Funktionale Programmierung
Prof. Claudia Fohry

Universität Kassel
Sommersemester 2020
Abgabe: 6.7.20 um 9.00 Uhr

Abschlussprojekt

Soweit nicht anders angegeben, achten Sie bitte bei allen Lösungen auf Kürze, Klarheit und Effizienz. Verwenden Sie, falls passend, fortgeschrittene Sprachkonstrukte wie Funktionen höherer Ordnung, unendliche Listen, Typdefinitionen, `$` und `$!`. Diese Aspekte sind ein wesentlicher Bestandteil der Bewertung! Sie dürfen bei allen Aufgaben davon ausgehen, dass die Nutzereingaben korrekt sind, d.h., eine Fehlerbehandlung ist nicht nötig.

Bitte sehen Sie während der Bearbeitungszeit regelmäßig ins Diskussionsforum. Falls Sie Fragen zu den Aufgaben haben, stellen Sie diese bitte im Forum. Achten Sie bei der Formulierung darauf, nichts von Ihrer Lösung zu verraten. Es werden nur Fragen zum Verständnis der Aufgabenstellungen bzw. zum Lesestoff beantwortet; Hilfe beim Debugging gibt es nicht. Sobald Sie erste Ergebnisse haben, können Sie gern exemplarische Ein-/Ausgaben im Forum posten, um sie mit den Ergebnissen anderer Teams zu vergleichen.

Aufgabe a) kann mit den bisherigen Kenntnissen gelöst werden. Bei Aufgabe b) benötigen Sie für die Ein-/Ausgabe zusätzlich Stoff der Woche vom 3.6. Bei Aufgabe c) benötigen Sie auch Stoff der Woche vom 8.6. Konstrukte, die nach den angegebenen Daten behandelt werden (z.B. Monaden) dürfen verwendet werden, müssen aber nicht. Unabhängig davon sind sie Bestandteil des Abschlussgesprächs.

a) Gegeben sind ein $n \times m$-Zahlenfeld mit Werten zwischen `1` und `9` sowie zwei voneinander verschiedene Positionen (i_s, j_s) und (i_e, j_e) mit $0 \leq i_s, i_e < n$ und $0 \leq j_s, j_e < m$. Der Wert in Zelle (i_s, j_s) entspricht einem Startguthaben.

   Wir betrachten Wege von (i_s, j_s) nach (i_e, j_e), die jede Zelle des Zahlenfeldes genau einmal durchlaufen. Die Wege sind aus horizontalen und vertikalen Schritten zur Nachbarzelle zusammengesetzt, bei denen sich das Guthaben wie folgt verändert:

   - Schritt nach rechts: Wert der neuen Zelle wird zum Guthaben addiert
   - Schritt nach links: Wert der neuen Zelle wird vom Guthaben subtrahiert
   - Schritt nach unten: Guthaben wird mit Wert der neuen Zelle multipliziert
   - Schritt nach oben: Guthaben wird durch Wert der neuen Zelle dividiert (ganzzahlige Division)

   Ein Weg ist zulässig, wenn das Guthaben nach keinem Schritt negativ wird.

   Das Programm soll eine Liste aller zulässigen Wege ermitteln. Es muss sich im ghci mittels

   ```
   ghci> pathsShort n m is js ie je z1 z2 z3
   ```

   aufrufen lassen (zu `z1`, `z2`, `z3` siehe unten). Beispiel:

   ```
   ghci> pathsShort 3 3 0 0 2 2 1 2 3
   [[(0,0,1,1),(1,0,2,2),(2,0,3,6),(2,1,2,8),(1,1,4,2),(0,1,7,0),(0,2,1,1),
     (1,2,9,9),(2,2,5,45)],
    [(0,0,1,1),(0,1,7,7),(0,2,1,8),(1,2,9,72),(1,1,4,68),(1,0,2,66),
     (2,0,3,198),(2,1,2,200),(2,2,5,205)]]
   ```

In der Ausgabe entspricht jede der beiden Teillisten einem Weg. In einem Tupel wie beispielsweise `(2, 0, 3, 198)` sind `(2, 0)` die Koordinaten einer Zelle, `3` der Wert im Feld, und 198 der Score an dieser Stelle des Weges. Ihre Ausgabe sollte die gleichen Informationen enthalten, das Format ist egal. Das obige Beispiel entspricht dem folgenden Feldinhalt (muss nicht ausgegeben werden):

```
1 7 1
2 4 9
3 2 5
```

Schreiben Sie zwei Haskell-Programme, `pathsShort.hs` und `pathsFast.hs`, zur Lösung des Problems. Dabei soll

- `pathsShort` möglichst *kurz* sein, **egal** mit welcher Effizienz. Kurz bedeutet, dass das Programm (inkl. aller Hilfsfunktionen) möglichst wenige Konstrukte verwendet, bei Bezeichnerlängen und anderen Stilelementen bitte nicht sparen.
- `pathsFast` *schnell* sein. Dabei reicht es, wenn die Haskell-Umsetzung effizient ist und der Algorithmus die Konstruktion aussichtsloser Wege vermeidet bzw. abbricht (zumindest in einem signifikanten Teil der Fälle).

