Foredrag om Prosjektplanleggeren

Cindy Madeleine Svendsen Ceesay

March 13, 2017

Oppgaver tar en bestemt tid og krever en viss mengde arbeidskraft for å fullføre. Noen ganger kan det være vanskelig å bestemme seg for hvilke oppgaver man bør få gjort først. Andre ganger kan det være vanskelig å bestemme seg for hvilken oppgave man må få gjort når dersom man har et stort antall oppgaver som man må få gjort før man kan begynne med en viss oppgave. Særlig kan det virke vanskelig dersom alle oppgaver er et stort nett av oppgaver som alle avhenger av hverandre, eller i allefall virker det som det.

Prosjektplanleggeren er et program som hjelper nettopp deg med å løse denne oppgaven. Den beregner når en oppgave tidligst og senest kan avslutte kun ved å få oppgitt hvilke oppgaver den avhenger av, hvilke oppgaver som avhenger av den og hvor lang tid selve oppgaven tar. For å gjøre det enkelt for deg som bruker å lett gjenkjenne oppgavene har vi besluttet oss for at dere kan oppgi id og navn på hver oppgave.

Alt dere trenger for å benytte dere av programmet er å skrive inn en fil. På første linje skriver dere inn antall oppgaver dere har. Den andre linjen inneholder informasjon om den første oppgaven. Prosjektplanleggeren trenger å vite oppgavens id, oppgavens navn, hvor lang tid den tar, hvor mye arbeidskraft den behøver samt hvilken id den har på de oppgavene den avhenger av. Den tredje linjen inneholder informasjon om den andre oppgaven. Den fjerde linjen inneholder informasjon om den tredje oppgaven. Den femte linjen inneholder informasjon om den femte linjen og så videre og videre. Dere kan ha så mye luft dere ønsker selv mellom hver linje.

Dere lurer nå sikkert på hvordan jeg kom på å lage et slikt prosjekt? Helt ærlig er det ikke jeg som kom på selve grunnideen, det er skolen som utleverte en oppgave om å lage en oppgaveplanlegger.

Med en gang jeg fikk oppgaveteksten leste jeg over oppgaven, slik at jeg fikk en ide om hva jeg skulle lage og hva slags pensum som var relevant for oppgaven. Da jeg hadde et godt innblikk i hva jeg skulle lage begynte jeg å implementere de ulike interface metodene. Etter interfacet var ferdig begynte jeg å lage tok jeg en titt på de ulike illustrasjonene som var vedlagt oppgaven. Jeg tegnet deretter ulike test cases, slik at jeg fikk et dypere innblikk for hvordan koden skulle fungere og hva jeg var nødt til å ta høyde for ved hver situasjon.

Da jeg hadde en god ide om hvordan jeg skulle implementere denne oppgaven satt jeg med ned for å implementere kravspesifkasjonen. Først startet jeg med å skrive et interface, slik at jeg var sikker på at jeg hadde alle metoder som

skulle til for å tilfredstille oppgaveteksten. Jeg implementerte deretter en og en klasse for seg selv med alle metodene til interfacet. Jeg testet at alle metodene i interfacet tilfredstilte alle kravene i oppgaveteksten.

Til slutt implementerte jeg kodene til en og en metode. Under implementasjonsprosessen testet jeg hyppig at metoden ga riktig output i henhold til test tilstanden og at integrasjonen mellom metodene fungerte slik den skulle. Hver gang jeg var ferdig med et inkrement dobbeltsjekket jeg at metodene hadde riktig logisk semantisk betydning -dersom jeg utførte en endring testet jeg endringen jeg hadde utført og at ting fremdeles ga riktig output på testene.

Om det var noe jeg ble usikker på underveis forsøkte jeg å finne ut hva jeg kunne bli mer sikker på og hvordan jeg kunne bli mer sikker på det. Ofte endte jeg da opp med å tegne tegninger eller notere ned de ulike tilstandene til små deler av systemet. Jeg leste også oppgaveteksten på nytt igjen for å sjekke om det var noe jeg ikke hadde forstått, det var det sjeldent.

