

Programmierübungen

Wintersemester 2007/2008

11. Übungsblatt

29. Januar 2008

Abgabe bis Freitag, 1. Februar 23:59 Uhr

Die Abgabe Ihrer Bearbeitung können Sie im eClaus-System durchführen. Erarbeiten Sie Lösungsideen zu den Aufgaben möglichst in Kleingruppen. Es wird jedoch von Ihnen erwartet, dass jeder Teilnehmer eine eigene Lösung abgibt. Sollten kopierte Quelltexte abgegeben werden, so werden grundsätzlich alle Kopien mit 0 Punkten bewertet. In den Vortragsfolien der Programmierübungen oder im Skript zur Einführung in die Informatik abgedruckte Quelltexte können verwendet werden, müssen aber der Programmierrichtlinie entsprechend formatiert und kommentiert werden.

Beachten Sie die Programmierrichtlinie und kommentieren Sie Ihren Quelltext! Dokumentieren Sie unbedingt Ihre Lösungsidee in den Quelltext-Kommentaren.

<http://www.iste.uni-stuttgart.de/ps/Lehre/WS0708/inf-prokurs>

Bitte beachten Sie, dass zum Scheinerwerb unter anderem auf den letzten 4 Aufgabenblättern 50 % der Punkte erreicht werden müssen. Das letzte Aufgabenblatt wird Nummer 12 sein.

Aufgabe 11.1: Generische Pakete

(8 Punkte)

Im Kapitel 3.7.6 des Skripts zur „Einführung in die Informatik 1“ werden Suchbäume definiert. Erstellen Sie ein generisches Paket, das Suchbäume durch einen abstrakten Datentyp `Tree` anbietet. Als generische Parameter sollen ein Typ `Item_Type`, eine Vergleichsfunktion `"<"` und eine Prozedur `Visit` verwendet werden. Die Deklaration des Pakets sieht also wie folgt aus (die Deklarationen innerhalb des Pakets sind hier nicht angegeben):

```
generic
  type Item_Type is private ;

  with function "<"
    (Left  : in Item_Type ;
     Right : in Item_Type)
    return Boolean ;

  with procedure Visit
    (Element : in Item_Type) ;

package Search_Trees is

  -- ...

end Search_Trees ;
```

Fügen Sie Unterprogramme für folgende Operationen ein:

- Einfügen eines neuen Elements (des Typs `Item_Type`) in einen Suchbaum, ist das neue Element bereits im Baum enthalten, so wird es kein zweites Mal eingefügt.

Dieses Unterprogramm soll zusätzlich berechnen, wie viele Vergleiche zum Einfügen benötigt wurden,

- Abfrage, ob ein bestimmtes Objekt in einem Suchbaum enthalten ist; dieses Unterprogramm soll zusätzlich berechnen, wie viele Vergleiche zur Suche benötigt wurden,
- Berechnen der Höhe eines Suchbaums,
- Durchlauf durch einen Suchbaum in Inorder-Reihenfolge, wobei die Prozedur Visit mit jedem Element des Suchbaums als aktueller Parameter aufgerufen wird.

In den zu berechnenden Anzahlen von Vergleichen zählt jeder Aufruf des generischen Parameter-Operators "<" sowie jede Verwendung des Operators "=" für den Typ Item_Type. Sonstige Vergleiche, z. B. Tests auf null-Zeiger werden nicht mitgezählt.

Aufgabe 11.2: Generische Instanz

(6 Punkte)

Erstellen Sie ein Programm Date_Manager. Das Programm verwendet eine Instanz des generischen Pakets Search_Trees aus der vorigen Aufgabe. Als Item_Type soll der Typ Ada.Calendar.Time verwendet werden. Als Visit soll eine Prozedur übergeben werden, die ein Datum gefolgt von zwei Leerzeichen ausgibt.

Das Programm Date_Manager soll die Datei dates.txt einlesen und die darin enthaltenen Daten der Reihe nach in einen Suchbaum einfügen. Nach jedem Einfügen soll die Höhe des Baums und der gesamte Inhalt sortiert ausgegeben werden. Schließlich soll das Programm den Benutzer zur Eingabe eines weiteren Datums auffordern. Das Programm prüft, ob dieses Datum in dem Baum enthalten ist (ohne es einzufügen), gibt das Resultat aus und teilt die Anzahl der benötigten Vergleiche mit.

Format der Datei dates.txt: In jeder Zeile steht genau ein Datum im Format t.m.j wobei t, m und j als Zahlen mit beliebiger Anzahl Stellen dargestellt sind:

```
1.1.2007
2.5.2004
7.3.2008
1.12.1993
17.2.1970
2.5.2004
5.6.2005
```

Die Ausgabe des Programms könnte so aussehen (Hinweis: die Anzahl der Vergleiche kann gegenüber Ihrer Implementierung variieren, die Höhe des resultierenden Baums jedoch nicht):

```
Eingefügt: 01.01.2007, Anzahl Vergleiche: 0
Höhe des Baums: 0, Inhalt: 01.01.2007
```

```
Eingefügt: 02.05.2004, Anzahl Vergleiche: 1
Höhe des Baums: 1, Inhalt: 02.05.2004 01.01.2007
```

```
Eingefügt: 07.03.2008, Anzahl Vergleiche: 2
```

