B-I-1 Pokémoni

Odpovedzme si najskôr na otázku, ktorá sa objavila aj v návodných úlohách. Oplatí sa Peťovi odísť z pokéstopu a mať pri sebe menej ako m pokélôpt? Samozrejme, že nie. Veď čím viac prázdnych pokélôpt nesie, tým viac pokémonov môže chytiť. A ak sa mu aj stane, že niektorú pokéloptu nevyužije, tak sa nič nedeje – nosil ju síce zbytočne, ale to mu vadiť nemusí.
Povedané formálne: k ľubovoľnému riešeniu, v ktorom Peťo opustí pokéstop s k pokéloptami (pričom $k < m$) vieme vytvoriť rovnako dobré riešenie, v ktorom z tohto pokéstopu odíde s m loptami: jednoducho tých $m - k$ pokélôpt, ktoré nesie naviac, na nič nevyužije.
Teraz si uvedomme, že vždy keď príde na pokéstop, ocitá sa akoby na začiatku. Nič z toho čo sa dovtedy stalo totiž nemá vplyv na jeho ďalšie rozhodovanie. K dispozícii má totiž znovu m pokélôpt a k pokémonom, ktorých stretol cestou do tohto pokéstopu, sa nemôže vracať. Zaujímavá je teda zjavne cesta Peťa od jedného pokéstopu k druhému (poprípade od začiatku k prvému pokéstopu).
Zoberme si preto nejaký úsek udalostí medzi dvoma pokéstopmi. V tomto úseku stretol niekoľko rôzne silných pokémonov. Aký je najväčší súčet síl pokémonov, ktorých vedel na tomto úseku pochytať? Je jasné, že v každom úseku nemôže chytiť viac ako m pokémonov, lebo len toľko pokélôpt má. A keďže každý pokémon má kladnú silu[1] neoplatí sa mu nechytiť niektorého z pokémonov a zároveň nevyužiť pokéloptu. Ak to teda pôjde, bude sa vždy snažiť chytiť m pokémonov.
To samozrejme nebude možné vtedy, ak ich v danom úseku stretne menej ako m. V tom prípade je však situácia jednoduchá, pochytá všetkých, ktorých stretne. O niečo komplikovanejšie to je, ak je v danom úseku viac ako m pokémonov, pretože si musí vybrať, ktorých m z nich chytí.
Jeho cieľom je však maximalizovať súčet síl pokémonov, ktorých chytí. Je preto jasné, že chce chytiť m najsilnejších z týchto pokémonov. (Najlepšie sa niečo takéto dokáže nasledovne: Predstavme si, že Peťo chytil nejakého pokémona P_1, ktorý medzi m najsilnejších nepatrí. Keďže dokopy mohol chytiť len m pokémonov, musí existovať aspoň jeden spomedzi m najsilnejších, ktorého nechytil. Nazvime ho P_2. Ak by ale chytil P_2 namiesto P_1, dostal by lepšie riešenie.)
Toto nám dáva aj jednoduchý popis algoritmu, ktorý rieši našu úlohu. Pre každý úsek pokémonov medzi dvoma pokéstopmi sa najskôr pozrieme, či ich je najviac m. A áno, tak pochytáme všetkých. A ak je ich viac, tak m-krát budeme hľadať najsilnejšieho z nich. Keď ho nájdeme tak ho odstránime (alebo ho nahradíme pokémonom so silou 0), pripočítame si jeho silu k výsledku a pokračujeme. Takéto riešenie je však ešte vcelku pomalé, má časovú zložitosť $O(nm)$.

Chceme preto hľadanie m najväčších pokémonov nejakým spôsobom urýchliť. Napríklad by sme si ich mohli usporiadať od najsilnejšieho po najslabšieho a potom len zobrať prvých m. Vieme však rýchlo usporiadať veľké množstvo čísel podľa veľkosti?
Našťastie, tento problém je v programovaní tak rozšírený, že väčšina programovacích jazykov má vlastnú funkciu, ktorá usporadúva čísla v poli podľa veľkosti. A to dokonca s časovou zložitosťou $O(k \log k)$, kde k je počet prvkov, ktoré chceme usporiadať. Použitím takejto funkcie preto vieme našu úlohu vyriešiť v časovej zložitosti $O(n \log n)$.
Nižšie nasledujú dve implementácie vzorového riešenia za pomoci knižničných funkcií, jedno v jazyku C++ a druhé v jazyku Python.

