Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Grafische ondersteuning bij programmeeronderwijs

Jan Kuper, Universiteit Twente

Deze bijdrage beschrijft de ervaringen met een softwarepakket waarmee grafische ondersteuning wordt geboden bij het programmeeronderwijs. De software richt zich op het weergeven van grafische datastructuren (bomen, grafen), maar de positieve ervaringen ermee hebben geleid tot het ontwikkelen van grafische ondersteuning bij andere thema's, zoals wiskundige grafieken, natuurkundige simulatie, computational geometry.

Omdat het pakket is geschreven in een functionele taal, is het op dit moment alleen bruikbaar bij het onderwijs in functioneel programmeren. De benadering kan echter voor het onderwijs in elk programmeerparadigma zinvol zijn.

Boomstructuren

In het programmeeronderwijs worden, ongeacht het gekozen programmeer paradigma, boomstructuren door studenten in eerste instantie veelal als moeilijk ervaren. Studenten hebben vaak relatief veel tijd nodig de practicumopdrachten over dit onderwerp af te ronden waardoor er weinig opdrachten binnen één practicumbijeenkomst aan de orde gesteld kunnen worden. Het verband tussen de syntactische formulering van bomen, en de grafische intuïtie bleek voor veel studenten moeilijk te zijn. Om die reden is een programma ontwikkeld waarmee algemene boomstructuren eenvoudig kunnen worden afgebeeld op het scherm zodat studenten het resultaat van hun programma's zichtbaar kunnen maken. De hoop was dat daarmee een groter aantal opdrachten binnen kortere tijd kon worden gemaakt, waardoor het begrip bij studenten zich beter en sneller zou ontwikkelen. Die hoop is boven verwachting uitgekomen, het aantal opdrachten dat tijdens de eerste practicumbijeenkomsten over bomen wordt afgerond, is vele malen groter dan in het verleden, en bovendien is de moeilijkheidsgraad groter.

Het ondersteunende programma gaat uit van het volgende algemene type (*rosetree* genaamd):

```
roseTree
  ::= RoseNode
        [char]
        [roseTree]
```

Dat wil zeggen dat aan elke knoop (*node*) in een boom van dit type een string kan worden opgeslagen, en bovendien heeft elke knoop een willekeurig aantal subbomen. Een node is een *blad*, als hij geen subbomen heeft. Het type `roseTree` is dus vrij algemeen, en kan diverse structuren weergeven. Een beperking is het feit dat aan de knopen slechts een enkele string kan worden opgeslagen, maar omdat veel waarden (getallen, getallenparen, waarheidswaarden, etc) in stringnotatie kunnen worden weergegeven, wordt deze beperking pas relevant bij het weergeven van gestructureerde en/of multimedia waarden.

Om het programma te kunnen gebruiken, hebben studenten de beschikking over twee functies: `showTree` en `showTreeList` die als argument respectievelijk een enkele boom, of een lijst van bomen meekrijgen. Deze functies kunnen echter alléén op bomen van het type `roseTree` worden toegepast. Om die reden moeten hun eigen boomstructuren eerst omgezet worden naar bomen van het type `roseTree`. Didactisch gesproken blijkt dat echter veeleer een voordeel dan een nadeel te zijn: dergelijke omzet programmaatjes zijn eenvoudige eerste introducties in het schrijven van recursieve boomprogramma's. Als eerste opdracht moeten studenten een binaire bomen definiëren dat getallen aan de bladeren en aan de interne knopen kan bevatten. Een typedefinitie waarmee dergelijke bomen weergegeven kunnen worden is bijvoorbeeld:

```
Boom
  ::= Blad num
    | Knoop num boom boom
```

Studenten moeten vervolgens enkele voorbeeldexpressies van dit type geven, zoals:

```
t1 = Blad 37
t2 = Knoop 15 (Blad 20)
              (Blad 30)
t3 = Knoop 15 t1 t2
```

