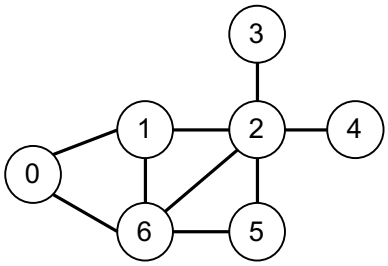
1. **グラフ構造**とは，右図のようにノード（または頂点）がリンク（または辺，エッジ）によって連結された構造である。木もグラフの一種である。数学ではグラフの連結の表現として，**隣接行列**が用いられる。これは，グラフGにおいて行列Aを定義し，Aの成分a_{ij}の値をノードv_iとv_jを直接つなぐリンクの本数とするものである。隣接行列はプログラム言語では 2 次元配列で実現できる。グラフ構造のデータ表現には，他にも木構造のように構造体とポインタを用いる方法などがある。



　下記のプログラムは，右図のグラフを隣接行列で表し，単純な深さ優先探索を用いて指定されたホップ数以内で到達できるノードを列挙するものである。空欄を適切に埋めてプログラムを入力して実行せよ。

```
import java.util.*;

class Graph {

   // 隣接行列：ノード i と j が直接つながっていれば
   // adj[i][j]==1，そうでなければ 0 を設定する
   private final static int[][] adj = {
      {   ,   ,   ,   ,   ,   ,   },
      {   ,   ,   ,   ,   ,   ,   },
      {   ,   ,   ,   ,   ,   ,   },
      {   ,   ,   ,   ,   ,   ,   },
      {   ,   ,   ,   ,   ,   ,   },
      {   ,   ,   ,   ,   ,   ,   },
      {   ,   ,   ,   ,   ,   ,   }};

   // グラフのノード数
   private final static int N = adj.length;

   static void traverse(int n, int hops) {
      if (hops < 0) return;
      System.out.println(n);
      for (int i = 0; i < N; i++) {
         if (adj[n][i] == 1)
            traverse(i, hops - 1);
      }
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.printf("始点 (0-%d): ", N-1);
      int start = sc.nextInt();
      System.out.print("最大ホップ数: ");
      int hops = sc.nextInt();

      traverse(sc, hops);
   }
}
```

2. Java では，標準の「コレクションフレームワーク」において基礎的なデータ構造とアルゴリズムが提供されており，パッケージ `java.util` では，ジェネリックインタフェースで以下のようなデータ構造がモデル化されている。

インタフェース	説明
`List<E>`	要素を先頭から順にアクセスできる一般的なリストを表すインタフェース 主なメソッド： `.add(data) .get(index) .indexOf(data) .size() .clear()`
`Set<E>`	要素の重複を許さずにデータを格納する集合を表すインタフェース 主なメソッド： `.add(data) .contains(data) .remove(data) .isEmpty() .size()`
`Map<K, V>`	キーと値のペアのデータを格納するマップ（写像）を表すインタフェース 主なメソッド： `.put(key, value) .get(key) .remove(key) .size() .values()`

そして，これらのインタフェースを典型的な手法で実装したのが，以下のジェネリッククラスである。

クラス	説明
`ArrayList<E>`	内部構造が配列によって実現された `List` の実装（添字によるアクセスが高速）
`LinkedList<E>`	内部構造が連結リストによって実現された `List` の実装（`.push(data) .pop()` 等を実装）
`HashSet<E>`	内部構造がハッシュテーブルによって実現された `Set` の実装
`TreeSet<E>`	内部構造が 2 分探索木によって実現した `Set` の実装
`HashMap<K, V>`	内部構造がハッシュテーブルによって実現された `Map` の実装
`TreeMap<K, V>`	内部構造が 2 分探索木によって実現した `Map` の実装

これらのうち特に `Set` と `Map` は，なるべく `Map<String, String> map = new HashMap<String, String>();` のようにインタフェースを介して使用される。こうすれば，実装クラスを変更してもコードの修正が最小限になる。

(a) クラス `LinkedList<String>`のインスタンスにいくつかの文字列を登録（追加）し，最後に全要素を表示するプログラムを作成せよ。`for (String s : list)` のような書き方を用いるとよい。

(b) クラス `HashMap<String, String>`を用いて，キーボードから国名を入力するとその首都名を表示するプログラムを作成せよ。10 カ国程度登録すればよい。`HashMap` の使い方は Web サイト等を参考にせよ。

(c) クラス `TreeMap<String, Integer>`を用いて，キーボードから単語を 1 語ずつ読み込んで出現回数を数えるプログラムを作成せよ。最後に全要素を表示する方法を調べてみよ。