Toteutusdokumentti, Pistepeli

Eeva Nikkari

Algoritmin toimintaidea pseudokoodina:

```
function topoAlgoritmi(G)
    pino = Topological-Sort(G)
    while pino NOT EMPTY do
        u = pino.pop()
        for all v in vierus[u] do
            Relax(u,v)
        end for
    end while
end function

function Relax(u,v)
    if pisteSaalis[v] < pisteSaalis[u] + v.pisteArvo then
        pisteSaalis[v] = pisteSaalis[u] + v.pisteArvo
        reitti[v] = u
    end if
end function
```

Algoritmi kutsuu ensin verkolle G Topological-Sort(G) -algoritmia, jonka aikavaativuus on tunnetusti O(|V|+|E|), missä V on solmujen- ja E kaarien joukko. Sitten While-looppi käy läpi kerran jokaisen solmun verkossa eli suorittaa itsensä |V| kertaa. Loopissa käydään yhteensä kerran läpi jokainen solmu ja jokainen kaari. Relax-metodi on aikavaativuudeltaan vakio. Koko algoritmin aikavaativuudeksi tulee siis O(2(|V|+|E|))=**O(|V|+|E|)**. Käytännössä algoritmi vielä lopuksi etsii parhaan pistemäärän taulukosta pisteSaalis ja tulkitsee parhaan reitin taulukosta reitti. Nämä toiminnot ovat kuitenkin lineaarisia solmujen määrän suhteen, joten aikavaativuusarvio säilyy samana.

Algoritmi käyttää aputaulukkoja pisteSaalis, reitti sekä pinoa. Lisäksi Topological-Sort -algoritmi käyttää aputaulukkoa kayty. Kaikkien taulukkojen ja pinon tilavaativuus on O(|V|). Koko algoritmin tilavaativuus on siis O(4|V|)=**O(|V|)**.

Algoritmin eräs puutos on, että se ei tunnista, jos sille annetaan verkko, jossa on sykli. Algoritmi ei siis toimi oikein, jos verkossa on sykli. Käytännössä ongelmia tulee, kun algoritmi yrittää tulkita reittitaulukkoa ja algoritmi saattaa jäädä looppiin. Tähän kohtaan koodia on lisätty ominaisuus, että algoritmi huomaa, jos muodostuva polku on pitempi, kuin verkossa on solmuja ja lopettaa suorituksen. Näin algoritmi ei jää looppiin vaikka ei toimisikaan oikein annetulla verkolla.