Boomexpressies lopen snel uit de hand en zijn dan volstrekt onleesbaar voor een menselijke lezer, maar de functie `zetom` om zo'n expressie om te zetten naar het type `roseTree` zodat hij grafisch kan worden getoond, is een zeer eenvoudig eerste voorbeeld van een recursieve functie op boomstructuren:

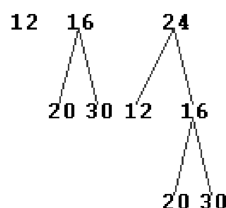
```
zetom (Blad n)
  = RoseNode (itoa n) []

zetom (Knoop n t1 t2)
  = RoseNode (itoa n)
      [zetom t1, zetom t2]
```

De aanroep

```
showTreeList
  (map zetom [t1, t2, t3 ])
```

levert de volgende afbeelding van de drie bomen naast elkaar:



Ditzelfde moeten de studenten doen met enkele varianten van boomtypes, zodat steeds opnieuw een zetom-functie geschreven moet worden. Daardoor wordt het patroon van recursieve functiedefinities op bomen heel snel herkend – en bovendien krijgen studenten snel behoefte aan een algemenere definitie van een zetom-functie, die alle verschillende varianten in één keer definieert, een goede gelegenheid om *generic funcitions* te intorduceren.

Door deze grafische weergave van bomen, kunnen studenten zich concentreren op het schrijven van typische boomfuncties zoals een functie om bomen te spiegelen, bewerkingen op de elementen in een boom uit te voeren, subbomen te selecteren, bomen sorteren, balanceren, etcetera. Het resultaat van hun functies op een voorbeeldboom `b`

kunnen zij onmiddellijk controleren door aanroepen als:

```
showTreeList
  [ b , zetom b
  , zetom (spiegel b) ]
```

Binnen een enkele practicum-bijeenkomst kunnen 15-20 van dergelijke opgaaven worden gemaakt, zodat na afloop studenten blindelings het recursieve schema kunnen produceren.

Grafen

Geïnspireerd door het succes van de grafische ondersteuning bij bomen is er vervolgens hetzelfde gedaan bij opgaven over *grafen*. Bij grafen echter, is geen recursieve definitie van de datastructuur voorhanden, en bovendien is het niet zo duidelijk hoe een graaf middels een expressie kan worden weergegeven. Daarom is een programma ontwikkeld waarmee studenten door muisklikken een graaf kunnen tekenen. Daarnaast hebben zij functies tot hun beschikking waarmee onderdelen van de graaf een andere kleur gegeven kunnen worden. Bij dit onderdeel komen *algebraïsche datatypes* op een zeer natuurlijke wijze naar voren omdat in de gebruikte programmeertaal (*Amanda*, ontwikkeld door Dick Bruin) grafische mogelijkheden daarmee zijn vormgegeven. Voorbeelden van algebraïsche constructoren voor de grafische toepassing zijn `GraphPolyLine`, waaraan een eental punten (gegeven als coördinatenparen) kunnen worden meegegeven die door deze constructor worden verbonden, en `GraphEllipse`, waarmee een ellips getekend kan worden, gegeven twee hoekpunten van de rechthoek die de ellips insluit.

Voor *inputevents* geldt hetzelfde: toetsaanslagen en muisklikken worden ook met waarden in een algebraïsch type weergegeven, bijvoorbeeld `KeyIn`, `MouseDown`.

Met behulp van dergelijke constructoren zijn de functies `drawNode`, `drawEdge`, `drawGraph` geschreven die aan studenten beschikbaar worden gesteld. Deze functies kunnen de punten van een graaf (*nodes*) in verschillende kleuren weergeven, en de bindingen tussen nodes (*edges*) in verschillende diktes en kleuren. Daarmee kunnen de resultaten van graafprogramma's zichtbaar gemaakt worden, zoals het bepalen van de buren van een node, het één voor één weergeven van alle paden van een gegeven node naar een andere node, het geven van dezelfde kleur aan alle nodes van een samenhangende subgraaf, etc.
Behalve het weergeven van het resultaat van een berekening, kan ook het *proces* van die berekening tot op zekere hoogte zichtbaar worden gemaakt. Een aansprekend voorbeeld hiervan is het bepalen van alle paden in een graaf van een gegeven node naar een andere node.