Implementácia v jazyku C++

Funkcia, ktorá triedi prvky poľa sa volá `sort()` a nachádza sa v knižnici `algorithm`. Ako parametre musíme tejto funkcii zadať začiatok a koniec poľa, ktoré chceme usporiadať. Ak máme napríklad pole s názvom `A[]`, ktoré má n prvkov, zavoláme túto funkciu príkazom `sort(A, A+n);`. Po skončení tejto funkcie sú hodnoty v poli `A[]` usporiadané od najmenšej po najväčšiu.
V prípade, že chcete používať dynamické polia – vectory – môžete túto funkciu zavolať ako `sort(V.begin(), V.end());`, kde `V` je príslušný `vector`.
Funkcia `sort()` vie okrem čísel triediť aj ľubovoľné iné objekty, ktoré sa dajú porovnávať, napríklad reťazce, znaky, desatinné čísla (alebo aj naše vlastné typy, ak im buď definujeme `operator<` alebo funkcii `sort()` ako tretí parameter odovzdáme meno porovnávacej funkcie).

[1] Áno, aj psyduck.

Listing programu (C++)

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

int udalosti[1000047];
int tried[1000047];

int main() {
    int n,m;
    scanf("%d %d",&n,&m);
    for(int i=0; i<n; i++)
        scanf("%d",&udalosti[i]);
    udalosti[n++] = -1;  // pridám si pokéstop na koniec ako zarážku
    int zaciatok_useku=0;
    long long vysledok=0;
    while(zaciatok_useku < n) {
        int velkost_useku = 0;
        while(udalosti[zaciatok_useku + velkost_useku] != -1) {
            tried[velkost_useku] = udalosti[zaciatok_useku + velkost_useku];
            velkost_useku++;
        }
        sort(tried, tried + velkost_useku); // usporiada pole od najmenšieho po najväčšie číslo
        for(int i=1; i <= min(velkost_useku,m); i++)
            vysledok += tried[velkost_useku - i];
        zaciatok_useku += velkost_useku + 1;
    }
    printf("%lld\n",vysledok);
}
```

Implementácia v jazyku Python

V jazyku Python je triedenie implementované ako metóda samotného poľa (listu). Ak teda máme naše údaje uložené v poli `A`, stačí zavolať `A.sort()` a prvky v `A` sa preusporiadajú od najmenšieho po najväčší.
Navyše z poľa ľahko vyberieme posledných m prvkov: `A[-m]` je m-tý prvok od konca, a `A[-m:]` sú všetky prvky od neho (vrátane) až po koniec poľa.

Listing programu (Python)

```python
# načítame vstup
N, M = [ int(x) for x in input().split() ]
pokemoni = [ int(x) for x in input().split() ]

# na koniec si pridáme ešte jeden pokéstop ako zarážku
pokemoni.append(-1)

# prechádzame pole pokémonov a pri každom pokéstope vyhodnotíme aktuálny úsek
odpoved = 0
aktualny_usek = []
for x in pokemoni:
    if x == -1:
        # vyberieme najsilnejších m pokémonov (alebo všetkých, ak ich je menej)
        aktualny_usek.sort()
        if len(aktualny_usek) < M:
            odpoved += sum(aktualny_usek)
        else:
            odpoved += sum( aktualny_usek[ -M : ] )

        # a začneme nový úsek
        aktualny_usek = []
    else:
        # pridáme pokémona do aktuálneho úseku
        aktualny_usek.append(x)

# hotovo
print(odpoved)
```

Pre zaujímavosť uvádzame aj stručnejšiu implementáciu, ktorá ukazuje, čoho všetkého je Python schopný :)

Listing programu (Python)

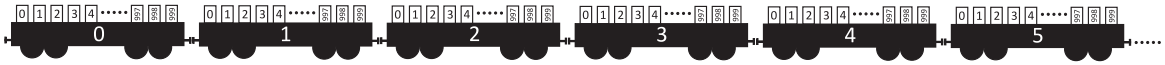
```python
def numbers(string): return [ int(x) for x in string.split() ]

def solve(segment,M): return sum( sorted(numbers(segment))[-M:] )

