Na řešení úloh máte 4,5 hodiny čistého času.

Řešení každého příkladu musí obsahovat:

- **Popis řešení,** to znamená slovní popis použitého algoritmu, argumenty zdůvodňující jeho správnost (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není vhodné odkazovat se na Vaše řešení předchozích kol, opravovatelé je nemají k dispozici; na autorská řešení se odkazovat můžete.
- **Program.** V úlohách **P-III-1** a **P-III-2** je třeba uvést dostatečně podrobný zápis algoritmu, např. ve tvaru zdrojového textu nejdůležitějších částí programu v jazyce Pascal nebo C. Ze zápisu můžete vynechat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů apod. V řešení úlohy **P-III-3** je nutnou součástí řešení program pro grafomat.

Hodnotí se nejen správnost programu, ale také kvalita popisu řešení a efektivita zvoleného algoritmu.

P-III-1 Pizza vrací úder

Rozmach Marcovy firmy narazil na konec roku, přesněji řečeno na daňové přiznání. Během rozvážení pizz a zřizování nových poboček Marco neměl čas zabývat se účetnictvím a nyní s úděsem zjistil, že si u svých zběžných poznámek zapomněl zapsat, co jsou příjmy a co výdaje. Nicméně nezpanikařil, vzpomněl si na příběhy, které vykládal jeho dědeček (bývalý kápo italské mafie), a jal se účetní záznamy doplnit (zfalšovat).

Aby výsledek vypadal co nejdůvěryhodněji, rozhodl se použít čísla ze svých poznámek a pouze si u nich zvolit, které jsou příjmy a které výdaje. Navíc se rozhodl, že když falšovat, tak pořádně. Rád by vypadal jako úspěšný obchodník, a tedy nekončil ve ztrátě. Na druhou stranu, chce platit co nejmenší daně, tedy jeho zisk by měl být co nejmenší. Při řešení tohoto nelehkého úkolu by mu mohlo pomoci to, že velikosti čísel v jeho poznámkách jsou podstatně menší než jejich počet.

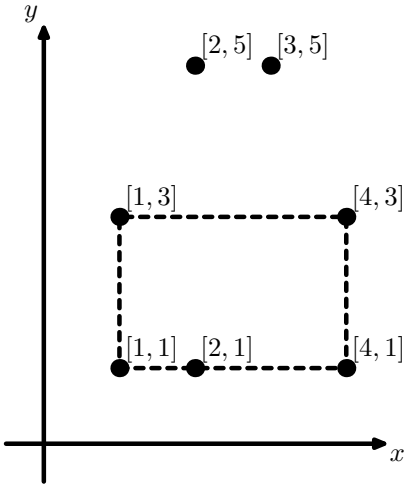
Soutěžní úloha:

Je dáno n přirozených čísel $a_1, \ldots, a_n$ z rozsahu 1 až k; hodnota k je typicky řádově menší než n. Vaším úkolem je nalézt čísla $s_i \in \{+1, -1\}$ taková, že součet $z = s_1a_1 + s_2a_2 + \ldots + s_na_n$ je nezáporný a zároveň nejmenší možný. Např. pro $n = k = 4$ a $a_i = i$ pro $i = 1, 2, 3, 4$ je optimální řešení $s_1 = s_4 = +1$ a $s_2 = s_3 = -1$ s velikostí součtu $z = 0$.

P-III-2 Obdélník

V rovině je dáno n vesměs různých bodů $B_1, \ldots, B_n$. Vaším úkolem je navrhnout algoritmus, který nalezne obdélník $A_1A_2A_3A_4$, jehož vrcholy jsou některé ze zadaných bodů, tj. $A_1 = B_i$ pro nějaké i, $1 \leq i \leq n$, analogicky pro A_2, A_3 a A_4, a počet bodů B_i ležících uvnitř obdélníku $A_1A_2A_3A_4$ je maximální možný. Navíc se požaduje, aby hrany obdélníku $A_1A_2A_3A_4$ byly rovnoběžné s osami, tj. x-ové souřadnice vrcholů A_1 a A_4 a vrcholů A_2 a A_3 byly stejné a rovněž y-ové souřadnice vrcholů A_1 a A_2 a vrcholů A_3 a A_4 byly stejné. Body, které leží na hranách obdélníku $A_1A_2A_3A_4$, považujeme za body ležící uvnitř obdélníku. Pokud zadané body netvoří žádný obdélník $A_1A_2A_3A_4$, který by vyhovoval podmínkám zadání, program vypíše vhodnou zprávu.