Wiskundig wordt een graaf gedefinieerd als een verzameling nodes samen met een verzameling geordende paren van die nodes. Als we de nodes omwille van de eenvoud als getallen representeren, is een mogelijke definitie van het type van grafen:

```
graph
  ::= {nodes::[num]
      ,edges::[(num,num)]
      }
```

Neem aan dat de functie `buren`, die de lijst van direct aangrenzende buren van een gegeven node in een graaf oplevert, al bestaat. Dan kunnen de paden van een gegeven node `p` naar een node `q` recursief worden bepaald door vanuit alle directe buren van `p` de paden naar `q` te bepalen, en de eerste stap vanuit `p` er voor te "plakken". De recursieve clause van de definitie van de functie `paden` kan dus als volgt geschreven worden:

```
paden g p q
 = concat
   [ map ((p,x):) ps
     | x <- buren p
     ; ps := paden g' x q
     ]
```

Hierin is `g` de graaf, en `p`, `q` zijn de nodes in `g` die door paden verbonden moeten worden. De node `x` varieert over alle directe buren van `p`, en `ps` is de lijst van alle paden van `x` naar `q` binnen de graaf `g'`. Deze is uit `g` ontstaan door de node `p` te verwijderen zodat cykels worden voorkomen. De edge `(p,x)` wordt met de standaardfunctie `map` voor elk pad in `ps` "geplakt".

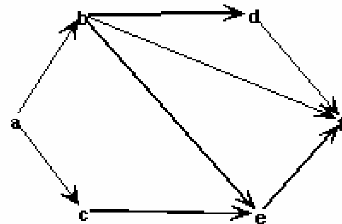
Uiteraard moeten de basisgevallen nog aan de definitie van `paden` worden toegevoegd. Daarin zijn twee gevallen te onderscheiden:

- als we in `q` "aangekomen" zijn, dwz als `p=q`: in dat geval is het resultaat de lijst met alleen het lege pad `[[]]`,
- als we vanuit `p` binnen `g'` niet "verder kunnen", dus als `p` geen buren heeft: in dat geval is het resultaat de lege lijst `[]`.

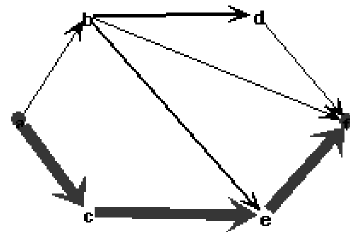
Een pad `pq` is nu een lijst van edges, en die kan visueel worden weergegeven door de functie `drawEdge` met de adequate parameters (bijvoorbeeld 3 voor de dikte van de edge) op elke edge in het pad toe te passen:

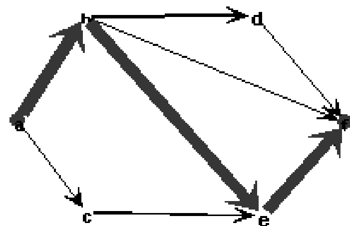
```
map (drawEdge 3 red) pq
```

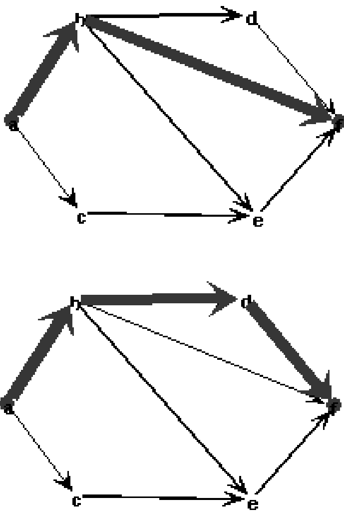
Als we dit algoritme testen op de volgende graaf:



en de paden van `a` (links) naar `f` (rechts) tonen, zien we het volgende resultaat:







Overige toepassingen

Vanwege de uiterst positieve ervaringen met deze ondersteunende methodes, en de indruk dat studenten een grafische output als stimulerend ervaren, is er ook moeite gedaan om andere opdrachten zodanig te kiezen dat een grafische weergave van de resultaten mogelijk is. Die opdrachten zijn gevonden op diverse gebieden:

