Online algoritmusok

Online problémáról beszélünk azokban az esetekben, ahol nem ismert az egész input, hanem az algoritmus az inputot részenként kapja meg, és a döntéseit a megkapott részletek alapján a további részekre vonatkozó információk nélkül kell meghoznia. Online algoritmusokat sok probléma esetén alkalmaznak az operációkutatás, az elméleti számítástudomány és a közgazdaságtan különböző területein.

Az algoritmusok hatékonyságát általában a versenyképességi analízis alapján mérik (a másik használt elemzési módszer az átlagos eset analízis).

Optimalizálási feladatok esetén minden I inputra $OPT(I)$ jelöli az optimális megoldás célfüggvényértékét és egy A algoritmusra $A(I)$ jelöli az algoritmus által az A inputon felvett célfüggvényértéket.

Minimalizálási feladatok esetén egy online A algoritmusra azt mondjuk C-versenyképes, ha minden I inputra teljesül $A(I) \leq C \cdot OPT(I)$. Gyakran a definícióban megengednek egy inputtól független D additiv konstanst, azaz az $A(I) \leq C \cdot OPT(I) + D$ egyenlőtlenségnek kell teljesülnie minden inputra. Bizonyos esetekben a C függhet az input méretétől. Ekkor az $A(I) \leq C(n) \cdot OPT(I)$ egyenlőtlenségnek kell teljesülnie minden n méretű inputra. Egy algoritmus versenyképességi hányadosának a legkisebb C számot nevezzük, amire C-versenyképes.

Síbérlési feladat

A síbérlési feladat esetén, adott egy sí, amit vagy 1 egységért bérelhetünk vagy megvásárolhatunk B egységért, a feladat eldönteni mikor vásároljuk meg.

B algoritmus: A B-edik napig béreljük a sít, a B-edik napon megvásároljuk.

Tétel A B algoritmus $(2B-1)/B$-versenyképes.

Bizonyítás A feladat inputja I, azon napok száma, ahány napig síelünk.

- Ha $I < B$, akkor mind az online algoritmusnak mind pedig az optimális offline algoritmusnak a költsége I, így B(I)/OPT(I)=1.
- Ha $I \geq B$, akkor $OPT(I) = B$, továbbá $B(I) = B - 1 + B$, így $B(I)/OPT(I) = (2B-1)/B$

Tétel Nincs olyan online algoritmus, amelynek kisebb a versenyképességi hányadosa, mint $(2B-1)/B$.

Bizonyítás Legyen A az az időpont, amikor az algoritmus megvásárolja a sít. A feladat inputja legyen A napnyi síelés, az algoritmus költsége $A - 1 + B$.

- Ha $A < B$, akkor az optimális költség A. Másrészt $(A-1+B)/A = 1 + (B-1)/A \geq (2B-1)/B$.
- Ha $A \geq B$, akkor az optimális költség B. Másrészt $(A-1+B)/B \geq (2B-1)/B$.

Lapozás

A lapozási problémában a számítógépek gyorsmemóriájának a kezelését modellezik. Adott lapok egy univerzuma, és innét származó lapok egy sorozata az input.

Az algoritmusnak egy k lap kapacitású gyorsmemóriát kell kezelnie. Ha az aktuálisan igényelt lap nincs a memóriában, akkor a lapot az algoritmusnak be kell tennie és ehhez, amennyiben már nincs hely, valamely lapot ki kell raknia.

Ezt az eseményt, amikor egy igényelt lap nincs a memóriában hibának hívjuk, és a cél a hibák számának minimalizálása.

Alsó korlát

Tétel Nincs olyan online algoritmus a lapozási problémára, amely versenyképességi hányadosa kisebb, mint k.

Vegyünk k+1 lapot és legyen A egy tetszőleges online algoritmus. Legyen L_n az az n elemből álló input, amelyben az új elem mindig az a lap, amely lap éppen hiányzik A memóriájából. Legyen továbbá n osztható k-val. Ekkor A minden lapnál hibázik, így $A(L_n) = n$.

Legyen LFD (longest forward distance) az az offline algoritmus, amely ha egy lapot ki kell rakni a memóriából mindig azt a lapot választja a lapok közül, amelyre a következő igény a legkésőbb érkezik. Ez az optimális.

Ha LFD hibázik, akkor a következő k-1 kérés során nem hibázhat újra, így $LFD(L_n) \leq n/k$.

Tehát $A(L_n)/OPT(L_n) \geq n/(n/k)$, amivel a tétel állítását igazoltuk.

