**LK-Q1-VI**

1. **Analyse von Graphen in verschiedenen Kontexten**

**(a) Grundlegende Begriffe**

**Beispiel (a): Haus des Nikolaus**

|  |  |
| --- | --- |
|  | Wie viele Möglichkeiten gibt es, das Haus des Nikolaus zu zeichnen?  Das Haus des Nikolaus ist ein Zeichenrätsel. Ziel ist es, das spezielle Haus in einem Linienzug zu zeichnen, ohne eine der Strecken mehrfach zu durchlaufen.  Begleitet wird das Zeichnen durch folgenden gesprochenen Reim:  „Das ist das Haus vom Ni-ko-laus“ |

Mathematisch gesehen handelt es sich um ein Problem der Graphentheorie.

Von einem Knoten aus werde alle Kanten nach einer gewissen Vorschrift durchlaufen.

|  |  |  |  |
| --- | --- | --- | --- |
| 120830_NikoHaus0 | Das Haus des Nikolaus als Prinzipzeichnung | 120830_NikoHaus2 | Die Ecken kann man als Knoten und die Linien als Kanten  eines Graphen interpretieren. |

**Grundlegende Definition**

Ein **ungerichteter** **Graph** besteht aus einer Menge V von Knoten (Vertices) und einer Menge E von Kanten (Edges). Eine Kante ist gegeben durch ein ungeordnetes Paar von Knoten.

Ein **gerichteter** **Graph** besteht aus einer Menge V von Knoten (Vertices) und einer Menge E von gerichteten Kanten (Edges). Eine gerichtete Kante ist gegeben durch ein geordnetes Paar von Knoten.

Haus des Nikolaus

Knotenmenge E = { A, B, C, D, E }

Kantenmenge K = { (A/B), (A/C), (A/E), (B/C), (B/E), (C/D), (C/E), (D/E) }

Bei einem ungerichteten Graphen bezeichnen (A/B) und (B/A) dieselbe Kante.

**Datenstrukturen für Graphen**

Adjazenzmatrix

Die einfachste Art der Darstellung eines Graphen ist die Darstellung als **nxn**-Matrix. Hierbei steht der Eintrag **aij = 1**, falls es eine Kante von **i** nach **j** gibt, und **aij = 0**, falls keine Kante von **i** nach **j** existiert.

Die folgende Tabelle stellt die Adjazenzmatrix für den Graphen „Haus des Nikolaus“ dar:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | 0 | 1 | 1 | 0 | 1 |
| B | 1 | 0 | 1 | 0 | 1 |
| C | 1 | 1 | 0 | 1 | 1 |
| D | 0 | 0 | 1 | 0 | 1 |
| E | 1 | 1 | 1 | 1 | 0 |

Für die Adjazenzmatrix werden unabhängig von der Anzahl der Kanten **n2** Speicherplätze benötigt. In einem ungerichteten Graphen ist die Adjazenzmatrix symmetrisch zur Hauptdiagonalen.

**Aufgabe**

Die folgende Klasse Graph implementiert einen Graphen in Form einer AdjazenzMatrix:

1. Analysieren Sie die Klasse Graph und dokumentieren Sie die Klasse.
2. Erzeugen Sie mit der Klasse Graph das Objekt hausDesNikolaus.
3. Implementieren Sie die Methode istRundweg, die genau dann true liefert, wenn die Knotenfolge, die in dem String-Objekt, das als Parameter übergeben wird, ein Rundweg im Graphen ist. (Es muss nicht geprüft werden, ob alle Knoten besucht wurden).
4. Testen Sie die Methode mit dem Graphen hausDesNikolaus anhand verschiedener Wege.

**public** **class** Graph {

**private** **final** **int** anzahl = 5;

**private** **char**[] knoten = **new** **char**[anzahl];

**private** **boolean**[][] adjazenzMatrix = **new** **boolean**[anzahl][anzahl];

**private** **int** anzahlKnoten = 0;

**public** Graph() {

**for** (**int** i = 0; i<anzahl; i++){

knoten[i] = ' ';

}

**for** (**int** i = 0; i<anzahl; i++){

**for** (**int** j = 0; j < anzahl; j++){

adjazenzMatrix[i][j] = **false**;

}

}

}

**public** **void** knotenHinzu(**char** k){

knoten[anzahlKnoten] = k;

anzahlKnoten++;

}

**public** **int** sucheKnotenIndex(**char** zeichen){

**int** kIndex = -1;

**int** zaehler = 0;

**while** (kIndex == -1 && zaehler < anzahl){

**if** (knoten[zaehler] == zeichen){

kIndex = zaehler;

} **else** {

zaehler++;

}

}

**return** kIndex;

}

**public** **void** kanteHinzu(**char** k1, **char** k2){

**int** indexK1 = sucheKnotenIndex(k1);

**int** indexK2 = sucheKnotenIndex(k2);

**if** (indexK1 != -1 && indexK2 != -1) {

adjazenzMatrix[indexK1][indexK2] = **true**;

adjazenzMatrix[indexK2][indexK1] = **true**;

}

}

**public** **boolean** hatKante(**char** k1, **char** k2){

**int** indexK1 = sucheKnotenIndex(k1);

**int** indexK2 = sucheKnotenIndex(k2);

**if** (indexK1 != -1 && indexK2 != -1) {

**return** adjazenzMatrix[indexK1][indexK2];

} **else** {

**return** **false**;

}

}

**public** **boolean** istRundWeg(String weg){

}

}

**Literatur:**

Vöcking B. u.a. (2008) u.a.. Taschenbuch der Algorithmen, S. 29 Die Eulertour