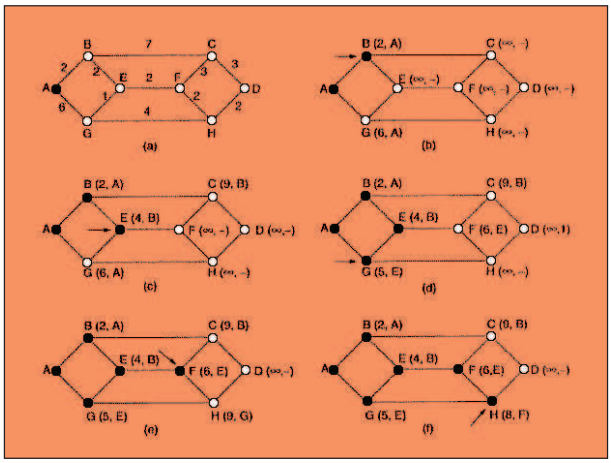
Számítógép hálózatok (15. rész)

Statikus forgalomirányítás, távolságvektor alapú forgalomirányítás, megosztott láthatár.

Tovább ismerkedünk a forgalomirányító algoritmusokkal, először megnézünk pár egyszerűbb, statikus eljárást, majd olyanokat, amelyek az alhálózati topológián kívül más tényezőket is figyelembe vesznek.
Az alhálózatnak törekednie kell arra, hogy a csomagokat a lehető leggyorsabban juttassa el rendeltetési helyére. Erre a legkézenfekvőbb megoldás az, ha a csomag számára a forrás és a cél között fellelhető legrövidebb utat jelöli ki. Ez az elnevezés azonban félrevezető, mivel az alhálózatbeli legrövidebb út nem biztos, hogy egybeesik a „földrajzi” értelemben vett legrövidebb úttal.
A hálózat két pontja közötti távolságot ugyanis nem csak azzal jellemezhetjük, hogy az adatoknak milyen hosszú kábelen (vagy egyéb más csatornán) kell áthaladniuk. A távolságot definiálhatjuk például az ugrások számával, azaz hogy a csomag hány útválasztó érintésével jut el a célállomásáig. Ugyanakkor az is elképzelhető, hogy a csomagokat a legkisebb kommunikációs költségű vonalakon keresztül szeretnénk áramoltatni. Ilyenkor a legrövidebb útnak a legolcsóbb útvonalat tekintjük. Sőt, az általános igény az, hogy ezeket a paramétereket kombinálni lehessen, azaz a legrövidebb út a távolság és a kommunikációs költség mellett az általános terheltség, a sávszélesség és az átlagos sorbanállási idő függvényeként legyen kiszámolva.
Ahhoz, hogy ezt az utat számítógépek segítségével (azaz matematikai úton) meghatározhassuk, a feladatot általánosítanunk kell úgy, hogy azt a nem túlzottan okos gépek is megértsék. Az előző részben az alhálózatot egy gráffal illusztráltuk, amelynek csomópontjai az útválasztók voltak. Két csomópontot akkor kötöttünk össze éllel, amikor a pontok által illusztrált útválasztók között létezett fizikai összeköttetés.
Most ezt a modellt egészítjük ki úgy, hogy minden élhez hozzárendelünk egy értéket, amelyet az adott élhez tartozó súlynak nevezünk. Ez az érték lesz az él által összekötött két csomópont távolsága. (Ezt a gyakorlatban a sávszélesség, a késleltetési idő, az átlagos sorbanállási idő és egyéb paraméterek függvénye szerint számolják ki. A súlyok meghatározásának módja azonban nem érinti a megoldás menetét).
Az eredeti feladatot tehát úgy fogalmazhatjuk át, hogy azt az A és B közötti utat keressük, amelyben a benne szereplő élek súlyainak az összege a lehető legkisebb. Ennek a feladat megoldására sok algoritmus létezik, mi most megnézzük ezek közül az egyik legnépszerűbbet.