N, M = numbers(input())
segments = input().split('-1')
print( sum( solve(segment,M) for segment in segments ) )
```

No a úplne na koniec ešte spomenieme, že ani efektívne triedenie nie je najefektívnejším riešením tejto súťažnej úlohy (hoci na plný počet bodov bez problémov stačilo). Existujú totiž aj pokročilejšie algoritmy, pomocou ktorých vieme m najväčších spomedzi daných n prvkov nájsť v čase $O(n)$.

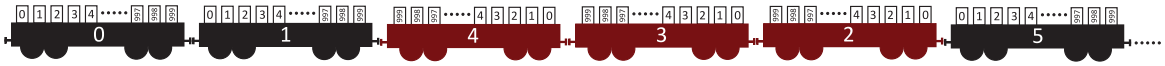
B-I-2 Obracanie poľa

Existuje viacero efektívnych riešení tejto súťažnej úlohy. Jedno z nich (to, ku ktorému sme sa vás snažili nasmerovať zadaním aj návodnými úlohami) si ukážeme do detailov, o inom si na konci aspoň spomenieme, že existuje.

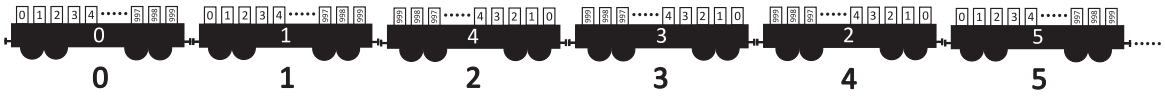
Jednotlivé políčka poľa, ktoré ideme meniť, si predstavme ako krabice. Každá krabica obsahuje práve jednu hodnotu. Ďalej si naše pole rozdeľme na úseky dĺžky 1000 a predstavme si, že každý úsek sme naložili na jeden *vagón*. Vagóny si očíslujeme od 0 po $v - 1$, kde $v = n/1000$ je počet vagónov. Krabice na každom vagóne si očíslujeme od 0 po 999. Na začiatku teda celá situácia vyzerá nasledovne:



Všimnime si, čo sa teraz stane, keď nejaký úsek poľa *reverzneme*. Keďže hranice každého reverzu sú na násobkoch 1000, znamená to vlastne, že zoberieme niekoľko po sebe idúcich vagónov a vrátime ich naspäť obrátene. Ak by sme napr. v našom vyššie znázornenom príklade reverzli úsek na pozíciách od 2000 po 5000, dostali by sme nasledovnú situáciu:



Čo sa teda vlastne zmenilo? V reverznutom úseku sa stali dve veci: 1. reverzlo sa *poradie vagónov*, 2. reverzlo sa aj *poradie krabíc na každom z nich*. Ako takúto operáciu spraviť šikovne? Napíšeme si na zem ku vlaku čísla od 0 po $v - 1$:



Teraz nám stačí pamätať si nasledovné veci:

- Ku každému číslu na zemi číslo vagóna, ktorý tam práve stojí. (Napr.: „na pozícii 2 máme vagón číslo 4“.) Na toto nám stačí obyčajné pole.
- Ku každému číslu vagóna pole dĺžky 1000: obsahy krabíc naložených na ňom. Na toto nám stačí obyčajné dvojrozmerné pole: prvý rozmer je číslo vagóna, druhý je číslo krabice na ňom.
- Ku každému číslu vagóna ešte jednu boolovskú premennú, hovoriacu, či je momentálne zaradený vo svojom pôvodnom smere.

Pomocou takýchto informácii:

- Vieme v konštantnom čase „pristupovať do poľa“. Ak chceme krabicu, ktorá sa aktuálne nachádza na indexe i, spravíme to nasledovne:
  1. Spočítame si, že správne miesto na zemi má číslo $a = \lfloor i/1000 \rfloor$ a že na tomto mieste chceme krabicu s poradovým číslom $b = i \bmod 1000$ zľava.
  2. Pozrieme sa, ktorý vagón stojí na mieste a.
  3. Ak je tento vagón zaradený v pôvodnom smere, chceme z neho krabicu číslo b. V opačnom prípade chceme krabicu číslo $999 - b$.

- Vieme v čase nanajvýš priamo úmernom počtu vagónov odsimulovať ľubovoľný reverz. Stačí v poli, kde si pamätáme postupnosť čísel vagónov, reverznúť príslušný úsek, a navyše každému z dotknutých vagónov znegovať jeho boolovskú premennú.

  Všimnite si, že takáto implementácia reverzu je zhruba 1000× rýchlejšia ako keby sme postupne po jednej presúvali všetky krabice – presunutím jedného vagónu ich presunieme tisíc naraz.

Listing programu (Python)

