

1. Tetszőleges  $n > 3$  esetén előfordulhat, például úgy, hogy az összes többi helyen törölt bejegyzés van és a keresett elem nincs a három tárolt között. Ekkor a keresés során a teljes táblát végig kell járnunk.
2. A beszúrásosnál végig kell csinálni az egészet, mert pl. rossz esetben az utolsó elem a legkisebb, így ez  $O(n \log n)$ , ha bináris kereséssel csináljuk a beszúrást.  
Összefésülésnél annyit lehet csak nyerni, hogy a legutolsó menetben megállhatunk az első  $k$  összehasonlítás után, de ez nagyságrendileg nem csökkent semmit az  $O(n \log n)$  lépésen.  
Kupacnál meg kell építeni a kupacot, ez  $O(n)$ , majd kell  $k$  darab MINTÖR, ez  $O(k \log n)$ , összesen  $O(n + k \log n)$ .

3. (a) Igaz, ha például a bejárást az él egyik végpontjából, az élen át kezdjük.  
(b) Mint az (a)-nál.  
(c) Nem igaz, ellenpélda az  $n > 3$  csúcsú teljes gráf. Ebben minden mélységi fa egy út, de van benne más fa is.  
(d) Nem igaz, ellenpélda ismét az  $n > 3$  csúcsú teljes gráf. Ebben minden szélességi fa egy csillag, de van benne más fa is.

4.  $O(n)$ -ben az eredeti kupac nélkül is tudunk kupacot építeni a  $(-1)$ -szeresekéből.

$\Omega(n)$ : ha menne gyorsabban is, akkor tudnánk  $N$  pozitív szám közül legnagyobbat találni lineárisnál gyorsabban, amiről tudjuk, hogy nem lehetséges. Ez a következőképpen menne: építsünk egy kupacot, melyben az  $N$  elemünk a levelekbe kerül, a belső csúcsokat pedig töltjük fel növekvő sorrendben negatív számokkal. Ez nulla összehasonlítással megvan, mert csak be kell írni a számokat a kupac csúcsaiba. Ezen kupac legnagyobb eleme a levelekben van, az  $N$  szám között. Ha az elemeknek a  $(-1)$ -szereseit vesszük és tudunk belőlük lineárisnál gyorsabban új kupacot csinálni, akkor az új kupacból nulla összehasonlítással megvan az új kupac legkisebb, azaz a régi legnagyobb eleme. Összesen lineárisnál kevesebb összehasonlítás volt, ami nem lehet.

5. *Algoritmus:*

Készítsünk egy új gráfot, melynek csúcsai a benzinkutas falvak és két falu között akkor legyen él, ha kettejük távolsága legfeljebb 600 kilométer. Ekkor az él súlya legyen a kettejük közötti legrövidebb út hossza. Ez a gráf elkészíthető  $k$  darab Dijkstra algoritmussal, minden benzinkutas faluból indítva egyet-egyet, közben tárolhatjuk a legrövidebb utakat is.

Ezt a gráfot átírjuk éllistás megadásra, majd legrövidebb utat keresünk  $A$ -ból  $B$ -be, ez még egy Dijkstra (itt is tároljuk magukat az utakat is). A legrövidebb útvonalat az eredeti gráfban össze tudjuk rakni úgy, hogy az új gráfban talált legrövidebb úton fekvő éleket „kifejtjük” az eredeti gráfból meghatározott legrövidebb utak segítségével.

*Ez jó lesz:* Az új gráfban pontosan akkor van út két csúcs között, ha az eredeti gráfban az előírt feltétellel van útvonal a megfelelő két benzinkutas falu között. Így, ha megkeressük az új gráfban a legrövidebb utat  $A$ -ból  $B$ -be, akkor az éppen a legrövidebb kívánt útvonal lesz az eredetiben.

*Lépésszám:* A  $k$  darab Dijkstra az eredeti gráfban  $O(ke \log n)$ , mert darabja  $O(e \log n)$ , ebbe az utak nyomonkövetése is belefér. Az új gráfot éllistásan megadni  $O(k^2)$  lépés, az új gráfban futtatott Dijkstra pedig  $O(k^2 \log k)$ , mert az új gráfnak  $k$  csúcsa és maximum  $k^2$  éle van. Mivel  $k \leq n$  és az eredeti gráf összefüggő, ezért  $k^2 \leq kn \leq ke$ , vagyis az új gráfon végzett munka is belefér a kívánt  $O(ke \log n)$ -es időbe.

6. Dinamikus programozással dolgozunk, 1-től kezdve, növekvően, minden  $1 \leq j \leq n$ -re meghatározzuk az  $l_1, l_2, \dots, l_j$  szavak egy optimális tördelésének hibáját (ezt  $T[j]$ -vel jelöljük) és eközben magát a tördelést is tároljuk.

$$T[1] = (s - l_1)^2$$

később pedig a következő képlet alapján dolgozunk:

$$T[j] = \min\{T[i - 1] + (s - (l_i + l_{i+1} + \dots + l_j + j - i))^2\}$$

ahol a minimumot azon  $1 \leq i \leq j$  indexekre vesszük, melyekre  $l_i + l_{i+1} + \dots + l_j + j - i \leq s$ . Amelyik  $i$  indexnél a minimum felvétetik, azt megjegyezzük  $T[j]$  mellett, ez mutatja majd, hogy az utolsó sortörés melyik szó előtt lesz az  $l_1, l_2, \dots, l_j$  szavak egy optimális tördelésében.

A módszer helyessége azon múlik, hogy az  $l_1, l_2, \dots, l_j$  szavak egy optimális tördelését úgy kaphatjuk, hogy néhány szót ( $l_i, l_{i+1}, \dots, l_j$ -t) az utolsó sorba teszünk, a korábbiakat pedig optimálisan tördeljük. Ezen esetek közül kiválasztjuk a legkisebbet és megjegyezzük a törés helyét.

A végén  $T[n]$  megadja az optimális tördelés hibáját, maga a tördelés pedig visszakereshető a törési pontokat jelző indexek segítségével.

Lépésszám:  $n$  darab  $T[j]$ -t kell kiszámolnunk, minden egyes  $T[j]$  kiszámolása  $O(n)$  lépésben megvan, mert maximum  $n$  érték közül keressük a legkisebbet és az egyes értékek  $O(1)$ -ben megvannak. ( $l_i + l_{i+1} + \dots + l_j + j - i$  kiszámolása, az egyenlőtlenség ellenőrzése, az utolsó sor hibájának kiszámolása  $O(1)$ -ben megvan, ha  $l_{i+1} + l_{i+2} + \dots + l_j + j - (i + 1)$  már ismert.) Így ha visszafele haladunk az  $i$  értékkel  $j$ -től kezdve, akkor  $O(n)$  időben megvannak azok az értékek, amikből minimumot kell keresnünk.)