1. ábra A Dijkstra algoritmus működése

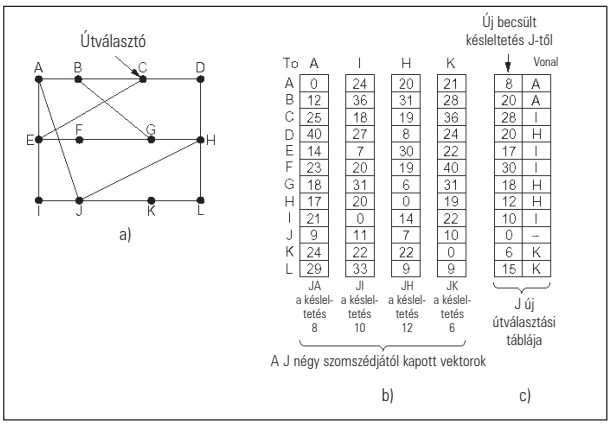
A legrövidebb útvonal (shortest path) algoritmus

Ez az algoritmus ***Dijkstra***, holland származású matematikus nevéhez fűződik (akit algoritmusai mellett az operációs rendszerek világában felhasznált ötletei (például a szemaforok) is ismerté tettek). Az algoritmus egy adott forrástól elkezdve folyamatosan keresi a célhoz vezető legrövidebb utakat. Minden csomóponthoz egy címkét rendel, amelyben eltárolja, hogy a kérdéses csúcs az eddig megtalált legrövidebb úton haladva milyen messze található a forrástól. Először persze egy utat sem ismerünk, így a címkék értéke végtelen. Ahogy telik az idő, az algoritmus egyre jobb utakat talál, így a címkék értéke folyamatosan változhat.
Amikor kiderül, hogy egy csomóponthoz nem lehet a már megismertnél rövidebb úton eljutni, akkor annak a címkéje véglegessé válik, értéke a továbbiakban nem módosul.
Az algoritmus működését az 1. ábrán szemléltetjük. Az „a”-részben látjuk az alhálózatunk felépítését, illetve az élek súlyait, azaz a csomópontok közötti távolságokat. A feladat az A-ból D-be vezető legrövidebb út megtalálása, ennek menetét láthatjuk az ábra többi részén. A már végleges címkével rendelkező csúcsokat besötétített karikával jelezzük.
Kezdetben csak az A csomópontot ismerjük, így azt nyugodtan besötétíthetjük. Ezután megnézzük, hogy az A csúcs mely más csúcsokkal van összekötve, jelen esetben a B és a G csomópontokkal. E két csúcs címkéjét átírjuk az

A-tól vett távolságukkal, azaz B címkéjének új értéke 2, G-nek pedig 6 lesz. Ahhoz, hogy a végeredményül kapott utat rekonstruálhassuk, szükséges, hogy a címkékben a távolságon kívül feltüntessük azt is, hogy az iménti vizsgálatot melyik csúcsból végeztük. Ezt a csúcsot az adott lépés munkacsomópontjának nevezzük (az ábrán egy nyíllal jelöljük). Ez az első lépésnél mindig a forrás, tehát az A csúcs. Ezután az ideiglenes (még be nem satírozott) címkék közül kiválasztjuk azt, amelynek az értéke a legkisebb. Ez most a B csúcs. A B címkéjét állandóra állítjuk, és megkeressük a szomszédait. Ha B egy szomszédjának címkéje kisebb, mint a B címkéje, és a vizsgálandó csúcs súlyának összege, akkor új rövidebb utat találtunk, és a csomópontot újra címezzük. Ha megvizsgáltuk B összes szomszédját, akkor megint kiválasztjuk az ideiglenes címkék közül a legkisebbet, és az eljárást ugyanígy folytatjuk.
Az algoritmus egészen addig nem áll le, amíg az összes utat meg nem találta. Fontos megjegyezni, hogy ez csak olyan gráfok esetén alkalmazható, amelyek nem tartalmaznak negatív súlyú éleket.