Höhe des Baums: 1, Inhalt: 02.05.2004 01.01.2007 07.03.2008

Eingefügt: 01.12.1993, Anzahl Vergleiche: 2

Höhe des Baums: 2, Inhalt: 01.12.1993 02.05.2004 01.01.2007
07.03.2008

Eingefügt: 17.02.1970, Anzahl Vergleiche: 3

Höhe des Baums: 3, Inhalt: 17.02.1970 01.12.1993 02.05.2004
01.01.2007 07.03.2008

Nicht eingefügt: 02.05.2004, Anzahl Vergleiche: 2

Höhe des Baums: 3, Inhalt: 17.02.1970 01.12.1993 02.05.2004
01.01.2007 07.03.2008

Eingefügt: 05.06.2005, Anzahl Vergleiche: 3

Höhe des Baums: 3, Inhalt: 17.02.1970 01.12.1993 02.05.2004
05.06.2005 01.01.2007 07.03.2008

Suchdatum: 1.12.1993

Datum ist im Baum enthalten, Anzahl Vergleiche: 4

Suchdatum: 30.12.2006

Datum nicht enthalten, Anzahl Vergleiche: 5

Suchdatum: 6.3.2008

Datum nicht enthalten, Anzahl Vergleiche: 3

Hinweis: Die Funktion `Ada.Strings.Fixed.Index` kann genutzt werden um die Positionen der Punkte in einem Datums-String zu finden.

Aufgabe 11.3: Erreichbarkeit in Graphen

(6 Punkte)

Schreiben Sie ein Programm `Reachability`, das zunächst einen Graph aus einer Datei `graph.txt` einliest und danach den Benutzer zur Eingabe einer Knoten-Id auffordert. Gültige Knoten-Ids sollen alle Texte sein, die keine Leerzeichen enthalten. Die eingelesene Knoten-Id identifiziert einen Start-Knoten in dem Graph. Verwenden Sie den Algorithmus zum Graphendurchlauf um von diesem Start-Knoten aus alle Knoten T_i aufzulisten, die von S aus erreichbar sind.

Die Datei `graph.txt` enthält eine Adjazenzmatrix für einen gerichteten Graph mit n Knoten für irgendeine nicht-negative ganze Zahl n : In der ersten Zeile stehen die Knoten-Ids aller n Knoten getrennt durch Leerzeichen (Spaltenüberschriften). Am Anfang der Zeile dürfen weitere Leerzeichen stehen, die keine Bedeutung haben.

Danach folgen n weitere Zeilen mit diesem Inhalt: Am Anfang jeder dieser Zeilen steht eine Knoten-Id `name` gefolgt von mindestens einem Leerzeichen und danach folgt eine Liste g_1, g_2, \dots, g_n nicht-negativer ganzer Zahlen, getrennt durch Leerzeichen. Eine Zahl $g_i > 0$ bedeutet, dass eine Kante mit Gewicht g_i von dem Knoten mit `name` zu dem Knoten mit Index i existiert. Ist ein $g_i = 0$, so fehlt die entsprechende Kante. Jeder `name` steht am Anfang nur einer dieser Zeilen.

Beispiel für die Datei `graph.txt`, siehe auch Abbildung 1:

	A	B	C	D	E	F	X	Y
A	2	0	1	0	2	4	0	0
B	0	0	0	0	2	0	0	0
C	0	0	0	0	3	2	0	0
D	0	0	0	0	0	0	0	0
E	3	0	0	5	0	0	0	0
F	0	0	0	2	0	0	0	0
X	0	0	0	0	0	0	0	2
Y	0	0	0	0	0	0	0	0

Beispiel für einen Programmablauf:

Bitte Startknoten eingeben: A
C, D, E, F

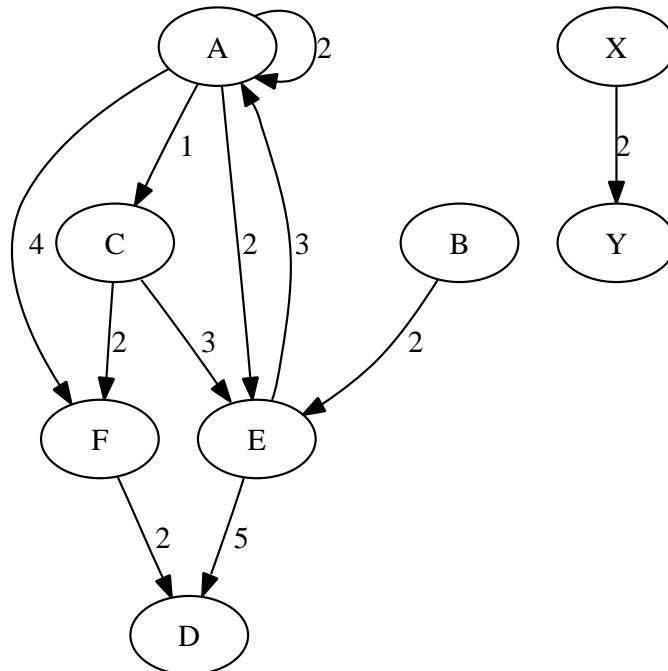


Abbildung 1: Beispielgraph