Bélyegző algoritmus

Az algoritmus bélyegeket használ a memóriában. Kezdetben egyetlen lap sincs megjelölve. Egy kérés érkezésekor az algoritmus a következő lépéseket hajtja végre.

1. Ha a kért lap a memóriában van, akkor amennyiben ez a lap még jelöletlen, akkor megjelöljük.
2. Ha a kért lap nincs a memóriában, és nincs már jelöletlen lap a memóriá- ban, akkor az összes jelölést töröljük. Ezt követően veszünk egy jelöletlen lapot a memóriából (az előző lépés miatt van ilyen) a kért lapot ennek a lapnak a helyére berakjuk, majd megjelöljük.

Az algoritmus viselkedése nagymértékben függ attól, hogy milyen szabály alapján választjuk ki a törlendő lapot, de a következő tétel mutatja, hogy a versenyképességi hányadosa nem függ ettől.

FIFO: A legrégebben bekerült lapot dobja ki (nem bélyegző).

LRU: Mindig a legrégebben használt lapot dobja ki (bélyegző).

Tétel A B algoritmus versenyképességi hányadosa k.

Bizonyítás

Bontsuk fel ezt az inputot fázisokra. Az első fázis kezdődjön az első elemnél. Ezt követően minden egyes fázis a megelőző fázis utolsó eleme után jövő elemnél kezdődik, és a leghosszabb olyan sorozatát tartalmazza a kéréseknek, amelyek legfeljebb k különböző lapot igényelnek.

(Tehát a következő fázis akkor kezdődik, amikor a k+1-edik különböző lap megjelenik.) Érdemes megjegyezni, hogy a fázis a bélyegek kiosztásával is definiálható, akkor kezdődik új fázis, amikor a B algoritmus törli a bélyegeket a memóriában.

Az algoritmus definíciójából következik, hogy B legfeljebb k hibát vét egy fázis alatt, (ha egy lapon hibázik, azon többet már nem fog a fázisban, hiszen megjelölte).

Egy tetszőleges fázis második kérésénél az offline memóriában benne van a fázis első eleme. Ezt követően a következő fázis első eleméig k további elem jelenik meg, így az offine algoritmusnak legalább egy hibát kell vétenie, az adott fázis második elemétől, a következő fázis első eleméig tartó kéréssorozaton.

Következésképpen minden fázishoz (kivéve esetleg az utolsót) hozzárendeltük az offline algoritmus egy-egy hibáját. Jelölje r a fázisok számát. Ekkor $B(I) \leq rk + k - 1$ és $OPT(I) \geq r$, következésképp $B(I) \leq k \cdot OPT(I) + k - 1$, amivel a tétel állítását igazoltuk.

Lista karbantartási probléma

Adott egy lista, amelyen a következő műveleteket tudjuk végrehajtani:

- Keres(x)
- Torol(x)
- Beszúr(x)

Mi csak a statikus modellt vizsgáljuk, ahol adottak a lista elemei és csak Keres(X) műveleteket hajtunk végre. Amennyiben a lista i-edik elemét keressük, akkor a költség i.

A megengedett karbantartási műveletek:

- a Keres(x) művelet során az x elemet ingyen tetszőleges helyre előre mozgathatjuk
- fizetett cserék: bármely két egymás melletti elemet felcserélhetünk 1 költséggel

Példa Legyen a lista x_1, x_2, x_3 a kéréssorozat x_3, x_2, x_3, x_2. Esetszétválasztással igazolható, hogy amennyiben nem használ egy algoritmus fizetett cseréket a költsége legalább 9. (Az első kérés költsége 3, ezt követoen megvizsgálva, hogy hova rakja az algoritmus az x_3 elemet igazolható az állítás.)

Ezzel szemben ha két fizetett cserével az utolsó helyre visszük x_1-et, akkor a cserék költsége 2 és utána nem mozdítva több elemet a kérések költsége 6. Következésképpen fizetett cserével jobb megoldást kaphatunk.

MTF algoritmus

Az algoritmus inputja elemek egy kezdeti $x_1, \ldots, x_n$ listája, és kérések egy $\sigma = \sigma_1, \ldots, \sigma_m$ sorozata, ahol $\sigma_i \in \{x_1, \ldots, x_n\}$. A i-edik kérés hatására a KERES(σ_i) műveletet kell végrehajtani.

MTF algoritmus Az algoritmus a kért elemet a lista elejére mozdítja.

Tétel Az MTF algoritmus 2-versenyképes.