Elárasztás (flooding)

Egy másik egyszerű algoritmus az elárasztás. A feladat csupán annyi, hogy a bejövő csomagot az útválasztó minden szomszédjának továbbküldje (kivéve persze annak, akitől a csomag érkezett). Az egyetlen megoldandó probléma a kettőzött csomagok kezelése. Könnyen belátható, hogy beavatkozás nélkül minden csomagból végtelen számú példány keringene az alhálózaton.
Két módszer is ismert, amely segítségével megmenthetjük alhálózatunkat attól, hogy belefulladjon a csomagáradatba. Az egyik elképzelés szerint minden csomag fejlécének kéne tartalmaznia egy úgynevezett ugrásszámlálót, amelynek értéke mindig eggyel csökken, ahányszor a csomag áthalad egy útválasztón. Amikor a számláló eléri a nullát, a csomag megsemmisül. Az egyetlen kérdés az, hogy mekkora legyen a számláló kezdeti értéke. A módszer akkor a leghatékonyabb, ha az ugrásszámláló a forrás és a cél közötti távolság értékéről indul. Ez azonban nem mindig ismert, így a számlálót úgy kell beállítanunk, hogy a csomag a legrosszabb esetben is elérje célját. Ilyen érték lehet például az alhálózat teljes átmérője.
A másik megoldás az, ha minden útválasztó észben tartja, hogy mely csomagokkal végzett már elárasztást. Annak érdekében, hogy a csomagok azonosíthatóak legyenek, az útválasztó a beérkező csomagba – mielőtt tovább küldené a szomszédságának – egy azonosítószámot helyez. Minden útválasztó az összes portjához egy listát rendel, amelyben feljegyzi, hogy az adott forrástól eddig milyen sorszámú csomagokat kapott. Ha egy beérkező csomag még semelyik listában sem szerepel, akkor az útválasztó elvégzi az elárasztást, egyébként pedig eldobja azt. A probléma csak az, hogy a listák mérete folyamatosan növekszenek, így azok előbb vagy utóbb nem fognak elférni a memóriában. Az útválasztók ezért szomszédaikhoz lista helyett inkább egy számlálót rendelnek. A forrás a sorszámokat egyesével, növekvő sorrendben osztja ki, így ha a számláló értékét mindig az utolsó elárasztásra került csomag sorszámára állítva a beérkező csomagokról el tudjuk dönteni, hogy másodpéldányok-e vagy sem (ha a sorszám kisebb mint a számláló aktuális értéke, akkor biztosan másodpéldány).



2. ábra Távolságvektor-alapú forgalomirányítás
a) egyetlen alhálózat b) J útválasztóhoz a szomszédoktól érkező késleltetési vektorok c) J új forgalomirányító táblázata

Az elárasztás módszere kétségtelenül nem egy hatékony eljárás. Valamit javíthatunk a dolgon azzal, ha az elárasztást szelektíven végezzük. Ez alatt azt értjük, hogy a csomagokat csak azokra a vonalakra továbbítjuk, amelyek hozzávetőlegesen a cél felé tartanak. Általában felesleges egy keleti irányban lévő cél felé tartó csomagot a nyugatra mutató portokon is elárasztani.
Nehéz elképzelni, hogy lehet haszna egy olyan forgalomirányító algoritmusnak, amely az alhálózaton átmenő csomagok másolatainak tömkelegét állítja elő, és szerteküldi azokat a hálózat minden szegletébe. Ennek az eljárásnak azonban van pár egyedülálló képessége. Ilyen például az, hogy akár atomtámadás ellen is véd. Ha ugyanis hirtelen útválasztók tucatjai válnak működésképtelenné, a csomagok, igaz más útvonalon, de így is célba érkezhetnek. Egy katonai hálózat esetében ez egy hasznos tulajdonságnak bizonyulhat.
Az elárasztás igazi jelentősége azonban nem a gyakorlati felhasználásában rejlik, hanem abban, hogy összehasonlítási alapul szolgálhat más algoritmusok számára. Ez az eljárás ugyanis a csomagok számára az összes lehetséges utat egyszerre választja ki, amelyek között megtalálható a legrövidebb út is. Ez azt jelenti, hogy nem létezik olyan más algoritmus, amely ennél rövidebb késleltetési időt eredményezne (persze leszámítva az elárasztási műveletek által keltett késleltetést).