- *Wiskundige grafieken*: een eenvoudige toepassing van de grafische mogelijkheden is om een functie te schrijven die, gegeven een wiskundige functie en een interval op de *x*-as de grafiek van die functie op dat interval tekent,
- *Computational geometry*: bepaal de convexe hull van een aantal punten in het platte vlak. Die punten moeten door muisklikken worden aangebracht, en de convexe hull moet getekend worden. Andere voorbeelden van opdrachten zijn het bepalen van alle triangulaties op een verzameling punten, het bepalen van een Delauney triangulatie op een verzameling punten, etc. Een bijkomend voordeel van dergelijke opdrachten is het feit dat studenten geconfronteerd worden met afrondingsfouten bij binaire getalrepresentaties. Dat geeft onverwachte effecten die in de grafische representatie nadrukkelijk zichtbaar zijn.
- *Natuurkundige simulatie*: het weergeven van bewegende objecten is voor studenten zeer stimulerend. Voorbeelden van eenvoudige opdrachten zijn stuiterende balletjes, verende elastiekjes, etc. Daarvoor moeten studenten overigens wel enig begrip hebben van de wiskundige beschrijving van bewegeingen. Bij stuiterende balletjes en verende elastiekjes gaat dat niet evrder dan een polynoom, maar mij een slinger-beweging wordt dat ingewikkelder. Een bijzonder aardig voorbeeld is het laten bewegen van een balletje tussen objecten waar het tussen heen en weer stuitert, met name als die objecten verschillende vormen kunnen aannemen,
- *Bord- en kaartspelen*: het acht-koninginnenprobleem is overbekend. Maar om de stukken daadwerkelijk op een schaakbord af te laten beelden geeft een extra dimensie aan de opgave. Het aantal variaties op opdrachten in de sfeer van dergelijke spelletjes is eindeloos. Studenten ervaren opdrachten van deze aard altijd weer als leuk.

Evaluatie

Er is geen systematische evaluatie gedaan van de verschillen tussen een practicum met bovenbeschreven middelen, maar er bestaat een sterke tot zeer sterke indruk dat er een aantal positieve kanten aan zitten. Die indruk is gebaseerd op de volgende observaties:

- student-assistenten kunnen vergelijken met de wijze waarop zij zelf het

betreffende onderwijs hebben ontvangen. Hun reacties waren zeer positief: “leuk, ik wou dat ik zelf het vak op deze wijze had gehad”,
- het is zichtbaar dat de directe feedback die met behulp van de grafische weergave bijvoorbeeld bij boom-programma’s wordt bereikt, een positief effect heeft op de begripsontwikkeling van de studenten,
- in de practicumzaal is zichtbaar dat studenten gemotiveerd en enthousiast met de opdrachten bezig zijn. Ze roepen elkaar: “kom eens kijken!”, en geven elkaar complimentjes over de resultaten. Daarvoor is vermoedelijk bevorderlijk dat niet iedereen dezelfde opdracht zit te maken, maar dat er enige variatie wordt aangebracht. Die variatie is eenvoudig aan te brengen, en bovendien biedt een grafische omgeving de mogelijkheid tot variatie in vormgeving door de studenten zelf. Daar maken ze uitvoerig gebruik van,
- ieder vak aan de opleiding Informatica in Enschede wordt geëvalueerd. Dit vak wordt in die evaluaties als zeer positief beoordeeld. Onder andere wordt veelvuldig het practicum genoemd als een zeer leerzame en stimulerende omgeving.
- bovenstaande observaties zijn gemaakt aan de hand van verschillende studentengroepen: eerstejaars informatica en eerstejaars bedrijfs-kundige informatica studenten, derde jaars wiskunde studenten, derdejaars informatica studenten.

Conclusie

De conclusie is dat grafische ondersteuning bij programmeer-onderwijs een gunstig effect heeft op de motivatie en op het leereffect bij studenten.