Bizonyítás Vegyünk egy optimális megoldást, jelölje OPT. Legyen a $\Phi(i)$ potenciálfüggvény az inverziók száma az i-edik kérés után az MTF által karbantartott listában az optimálishoz képest, azaz azon x, y párok száma, amelykre x megelőzi y-t OPT listájában de nem előzi meg MTF listájában.

Vizsgáljuk meg a k-adik kérés, $Keres(\sigma_k)$ művelete során fellépő költségeket és potenciálváltozást. Legyen p azon elemek száma, amelyek mind MTF mind OPT listájában megelőzik σ_k-t, q azon elemek száma, amelyek csak MTF listájában előzik meg σ_k-t.

MTF költsége $p+q+1$, az optimális költség legalább $p+1$. A Φ függvény értéke $-q+p$-vel változik. Tehát
$MTF(\sigma_i) + \Phi(k) - \Phi(k-1) = p+q+1-q+p = 2p+1 \leq 2OPT(\sigma_i)$

Következésképpen
$MTF(\sigma) + \Phi_n - \Phi_0 \leq 2OPT(\sigma)$

FC algoritmus

FC algoritmus Az elemeket mindig a gyakoriságuk sorrendjében tartja sorban. (Ezt megteheti fizetett cserék nélkül, mivel mindig csak a keresett elem gyakorisága növekszik, így esetleg azt kell előre mozgatni.)

Lemma FC nem konstans versenyképes.

Bizonyítás Legyen a kezdeti lista $x_1, \ldots, x_n$, a kéréssorozat pedig legyen n-szer x_1, n-1-szer x_2, és így tovább $n+1-i$-szer x_i. Ekkor FC sosem változtat a listán és a kiszolgálás teljes költsége

$$\sum_{i=1}^{n}(n+1-i)i = (n+1)\sum_{i=1}^{n} i - \sum_{i=1}^{n} i^2 = (n+1)^2 n/2 - n(n+1)(2n+1)/6 = \Theta(n^3).$$

Ezzel szemben MTF költsége $\sum_{i=1}^{n} i + \sum_{i=1}^{n-1} i = \Theta(n^2)$. Következésképpen $FC(I)/OPT(I) \to \infty$, ha n tart végtelenbe.

TRANSPOSE algoritmus

TRANSPOSE algoritmus: A keresett elemet, ha nem az első a listában egy hellyel előrébb mozgatja.

Lemma TRANSPOSE nem konstans versenyképes.

Bizonyítás: Legyen a lista $x_1, \ldots, x_n$ és vegyük a következő kéréssorozatot $(x_n, x_{n-1})^M$. Ekkor Transponse mindig a kért utolsó elemet felcseréli az előtte álló elemmel, így a teljes költsége $2M \cdot n$. Másrészt az MTF algoritmusnak,

amely mindig legelőre hozza az aktuális elemet a költsége az első két kéréstől eltekintve mindig 2, így összesen $2n+2(M-1)2$. Következésképpen $Transpose(I)/OPT(I) \geq 2M \cdot n/(2n+4(M-1) \rightarrow n/2$ ha M tart végtelenbe.

Alsó korlát

Tétel Ha egy online algoritmus c-versenyképes a lista karbantartási feladatra, akkor $c \geq 2$.

Bizonyítás Vegyünk egy tetszőleges online algoritmust. Definiáljunk egy m hosszú inputot, amely mindig az algoritmus listájának utolsó elemét kéri. Ekkor az algoritmus költsége $m \cdot n$.

Az optimális költség becsléséhez vegyünk két statikus algoritmust. STAT1 nem mozdít egyetlen elemet sem, STAT2 pedig először megfordítja az elemek sorrendét és utána nem mozdít egyetlen elemet sem.

Ekkor bármilyen kérés érkezik annak a teljes költsége a két algoritmusra nézve pontosan $n+1$. Továbbá STAT2 esetén a kezdeti sorrend elérésének költsége $(n-1)n/2$. Következésképpen STAT1 és STAT2 együttes költsége az inputra $(n-1)n/2+m(n+1)$, így az optimális költség legfeljebb $((n-1)n/2+m(n+1))/2$.

Tehát azt kaptuk, hogy erre az inputra $ALG(I)/OPT(I)$ hányadosra kapott alsó korlát tart az $2n/(n+1)$ értékhez, ahogy m tart a végtelenbe. Másrészt ezen hányados határértéke 2, ha n tart a végtelenbe, amivel a tételt igazoltuk.

k-szerver probléma

Egy (M,d) párost, ahol M a metrikus tér pontjait tartalmazza, d pedig az $M \times M$ halmazon értelmezett távolságfüggvény metrikus térnek nevezünk, ha a távolságfüggvényre teljesülnek az alábbi tulajdonságok

- $d(x,y) \geq 0$ minden $x,y \in M$ esetén,
- $d(x,y) = d(y,x)$ minden $x,y \in M$ esetén,
- $d(x,y)+d(y,z) \geq d(x,z)$ minden $x,y,z \in M$ esetén,
- $d(x,y) = 0$ akkor és csak akkor teljesül, ha $x = y$.

