Obejktorienterad analys

MatLi

Linda Rydström (linry786)
Li Sandsveden (lisa459)
Mari Ahlqvist (marah803)
Jonas Granholm (jongr157)
Emil Karlsson (emika583)
David Larsson (davla210)

5 november 2012

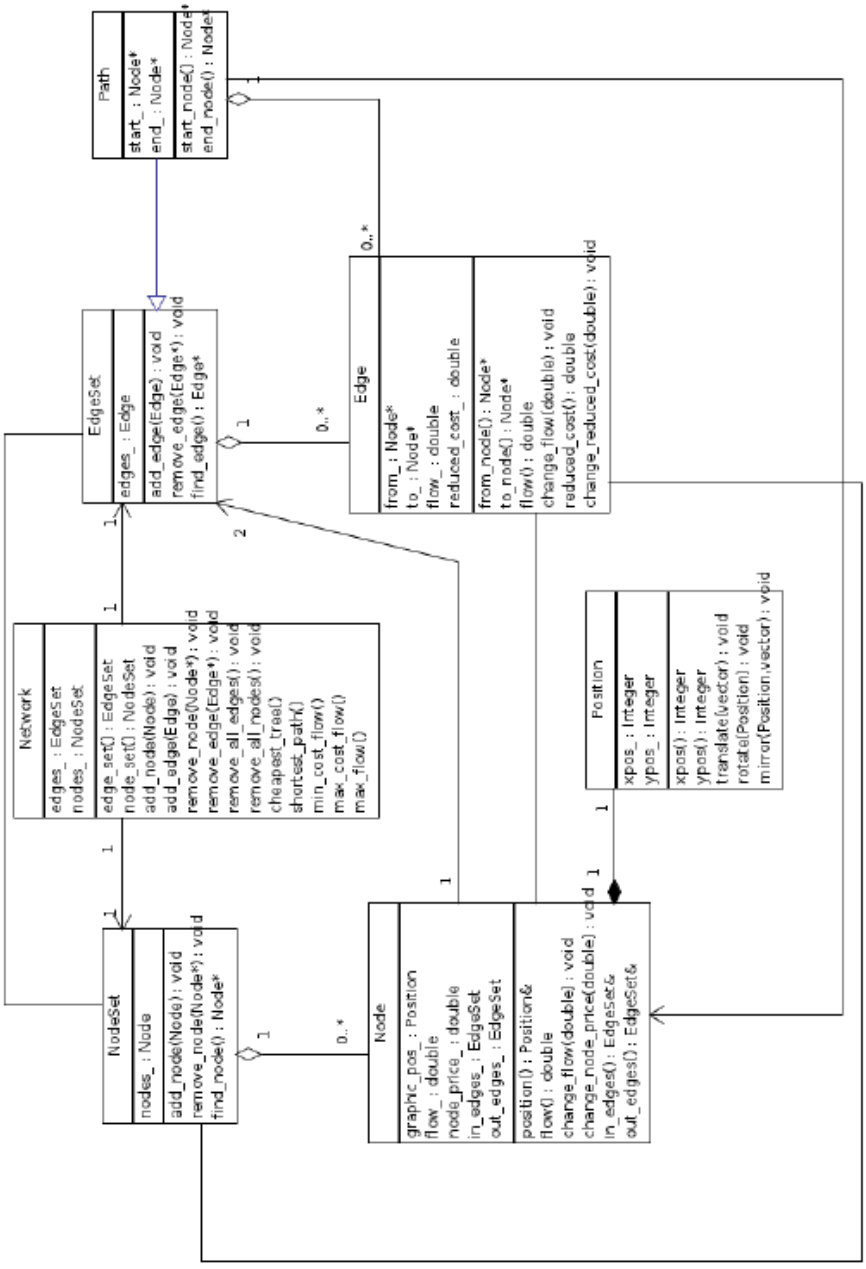
Innehåll

1 Sammanfattning

I det här dokumentet finns information om de klasser som bör ingå i projektet. Projektet är en del av kursen TDDC76, Programmering och datastrukturer, som ges på Institutionen för Datavetenskap vid Linköpings Universitet.

2 Klassdiagram

I figur 1 nedan följer ett klassdiagram som ska ge en övergripande blick över systemets klasser och hur de är relaterade till varandra.



Figur 1: Klassdiagram

3 Klassbeskrivningar

För att analysera vilka klasser som behövs för systemet användes klasskort, så kallade CRC-kort. I tabellerna nedan visas informationen från CRC-korten.

