Prof. Dr. F. Sadlo *Heidelberg, 4. November 2015*

Übungsblatt 3

Anmerkungen: Bei der Lösung der Programmieraufgaben dürfen ausschließlich die C++-Konstrukte verwendet werden, die bisher in der Vorlesung behandelt wurden, d. h. keine Schleifen, Variablenzuweisungen, usw. sondern ausschließlich Funktionsdefinitionen, rekursive Aufrufe und Funktionen des auf der Website der Veranstaltung zur Verfügung gestellten Headers `fcpp.hh`, wie z. B. `cond`.

Aufgabe 2.1 Algorithmische Komplexität [10 Punkte]

a) Ein Programm benötigt zur Bearbeitung einer Menge von Daten der Anzahl n 4 Sekunden auf einem Computer. Das Programm habe die algorithmische Komplexität $f(n)$. Wenn sich die Menge der Eingabedaten auf $2n$ verdoppelt, wie lange läuft dann das Programm wenn seine algorithmische Komplexität $f(n)$

i) $\operatorname{ld} n$ ii) n iii) $n \operatorname{ld} n$ iv) n^3 v) 2^n

beträgt? Wie sieht es aus für eine Ausgangslaufzeit von 10 und 100 Sekunden?

b) Sortieren Sie die folgenden Funktionen danach wie schnell sie mit n wachsen. Beginnen Sie mit der langsamsten.

$$n^{\log n} \quad c^n \quad c^{(c^n)} \quad \log\log n \quad n^\epsilon \quad \log n \quad 1 \quad n^n \quad n^c$$

Hier gilt $0 < \epsilon < 1 < c$. Aus Aufgabe c) folgt, dass es nicht entscheidend ist welche Basis der Logarithmus hat.

c) Beweisen Sie folgende Behauptungen:

1. $x^a = O(x^b)$ genau dann, wenn $a - b \leq 0$.
2. $\log_a(x) = \Theta(\log_b(x))$ für alle a, b.
3. $a^x = O(b^x)$ genau dann, wenn $0 \leq a \leq b$.

Aufgabe 2.2 Binomialkoeffizient [10 Punkte]

In der Vorlesung haben Sie mit den Fibonaccizahlen ein Beispiel für eine Rekursionformel mit einem Argument n kennengelernt. Ein weiteres Beispiel ist der Binomialkoeffizient

$$\begin{aligned} B_{n,0} &= B_{n,n} = 1 && n \geq 0 \\ B_{n,k} &= B_{n-1,k-1} + B_{n-1,k} && 0 < k < n \end{aligned} \tag{1}$$

mit zwei Argumenten n und k. Dieser lässt sich grafisch in Form des Pascalschen Dreiecks veranschaulichen:

				$B_{0,0}$								1							
			$B_{1,0}$		$B_{1,1}$						1		1					
		$B_{2,0}$		$B_{2,1}$		$B_{2,2}$				1		2		1				
	$B_{3,0}$		$B_{3,1}$		$B_{3,2}$		$B_{3,3}$			1		3		3		1		
$B_{4,0}$		$B_{4,1}$		$B_{4,2}$		$B_{4,3}$		$B_{4,4}$		1		4		6		4		1

Eine explizite Formel zur Berechnung des Binomialkoeffizienten ist gegeben durch

$$B_{n,k} = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2}$$

a) Schreiben Sie ein Programm, das den Binomialkoeffizient für beliebige n und k nach der in (1) gegebenen rekursive Formel berechnet. Benutzen Sie die Funktionen `enter_int` oder `readarg_int`, um n und k von der Standardeingabe einzulesen oder als Kommandozeilenparameter zu übergeben.

Benutzen Sie die Unix-Funktion `time`, um die Laufzeit Ihres Programms für verschiedene Kombinationen von n und k zu analysieren. Geben Sie eine Kombination für n und k an, bei der Ihr Programm länger als 10 Sekunden für die Berechnung benötigt. Was liefert Ihr Programm für $n = 34$, $k = 18$? Können Sie dieses Ergebnis erklären?

b) In der Vorlesung wurde der (exponentielle) Rechenaufwand für die rekursive Berechnung der Fibonaccizahlen ermittelt. Versuchen Sie nun einen Ausdruck $A_{n,k}$ zur Beschreibung der Komplexität der rekursiven Berechnung des Binomialkoeffizienten zu finden.

c) Schreiben Sie nun ein Programm, das schneller arbeitet als das aus a). Benutzen Sie hierfür die explizite Formulierung aus (2). Welche asymptotische Komplexität hat Ihre schnellere Variante?

Vergleichen Sie die beiden Programme aus a) und c) hinsichtlich der Geschwindigkeit und der berechneten Ergebnisse. Was stellen Sie für höhere Werte fest? Können Sie dies erklären?

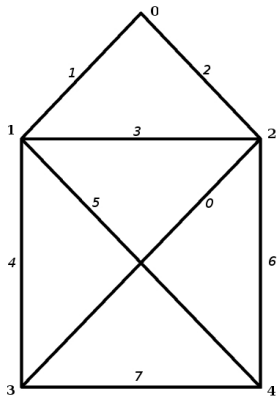
d) Welchen Aufwand hat ein Algorithmus, der die ersten n Zeilen des Pascalschen Dreiecks auf den Bildschirm ausgibt? Vergleichen Sie den Aufwand mit dem aus a) und versuchen Sie eine Erklärung für den Unterschied zu geben.

