Theory: Representation of graphs

12 minutes 9 / 9 problems solved

Start practicing

We demonstrated that graphs provide a visual approach to represent and solve a variety of problems, yet particularly large problems can be very difficult to solve manually. Fortunately, your computer can help if you present your problem as a graph in the computer memory and implement a specific algorithm to solve it. But how can this be presented in computer memory?

§1. Adjacency matrix

The first way is known as the **adjacency matrix**. This is a matrix in which each entry indicates whether a particular edge exists in the graph. Let's look at a specific example. Say we have a graph like this:

Here is the adjacency matrix for it:

The value 1 in a cell means that the edge exists in the graph, while 0 means there's no such edge. This matrix allows you to check whether a node is connected to another one in constant time $O(1)$.

For undirected graphs, the adjacency matrix is always symmetric: $A[i, j] = A[j, i]$. Since we don't allow edges from a node to itself, the diagonal elements $A[i, i]$ are all zeros, but it is possible in the common case. The adjacency matrix for directed graphs is not symmetric.

When we have an adjacency matrix, we can decide whether two nodes are connected by an edge just by looking at the appropriate slot in the matrix. We can also list all the neighbors of a node by scanning the corresponding row (or column).

The graph representation in the form of an adjacency matrix needs $O(V^2)$ memory where V is a number of nodes in computer memory. Such a representation is especially well suited for graphs where the number of edges is close to the maximum (dense graphs).

§2. Adjacency lists

Another common way to represent a graph in the computer's memory is adjacency lists where each list stores the neighbors of the corresponding node.

Here are adjacency lists for the graph above:

For undirected graphs, each edge is stored twice (like 'a'→'b' and 'b→'a' in the top example). For directed graphs, each edge is stored only once; if the direction in our initial graph (see top image) was from 'b' to 'a', then the first adjacency list would be 'a'→'c'.

Graph representation in the form of an adjacency list needs $O(V + E)$ computer memory. It is more effective than the adjacency matrix for sparse graphs in which the number of edges is much less than the number of nodes in the square ($E << V^2$).

However, this representation has a drawback that can be serious in some types of problems. We cannot check whether one node is connected to another one without passing through the elements of an adjacency list. This is much more difficult than simply using the adjacency matrix.

§3. Other representations

We considered two common ways to represent a graph in computer memory, but there are more options. The simplest one is a list of all edges in a graph: 'a'→'b', 'a'→'c', 'b'→'c', and so on. Some problems may also need a more specific graph representation for efficiency.

Report a typo

😐 Thanks for your feedback!

Write here how we could improve this theory

Start practicing

This content was created almost 2 years ago and updated 7 days ago. Share your feedback below in comments to help us improve it!

Comments (1) Hints (0) Useful links (1) Show discussion