**489 – Hangman Judge:**

A sokak által ismert „Akasztófa” játék. Van egy adott, ismeretlen szó, amit úgy kell kitalálnunk, hogy betűket mondunk. Ha a betű szerepel az adott szóban, akkor minden előfordulását beírjuk, hogy pontosan hol helyezkedik el a szóban. Azonban ha olyan betűt mondunk amely nem szerepel, akkor behúznak nekünk 1 hibát (stroke). 7 hibánál elveszítjük a játékot, azonban ha a 7 hiba behúzása előtt kitaláljuk a szót, akkor mi nyertünk. A szó és a tippelt betűk beolvasása után inicializálunk egy letters nevű, 26 méretű tömböt, amiben az angol ABC betű előfordulásait fogjuk jelölni, ezzel együtt számolva azt, hogy hány egyedi betű van a szóban (count változó). A solve függvényben végigjárjuk a beolvasott guess stringünket, amiben a tippelt betűk vannak. Itt 2 fajta lehetséges eset van.

**1.eset:** Ha az adott betű szerepel a szóban (a letters tömbben 1 érték szerepel), akkor ezt az értéket átírjuk 2-re (jelezve ezzel azt, hogy ezt a karaktert már megtaláltuk, így ha újra azt a karaktert olvasnánk, akkor nem kapunk miatta hibát), majd csökkentjük a count változót. Ha a count változó eléri a 0-t, az azt jelenti, hogy a szóban található összes egyedi betűt megtaláltuk, azaz nyertünk.

**2.eset:** Ha az adott betű nem szerepel a szóban (a letters tömbben 0 érték szerepel), akkor ezt az értéket átírjuk -1-re (jelezve ezzel azt, hogy a karaktert már betippeltük, és nem volt jó, így ha újra azt a karaktert olvasnánk, nem kapunk miatta hibát), majd növeljük a stroke változót. Ha a stroke változó elérti a 7-et, azaz 7 rossz tippünk volt, akkor veszítettünk.

Ha sem a count nem éri el a 0-t, se a stroke a 7-et, de elfogynak a tippelt betűk, akkor a játéknak nem lett vége, időben feladtuk.

A print függvény megkapja a jelenlegi kör sorszámát és a solve függvény visszatérési értékét (0, 1 vagy 2), és ez alapján kiírja hogy Nyertünk, Vesztettünk vagy Feladtuk a játékot.

**10205 – Stack ’em Up:**

A feladat során egy kaszinóban dolgozó osztót kell megfigyelnünk, akinek n darab számú „trükkös keverése” van. Ez annyit jelent, hogy minden egyes ilyen keverése során tudja, hogy keveri a kártyákat, így bármilyen sorrendet előállíthat a pakliban. A tesztesetek beolvasása után megkapjuk az osztó trükkös keveréseinek számát (n darab), majd pontosan ennyi keverés leírását. Ezek után véletlen számú keveréseket hajtunk végre, majd a végén kiírjuk a kártyák sorrendjét. Eredetileg a kártyák minden egyes tesztesetnél sorba vannak rakva (2-től az Ászig, Treff – Káró – Kőr – Pikk sorrendben).

A keverések leírása a következő: megkapunk 52 darab számot, amik a pakliban lévő jelenleg elhelyezkedő kártyák új helyét írja le a keverés után.

*Pl.: 2 1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 52 51. – A keverés azt jelenti, hogy az eredetileg 1. helyen lévő kártya a 2. helyre került a keverés után, az eredetileg 2. az 1. helyre, az eredetileg 51. az 52. és az eredetileg 52. helyen lévő kártya az 51. helyre került.*

A feladat megoldásához 2 darab 52 méretű tömböt tartunk fent. Ezeket először inicializáljuk az init eljárásban, azaz 1-től 52-ig feltöltjük őket (ezek a kártyák eredeti helyei). A solve eljárásban addig olvassuk be az elvégenzdő keveréseket, amíg vannak, majd meghívjuk a shuffle eljárást, a következőt csinálja: a tömböket olyan módon fogjuk használni, hogy mindig lesz 1 jelenlegi, kijelölt tömbünk (a set változó által mutatott tömb), amiből az elemeket áttöltjük a másikba a keverés leírása alapján, majd az így feltöltött tömb lesz a jelenlegi és a következő keverés során innen töltjük át a másikba az elemeket. Ez megy egészen addig, amíg van keverésünk. Ha elfogytak a keverések, akkor a jelenlegi tömbünkben lévő értékek alapján tudjuk, hogy adott helyen milyen kártya van, amit a print eljárás segítségével kiíratunk.

