Bevezetés

A *Kezdőkönyv az algoritmusokról* lépésenkénti bevezetőt nyújt a számítástechnikai algoritmusok életszerű használatának világába.

A fejlesztők mindennapi munkájuk során algoritmusokkal és adatstruktúrákkal dolgoznak. Az algoritmusok alapos ismerete és annak felismerése, hogy mikor kell alkalmazni őket, nélkülözhetetlen a szoftverek készítése során, hogy azok nemcsak helyesen, hanem megfelelő teljesítménnyel is működjenek.

A könyv célja, hogy a napról napra haladó szoftverfejlesztés során leggyakrabban előforduló algoritmusokat és adatstruktúrákat bemutassa, ugyanakkor maradjon gyakorlatias, pontos, lényegre törő, és igyekezzen nem eltérni az alapszintű témaköröktől és példáktól.

Kinek szól a könyv?

A könyv azoknak szól, akik alkalmazásokat fejlesztenek, vagy éppen fejlesztésbe fognak, és szeretnék megérteni az algoritmusokat és az adatstruktúrákat. A célközönség a programozók, fejlesztők, szoftvermérnök-hallgatók, információrendszer-hallgatók és informatikushallgatók népes tábora.

A könyv szerzői feltételezik, hogy a számítógépes programozás általános ismeretei a birtokukban vannak, és remélik, hogy a kötet a kód kihagyásával – még ha nagyrészt fogalmi szinten is – olvasható és követhető az első oldaltól az utolsóig. Ebből kifolyólag csoportvezetők, építészek és üzleti elemzők is haszonnal forgathatják.

Elvárt előismeretek

Mivel a példaprogramok mindegyike a Java programozási nyelv felhasználásával készült, használható Java-tudásra, valamint a szabványos Java-könyvtárak – különösen a `java.lang` csomag – ismeretére szükség lehet. A tömbökkel, ciklusokkal és egyéb programozási technikákkal sem árt tisztában lenni, és természetesen a Java-osztályok létrehozásának és fordításának miként je is lényeges.

Az itt említett előismereteken kívül más követelmény nem szükséges a kötet adatstruktúrákra vagy algoritmusokra vonatkozó ismeretanyagának elsajátításához.

A könyv témája

A kötetben részletes magyarázatokat, néhány megvalósítást, a mindennapi használatra vonatkozó példákat és gyakorlatokat találunk, amelyek mindegyikének célja, hogy olyan tudás birtokába jussunk, amellyel új ismereteinket az életben is kamatoztatni tudjuk. A könyvben található példák ritkán elméleti természetűek. Az egyes fejezetek kódjait különös gonddal válogattuk össze, és azokat a legtöbb esetben akár azonnal is használhatjuk életszerű alkalmazásokban.

Próbáltunk ragaszkodni a legáltalánosabban elfogadott szoftverfejlesztési gyakorlatokhoz. Ezek közé tartozik a tervezési minták [GoF – Gang of Four, Design Patterns], a kódolási konvenciók, a minőség-ellenőrzések és a teljesen automatizált egységtesztek használata. Remélhetőleg az algoritmusok és az algoritmusok problémamegoldásban betöltött rendkívül fontos szerepének megértésén kívül megtanuljuk, hogy a robusztus, bővíthető és természetesen működő szoftverek építése tiszteletet érdemlő tevékenység.

A Java-nyelvben járatos olvasók felfedezhetnek némi átfedést a könyvben ismertetett osztályok és a `java.util` csomag osztályai között. A könyv nem foglalkozik a Java-könyvtárakban található specifikus megvalósításokkal. Ehelyett inkább bepillantást enged abba, miért tartották fontosnak a Java-nyelv tervezői bizonyos algoritmusok és adatstruktúrák megvalósításainak beépítését csakúgy, mint azok működését és használatát is.

A kötet nem a számítógépes programozás alapjait tanítja meg, sem általában, sem a Java-programozás tekintetében. Nem ismerteti a szabványos Java-könyvtárak használatának szabályait sem: nem ez célja. Noha a példaprogramok használják a `java.lang` osztályait és néhány esetben a `java.io` csomagokat, az összes többi Java-csomag túlmutat a könyv témáján. Ehelyett az összes szükséges osztályt kézzel építjük meg, ezáltal tapasztalhatjuk az algoritmusok felfedezésének örömét.

