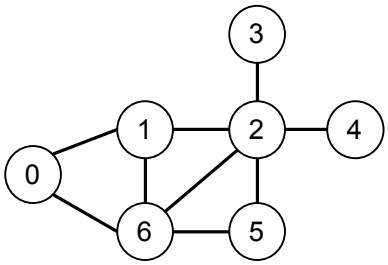
1. **グラフ構造**とは，右図のようにノード（または頂点）がリンク（または辺，エッジ）によって連結された構造である。木もグラフの一種である。



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，あるノードから他のノードへの最短距離とその経路を**ダイクストラ法**というアルゴリズムによって求めるものである。空欄を適切に埋めてプログラムを入力して実行し，説明を読んでアルゴリズムについて理解せよ。

```
import java.util.*;

class Dijkstra {

   // 距離無限大を表す十分に大きな数
   private final static int INF = 9999;

   // 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;

   public static void main(String[] args) {
      // 始点ノードの入力
      Scanner sc = new Scanner(System.in);
      System.out.printf("始点 (0-%d): ", N-1);
      int start = sc.nextInt();

      // 始点ノードから各ノードまでの最短距離
      int[] dist = new int[N];
      // 各ノードへの経路の 1 つ前の経由ノード
      int[] back = new int[N];

      // 始点からの距離が未確定のノードの集合
      //  （コレクションクラスの集合を利用した）
      Set<Integer> Q = new HashSet<Integer>();

      // 始点から全ノードへの距離を無限大に初期化
      for (int node = 0; node < N; node++) {
         dist[node] = INF;
         Q.add(node);  // 最初は全ノードが未確定
      }

      dist[start] = 0;   // 始点→始点は距離 0
      back[start] = -1;  // 始点の 1 つ前はない

      // 距離未確定ノードがある間，繰り返す
      while (Q.size() > 0) {
         // まだ距離が未確定にノードの中で
         // 始点に一番近いものを選び出す
         int nearest = -1;    // 一番近いノード
         int min_dist = INF;  // その距離

         // Q の全要素のループ
         for (int node : Q) {
            if (dist[node] <            ) {

                        = dist[node];

                        = node;
            }
         }
         // Q の残りが到達できないノードだけなら終了
         if (min_dist == INF) break;

         // 距離が最短だったノード（nearest）へは，
         // もうこれ以上短い経路は存在し得ない
         // よって，Q から取り除き，距離を確定する
         Q.remove(nearest);

         // 距離未確定の全ノードについて，nearest を
         // 通った場合の距離を計算し，今までよりも
         // 短くなるなら距離と経由ノードを更新する
         for (int node : Q) {
            // nearest→node の接続があるなら
            if (adj[nearest][node] > 0) {
               // 現在の node への最短距離よりも
               // nearest への距離+1 のほうが短ければ
               if (dist[node] >             + 1) {

                  dist[node] =

                  back[node] =
               }
            }
         }
      }

      // 最後に始点から全ノードへの距離を表示
      for (int node = 0; node < N; node++) {
         System.out.printf("%d->%d 距離:%d 経路:",
                           start, node, dist[node]);
         // 経路を逆にたどって表示
         for (int x = node; x != -1; x = back[x])
            System.out.print(x + " ");
         System.out.println();
      }
   }
}
```