

# Fejlesztői dokumentáció

Balog Ádám Márk (ELAO0E)

May 2020

## 1. Osztályok használata

A felhasználó az általa írt programban meghívja a osztálykönyvtárat a következő sorral a fejlécben:

```
#include "simplex.h"
```

Ezután használhatja az alább kifejtett osztályok, és azok publikus tagfüggvényeinek bármelyikét.

## 2. Osztályok, tagfüggvények:

### 2.1. class Rac

Racionális számokat számláló/nevező alakban, relatív prímeikként tároló osztály.

#### 2.1.1. private: (ezeket közvetlenül nem tudja hívni/elérni)

- **int** sz: számláló egész értéke
- **int** n: nevező egész értéke
- **void** simplify():
  - **Működése:** Egyszerűsítő függvény, melyet a konstruktor minden alkalommal hív, ezáltal biztosítva sz és n relatv prím viszonyát.

#### 2.1.2. public:

- **Rac**(int sz, int n)
  - Paraméterek:
    - \* **int** sz: alapértelmezetten 0, a szám számlálója
    - \* **int** n: alapértelmezetten 1, a szám nevezője

- **Működése:** Konstruktor, létrehoz egy példányt a `Rac` osztályból a megfelelő paraméterekkel. 0-val a nevezőben hibát dob. Relatív prímeket tárol
- **int** `lnko(int x, int y);`
  - **Paraméterek:**
    - \* **int** `x`: az egyik egész
    - \* **int** `y`: másik egész
  - **Működése:** Megkeresi a két szám legnagyobb közös osztóját az Euklideszi algoritmus segítségével.
- **int** `getsz()const;`
  - **Működése:** Visszatér az adott példány számlálójával.
- **int** `getn()const;`
  - **Működése:** Visszatér az adott példány nevezőjével.
- **void** `setsz(int a);`
  - **Paraméterek:**
    - \* **int** `a`: az a szám amelyre szeretnénk állítani a már meglévő példány számlálóját.
  - **Működése:** Átrja a private `int` sz értéket.
- **void** `setn(int a);`
  - **Paraméterek:**
    - \* **int** `a`: az a szám amelyre szeretnénk állítani a már meglévő példány nevezőjét.
  - **Működése:** Átrja a private `int` `n` értéket, 0 esetén hibát dob.
- **void** `print()const;`
  - **Működése:** Kiíró függvény, console-ra írja a példányt számláló/nevező alakban.
- **const Rac** `operator+(const Rac& x)const;`
  - **Paraméterek:**
    - \* **const Rac&** `x`: az a racionális szám mellyel össze akarjuk adni a példányt, melyre hívtuk.
  - **Működése:** Visszatér egy új példánnyal, melynek értéke a két szám összege.
- **const Rac&** `operator+=(const Rac& x);`

- **Paraméterek:**
  - \* **const Rac& x:** az a racionális szám mellyel össze akarjuk adni a példányt, melyre hívtuk.
- **Működése:** Visszatér a már meglévő példány összegnek megfelelő módosított értékével.
- **const bool operator==(const Rac& x) const;**
  - **Paraméterek:**
    - \* **const Rac& x:** az a racionális szám mellyel össze akarjuk hasonlítani a példányt, melyre hívtuk.
  - **Működése:** Egyenlőség fennállásától függően visszatér true vagy false értékkel.
- **const Rac operator-(const Rac& x) const;**
  - **Paraméterek:**
    - \* **const Rac& x:** az a racionális szám melyet ki akarjuk vonni a példányból, melyre hívtuk.
  - **Működése:** Visszatér egy új példánnyal, melynek értéke a két szám különbsége.
- **const bool operator!=(const Rac& x) const;**
  - **Paraméterek:**
    - \* **const Rac& x:** az a racionális szám mellyel össze akarjuk hasonlítani a példányt, melyre hívtuk.
  - **Működése:** Nem egyenlőség fennállásától függően visszatér true vagy false értékkel.
- **operator double() const;**
  - **Működése:** Visszatér a racionális szám által reprezentált double értékkel.
- **Rac reciprok();**
  - **Működése:** Visszatér azon racionális szám reciprokával, melyre hívtuk. 0 számláló esetén hibát dob.
- **const Rac operator\*(const Rac& x) const;**
  - **Paraméterek:**
    - \* **const Rac& x:** az a racionális szám mellyel szorozni szeretnénk.
  - **Működése:** Visszatér egy új példánnyal, melynek értéke a két szám szorzata.

- **const Rac operator\*=(const Rac& x);**
  - **Paraméterek:**
    - \* **const Rac& x:** az a racionális szám mellyel szorozni szeretnénk.
  - **Működése:** Visszatér a már meglévő példány szorzatnak megfelelő módosított értékével.
- **const Rac operator/(Rac& x)const;**
  - **Paraméterek:**
    - \* **const Rac& x:** az a racionális szám mellyel osztani szeretnénk.
  - **Működése:** Visszatér egy új példánnyal, melynek értéke a két szám hányadosa.

