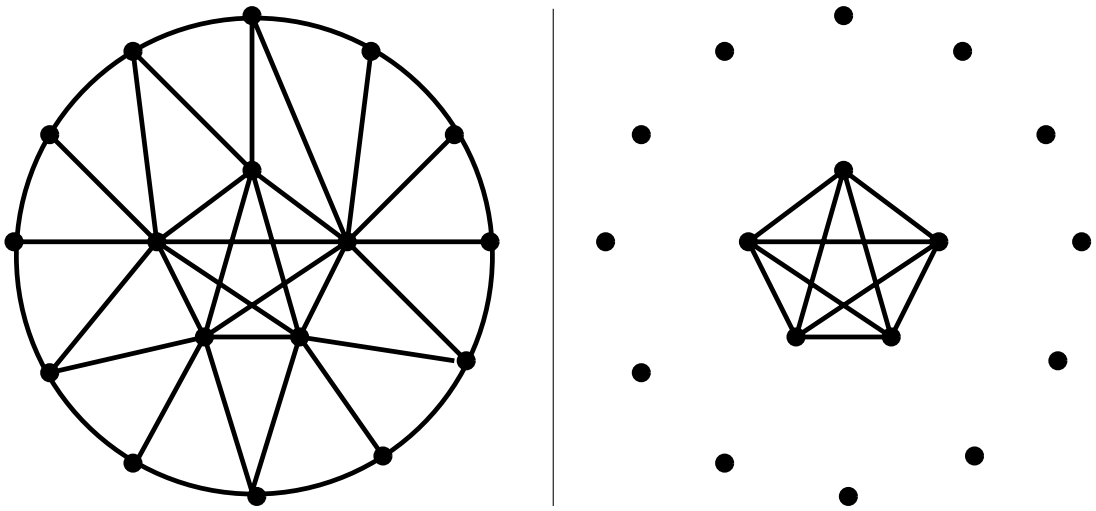

INPUT OUTPUT

16.1 Clique

Input description: A graph $G = (V, E)$.

Problem description: What is the largest $S \subset V$ such that for all $x, y \in S$, $(x, y) \in E$?

Discussion: In high school, everybody complained about the "clique,"—a group of friends who all hung around together and seemed to dominate everything social. Consider a graph representing the school's social network. Vertices correspond to people, with edges between any pair of people who are friends. Thus, the high school clique defines a (complete subgraph) clique in this friendship graph.

Identifying "clusters" of related objects often reduces to finding large cliques in graphs. An interesting example arose in a program the Internal Revenue Service (IRS) developed to detect organized tax fraud. In this scam, large groups of phony tax returns are submitted in the hopes of getting undeserved refunds. But generating large numbers of *different* phony tax returns is hard work. The IRS constructs graphs with vertices corresponding to submitted tax forms and edges between any two forms that appear suspiciously similar. Any large clique in this graph points to fraud.

Since any edge in a graph represents a clique of two vertices, the challenge lies not in finding a clique, but in finding a large clique. And it is indeed a challenge, for finding a maximum clique is NP-complete. To make matters worse, it is provably

hard to approximate even to within a factor of $n^{1/2-\epsilon}$. Theoretically, clique is about as hard as a problem in this book can get. So what can we hope to do about it?

- *Will a maximal clique suffice?* – A *maximal* clique is a clique that cannot be enlarged by adding any additional vertex. This doesn't mean that it has to be large relative to the largest possible clique, but it might be. To find a nice maximal (and hopefully large) clique, sort the vertices from highest degree to lowest degree, put the first vertex in the clique, and then test each of the other vertices to see whether it is adjacent to all the clique vertices added thus far. If so, add it; if not, continue down the list. By using a bit vector to mark which vertices are currently in the clique, this can be made to run in $O(n+m)$ time. An alternative approach might incorporate some randomness into the vertex ordering, and accept the largest maximal clique you find after a certain number of trials.

- *What if I will settle for a large, dense subgraph?* – Insisting on cliques to define clusters in a graph can be risky, since a single missing edge will eliminate a vertex from consideration. Instead, we should seek large *dense* subgraphs—i.e. , subsets of vertices that contain a large number of edges between them. Cliques are, by definition, the densest subgraphs possible.

