FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

4. Übung

Abgabe bis 20.11.2017, 10:00 Uhr

Einzelaufgabe 4.1: Rucksack packen — 10 EP

In dieser Aufgabe soll eine abgewandelte Form des in der Vorlesung vorgestellten Algorithmus für das Packen eines Rucksacks implementiert werden. Laden Sie dazu den vorgegebenen Rumpf `RucksackPacken.java` von der Übungsseite und machen Sie sich damit vertraut. Ihre Implementierung muss rekursiv, aber ohne besondere Maßnahmen zur Effizienzsteigerung sein!
Es gilt einen Rucksack mit Gegenständen zu füllen, wobei jeder Gegenstand eine Größe (wie in der Vorlesung) und einen Wert (*anders* als in der Vorlesung) hat. Der Wert eines Gegenstandes trägt zum Gesamtwert des gepackten Rucksacks bei, sofern sich dieser Gegenstand im Rucksack befindet.
Ziel ist es, den Rucksack so zu packen, dass die Summe der Werte der gepackten Gegenstände maximal ist. Hierbei ist zu beachten, dass jeder Gegenstand **höchstens einmal** in den Rucksack gepackt werden kann. Im Gegensatz zur Vorlesung, gibt es immer eine Lösung, auch wenn der Rucksack am Ende nicht komplett gefüllt ist (er könnte sogar leer bleiben).
Ihre Rekursion muss nacheinander die verschiedenen Gegenstände durchprobieren und dabei **mit dem letzten** Gegenstand beginnen. Die übergebenen Felder dürfen Sie dabei *niemals* verändern!
Vervollständigen Sie zuerst die Methode `packeSack()` so, dass die darin aufgerufene Rekursionsmethode `packeSackHelfer()` mit der Betrachtung des richtigen Gegenstands beginnt. Ihre `packeSackHelfer()` muss zwei unterschiedliche Fälle in *genau* dieser Reihenfolge betrachten:

- Welchen Wert hat der Rucksack, wenn man ihn **ohne** `naechsterGegenstand` befüllt?
- Welchen Wert hat der Rucksack, wenn man ihn **mit** `naechsterGegenstand` befüllt (sofern dieser überhaupt noch in den Rucksack passt)?

ACHTUNG – WICHTIG: Gleich zu Beginn der Methode `packeSackHelfer()` **müssen** Sie unbedingt die Methode `rpp.packeSackHelferAufgerufen()` aufrufen und ihr die Argumente des aktuellen Aufrufs übergeben. Diese Methode dient der Überprüfung Ihrer Implementierung und ist für die automatische Bewertung im EST sehr wichtig! Testen Sie Ihre Lösung mit dem vorgegebenen öffentlichen Test und ergänzen Sie bei Bedarf weitere Testfälle!
Geben Sie Ihre Lösung als `RucksackPacken.java` über EST ab.

Einzelaufgabe 4.2: Soziales Netzwerk — 15 EP

In einem sozialen Netzwerk sind n Nutzer registriert. Jeder Nutzer kann mit beliebig vielen anderen Nutzern befreundet sein. Die Freundschaftsbeziehung ist *reflexiv*, da jeder Nutzer mit sich selbst befreundet ist und *symmetrisch*, da sie auf Gegenseitigkeit beruht: Wenn Nutzer $\mathcal{A}$ mit Nutzer $\mathcal{B}$ befreundet ist, dann ist auch Nutzer $\mathcal{B}$ mit Nutzer $\mathcal{A}$ befreundet. Die *Distanz* $\delta_{\mathcal{Y}}^{\mathcal{X}}$ zwischen zwei Personen $\mathcal{X}$ und $\mathcal{Y}$ im Netzwerk sei dabei die minimale Anzahl der Freundschaften, über die Person $\mathcal{X}$ mit Person $\mathcal{Y}$ verbunden ist.
In dieser Aufgabe soll ein einfaches soziales Netzwerk modelliert werden, indem die Methoden zum Erstellen und Löschen von Freundschaftsbeziehungen implementiert und die Distanzen zwischen den einzelnen Nutzern untersucht werden. Die n Nutzer werden anhand ihrer eindeutigen Kennzahlen im Intervall $\{0, n-1\}$ verwaltet. Die Freundschaftsbeziehungen werden in der Freundschaftstabelle (einem zweidimensionalen Feld vom Typ `boolean`) gespeichert, wobei die Feld-Größe in beiden Dimensionen der Gesamtzahl der Nutzer entspricht und der Wert `true` an der Stelle i, j die Freundschaft zwischen dem i-ten und dem j-ten Nutzer widerspiegelt.