```
# načítame dĺžku poľa, spočítame počet vagónov

N = int( input() )
V = N // 1000

# vyrobíme si polia spomínané vo vzorovom riešení
poradie_vagonov = [ v for v in range(V) ]
je_vagon_naopak = [ False for v in range(V) ]
obsahy_vagonov  = []

for v in range(V):
    vagon = []
    for i in range(1000): vagon.append( 1000*v+i )
    obsahy_vagonov.append( vagon )

# pomocná funkcia, ktorá zmení index do poľa na dvojicu (vagón, číslo krabice na ňom)

def najdi_index(index):
    cislo_na_zemi = index // 1000
    cislo_vagona = poradie_vagonov[ cislo_na_zemi ]

    cislo_krabice = index % 1000
    if je_vagon_naopak[cislo_vagona]: cislo_krabice = 999 - cislo_krabice

    return cislo_vagona, cislo_krabice

# načítame počet udalostí

Q = int( input() )

# spracúvame udalosti

for query_id in range(Q):
    query = [ int(x) for x in input().split() ]

    if query[0] == 1:
        # zápis do poľa
        cislo_vagona, cislo_krabice = najdi_index( query[1] )
        obsahy_vagonov[ cislo_vagona ][ cislo_krabice ] = query[2]

    if query[0] == 2:
        # reverz kusu poľa
        lo, hi = query[1], query[2]
        otocit = poradie_vagonov[lo:hi]

        for vagon in otocit: je_vagon_naopak[vagon] = not je_vagon_naopak[vagon]
        poradie_vagonov = poradie_vagonov[:lo] + otocit[::-1] + poradie_vagonov[hi:]

    if query[0] == 3:
        # výpis
        vystup = []
        for index in range( query[1], query[2]+1 ):
            cislo_vagona, cislo_krabice = najdi_index(index)
            vystup.append( obsahy_vagonov[ cislo_vagona ][ cislo_krabice ] )