**10409 – Die Game:**

A forgatás darabszámának beolvasása után beállítjuk a kezdőállapotot (felül az 1-es, északon a 2-es, nyugaton pedig a 3-as szám látható a dobókockán). Ezután a solve függvénynek paraméterként átadva a darabszámot, forgatásokat hajtunk végre a dobókockán. A dobókockák szabályai miatt (2 szemben lévő oldal összege mindig 7), elég a 3 kezdőállapot (felül, észak, nyugat) nyilván tartanunk, mivel a velük szemben lévő oldalak mindig kiszámolhatóak. Az észak-dél irányú forgatás nem befolyásolja a nyugati oldal értékét, ezzel párhuzamosan a nyugat-kelet irányú forgatás sem változtatja meg az északi oldal értékét, így mindig csak 2 oldal állapotát kell frissíteni. A 2 féle forgatás közül az egyik forgatáskor mindig az kerül a felső oldalra, amivel meghívtuk a függvényt, az ő helyére pedig az előző felül lévő érték 7-től való távolsága (rotate\_b), míg a másikban a meghívott érték ellentéte kerül felülre, az ő helyére pedig az előző felül lévő érték (rotate\_a). Végül kiírjuk a forgatások után felül maradt értéket.

**10646 – What is the Card?:**

A feladat megoldása történhet kétféleképpen is:

A megoldás: végigszámolunk mindent ahogyan a feladat is írja. Beolvassuk az 52 kártyát, majd miután „levettünk” 25 kártyát a pakli tetejéről, a 27. kártyától kezdve végrehajtjuk a feladatban leírt lépéssorozatot háromszor. 1 lépés során (ahol X a kártya értéke) leveszünk 10 - X lapot az asztalon lévő 27 lap közül, plusz még az éppen nézett kártyánkat. Azonban a 3 lépést követően (miután a megmaradt pakli tetejére raktuk az elején levett 25 lapot), át kell „ugranunk” annyi lapot, amennyi a 3 megnézett kártyánk értékének az összege volt. Ez annyit jelent, hogy 1 ilyen lépés során 10 – X + 1 + X (= 11) kártyát kell figyelmen kívül hagynunk. Azaz a 3 lépés után a 33. kártya lesz a kiválasztott kártyánk.

B megoldás: miután beolvastuk az 52 kártyánkat, az előző megoldásban leírt gondolatmenetet végigkövetve egyszerűen kiírjuk a 33. beolvasott kártyalapot.

**10849 – Move the Bishop:**

A feladat egy, a sakkból jól ismert játékfigura, a futó lépéseit használja fel. Adott egy N x N méretű sakktábla, amin megkapjuk a futó sor- és oszlopszámát és egy célterület sor- és oszlopszámát. A feladat az, hogy meg kell állapítani, hogy a futó hány lépésből tud eljutni a célterületre – ha el tud.

A tesztesetek számának, majd a tábla adatainak beolvasása után minden egyes futó - célterület párra meghívjuk a solve függvényt. Ez paraméterként megkapja a futó és célterület sorának távolságát és a futó és célterület oszlopának távolságát. Ha mindkét érték 0, akkor ugyanott van a futó, ahol a célterületünk, ezért 0 lépés szükséges. Ha a 2 különbség ugyanolyan paritással rendelkezik (mindkettő páros vagy páratlan), és a 2 különbség megegyezik, akkor 1 lépésből eljuthatunk a célba (ez annak köszönhető, hogy a futó 1 lépés során ugyanannyival változtatja meg a sorának és oszlopának is az értékét). Ha a 2 különbség nem egyezik meg, akkor 2 lépésből elléphetünk a célterületre, mivel 1 lépésből el tudunk lépni egy olyan területre, ahonnan 1 lépésből megoldható (előző eset). Azonban ha a paritás nem egyezik meg (különböző színű területen vannak), akkor soha nem tudjuk elérni a célterületet, mivel különböző színű területen vannak, a futó pedig csak 1 színen léphet.