Noha az egységtesztelés minden fejezetben kiemelt figyelmet kap, a kötet nem egységtesztelési kézikönyv vagy útmutató. Inkább az egységtesztek kódolásának bemutatásával próbálja meg elsajátíttatni az alapszintű egységtesztelés alapismereteit.

A könyv használata

A könyvet az elejétől a végéig érdemes elolvasni. Rendezési, keresési és egyéb meghatározott algoritmusok segítségével a kötet az algoritmusok, adatstruktúrák és teljesítménykarakterisztikák alapjain vezeti végig az olvasót. A könyv négy fő részből áll.

- Az első öt fejezet az algoritmusok alapjait, például az iterációt, a rekurziót ismerteti, mielőtt bevezetné az olvasót az alapvető adatstruktúrák, a listák, a vermek és a sorok világába.
- A 6–10. fejezet különböző rendezési algoritmusokkal foglalkozik, valamint olyan nélkülözhetetlen témákkal, mint a kulcsok és a sorrend kérdése.
- A 7–15. fejezet a tárolás és keresés hatékony módszereivel foglalkozik hasítótáblák, fák, halmazok és leképezések segítségével.
- A 16–19. fejezet speciális és bonyolultabb témaköröket érint, emellett részletezi az általános teljesítménybeli buktatókat és az optimalizálási módszereket.

Minden fejezetben újabb, az előző fejezetek témaköreire épülő fogalmakkal találkozunk, amelyek megalapozzák a következő fejezetek ismeretanyagát. Tehát a könyvet bármelyik fejezetnél felüthetjük, és néhány fejezet átlapozásával megfelelő képet kaphatunk a témáról. Mindenesetre tanácsos minden fejezetben elvégezni a példabeli megvalósításokat, példaprogramokat és gyakorlatokat, hogy a tárgyalt fogalmak és elvek teljesen letisztulhassanak. A könyv végén lévő függelékekben megtaláljuk a további ajánlott olvasmányok listáját, a felhasznált weboldalak listáját és a bibliográfiát.

A megközelítés alapelvei

A kód megértésének többnyire az a legnehezebb része, hogy átlássuk a döntéshozatali folyamatot befolyásoló, gyakran íratlan feltételezéseket és elveket. Ezért tartjuk fontosnak, hogy részleteiben megvilágítsuk a könyvben alkalmazott megközelítést. Betekintést engedünk a logikai alapokba, amelyeket alapvető fejlesztési gyakorlatnak tekintettünk a könyv megírása során. A könyv elolvasása után remélhetőleg az olvasó is méltányolja majd, hogy miért hiszünk a következő elvekben.

- Az egyszerűség jobb kódot eredményez.
- Ne optimalizáljunk idejekorán!
- Az interfészek hozzájárulnak a tervezés rugalmasságához.
- A kódot automatikus egység- és funkcionális tesztelésnek kell alávetni.
- Az assertion technika a fejlesztő legjobb barátja.

Törekedjünk az egyszerűségre!

Milyen gyakran halljuk ezt a megjegyzést: „Ó, ez túl bonyolult! Úgysem értené meg." Vagy: „A kódunk túl nehezen tesztelhető." A szoftvermérnökség lényege a bonyolultság kezelése.

Ha sikerült a célnak megfelelő rendszert építenünk, de a rendszer ismertetése vagy tesztelése túl bonyolult, akkor a rendszer csak véletlenül működik. Gondolhatjuk azt, hogy a megoldást szándékosan valósítottuk meg az adott módon, de a tény, hogy a rendszer működése inkább a valószínűségtől és nem a tiszta determinizmustól függ.

Ha túl összetettnek tűnik, bontsuk le a feladatot kisebb, könnyebben kezelhető részekre. Kezdjük a kisebb problémák megoldásával. Majd a közös kód, a közös megoldások alapján kezdjük el átszervezni és absztrahálni a problémákat. Ily módon a nagy rendszerek kisebb feladatok összetett elrendezésévé alakulnak.

Az „EHMM – egyszerűen, hogy mindenki megértse" jelszóhoz ragaszkodva a könyv összes példája a lehető legegyszerűbb. Mivel a könyv célja, hogy gyakorlati segédletet biztosítson az algoritmusokhoz, a példaprogramokat az életszerű alkalmazásokhoz a lehető legközelebb igazítottuk. Bizonyos esetekben azonban a metódusokat kissé hosszabbra kellett hagynunk, mint szerettük volna, de végül is oktató célzattal készült könyvről van szó, és nem a lehető legtömörebb kód megírásáról.

