|  |
| --- |
| Högskolan i Gävle |
| Projektarbete |
| Funktionell programmering och diskret matematik, 7,5 hp  VT 19, period 2  GraphAnalyze, skapande och analys av riktade grafer  av  *Thomas Lundgren*  *900802-1835*  [*thomaslundgren@live.com*](mailto:thomaslundgren@live.com)  *M*attias Melchior  *920514-0818*  *mehasdagger@gmail.com* |
|  |
| **Institutionen för matematik, natur- och datavetenskap**  **Högskolan i Gävle**  S-801 76 Gävle, Sweden |
|  |
|  |

**Innehållsförteckning**

* **Inledning**

Projektarbetets syfte var att skriva ett program som genom ett kommando i skalet skulle ta emot en textfil som beskriver kanterna i en riktad graf och baserat på dessa kanter skriva ut information om grafen. Informationen som skulle skrivas ut var:

* Antal hörn
* Antal kanter
* Alla hörns valenser
* Antal cykler i grafen
* Hörnen som utgör den längsta stigen i grafen
* En grafisk representation av hela grafen (en lista över grafens kanter var tillräcklig)
* **Metod och utförande**

Arbetet inleddes med att bestämma vilken datastruktur som skulle användas för att lagra grafens hörn och kanter. Ett flertal alternativ undersöktes och till slut valdes en Map vars nycklar är hörn i grafen och värdet som associeras med nyckeln är en mängd (Set) som det aktuella hörnet har en riktad kant emot. Lösningen ansågs den bästa då datatypen Set inte tillåter dubbletter, det vill säga flera hörn med samma värden.

Nästa steg i utvecklingsprocessen var att skriva programmet – som kom att döpas till GraphAnalyze – som läser in en textfil, omvandlar innehållet i textfilen till en graf och sedan skriver ut information om grafen till skalet. Här betraktades graf-datastrukturen som en abstrakt datatyp då implementationen inte ännu hade skrivits.

Utvecklingen av datastrukturen Graph påbörjades sedan genom att först definiera signaturerna för alla funktioner som datastrukturen skulle komma att behöva. En testfil skrevs parallellt för att testa och kvalitetssäkra datastrukturen under utvecklingen.

* **Resultat**
* **Signaturen för den abstrakta datatypen**

Signaturen (interface:t) för datastrukturen Graph definieras i modulen Graph. Funktionerna och en kort beskrivning av dessa finnes i Tabell 1 nedan.

*Tabell 1: Signaturerna för alla typer och funktioner som definierats för datastrukturen Graph.*

|  |  |
| --- | --- |
| **Funktions-/Typsignatur** | **Beskrivning** |
| type key | Hörnens typ (t.ex. heltal, tecken, etc.) |
| type graph | Datastrukturen för grafen. |
| val empty : graph | Returnerar en tom graf. |
| val num\_vertices : graph int | Tar emot en graf och returnerar antalet kanter i grafen. |
| val add\_vertex : key graph graph | Tar emot ett hörn (värde) och en graf och returnerar den resulterande grafen innehållande det tillagda hörnet. |
| val add\_edge : key key graph graph | Tar emot två horn och en graf och returnerar den resulterande grafen innehållande de tillagda hörnen och kanten dem emellan. |
| val num\_edges : graph int | Tar emot en graf och returnerar antalet kanter i grafen. |
| val num\_cycles : graph int | Tar emot en graf och returnerar antalet cyckler i grafen. |
| val longest\_path : graph (key \* key) list | Tar emot en graf och returnerar en lista över de kanter som utgör den längsta stigen i grafen. |
| val vertex\_degree\_list : graph (key \* int) list | Tar emot en graf och returnerar en lista med tupler innehållande alla hörn och dess valenser. |
| val from\_list : (key \* key) list graph | Tar emot en lista med tupler som representerar kanter och returnerar en graf. |
| val to\_list : graph (key \* key) list | Tar emot en graf och returnerar en lista innehållande grafens kanter. |
| val string\_of\_graph : graph string | Tar emot en graf och returnerar en textsträngsrepresentation av grafen. |

Typerna har specificerats på så sätt att grafens inre datastrukturer inte blottas för användaren.

* **Funktorn Graph.Make**

Vid skapandet av en graf måste grafens hörns typ först specificeras. Detta görs genom funktorn Graph.Make. Funktorn definierades enligt följande:

Module Make(Ord: OrderedType) : Graph with type key = Ord.t

Make tar emot en modul av typen OrderedType som också definieras i modulen Graph. Modulen Orderedtype kräver att typen t och funktionerna compare och to\_string definieras.