## 2.2. class Simplex

### 2.2.1. private: (ezeket közvetlenül nem tudja hívni/elérni)

- **vector<vector<Rac>>** A: Együttható mátrixot reprezentáló, Rac példányokat tároló vector-ra mutató vector.
- **vector<vector<Rac>>** PivotT: Eleinte az indulási táblát reprezentáló Rac példányokat tároló vector-ra mutató vector. Már tartalmazza a hozzáadott bázisokat, ezt változtatják a megoldáshoz szükséges függvények.
- **vector<Rac>** b: Eredményvektort reprezentáló Rac példányokat tároló vector.
- **vector<Rac>** Z: Célfüggvény-együtthatókat reprezentáló Rac példányokat tároló vector.
- **vector<int>** egyenlo: Azon sorok indexeit tároló vector, melyeknél egyenlőség feltétel található.
- **vector<int>** nagyobb: Azon sorok indexeit tároló vector, melyeknél nagyobb-egyenlőség feltétel található.
- **vector<int>** mmek: Bázisba kerülhető oszlopvektorok indexeit tartalmazó vector.
- **bool minn:** Alapértelmezetten false, a példány max/min létéről tanúskodik. (minimalizálás: false)
- **double ertek:** Megoldás után az optimum érték double-ben.
- **void korlell();**
  - **Változók:**
    - \* **unsigned int i=0:** ciklusváltozó

- \* **unsigned int** j=0: ciklusváltozó
- \* **bool** nemko=true: nem korlátosságról tanúskodó bool változó
- \* **bool** nincsmo=true: nem megoldhatóságról tanúskodó bool változó
- **Működése:** Oszloponként végigmegy az adott fázis nagy pivottáblájának megengedett bázisain, és ellenőrzi a korlátosság/megoldhatóság feltételeit. Amennyiben nem teljesülnek, hibát dob.
- **void PivotTk();**
  - **Változók:**
    - \* **unsigned int** i=0: ciklusváltozó
    - \* **unsigned int** j=0: ciklusváltozó
    - \* **bool** megengedett=true: A tábla felépítése során jelzi az adott indexről, hogy megengedett bázishoz tartozik-e.
  - **Működése:** Amennyiben nem adtak még meg célfüggvényt, vagy nem egyezik a dimenzió, hibát dob. Ellenkező esetben először feltölti az A mátrix együtthatóival. Ha van nagyobb vagy egyenlő feltétel, azoknak összegét +1 utolsó sorként a kétfázisú szimplex algoritmusnak megfelelően újabb feltételként kezeli. Következőnek hozzáadja a célfüggvény együtthatóit utolsó sorként. A feltételeknek megfelelően ezután a sorvektorok végére a megfelelő 0/1/-1 értékű Rac példányok kombinációjával felépíti az induló bázist. Majd végül minden sor végére az adott eredményvektor értékét illeszti.

```
indulasi tabla:
1 1 1 1 3 0 1 0 0 -1 100
5 3 2 0 0 2 0 1 0 0 650
1 5 1 3 2 1 0 0 1 0 150
6 4 3 1 3 2 0 0 0 -1 750
100 160 250 0 15 30 0 0 0 0 0
```

1. ábra. Példa egy nagy pivottáblára

- **void printpivot();**
  - **Változók:**
    - \* **unsigned int** i=0: ciklusváltozó
    - \* **unsigned int** j=0: ciklusváltozó
  - **Működése:** Kiírja az aktuális PivotTábla értékét. (Alapértelmezetten az induló és leálló táblát íratjuk ki vele. Ha szeretnénk látni a lépéseket, a solve() tagfüggvény megfelelő ciklusából kommenteljük ki a kiíró metódusokat.
- **const unsigned int Pivotelem(const int x)const;**
  - **Paraméterek:**

- \* **const int** x: az oszlop indexe, ahol pivotelemet keresünk
- **Változók:**
  - \* **vector<int>** megengedett: Az adott oszlop megengedett indexeit tartalmazó vector.
  - \* **bool** korl: korlátosságról tanúskodó bool változó
  - \* **bool** nemures: nemürességről tanúskodó bool változó
  - \* **unsigned int** i: ciklusváltozó
  - \* **int** eredmény: a pivotálandó elem sorindexe
- **Működése:** Leellenőrzi a paraméterként kapott oszlopban a megengedettséget, majd a hányadosszabály szerint visszaadja a megfelelő indexet(sort).
- **void** pivotalas(const unsigned int x,const unsigned int y);
  - **Paraméterek:**
    - \* **const unsigned int** x: a sor indexe, ahol a pivotelemünk van
    - \* **const unsigned int** y: az oszlop indexe, ahol a pivotelemünk van
  - **Változók:**
    - \* **unsigned int** i: ciklusváltozó
    - \* **unsigned int** j: ciklusváltozó
    - \* **Rac** konstans: Az adott pivotelem.
  - **Működése:** A pivottáblán elvégzi az adott elem szerinti pivotálást, azaz beviszi az y-ik oszlopvektort a bázisba. Először leosztja a saját sorának minden elemét a pivotelemmel, majd annak megfelelő többszörösét kivontja a felette és alatta található sorvektorokból úgy, hogy az y-ik oszlopban a pivotelem helyén megjelenő 1-en kívül már csak 0-k találhatók.