A k-szerver problémában adott egy metrikus tér, és van k darab szerverünk, amelyek a térben mozoghatnak. A probléma során a tér pontjaiból álló kérések egy listáját kell kiszolgálni azáltal, hogy a megfelelő kérések helyére odaküldünk egy-egy szervert.

A probléma on-line, ami azt jelenti, hogy a kéréseket egyenként kapjuk meg, és az egyes kéréseket a további kérések ismerete nélkül azok érkezése előtt kell kiszolgálnunk. A cél a szerverek által megtett össztávolság minimalizálása.

Lapozás mint speciális eset

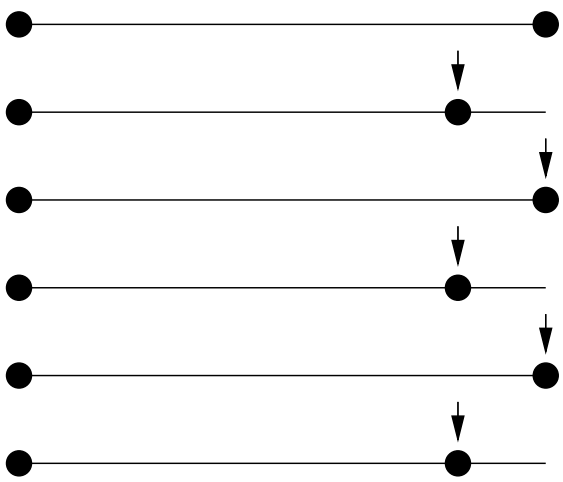
A lapozási problémában a számítógépek gyorsmemóriájának a kezelését modellezi. Adott lapok egy univerzuma, amely lapok egy sorozata az input. Az algoritmusnak egy k lap kapacitású gyorsmemóriát kell kezelnie. Ha az aktuálisan igényelt lap nincs a memóriában, akkor a lapot az algoritmusnak be kell tennie és ehhez, amennyiben már nincs hely, valamely lapot ki kell raknia. Ezt az esenényt, amikor egy igényelt lap nincs a memóriában hibának hívjuk, és a cél a hibák számának minimalizálása.

Ha az univerzum a metrikus tér és a memória celláit a szervereknek feleltetjük meg (egy szerver azon a ponton van, amely lapot a cella tartalmazza), akkor egy k-szerver feladatot kapunk. A metrikus térben bármely két pont távolsága 1. Az ilyen tereket unifrom térnek nevezzük.

következmény: Uniform tereken vannak k-versenyképes algoritmusok, és nincs olyan algoritmus, amely versenyképesse hányadosa kisebb, mint k.

Mohó algoritmus

Természetes gondolat a kéréseket mindig a legközelebbi szerverrel kiszolgálni.



1. ábra.

Lemma A mohó algoritmus nem konstans versenyképes.

Egyensúly algoritmus

Az eljárás során a szerverek mindig különböző pontokban helyezkednek el. Az algoritmus a futása során minden szerverre számon tartja, hogy az aktuális időpontig összesen mekkora távolságot tett meg. Jelölje rendre $s_1, \ldots, s_k$ a szervereket és a pontokat is, ahol a szerverek elhelyezkednek. Továbbá jelölje $D_1, \ldots, D_k$ rendre a szerverek által az adott időpontig megtett összutat.

Ekkor, amennyiben egy P pontban megjelenik egy kérés akkor a ES algoritmus azt az i szervert választja a kérés kiszolgálására, ahol a $D_i + d(s_i, P)$ érték minimális. Tehát az algoritmus számon tartja egy k méretű S tömbben szerverek konfigurációját, egy D tömbben a szerverekhez rendelt távolságokat. Az algoritmus egy $I = P_1, \ldots, P_n$ pontsorozatra (amely pontokat egy D tömb által kapunk meg) a következőképpen fut le

```
ES(I) algoritmus
 for j=1 to n
    i= argmin D[i]+d(s[i],P[j])
    szolgáljuk ki a kérést az i-edik szerverrel
    D[i]:=D[i]+d(s[i],P[j])
    s[i]:=P[j]
```

Példa

Tekintsük a kétdimenziós Euklédeszi teret, mint metrikus teret. A pontok (x,y) valós számpárokból állnak, két pontnak (a,b)-nek és (c,d)-nek a távolsága $\sqrt{(a-c)^2+(b-d)^2}$. Legyen két szerverünk kezdetben a $(0,0)$ és $(1,1)$ pontokban.

