Karlsruher Institut für Technologie

Algorithmen 1 – Sommersemester 2018
Zuverlässige Softwaresysteme im Kontext der Automobilindustrie
`http://verialg.iti.kit.edu`

Prof. Dr. Carsten Sinz – Markus Iser

Übungsblatt 10

Ausgabe: 27.06.2018 – 15:30
Abgabe: 04.07.2018 – 13:00

A Tiefensuche iterativ (2 Punkte)

Implementieren Sie eine nicht rekursive Tiefensuche ausgehend von einem Knoten s. Die Laufzeit $\mathcal{O}(m+n)$ darf nicht überschritten werden.

Lösung

Die folgende nicht rekursive Implementierung der Tiefensuche beruht auf der Verwendung eines Stacks. Die Idee ist es, jeweils die Nachbarn des derzeit bearbeiteten Knotens auf einen Stack zulegen. Man initialisiert den Stack S mit dem Startknoten s. Solange der Stack S nicht leer ist wird das vorderste Element vom Stack heruntergenommen und anschließend alle noch nicht gesichteten Nachbarn vorne auf den Stack gelegt und gleich als gesichtet markiert. So ist sichergestellt, das man zuerst in die Tiefe geht und dabei keine Knoten doppelt besucht.

Algorithm 1: DFS

```
Input: NodeId s, Graph G
1 Stack S ← ⟨s⟩
2 Array visited ← ⟨false, ..., false⟩
3 visited[s] ← true
4 while S ≠ ∅ do
5     NodeId u ← S.pop()
6     for (u, v) ∈ E do
7         if not visited[v] then
8             visited[v] ← true
9             S.push(v)
```

B Inseln zählen und Ackerfläche zuteilen

Matrixnesien ist eine idyllisch gelegene Inselgruppe im Pazifischen Ozean mit algorithmisch interessierten Ureinwohnern. Sie haben eine grob gerasterte Landkarte ihrer Inselgruppe als $o \times p$ große $\{0,1\}$-Matrix gegeben. Eine 0 symbolisiert Wasser und eine 1 symbolisiert Land. Es kann davon ausgegangen werden, dass sowohl horizontal und vertikal, als auch schräg benachbarte Landfelder, eine einzige zusammenhängende Landmasse (Insel) darstellen.

B.1 Inseln zählen (3 Punkte)

Implementieren Sie einen Algorithmus, der in Zeit $\mathcal{O}(o \cdot p)$ die Zahl der Inseln auf gegebenen Landkarte berechnet, und überzeugen Sie die Ureinwohner von der Korrektheit ihres Verfahrens und zeigen Sie seine Laufzeit.

Lösung

Die Aufgabe war in einem Google Telefoninterview zu lösen in 15 min per Codeeingabe in Google Docs ohne Compiler.

Im Grunde wird der durch die Matrix *implizit* definierte Graph traversiert. Eine Kante (s,t) existiert, wenn $s = 1 = t$ ist und s in der 1-Umgebung (neighbourhood) von t liegt. Algorithmus 2 (Count Islands) läuft über $\mathcal{O}(o \cdot p)$ Felder und ruft bei Landfeldern eine BFS (Algorithmus 3, Sink Island) auf. Jedes Landfeld wird bei Antreffen versenkt und nur einmal in die BFS-Queue hinzugefügt. Da es maximal $o \cdot p$ viele Landfelder gibt, ist auch hier der Aufwand $\mathcal{O}(o \cdot p)$.

Zur Korrektheit reicht es zu sehen, dass eine Insel vollständig versenkt wird, sobald sie gezählt wurden. Dies folgt aus der Korrektheit der BFS.

Algorithm 2: Count Islands

Input: Matrix M
Output: Count c

```
1 c ← 0
2 for i ∈ [0, o] do
3     for j ∈ [0, p] do
4         if M_ij = 1 then
5             c ← c + 1
6             sink-island(M, i, j)
7 return c
```

Algorithm 3: Sink Island

Input: Matrix M, Coordinates c_i, c_j

```
1 Queue q
2 pushback(q, (c_i, c_j))
3 while size(q) > 0 do
4     (i, j) ← popfront(q)
5     M_ij ← 0
6     for (i', j') ∈ neighbourhood(i, j) do
7         if M_i'j' = 1 then
8             pushback(q, (i', j'))
```

Algorithm 4: Neighbourhood

Input: Matrix M, Coordinates i, j
Output: List of Tuples $\{(i', j') | (i', j') \text{ adjacent to } (i, j)\}$

```
1 List L
2 for i' ∈ {max(i − 1, 0), ..., min(i + 1, o)} do
3     for j' ∈ {max(j − 1, 0), ..., min(j + 1, p)} do
4         pushback(L, (i', j'))
5 return L
```

B.2 Ackerfläche zuteilen (3 Punkte)

Auf jeder Insel liegen genau zwei möglichst weit voneinander entfernte Dörfer. Wir betrachten im folgenden nur eine Insel. Die Ureinwohner der zwei Dörfer betreiben auf den umliegenden (Matrix)Feldern Landwirtschaft, wobei aus Tradition ein Dorf nur Mais anbaut und das andere ausschließlich Weizen. Beide Dörfer florieren und brauchen immer mehr Ackerfläche in ihrer unmittelbaren Umgebung um die Bewohner zu ernähren.