Távolságvektor alapú forgalomirányítás

Eddig csak statikus algoritmusokkal foglalkoztunk, amelyek döntéseikben kizárólag az alhálózat topológiája játszott szerepet. A továbbiakban olyan eljárásokat veszünk szemügyre, amelyek figyelembe veszik a hálózat terheltségét is.
Ezek közül az egyik módszer a távolságvektor alapú forgalomirányítás (distance vector routing).
Itt minden útválasztó rendelkezik egy táblázattal, amely két információt rendel minden egyes alhálózati csomóponthoz: a kimenetet és az ismert legrövidebb távolságot. A kimenet azt mondja meg, hogy az adott cél felé melyik porton keresztül juthatunk el a legrövidebben. A távolság értelemszerűen a cél távolsága. Tegyük fel, hogy most a távolság mértékegységének a késleltetési időt adjuk meg. Természetesen a távoság más is lehet, például az ugrások száma.

Az útválasztóknak táblázataikat rendszeres időközönként frissíteniük kell, ugyanis nem biztos, hogy az aktuálisan legrövidebb út egy óra múlva is a legrövidebb lesz (például egy útbaeső útválasztó annyira leterhelté válik, hogy azt megkerülve a csomagot hamarabb célba juttathatjuk). Ehhez azonban szükség van arra, hogy az útválasztók időnként információt cseréljenek egymás között.

A frissítési folyamatot a 2. *ábrán* illusztráltuk. Az ábra „a” részében láthatjuk magát az alhálózatot. Mi most a J router szemszögéből fogunk vizsgálódni.

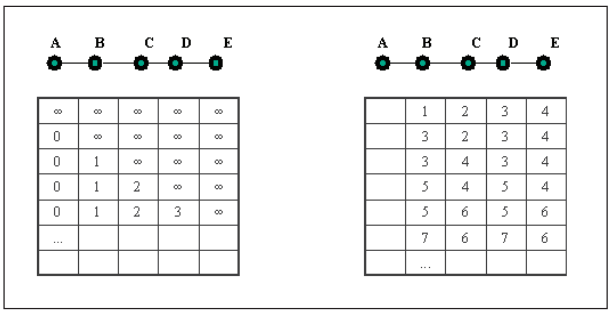
Az első és legfontosabb dolog, hogy minden útválasztó ismerje a szomszédai távolságát. Mivel a távolság most a késleltetés, amely időben folyamatosan változik, ezért az útválasztóknak szabályos időközönként méréseket kell végezniük. A késleltetést mérése úgy zajlik, hogy az útválasztó egy speciális csomagot, úgynevezett ECHO csomagot küld a szomszédjának. Ezzel a csomaggal az útválasztók nem csinálhatnak semmit, csupán beleírják az aktuális időt (úgynevezett időbélyeggel látják el), és visszaküldik a feladónak.

Az útválasztók időnként egy listát küldenek szomszédaiknak, amely tartalmazza az alhálózat összes pontjához tartozó becsült késleltetéseket. A 2*/b ábrán* feltüntettük, hogy a J útválasztó milyen – úgynevezett – késleltetési vektorokat kapott a szomszédaitól. A baloldali oszlop például az A útválasztótól érkező késleltetési vektor. Ha a vektor összes eleméhez hozzáadjuk a J és az A útválasztó közötti késleltetést (amely jelen esetben 8), akkor megkapjuk, hogy az A felé vezető kimeneten indulva mekkora utat kell megtennünk, hogy elérjük az alhálózat egyes pontait. Ezt a számítást el kell végeznünk a többi útválasztótól kapott késleltetési vektorra is. Ezután már csak ki kell választanunk az egyes alhálózati pontokhoz azokat a kimeneteket, amelyen keresztül a leggyorsabban eljuthatunk hozzájuk (2*/c ábra*). Nézzünk meg egy konkrét példát: a J és a G közötti útvonalat. Az A útválasztó azt állítja, hogy ő 18 ms alatt képes a csomagokat eljuttatni a G útválasztóhoz. Tudjuk, hogy az A tőlünk 8 ms-nyi késleltetésre van, tehát az A-n keresztül a G-t 18 + 8 = 26 ms alatt érhetjük el. Nézzük meg, mit mond a többi szomszédunk! Az I-n keresztül 31 + 10 = 41 ms, a H-n keresztül 6 + 12 = 18 ms, a K-n keresztül pedig 31 + 6 = 37 ms időbe telik, míg egy G felé tartó csomag célbaér. Ezek közül a H ajánlata volt a legkedvezőbb, azaz 18 ms, így az új forgalomirányító táblázatban a G ponthoz a H kimenetet rendeljük.