        print('␣'.join( str(x) for x in vystup ))
```

Poznámky na záver:
Trik s vagónmi, ktorý sme si ukázali v riešení tejto úlohy, sa dá úspešne použiť aj pri riešení mnohých iných úloh. Ide zväčša o úlohy typu „tu máš n-prvkové pole, potrebujeme efektívne spracúvať také a onaké operácie, pričom každá operácia pracuje s nejakým súvislým úsekom poľa“. Pole si rozdelíme na $\sqrt{n}$ vagónov a na každý naložíme $\sqrt{n}$ prvkov (vhodne zaokrúhlené). Keď chceme spracovať ľubovoľný úsek poľa, vieme to spraviť tak, že sa pozrieme na niekoľko (nanajvýš $\sqrt{n}$) celých vagónov a nanajvýš na dvoch vagónoch – jeden na ľavom a jeden na pravom konci úseku – sa pozrieme na jednotlivé krabice (ktorých je dokopy nanajvýš $2\sqrt{n}$).

Existujú aj iné riešenia tejto súťažnej úlohy, vrátane riešení, ktoré sú efektívnejšie a dokonca aj všeobecnejšie. Dá sa napríklad implementovať riešenie, ktoré bude vedieť úplne ľubovoľný úsek n-prvkového poľa reverznúť v čase $O(\log n)$ a v rádovo rovnakom čase budeme vedieť aj indexovať do takéhoto poľa. Takéto riešenia sú

však veľmi náročné na implementáciu – ani od riešiteľov kategórie A by sme ich nechceli :) Jedno možné takéto riešenie je založené na tom, že si prvky pamätáme vo vrcholoch tzv. splay stromu, a navyše si v každom vrchole pamätáme, či je jeho celý podstrom momentálne reverznutý.

B-I-3 Hra s kartami

V tejto úlohe máme daný zoznam kariet a chceli by sme v ňom nájsť najdlhší úsek, v ktorom sa žiadna hodnota nevyskytuje dvakrát.
Predstavme si, že sme karty uložili v danom poradí do radu idúceho zľava doprava. Pozície v tomto rade si očíslujeme od 0 po $n-1$. Predstavme si ďalej, že sme si zvolili index z: číslo pozície, na ktorej chceme mať začiatok úseku. Ako nájdeme najdlhší dobrý úsek, ktorý na tomto indexe začína? Jednoducho: budeme sa postupne pozerať na karty na indexe z, $z+1$, ..., až kým sa nám nejaká hodnota nezopakuje. Index, na ktorom leží táto hodnota, označme k_z („koniec zodpovedajúci začiatku z“). Ak sa až po koniec radu žiadna hodnota nezopakuje, bude $k_z = n$.
Ku konkrétnemu začiatku z takto vieme v čase $O(n)$ nájsť zodpovedajúci koniec k_z. Dĺžka práve nájdeného dobrého úseku kariet je $k_z - z$. (Pripomíname, že karta na indexe z do nájdeného úseku patrí, ale karta na indexe k_z už nie.)
Vyššie popísaná úvaha nám súťažnú úlohu vyrieši v čase $O(n^2)$. Stačí si uvedomiť, že existuje len n možných začiatkov. Môžeme teda postupne vyskúšať každý z nich. Na záver už len spomedzi n nájdených dobrých úsekov vyberieme ten najdlhší.

Riešenie v lineárnom čase

Táto súťažná úloha však má aj efektívnejšie riešenie. Vieme ho vyrobiť tak, že z predchádzajúceho riešenia šikovne vynecháme niektoré zbytočné časti.
Všimnime si situáciu, keď sme práve k nejakému začiatku z našli zodpovedajúci koniec k_z. V pamäti ešte máme uloženú množinu hodnôt, ktoré sa nachádzajú v úseku od z po $k_z - 1$.
Teraz by sme chceli ku začiatku $z+1$ nájsť zodpovedajúci koniec k_{z+1}. Tento ale nemusíme začať hľadať úplne odznova! V prvom rade, určite platí $k_{z+1} \geq k_z$: tým, že sme posunuli začiatok z pozície z na pozíciu $z+1$, sme len z nášho úseku odstránili jednu hodnotu. Keďže všetky hodnoty na pozíciách z až $k_z - 1$ boli navzájom rôzne, tým skôr musia byť navzájom rôzne hodnoty na pozíciách $z+1$ až $k_z - 1$.
A navyše presne vieme, ktoré hodnoty na týchto pozíciách máme: z množiny, ktorú máme uloženú v pamäti, stačí odstrániť hodnotu z pozície z. Takže namiesto toho, aby sme koniec k_{z+1} hľadali začínajúc na pozícii $z+1$, vieme ho začať hľadať priamo od pozície k_z.
A to už je všetko. Postupne budeme skúšať hodnoty $z = 0, 1, \ldots, n-1$ a zakaždým posúvať koniec úseku doprava, kým nenarazíme na prvý duplikát. A keďže koniec posúvame len doprava, dokopy počas celého algoritmu ho posunieme nanajvýš n-krát. Celkový počet krokov preto bude lineárny od počtu kariet.

Listing programu (Python)

```
# načítame vstup

N, M = [ int(x) for x in input().split() ]
karty = [ int(x) for x in input().split() ]

# inicializujeme si doteraz najlepšie riešenie na 1-prvkový úsek

best_z, best_k = 0, 0

# spravíme si pomocné pole, v ktorom si budeme pamätať, ktoré hodnoty sú v aktuálnom úseku

vnutri = [ False for m in range(M+1) ]

# ideme postupne skúšať všetky začiatky

k = 0
for z in range(N):
    # ak je z > 0, odstránime z aktuálneho úseku prvok na indexe z-1

    vnutri[ karty[z-1] ] = False
```

```
    # aktuálny úsek je z .. k-1, ideme postupne posúvať k, kým sa to dá

    while (k<N) and not vnutri[ karty[k] ]:
        vnutri[ karty[k] ] = True
        k += 1