Den första kolumnen listar de ansvar som den aktuella klassen har, nästa listar vilka andra klasser som behöver komuniceras med för att uppfylla ansvarsområdet. Längst till höger finns det som brukar stå på CRC-kortens baksida, idéer på ingående variabler och funktioner i klassen.

Edge		
Hålla koll på bågdata	-	kostnad
Ändra bågdata	-	max- och minflöde
		från-nod och till-nod
		reducerad kostnad
		flöde

Node		
Hålla koll på noddata	Position	flöde
Hålla koll på ingående och utgående bågar	EdgeSet, Edge	grafisk position in- och utgående bågar
Ändra noddata	Position, Edge	nodpris

Position		
Hålla koll på grafisk position	-	x, y
Utföra affina transformationer	-	

EdgeSet		
Hålla koll på sammanlänkade bågar	Edge	kanter på något sätt...
Lägga till och ta bort bågar	Edge, Node	
Leta upp båge med viss egenskap	Edge	

NodeSet		
Hålla koll på sammanlänkade noder	Node	
Lägga till och ta bort noder	EdgeSet, Node	
Leta upp nod med viss egenskap	Node	

Path		
Hålla koll på vilka bågar som ingår	EdgeSet	
Känna till min- och maxflöde	EdgeSet, Edge	

Network		
Sammankoppla noder och bågar	EdgeSet, NodeSet	funktioner som löser nätverket
Lägga till och ta bort noder och bågar	EdgeSet, NodeSet	

4 Användningsfall

Systemet har bara en typ av användare. Nedan listas ett antal operationer som användaren kan vilja utföra. De tre sista användningsfallen (markerade med eventuellt) kommer bara att implementeras i mån av tid.

- Lägg till nod
- Ta bort nod
- Flytta nod
- Ändra noddata
- Rotera nod och grupper av noder
- Lägg till båge
- Ta bort båge
- Lägga till värden på allt (både, nod)
- Ta bort och ändra värden på allt (både, nod)
- Ta bort hela nätverket, *clear*
- Generera lösningar
  - billigaste vägproblem
  - min/max-kostnadsproblem
  - BUT (billigaste uppspännande träd)
  - maxflödesproblem
- Spara problem
- Spara lösningar
- Öppna gamla problem och lösningar
- Nollställa lösningen
- Uppdatera lösningen
- Kunna ångra sig, *ctrl + z* (eventuellt)
- Kunna stega framåt efter att man ångrat sig (evenutellt)
- Se historik och välja version (eventuellt)

4.1 Noggrannare beskrivning av användningsfall

Vi beskriver nedan fem av användningsfallen mer ingående.

Ta bort båge - Använder sig av *ändra nod*. När en båge tas bort ändras noddatan så att noderna vet att det inte längre finns en båge mellan dem.

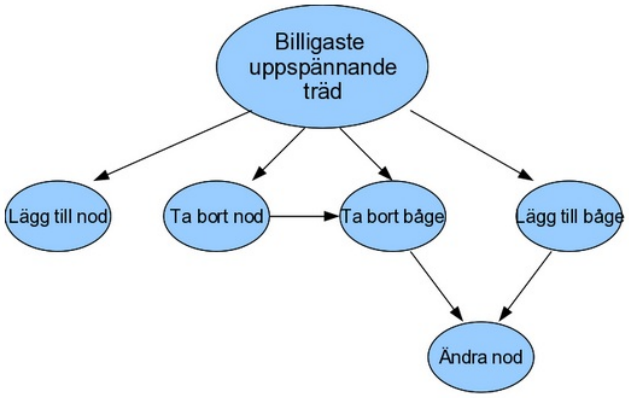
Ta bort nod - Använder sig av *ta bort båge*. Först tas alla inkommande och utgående bågar bort. Därefter tas själva noden bort.

Lägg till båge - Använder sig av *ändra nod*. Eftersom en båge ska gå från en nod till en annan så får respektive nod veta att de har fått en ny inkommande/utgående båge.

Ta bort hela nätverket, *clear* - Använder sig först av *ta bort båge* och sedan av *ta bort nod*. Se figur 2a.

Billigaste uppspännande träd - Använder sig av *lägg till nod, ta bort nod, lägg till båge* och *ta bort båge*. Se figur 2b.

(a) Användningsfallet *clear*



(b) Användningsfallet *Billigaste uppspännande träd (BUT)*

Figur 2: Användningsfall