Ne optimalizáljunk előre!

Csábító lehet rögtön a kezdetektől fogva a kód gyorsaságára törekedni. Az optimalizálás és a teljesítmény érdekessége, hogy a szűk keresztmetszetek sohasem ott vannak, ahol várnánk, és nem is olyan természetűek, mint amilyeneket várnánk. Az ilyen kényes pontok előzetes találgatása költséges gyakorlat. Sokkal jobban járunk, ha jól megtervezzük a kódot, és külön kezeljük a teljesítményjavítás feladatát, amihez a 19. fejezetben ismertetett speciális ismeretekre lesz szükség.

Ha a könyvben kompromisszumot kellett kötni a teljesítmény és az érthetőség között, igyekeztünk az érhetőségre törekedni. Sokkal fontosabb, hogy megértsük a kód elvét és célját, minthogy milliszekundumokat lefaragjunk a futásidőből.

A jó tervet sokkal könnyebb profilírozni és optimalizálni, mint az „okos" kódolással előállított spagettikódot, és tapasztalataink szerint az egyszerű terv eredményeként készített kód kis optimalizálás mellett is remekül teljesít.

Felhasználói interfészek

Az adatstruktúrák és algoritmusok nagy része ugyanazt a külső működést mutatja, még akkor is, ha a mögöttes megvalósítás eléggé eltérő. Az életszerű alkalmazásokban a különböző megvalósítások között gyakran feldolgozási vagy memóriamegszorítások miatt kell választanunk. Az esetek zömében ezek a megszorítások előre nem ismertek.

Az interfészek lehetővé teszik, hogy mögöttes megvalósításra való tekintet nélkül meghatározzuk a megállapodást. Ebből kifolyólag a tervezést a megvalósítás beköthetőségének támogatásával teszik rugalmassá. Ezért van szükség arra, hogy minél inkább az interfészeknek megfelelően kódoljunk, és így lehetővé tegyük a különböző megvalósítások helyettesítését.

A könyv minden példabeli megvalósítása a meghatározott működés interfészműveletekre való fordításával kezdődik. A legtöbb esetben ezek a műveletek a következő két csoport valamelyikébe sorolhatóak: alapszintű vagy elhagyható.

Az alapszintű műveletek biztosítják az adott interfészhez szükséges alapműködést. A megvalósítások általában az első elvekből származnak, és ezért szorosan összefüggnek egymással.

Az elhagyható műveleteket ezzel szemben az alapszintű műveletekre alapozva valósíthatjuk meg, és rendszerint a fejlesztő kényelmét szolgálják. Szükség szerint magunk is könnyűszerrel megvalósíthatjuk őket saját alkalmazáskódunkban. Mivel a gyakorlatban sokan használják őket, ezeket a műveleteket az alapszintű API részének tekinthetjük, és egy adott témakör tárgyalását addig nem fejezzük be, amíg mindegyiket részleteiben meg nem valósítottuk.

Tesztelni, tesztelni!

A korszerű fejlesztési gyakorlat megköveteli, hogy a szoftver szigorúan egyesített és funkcionálisan tesztelt legyen a kód integritásának biztosítása érdekében. A megközelítést követve az interfész definiálása után, de még bármilyen konkrét megvalósítás definiálása előtt funkcionális követelményeinket tesztesetre fordítjuk annak ellenőrzésére, hogy minden feltétellel foglalkoztunk, és megerősítettünk őket.

A tesztek a JUnit segítségével készültek, amely a Java tényleges szabványos tesztelési keretrendszere, és a tesztek a megvalósítás minden funkcionális szempontját ellenőrzik.

A tesztek a meghatározott interfészek alapján, nem pedig bármilyen konkrét megvalósítás alapján készültek. Ez lehetővé teszi, hogy az összes megvalósítás esetén ugyanazokat a teszteket alkalmazzuk, és így biztosítsuk az egységes minőséget. Ezenkívül a különböző teljesítménykarakterisztikákat is bemutatják, ami akkor fontos, ha az alkalmazásban a használni kívánt különböző megvalósítások között válogatunk.

A tesztelés puristái kifogásolják, hogy a tesztek az ő ízlésük szerint túl hosszúak, és egy metódusban túl sok dolgot tesztelnek. Hajlamosak lennénk egyetérteni velük, de a megértés támogatása érdekében leegyszerűsítjük a dolgokat, és alkalmanként úgy gondoltuk, hogy vehetjük magunknak a bátorságot, és néhány helyzetet összevonhatunk egyetlen tesztmetódusban.

