ll expo) {
      if (expo == 0)
            return Identity(mat.n);
      if (expo == 1)
            return mat;
      Matrix ans = power(mat, (expo >> 1));
      ans = ans * ans;
      if (expo & 1) ans = ans * mat;
      return ans;
}
```

Since, the number of walks of length n between two vertices is just A^n, where A is the adjacency matrix. Thus, a_{ij} of A^n will give the number of such walks.

A slight variation to this problem is the shortest walk of n steps between any two vertices. Replacing a normal product by the distance product in multiplication operator overloading will give the desired result.

```
Matrix operator*(const Matrix& b) const {
      int n = _n;
      Matrix ans(n);
      for (int i = 0; i < _n; i++) {
            for (int j = 0; j < _n; j++) {
                  ans.val[i][j] = INF;
                  for (int k = 0; k < _n; k++) {
                        ans.val[i][j] =
                          min(ans.val[i][j], val[i][k] + b.val[k][j]);
                  }
            }
      }
      return ans;
}
```

where INF is some large value.

Another variation of the above problem could be if the walk length is bounded by m instead of being equal to m. If we calculate all powers of A, from 1 to m, time complexity will be $\Theta(n^3 m)$, checking minimum (or adding the number of paths) at each power of A. However, adding a *dummy node* and *dummy directed edges* from this node to all nodes and to itself (self-loop) will solve the problem. Thus, making a $(n + 1) \times (n + 1)$ matrix will reduce the time complexity to $\Theta(n^3 \log m)$ in compensation for a slight increase in the space.

3.3 Paths on chessboard

Every chess piece has a restricted set of possible moves on the chessboard. Visualizing the chessboard as a 64-node graph with the edges, depending on the chess piece, representing that the piece has a legal move between the pair of nodes.

For example, consider the *Knight's graph*, which represents all legal moves of the knight chess piece on a chessboard. Each vertex of a knight's graph represents a square on a chessboard, and each edge represents a legal move from one square to another.

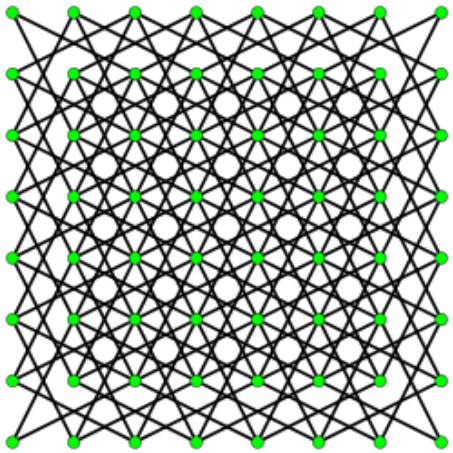


Fig 3.1 Knight's graph
Image source: Wikipedia

For example, consider the problem where a knight stands on the top-left corner of an 8 × 8 chessboard, and one needs to count paths of cells (C_1, C_2,..., C_s) such that $1 \leq s \leq k+1$ and a knight can move from C_i to C_{i+1} $\forall$ i.

Making an adjacency matrix A := 65 × 65 matrix (adjacency matrix of 64-node graph + a dummy vertex) and calculating A^{k+1}.result[0][64] (number of paths of length k+1 from top-left corner to dummy vertex) will suffice.

Chapter 4

Conclusion and Discussion

4.1 Result

We examine the various uses of the adjacency matrix, most of which revolves around the crucial results obtained from the random walks on Markov Chains. Apart from having extra perks, the adjacency matrix also proves to be an optimal way to represent a graph to a computer for computer processing. It holds a considerable advantage over the incidence matrix when supporting basic operations on the graph in terms of the time taken to do the process. A slight variation to the adjacency matrix, adjacency list, is another efficient way to represent a graph. Adjacency list is preferred when dealing with sparse graphs, while an adjacency matrix deals with dense graphs more efficiently. The following table compares the adjacency list, adjacency matrix, incidence matrix in terms of time complexity to perform some common operations.

	Adjacency list	Adjacency matrix	Incidence matrix
Store graph	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Add vertex	$O(1)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Add edge	$O(1)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Remove edge	$O(\|V\|)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Are vertices *x* and *y* adjacent (assuming that their storage positions are known)?	$O(\|V\|)$	$O(1)$	$O(\|E\|)$

Table 4.1 Graph and its incidence matrix

Image source: Wikipedia

4.2 Conclusion

In summary, we discussed two ways to represent a graph through a matrix, the incidence matrix and the adjacency matrix, which describes a simple graph completely, up to isomorphism. The adjacency matrix proves to be more efficient than the incidence matrix for supporting the modifications to the graph. The adjacency matrix can be modified slightly to the adjacency list, proving to be more efficient than the matrix when dealing with sparse graphs. We also discussed the random walks on a graph, extending results of Markov chains.

Bibliography

Graph Theory with Applications to Engineering & Computer Science by Narsingh Deo

AITKEN , A.C, “Determinants and Matrices,” 9th ed., Oliver & Boyd Ltd., Edinburgh, 1956.

B ERGE , C., The Theory of Graphs and Its Applications, John Wiley & Sons, Inc., New York, 1962. English translation of the original book in French: Théorie des graphes et ses applications, Dunod Editeur, Paris, 1958.

Wikipedia contributors. (2021, October 4). Adjacency matrix. In Wikipedia, The Free Encyclopedia. Retrieved 21:30, November 30, 2021, from https://en.wikipedia.org/w/index.php?title=Adjacency_matrix&oldid=1048154108

Wikipedia contributors. (2021, October 29). Graph (abstract data type). In Wikipedia, The Free Encyclopedia. Retrieved 21:35, November 30, 2021, from https://en.wikipedia.org/w/index.php?title=Graph_(abstract_data_type)&oldid=1052445998