Kezdetben $D_1 = D_2 = 0$, $s_1 = (0,0)$, $s_2 = (1,1)$. Az első kérés legyen az $(1,4)$ pontban. Ekkor $D_1 + d((0,0)(1,4)) = \sqrt{17} > D_2 + d((1,1)(1,4)) = 3$, így a második szervert használjuk és a kérés kiszolgálása után $D_1 = 0, D_2 = 3$, $s_1 = (0,0)$, $s_2 = (1,4)$ teljesül.

Legyen a második kérés $(2,4)$, ekkor $D_1 + d((0,0)(2,4)) = \sqrt{20} > D_2 + d((1,4)(2,4) = 3+1 = 4$, így ismét a második szervert használjuk, és a kérés kiszolgálása után $D_1 = 0, D_2 = 4$, $s_1 = (0,0)$, $s_2 = (2,4)$ teljesül.

A harmadik kérés legyen ismét az $(1,4)$ pontban, ekkor $D_1 + d((0,0)(1,4)) = \sqrt{17} < D_2 + d((2,4)(1,4) = 4+1 = 5$, így az első szervert használjuk és a kérés kiszolgálása után $D_1 = \sqrt{17}, D_2 = 4$, $s_1 = (1,4)$, $s_2 = (2,4)$ teljesül.

tétel Amennyiben a metrikus tér $k+1$ pontot tartalmaz, akkor az ES algoritmus enyhén k-versenyképes.

Munkafüggvény algoritmus

Azt a metrikus tér pontjaiból álló multihalmaz, amely megadja mely pontokban helyezkednek el a szerverek (azért kell multihalmazokat használnunk, mert egy pontban több szerver is lehet) a szerverek konfigurációjának nevezzük.

Legyen A_0 az on-line szerverek kezdeti konfigurációja. Ekkor a t-edik kérés utáni X multihalmazra vonatkozó munkafüggvény $w_t(X)$ a minimális költség, amellyel kiszolgálható az első t kérés az A_0 konfigurációból kiindulva úgy, hogy a szerverek az X konfigurációba kerüljenek a kiszolgálás végén. A munkafüggvény algoritmus a munkafüggvényt használja.

Legyen A_{t-1} a szervereknek a konfigurációja közvetlenül a t-edik kérés érkezése előtt. Ekkor a munkafüggvény algoritmus azzal az s szerverrel szolgálja ki az R_t pontban megjelent kérést, amelynek a P helyére a $w_{t-1}(A_{t-1} \setminus P \cup R_t) + d(P,R_t)$ érték a minimális.

Példa

Tekintsük azt a metrikus teret, amelyben három pont van A, B és C, a távolságok $d(A,B) = 1$, $d(B,C) = 2$, $d(A,C) = 3$. A kezdeti szerver konfiguráció legyen A,B.

Ekkor a kezdeti munkafüggvények $w_0(\{A,A\}) = 1$, $w_0(\{A,B\}) = 0$, $w_0(\{A,C\}) = 2$, $w_0(\{B,B\}) = 1$, $w_0(\{B,C\}) = 3$, $w_0(\{C,C\}) = 4$.

Legyen az első kérés a C pontban. Ekkor $w_0(\{A,B\} \setminus \{A\} \cup \{C\}) + d(A,C) = 3+3 = 5$ és $w_0(\{A,B\} \setminus \{B\} \cup \{C\}) + d(B,C) = 2+2 = 4$, így a MUNKAFÜGGVÉNY algoritmus a B pontban levő szerver küldi a kérés kiszolgálására.

Tétel A MUNKAFÜGGVÉNY algoritmus enyhén $2k-1$-versenyképes.

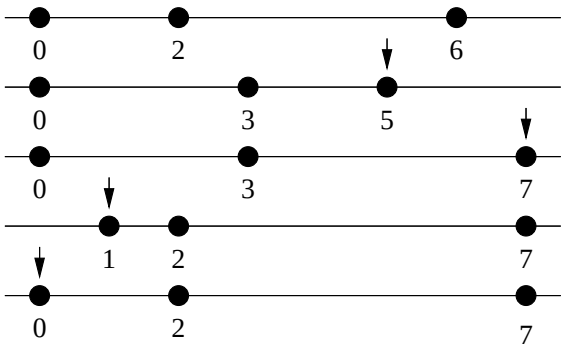
Dupla lefedő algoritmus

Egy másik vizsgált speciális tér az egyenes. Az egyenes pontjait a valós számoknak feleltetjük meg, és két pontnak a-nak és b-nek a távolsága $|a-b|$. Erre az esetre, ahol a metrikus tér egy egyenes, Chrobak és Larmore kifejlesztett egy k-versenyképes algoritmust, amelyet dupla lefedő algoritmusnak nevezünk.

