**Óbudai Egyetem**

Programozás I. féléves beadandó



**Alkimisták**

Biró Attila

Laborvezető:

Nagy Tibor István

Neumann János Informatika Kar

Budapest, 2013. november 29.

Tartalom

[1. Feladat 3](#_Toc374108199)

[Alkimisták 3](#_Toc374108200)

[2. Megvalósítás 3](#_Toc374108201)

[2.1 A megvalósítás során alkalmazott osztályok 3](#_Toc374108202)

[2.1.1 Bejegyzesek osztály 4](#_Toc374108203)

[2.1.2 Program osztály 4](#_Toc374108204)

[2.1.3 Feladat\_Rek osztály 5](#_Toc374108205)

[2.2. Alkalmazott fontosabb algoritmusok ismertetése 6](#_Toc374108206)

[2.2.1. Rekurzív bejárás 6](#_Toc374108207)

[2.2.2. Beolvasás 8](#_Toc374108208)

[2.3. Specifikáció 8](#_Toc374108209)

[3. Felhasználó segédlet 9](#_Toc374108210)

[4. Felhasznált irodalom 11](#_Toc374108211)

# 1. Feladat

## Alkimisták

Az alkimisták hosszas kísérleteiket arról, hogy milyen anyagok milyenekké alakíthatók át, gondosan feljegyezték könyveikbe. Bejegyzéseik a lehető legegyszerűbbek voltak: Megadták a kiindulási anyag nevét, majd azt az összetevőt (ún. katalizátort), amit hozzákeverve ehhez létrejött valamilyen végtermék. A bejegyzéseket tanulmányozva megállapították, hogy egyetlen anyag sem állítható elő önmagából, egy vagy több lépésben sem, továbbá nincs két különböző bejegyzés, amelyekben mind az első, mind a második anyag megegyezne. Az anyagok neveit számokkal, míg a katalizátorokat az angol ABC nagy betűivel jelölték A-tól Z-ig. Az alkimisták aranyat szerettek volna előállítani vasból. Feljegyzéseikben az aranyat 0-val, míg a vasat 1-el jelölték.

Az ARANY.BE állomány első sorában megadjuk, hogy hány bejegyzés szerepel a sorozatban, a többi sora pedig a fenti példában szereplő formátumban tartalmazza az egyes bejegyzéseket (a kiindulási anyag száma, szóköz, a katalizátor betűjele, szóköz és a végtermék száma). A használt anyagok száma maximum 200, a katalizátoroké pedig 26. A bejegyzések száma legfeljebb 1000.

Az ARANY.KI állományba a nélkülözhetetlen katalizátorok betűjelét kell írni, egy sorban szóközzel elválasztva. Ha aranyat nem lehet előállítani semmilyen katalizátorral, akkor NEM LEHET szerepeljen a kimenetben. Ha egyik katalizátor sem nélkülözhetetlen, akkor EGYIK SEM KELL legyen a sor tartalma.

# 2. Megvalósítás

A feladat megoldásához használtam egy verziókövető rendszert, a git-et. Így könnyen vissza tudtam keresni az előző működő verzióját a programomnak, ha éppen valami nem működött (<https://github.com/Hatbatamado/beadando>).

A feladatot rekurzív függvénnyel valósítottam meg, mert nem lehet előre tudni, hogy az anyagok hány fele fognak elágazni. Tanult módszerek alapján készítettem bele hibakereső részt is, így a felhasználó nem tud, olyan fájlt megadni, ami alapján a program nem futna le. A program egy string tömb alapján tudja, hogy melyik elágazásokon ment át és folyamatosan változtatja ennek a tömbnek méretét, ahogy éppen szükséges (új elágazásnál hozzáad, bejárt elágazás után töröl). Így be tudja járni az összes lehetséges útvonalat és meg találja, hogy melyik katalizátorokkal lehet előállítani vasból aranyat.

2.1 A megvalósítás során alkalmazott osztályok  
A feladat megoldásához két osztályt hoztam létre. A Bejegyzesek osztályt, amely alapján egy tömbben el tudom tárolni a beolvasott bejegyzéseket. A Feladat\_Rek osztályt, amelyben beolvasás után lefut program. A fájl beolvasását és tartalmának előrzését az alap osztályban, a Program osztályban hajtottam végre.

