**Ülesanne 1**

**Kood on eraldi python failis “BFS”**

**Selgitus:**

Graph klass esindab graafi, millel on meetod add\_edge servade lisamiseks.

bfs meetod teostab laius-esmalt otsingu algusest antud graafis.

Näites on loodud graaf, kus 1 on seotud 2 ja 3-ga, 2 on seotud 4 ja 5-ga ning 3 on seotud 6-ga.

Rakendatakse BFS algoritmi, alustades tippu 1. Algoritm järgib igal sammul kõiki naabreid enne järgmise taseme tippude uurimist.

Tulemusena kuvatakse läbi käidud tipud: 1, 2, 3, 4, 5, 6.

**Ülesanne 2**

**Kood on eraldi python failis “DFS”**

**Aja- ja Ruumikomplekssus:**

Aja komplekssus: O(V + E), kus V on tippude arv ja E on servade arv. DFS külastab iga tippu ja iga serva üks kord.

Ruumikomplekssus: O(V), kus V on tippude arv. DFS kasutab sügavust ja hoiab rekursioonipilvedes külastatud tippude teavet. Ruumikomplekssus sõltub puu kõrgusest, mis võib olla kuni V.

**Ülesanne 3**

Dijkstra Algoritmi analüüs

**Kirjeldus:**

Dijkstra algoritm on lühima tee leidmise algoritm, mida kasutatakse graafides, kus iga serva kaal on mitte-negatiivne. Algoritm leiab lühima tee ühest alguspunktist kõikidesse teistesse punktidesse.

**Algoritm:**

Alusta alguspunktist.

Initsialiseeri lühim tee kaugus alguspunktist kõigisse teistesse punktidesse lõpmatuks, välja arvatud alguspunkt ise, mille tee kaugus on 0.

Vali hetkeliseks punktiks see punkt, mille tee kaugus on hetkel kõige väiksem.

Uuenda lühima tee kaugusi kõigi selle punkti naabrite suhtes, arvestades, et tee pidi kuni selle punktini on juba leitud.

Korda samme 3-4 seni, kuni kõik punktid on külastatud või lõpp-punktini jõutakse.

Efektiivsus:

**Efektiivne:**

Kui graaf on hõre ja servade arv on oluliselt väiksem tippude arvust.

Kui kaalud on mitte-negatiivsed, sest Dijkstra eeldab, et kõik kaalud on mitte-negatiivsed.

**Ebaefektiivne:**

Kui graaf on tihe ja servade arv on lähedane tippude arvule.

Kui kaalud võivad olla negatiivsed, sest Dijkstra ei käitle negatiivsete kaaludega graafides korrektselt. Negatiivsed kaalud võivad viia valede tulemusteni.

Olukorrad, Kus Dijkstra Algoritm Sobib Hästi:

Teede võrgud: Kasutatakse sageli linnade või asulatevaheliste lühimate teede leidmiseks.

Sidevõrgud: Efektiivne teekonna leidmisel andmeedastusvõrkudes.

Logistikasüsteemid: Optimaalsete marsuutide planeerimine transportimiseks.

Olukorrad, Kus Dijkstra Algoritm Võib Olla Ebaefektiivne:

Suurte graafide korral: Kui graaf on väga suur ja hõre on Dijkstra väga efektiivne, kuid tihe graaf võib muuta selle aeglasemaks.

Negatiivsete kaalude korral: Dijkstra ei saa käidelda negatiivseid kaale, mis võivad põhjustada ebatäpseid tulemusi.

Kokkuvõttes on Dijkstra algoritm võimas tööriist lühimate tee leidmiseks paljudes olukordades, kui selle piirangud on teada ja mõistetud.

**Ülesanne 4**

Bellman-Fordi Algoritmi analüüs

**Kirjeldus:**

Bellman-Fordi algoritm leidmiseks lühimad teed graafides, kus servadel võivad olla nii positiivsed kui ka negatiivsed kaalud. See suudab tuvastada ka negatiivseid tsükleid.

**Erinevused Dijkstra Algoritmiga:**

Sobib negatiivsete kaaludega graafidele, mida Dijkstra ei saa käidelda.

Ei nõua prioriteetset järjekorda.

**Negatiivsete Tsüklite Tuvastamine:**

Algoritm suudab avastada negatiivseid tsükleid, mis on praktiliselt oluline, kuna need võivad põhjustada ebasoovitavaid või ohtlikke olukordi.

Praktiline tähtsus:

Sobib hästi olukordadesse, kus graafi kaalud võivad olla nii positiivsed kui ka negatiivsed, ning kus on oluline negatiivsete tsüklite avastamine ja vältimine.

**Ülesanne 5**

**Kirjeldus:**

Graafide värvimise probleem seisneb selles, kuidas värvida graafi tipud nii, et kaks omavahel ühendatud tippu (sõlme) ei oleks sama värvi. Värvide arv vastab ressursside arvule või konfliktide vältimisele mingis kontekstis. Iga värviga määratakse tipp nii, et ükski kaks omavahel ühendatud tippu ei ole samas värvigrupis.

**Tähtsus arvutiteaduses:**

Ressursside jaotamine: Graafide värvimist kasutatakse sageli ressursside tõhusaks jaotamiseks. Näiteks saab igale ressursile (nt protsessor, ajaline intervall, sagedusala) määrata erineva värvi, tagades seeläbi optimaalse kasutamise ja vältides ressursside konflikte.

Konfliktide lahendamine: Graafide värvimine on oluline konfliktide vältimiseks erinevate üksuste (nt ajakavad, ressursid) vahel. Konfliktide vältimine on oluline süsteemides, kus üksused jagavad ressursse ja konfliktid võivad põhjustada ebaefektiivsust või ebastabiilsust.

Telekommunikatsioon ja sagedusjaotus: Graafide värvimist võib kasutada sagedusjaotuse probleemide lahendamiseks telekommunikatsioonis, kus iga sidekanalile tuleb määrata erinev sagedus, et vältida signaalide segunemist.

**Graafide värvimine**

Optimeerimis probleemide lahendamine: Graafide värvimist saab kasutada erinevate optimeerimise probleemide lahendamiseks, kus ressursse tuleb jagada ja konflikte vältida.

Järjestus süsteemide optimeerimine: Näiteks tööde järjestamine tootmisliinil või ülesannete jagamine ajakavas.

Marsruudi planeerimine võrgus: Sagedus kanalite jagamine, marsruutide leidmine võrgus või aja optimeerimine.

Graafide värvimise probleemi lahendamine on oluline praktiliste rakenduste ja süsteemide optimeerimisel, kus ressursside tõhus jaotamine on kriitiline ning konfliktide vältimine on vajalik.