Az algoritmus a P kérést a P-hez legközelebb eső s szerverrel szolgálja ki, és amennyiben vannak szerverek P-nek az s-el átellenes oldalán is, akkor azon szerverek közül a P-hez legközelebbit $d(s,P)$ egységnyit mozdítja P felé. A továbbiakban a dupla lefedő algoritmust DL-el jelöljük.

Tétel A DL algoritmus k-versenyképes ha a metrikus tér egy egyenes.

```
DL(I,S) algoritmus
  for j=1 to n
    i:=argmin {d(P[j],s[l]) : l=1,..,n}
    if s[i]=min{s[l] : l=1,..,n} or  s[i]=min{s[l] : l=1,..,n}
       then a kérést az i szerver szolgálja ki
            s[i]:=P[j]
       else if s[i]<= P[j]
               then  m:={argmin}{d(s[l],P[j]) s[l] > P[j]}
                     a kérést az i szerver szolgálja ki
                     s[m]:=s[m]- d(s[i],P[j])
                     s[i]:=P[j]
               else r:={argmin}{d(s[l],P[j]) s[l] < P[j]}
                    a kérést az i szerver szolgálja ki
                    s[r]:=s[r]+ d(s[i],P[j])
                    s[i]:=P[j]
```



2. ábra.

Példa

Bizonyítás I.

Vegyünk egy tetszőleges sorozatát a kéréseknek, jelölje ezt az inputot I. Az eljárás elemzése során feltételezzük, hogy párhuzamosan fut a DL algoritmus és egy optimális off-line algoritmus.

Szintén feltesszük, hogy minden kérést elsőként az off-line algoritmus szolgál ki, utána pedig az on-line algoritmus. Az on-line algoritmus szervereit és egyben a szerverek pozícióit, (amelyek valós számok az egyenesen) $s_1, \ldots, s_k$ jelöli, az optimális off-line algoritmus szervereit és egyben a szerverek pozícióit $x_1, \ldots, x_k$ jelöli.

Mivel a szerverek rendszeres átjelölésével ez elérhető feltételezzük, hogy $s_1 \leq s_2 \leq \cdots \leq s_k$ és $x_1 \leq x_2 \leq \cdots \leq x_k$ mindig teljesül.

A tétel állítását a potenciálfüggvény technikájával igazoljuk. A potenciálfüggvény a szerverek aktuális pozíciójához rendel egy értéket, az on-line és az off-line költségeket a potenciálfüggvény változásainak alapján hasonlítjuk össze. Legyen a potenciálfüggvény

$$\Phi = k \sum_{i=1}^{k} |x_i - s_i| + \sum_{i<j} (s_j - s_i).$$

Az alábbiakban igazoljuk, hogy a potenciálfüggvényre teljesülnek az alábbi állítások.

- Amíg OPT szolgálja ki a kérést, addig a potenciálfüggvény növekedése legfeljebb k-szor az OPT szerverei által megtett távolság.
- Amíg DL szolgálja ki a kérést, addig Φ legalább annyival csökken, mint amennyi a kérés kiszolgálásának költsége.

Amennyiben a fenti tulajdonságok teljesülnek, akkor a tétel állítása következik, hiszen ebben az esetben adódik, hogy $\Phi_v - \Phi_0 \leq k \cdot \mathrm{OPT}(I) - \mathrm{DL}(I)$, ahol Φ_v és Φ_0 a potenciálfüggvény kezdeti és végső értékei.

Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $\mathrm{DL}(I) \leq k\mathrm{OPT}(I) + \Phi_0$, azaz azt kapjuk, hogy a DL algoritmus enyhén k-versenyképes.

Bizonyítás II.

Elsőként vizsgáljuk azt az esetet, amikor OPT valamely szervere d távolságot mozog. Ekkor a potenciálfüggvényben szereplő első rész legfeljebb kd-vel növekszik a második rész nem változik, tehát az első tulajdonsága a potenciálfüggvénynek valóban fennáll.

Most vizsgáljuk DL szervereit. Legyen P a kérés melyet ki kell szolgálni. Mivel elsőként OPT szolgálta ki a kérést, ezért $x_j = P$ valamely szerverre. Most különböztessünk meg két esetet DL szervereinek elhelyezkedésétől függően.