Az alhálózat többi pontjához tartozó új kimeneteket ugyanígy kaphatjuk meg.

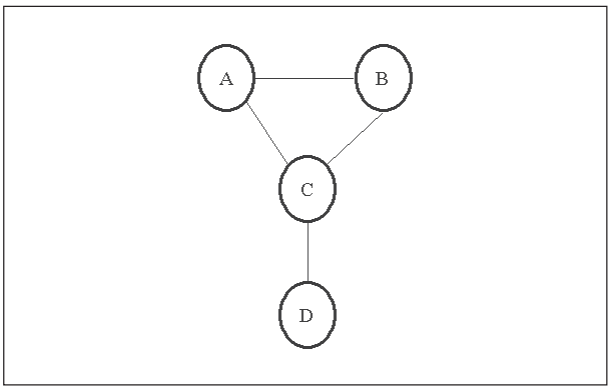
A végtelenig számolás problémája

Sajnos a távolságvektor alapú forgalomirányítás nem egy tökéletes algoritmus, a gyakorlati felhasználás során ugyanis adódig egy rendkívül kellemetlen probléma: míg a jó hírek (például egy korábban elhalálozott útválasztó dicsőséges feltámadása) gyorsan elterjednek az útválasztók között, addig a rosszakra az algoritmus hihetetlenül lassan reagál. Először nézzük meg, mi történik egy jó hír esetén. Az útválasztók közötti távolság legyen most az ugrások száma, továbbá tegyük fel azt is, hogy az útválasztók mindig szabályos időközönként, mindannyian egyszerre egyeztetnek egymással.



3. ábra A végtelenig számlálás problémája, a jó hírek (bal oldal) és a téves hírek (jobb oldal) következményei

A *3. ábrán* két táblázatot láthatunk, ezek közül a baloldali az, ami modellezi, hogy mi történik egy jó hír esetén. Legyen most a jó hír az eddig betegeskedő A útválasztó megjavulása. Kezdetben erről még egyik útválasztó sem tud semmit, ezért mindegyik végtelenre állítja az A-tól való távolságukat. Miután megtörténik az első útválasztók közötti információcsere, a B azt látja, hogy a baloldali szomszédja nulla távolságra van A-tól. Be is jegyzi a forgalomirányító táblázatába, hogy A tőle egy ugrásnyira van (ezt láthatjuk a baloldali táblázat második sorában). A többi útválasztó azonban erről még nem értesült. A következő cserénél a C útválasztó azt kapja, hogy a B egységnyi távolságra van A-tól. Mivel a C pontosan egy ugrásra van B-től, ezért bejegyzi a táblázatába, hogy tőle az A két ugrásnyira van. Láthatjuk, hogy a jó hír ugrás/csere sebességgel terjed, tehát egy olyan alhálózatban, ahol N a leghosszabb út hossza, ott N csere után minden útválasztó értesül a jó hírről.
Nézzük mi történik egy igazán rossz hír esetén, például az A elromlásakor (3/b ábra). Az első cserénél a B semmit sem kap A-tól. A C-től viszont értesül, hogy rajta keresztül létezik elindulva létezik egy 3 távolságú út. A B viszont nem tudhatja, hogy ez az út igazából olyan, hogy saját magán megy keresztül. Ezért a B-nek el kell fogadnia a C állítását, így beírja a táblázatába, hogy a C felé menve három lépésen belül el lehet jutni A-ba.
A második információcsere pillanatában a C azt kapja a szomszédaitól, hogy mindketten 3 távolságra vannak A-tól. Ekkor beírja a táblázatába, hogy az A-tól 4 ugrásra van, a két kimenet közül pedig kiválaszt egyet véletlenül (harmadik sor). A következő cserénél ugyanez a folyamat játszódik le. A probléma nyilvánvalóan az, hogy az útválasztók az A-tól való távolságukat minden cserénél csak eggyel növelik meg. Az algoritmus működése helyes, hiszen a cserék hatására a végeredmény konvergál a végtelenhez, csak ezt kivárni elég hosszú időbe telik. Ezt nevezzük a végtelenig számolás problémájának.
Hogy a gyakorlatban ez pontosan meddig is tart, az attól függ, hogy milyen számmal ábrázoljuk a végtelent.
Ha a távolság az ugrások száma, akkor annyival szerencsésebb a helyzet, hogy a végtelent megválaszthatjuk a leghosszabb út hosszától eggyel nagyobb értéknek. Ez az ugrások számára nézve egy biztos felső korlát. Ha a távolság a késleltetés, akkor elméletileg nincs ilyen felső korlát. Annyit tehetünk, hogy önkényesen kiválasztunk egy értéket, amelynél hosszabb késleltetés esetén az adott vonalat működésképtelennek tekintjük.