## 2.1.1 Bejegyzesek osztály

Az osztály mezőinek áttekintése:

**kezdo\_anyag:** a kiindulási anyag száma

**veg\_anyag:** végtermék száma

**katalizator:** katalizátor betűjele

Az osztály tulajdonságainak áttekintése:

**Kezdo\_anyag:** Publikus olvasható tulajdonság, a kezdo\_anyag mező értékét adja meg

**Veg\_anyag:** Publikus olvasható tulajdonság, a veg\_anyag mező értéket adja meg

**Katalizator:** Publikus olvasható tulajdonság, a katalizator mező értéket adja meg

Az osztály nem tartalmaz metódust, csak egy konstruktort, ami beállítja a mezők értékét.

## 2.1.2 Program osztály

Egy statikus mezőt tartalmaz az osztály:

**bejegy:** egy tömb, ami a Bejegyzes osztálynak a mezői alapján tárol adatokat, a tömb méretét az egyik metódus állítja be

Az osztály metódusainak áttekintése:

**Beolvas()**: Rejtett int metódus, ami visszatér egy „hibakóddal” vagy pedig beállítja a bejegy méretét és feltölti azt a helyes adatokkal.

**Ellenorzes():** Rejtett int metódus, ami paraméterként egy int értéket vár (fájl első sorában beolvasott értéket), ha az nagyobb, mint 1000, akkor az **1-es „hibakóddal”** tér vissza, külön pedig a **0-ssal**

**Ellenorzes():** Rejtett int metódus, ami paraméterként két int értéket vár (fájl első sorában beolvasott érték; a fájl további sorainak a száma). Ha a második int érték nagyobb, mint 1000, akkor **2-es kóddal** tér vissza, különben, ha a két érték nem egyezik meg, akkor a 3-as kóddal tért vissza, egyébként pedig a **0-ssal**

**Ellenorzes():** Rejtett int metódus, ami paraméterként két int és egy string értéket vár (kiindulási anyag számát, végtermék számát és a katalizátor betűjelét). Ha a kiindulási vagy a végtermék anyag száma nagyobb, mint 200, akkor 4-es **kóddal** tér vissza, különben megvizsgálja a katalizátort, hogy az angol ABC betűi közé tartozik-e, ha nem, akkor **6-os kóddal** tér vissza, egyébként pedig a **0-ssal**

**Ellenorzes():** Rejtett int metódus, ami paraméterként egy bejegy alakú tömböt vár, aminek az összes elemét megvizsgálja, hogy van-e olyan két különböző bejegyzés, ahol a kiindulási anyag száma és a végtermék száma megegyezik-e. Ha van ilyen, akkor **5-ös kóddal** tér vissza, különben pedig **0-ssal**

**Main():** főprogram, bekéri a fájl útvonalát, majd pedig meghívja, a Beolvas() metódust és vagy a hibakódnak megfelelő szöveget futtatja le, vagy pedig megoldja a feladot.

## 2.1.3 Feladat\_Rek osztály

Az osztály mezőinek áttekintése:

**db:** keresett anyag db száma

**elag\_in:** elágazások indexét tartalmazó string

**elag\_t:** string tömb, az elágazások indexének tömbje, mérete változik a program futása során

**elag\_sor:** elágazások tömbjének sorszáma

**vegig:** string tömb, ami a bejárt útvonalat tárolja vasból-aranyig indexekkel, mérete változik a program futása során

**bejegy:** a főprogramból átküldött tömb, ami a Bejegyzes osztálynak a mezői alapján tárol adatokat

Az osztály metódusainak áttekintése:

**Rekur():** Rejtett void metódus, amely paraméterként egy int értéket vár. Ez alapján megkeresi a soron következő anyago(ka)t, elmenti az indexet, ahol elágazás van, hogy vissza tudjon oda ugorni, törli a bejárt (felesleges) elágazásokat, ha aranyat talál, akkor menti az útvonalat, minden egyéb esetben törli azt.

**Tomb():** Rejtett string tömbbel visszatérő metódus, ez a metódus állítja be az *elag\_t* méretét és növeli azt, ha újabb anyag kerül bele.

**Torol():** Rejtett string tömbbel visszatérő metódus, megkeresi az *elag\_t* tömb adott indexén azt az elágazást, amin még nem haladtunk át, vagy pedig addig csökkenti az indexet, amíg nem talál ilyet, vagy el nem fogy az index. Ha találunk még elágazást, akkor az alapján létrehozzuk a tömböt, ha nem akkor null-al térünk vissza.

**Tomb\_Torol():** Rejtett void metódus, a Torol() metódussal beállítja az *elag\_t* új méretét és értékeit, ha ez nem null, akkor meghívja a Rekur() metódust a következő anyag számával.

