Generikus vector és kupac, alkalmazva rendezésre és dijkstra algoritmusra

Szerkezetek rendelkeznek virtuális függvényekkel a származtatás érdekében. Ezeket V-vel jelölöm.

Vector

- `int` length() V
- resize(size, defaultValue) V
- clear() V
- pushBack(T) V
- T popBack() V
- operator=(Vector) V
- operator={} V
- operator[] V
- operator+(T) V
- operator+(Vector) V
- operator+=(T) V
- operator+=(Vector) V
- Vector()
- Vector(size, defaultValue)
- Vector(Vector)
- Vector({})
- `static` Test()
- kívül
    - operator+(T, Vector)
    - operator<<
        - formátum: "[méret](1.elem, 2.elem, ...)"

BinaryHeap

- clear() V
- insert(T) V
- insert(Vector) V
- T top() V
- T pop() V
- bool empty() V
- int length() V
- BinaryHeap()
- BinaryHeap(Vector)
- operator+(T) V
- operator+(Vector) V
- opartor+=(T) V
- opartor+=(Vector) V
- static Sort(Vector)
- static Test()

- kívül
    - operator(T, BinaryHeap)
    - operator(Vector, BinaryHeap)

Dijkstra

- int length()
- Vector parents
- Vector weights
- printPath(int to) `V`
- Dijkstra(filename, startIndex)

Dijkstra esetében a rejtett `read(filename)` és `printPathRec(int to)` is virtuális a bővíthetőség érdekében.

Az egyes adatszerkezetek ha tartalmaznak `static Test()`-et, akkor annak célja, hogy változatos eseteket, funkcionalitásokat kipróbáljanak és assert-tel leteszteljék azokat.

Ezekhez a szerkezetekhez Exception is tartozik, melyeket az őket tartalmazó namespacebe helyezek (kivétel Dijkstra...), hogy más megegyező funkcionalitású adatszerkezetek is felhasználhassák ezeket.

Feladat szintén egy pl egész számokat tartalmazó fájlt vektorba beolvasni, majd BinaryHeap-pel sortolva kiírni a beépített `operator<<`-ral. Szintén kell egy a Dijkstrát kipróbáló rész, ahol az egyik esetben beolvassa a fájlt (két különböző súlytípusú példa kell), majd kiírja a legrövidebb utakat az összes csúcsba, míg másik esetben detektálja, hogy nem alkalmazható az algoritmus.