A lényeg, hogy mielőtt bármilyen megvalósítási kódot elkészítenénk, először írjuk meg a teszteket. Ez a megközelítés, a *tesztelésen alapuló programozás (test-driven development, TDD)* az osztályok megállapodására, azaz a közzétett viselkedésre összpontosít, nem a megvalósításra. Lehetővé teszi, hogy a teszteseteket majdhogynem a kód követelményeiként vagy használatának eseteiként kezeljük; a tapasztalatok szerint ez is egyszerűsíti az osztályok terveit. Mint a példákban látni fogjuk, azáltal, hogy az interfészekhez kódoljuk tesztjeinket, gyerekjáték lesz a tesztelésen alapuló programozás.

Legyünk alaposak!

A tesztelés szigorúsága miatt önelégültté válhatunk, és azt hihetjük, hogy a kódunkat teljes alapossággal teszteltük, ezért az hibamentes. A baj csak az, hogy a tesztek nem feltétlenül bizonyítják, hogy a szoftver azt a feladatot hajtja végre, amit kell. Ehelyett csupán azt igazolják, hogy a szoftver az adott helyzetekben és feltételekkel működik, de ezek nem mindig fedik le a valóságot. Lehet, hogy a világ legnagyszerűbb, legátfogóbb tesztcsomagjával rendelkezünk, de ha rossz dolgokat tesztelünk, semmit sem ér az egész.

A gyors hibázás elve alapján ajánlott a defenzív programozás: ellenőrizzük a nullmutatókat, győződjünk meg róla, hogy az objektumok a metódus elején megfelelő állapotban vannak, és így tovább. A gyakorlat bebizonyította, hogy ezzel a programozási móddal hamarabb megtalálhatjuk az összes különös programhibát, és nem kell a `NullPointerException` kivételre várnunk.

Mielőtt bármilyen objektum állapotáról vagy paraméter típusáról bármit is feltételeznénk, a kód vizsgálatával ellenőrizzük a feltevést. Ha bármikor azt gondoljuk, hogy valami sohasem fordulhat elő, ezért nem is kell aggódnunk miatta, végezzünk kódszintű érvényességvizsgálatot!

Képzeljük el például, hogy az adatbázisban van egy pénzügyi mező, amelyről „tudjuk", hogy „soha" nem tartalmaz majd negatív értéket. Ha a vizsgálatokat kikapcsoljuk, valamikor, valahogyan egy negatív érték egészen biztosan *megjelenik* a mezőben. Lehet, hogy napok, hónapok vagy évek telnek el, mire észrevesszük ennek következményeit. Előfordulhat, hogy a rendszer más részeiben is befolyásolja más számítások működését. Ha az összeg –0,01 cent volt, a különbség alig észrevehető. Mire felfedezzük a problémát, már nem tudjuk az összes káros mellékhatást meghatározni, nem is beszélve arról, hogy ki is kellene javítani őket.

Ha engedélyeztük volna a kódszintű érvényességvizsgálatot, a szoftver teljesen megjósolható módon hibát jelzett volna abban a pillanatban, hogy a baj előállt, és valószínűleg a probléma diagnosztizálásához szükséges összes információ is a rendelkezésünkre állt volna. Ehelyett jóvátehetetlenül megsérültek a rendszer adatai.

Az éles kód vizsgálata lehetővé teszi, hogy a kód hibái megjósolható módon álljanak elő, ami lehetővé teszi a probléma okának és természetének könnyű és gyors azonosítását, és elhanyagolható segédszámítási költségekkel jár. Egyetlen pillanatig se gondoljuk, hogy a vizsgálatok hátrányosan befolyásolják a rendszer teljesítményét. Jó esélyünk van arra, hogy a kód összes vizsgálatához szükséges idő nem összemérhető egy távoli eljáráshívásban vagy adatbázis-lekérdezésben töltött idővel. Ajánlatos az éles kódban bekapcsolt állapotban hagyni a vizsgálatokat.

Mire van szükség a könyv használatához?

A felépítés és futtatás nem is lehetne könnyebb. Ha kezdeti előnnyel szeretnénk indulni, a teljesen működőképes projektet forráskóddal, tesztekkel együtt, valamint az automatizált parancssori verziót letölthetjük a Wrox webhelyéről (lásd a „Forráskód" című részt).