Um einem drohenden Krieg zwischen den Dörfern aus dem Wege zu geben, bitten die Dorfältesten Sie ein faire Zuteilung der Ackerfelder zu beiden Dörfern vorzunehmen. Die Zuteilung soll folgende Eigenschaften erfüllen:

- Beide Dörfer müssen gleich viele Felder erhalten.
- Die Ackerbereiche der beiden Dörfer müssen zusammenhängend sein.

Gleichzeitig müssen die folgenden beiden Optimierungsziele in angegebener Priorität erfüllt werden:

1. Die Anzahl nicht zugeteilter Felder ist so klein wie möglich.
2. Die Summe der Transportwege der beiden Dörfer zu ihren Feldern ist so klein wie möglich.

Implementieren Sie einen Algorithmus, der anhand der gegebenen Landkarte eine Zuteilung der Ackerfelder berechnet und überzeugen Sie die Dorfältesten beider Dörfer, dass die berechnete Ackerzuteilung unter den obigen Bedingungen bestmöglich ist.

Lösung (fehlerhaft)

Die Idee ist zwei BFS ausgehend von den Dörfern abwechselnd zu berechnen. Hierdurch wird in jedem Schritt den zwei Dörfern ein neues Matrixfeld zugeteilt, das kleinste zusätzliche Transportkosten hat.

Der BFS Algorithmus konstruiert zwei Bäume in dem Matrixgraphen mit den Dörfern an den Wurzeln. Diese Bäume haben die Eigenschaft, dass die Summe der Wege minimale Pfadkosten auf dem zur Verfügung stehenden Gebiet haben.

Im Algorithmus werden die BFS-Bereiche abwechselnd vergrößert und jeweils ein Feld einem Dorf zugeteilt. Die while-Schleife in Algorithmus 5 (Land Distribution) hat also als Invariante, dass beide Dörfer gleich viele Felder haben.

Die berechneten Ackerflächen hängen zusammen, da in der BFS-Queue nur Felder enthalten sind, an denen ein bereits zugeteiltes Feld anliegt.

Die zweifache BFS teilt abwechselnd immer ein Feld zu, bis einem Dorf keine Zuteilung mehr möglich ist. Daher unterscheidet sich die Zahl der zugeteilten Felder nicht.

Algorithm 5: Land Distribution

Input: Matrix M, Village Coordinates (a_i, a_j) and (b_i, b_j)
Output: Matrix M with land distributed (Land A if $M_{ij} = 11$, Land B if $M_{ij} = 21$)

```
1  Queue A
2  Queue B
3  pushback(A, (a_i, a_j))
4  pushback(B, (b_i, b_j))
5  while not (empty(A) and empty(B)) do
6      T_A ← getLand(M, A, 11)
7      if T_A = nil then
8          return M
9      T_B ← getLand(M, B, 21)
10     if T_B = nil then
           // Backtrack last assignment:
11         (t_0, t_1) ← T_A
12         M_{t_0,t_1} ← 1
13         return M
```

