J2 Kassiopeia

Die Schildkröte Kassiopeia will wissen, ob sie in Quadratien jedes weiße Feld erreichen kann.

Dazu sollte zuerst bemerkt werden, dass für die Antwort unerheblich ist, wo die Schildkröte sich befindet. In dieser Aufgabe darf sie sowohl jedes Feld als auch jeden Feldübergang beliebig häufig nutzen. Wenn sie also von ihrem Feld K aus ein Feld F nicht erreichen kann, dann könnte sie von Feld F aus auch nicht Feld K erreichen. Im Folgenden ignorieren wir also die Information, wo K ist. Dieses Feld wird wie jedes andere weiße Feld behandelt.

J2.1 Der direkte Weg

Um diese Aufgabe zu lösen, muss zunächst die in der Eingabe enthaltene Landkarte von Quadratien gespeichert werden. Dafür kann man einen Array nehmen, dessen Größe in der ersten Eingabezeile vorgegeben wird. In die Zeilen des Arrays speichert man dann die Inhalte der Eingabezeilen, und zwar ein Eingabezeichen pro Array-Element. Auch andere Datenstrukturen sind möglich. Wichtig ist, dass man für ein Landkartenfeld leicht die Elemente in der Datenstruktur bestimmen kann, die den oben, unten, rechts bzw. links gelegenen Feldern entsprechen.

Außerdem benötigt man eine Liste – oder irgendeine andere Datenstruktur, in der mehrere Elemente gespeichert werden können, z. B. Mengen, Stacks oder Warteschlangen (englisch queue). Wir verwenden eine Liste und speichern darin, von wo aus Kassiopeia noch weitergehen kann. Diese Liste nennen wir nun L_W. Zu Beginn enthält L_W nur das Startfeld.

Von jedem Feld aus kann Kassiopeia theoretisch einen Schritt nach oben, unten, rechts oder links machen. Praktisch geht das natürlich nur, wenn das jeweilige Nachbarfeld weiß ist. Wenn ein Nachbarfeld eines aktuell untersuchten Feldes also weiß ist, wird es in L_W gesteckt. Wenn wir die vier Nachbarfelder überprüft haben, dann sind wir mit dem aktuellen Feld fertig. Damit wir das Feld nicht nochmals besuchen, färben wir es ein. Natürlich müssen wir das aktuell untersuchte Feld auch aus L_W entfernen, da wir ja schon alle Möglichkeiten betrachtet haben, wie man von dort aus weitergehen kann. Dann untersuchen wir das nächste Feld in L_W, und so weiter.

Wenn L_W leer ist, hat Kassiopeia alle Felder besucht, die sie besuchen kann. Nun müssen wir überprüfen, ob das tatsächlich alle Felder sind. Dazu schauen wir uns alle Felder an. Wenn ein Feld noch weiß ist, dann kann Kassiopeia dieses Feld nicht erreichen.

J2.2 Repräsentation der Landkarte als Graph

Die Beispieldaten liegen in dem Format vor, das in Abb. 4a dargestellt ist. Dabei steht `#` für ein schwarzes Feld, und `K` sowie Leerzeichen stehen für weiße Felder. Die erste Zeile der Eingabe gibt die Höhe bzw. Breite von Quadratien an.

Wie oben beschrieben, kann die analoge Repräsentation dieser Landkarte in einem Array durchaus verwendet werden, um die Aufgabe zu lösen. Allerdings ist die Repräsentation als *Graph* üblicher und erlaubt die Lösung mit bekannten Methoden. *Graphen* sind mathematische Objekte, die aus einer *Knotenmenge* V und einer *Kantenmenge* E bestehen.[5] Jede Kante stellt eine

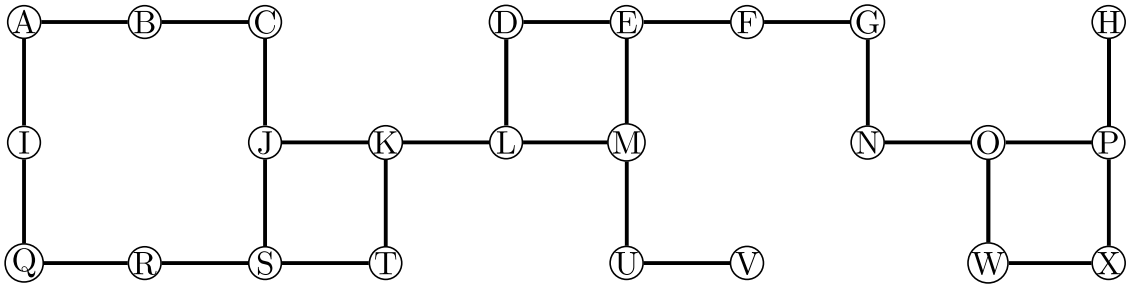
[5] Auf Englisch heißen Knoten „**v**ertices“ und Kanten „**e**dges“.

```
5 12
############
#K  #    # #
# #    #   #
#    #  #  #
############
```

(a) Eingabedatensatz Nr. 7

```
5 12
############
#ABC#DEFG#H#
#I#JKLM#NOP#
#QRST#UV#WX#
############
```

(b) Beschriftungen A–Z für die Felder von Welt 7



(c) Welt 7 als Graph

Abbildung 4: Repräsentation einer Landkarte als Graph

Beziehung zwischen zwei Knoten aus V her. In unserem Beispiel kann man jedes weiße Feld als einen Knoten sehen. Dann sind zwei Knoten durch eine Kante verbunden, wenn sie benachbarten Feldern entsprechen. Wir tun also so, als ob schwarze Felder einfach gar nicht vorhanden wären. Eine typische Art einen Graphen bildlich darzustellen ist in Abb. 4c zu sehen.