**Tomb\_Eredmeny():** Rejtett void metódus, paraméterként egy stringet vár, a bejárt indexeket. Ez alapján beállítja a *vegig* tömb méretét és/vagy feltöti azt az új bejárt indexekkel.

**Tomb\_Eredmeny\_Torles()**: Rejtett void metódus, paraméterként egy int értéket vár, ami alapján el tudjuk dönteni, hogy a program aranyat talált, vagy pedig nem találtunk, vagy az anyag önmagát állítja elő. Ha aranyat találtunk, akkor *vegig* tömb végére berakjuk az első sor értékét (itt találtuk az aranyat) majd pedig a többit előrébb hozzuk. Ha nem találtunk vagy önmagát állítja elő, akkor töröljük az első sor tartalmát és a többit előrébb hozzuk.

**Ellenorzes():** Rejtett bool metódus, paraméterként egy int értéket vár, a keresett anyagot. Megnézi, hogy az adott anyag szerepel-e már *vegig* tömbben (önmagát állítja-e elő), ha szerepel, akkor hamis értékkel térünk vissza, különben igazzal.

**Kiir():** Rejtett string metódus, a *vegig* tömböt alakítja át string-é, mivel már csak a vas-arany útvonalak vannak benne, csak a katalizátor betűjelét használva (publikusként a teljes megoldást ki lehet vele íratni nélkülözhető + nélkülözhetetlen anyagok).

**Megoldas():** Publikus string metódus, Kiir() metódus stringje alapján dolgozik. Ha talál egy olyan katalizátort, ami egynél többször szerepel, akkor az nélkülözhető anyag. Ha talál olyant, ami csak egyszer szerepel, akkor az nélkülözhetetlen anyag és ezt belerakja a visszatérési stringbe. Ha egy nélkülözhetetlen sincs, akkor az EGYIK SEM KELL szöveg lesz a stringben, ha nem lehet arany előállítani, akkor pedig NEM LEHET lesz a stringben.

# 2.2. Alkalmazott fontosabb algoritmusok ismertetése

## 2.2.1. Rekurzív bejárás

Mivel nem tudjuk előre, hogy hány fele ágaznak el a bejegyzések, ezért rekurzív módszert választottam a probléma megoldására. A program string tömb segítségével addig fut, amíg az összes lehetséges útvonalat be nem járta. 1-es kezdő anyaggal, vassal indul. Az út bejárása során az elágazásokat a string tömbbe elmenti. Ez alapján járja be a bejegyzéseket. Ha talál 0-t, azaz aranyat, akkor az eredmény string tömb végére berakja. Majd pedig törli az elágazás tömbből a bejárt útvonalat, és ha még van olyan, amin nem haladott át, akkor azzal folytatja. Ha nem talál 0-t, vagy az Ellenorzes metódus olyan anyagot talál, ami önmagát állítja elő, akkor törli az adott útvonalat az eredmény tömbből és az elágazások tömbjéből is. Majd megkeresi, hogy van-e olyan elágazás, amin még nem haladott át, ha van ilyen, akkor azzal meghívja a rekurzív függvényt újra.

bejegy\_tomb 🡨 beolvasott bejegyzések tömbje

elag\_in 🡨 elágazások indexe (keresett anyag(ok) helye bejegy\_tomb-ben)

Elagazas\_tomb 🡨 elágazások indexének a tömbje  
Eredmeny\_tomb 🡨 vas-arany útvonal indexeinek tömbje

**Függvény** Rekur (keresett\_anyag)

**Ha** Ellenorzes(keresett\_anyag) **akkor**

**Ha** keresett\_anyag != 0 **akkor**

db 🡨 Keresett\_anyag\_megszamlalas

elag\_in 🡨 elag\_in + keresett\_anyag index(ek)

**Ha** db > 1 **akkor**

Elagazas\_tomb 🡨 Elagazas\_tomb + elag\_in

Eredmeny\_tomb 🡨 Eredmeny\_tomb + elag\_in Rekur(bejegy[Elagazas\_tomb[0]].Veganyag)

**Különben Ha** db = 1 **akkor**

Eredmeny\_tomb 🡨 Eredmeny\_tomb + elag\_in

Rekur(bejegy[elag\_in].Veganyag)

**Különben**

Elagazas\_tombbol\_bejart\_ut\_torlese

Eredmeny\_tombbol\_bejart\_ut\_torlese

**Ha** Elagazas\_tomb\_bejaratlan\_ut\_kereses **akkor**

Rekur(Elagazas\_tomb[0].Veganyag)