Ha a „csináld magad" megközelítés hívei vagyunk, szerencsénk van, mert így minimalizálhatjuk a függőségek számát. Kiindulásként a következőkre van szükségünk:

- Java Development kit (JDK) 1.4 vagy újabb verziója, amely tartalmazza a kód fordításához és futtatásához szükséges összes komponenst;
- JUnit-könyvtár, amely egyetlen jar fájlból áll, és ha egységteszteket szeretnénk fordítani és futtatni, a classpath környezeti változónak tartalmaznia kell a fájlt;
- szövegszerkesztő vagy integrált fejlesztői környezet (Integrated Development Environment – IDE) a kódoláshoz.

Az első két tétel (a JDK és a JUnit) ingyenesen letölthető az internetről (lásd a B függeléket). Az utolsó követelmény tekintetében nem szeretnénk vitát kirobbantani, ezért a választást az olvasóra bízzuk. Egészen biztosan van kedvencünk, ragaszkodjunk hozzá! Ha nincsen olyan program, amellyel kódolhatnánk, kérdezzük meg barátainkat, hallgatótársainkat, előadóinkat vagy kollégáinkat. Egészen biztosan szívesen megosztják velünk véleményüket.

Mivel a Javáról van szó, a példaprogramokat bármely operációs rendszeren lefordíthatjuk és futtathatjuk. A könyvet Apple Macintosh és Windows alapú számítógépeken írtuk és fejlesztettük. Egyetlen kód sem különösebben processzorintenzív, tehát a szoftverfejlesztéshez használt hardverünk biztosan megfelel majd.

A könyvben használt jelölések

Annak érdekében, hogy a legtöbb új ismeret birtokába juthassunk, és nyomon tudjuk követni, mi történik, a könyvben az alábbi jelöléseket alkalmaztuk:

Gyakorlófeladat

A *Gyakorlófeladat* elnevezésű részben érdemes végigcsinálni a feladatot a könyv utasításait követve.

1. A *Gyakorlófeladat* rendszerint több kódolt lépést tartalmaz.
2. A lépések nem mindig számozottak, néhányuk nagyon rövid, míg mások a nagyobb, végső célhoz vezető, kis lépések sorozatából állnak.

A megvalósítás működése

Minden *Gyakorlófeladat* után *A megvalósítás működése* című részben találjuk a kódblokkok működésének részletes magyarázatát. A könyv témája, az algoritmusok kérdése nem igazán felel meg számozott feladatoksorok elvégzésének, sokkal inkább a gyakorlati példáknak, tehát észre fogjuk venni, hogy a *Gyakorlófeladat* és *A megvalósítás működése* megfelelően módosult. Az alapelv az, hogy alkalmazzuk a megszerzett tudást.

Az ilyen dobozokban a közvetlenül a dobozt körülvevő szövegre vonatkozó fontos információkat találunk, amelyekről nem szabad elfeledkeznünk.

Az aktuális témára vonatkozó tippek, ötletek, trükkök dőlt betűvel, kissé beljebb húzva szerepelnek.

A szövegben megjelenő betűtípusokkal kapcsolatban:

- A fontos szavakat bevezetésük során kiemeljük.
- A billentyűleütések a következő formában jelennek meg: Ctrl+A.

- A fájlnevek, az URL-ek és a kódok a következőképpen szerepelnek a szövegben: `persistence.properties`.
- A kódokat kétféle változatban láthatjuk:

```
A példaprogramokban az új és fontos kódot szürke háttérrel
emeljük ki.
```

```
A szürke háttér nem jelenik meg az aktuális témában kevésbé fontos
vagy már korábban bemutatott kód mögött.
```

Forráskód

A könyv példáinak elvégzésekor mi magunk is begépelhetjük kézzel a kódot, vagy használhatjuk a könyvhöz tartozó forráskódfájlokat is. A könyvben használt példák forráskódja letölthető a `http://www.wrox.com` címről. Ha már ezen a címen járunk, keressük meg a könyvet (a Search doboz vagy az egyik címlista segítségével), majd kattintsunk a könyvet részletező oldal Download Code hivatkozására, és töltsük le a könyv összes forráskódját!