Einen Graph nennt man *zusammenhängend*, wenn man von jedem Knoten aus jeden anderen über Kanten und andere Knoten erreichen kann. Kassiopeia kann also ein Feld nicht erreichen, wenn der Graph nicht zusammenhängend ist.

Doch bevor wir das algorithmisch bestimmen, müssen wir den Graphen als Datenstruktur repräsentieren. Dazu gibt es zwei verbreitete Möglichkeiten, die beide ihre Stärken haben: Die *Adjazenzmatrix* und die *Adjazenzliste*. Eine Adjazenzmatrix ist eine Tabelle aus Nullen und Einsen. Eine Eins in Zeile i und Spalte j bedeutet, dass man von Knoten i zu Knoten j kommt. Da es auf Quadratien keine Einbahnstraßen gibt, muss diese Tabelle (bzw. *Matrix*) symmetrisch sein. Wenn ich also von i nach j komme, dann auch von j nach i.

Um die Matrix für den Graphen der weißen Felder aufzubauen, kann man zuerst den kompletten Gittergraphen als Matrix abspeichern. Es wird also ein Graph mit (Breite von Quadratien) · (Höhe von Quadratien) Knoten erstellt, wobei die inneren Knoten mit jeweils vier Nachbarn verbunden sind und die äußeren mit jeweils zwei Nachbarn. Sobald man diesen Graphen hat, löscht man die Knoten heraus, die schwarze Felder repräsentieren. Dies macht man, indem die jeweilige Spalte und Zeile des Knotens aus der Matrix entfernt werden. Um Fehler bei der Verschiebung der Indizes zu vermeiden empfiehlt es sich, die zu löschenden Felder einmal zu berechnen, nach ihrem Index absteigend zu sortieren und dann in dieser Reihenfolge zu entfernen.

Eine Adjazenzliste ist eine Liste von Listen. Jede Teilliste steht für einen Knoten. Die Liste des Knotens i beinhaltet die Namen aller Knoten, zu denen man von i aus kommen kann.

Beispiel	0	1	2	3	4	5	6	7
Zusammenhängend	Ja	Nein	Ja	Ja	Ja	Ja	Ja	Ja

Tabelle 1: Ergebnisse für die Beispieldaten

J2.3 Prüfung auf Zusammenhang

Um zu überprüfen, ob ein Graph zusammenhängend ist, gibt es zahlreiche Algorithmen. Die beiden wohl einfachsten und nützlichsten sind die *Breitensuche* bzw. *Tiefensuche*. Die Idee ist dabei, den Graphen zielgerichtet abzulaufen (bzw. zu *traversieren*, wie man in der Graphentheorie sagt).

Algorithmus 2 Graphen traversieren

```
function IstZusammenhaengend(Graph G = (V,E))
    ZuLaufen ← Liste
    start ← x ∈ V                          ▷ wähle zufällig einen Knoten aus
    Hinzufuegen(ZuLaufen, start)
    Gelaufen ← { start }
    while |ZuLaufen| > 0 do
        x ← Ziehe(ZuLaufen)                ▷ das erste Element, also von Vorne
        Gelaufen ← Gelaufen ∪ x
        for Nachbar x_j ∈ Nachbarn(x) do
            if x_j ∉ Gelaufen then
                Hinzufuegen(ZuLaufen, x_j)
            end if
        end for
    end while
    return V \ Gelaufen = ∅
end function
```

Sowohl bei der Breitensuche als auch bei der Tiefensuche hat man (a) eine Liste von Knoten, die man noch betrachten will, (b) eine Liste von Knoten, die bereits betrachtet wurden, sowie (c) den Graphen selbst. Algorithmus 2 beschreibt mit Pseudocode die grundsätzliche Vorgehensweise beim Traversieren eines Graphen. Je nachdem, wie man dabei die Funktion Hinzufuegen implementiert, wird daraus eine Breiten- oder Tiefensuche. Fügt man die Elemente vorne in die Liste ein, so ist es eine Tiefensuche; fügt man die Elemente hinten in die Liste ein, ist es eine Breitensuche. In beiden Varianten geht man alle Knoten durch, die man erreichen kann.

J2.4 Bewertungskriterien

- Bei einer Junioraufgabe ist eine Lösung auf Basis von Graphen natürlich nicht gefordert. Alle funktionierenden Vorgehensweisen sind akzeptabel.
- Das Verfahren sollte aber nicht unnötig aufwändig sein. Insbesondere ist entscheidend, die bereits besuchten Felder sauber zu kennzeichnen, um nicht unnötig viele Möglichkeiten prüfen zu müssen.

- Der Startpunkt von Kassiopeia ist für die Berechnung der Erreichbarkeit irrelevant. Diese Einsicht könnte man haben – aber das wird nicht erwartet. Für das Verfahren ist es in Ordnung, immer vom Startpunkt auszugehen.
- Die in der Aufgabenstellung vorgegebenen Bedingungen müssen beachtet sein:
  - Kassiopeia betritt nur weiße Felder.
  - Sie bewegt sich in einem Schritt auf ein horizontal oder vertikal (nicht diagonal!) benachbartes Feld.
  - Das mit `K` markierte Feld ist auch ein weißes Feld.
- Das Verfahren und seine Implementierung sollten korrekte Ergebnisse liefern. Insbesondere sollen die Ergebnisse für die 7 Beispieldatensätze korrekt sein (vgl. Tabelle 1).
- Es sollen wenigstens so viele Beispiele (und die zugehörigen Ergebnisse) auch in der Dokumentation behandelt sein, dass das korrekte Funktionieren der Lösung klar wird.