An introduction to Graphs

Formulating a simple, precise specification of a computational problem is often a prerequisite to writing a computer program for solving the problem. Many computational problems are best stated in terms of graphs: A directed graph $G(V,E)$ consists of a finite set of vertices V and a set of (directed) edges or arcs E. An arc is an ordered pair of vertices (v,w) and is usually indicated by drawing a line between v and w, with an arrow pointing towards w. Stated in mathematical terms, a directed graph $G(V,E)$ is just a binary relation $E \subseteq V \times V$ on a finite set V. Undirected graphs may be regarded as special kinds of directed graphs, such that $(u,v) \in E \leftrightarrow (v,u) \in E$. Thus, since the directions of the edges are unimportant, an undirected graph $G(V,E)$ consists of a finite set of vertices V, and a set of edges E, each of which is an unordered pair of vertices $\{u,v\}$.

Graphs are useful for modeling a diverse number of situations. For example, the vertices of a graph might represent cities, and edges might represent highways that connect them. In this case, each edge might also have an associated length. Alternatively, an edge might represent a flight from one city to another, and each edge might have a weight which represents the cost of the flight. The typical problem in this context is computing shortest paths: given that you wish to travel from city X to city Y, what is the shortest path (or the cheapest flight schedule). There are very efficient algorithms for solving these problems. A different kind of problem — the traveling salesman problem — is very hard to solve. Suppose a traveling salesman wishes to visit each city exactly once and return to his starting point, in which order should he visit the cities to minimize the total distance travelled? This is an example of an NP-complete problem, and one we will study towards the end of this course.

A different context in which graphs play a critical modeling role is in networks of pipes or communication links. These can, in general, be modeled by directed graphs with capacities on the edges. A directed edge from u to v with capacity c might represent a pipeline that can carry a flow of at most c units of oil per unit time from u to v. A typical problem in this context is the max-flow problem: given a network of pipes modeled by a directed graph with capacities on the edges, and two special vertices — a source s and a sink t — what is the maximum rate at which oil can be transported from s to t over this network of pipes? There are ingenious techniques for solving these types of flow problems.

In all the cases mentioned above, the vertices and edges of the graph represented something quite concrete such as cities and highways. Often, graphs will be used to represent more abstract relationships. For example, the vertices of a graph might represent tasks, and the edges might represent precedence constraints: a directed edge from u to v says that task u must be completed before v can be started. An important problem in this context is scheduling: in what order should the tasks be scheduled so that all the precedence constraints are satisfied.

Hypercubes

Recall that the set of all n-bit strings is denoted by $\{0,1\}^n$. The n-dimensional hypercube is a graph whose vertex set is $\{0,1\}^n$ (i.e. there are exactly 2^n vertices, each labeled with a distinct n-bit string), and with an edge between vertices x and y iff x and y differ in exactly one bit position. i.e. if $x = x_1x_2\ldots x_n$ and $y = y_1y_2\ldots y_n$, then there is an edge between x and y iff there is an i such that $\forall j \neq i, x_j = y_j$ and $x_i \neq y_i$.

There is another equivalent recursive definition of the hypercube:

The n-dimensional hypercube consists of two copies of the $n-1$-dimensional hypercube (the 0-subcube and the 1-subcube), and with edges between corresponding vertices in the two subcubes. i.e. there is an edge between vertex x in the 0-subcube (also denoted as vertex $0x$) and vertex x in the 1-subcube.

Claim: The total number of edges in an n-dimensional hypercube is $n2^{n-1}$.

Proof: Each vertex has n edges incident to it, since there are exactly n bit positions that can be toggled to get an edge. Since each edge is counted twice, once from each endpoint, this yields a grand total of $n2^n/2$.

Alternative Proof: By the second definition, it follows that $E(n) = 2E(n-1) + 2^n$, and $E(1) = 1$. A straightforward induction shows that $E(n) = n2^{n-1}$.

We will prove that the n-dimensional hypercube is a very robust graph in the following sense: consider how many edges must be cut to separate a subset S of vertices from the remaining vertices $V - S$. Assume that S is the smaller piece; i.e. $|S| \leq |V - S|$.

Theorem: $|E_{S,V-S}| \geq |S|$.

Proof: By induction on n. Base case $n = 1$ is trivial.

For the induction step, let S_0 be the vertices from the 0-subcube in S, and S_1 be the vertices in S from the 1-subcube.

Case 1: If $|S_0| \leq 2^{n-1}/2$ and $|S_1| \leq 2^{n-1}/2$ then applying the induction hypothesis to each of the subcubes shows that the number of edges between S and $V - S$ even without taking into consideration edges that cross between the 0-subcube and the 1-subcube, already exceed $|S_0| + |S_1| = |S|$.

Case 2: Suppose $|S_0| > 2^{n-1}/2$. Then $|S_1| \leq 2^{n-1}/2$. But now $|E_{S,V-S}| \geq 2^n - 1 \geq |S|$. This is because by the induction hypothesis, the number of edges in $E_{S,V-S}$ within the 0-subcube is at least $2^{n-1} - |S_0|$, and those within the 1-subcube is at least $|S_1|$. But now there must be at least $|S_0| - |S_1$ edges in $E_{S,V-S}$ that cross between the two subcubes (since there are edges betwen every pair of corresponding vertices. This is a grand total of $2^{n-1} - |S_0| + |S_1| + |S_0| - |S_1| = 2^{n-1}$.