Kanskje lurer dere på hvordan selve programmet fungerer? Siden jeg tidligere har nevnt hvordan det fungerer, så tenker jeg nå å gå inn på de ulike algoritmene programmet anvender.

Du sjekker om programmet er realiserbart ved å beregne når hver oppgave tidligst kan avslutte og etter den har kalkulert ut dette så sjekker den ut om det finnes noen sykkel. Den beregner den tidligste avslutningspunktet ved å kalle metoden setearliesttime(). setearliesttime() legger first til alle oppgaver som ikke har noen avhengigheter i køen. Mens køen ikke er tom kan vi fjerne et element fra køen. Så legger vi det inn i stien. Vi setter så den slutttid til den nåværendes oppgave lik det tidligste sluttidspunktet for det nåværende oppgaven. For hver oppgave o som er avhengig av den nåværende oppgaven n setter vi det tidligste starttidspunktet til o lik endetiden til n og minker antallet forgjengere med en. Med en gang en oppgave ikke har flere forgjengere har legger vi den til i køen. Kort sagt har jeg anvendt topsort her.

Jeg beregner det seneste tidspunktet ved å bruke depth-first search på hver node. Hvis task t allerede er sett gjør den ingenting. Om den ikke er set så blir den satt til at den er sett og deretter blir en av to caser utført. Den første casen er at task t ikke har noen utkanter eller at utkantene er null. Task t sitt seneste startpnkt er satt lik den tidliste start. Den andre casen er at task t har minst en eksisterende outEdge w. La oss anta at vi har en variabel smallesttime. Sett smallesttime lik w tidliste starttidspunkt minus den estimerte tiden til v. Om smallest time er større enn v seneste startpunkt setter vi v seneste startpunkt lik smallest time.

Prosjektplanleggeren må selvfølgelig prøvekjøre prosjektet. Dette gjør koden ved å kalle metoden runProject(). Metoden beregner hvor mye manpower den trenger på den nåværende tidspunktet og bestemmet hvilken oppgave som bør starte og hvilken oppgave som bør avslutte. Først sorterer den alle oppgavene i oppgavelisten basert på deres tidligste start og deres estimerte tid. Vi setter så tiden lik den første oppgaven i listen. Først sjekker vi at det fremdeles er en oppgave som ikke har startet. Hvis det er en oppgave som ikke har startet enda og dens start tidspunkt kommer før den den oppgaven som skal ende starter den denne. Ellers avslutter vi første oppgave. Både når vi avslutter og starter

en oppgave oppdaterer vi den nåværende arbeidskraften. Det nåværende tidspunktet oppdateres når vi enten starter en oppgave og tidspunktet oppgaven starter på kommer etter nåværende tidspunkt eller når vi avslutter og tidspunktet oppgaven avslutter på kommer etter nåværende tidspunkt.

For å sortere oppgavelisten har jeg i mitt program valgt å anvende quicksort. Vi starter ved å sette i lik low og j lik høy. Vi setter deretter pivotTask mellom høy o lav. Metoden kalkulerer så når det tidligste avslutningstidspunktet for en pivot oppgaven. Metoden sorterer deretter på rekurive kall inntil begge oppgaver i avslutter tidligere enn oppgave j. Når man utfører sorteringen sammenligner alle oppgavene fra low til pivot oppgaven inntil en oppgave har senere tidspunkt enn pivotens ved å øke i. Vi sammenligner deretter alle oppgavene fra high til pivot ved å øke j. Når vi er ferdig med sammenligningen sjekker vi om i er mindre enn j, om dette er tilfelle øker vi både i og j. Om i er større enn j slutter vi å sortere. Vi starter deretter å kalle quicksort dersom oppgaven vi startet å sortere på ender tidligere og/eller tidligere enn den siste oppgaven vi utførte sortering på.