**Elágazás** **vége**  
**Elágazás** **vége**

**Különben**

Bejart\_utvonal\_Eredmeny\_tomb\_vegere

Elagazas\_tomb\_bejart\_ut\_torlese

**Ha** Elagazas\_tomb\_bejaratlan\_ut\_keresese **akkor**

Rekur(Elagazas\_tomb[0].Veganyag)

**Elágazás** **vége**

**Elágazás** **vége**

**Különben**

Elagazas\_tombbol\_bejart\_ut\_torlese

Eredmeny\_tombbol\_bejart\_ut\_torlese

**Ha** Elagazas\_tomb\_bejaratlan\_ut\_kereses **akkor**

Rekur(Elagazas\_tomb[0].Veganyag)

**Elágazás** **vége**

**Elágazás** **vége**

**Függvény** **vége**

## 2.2.2. Beolvasás

A fájl beolvasását részletesen elemeztem, majd pedig a főprogramban e szerint készítettem hibaüzeneteket, ha a felhasználó rossz elérési útvonalat, nem megfelelő fájl formátumot adna meg. Így nem kell semmilyen megszorítást alkalmaznom és a felhasználó könnyen rájön, hogy mit rontott el.

bejegy\_tomb 🡨 beolvasott bejegyzések tömbje

**Függvény** int Beolvas(eleres)

hiba 🡨 -1  
 **Ha** eleres létezik **akkor**

db 🡨 első beolvasott sor (bejegyzések db száma)

hiba 🡨 Ellenorzes(db)

**Ha** hiba = 0 **akkor**

bejegyzesek\_db 🡨 bejegyzések megszámlálása

hiba 🡨 Ellenorzes(db, bejegyzesek\_db)

**Ha** hiba = 0 **akkor**

bejegy\_tomb\_meret 🡨 db

i 🡨 1

**Ciklus amíg** nincs\_vege\_a\_fajlnak && hiba = 0

kezd\_a 🡨 beolvasott kiindulási anyag száma

veg\_a 🡨 beolvasott végtermék száma

hiba 🡨 Ellenorzes(kezd\_a, beolv.\_katalizator, veg\_a)

**Ha** hiba = 0 **akkor**

bejegy\_tomb[i++] 🡨 beolv. aktuális bejegyzés

**Elágazás** **vége**

**Ciklus vége**

**Elágazás** **vége**

**Elágazás** **vége**

**Ha** bejegy\_tomb\_meret != 0 && hiba = 0 **akkor**

hiba 🡨 Ellenorzes(bejegy)

**Elágazás** **vége**

**Elágazás** **vége**

return hiba

**Függvény** **vége**

# 2.3. Specifikáció

A program olyan fájlt vár, aminek a formátuma:

1. sor: bejegyzések db száma (kisebb, mint 1000) itt számnak kell szerepelnie

bejegyzések formátuma:  
kiindulási\_anyag\_száma szóköz katalizátor\_betűjele szóköz végtermék\_száma

kiindulási\_anyag\_száma és a végtermék\_száma maximum 200 lehet

katalizátor\_betűjele 26 féle lehet, az angol ABC-nek megfelelően

Pl.:

4

1 A 2

2 B 3

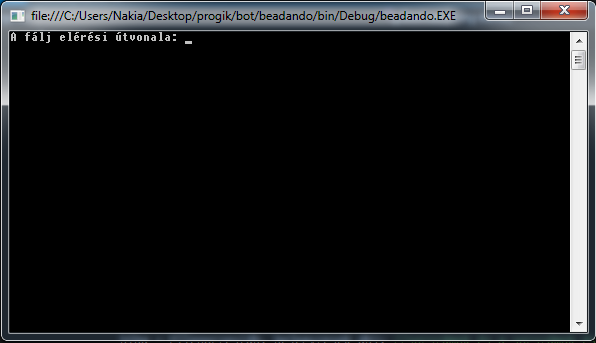
3 C 4

4 D 0

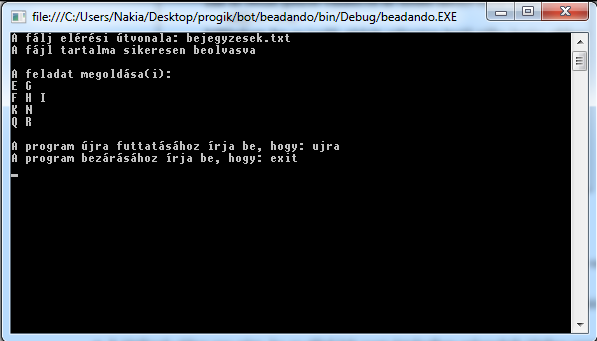
A felhasználó megadhat rossz elérési utat és fájl formátumot, mert ezt a program hibaüzenetekkel le tudja kezelni.