Beispiel: Die unten angegebene Freundschaftstabelle speichert die Freundschaften zwischen den Nutzern 0 und 1 sowie zwischen 1 und 2. Die Distanz δ_2^0 zwischen den Nutzern 0 und 2 beträgt demnach $\delta_2^0 = 2$, da sie nicht direkt, sondern lediglich über den Nutzer 1 „befreundet“ sind.

Nutzer	0	1	2
0	`true`	`true`	`false`
1	`true`	`true`	`true`
2	`false`	`true`	`true`

a) Ergänzen Sie in der Klasse `SozialesNetzwerk` die Methode `initialisiere`, die für die übergebene Anzahl n das Netzwerk für maximal n Nutzer initialisiert. Dabei soll das Array `freundschaft` für die Freundschaftsbeziehungen mit der entsprechenden Größe angelegt werden.

b) Implementieren Sie die Methode `fuegeNutzerHinzu`, die das Hinzufügen eines Nutzers ermöglichen soll. Dem neuen Nutzer soll die kleinste bisher freie Nutzerkennung zugewiesen werden, die die Methode zurückgibt. Auf den möglichen Überlauf der in der `initialisiere`-Methode festgelegten maximalen Größe des Netzwerks muss an dieser Stelle nicht geachtet werden.

c) Ergänzen Sie die Methoden `fuegeFreundschaftHinzu`, `entferneFreundschaft` und `testeFreundschaft`, die für das Erstellen, das Löschen und das Prüfen von Freundschaftsbeziehungen zuständig sind. Jede dieser Methoden soll jeweils zwei `int`s als Parameter übernehmen, die den Nutzerkennungen entsprechen.

d) Implementieren Sie eine **rekursive** Methode `istErreichbar`, die für zwei Nutzerkennungen `id0` bzw. `id1` und eine Distanz `e` berechnet, ob die entsprechenden Nutzer innerhalb der Distanz[1] `e` untereinander „bekannt“ sind. Verwenden Sie eine naive Rekursion ***ohne Optimierungen*** (d.h. brechen Sie die Rekursion erst ab, wenn die Freundschaft erkannt wurde oder die Distanzobergrenze nicht eingehalten werden kann – aber nicht früher!).

ACHTUNG – WICHTIG: Gleich zu Beginn Ihrer Methode `istErreichbar()` **müssen** Sie unbedingt die Methode `snmp.istErreichbar()` aufrufen und ihr dabei die Argumente des aktuellen Aufrufs weiterreichen. Diese Methode dient der Überprüfung Ihrer Implementierung und ist für die automatische Bewertung im EST sehr wichtig! Testen Sie Ihre Lösung mit dem vorgegebenen öffentlichen Test und ergänzen Sie bei Bedarf weitere Testfälle! Geben Sie Ihre Lösung als `SozialesNetzwerk.java` über EST ab.

[1] Bemerkung: Die durch Freundschaft/Bekanntschaft definierten Distanzen in echten sozialen Netzwerken wurden durch mehrere Forschungen untersucht, bekannt unter dem Namen „*Small World Experiments*“. Die Ergebnisse haben gezeigt, dass der Mittelwert solcher Distanzen bei gerade mal 6 liegt.