* **Implementation av typerna key och graph**

Typen graph implementeras som en ”Adjacency map”, en tabell (Map) där nycklarna representerar hörn och värdena som associeras med nycklarna representerar hörnets utgrannar i grafen.

Tabellen skapas genom funktorn Map.Make, som tar emot datatypen OrderedType för att sätta typen hos ”tabellens” nycklar. Värdena som nycklarna pekar på är en mängd (ett Set) som skapas genom funktorn Set.Make. Båda dessa funktorer tar emot typen OrderedType, således får såväl nycklarna som elementen i varje Set typen key = t.

* **Implementation av grafens funktioner**

För implementationen av grafens funktioner, se den bifogade filen graph.ml.

* **Programmet GraphAnalyze**

Programmet GraphAnalyze startas genom skalet (kommandotolken) och tar emot en textfil som ett argument. Ett exempel på en textfil ges nedan:

A – B  
B – C  
C – A   
D

Kanter mellan två hörn beskrivs med två värden som förbinds med ett minustecken enligt ovan. Ett hörn som inte har någon utkant kan läggas till genom att skrivas ensamt på en egen rad (se tecknet ”D” ovan). För nuvarande är programmet hårdkodat till att endast ta emot hörn av typen OrderedType med typen t = Pervasives.Char. Textfilen ovan beskriver grafen nedan (Figur 1).



* **Testning av graph.ml**

Nedan är resultatet av exekvering av testfilen test\_graph.ml (se bifogat för denna fil):

vagrant@vm3110-2019sp:~/OCaml/Projekt/src$ ./test\_graph.native

After creating empty graph:

num\_edges should be zero: 0

num\_vertices should be zero: 0

Test add\_vertex:

Output:

'a':

'b':

After add\_edge 'a' 'b':

num\_edges should be 1: 1

num\_vertices should be 2: 2

Test from\_list by building graph from the list [('a', ['b';'d']); ('b', ['c']); ('c', ['a'])]:

num\_edges should be 4: 4

num\_vertices should be 4: 4

Test to\_list by printing the list made from the previously made graph:

'a': 'b', 'd',

'b': 'c',

'c': 'a',

'd':

Test vertex\_degree\_list on the graph:

Output should be:

'a': 3

'b': 2

'c': 2

'd': 1

Output:

'a': 3

'b': 2

'c': 2

'd': 1

Test to\_string:

Output should be:

'a': 'b', 'd'

'b': 'c'

'c': 'a'

'd':

Output:

'a': 'b', 'd'

'b': 'c'

'c': 'a'

'd':

Test num\_cycles:

Output should be: 2

Output: 2

* **Exekvering av GraphAnalyze**

Nedan är resultatet av exekvering av GraphAnalyze.ml med textfilen edges.txt som indata, se Figur 2. Båda dessa filer är bifogade med denna rapport.



* **Diskussion**
* **Implementation av num\_cycles**

Vi hade stora problem med att få funktionen num\_cycles att fungera och till slut hittade vi en fungerande implementation på hemsidan <https://ocaml.org/learn/tutorials/99problems.html> (se problem 81 och 82). Vi har använt oss av denna implementation. Vi förstod hur problemet skulle lösas: genom en depth first search utmed varje kant ut från ett hörn, där varje sökning som avslutades i samma hörn som den påbörjades skulle sparas till en lista. Sedan kunde listans längd enkelt tas fram. Tyvärr var det för komplicerat då vi inte känner oss tillräckligt bekväma med de datastrukturer vi valt att använda och det rekursiva tänket som krävs i funktionell programmering. Med det sagt så förstår vi hur den nuvarande koden fungerar, men vi lyckades inte skriva kod som använde sig av de datastrukturer som vi har valt att använda oss av.

* **Programmet GraphAnalyze**

Programmet är hårdkodat för att ta emot grafer med typen

module OrdChar = struct

type t = char

let compare = Char.compare

let to\_string t = Printf.sprintf "%C" t

end

Man kan istället tänka sig att användaren ges möjligheten att skapa egna typer och skicka in dem till programmet genom kommandotolken.

Programmet saknar också en del felhantering. För detta skulle man kunna använda datatypen Left/Right som diskuterats av kursens lärare. Tyvärr hann vi inte med att implementera detta.