    # keď sa vyššie uvedený cyklus zastavil, máme ku aktuálnemu začiatku z
    # nájdený najlepší možný koniec k

    if k - z > best_k - best_z:
        best_z, best_k = z, k

# vypíšeme najlepšie riešenie (s tým, že v zadaní indexujeme od 1)

print( 'dlzka', best_k-best_z, 'zaciatok', best_z+1 )
```

B-I-4 Tabuľa

Kto si vyskúšal, ako sa hra zo zadania správa pre hodnoty menšie ako 50, určite nemohol mať problém získať za túto úlohu nejaké body. Pozrime sa postupne na to, ako sa dali riešiť jednotlivé podúlohy.

Podúloha A: vyrob 1

Existuje veľa ľahko popísateľných postupov. Náš obľúbený je nasledovný: Na začiatku si čísla rozdelíme do dvojíc: 1 s 2, 3 so 4, 5 so 6, ..., až 49 s 50. Každú dvojicu od seba odčítame, čím na tabuli dostaneme 25× číslo 1.
Jednu jednotku si „odložíme nabok“. Ostatných 24 rozdelíme do 12 dvojíc. Každú dvojicu jednotiek zmažeme a namiesto nej dostaneme nulu. V tejto chvíli teda máme na tabuli jednu 1 a dvanásť 0. A odtiaľto je už cesta do cieľa zjavná, ba priam by sa dalo povedať, že už nie je čo pokaziť :)

Podúloha B: najväčšie číslo počas hry

Každé číslo na tabuli bude nezáporné, minimum je teda 0. Najväčšie číslo na začiatku je 50. Každé ďalšie číslo vznikne ako rozdiel dvoch už existujúcich. Nikdy preto nevyrobíme číslo väčšie ako $50 - 0 = 50$.
Ostáva už len dodať, že Kaja vie skutočne aj počas hry zapísať číslo 50. Toto vieme dosiahnuť napr. nasledovne: $3 - 2 = 1$, $1 - 1 = 0$, $50 - 0 = 50$.

Podúloha C: dá sa 14?

Kto si vyskúšal všetky možné priebehy hry, ktorá začína tým, že sú na tabuli len čísla 1, 2, 3, 4, mohol si všimnúť, že posledné číslo na konci hry je vždy párne. Naopak, ak sú na začiatku na tabuli čísla od 1 po 5, skončíme vždy s nepárnym výsledkom. Prečo ale?
Odpoveď je jednoduchá. Stačí sa pozerať na to, koľko máme v ktorej chvíli na tabuli nepárnych čísel.

- Rozdiel dvoch párnych čísel je párny. Ak teda zmažeme dve párne čísla, napíšeme nové párne číslo. Počet nepárnych sa nezmenil.
- Aj rozdiel dvoch nepárnych čísel je párny. Ak teda zmažeme dve nepárne čísla, napíšeme nové párne číslo. Počet nepárnych klesol o 2.
- No a rozdiel nepárneho a párneho čísla (v ľubovoľnom poradí) je vždy nepárny. Ak teda zmažeme nepárne a párne číslo, napíšeme namiesto nich nepárne číslo. Počet nepárnych čísel na tabuli sa teda opäť nezmenil.

Čo sa teda stane, ak máme na začiatku na tabuli čísla od 1 do 50? Nepárnych čísel medzi nimi je presne 25. Ako bude hra pokračovať, v niektorých ťahoch sa ich počet zmenší, ale vždy presne o 2. Preto vždy bude na tabuli nepárnych čísel nepárny počet – a teda aspoň jedno. No a z toho už je zjavné, že úplne na konci hry, keď nám už ostalo jedno jediné číslo, musí toto číslo byť nepárne.
Nech teda Kaja hrá ako len chce, číslo 14 na konci hry dosiahnuť nemôže.

Podúloha D: nájdi všetky možné hodnoty na konci hry

Už vieme, že posledné číslo musí byť nepárne a musí ležať medzi 0 a 50. Zostáva ukázať, že hociktoré takéto číslo vieme vyrobiť. Zovšeobecníme postup z podúlohy A.

Povedzme, že chceme na konci vyrobiť číslo $2k + 1$ (pre nejaké $k > 0$). „Odložíme si nabok“ čísla 1 a $2k + 2$. Ostatné čísla popárujeme za radom do dvojíc. Všimnime si, že v každej dvojici sa čísla líšia o 1.

(Príklad: Nech $k = 3$. Nabok si odložíme čísla 1 a 8. Páry, ktoré vyrobíme, budú 2-3, 4-5, 6-7, 9-10, 11-12, ..., 49-50.)

Postupne každý pár zmažeme a napíšeme jeho rozdiel, teda číslo 1. Máme 24 párov, dostaneme teda 24 jednotiek. Tieto popárujeme, dostaneme 12 núl.

No a teraz už len od seba odčítame $2k + 2$ a 1, čím dostaneme želané číslo $2k + 1$. V tejto chvíli máme na tabuli jedenkrát číslo $2k + 1$ a dvanásťkrát číslo 0. Dohrať hru do úspešného konca už určite zvládnete.

TRIDSIATY DRUHÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2016