Gruppenaufgabe 4.3: Induktionsbeweis — 11 GP

Gegeben sei folgende Methode `a(n)` für $n \geq 0$:

```
long a(long n) {
    if (n == 0) {
        return 1;
    } else if (n == 1) {
        return 9;
    } else {
        return 2 * a(n - 1) - a(n - 2) + 8;
    }
}
```

a) Beweisen Sie *formal* mittels *vollständiger Induktion* die Korrektheit der Methode `a(n)` hinsichtlich:

$$\forall n \geq 0 : \texttt{a(n)} \equiv (2 \cdot n + 1)^2$$

Sie dürfen bei Ihrem Beweis vereinfachend annehmen, dass die benutzten Datentypen unbeschränkt sind und daher keine Überläufe auftreten. Geben Sie Ihren Beweis möglichst detailliert an und führen Sie *alle* Voraussetzungen und Zwischenschritte *explizit* auf.

b) Ergänzen Sie nun Ihren Beweis um eine geeignete Terminierungsfunktion für `a(n)` und begründen Sie Ihre Wahl!

Geben Sie Ihre Lösung als `Induktion.pdf` über EST ab.

Gruppenaufgabe 4.4: Rekursive schriftliche Multiplikation — 24 GP

In der Grundschule lernen Kinder das Multiplizieren von Zahlen mit einem schriftlichen Verfahren. Dieses Verfahren funktioniert im Dezimal- sowie im Binärsystem gleichermaßen. Frischen Sie bei Bedarf Ihr Wissen über das Verfahren der *schriftlichen Multiplikation* anhand der aufgeführten Beispielrechnungen mit den Zahlen $(12)_{10} = (1100)_2$ und $(25)_{10} = (11001)_2$ auf.

1	2	$*$	2	5
		2	4	
		$_1$	6	0
		3	0	0

1	1	0	0	$*$	1	1	0	0	1
		1	1	0	0				
			1	1	0	0			
				0	0	0	0		
					0	0	0	0	
	$_1$	$_1$				1	1	0	0
	1	0	0	1	0	1	1	0	0

Bei diesem Verfahren ist eine hohe Anzahl von Rechenoperationen (Multiplikation und Addition) nötig. Daher entwickeln Sie in dieser Aufgabe ein effizienteres rekursives Verfahren. Dieses beruht auf der Binärdarstellung einer Zahl. Sei eine beliebige Zahl x mit n Bits gegeben. Mit den Zahlen x_0, welche aus den unteren $\lceil \frac{n}{2} \rceil$ Bits von x besteht, und x_1 bestehend aus den oberen $\lfloor \frac{n}{2} \rfloor$ Bits von x, lässt sich die Zahl x wie folgt darstellen:

$$x = \underbrace{x_1}_{\lfloor \frac{n}{2} \rfloor \text{ Bits}} * 2^{\lceil \frac{n}{2} \rceil} + \underbrace{x_0}_{\lceil \frac{n}{2} \rceil \text{ Bits}} \tag{1}$$

Zum Beispiel ergibt sich für die Zahl 300 folgende Aufteilung:

$$x = (300)_{10} = (\underbrace{1001}_{x_1}\underbrace{01100}_{x_0})_2 = (1001)_2 * 2^{\lceil \frac{9}{2} \rceil} + (01100)_2$$

Die Multiplikation zwei Zahlen x und y ergibt sich wie folgt (sofern beide Zahlen mit n Bits dargestellt sind):

$$\begin{aligned} x * y &= \left(x_1 * 2^{\lceil \frac{n}{2} \rceil} + x_0\right) * \left(y_1 * 2^{\lceil \frac{n}{2} \rceil} + y_0\right) \\ &= \underbrace{(x_1 y_1)}_{\texttt{up}} * 2^{2\lceil \frac{n}{2} \rceil} + \underbrace{(x_1 y_0 + x_0 y_1)}_{\texttt{mid}} * 2^{\lceil \frac{n}{2} \rceil} + \underbrace{(x_0 y_0)}_{\texttt{low}} \end{aligned} \tag{2}$$