Zwecks Vergleichbarkeit der Ergebnisse initialisieren Sie bitte das Array mittels

```
for (i = 0; i < n; ++i) {  // Zeilen
  for (j = 0; j < m; ++j) {  // Spalten
    a(i, j) = ((z1 * i + z2 * j) % 9) + 1;
    z1 = (z1 + z3) % 100;
    z2 = (z2 * z3) % 100;
  }
}
(umzuschreiben nach Haskell)
```

Ihr Algorithmus darf nicht auf diese Initialisierung zugeschnitten sein.

Hier sind einige weitere Beispiele (ohne Garantie):

```
pathsShort 2 3 0 0 1 2 1 2 3
[[(0,0,1,1),(1,0,2,2),(1,1,4,6),(0,1,7,0),(0,2,1,1),(1,2,9,9)]]

pathsShort 2 4 0 0 1 3 123 456 789
[]

pathsShort 2 4 0 0 1 2 5 5 5
[[(0,0,1,1),(1,0,8,8),(1,1,2,10),(0,1,8,1),(0,2,6,7),(0,3,4,11),(1,3,8,88),(1,2,5,83)]]
```

b) Eine Gemeinschaft aus $n = 23$ Personen hat sich schon lange auf ein Wiedersehen gefreut, welches kurzfristig durch drei Videokonferenzen in Gruppen zu je 7 bis 8 Teilnehmern ersetzt werden muss.

Jeder Teilnehmer darf die Namen von ein bis drei Personen nennen, mit denen er gern in der gleichen Gruppe sein möchte. Dabei hat die zuerst genannte Person die höchste Priorität, gefolgt von der zweiten und dritten.

Schreiben Sie ein Haskell-Programm `groups.hs` zur Berechnung einer Gruppeneinteilung, die möglichst viele Wünsche zu den Gruppenpartnern berücksichtigt. Genauer gesagt, soll das Programm eine Zielfunktion maximieren, bei der jeder erfüllte Erstwunsch mit 10 Punkten, jeder erfüllte Zweitwunsch mit 5 Punkten und jeder erfüllte Drittwunsch mit 1 Punkt bewertet und dann die Summe gebildet wird.

Lösen Sie die Aufgabe durch *lokale Suche*. Wie Sie vielleicht wissen, ist dies ein Grundalgorithmus zur Berechnung von Näherungslösungen für kombinatorische Optimierungsprobleme. Bitte beziehen Sie sich auf die folgende, auf unser Problem zugeschnittene Variante der lokalen Suche:

- Die lokale Suche beginnt bei einer beliebigen zulässigen Initiallösung. In unserem Fall ist dies eine beliebige Einteilung in Gruppen mit 7 bis 8 Teilnehmern.
- Anfangs entspricht die aktuelle Lösung der Initiallösung. Danach durchsucht das Programm solange die „Nachbarschaft“ der aktuellen Lösung, bis es bei einer aktuellen Lösung angekommen ist, für die es keinen Nachbarn mit höherem Zielfunktionswert mehr gibt. Existiert dagegen mindestens ein Nachbar mit höherem Zielfunktionswert, so wird die aktuelle Lösung durch einen solchen Nachbarn ersetzt. Wie Sie den Nachbarn auswählen und wieviele Nachbarn überhaupt berechnet werden, ist Ihnen überlassen.
- Die „Nachbarschaft“ der aktuellen Lösung entspricht allen zulässigen Gruppeneinteilungen, die sich durch geringfügige Änderungen aus der aktuellen Einteilung ergeben. Bitte betrachten Sie mindestens die folgenden Änderungen:
  - Verschieben eines Mitglieds einer 8-er Gruppe in eine 7-er Gruppe,
  - Tausch eines Mitglieds von Gruppe `A` mit einem Mitglied von Gruppe `B`. Nach dem Tausch ist also das frühere `A`-Mitglied in `B` und umgekehrt.
  - Tausch von zwei Mitgliedern von Gruppe `A` mit zwei Mitgliedern von Gruppe B. Danach sind also die beiden bisherigen `A`-Mitglieder in `B` und umgekehrt.
- Geben Sie die initiale und alle danach gefundenen aktuellen Lösungen zusammen mit ihren Zielfunktionswerten am Bildschirm aus, so dass man am Bildschirm verfolgen kann, wie sich der Zielfunktionswert schrittweise verbessert.