Elsőként tegyük fel, hogy minden szerver P-nek ugyanarra az oldalára esik. Feltehetjük, hogy minden szerver pozíciója nagyobb P-nél, a másik eset teljesen hasonló. Ekkor s_1 a legközelebbi szerver P-hez és DL s_1-et küldi P-be, más szervert nem mozgat.

Tehát DL költsége $d(s_1,P)$. A potenciálfüggvényben szereplő első összegben csak az $|x_1 - s_1|$ tag változik és ez csökken $d(s_1,P)$ egységgel, tehát az első rész csökken $kd(s_1,P)$ egységgel. A második tag növekszik $(k-1)d(s_1,P)$ egységgel, így Φ értéke csökken $d(s_1,P)$ egységgel.

Most tekintsük a másik esetet. Ekkor P-nek mindkét oldalára esik szerver, legyenek ezek a szerverek s_i és s_{i+1}. Tegyük fel, hogy s_i esik közelebb P-hez, a másik eset teljesen hasonló.

Tehát DL költsége $2d(s_i,P)$. Vizsgáljuk a potenciálfüggvény első részének változásait. Az i-edik és az $i+1$-edik tag változik. Az egyik tag növekszik, a másik csökken $d(s_i,P)$ egységgel, tehát az első rész összességében nem változik.

A Φ függvény második részének változása

$$d(s_i,P)\big(-(k-i)+(i-1)-(i)+(k-(i+1))\big) = -2d(s_i,P).$$

Tehát ebben az esetben is fennáll a potenciálfüggvény második tulajdonsága.

Fa metrikus tér

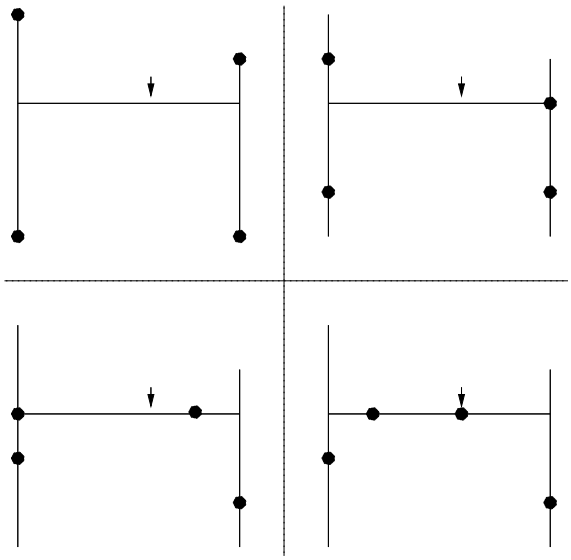
Az egyenes általánosításaként kaphatjuk a fa metrikus teret. vegyünk egy tetszőleges súlyozott fát. A metrikus tér pontjai a fa csúcsai továbbá minden élhez és minden $0 < \lambda < 1$ számhoz definiáljuk az élet λ és $1-\lambda$ arányban felosztó pontokat.

Mivel a fa körmentes ezért bármely két pont között egyetlen út vezet. A két pont közötti távolság a közöttük vezető út hossza. Az élek belső pontjai esetén az él hosszának a felosztásnak megfelelő részét vesszük figyelmebe.

Azt mondjuk, hogy egy szerver lát egy pontot, ha a közte és a pont között levő úton nincs másik szerver.

Lefedő algoritmus Az algoritmus egy kérést úgy szolgál ki, hogy minden szervert, ami látja a kérés helyét egyenletes sebességgel mozgat a szerver felé. (A kérést látó szerverek halmaza változik a szerverek mozgatása során.)

tétel A lefedő algoritmus enyhén k-versenyképes, ha a metrikus tér egy fa.



3. ábra.

tétel Nincs olyan legalább $k+1$ pontból álló metrikus tér, ahol megadható olyan on-line algoritmus, amelynek kisebb a versenyképességi hányadosa, mint k.

bizonyítás Tekintsünk egy tetszőleges legalább $k+1$ pontból álló teret, és egy tetszőleges on-line algoritmust. Jelölje az algoritmust ONL, a pontokat ahol kezdetben ONL szerverei állnak $P_1, P_2, \ldots, P_k$, a térnek egy további pontját jelölje P_{k+1}.

Vegyük kéréseknek egy hosszú $I = Q_1, \ldots, Q_n$ sorozatát, amelyet úgy kapunk, hogy a következő kérés mindig a $P_1, P_2, \ldots, P_{k+1}$ pontok közül abban a pontban keletkezik, ahol nincs szerver.