### 2.2.2. public:

- Simplex();
  - **Működése:** Létrehoz egy teljesen üres feladatot a Simplex osztály példányaként. Alapértelmezetten maximalizálásra van állítva. (minn=false)
- ~Simplex();
  - **Változók:**
    - \* **unsigned int** i: ciklusváltozó
  - **Működése:** Destruktor, felszabadítja a példányosítás során lefoglalt memóriát.
- **void** filebol(const std::string& hely);

- **Paraméterek:**
    - \* **const std::string&** hely: az olvasandó fájl helye
  - **Változók:**
    - \* **std::string** str: buffer
  - **Működése:** Végighalad soronként a megfelelő struktúrájú fájlban, és egyenként hozzáadja a feladathoz megfelelő feltételként. Min/max kezdetű sort célfüggvényként olvassa be.
- **void insert\_cond(const std::string& szov);**
    - **Paraméterek:**
      - \* **const std::string&** szov: az olvasandó sor referenciája
    - **Változók:**
      - \* **int** i=0: ciklusváltozó
      - \* **int** j=0: ciklusváltozó
      - \* **double** szam: adott együttható double értéke
      - \* **int** ht=0: adott együtthatóban hány tizedesjegy található
      - \* **int** elsz=0: előző szóköz helye
      - \* **char\*** ideigl: buffer
      - \* **vector<Rac>** beszur: vector mely végén az A mátrix eleme lesz feltételként
    - **Működése:** A beolvasott sort feldarabolja szóközönként. A tizedesvessző pozíciójától függően átalakítja Rac példánnyá, majd hozzáadja a gyűjtővektorhoz. Közben tárolja, milyen ípusú feltételről van szó és ahhoz igazítja a bázisok megengedettségét. Ha nem megfelelő a vector dimenziója (nem egyezik A-val), hibát dob. Ellenőrzi a felvitt feltétel primál-megengedettségét. (jobb oldal nem negatív)
  - **void insert\_Z(const std::string& szov);**
    - **Paraméterek:**
      - \* **const std::string&** szov: az olvasandó sor referenciája
    - **Változók:**
      - \* **int** i=4: ciklusváltozó
      - \* **int** j=0: ciklusváltozó
      - \* **double** szam: adott együttható double értéke
      - \* **int** ht=0: adott együtthatóban hány tizedesjegy található
      - \* **int** elsz=4: előző szóköz helye
      - \* **char\*** ideigl: buffer
      - \* **vector<Rac>** beszur: vector mely végén az A mátrix eleme lesz feltételként

- **Működése:** A beolvasott sort feldarabolja szóközönként. A tizedesvessző pozíciójától függően átalakítja Rac példánnyá, majd hozzáadja a gyűjtővektorhoz. Közben árolja az optimalizálás milyenségét, ezzel átállítva a feladat megfelelő pramétereit. Ha nem megfelelő a vector dimenziója (nem egyezik A-val), hibát dob. Minimalizálás esetén a az együtthatók ellentettjét tárolja.
- **void Aprint();**
  - **Változók:**
    - \* **int** i=0: ciklusváltozó
    - \* **int** j=0: ciklusváltozó
  - **Működése:** Kiírja consolra az együtthatómátrixot szóközökkel elválasztva.
- **void bprint();**
  - **Változók:**
    - \* **int** i=0: ciklusváltozó
  - **Működése:** Kiírja consolra az eredményvektort szóközökkel elválasztva.
- **void Zprint();**
  - **Változók:**
    - \* **int** i=0: ciklusváltozó
  - **Működése:** Kiírja consolra a célfüggvény együtthatóit szóközökkel elválasztva.
- **void printmeg();**
  - **Változók:**
    - \* **int** i=0: ciklusváltozó
  - **Működése:** Kiírja consolra a megengedett bázisok indexét szóközökkel elválasztva.
- **void solve();**
  - **Változók:**
    - \* **int** i=0: ciklusváltozó
    - \* **int** j=0: ciklusváltozó
    - \* **bool** vanpoz: megfelelő célfüggvényhez tartozó redukált költségeinek előjeléről tanúskodik
    - \* **bool** notsolved=true: feladat nem-megoldottságáról tanúskodik



