4 Compelxity and theoretical analysis

1. Tidskompleksitet

- Total tidskompleksitet er O (V + E), hvor V er antall oppgaver (som toppunkt for grafen) og E er antall kanter mellom oppgaven.
- **Realiserbarhet -**O (V + E): Dette er implementert ved bruk av DFS fra en oppgave som har null uegnet. DFS fortsetter til alle oppgave nodene blir besøkt og merket som ferdige, eller hvis det er noen sykel. Dermed på worst case algoritmen traverserer alle Oppgave nodene langs kantene.
- **Optimal tidsplan -** For å finne optimal tidsplan må vi sjekke Realiserbarhet før den, fordi det kan eksistere en sykel. Her starter algoritmen fra en oppgave A med uegnet null, dvs. uten avhengigheter og er merket som startpunktet for prosjektet. Deretter utføres DFS fra denne oppgave A-noden til oppgave B-noden, og samtidig beregnes tidlig_start og tidlig_ferdig attributt for oppgave B, og dette fortsetter til alle oppgave-noder blir besøkt.
- **Nyeste start, slack og kritisk oppgaver -** O (V + E). Denne algoritmen besøker alle oppgavene i planen en gang og sjekker hver kant en gang. Algoritmen bruker sluttidspunktet for et prosjekt, derfor må optimal tidsplan-algoritmen kjøres før denne her. Algoritmen starter med på vilkårlig oppgave. DFS kjører fra denne oppgaven A til en oppgave B som ikke har noen oppgaver som kommer etter, eller stammer fra oppgave B. Dette betyr at oppgave B er på slutten av planen og slack, sent_start, sent_ferdig og kristisk oppgave kan beregnes ved hjelp av prosjektets kjøretid. Rekursjonen går tilbake til høyere nivå på samme måte som den gjør i den optimal tidsplan-algortimen.
  Når alle slack-attributter er beregnet for avstamningene, vil den tidligste sent_start av avstamningene bli sammenlignet med denne oppgavens tidlig_ferdig. Slack-attributtene vil da bli beregnet med den verdien. Når denne rekursjonen er ferdig, vil algoritmen finne en oppgave som ikke har blitt kjørt gjennom, starte på nytt, frem til alle oppgavene er beregnet.
- **Simulering med heap implementering -** O (V*log (V)). Denne algoritmen lager en heap med oppgaven og bruker sent_start som nøkkel. Så lenge heapen ikke er tom, vises oppgaven med den laveste nøkkelen. Hvis gjeldende tid er lik nøkkelen til den gjeldende oppgaven, eller den nåværende tiden ikke er lik nøkkelen. Hvis den nåværende tiden tilsvarer sent_start, blir info skrevet ut og den samme oppgaven settes inn i heapen igjen med nøkkelen sent_ferdig av oppgaven. Hvis tiden tilsvarer sent_ferdig av oppgaven, blir info skrevet ut og oppgaven fjernet fra heapen.
- **Simulering** - O (V * minimum_ferdig_tid): Skriver ut alle nødvendige output på en veldig effektiv måte.

2. Krav

Inngangsgrafen må være asyklisk og rettet.

Grafer som blir dirigert kan lett realiseres ved hvilken oppgave som er avhengig av hvilken oppgave gjennom rettet kant.

○ Å være syklisk vil bekrefte at prosjektet er realiserbart, dvs. at det ikke dannes noen sykler mellom to eller flere oppgaver.

3. graf-algoritmer

DFS brukes gjennom hele oppgaven.