Jedno z možných zadání a příklad řešení (v tomto případě jediného optimálního) jsou na následujícím obrázku. Je zde $n = 7$ bodů se souřadnicemi $[1, 1]$, $[2, 1]$, $[4, 1]$, $[1, 3]$, $[4, 3]$, $[2, 5]$ a $[3, 5]$. Optimálním řešením je obdélník s rohy $[1, 1]$, $[4, 1]$, $[4, 3]$ a $[1, 3]$, který obsahuje pět bodů.



P-III-3 Grafomat a šamani

Na indiánské vesnice kmene Grafomatonů se zvolna snáší soumrak. Příštího dne začne největší ze slavností tohoto léta, na níž se sejdou všichni členové kmene. Indiáni se v tichém očekávání chystají ulehnout do svých teepee, pouze šamani postávají u signálních ohňů a pilně vysílají kouřové signály do sousedních vesnic. Rituál totiž vyžaduje, aby lidé z právě poloviny vesnic přišli pomalováni červeně a z druhé poloviny zeleně. Jen šamani vědí, jak se na tom dokáží domluvit – možných signálů je totiž pomálu a každá vesnice vidí jen na tři sousední. Jak to mohou dělat? Jak? ...

Když jste se ze sna probudili, napadlo vás, že indiánské domlouvání docela přesně odpovídá této úloze pro grafomat:

Soutěžní úloha:

Napište program pro grafomat, který v zadaném 3-grafu s jedním označeným vrcholem označí právě polovinu vrcholů a skončí. Předpokládejte, že graf je souvislý (z každého vrcholu jde po hranách dojít do každého) a že má sudý počet vrcholů, takže rozdělení na poloviny je vždy možné. (Sudý počet vrcholů mají ve skutečnosti všechny 3-grafy, ale to zde nebudeme dokazovat.)

Vstup bude tvořen proměnnou x, která bude v označeném vrcholu rovna jedné a všude jinde nulová.

Výstupem programu bude proměnná y, nulová v právě polovině vrcholů a jedničková ve zbývajících.

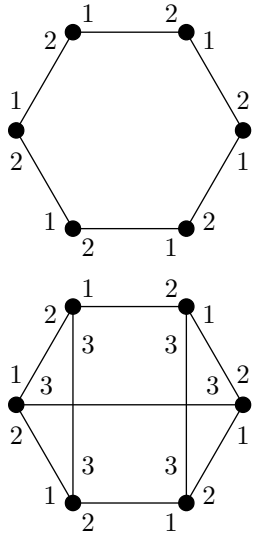
Nápověda: Zkuste si nejdříve rozmyslet, jak by se úloha dala řešit, pokud by namísto obecného 3-grafu byl zadán strom (tj. souvislý graf bez cyklů).

Studijní text:

Tento studijní text je stejný jako v krajském kole. Nadto si dovolujeme připomenout, že počet stavů každého automatu musí být konečný, takže nelze používat proměnné, jejichž rozsah hodnot závisí na velikosti vstupu.

Grafem nazveme libovolnou konečnou množinu V *vrcholů* grafu spolu s množinou E *hran,* což jsou neuspořádané dvojice vrcholů. Žádné dva vrcholy nejsou spojeny více hranami, žádná hrana nespojuje vrchol se sebou samým.

