



A feladat megoldását a Program.cs fájlba készítse el, melyet beadás előtt nevezzen át. A beadandó forrásfájl elnevezése a feladat azonosítója és a saját neptunkódja legyen alulvonással elválasztva, nagybetűkkel: **AZONOSÍTÓ_NEPTUNKOD.cs**

A feladattal kapcsolatos további információk az utolsó oldalon találhatóak (ezen ismeretek hiányából adódó reklamációt nem fogadunk el!).

A klasszikus dominóban két játékos felváltva építi a sort, azonban létezik egyszemélyes változata is a játéknak, amikor a felhasználó a rendelkezésre álló dominókból próbálja a leghosszabb sort létrehozni úgy, hogy minden dominót felhasznál.

A feladat, hogy megmondja, hogy a rendelkezésre álló (N) dominókból lehetséges-e olyan útvonal létrehozása ("Y") vagy sem ("N"), mely nem tartalmaz elágazást és összefüggő.

Bemenet (Console)

- *első sor* - a dominók (N) száma
- *további N sor* - a dominón található A és B szám | karakterrel elválasztva

Kimenet (Console)

- a Console-on megjelenítve vagy az "Y" (ha lehetséges létrehozni az útvonalat az összes dominót felhasználva), vagy az "N" karakter (ha nem)

Megkötés(ek)

- $1 \leq N \leq 100$
- $0 \leq A, B \leq 6$

Megjegyzés

- a dominók szükség szerint megfordíthatók, illetve átrendezhetők

Példa

| Console input | Console output |
|---------------------|----------------|
| 2 1 6 6 3 | Y |

Értelmezés

Az első sor alapján ($N=2$) darab dominót kell megpróbálnunk sorba lerakni. Az első dominó az 1 | 6, a másodikon a 6 | 3 számok találhatók.

Ezeket egymás mellé tudjuk helyezni, így lehetséges olyan sort létrehozni, amiben minden dominót felhasználtunk, vagyis az Y karaktert kell megjeleníteni a felhasználónak.

Tesztesetek

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1.

Console input

2
1 | 6
6 | 3

Console output

Y

2.

Console input

2
1 | 6
3 | 6

Console output

Y

3.

Console input

3
1 | 1
1 | 1

Console output

Y



4.

Console input

```
3
1 | 1
2 | 2
3 | 3
```

Console output

N

5.

Console input

```
5
0 | 1
1 | 2
2 | 3
3 | 4
4 | 5
```

Console output

Y

6.

Console input

```
3
0 | 1
1 | 2
3 | 4
```

Console output

N

7.

Console input

```
20
0 | 1
2 | 3
5 | 6
1 | 2
6 | 4
5 | 0
3 | 4
4 | 2
2 | 1
4 | 5
1 | 3
3 | 5
0 | 5
5 | 2
0 | 3
3 | 3
2 | 6
6 | 1
1 | 4
4 | 0
```

Console output

Y

Tájékoztató

A feladattal kapcsolatosan általános szabályok:

- A feladat megoldását egy Console Application részeként kell elkészíteni.
- A feladat megoldásaként beadni vagy a betömörített solution mappa egészét vagy a Program.cs forrásfájlt kell (hogy pontosan melyiket, azt minden feladat külön definiálja), melynek elnevezése a feladat azonosítója és a saját neptunkódja legyen alulvonással elválasztva, nagybetűkkel: **AZONOSÍTÓ_NEPTUNKOD**[.zip|.cs]
- A megvalósítás során lehetőség szerint alkalmazza az előadáson és a laboron ismertetett programozási tételeket és egyéb 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, utasításoktól mentes*) kód kialakítására, mely 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.
- **Idő 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 esetében 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 esetben 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.
- A kiértékelés automatikusan történik, így különösen fontos a megfelelő alkalmazás elkészítése, ugyanis amennyiben nem a leírtaknak megfelelően készül el a megoldás úgy kiértékelése sikertelen lesz, a megoldás pedig hibás.
- Az automatikus kiértékelés négy részből áll:
 - Unit Test-ek - az alkalmazás futásidejű működésének vizsgálatára
 - Szintaktikai ellenőrzés - az alkalmazás felépítésének vizsgálatára
 - Duplikációk keresése - az azonos megoldások kiszűrésére
 - Metrikák meghatározása - tájékoztató jelleggel
- A kiértékelések eredményéből egy HTML report generálódik, melyet minden hallgató megismerhet.
- A leadott megoldással kapcsolatos minimális elvárás:
 - Nem tartalmazhat fordítás idejű figyelmeztetést (`Solution contains 0 compile time warning(s)`).
 - Nem tartalmazhat fordítási hibát (`Solution contains 0 compile time error(s)`).
 - Minden szintaktikai tesztet teljesít (`0 test warning, 0 test failed`).
 - Minden unit test-et teljesít (`0 test failed, 0 test warning, 0 test was not run`).



- A feladat megoldásához minden esetben elegendő a **.NET Framework 4.7.2**, illetve a **C# 7.3**, azonban megoldását elkészítheti **.NET 5**-öt, illetve a **C# 9**-et használva is, viszont a nyelv újjításait nem használhatja. További általános, nyelvi elemekkel való megkötés, melyet a házi feladatok során nem használhat a megoldásában (*a felsorolás változásának jogát fenntartjuk, a mindig aktuális állapotot a report HTML fogja tartalmazni*):
 - **Methods:** `Array.Sort`, `Array.Reverse`, `Console.ReadKey`, `Environment.Exit`
 - **LINQ:** `System.Linq`
 - **Attributes**
 - **Collections:** `ArrayList`, `BitArray`, `DictionaryEntry`, `Hashtable`, `Queue`, `SortedList`, `Stack`
 - **Generic collections:** `Dictionary<K,V>`, `HashSet<T>`, `List<T>`, `SortedList<T>`, `Stack<T>`, `Queue<T>`
 - **Keywords:**
 - **Modifiers:** `protected`, `internal`, `abstract`, `async`, `event`, `external`, `in`, `out`, `sealed`, `unsafe`, `virtual`, `volatile`
 - **Method parameters:** `params`, `in`, `out`
 - **Generic type constraint:** `where`
 - **Access:** `base`
 - **Contextual:** `partial`, `when`, `add`, `remove`, `init`
 - **Statement:** `checked`, `unchecked`, `try-catch-finally`, `throw`, `fixed`, `foreach`, `continue`, `goto`, `yield`, `lock`, `break` - *in loop*
 - **Operator and Expression:**
 - **Member access:** `^` - *index from end*, `..` - *range*
 - **Type-testing:** `is`, `as`, `typeof`
 - **Conversion:** `implicit`, `explicit`
 - **Pointer:** `*` - *pointer*, `&` - *address-of*, `*` - *pointer indirection*, `->` - *member access*
 - **Lambda:** `=>` - *expression, statement*
 - **Others:** `?:` - *ternary*, `!` - *null forgiving*, `?.` - *null conditional member access*, `?[]` - *null conditional element access*, `??` - *null coalescing*, `??=` - *null coalescing assignment*, `::` - *namespace alias qualifier*, `await`, `default` - *operator, literal*, `delegate`, `is` - *pattern matching*, `nameof`, `sizeof`, `stackalloc`, `switch`, `with` - *expression, operator*
 - **Types:** `dynamic`, `interface`, `object`, `Object`, `var`, `struct`, `nullable`, `pointer`, `record`, `Tuple`, `Func<T>`, `Action<T>`,