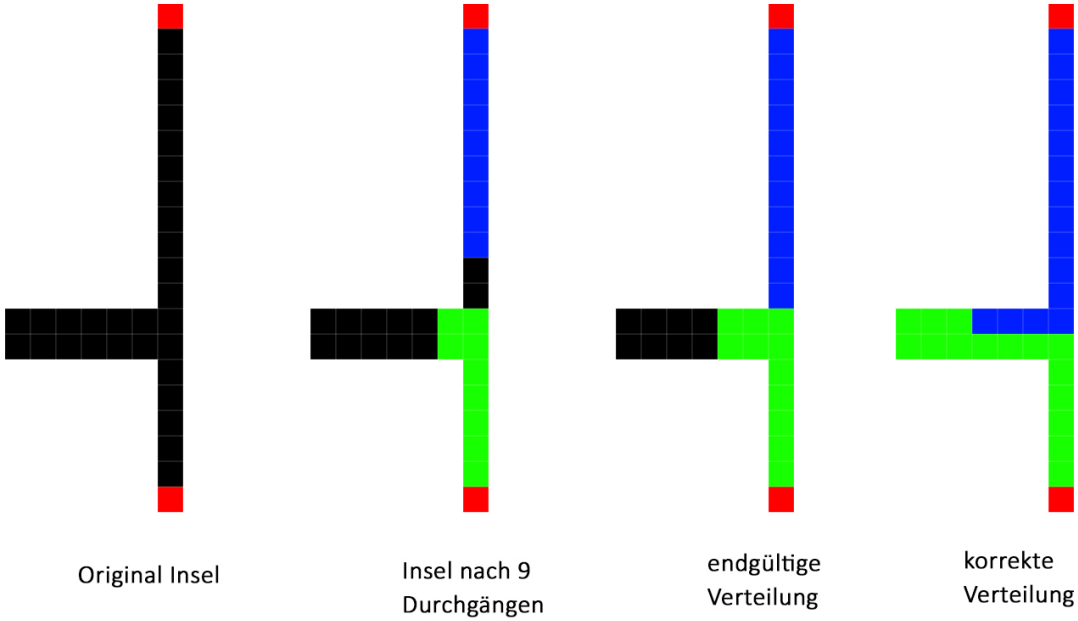
Algorithm 6: Get Land

Input: Matrix M, Coordinates Queue Q, Marker v
Output: Tuple T

```
// BFS:
1  while size(Q) > 0 do
2      (i, j) ← popfront(Q)
3      if M_ij = 1 then
4          M_ij ← v
5          for (i', j') ∈ neighbourhood(i, j) do
6              if M_ij = 1 then
7                  pushback(Q, (i', j'))
8          return (i, j)
9  return nil
```

Probleme mit Musterlösung und ein möglicher Fix

Der obige Algorithmus aus der originalen Musterlösung ist leider falsch, aber funktioniert mindestens für konvexe[1] Inseln. Leider liefert er nicht in allen möglichen Szenarien ein korrektes Ergebnis, wie das folgende Gegenbeispiel der Tutorenschaft zeigt.



Die Inselbewohner sind entsetzt und engagieren einen weiteren Algorithmiker. Dieser entwickelt einen Fix, welcher die spezielle Geometrie abnormaler Inseln berücksichtigt. Der Spezialfall wird durch Algorithmus 7 gelöst, in dem jedes Feld einmal "geklaut" werden darf. Das heißt Algorithmus 5 (Land Distribution) verwendet statt Algorithmus 6 (Get Land) nun Algorithmus 7 (Get Land With Stealing). Beachten Sie, dass Algorithmus 7 (Get Land with Stealing) den alten Algorithmus 6 (Get Land) als Subroutine verwendet.

Die Grundidee ist, das verbauen von schmalen Wegen dadurch zu kompensieren, dass jedes Feld einmal geklaut werden darf. Um Oszillationen zu vermeiden und wegen Zielfunktion 1 (möglichst kurze Pfade), wird in der Beispiel-Implementierung über die erste Dezimalstelle sichergestellt, dass jedes Feld höchstens einmal den Besitzer wechselt. Da Algorithmus 7 bevorzugt freie Felder belegt und nur im "Notfall" klaut, genügt das auch. Dadurch sind in der Ergebnismatrix die Felder von Dorf A sowohl mit 11 als auch mit 10 (falls einmal geklaut) markiert. Felder von Dorf B sind mit 21 b.z.w. 20 (falls geklaut) markiert.

Da in der Variante mit Klauen nicht mehr automatisch sicher ist, dass die Felder der Dörfer am Ende zusammenhängen, muss zusätzlich sichergestellt, dass das geklaute Feld den Graphen der Nachbarfelder nicht zerschneidet. Dieser Regel ist in Algorithmus 8 (Does not cut) ausgelagert.

[1] Wenigstens ein kürzester Pfad zwischen zwei beliebigen Inselfeldern liegt komplett auf der Insel.

