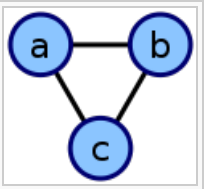
# Adjacency list

From Wikipedia, the free encyclopedia

In graph theory and computer science, an **adjacency list** representation of a graph is a collection of unordered lists, one for each vertex in the graph. Each list describes the set of neighbors of its vertex. See "Storing a sparse matrix" for an alternative approach.



This undirected cyclic graph can be described by the list {a,b}, {a,c}, {b,c}.

**Contents** [hide]
1 Implementation details
2 Operations
3 Trade-offs
4 Data structures
5 References
6 Further reading
7 External links

## Implementation details [edit]

An adjacency list representation for a graph associates each vertex in the graph with the collection of its neighboring vertices or edges. There are many variations of this basic idea, differing in the details of how they implement the association between vertices and collections, in how they implement the collections, in whether they include both vertices and edges or only vertices as first class objects, and in what kinds of objects are used to represent the vertices and edges.

| The graph pictured above has this adjacency list representation: | | |
|---|---|---|
| a | adjacent to | b,c |
| b | adjacent to | a,c |
| c | adjacent to | a,b |

- An implementation suggested by Guido van Rossum uses a hash table to associate each vertex in a graph with an array of adjacent vertices. In this representation, a vertex may be represented by any hashable object. There is no explicit representation of edges as objects.[1]
- Cormen et al. suggest an implementation in which the vertices are represented by index numbers.[2] Their representation uses an array indexed by vertex number, in which the array cell for each vertex points to a singly linked list of the neighboring vertices of that vertex. In this representation, the nodes of the singly linked list may be interpreted as edge objects; however, they do not store the full information about each edge (they only store one of the two endpoints of the edge) and in undirected graphs there will be two different linked list nodes for each edge (one within the lists for each of the two endpoints of the edge).
- The object oriented **incidence list** structure suggested by Goodrich and Tamassia has special classes of vertex objects and edge objects. Each vertex object has an instance variable pointing to a collection object that lists the neighboring edge objects. In turn, each edge object points to the two vertex objects at its endpoints.[3] This version of the adjacency list uses more memory than the version in which adjacent vertices are listed directly, but the existence of explicit edge objects allows it extra flexibility in storing additional information about edges.

## Operations [edit]

The main operation performed by the adjacency list data structure is to report a list of the neighbors of a given vertex. Using any of the implementations detailed above, this can be performed in constant time per neighbor. In other words, the total time to report all of the neighbors of a vertex *v* is proportional to the degree of *v*.

It is also possible, but not as efficient, to use adjacency lists to test whether an edge exists or does not exist between two specified vertices. In an adjacency list in which the neighbors of each vertex are unsorted, testing for the existence of an edge may be performed in time proportional to the degree of one of the two given vertices, by using a sequential search through the neighbors of this vertex. If the neighbors are represented as a sorted array, binary search may be used instead, taking time proportional to the logarithm of the degree.

## Trade-offs [edit]

The main alternative to the adjacency list is the adjacency matrix, a matrix whose rows and columns are indexed

by vertices and whose cells contain a Boolean value that indicates whether an edge is present between the vertices corresponding to the row and column of the cell. For a sparse graph (one in which most pairs of vertices are not connected by edges) an adjacency list is significantly more space-efficient than an adjacency matrix (stored as an array): the space usage of the adjacency list is proportional to the number of edges and vertices in the graph, while for an adjacency matrix stored in this way the space is proportional to the square of the number of vertices. However, it is possible to store adjacency matrices more space-efficiently, matching the linear space usage of an adjacency list, by using a hash table indexed by pairs of vertices rather than an array.

The other significant difference between adjacency lists and adjacency matrices is in the efficiency of the operations they perform. In an adjacency list, the neighbors of each vertex may be listed efficiently, in time proportional to the degree of the vertex. In an adjacency matrix, this operation takes time proportional to the number of vertices in the graph, which may be significantly higher than the degree. On the other hand, the adjacency matrix allows testing whether two vertices are adjacent to each other in constant time; the adjacency list is slower to support this operation.

## Data structures [edit]

For use as a data structure, the main alternative to the adjacency list is the adjacency matrix. 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 of contiguous space. Besides avoiding wasted space, this compactness encourages locality of reference.

However, for a sparse graph, adjacency lists require less storage space, because they do not waste any space to represent edges that are not present. Using a naïve array implementation on a 32-bit computer, an adjacency list for an undirected graph requires about $8|E|$ bytes of storage.

Noting that a simple graph can have at most $|V|^2$ edges, allowing loops, we can let $d = |E|/|V|^2$ denote the density of the graph. Then, $8|E| > |V|^2/8$, or the adjacency list representation occupies more space precisely when $d > 1/64$. Thus a graph must be sparse indeed to justify an adjacency list representation.

Besides the space tradeoff, the different data structures also facilitate different operations. Finding all vertices adjacent to a given vertex in an adjacency list is as simple as reading the list. With an adjacency matrix, an entire row must instead be scanned, which takes $O(|V|)$ time. Whether there is an edge between two given vertices can be determined at once with an adjacency matrix, while requiring time proportional to the minimum degree of the two vertices with the adjacency list.

## References [edit]


1. ^ Guido van Rossum (1998). "Python Patterns — Implementing Graphs".
2. ^ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2001). *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill. pp. 527–529 of section 22.1: Representations of graphs. ISBN 0-262-03293-7.
3. ^ Michael T. Goodrich and Roberto Tamassia (2002). *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons. ISBN 0-471-38365-1.


## Further reading [edit]


- David Eppstein (1996). "ICS 161 Lecture Notes: Graph Algorithms".


## External links [edit]

- The Boost Graph Library implements an efficient adjacency list
- Open Data Structures - Section 12.2 - AdjacencyList: A Graph as a Collection of Lists

Categories: Graph data structures