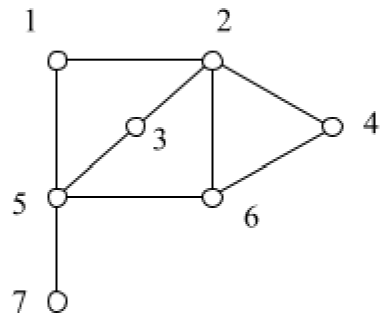
Graph Problems

Problem #1: Given a set of cell towers, assign a minimum number of frequencies to the cell towers so that towers within the same range are assigned different frequencies.

Vertex Coloring
Given a graph G, color the vertices of G using the **minimum** number of colors so that adjacent vertices are assigned different colors. The **chromatic number** of a graph G, denoted $\chi(G)$, is the minimum number of colors needed to color the vertices in this way.



Applications: Scheduling problems, register allocation, map coloring.

1. State the Vertex Coloring problem as a decision problem by introducing a bound k. This decision problem is called the k-Colorability problem.

   k-Colorability Problem
   Given: A graph G, and an upper bound *k*
   Problem: Is $\chi(G) \leq k$? That is, is there an assignment of at most *k* colors to the vertices of G so that adjacent vertices receive different colors?

2. Show that the k-Colorability problem is in the class NP.

   Assume `a` is the adjacency matrix of the graph G and that `k` is a positive integer. A guess will be an assignment of randomly chosen integers (representing colors) in the range 1, 2, …, k, stored in an array `color`, where `color[i]` will represent the color of vertex `i` in the graph. You must check that whenever two vertices are adjacent in the graph, the two vertices are assigned different colors. (Fill in the details on the next page.)

   Determine the running time of your algorithm and show that it is polynomial.

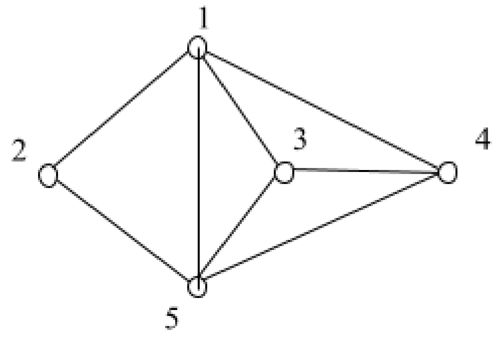
```
kColorVerify(a, k)   // a is the adjacency matrix
{
  for i = 1 to n
    color[i] = ______________________

  // go through the adjacency matrix  a  and
  // check that adjacent vertices received different colors
```

3. The k-Colorability problem is NP-hard because there is a polynomial-time reduction from another NP-complete problem called Satisfiability. What can you conclude?

4. NOTE: A breadth first search algorithm can be modified to determine if a graph is 2-colorable. So, for the special case of $k = 2$, this problem is in which class?

Problem #2: In a given set of users on Facebook, find the largest group of users among them such that every pair of users is friends with each other.

Maximum Clique
Given a graph G, find the **largest** subset S of vertices in G such that every pair of vertices of S is adjacent. Such a set of vertices forms a complete subgraph of G and is called a **clique**. The **clique number** of a graph G, denoted ω(G), is the order (number of vertices) in a largest clique of G.



Applications: Social network analysis, clustering gene expressions, data mining.

5. State the Maximum Clique problem as a decision problem by introducing a bound B.

   Maximum Clique Problem
   Given: A graph G, and ______________________________ *B*
   Problem:

6. Show that the Maximum Clique problem is in the class NP.

   Assume `a` is the adjacency matrix of the graph G and that `B` is a positive integer. A guess will be a random selection S of vertices (1, 2, …, *n*). This could be done in a number of ways, but one possibility is to randomly guess {true, false} for each vertex to indicate whether or not that vertex was chosen to be in the set S. Then you must check that there are at least B vertices in the guess S, and that the vertices of S form a clique, i.e., every pair is adjacent. (Fill in the details on the next page.)

   Determine the running time of your algorithm and show that it is polynomial.

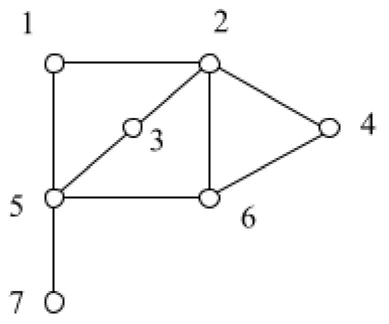
```
maximumCliqueVerify(a, B)
{
  // the following constructs a guess S
  // check that there are at least  B  vertices in  S

  count = 0
  for i = 1 to n
  {
    S[i] = guess{true, false}  // true ⇔ vertex i is in S



  // check that all pairs of vertices in S are adjacent
```

7. The Maximum Clique problem is NP-hard because there is a polynomial-time reduction from another NP-complete problem called Vertex Cover. What can you conclude?

   NOTE: Uriel Fage has written an approximation algorithm that finds a clique with a number of vertices within a certain factor of the maximum. It is the best known approximation to date.

Problem #3: A set of security cameras are to be placed at various intersections so that all roads are able to be viewed by at least one of the security cameras. Determine the minimum number of security cameras needed.

Vertex Cover
Given a graph G, determine a **smallest** subset S of vertices of G such that every edge in G is incident with at least one vertex of S.



Applications: Genetic algorithms, facility location, Boolean logic minimization.

8. State the Vertex Cover problem as a decision problem by introducing a bound B.

   Vertex Cover Problem
   Given: A graph G, and ______________________________ *B*
   Problem:

9. Show that the Vertex Cover problem is in the class NP.

   Assume `adj` is the adjacency list representation of the graph G and that `B` is a positive integer. A guess will be a random selection S of vertices (1, 2, …, *n*). This could be done in a number of ways, but one possibility is to randomly guess {true, false} for each vertex to indicate whether or not that vertex was chosen to be in the set S. Then you must check that there are at most B vertices in the guess S, and that every edge of G is incident with at least one vertex of S. (Fill in the details on the next page.)

   Determine the running time of your algorithm and show that it is polynomial.

```
vertexCoverVerify(adj, B)
{
  // randomly select vertices and store in S
  // check that there are at most  B  vertices in  S

  count = 0
  for i = 1 to n
  {
    S[i] = guess{true, false}  // true = vertex i is in S
```

```
  // check that every edge in G is incident with at least
  // one vertex of  S

  for i = 1 to n
  {
    // If vertex i belongs to S, then all its incident edges
    // are covered; if not, process i's list by checking to
    // see if each adjacent vertex is in S.

      trav = adj[i]
      while (trav != null)
      {
```

10. The Vertex Cover problem is NP-hard because there is a polynomial-time reduction from another NP-complete problem called 3-Satisfiability. What can you conclude?

NOTE: Fanica Gavril and Mihalis Yannakakis independently discovered this approximation algorithm for Vertex Cover: Pick an arbitrary edge. Include both incident vertices in the set S. Delete all edges covered so far and repeat. Continue until there are no uncovered edges. Using this algorithm, the number of vertices in the vertex cover S is at most twice the minimum possible. So the solution is within a factor of 2 of the optimal solution, and we say it is a 2-approximation.