Die Wünsche zu den Gruppenpartnern sind in einer Datei beschrieben, deren Name beim Programmaufruf zu übergeben ist. Bitte entnehmen Sie das Dateiformat dem folgenden Beispiel. Der Name der Person steht also vorn, und dahinter die ein bis drei Wünsche, jeweils durch ein Leerzeichen getrennt. Die Datei enthält keine Leerzeilen, auch nicht am Dateiende. Bitte halten Sie das Format genau ein.

```
anna susi lena
ben paul
carl ben paul anna
diana susi anna
...
zoe susi
aela lena ben diana
oener paul carl
```

Das Programm soll auch für andere Teilnehmerzahlen als $n = 23$ funktionieren. Für jedes n soll es die Teilnehmer in drei Gruppen ungefähr gleicher Größe aufteilen. Das Programm muss terminieren, also nach einer mehr oder weniger langen Zeit zum Abschluss kommen.

c) Wir betrachten die folgende Minivariante von „Mensch-ärgere-dich-nicht “: Zwei Mitspieler (wir nennen sie A und B) verfügen über je eine Figur. Das Spielbrett zeigt einen Kreis mit 24 Kästchen, welche mit 1 bis 24 beschriftet sind. Hinter der 24 kommt also wieder die 1. Spieler A setzt seine Figur bei der 1 ein, und Spieler B bei der 8. Ziel des Spiels ist es, mit der eigenen Figur einmal den Kreis zu durchlaufen. Spieler 1 ist also fertig, sobald seine Figur erneut auf der 1 angekommen ist oder diese überschritten hat. Spieler B ist fertig, wenn er die 8 erreicht oder überschritten hat. Wem das zuerst gelingt, der hat gewonnen.

   Die Spieler würfeln abwechselnd, Spieler A beginnt. Man darf immer sofort einsetzen. Würfelt beispielsweise A am Anfang eine 3, so setzt er seine Figur auf Feld 3. Danach ist B an der Reihe. Würfelt er eine 6, so setzt er seine Figur auf Feld 14. Landet man auf einem Feld, auf dem bereits die Gegnerfigur steht, so hat man zwei Optionen. Entweder man schlägt den Gegner, der daraufhin wieder von vorn beginnen muss, oder man stellt sich auf das Feld hinter dem Gegner. Beispiel: A steht auf 7 und B steht auf 12. A würfelt eine 5. Dann kann A entweder B schlagen, oder sich auf Feld 11 stellen.

   Wir betrachten die beiden folgenden Strategien:

   - *bad:* Immer wenn schlagen möglich ist, schlägt der Spieler die Gegnerfigur.
   - *nice:* Immer wenn schlagen möglich ist, stellt sich der Spieler hinter die Gegnerfigur.

   Lösen Sie die beiden folgenden Teilaufgaben:

   - `ludoInteractive`: Ein menschlicher Spieler übernimmt die Rolle von Spieler A und der Computer die von Spieler B. In jeder Runde wird die gewürfelte Augenzahl angezeigt, und der Spieler ggf. gefragt, ob er schlagen oder dahintersetzen möchte. Dann ist der Computer an der Reihe, und das Ergebnis wird am Bildschirm angezeigt (eine einfache Form der Anzeige genügt). Der Computer spielt nach der *bad*-Strategie. Am Spielende wird der Gewinner verkündet.
   - `ludoStatistic`: Zwei Computerspieler spielen eine vom Nutzer gewünschte Anzahl von Runden. Spieler A spielt nach der bad-Strategie, und Spieler B nach der nice-Strategie. Am Ende wird ausgegeben, in wievielen Runden A bzw. B gewonnen hat.

Abgabe: Alle Lösungen müssen bis spätestens Mo., d. 6.7. um 9:00 Uhr per Email an fohry@uni-kassel.de mit dem Betreff “FP Abgabe“ abgegeben werden. Dokumentieren Sie bitte, wie Ihre Programme zu laden, aufzurufen, übersetzen bzw. auszuführen sind.