Házi Feladat

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 találhatók (ezen ismeretek hiányából adódó reklamációt nem fogadunk el!).

Az egyszálú processzorok működésük során legfeljebb egy feladatot tudnak feldolgozni, de hogy pontosan melyiket, az a következőképpen alakul:

- ha a CPU *idle* állapotban van és nincsenek feldolgozható feladatok, akkor *idle* állpotban marad,
- ha a CPU *idle* állapotban van és vannak feldolgozásra váró feladatok, akkor a CPU a legrövidebb feldolgozási idővel rendelkezőt választja (ha több ilyen feladat is volna, akkor a legkisebb indexű feladatot választja),
- egy feladat elindítása után a CPU megszakítás nélkül dolgozza fel a teljes feladatot,
- a CPU egy feladat befejezését követően ugyanabban az időpillanatban elindíthatja egy másik feladat feldolgozását.

A CPU által feldolgozandó feladatokról (T) tudjuk, hogy melyik időpillanattól lehetséges a feldolgozásuk (T_e) és, hogy mennyi ideig tart azt feldolgozni (T_p).

A cél annak meghatározása, hogy a CPU milyen sorrendben dolgozza fel az N feladatot.

Bemenet (Console)

- első sor, a feldolgozandó feladatok száma (N)
- a következő N sor, az `i.` feladathoz tartozó rendelkezésre állási és feldolgozási idő páros vesszővel elválasztva (T_e, T_p)

Kimenet (Console)

- a T feladatok indexei a feldolgozási sorrendjük szerint rendezve

Megkötés(ek)

- $1 \leq N \leq 10^5$
- $1 \leq T_e, T_p \leq 10^9$

Példa

Console input

```
4
1,2
2,4
3,2
4,1
```

Console output

```
0,2,3,1
```

Értelmezés

A CPU az alábbiak szerint dolgozza fel a feladatokat:

- *Time=1, Tasks:{0}* - Az 1. időpillanatban a 0. feladat elérhető.
- *Time=1, Tasks:{}* - Szintén az 1. időpillanatban, mivel a CPU idle állapotban van, elkezdi feldolgozni a 0. feladatot.
- *Time=2, Tasks:{1}* - A 2. időpillanatban az 1. feladat elérhetővé válik.
- *Time=3, Tasks:{1, 2}* - A 3. időpillanatban a 2. feladat elérhetővé válik.
- *Time=3, Tasks:{1}* - Szintén a 2. időpillanatban a CPU befejezi a 0. feladat feldolgozását és elkezdi a rendelkezésre álló feladatok közül a 2. feladat feldolgozását, mivel annak feldolgozási ideje rövidebb.
- *Time=4, Tasks:{1, 3}* - A 4. időpillanatban a 3. feladat is elérhetővé válik.
- *Time=5, Tasks:{1}* - Az 5. időpillanatban a CPU végez a 2. feladattal és elkezdi a 3. felad feldolgozását.
- *Time=6, Tasks:{}* - A 6. időpillanatban végez a 3. feladat feldolgozásával és elkezdi az utolsó feladat feldolgozását.
- *Time=10, Tasks:{}* - A 10. időpillanatban a CPU végez az 1. feladat feldolgozásával és idle állapotba kerül.

A feladatok feldolgozásának sorrendje tehát: $0, 2, 3, 1$.

Tesztesetek

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1.

Console input
```
4
1,2
2,4
3,2
4,1
```

Console output
```
0,2,3,1
```

2.

Console input
```
5
7,10
7,12
7,5
7,4
7,2
```

Console output
```
4,3,2,0,1
```

3.

Console input
```
2
7,84
3,1
```

Console output
```
1,0
```

4.

Console input
```
7
1,10
2,3
5,6
15,5
16,10
18,4
20,1
```

Console output
```
0,1,2,6,5,3,4
```

A CPU az alábbiak szerint dolgozza fel az 4. tesztben látható feladatokat:

- *Time=1, Tasks:{0}* - Az 1. időpillanatban a 0. feladat elérhető, amit a CPU elkezd feldolgozni.
- *Time=2, Tasks:{1}* - A 2. időpillanatban az 1. feladat elérhetővé válik.
- *Time=5, Tasks:{1, 2}* - A 5. időpillanatban a 2. feladat elérhetővé válik.
- *Time=11, Tasks:{2}* - A 11. időpillanatban a CPU befejezi a 0. feladat feldolgozását és elkezdi a rendelkezésre álló feladatok közül az 1. feladat feldolgozását, mivel annak feldolgozási ideje rövidebb.
- *Time=14, Tasks:{}* - A 14. időpillanatban a CPU befejezi a 1. feladat feldolgozását és elkezdi a 2. feladat feldolgozását.
- *Time=15, Tasks:{3}* - A 15. időpillanatban a 3. feladat elérhetővé válik.
- *Time=16, Tasks:{3, 4}* - A 16. időpillanatban a 4. feladat elérhetővé válik.
- *Time=18, Tasks:{3, 4, 5}* - A 18. időpillanatban a 5. feladat elérhetővé válik.
- *Time=20, Tasks:{3, 4, 5, 6}* - A 20. időpillanatban a 6. feladat elérhetővé válik.
- *Time=20, Tasks:{3, 4, 5}* - Szintén a 20. időpillanatban a CPU befejezi a 2. feladat feldolgozását és elkezdi a rendelkezésre álló feladatok közül a 6. feladat feldolgozását, mivel annak feldolgozási ideje a legrövidebb.
- *Time=21, Tasks:{3, 4}* - A 21. időpillanatban a CPU befejeze a 6. feladat feldolgozását és elkezdi a rendelkezésre álló feladatok közül az 5. feladat feldolgozását, mivel annak feldolgozási ideje a legrövidebb.
- *Time=25, Tasks:{4}* - A 25. időpillanatban a CPU befejeze az 5. feladat feldolgozását és elkezdi a rendelkezésre álló feladatok közül a 3. feladat feldolgozását, mivel annak feldolgozási ideje rövidebb.
- *Time=30, Tasks:{}* - A 30. időpillanatban végez a 3. feladat feldolgozásával és elkezdi az utolsó feladat feldolgozását.
- *Time=40, Tasks:{}* - A 40. időpillanatban a CPU végez az 4. feladat feldolgozásával és idle állapotba kerül.

A feladatok feldolgozásának sorrendje tehát: $0, 1, 2, 6, 5, 3, 4$.

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 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 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.
- 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>`,