Ugeopgave 11g

Hold 3, Gruppe 7: Silas Lundin, Agnes Galster, Natasja Juul og Helga Ibsen

January 28, 2022

Del 1

(a)

Lad $e_1, e_2, ..., e_{k-1}, e_k$ være en vej. Brug regnereglerne for logaritmefunktionen til at omskrive -log(w(e_1)w(e_2)...w(e_{k-1})w(e_k)) til en sum, hvor sandsynlighederne langs vejen indgår i hvert led.

Først bruger vi produktreglen for logaritmer, $log_a(x * y) = log_a(x) + log_a(y)$, til at omskrive vores udtryk til følgende :

$(-log(w(e_1))) + (-log(w(e_2))) + ... + (-log(w(e_{k-1}))) + (-log(w(e_k)))$

Minus fordi $log(w) < 0$ når $0 < w < 1$.

Så bruger vi reglen om negative logaritmer, som siger, at $-log_a(b) = n$ også kan skrives som $log_a(1/b) = n$:

$log(\frac{1}{w(e_1)}) + log(\frac{1}{w(e_2)}) + ... + log(\frac{1}{w(e_{k-1})}) + log(\frac{1}{w(e_k)})$

Hermed har vi omskrevet opgavens udtryk til en sum, hvor sandsynlighederne langs vejen $e_1, e_2, ..., e_{k-1}, e_k$ indgår i hvert led.

(b)

Brug udregningen i (a) til at lave en algoritme, der finder veje med størst sandsynlighed mellem knuder i G.

Vi lader vægtene i G være $-log(w(e_i))$ og finder veje med størst sandsynlighed mellem knuder i G på to forskellige måder:

1. Enten ændrer vi alle vægtene til $-log(w(e_i))$ og så kører `DJIKSTRA` og `RELAX` uændret.
2. Eller lader `DJIKSTRA` være uændret og ændrer den del af `RELAX`, hvor vægtene indgår:

Algorithm 1 RELAX-MODIFIED(u,v)

```
1: if (v.d > u.d - log(w(u,v))) then
2:     v.d = u.d - log(w(u,v))
3:     DECREASE-KEY(P,v,v.d)
4:     v.π = u
```

(c)

Argumenter for korrektheden af jeres algoritme, dvs. hvorfor de fundne veje har størst sansynlighed blandt veje mellem knuder i G.

Hvis vi lader `DJIKSTRA` være uændret og i stedet omregner vægtene i grafen fra sandsynligheder $w(e_i)$ til $-log(w(e_i))$, så finder algoritmen den korteste vej mellem begyndelsesknuden og slutknuden. Når vi har fundet den mindst mulig værdi af summen af de omregnede vægte, så har vi også fundet vejen med den største sandsynlighed.

1 Del 2

(a)

Forklar hvorfor kørsel af de to algoritmer $BFS(G, s)$ og $Dijkstra(G, w, s)$ giver det samme resultat når $w(e) = 1$ for alle $e \in E$, dvs. begge algoritmer resulterer i samme værdier $v.d$ i alle knuder. Brug definitionen af $\delta_{BFS}(s, v)$ og $\delta_{SP}(s, v)$.

$BFS(G, s)$ finder den korteste vej fra s til v - mindste antal kanter fra s til v.
$\delta_{BFS}(s, v) = \delta(s, u) + 1$

$Dijkstra(G, w, s)$ giver længden af den korteste sti.
$\delta_{SP}(s, v) = \delta(s, u) + w(u, v)$

Hvis $w(e) = 1$ for alle $e \in E$ får vi at $\delta_{SP}(s, v) = \delta(s, u) + w(u, v) = \delta_{SP}(s, v) = \delta(s, u) + 1$

og vi har hermed at $\delta_{BFS} = \delta_{SP}$, hvilket betyder at vi vil få det samme resultat ved kørsel af de to algoritmer når $w(e) = 1$ for alle $e \in E$

(b)

Forklar hvad det betyder at en knude er sort under udførslen af BFS og relater til hvad det betyder at en knude er i mængden S under udførslen af Dijkstra's algoritme.

Hvis en knude er sort under udførslen af BFS, betyder det, at knuden er "discovered", og at alle de verticer, som knuden er forbundet, til også er "discovered". Det må derfor betyde, at en sort knude er i den korteste sti fra s til u.

At en knude er i mængden S under udførslen af Dijkstra's algoritme betyder, at knuden er en del af den mængde af knuder, hvor algoritmen har fundet den korteste sti fra s till u for hvert u $\in S$. Så på en måde kan man sige, at de to ting "betyder" det samme, og at de beskrevne knuder er en del af den komplette korteste vej fra s til v.

(c)

Sammenlign effekten af kaldet Relax(u,v,w) i Dijkstra's algoritme med opdateringerne af $v.d$ og $v.\pi$ i BFS når $w(e) = 1$ for alle $e \in E$. Hvordan er placeringen af knuderne i køen Q i forhold til $v.d$ i de to algoritmer?

I BFS-proceduren sker følgende : knuden v males grå, $v.d$ ændres til $u.d+1$, $v.\pi$ bliver gjort lig med u, og v tilføjes i halen af køen Q.

Dijkstras brug af Relax-algoritmen sørger lige som BFS for en opdatering af værdien for både $v.d$ og $v.\pi$, *såfremt* den korteste vej kan optimeres ved at gå gennem u. Dog bruger Dijkstras algoritme en prioritetskø, der opbevarer data i en u-ordnet rækkefølge, hvilket står i modsætning til BFS.

Til sidst kan man sige, at idet vi har, at $w(e) = 1$ for alle $e \in E$, vil Relax ikke opdatere nogen værdier i dette tilfælde, hvilket BFS til gengæld gør.

(d)

Når $w(e) = 1$ for alle $e \in E$, vil I så vælge BFS eller Dijkstra's algoritme til at finde korteste veje i grafen? Argumenter for jeres valg.

Når $w(e) = 1$ for alle $e \in E$ i grafen, så vil vi vælge at bruge `BFS` til at finde de korteste veje, fordi der vil ikke være brug for at sammenligne vægtene og placere dem ind i en prioritetskø, som `DIJKSTRA` gør. Når vægtene er ens, så er det altså smartere at bruge `BFS`, som finder den korteste vej ved blot at tælle antallet af kanter imellem startknuden og slutknuden, dvs. afstanden svarer til antallet af vægte. Hvis vægtene derimod er forskellige, så giver det meget bedre mening at bruge `DIJKSTRA`.

På den anden side kan man sige at hvis vægtene er ens, så vil `DIJKSTRA` opføre sig ligesom `BFS` og lægge de opdaterede knuder ind i en kø Q på samme måde som `BFS` gør. Men da selve sammenligningsdelen i `RELAX` ikke er relevent for undersøgelsen af ensvægtede grafer, vil brugen af `BFS` være mere indlysende, ikke midst fordi den vil give os laverer køretid.