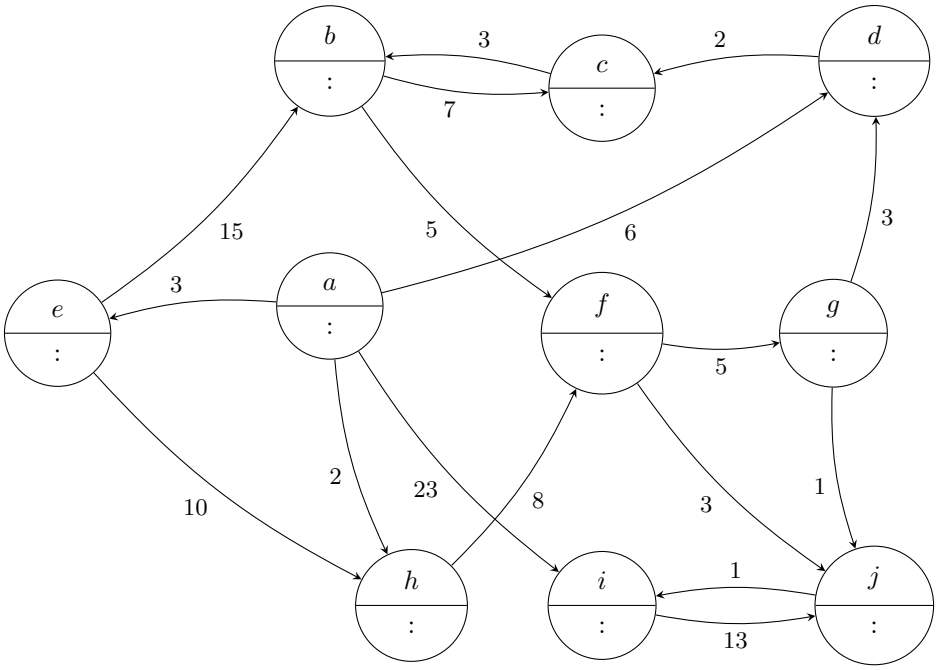
Algorithm 7: Get Land with Stealing

Input: Matrix M, Coordinates Queue Q, Marker v
Output: Tuple T

```
// Create Backup:
```
$B \leftarrow Q$
```
// Try to obtain next field with simple method:
```
$T \leftarrow \mathsf{getLand}(M, Q, v)$
if $T \neq \mathsf{nil}$ **then**
  return T
```
// Restore Q from Backup:
```
$Q \leftarrow B$
```
// Steal:
```
while $\mathsf{size}(Q) > 0$ **do**
  $(i, j) \leftarrow \mathsf{popfront}(Q)$
  if $M_{ij} \bmod 1 = 1$ *and* $M_{ij} \notin \{v, v-1\}$ *and* $\mathit{doesNotCut}(M, i, j)$ **then**
    $M_{ij} \leftarrow v - 1$
    for $(i', j') \in \mathsf{neighbourhood(i, j)}$ **do**
      if $M_{ij} \bmod 1 = 1$ *and* $M_{ij} \notin \{v, v-1\}$ **then**
        $\mathsf{pushback}(T, (i', j'))$
    return (i, j)
return nil
```

**Algorithm 8:** Does Not Cut

**Input**: Matrix $M$, Coordinates $i, j$
**Output**: true iff $M_{ij}$ can not disconnect the planar graph of neighbours with same value

val $\leftarrow M_{ij}$
num $\leftarrow \sum_{k=i-1}^{i+1} \sum_{l=j-1}^{j+1} \big((M_{k,l} = \text{val})\,?\,1 : 0\big)$
```
// Field names for readability (out of bounds implies false):
```
$A \leftarrow (M_{i-1,j-1} = \text{val})$ $B \leftarrow (M_{i-1,j} = \text{val})$ $C \leftarrow (M_{i-1,j+1} = \text{val})$
$D \leftarrow (M_{i,j-1} = \text{val})$ $E \leftarrow (M_{i,j} = \text{val})$ $F \leftarrow (M_{i,j+1} = \text{val})$
$G \leftarrow (M_{i+1,j-1} = \text{val})$ $H \leftarrow (M_{i+1,j} = \text{val})$ $I \leftarrow (M_{i+1,j+1} = \text{val})$
horizontal $= (B \wedge H) \rightarrow (D \vee F)$
vertical $= (D \wedge F) \rightarrow (B \vee H)$
edges $= (A \rightarrow (B \vee D)) \wedge (C \rightarrow (B \vee F)) \wedge (G \rightarrow (D \vee H)) \wedge (I \rightarrow (F \vee H))$
**return** num $\leq 1 \vee$ (horizontal $\wedge$ vertical $\wedge$ edges)
```

C Dijkstras Algorithmus

Führen Sie auf folgendem Graphen den Algorithmus von Dijkstra aus, beginnend mit Knoten a. Als Ergebnis soll in jedem Knoten folgendes stehen: links vom Doppelpunkt die Nummer des Schritts, in dem der Knoten aus der Queue genommen wurde; rechts vom Doppelpunkt die Länge des kürzesten Wegs zu a. Zeichnen Sie außerdem den Baum der kürzesten Wege von a aus ein. Der erste Knoten wird zum Zeitpunkt 1 aus der Queue entfernt. Sie können direkt in dieses Blatt einzeichnen.



Lösung