Ezt követően a print eljárásunk kiírja az értéket, amivel a solve függvény visszatért. Ha nem -1 ez az érték, akkor kiírja (ez a szükséges lépések száma), egyébként pedig azt, hogy nincs lépés amivel eljuthatnánk a célterületre.

**11309 – Counting Chaos:**

A feladatban egy séfet kell segítenünk abban, hogy egy adott pillanattól számítva, mi lesz a leghamarabb soron következő palindróm idő, amikor ki tudja venni a tortáját a sütőből. Palindróm idő alatt azt az időt értjük, ami balról-jobbra olvasva ugyanannyi, mint jobbról-balra (a vezető nullákat nem számolva, tehát a 00:01 palindróm idő, mert 1 visszafelé olvasva ugyanúgy 1).

Először kigeneráljuk az összes palindróm időt (ahol az óra 0-23-ig, a percek 0-59-ig tartanak) a gen eljárás segítségével, ami ezekkel a számokkal meghívja az is\_palindrom függvényt. Az is\_palindrome függvény stringgé alakítja a kapott számot, majd a string elejétől és végétől elindulva vizsgálja, hogy a 2 karakter megegyezik-e, egészen addig amíg az elejétől indított változó nagyobb nem lesz a végéről indított változótól. Ezeket a palindrómokat a globális palindrome tömbben tároljuk.

Ezután beolvassuk a tesztesetek számát, és minden egyes tesztesetnél beolvasunk egy időpontot HH:MM formátumban. Ezt a stringet tokenizáljuk a ’:’ karakter mentén, majd az az így kapott értékeket átalakítjuk számmá. Így megkapjuk az órát és a percet. A solve függvényt meghívjuk a (100 \* óra) + perc értékkel, amiben egy while ciklus megy egészen addig, amíg nem talál palindrómot a palindrome tömbben, vagy el nem éri a 2360-at (ez már egy olyan érték ami nem szerepelhet az órán). Ezután ha a megtalált palindrómunk nem a 2360 lesz, akkor visszaadjuk azt a számot, egyébként pedig 0-t (mivel ha elértük a 2360-at úgy hogy nem találtunk palindrómot, az azt jelenti hogy az óra 00:00-ra át fog váltani, ami viszont pont a legelső palindróm időpont lesz a napban, így ez lesz a megoldásunk). Ezután csak ki kell írnunk formázottan a kapott számunk óraszámát (szám / 100) és percszámát (szám % 100).

**11459 – Snakes and Ladders:**

A „Kígyók és létrák” egy népszerű társasjáték, amit egy 10 x 10-es négyzet alapú táblán játszanak, amin a mezők számozása 0-től 100-ig tart (sorfolytonosan, a bal alsó sarokból indulva, a sorok irányát folyamatosan változtatva). A táblán vannak még elhelyezve kígyók és létrák is, amelyek mindig 2 mezőt kötnek össze. Ha valaki egy olyan mezőre lép, ami egy létra aljánál van, akkor felmehet a létrán arra a mezőre, ami a létra végén van. Azonban ha egy olyan mezőre lép, amin egy kígyó szája van, akkor az adott játékos „lecsúszik a kígyón” arra a mezőre, ami a kígyó farkánál van. Ezek azonban nem köthetők össze, azaz ha a játékos egy létra/kígyó után egy olyan mezőre kerül ahol egy másik létra/kígyó van, azon nem léphet feljebb/lejjebb. A játékosok felváltva egy kockával dobnak és lépnek. Ha egy játékos rálép a 100-as mezőre, vagy olyat dob amivel a 100-as mezőn túllépne, nyer.

A players globális tömbben a játékosok jelenlegi tartózkodási helyét tároljuk, a step tömbben pedig azt, hogy ha adott mezőre lépünk, akkor onnan hova kerülünk.

A tesztesetek számának beolvasása után beolvasunk 3 újabb számot:

- a: a játékosok számát

- b: a kígyók/létrák számát

-c: a dobások számát.

A 3 változó beolvasása után inicializáljuk a kezdőállapotot az init eljárással. A győztest beállítjuk 0-ra (még nincs győztesünk), a players tömbben a játékosok kezdőállapotát beállítjuk az 1-es mezőre, majd a táblán beállítjuk a mezőket (ha rálépnük az hova visz el, ez eredetileg mindegyiknél önmaga). Ezután beolvasunk b darab sort, ami egy kígyót/létrát ír le olyan módon, hogy az első szám egy létra alja vagy kígyó szája, a második pedig egy létra teteje vagy egy kígyó farka, tehát ha az 1. számú mezőre lépünk, az elvisz minket a 2. számú mezőre -> a lépések tömbben beállítjuk az 1. számú indexen lévő értéket a 2. számra.