> *Mivel több, hasonló című könyv található az oldalon, keressünk az ISBN-szám segítségével; az eredeti könyv ISBN száma: 0-7645-9674-8 (a 2007 januárjában bevezetésre kerülő új, 13-jegyű ISBN-számozás szerint ez a szám 978-0-7645-9674-2 lesz).*

A kód letöltése után tömörítőeszközünk segítségével csomagoljuk ki a kódot. A másik lehetőség, ha a Wrox-kód letöltési oldalára, a `http://www.wrox.com/dynamic/books/download.aspx` címre lépünk, és megkeressük a könyv és más Wrox könyvek kódjait.

Hibajegyzék

Mindent elkövettünk annak érdekében, hogy a könyv szövege és a kódok ne tartalmazzanak hibákat. De senki sem tökéletes, és hibák előfordulhatnak. A könyvben talált hibákkal kapcsolatos visszajelzésekért hálásak vagyunk. Hibajegyzékek beküldésével egy másik olvasó számára megtakaríthatunk többórányi bosszankodást, ugyanakkor segíthetünk, hogy a könyv még jobb információkat biztosítson.

A könyv hibajegyzékoldalát a `http://www.wrox.com` címen találjuk, ha a Search doboz vagy az egyik címlista segítségével megkeressük a könyvet. A könyvet részletező oldalon kattintsunk a Book Errata hivatkozásra! Az oldalon megtaláljuk a

könyvvel kapcsolatban már bejelentett hibákat, amelyeket a Wrox szerkesztői küldtek el. A teljes könyvlista, amely az egyes könyvek hibajegyzékeit tartalmazza, a `www.wrox.com/misc-pages/booklist.shtml` címen található.

Ha nem találjuk „saját" hibánkat a hibajegyzékoldalon, a `www.wrox.com/contact/techsupport.shtml` oldalon töltsük ki az űrlapot, és küldjük el a felfedezett hiba leírását. A Wrox szerkesztői ellenőrzik az információkat, és ha szükséges, a könyv hibajegyzékében üzenet jelenik meg a hibáról, a könyv következő kiadásaiban pedig kijavítjuk.

p2p.wrox.com

A szerzőkkel és az olvasókkal a `p2p.wrox.com` címen a P2P vitafórumokhoz csatlakozva beszélgethetünk. A fórum olyan webes rendszer, amelyben a Wrox-könyvekre és azokkal kapcsolatos technológiákra vonatkozó üzeneteket küldhetünk, és a többi olvasóval, illetve a technológia felhasználójával folytathatunk beszélgetéseket. A fórumok előfizetési funkciót biztosítanak, amelynek segítségével a számunkra érdekes témakörökhöz való új hozzászólás érkezésekor e-mailben értesítést kapunk. A Wrox szerzői, szerkesztői, számítástechnikai szakértői és olvasói küldenek üzeneteket ezekbe a fórumokba.

A `http://p2p.wrox.com` címen több fórumot is találunk, amelyek nemcsak a könyv olvasását, de saját alkalmazásaink fejlesztését is segítik. Ha szeretnénk csatlakozni a fórumokhoz, kövessük az alábbi lépéseket:

1. Lépjünk a `p2p.wrox.com` címre, és kattintsunk a Register hivatkozásra!
2. Olvassuk el a felhasználás feltételeit, majd kattintsunk az Agree gombra!
3. Töltsük ki a csatlakozáshoz szükséges és az egyéb információkat, amelyeket szeretnénk megadni, majd kattinsunk a Submit gombra!
4. Ezután e-mailben kapunk értesítést arról, hogyan tudjuk ellenőrizni a fiókunkat, és befejezni a csatlakozási folyamatot.

A P2P-hez való csatlakozás nélkül is olvashatjuk a fórum üzeneteit, de ha saját üzenetet szeretnénk küldeni, akkor csatlakoznunk kell.

Ha csatlakoztunk, új üzeneteket küldhetünk, illetve válaszolhatunk más felhasználók üzeneteire. Az üzeneteket bármikor elolvashatjuk az interneten. Ha adott fórum új üzeneteit szeretnénk e-maiben megkapni, a fórumok listájában kattintsunk a fórum neve mellett a Subscribe to this Forum ikonra.

A Wrox P2P használatáról további információkat a P2P gyakran ismétlődő kérdések listáiban találunk, ahol a fórumszerver működésére, a P2P-re és a Wrox könyvekre vonatkozó kérdéseinkre is választ kaphatunk. A gyakran ismétlődő kérdéseket a GYIK-hivatkozásra kattintva bármely P2P oldalon elolvashatjuk.