K-graf budeme říkat takovému grafu, ve kterém s každým vrcholem sousedí právě K hran a konce těchto hran jsou očíslovány přirozenými čísly od 1 do K. Oba konce jedné hrany přitom mohou být očíslovány různě. Pokud budeme hovořit o hranách vycházejících z nějakého vrcholu v, budeme zmiňovat *místní* čísla hran (to jsou čísla konce, kterým je v) a čísla *protější* (to jsou ta zbývající). Pro každý vrchol jsou místní čísla všech jeho hran navzájem různá. Obrázek vpravo ukazuje příklad 2-grafu a 3-grafu.



Ohodnocením grafu nazveme přiřazení prvků nějaké konečné množiny vrcholům grafu – tedy například rozdělení vrcholů na černé a bílé nebo označení vrcholů čísly od 1 do 5.

Grafomat je zařízení pro automatické řešení grafových úloh. Jeho vstupem je libovolný K-graf G spolu s jeho ohodnocením; výstupem je nějaké další ohodnocení téhož grafu. Samotný výpočet je vykonáván *automaty* umístěnými v jednotlivých vrcholech grafu. Každý automat má svou paměť a řídí se programem. Programy všech automatů jsou identické, zatímco paměť má každý automat svoji a mimo to ještě může nahlížet do pamětí svých grafových sousedů.

Paměť automatu je tvořena konečným množstvím proměnných, které si můžeme představit jako pascalské proměnné typu interval. Obsahují tedy přirozená čísla v nějakém pevném rozsahu, který nezávisí na velikosti vstupu. Mimo to je také možné používat pole intervalových proměnných, jejichž indexy jsou opět z pevných intervalů. Žádné jiné typy proměnných (neomezeně velká čísla, ukazatele, ...) použít nelze.

Zvláštní roli hrají proměnné x a y. Proměnná x na počátku výpočtu obsahuje vstupní ohodnocení toho vrcholu grafu, ke kterému patří, hodnota proměnné y na konci výpočtu určí výstupní ohodnocení vrcholu. Všechny proměnné s výjimkou proměnné x mají svou počáteční hodnotu pevně určenu. Deklarace proměnných vypadá například takto:

```
var x: 1..5;                          { číslo od 1 do 5, na počátku vstup }
    y: 1..5 = 3;                      { číslo od 1 do 5, na počátku 3, na konci výstup }
    z: array [1..2] of 3..4 = (3, 4); { pole dvou čísel }
```

Řídící program automatu si můžeme představit jako pascalský program, v němž si zakážeme používat rekurzi a který bude manipulovat pouze s proměnnými v paměti automatu a případně i automatů sousedních. Na své vlastní proměnné se automat odkazuje jejich jmény, jako by to byly obyčejné pascalské globální proměnné, na proměnné sousedů pak konstrukcí $S[i].p$. Zde i je celočíselný výraz s hodnotou $1 \ldots K$, jenž značí, o kolikátého souseda se jedná, tedy místní číslo hrany, kterou je soused připojen; p je jméno libovolné proměnné. Proměnné sousedů je možné pouze číst.

Aby mohl program dávat do souvislostí své hrany s hranami svých sousedů, má k dispozici ještě proměnné $P[1], \ldots P[K]$, které jsou pevně nastaveny tak, že $P[i]$ obsahuje protější číslo hrany s místním číslem i. Výraz $S[i].S[P[i]].x$ je tedy totéž jako samotné x. (Pozor, zatímco druhé S je odkaz na proměnnou patřící sousedovi, proměnná P v indexu je opět místní.)

Výpočet grafomatu probíhá v taktech, a to následovně: V nultém taktu se proměnné všech automatů nastaví na počáteční hodnoty a proměnné x na vstupní ohodnocení jednotlivých vrcholů. V každém dalším taktu se pak vždy jednou spustí program každého automatu, přičemž proměnné svých sousedů vidí program ve stavu, v jakém byly na začátku taktu. Ačkoliv tedy jednotlivé automaty běží současně, nemůže se stát, že by jeden četl z proměnné, do které právě druhý zapisuje.