Zusatzaufgabe I Haus vom Nikolaus [✍ 5 Punkte]

Die Bearbeitung dieser Aufgabe ist freiwillig. Die Punktzahl dieser Aufgabe zählt am Semesterende nicht zur Maximalpunktzahl. Sie können also ein kleines Punktepolster anlegen.

Das Ziel dieser Aufgabe besteht darin, ein Programm zu schreiben, welches mit Hilfe der funktionalen Programmierung ermittelt, wieviele Möglichkeiten es – ausgehend von einem bestimmten Startknoten – gibt, das Haus vom Nikolaus zu lösen. Sollten Sie das Haus vom Nikolaus nicht kennen, informieren Sie sich unter `https://de.wikipedia.org/wiki/Haus_vom_Nikolaus` und lassen Sie sich von den dort verlinkten Implementierungstipps weder verwirren noch leiten.

Bei der Lösung des Problems ist eine Indizierung der 5 Knoten und 8 Kanten unumgänglich. Diese ist willkürlich und daher prinzipiell Ihnen überlassen, allerdings bieten gewisse Indizierungen Vorteile gegenüber anderen. Wir empfehlen Ihnen daher folgende Wahl:



Der Vorteil dieser Indizierung liegt im Zusammenhang zwischen Knoten- und Kantenindizierung. Indizierungen ohne solchen Zusammenhang hätten zwangsläufig ein Problem damit, zwei Knoten den Index ihrer gemeinsamen Kante zuzuordnen. In diesem Fall ist der Kantenindex $k(i, j)$ zu zwei Knoten i und j gegeben durch

$$k(i,j) = \begin{cases} ERR & \text{wenn } i = j \\ ERR & \text{wenn } (i,j) \in \{(0,3),(3,0),(0,4),(4,0)\} \\ 0 & \text{wenn } (i,j) \in \{(2,3),(3,2)\} \\ i+j & \text{sonst} \end{cases}$$

Dabei ist ERR ein beliebig definierter Fehlercode, der anzeigt, dass keine Kante zwischen den Knoten i und j existiert. Die zusätzliche Abfrage des Indexpaares $(2, 3)$ ist notwendig, um die Eindeutigkeit des Indexes zu gewährleisten ($1 + 4 = 2 + 3 = 5$).

Im Laufe des Programms ist es ebenfalls unumgänglich sich zu merken, welche Kanten bereits besucht (sprich: gezeichnet) worden sind. Verwenden Sie hierzu eine Zustandsvariable vom Typ **`unsigned char`**. Dieser Datentyp verfügt über genau 8 Bits. Ein gesetztes Bit an der entsprechenden Stelle gibt also an, dass die zu diesem Kantenindex korrespondierende Kante bereits besucht wurde. Lesen Sie sich im Internet oder dem Lehrbuch Ihrer Wahl die Funktionsweise der Bitoperatoren $\&$, $|$, $<<$ und $>>$ durch. Diese erlauben Modifizierung und Auslesen einzelner Bits – also genau die Operationen, die dem Nachschauen, ob eine Kante gezeichnet ist, und dem Zeichnen einer Kante entsprechen.

Die Anzahl der Möglichkeiten das Haus vom Nikolaus zu zeichnen wird schließlich in einem rekursiven Algorithmus ermittelt, welcher alle möglichen Wege überprüft. Diese Methode wird als Backtracking bezeichnet. Ist es auf einem Weg möglich, das Haus vom Nikolaus komplett zu zeichnen, so werden alle Bits der Zustandvariable 1 und der Gesamtwert somit 255 sein.

Aufgaben:

- Schreiben Sie eine Funktion `int kantenindex(int i, int j)`, die zu zwei gegebenen Knotenindizes den entsprechenden Kantenindex ermittelt.
- Schreiben Sie die Funktionen
  - `bool kante_besucht(int kante, unsigned char zustand)`
  - `unsigned char besuche_kante(int kante, unsigned char zustand)`

  welche mit Hilfe von Bitoperatoren die oben beschriebenen Aufgaben erledigen.
- Schreiben Sie eine Funktion `int anzahl(int knoten, unsigned char status, int naechster)` welche durch rekursiven Aufruf die Anzahl der Möglichkeiten zurückgibt, bei den durch `zustand` spezifizierten bereits besuchten Kanten, vom Knoten `knoten` ausgehend, das Haus vom Nikolaus korrekt zu zeichnen. Die Variable `naechster` gibt dabei an, welchen Knoten man als nächsten besuchen will. Übersteigt diese 4, also den höchsten Knotenindex, so lässt sich auf diesem Weg keine Möglichkeit mehr finden, das Haus zu zeichnen.
- Ein Aufruf mit `anzahl(startknoten, 0, 0)` liefert nun das gewünschte Ergebnis. Probieren Sie verschiedene Startknoten aus.

Bemerkung: Das Haus vom Nikolaus ist ein einfaches Beispiel eines Problems aus der Graphentheorie. Eine Möglichkeit, das Haus zu zeichnen, wird dort als Eulerweg bezeichnet.

Abgabe: 12. November 2015, 14:15 Uhr,
in die Zettelkästen in der Ecke zwischen HS 2 und Seifertraum (INF 288)
sowie per E-Mail (✉) an Ihren Tutor