Ihre Aufgabe ist die Implementierung des Verfahrens. Laden Sie dazu die vorgegebene Datei herunter und ergänzen Sie diese wie folgt:

a) Vervollständigen Sie die Methode `int countUsedBits(long num)`, die **rekursiv** die Anzahl der genutzten Bits in der Binärdarstellung der nicht-negativen Zahl `num` ausgibt.

b) Vervollständigen Sie `long extractLowerBits(int lowerBits, long num)` so, dass sie die unteren `lowerBits` Bits aus der Zahl `num` extrahiert und als `long` zurückgibt.

c) Vervollständigen Sie `long extractHigherBits(int lowerBits, long num)` so, dass sie die oberen (`n - lowerBits`) Bits aus der Zahl `num` extrahiert und zurückgibt.

d) Vervollständigen Sie `long combine(long up, long mid, long low, int bits)`, die die drei Teile `up`, `mid` und `low` eines Multiplikationsergebnisses entsprechend der Gleichung (2) zu einer Zahl zusammenrechnet. Der Parameter `bits` entspricht der Anzahl Bits im Wert x_0 bzw. y_0 (also der Anzahl der Bits in `mid` bzw. `low`).

e) Implementieren Sie in der Methode `writtenMulRec4(int x, int y)` ein **rekursives** Verfahren zur Multiplikation der beiden Zahlen `x` und `y`, das auf der Gleichung (2) beruht. Ihre Rekursion soll abbrechen, sobald Sie das Ergebnis ohne Durchführen einer Multiplikation aus den übergebenen Werten `x` und `y` bestimmen können. Nutzen Sie Ihre eigenen Methoden aus den vorherigen Teilaufgaben. Beachten Sie den Fall, dass die Parameter `x` und `y` über eine unterschiedliche Anzahl von Bits verfügen können (siehe Binärdarstellungen von 12 und 25 am Anfang der Aufgabe) – in diesem Fall müssen Sie größere Anzahl für die niederwertigen Bits beider Zahlen wählen!

f) Verbessern Sie nun die vorhergehende Lösung, indem Sie die Anzahl der rekursiven Aufrufe optimieren. Betrachten Sie das Ergebnis der Multiplikation $(x_0 + x_1) * (y_0 + y_1)$ und kombinieren Sie es mit der Gleichung (2). Leiten Sie aus Ihren Erkenntnissen ein **rekursives** Multiplikationsverfahren ab, welches nur noch **drei** rekursive Aufrufe benötigt. Implementieren Sie das Verfahren in der Methode `writtenMulRec3(int x, int y)`.

Geben Sie Ihre `WrittenRecursiveMultiplication.java` im EST ab.

Wichtige Hinweise:

- Nutzen Sie entweder BitShift-Operatoren und/oder das Masking-Verfahren zur Extraktion der Bits aus einer gegebenen Zahl.

- Sie können annehmen, dass die Eingaben für alle zu implementierenden Methoden nichtnegative Zahlen sind.
- Sie können davon ausgehen, dass die Eingaben für das Verfahren so gewählt sind, dass (bei korrekter Implementierung) keine Überläufe auftreten.

Achtung – Beschränkungen:

- **Beachten Sie, dass gegebenenfalls nur ein Teil dieser Beschränkungen automatisiert (d.h. vor Ende des Abgabezeitraums) geprüft wird.**
- Die Nutzung der Operatoren `*` und `/` ist im Rahmen dieser Aufgabe **untersagt**, da Sie ein eigenes Multiplikationsverfahren implementieren. Die nötige Multiplikation mit einer Potenz der Zahl 2 können Sie durch die Verwendung von Bit-Shift-Operatoren ausdrücken.
  (Die Nutzung der Operatoren + und − ist erlaubt.)
- Die Benutzung von Schleifen (`for`, `while`, `do`, etc.) ist **untersagt** (auch in Teilaufgabe a)!
- Es sind **sämtliche** Aufrufe in die Java-API **untersagt**, insbesondere `Math` und `Integer`.

25 EP + 35 GP = 60 Punkte