Vizsgáljuk elsőként a $\mathrm{ONL}(I)$ költséget. Mivel a Q_j pont kiszolgálása után a Q_{j+1} pont lesz szabad, ezért a Q_j pontot mindig a Q_{j+1} pontban álló szerver szolgálja ki, így a kiszolgálás költsége $d(Q_j, Q_{j+1})$.

$$\mathrm{ONL}(I) = \sum_{j=1}^{n} d(Q_j, Q_{j+1}) \ ,$$

ahol Q_{n+1} azt a pontot jelöli, ahol az $n+1$-edik kérés lenne, azaz azt a pontot, amelyről kiszolgáltuk az n-edik kérést.

Most vizsgáljuk a $\mathrm{OPT}(I)$ költséget. Az optimális off-line algoritmus meghatározása helyett definiálunk k darab off-line algoritmust, és ezek költségeinek az átlagát használjuk, mivel mindegyik algoritmus költsége legalább akkora mint a minimális költség, ezért a költségek átlaga is felső korlátja lesz az optimális költségnek.

Definiáljunk k darab off-line algoritmust, jelölje őket $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$. Tegyük fel, hogy a kiindulási állapotban az OFF_j algoritmus szerverei a $P_1, P_2, \ldots, P_{k+1}$ pontok közül a P_j pontot szabadon hagyják, és a halmaz további pontjainak mindegyikén egy szerver helyezkedik el. Ez a kiindulási állapot elérhető egy C_j konstans extra költség felhasználásával.

A kérések kiszolgálása a következőképpen történik. Ha a Q_i ponton van az OFF_j algoritmusnak szervere, akkor nem mozgat egyetlen szervert sem, ha nincs, akkor a Q_{i-1} ponton levő szervert használja. Az algoritmusok jól definiáltak, hiszen ha nincs szerver a Q_i ponton, akkor a $P_1, P_2, \ldots, P_{k+1}$ pontok mindegyikén, így Q_{i-1}-en is van szerver. Továbbá a $Q_1 = P_{k+1}$ ponton az OFF_j algoritmusok mindegyikének áll szervere a kezdeti konfigurációban.

Vegyük észre, hogy az $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$ algoritmusok szerverei rendre különböző konfigurációkban vannak. Kezdetben ez az állítás a definíció alapján igaz. Utána a tulajdonság azért marad fenn, mert amely algoritmusok nem mozdítanak szervert a kérés kiszolgálására, azoknak a szerver konfigurációja nem változik. Amely algoritmusok mozgatnak szervert, azok a Q_{i-1} pontról viszik el a szervert, amely pont az előző kérés volt, így ott minden algoritmusnak van szervere. Következésképp ezen algoritmusok szerver konfigurációja nem állítódhat azonos pozícióba olyan algoritmus szerver konfigurációjával, amely nem mozdított szervert. Másrészt, ha több algoritmus is mozgatná Q_{i-1}-ről Q_i-be a szervert azok szerver konfigurációja se válhat azonossá, hisz a mozgatást megelőzőleg különböző volt.

Következésképp egy Q_i kérés esetén, minden OFF_j algoritmusra más a szerver konfiguráció. Továbbá minden konfigurációnak tartalmaznia kell Q_{i-1}-et, tehát pontosan egy olyan OFF_j algoritmus van, amelynek nincsen szervere a Q_i ponton. Tehát a Q_i kérés kiszolgálásának a költsége az OFF_j algoritmusok egyikénél $d(Q_{i-1}, Q_i)$ a többi algoritmus esetén 0.

Következésképp

$$\sum_{j=1}^{k} \mathrm{OFF}_j(I) = C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}),$$

ahol $C = \sum_{j=1}^{k} C_j$ egy az input sorozattól független konstans, amely az off-line algoritmusok kezdeti konfigurációinak beállításának költsége.

Másrészt az off-line optimális algoritmus költsége nem lehet nagyobb semelyik off-line algoritmus költségénél sem, így $k \cdot \mathrm{OPT}(I) \leq \sum_{j=1}^{k} \mathrm{OFF}_j(I)$. Következésképpen

$$k \cdot \mathrm{OPT}(I) \leq C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \leq C + \mathrm{ONL}(I),$$

amely egyenlőtlenségből következik, hogy az ONL algoritmus versenyképességi hányadosa nem lehet kisebb, mint k, hiszen az inputsorozat hosszúságának növelésével a $\mathrm{OPT}(I)$ érték tetszőlegesen nagy lehet.