- **Működése:** Felépíti a nagy pivottáblát az előzőleg felvitt adatokból. Kiírja az indulási táblát, majd attól függően, hogy volt-e  $\geq$  vagy  $=$  feltétel egy, illetve kétfázisú simplex algoritmusba kezd: egyfázisú esetén rögtön az optimalizálandó célfüggvény együtthatóival dolgozik, célja, hogy addig pivotáljon amíg csak nem pozitív redukált költségek maradnak a sorában. Ekkor leáll és megfelelő pozícióban megtalálható az optimum értéke. Ha kétfázisú, először egy induló bázist keres az utolsó előtti sorra futtatott simplex algoritmussal. Ha sikerrel jár, innen folytatva, már megengedett báziscserét alkalmazva keresi az eredeti optimumot, és el is tárolja azt. Ellenkező esetben, nem korlátos célfüggvény vagy ellentmondásos rendszer során hibát dob.

- **const double** geter();

- **Működése:** Visszaadja az optimumértéket a feladat private érték változójából.

### 3. Fejlesztési lehetőségek:

A feltételek megadása során fontos a jobb oldal nem negativitása. A felhasználó szempontjából kényelmesebb lenne, ha a bevitt nem primálmegengedett feltétel esetén a függvény azt átalakítaná megfelelővé. Emellett a Simplex osztályt immáron egy lépés átalakítani egy egészértékű programozási feladatokat megoldó osztállyá. (Pl. Gomory vágások felvételével) A pivottáblák kiírásán még lehet javítani.

### 4. Teszt

```
uj.filebol("./text.txt");
```

```
1 1 0.3 0.5 -1 3 0 = 100
2 5 3 2 0 0 2 = 650
3 1 0.5 1 3 2 1 = 150
4 min 100 160 250 0 15 30
```

2. ábra. Fájlból való bevétel

```
Simplex proba;
proba.insert_cond("1 0.3 0.5 -1 3 0 <= 100");
proba.insert_cond("5 3 2 0 0 2 = 650");
proba.insert_cond("1 0.5 1 3 2 1 <= 150");
proba.insert_Z("max 100 160 250 0 15 30");
proba.solve();
```

3. ábra. Soronként történő bevétel

```

Simplex proba;
proba.insert_cond("1 0.3 0.5 -1 3 0 <= 100");
proba.insert_cond("5 3 2 0 0 2 = 650");
proba.insert_cond("1 0.5 1 3 2 1 <= 150");
proba.insert_Z("max 100 160 250 0 15 30");
proba.solve();
std::cout<<proba.geter();
std::cout<<std::endl;
Simplex uj;
uj.filebol("./text.txt");
std::cout<<std::endl;
uj.Aprint();
uj.Zprint();
uj.bprint();
uj.solve();
std::cout<<uj.geter();
return 0;

```

4. ábra. Teljes tesztelő kód

```

nagy_hazi
indulasi tabla:
1 3/10 1/2 -1 3 0 1 0 0 100
5 3 2 0 0 2 0 1 0 650
1 1/2 1 3 2 1 0 0 1 150
5 3 2 0 0 2 0 0 0 650
100 160 250 0 15 30 0 0 0 0
Bazisba kerulhet: 0 1 2 3 4 5 6 8

Lellasi tabla:
17/40 0 0 -47/20 21/10 -1/2 1 -1/40 -9/20 65/4
3/2 1 0 -3 -2 0 0 1/2 -1 175
1/4 0 1 3/2 3 1 0 -1/4 3/2 125/2
-405/2 0 0 -645 -415 -220 0 -35/2 -215 -43625
43625

1 3/10 1/2 -1 3 0
5 3 2 0 0 2
1 1/2 1 3 2 1
-100 -160 -250 0 -15 -30
100 650 150

indulasi tabla:
1 3/10 1/2 -1 3 0 1 0 0 100
5 3 2 0 0 2 0 1 0 650
1 1/2 1 3 2 1 0 0 1 150
7 19/5 7/2 2 5 3 0 0 0 900
-100 -160 -250 0 -15 -30 0 0 0 0
Bazisba kerulhet: 0 1 2 3 4 5

Lellasi tabla:
1 0 10/11 -2/11 72/11 0 20/11 -3/11 6/11 950/11
0 1 -15/11 -30/11 -130/11 0 -30/11 10/11 -20/11 500/11
0 0 17/22 50/11 15/11 1 -5/11 -2/11 15/11 450/11
0 0 -3895/11 -3500/11 -13315/11 0 -2950/11 1240/11 -2150/11 188500/11
17136.4
Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.

```

5. ábra. Tesztelő által konzolra írt információ