4. ábra Ilyen topológia esetén a megosztott láthatár módszere könnyen kudarcot vallhat

Megosztott láthatár

A ***megosztott láthatár (split horizon)*** egy népszerű ötlet a fenti probléma feloldására, ám ez is, mint a többi megoldási javaslat, bizonyos helyzetekben kudarcot vall. Sajnos idáig még nem született tökéletes megoldás, mindenesetre a megosztott láthatárt a gyakorlatban úgy-ahogy megállja a helyét, mivel csak néha hibázik.
Az ötlet az, hogy az útválasztó időnként hazudik a szomszédainak: egy X útválasztótól való távolságot végtelennek mond azon a vonalon, amelyre az X felé menő csomagokat irányítja. Csak az első hallásra bonyolult a dolog, a következő példából minden világos lesz. Nézzük megint a 3. ábrán lévő topológiát. A C útválasztó a D-nek elmondja az igazat az A-tól való távolságáról, viszont B-nek nem. Neki azt mondja, hogy végtelen a távolsága. Ugyanígy D is végtelent hazudik C-nek, E-vel viszont megosztja az A-tól való valódi távolságát.
Ez a megoldás valóban működik, ugyanis ha A tönkremegy, akkor B-nek C végtelen távolságot mond. Most már a rossz hír is ugrás/csere sebességgel terjed az alhálózaton.
Könnyű azonban olyan helyzetet kitalálni, amikor ismét szembesülnünk kell a végtelenig számlálás problémájával. Vegyük például a *4. ábrán* látható alhálózati topológiát, és tegyük fel, hogy a D útválasztó hirtelen meghibásodik. Az első csere pillanatában a C értesül erről, ugyanis D-től semmit sem kap, A-tól és B-től a megosztott láthatár következtében végtelen érték érkezik. Amint ez kiderül, C értesíti A-t és B-t, hogy a D elérhetetlen. Az A azonban tudja, hogy a B-nek van egy 2 hosszú útja D-be. A következő cserénél B meg arról értesül, hogy az A-nak létezik egy 3 hosszú útja D-be. Megint ugyanaz a helyzet, mint a 3. ábrán látott példa esetén.
Mindenesetre 1979-ig az internet elődje, az ***ARPANET*** ezt az algoritmust használta a forgalomirányításhoz. Lecserélését két ok indokolta: először is az algoritmus gyakran még így is túl lassan reagált a változásokra. Másodszor a távolság kiszámításához nem vették figyelembe a vonalak sávszélességét (habár ez a probléma egyszerűen megoldható lett volna). Így a távolságvektor alapú algoritmust felváltotta a kapcsolatállapot alapú forgalomirányító algoritmus, amellyel sorozatunk következő részében foglalkozunk.

Garzó András
garzo@interware.hu