Ezt követi c darab dobás, ami az adott játékosok dobását írja le. Beolvassuk a dobott értéket, és ha még nincsen nyertesünk, végrehajtjuk a lépést. A jelenlegi pozíciónkhoz hozzáadjuk a dobott értéket, és ha ez kisebb mint 100, akkor beállítjuk a játékosnak azt az értéket, ahova a kapott érték mutat a mezőn, egyébként pedig 100-ra (ezt nem léphetjük túl). Ha a játékos a 100-as mezőre lépett, akkor megvan a nyertesünk és több dobás így már nem is fog végrehajtódni. A dobások végeztével pedig kiírjuk az összes játékos tartózkodási helyét.

**11678 – Cards’ Exchange:**

Alice és Betty Pokémon kártyákat gyűjtenek, és ezeket akarják cserélgetni egymással, de mindketten csak olyan kártyát akarnak kapni, ami nekik még nincs meg. Ez alapján kell kiszámolnunk a maximális lapok számát amit cserélni tudnak.

A bemenetek mindig 3 sorból állnak: az első sorban megkapjuk, hogy hány darab kártyájuk van (A és B változó), majd az azt követő sorban A darab lap (Alice lapjai), majd pedig B darab lap (Betty lapjai). Ha A és B is 0, vége a bemeneteknek.

Az A és B beolvasása után inicializáljuk a kezdőállapotot, azaz a tömböket nullázzuk (Adeck, Bdeck - ezek 100000 méretűek, mivel ez a legnagyobb számú kártya amit kaphatnak) és a 2 változót is kinullázzuk, amivel majd azt számoljuk, hogy a lányok külön-külön hány kártyát tudnak cserélni.

Ezt követően az input eljárásban beolvasunk A darab számot, és az Adeck tömbben ezen helyen lévő értéket növeljük. Majd beolvasunk B darab számot, és a Bdeck tömbben ezen helyen lévő értéket növeljük. Ezzel mindkét lány esetében nyilván tudjuk tartani milyen kártyából, mennyi van neki.

A beolvasás után meghívjuk a solve függvényt, ami végigmegy a 2 lány tömbjén. Ha talál egy olyan lapot, ami Alice-nak nincs meg, de Bettynek igen, növeli az Anum változót, ha viszont egy olyan lapot talál, ami Alice-nak megvan, de Bettynek nincs, növeli a Bnum változót. Ezzel számoljuk azt, hogy hány olyan lap van, ami nekik kéne, mert nekik nincs meg. Ezután pedig visszaadjuk a 2 szám közül a kisebb értéket (ha egyenlő akkor mindegy melyiket), mivel ez a maximum szám, amit mindketten cserélni tudnak és ezt kiírjuk a teszteset végén.

**12239 – Bingo!:**

A feladatban megkapunk 2 változót N-t és B-t. B darab golyónk van egy dobozban, amik 0-tól N-ig vannak számozva. Meg kell állapítanunk, hogy 0-tól N-ig az összes számot meg lehet-e kapni úgy, hogy kihúzunk egy golyót, visszarakjuk, majd húzunk egy 2. golyót és a 2 golyó különbségének abszolút értékét vesszük. Az összes tesztesetről azt feltételezzük az elején, hogy jó lesz (good = 1).

Az első sorban beolvassuk N-t és B-t, majd a következő sorban B darab számot. Ezután beolvasunk B darab számot a t tömbünkbe. Ezután minden elemet mindennel megvizsgálunk (összes lehetséges kihúzás), majd a diff tömbben ezen helyen lévő értéket növeljük.

Ezután bejárjuk a diff tömbünket és bárhol olyan értéket találtunk, ami 0 (nem tudtuk azt az értéket kikombinálni a kihúzásokból), akkor a good változónkat 0-ra állítjuk, azaz ez a teszteset nem adott jó eredményt. Ezek alapján pedig ha a good 1 volt, kiírunk egy Y-t, egyébként pedig egy N-t.

A beolvasások mindaddig történnek, amig N nem 0 és B nem 0.