Házi Feladat

A feladat megoldása a Program.cs fájl legyen, melyet beadás előtt nevezzen át. A beadandó forrásfájl elnevezése a feladat azonosítója és a saját Neptun kódja legyen, alulvonással elválasztva, nagybetűkkel: **AZONOSÍTÓ_NEPTUNKOD.cs**

A feladattal kapcsolatos további információk az utolsó oldalon találhatók. Ezen ismeretek hiányából adódó reklamációt nem fogadunk el.

Készítsen egy szókereső játékhoz hasonló programot, amely megtalálja a titkos üzenetet!

A szókereső játék célja, hogy egy szövegben (T) megkeressük a megadott szavakat (W), ami alapján meg tudjuk határozni a titkos üzenetet (S). A keresendő szavak a szövegben szerepelhetnek balról jobbra olvasva, de fordított irányban is, mindig összefüggően. A megadott szavaknak több előfordulása is lehet a szövegben, illetve az egyes karakterek előfordulhatnak több szóban is (vagyis a szavak át is fedhetnek).

A szavak megkeresését követően lehetnek olyan betűk, amelyek egyik szóban sem kerültek felhasználásra. A feladat ezeknek a karaktereknek az összegyűjtése balról jobbra haladva, ezzel kapjuk meg a titkos üzenetet, amelyet ki kell írnunk a képernyőre.

Bemenet (Console)

- *1. sor* - a megkeresendő szavak száma (N)
- *következő N sor* - a megkeresendő W szavak
- *(N + 1). sor* - a T szöveg, amelyben a W szavakat kell megkeresni

Kimenet (Console)

- a T szövegben a megmaradt karakterekből előállított S titkos üzenet

Megkötés(ek)

- $1 \leq N \leq 100$
- $1 \leq |W| \leq 50$
- $|W| \leq |S| \leq 1\ 000$
- $T, W \in \{0-9a-zA-Z\}$
- $1 \leq |T| \leq 999$

Példa

Input
```
2
BAL
BOB
LABOBOR
```

Output
```
OR
```

Értelmezés

Az első beolvasott érték alapján kettő szót kell megkeresni a szövegben, amelyek a második, illetve a harmadik beolvasott értékek lesznek: `BAL` és `BOB`. A negyedik beolvasott érték a szöveg, amiben az előző szavakat kell megkeresni: `LABOBOR`.

Az első szót (`BAL`) most jobbról balra irányba olvasva találjuk meg, egy előfordulása van. A második szót (`BOB`) szintén egy előfordulásban találjuk meg, átfedésben az első szóval: `LABOBOR`. A megtalált szavakhoz tartozó karaktereket figyelmen kívül hagyva az `OR` szót kapjuk, ez a titkos üzenet, amit meg kell megjelenítenünk a kimeneten.

Tesztesetek

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1.

Console input
```
2
BAL
BOB
LABOBOR
```

Console output
```
OR
```

2.

Console input
```
2
ALMA
PAPA
ALMAPPAPA
```

Console output
```
P
```

3.

Console input
```
4
AZ
AZTA
ALAKZAT
ALKAT
AlAZTAKLAtaALAKZATt
```

Console output
```
Altat
```

4.

Console input
```
2
3MB3R
M3R
3M3R3MB3R3
```

Console output
```
33
```

5.

Console input
```
7
SR5
6
O6CSUS
C5R
2JY
4IHG
J
tSR5Cn2JYgGHI4wbO6CSUSjJ6
```

Console output
```
tngwbj
```

A fenti tesztesetek nem feltétlenül tartalmazzák az összes lehetséges állapotát a be- és kimenet(ek)nek, így saját tesztekkel is próbálja ki az alkalmazás helyes működését!

Tájékoztató

A feladattal kapcsolatosan általános szabályok:

- A feladat megoldását egy `Console App` részeként kell elkészíteni, "Do not use top-level statements" beállítással, illetve a szükséges "using"-ok megtartása mellett.
- Megoldásaként csak a `Program.cs` forrásfájlt kell beadni, amelynek elnevezése a feladat azonosítója és a szerző saját Neptun kódja legyen alulvonással elválasztva, nagy betűkkel: **AZONOSÍTÓ_NEPTUNKOD.cs**
- A megvalósítás során lehetőség szerint alkalmazza az előadáson és a laboron ismertetett algoritmusokat, figyelembe véve a *Megkötések* pontban definiáltakat. Ezeket leszámítva viszont legyen kreatív a feladat megoldásával kapcsolatban.
- Az alkalmazás elkészítése során minden esetben törekedjen a megfelelő típusok használatára, illetve az igényes (*formázott, felesleges változóktól és utasításoktól mentes*) kód kialakítására, amely magába foglalja az elnevezésekkel kapcsolatos ajánlások betartását is (bővebben).
- **Ne másoljon vagy adja be más megoldását!** Minden ilyen esetben az összes (felépítésben) azonos megoldás duplikátumként lesz megjelölve és a megoldás el lesz utasítva.
- **A határidő után leadott vagy helytelen elnevezésű megoldás, vagy a kiírásnak nem megfelelő megoldás, vagy fordítási hibát tartalmazó, vagy (helyes bemenetet megadva) futásidejű hibával leálló kód nem értékelhető!**
- A feladat leírása az alábbiak szerint épül fel (* - opcionális):
  - *Feladat leírása* - a feladat megfogalmazása
  - *Bemenet* - a bemenettel kapcsolatos információk
  - *Kimenet* - az elvárt kimenettel kapcsolatos információk
  - *Megkötések* - a bemenettel, a kimenettel és az algoritmussal kapcsolatos megkötések, melyek figyelembevétele és betartása kötelező, továbbá az itt megfogalmazott bemeneti korlátoknak a tesztek minden eseteben eleget tesznek, így olyan esetekre nem kell felkészülni, amik itt nincsenek definiálva
  - **Megjegyzések* - további, a feladattal, vagy a megvalósítással kapcsolatos megjegyzések
  - *Példa* - egy példa a feladat megértéséhez
  - *Tesztesetek* - további tesztesetek az algoritmus helyes működésének teszteléséhez, mely nem feltétlenül tartalmazza az összes lehetséges állapotát a be- és kimenet(ek)nek
- **Minden eseteben pontosan azt írja ki és olvassa be az alkalmazás, amit a feladat megkövetel, mivel a megoldás kiértékelése automatikusan történik!** Így például, ha az alkalmazás azzal indul, hogy kiírja a konzolra a `"Kérem a számot:"` üzenetet, akkor a kiértékelés sikertelen lesz, a megoldás hibásnak lesz megjelölve, ugyanis egy számot kellett volna beolvasni a kiírás helyett.
- A kiértékelés során csak a *Megkötések* pont szerinti helyes bemenettel lesz tesztelve az alkalmazás, a "tartományokon" kívüli értéket nem kell lekezelnie az alkalmazásnak.
- Elősegítve a fejlesztést, a beadott megoldás utolsó utasításaként szerepelhet egyetlen `Console.ReadLine()` metódushívás.
- Az automatikus kiértékelés négy részből áll:
  - Unit Test-ek - az alkalmazás futásidejű működésének vizsgálatára
  - Szintaktikai ellenőrzés - az alkalmazás felépítésének vizsgálatára
  - Tervezési irányelvek ellenőrzése - az alkalmazás "kinézetének" vizsgálatára
  - Duplikációk keresése - az azonos megoldások kiszűrésére
  - Metrikák meghatározása - tájékoztató jelleggel
- A kiértékelések eredményéből egy HTML report generálódik, melyet minden hallgató megismerhet.
- A feladat megoldásának minden esetben fordíthatónak és futtathatónak kell lennie ***C# 10*** és ***.NET 6*** keretrendszer használatával. Ettől függetlenül az elkészítés során használható egyéb változata a .NET keretrendszernek és a C# nyelvnek, azonban leadás előtt győződjön meg róla, hogy a megoldása kompatibilis a *.NET 6* és *C# 10* verzióval.

- A keretrendszer mellett további általános, nyelvi elemekkel való megkötés, melyek a házi feladatok során nem használhatók a megoldásában:
  - LINQ: `System.Linq` - *all query expressions within the namespace*
  - Attributes
  - Collections: `ArrayList`, `BitArray`, `DictionaryEntry`, `Hashtable`, `Queue`, `SortedList`, `Stack`
  - Keywords:
    - Modifiers: `abstract`, `async`, `event`, `external`, `in` - *generic modifier*, `new` - *member modifier*, `out` - *generic modifier*, `sealed`, `unsafe`, `virtual`, `volatile`
    - Statement: `break` - *in a loop*, `continue`, `goto`, `throw`, `try-catch-finally`, `checked`, `unchecked`, `fixed`, `lock`, `yield`
    - Namespace: `extern alias`
    - Generic type constraint: `new`, `where`
    - Access: `base`
    - Contextual: `add`, `partial` - *type, method*, `remove`, `required`, `when` - *filter condition*,
    - Query: `from`, `where`, `select`, `group`, `into`, `orderby`, `join`, `let`, `in`, `out`, `equals`, `by`, `ascending`, `descending`
  - Operators and Expressions:
    - Null-conditional operators: `?.` - *null conditional member access*, `?[]` - *null conditional element access*
    - User-defined conversion operators: `implicit`, `explicit`
    - Pointer: `*` - *pointer*, `&` - *address-of*, `*` - *pointer indirection*, `->` - *pointer member access*, `[]` - *pointer element access*, `+`, `-`, `++`, `--` - *pointer arithmetic operators*
    - Assignment: `ref`
    - Lambda: `=>` - *expression, statement*
    - Others: `!` - *null forgiving*, `??` - *null coalescing*, `??=` - *null coalescing assignment*, `::` - *namespace alias qualifier*, `await`, `default` - *operator, literal*, `delegate`, `sizeof`, `stackalloc`, `with` - *expression*, `operator`
  - Types: `dynamic`, `interface`, `object`, `Object`, `var`, `struct`, `nullable`, `pointer`, `record`, `Tuple`, `Func<T>`, `Action<T>`, `Expression<T>`, `Nullable<T>`, `Span<T>`
  - Preprocessor directives: `#nullable`, `#if`, `#elif`, `#else`, `#endif`, `#define`, `#undefine`, `#undef`, `#error`, `#warning`, `#line`, `#pragma`
- Névterek, melyek kizárólagosan importálhatók a megoldásban (minden további névtér import törlésre kerül, illetve, ha az alábbiak közül valamelyik hiányzik, akkor hozzáadásra kerül a megoldáshoz):
  - `System`
  - `System.Collections.Generic`
  - `System.IO`
  - `System.Threading`