  The largest set of vertices whose induced (defined) subgraph has minimum vertex degree $\geq k$ can be found with a simple linear-time algorithm. Begin by deleting all the vertices whose degree is less than k. This may reduce the degree of other vertices below k, if they were adjacent to sufficiently deleted low-degree vertices. Repeating this process until all remaining vertices have degree $\geq k$ constructs the largest high-degree subgraph. This algorithm can be implemented in $O(n+m)$ time by using adjacency lists and the constant-width priority queue of Section 12.2 (page 373). If we continue to delete the lowest-degree vertices, we eventually end up with a clique or set of cliques, – but they may be as small as two vertices.

- *What if the graph is planar?* – Planar graphs cannot have cliques of a size larger than four, or else they cease to be planar. Since each edge defines a clique of size 2, the only interesting cases are cliques of three and four vertices. Efficient algorithms to find such small cliques consider the vertices from lowest to highest degree. Any planar graph must contain a vertex of at most 5 degrees (see Section 15.12 (page 520)), so there is only a constant-sized neighborhood to check exhaustively for a clique containing it. We then delete this vertex to leave a smaller planar graph, containing another low-degree vertex. Repeat this check and delete processes until the graph is empty.

If you *really* need to find the largest clique in a graph, an exhaustive search via backtracking provides the only real solution. We search through all k-subsets of the vertices, pruning a subset as soon as it contains a vertex that is not adjacent to all the rest. A simple upper bound on the maximum clique in G is the highest vertex

degree plus 1. A better upper bound comes from sorting the vertices in order of decreasing degree. Let j be the largest index such that degree of the jth vertex is at least $j-1$. The largest clique in the graph contains no more than j vertices, since no vertex of degree $< (j-1)$ can appear in a clique of size j. To speed our search, we should delete all such useless vertices from G.

Heuristics for finding large cliques based on randomized techniques such as simulated annealing are likely to work reasonably well.

Implementations: `Cliquer` is a set of C routines for finding cliques in arbitrary weighted graphs by Patric Östergard. It uses an exact branch-and-bound algorithm, and is available at *http://users.tkk.fi/~pat/cliquer.html*.

Programs for finding cliques and independent sets were sought for the Second DIMACS Implementation Challenge [JT96]. Programs and data from the challenge are available by anonymous FTP from *dimacs.rutgers.edu*. Source codes are available under *pub/challenge/graph* and test data under *pub/djs*. `dfmax.c` implements a simple-minded branch-and-bound algorithm similar to [CP90]. `dmclique.c` uses a "semi-exhaustive greedy" scheme for finding large independent sets from [JAMS91].

Kreher and Stinson [KS99] provide branch-and-bound programs in C for finding the maximum clique using a variety of lower-bounds, available at *http://www.math.mtu.edu/~kreher/cages/Src.html*.

GOBLIN (*http://www.math.uni-augsburg.de/~fremuth/goblin.html*) implements branch-and-bound algorithms for finding large cliques. They claim to be able to work with graphs as large as 150 to 200 vertices.

Notes: Bomze, et al. [BBPP99] give the most comprehensive survey on the problem of finding maximum cliques. Particularly interesting is the work from the operations research community on branch-and-bound algorithms for finding cliques effectively. More recent experimental results are reported in [JS01].

The proof that clique is NP-complete is due to Karp [Kar72]. His reduction (given in Section 9.3.3 (page 327)) established that clique, vertex cover, and independent set are very closely related problems, so heuristics and programs that solve one of them should also produce reasonable solutions for the other two.

The *densest subgraph problem* seeks the subset of vertices whose induced subgraph has the highest average vertex degree. A clique of k vertices is clearly the densest subgraph of its size, but larger, noncomplete subgraphs may achieve higher average degree. The problem is NP-complete, but simple heuristics based on repeatedly deleting the lowest-degree vertex achieve reasonable approximation ratios [AITT00]. See [GKT05] for an interesting application of densest subgraph, namely detecting link spam on the Web.

That clique cannot be approximated to within a factor of $n^{1/2-\epsilon}$ unless $P = NP$ (and $n^{1-\epsilon}$ under weaker assumptions) is shown in [Has82].

Related Problems: Independent set (see page 528), vertex cover (see page 530).