# 3. Felhasználó segédlet

A program először bekéri a fájl elérési útvonalát:



Ezt követően, ha a felhasználó megfelelő formátumú és elérési fájlt adott meg, akkor a program kiírja, hogy sikeresen beolvasta a fájlt. Majd pedig a feladat megoldását. Ez után pedig a felhasználó dönthet arról, hogy újra lefuttatja a programot a fájl tartalmát vagy elérési útját megváltoztatva. Ehhez az „ujra” szöveget kell beírni, ha nem akar újból futtatni, akkor az „exit” szót kell beírnia a felhasználónak.



**-Rossz elérési útvonal esetén, vagy ha a fájl nem létezik, akkor a program kiírja, hogy:**

*„A megadott helyen fájl nem található vagy nem létezik”*

**-Ha a fájl első sorában túl nagy (>1000) érték szerepel, akkor ezt írja ki:**

*„A fájl első sorában található bejegyzések db száma túl nagy, max 1000 lehet*

*Javítsa ki a fájlt, majd ha kész, akkor program újraindításához írja be, hogy: vissza*

*A program bezárásához írja be, hogy: exit”*

**-Ha a bejegyzések db száma meghaladja az 1000-et, akkor ezt írja ki:**

*„A fájlban található bejegyzések száma több mint 1000, maximum 1000 lehet*

*Javítsa ki a fájlt, majd ha kész, akkor program újraindításához írja be, hogy: vissza*

*A program bezárásához írja be, hogy: exit”*

**-Ha a bejegyzések db száma és az első sorban lévő érték nem egyezik meg, akkor ezt írja ki:**

*„Az első sorban lévő db szám és a bejegyzések száma nem egyezik meg*

*Javítsa ki a fájlt, majd ha kész, akkor program újraindításához írja be, hogy: vissza*

*A program bezárásához írja be, hogy: exit”*

**-Ha a kiindulási anyag száma vagy a végtermék száma nagyobb, mint 200, akkor ezt írja ki:**

*„Valamelyik kezdő/vég anyag száma meghaladja a maximumot, a 200-at*

*Javítsa ki a fájlt, majd ha kész, akkor program újraindításához írja be, hogy: vissza*

*A program bezárásához írja be, hogy: exit”*

**-Ha van két különböző bejegyzés, ahol a kezdő és a vég anyag megegyezik, akkor ezt írja ki:**

*„Van két különböző bejegyzés, ahol a kezdő és a vég anyag megegyezik*

*Javítsa ki a fájlt, majd ha kész, akkor program újraindításához írja be, hogy: vissza*

*A program bezárásához írja be, hogy: exit”*

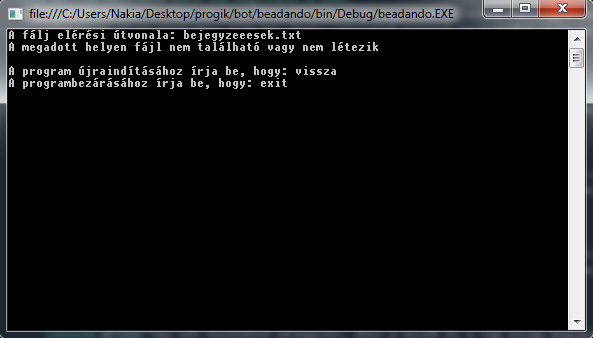
**-Ha a katalizátornak nem megfelelő érték van adva, akkor ezt írja ki:**

*„Az egyik katalizátor nem az angol abc betűi közé tartozik*

*Javítsa ki a fájlt, majd ha kész, akkor program újraindításához írja be, hogy: vissza*

*A program bezárásához írja be, hogy: exit”*

Pl.:



# 4. Felhasznált irodalom

*1. Moodle: 2013/14/1 - Programozás I. - NSTPR1SAND/PR1\_EA tárgy diái*

*2. Moodle: 2013/14/1 - Programozás I. - NSTPR1SAND/PR1\_LA\_01-19 tárgy diái*