Výpočet pokračuje tak dlouho, dokud v nějakém taktu všechny automaty neprovedou příkaz `stop`. Pak se výpočet zastaví a z proměnných y grafomat přečte výstupní ohodnocení grafu. Pokud příkaz `stop` provedou jen některé automaty, výpočet pokračuje, a to i na těchto automatech. Struktura grafu, jakož i obsah proměnných P zůstává po celou dobu výpočtu konstantní.

Za *časovou složitost* výpočtu budeme považovat počet taktů, které uběhnou do zastavení. Nijak tedy nezávisí na rychlosti programů jednotlivých automatů. Podobně jako u časové složitosti klasických algoritmů nebudeme hledět na multiplikativní konstanty a bude nás zajímat pouze asymptotické chování složitosti, tedy zda je lineární, kvadratická, atd. Případy, kdy výpočet neskončí, nebudeme připouštět, pro úplnost ale dodejme, že tehdy se nutně musí hodnoty proměnných periodicky opakovat.

Příklad 1: Je dán 3-graf a v něm vyznačen jeden vrchol v, a to tak, že jeho proměnná x bude inicializována jedničkou, zatímco všem ostatním vrcholům nulou. Napište program pro grafomat, který označí všechny vrcholy z vrcholu v dosažitelné po hranách, a to tak, že jejich proměnná y bude na konci výpočtu rovna jedné, zatímco u nedosažitelných vrcholů bude nulová.

Řešení: Inspirujeme se prohledáváním grafu do šířky. V každém taktu se každý vrchol podívá, zda některý z jeho sousedů je již označen a pokud ano, také se sám označí. Pokud se označení nezmění, vrchol voláním `stop` souhlasí se zastavením. Průběh výpočtu tedy bude vypadat tak, že v i-tém taktu budou označeny ty vrcholy, jejichž vzdálenost od v je menší nebo rovna i. Výpočet zastaví, jakmile se hodnoty proměnných přestanou měnit, tj. po nejvýše N taktech. Proto je časová složitost našeho programu lineární v počtu vrcholů (na rozdíl od klasického průchodu do šířky nezávisí na počtu hran).

Program vypadá následovně:

```
var x: 0..1;                                  { byl vrchol označen ve vstupu? }
    y: 0..1 = 0;                              { je označen teď? }
    prev: 0..1 = 0;                           { předchozí stav }
    i: 1..3;
begin
   prev := y;                                 { zapamatujeme si, jestli už byl označen }
   if x=1 then y := 1;                        { přeneseme označení ze vstupu }
   for i := 1 to 3 do                         { podívejme se na všechny sousedy }
      if S[i].y <> 0 then                     { je-li i-tý soused označen, }
         y := 1;                              { označ i sebe sama }
   if y = prev then stop;                     { pokud se nic nemění, můžeme končit }
end.
```

Příklad 2: Mějme 2-graf složený z jediného cyklu sudé délky (tj. z vrcholů očíslovaných $0 \ldots N-1$, přičemž vrchol i je spojen hranou označenou 1 s vrcholem $(i+1) \bmod N$ a hranou označenou 2 s vrcholem $(i-1) \bmod N$; příklad takového grafu pro $N = 6$ najdete na obrázku na začátku tohoto textu). V tomto grafu je vyznačen jeden vrchol v. Napište program pro grafomat, který označí vrchol protilehlý k v, tedy vrchol s číslem $(v + N/2) \bmod N$.

Řešení: Vyšleme „signál“ putující z vrcholu v ve směru jedničkových hran rychlostí 1 vrchol za takt a druhý signál putující stejnou rychlostí opačným směrem. Jakmile nějaký vrchol zjistí, že do něj přišly oba signály, označí se a signály již dál nepředává.

```
var x: 0..1;                                  { vstupní značka u vrcholu }
    y: 0..1 = 0;                              { výstupní značka }
    l, r: 0..1 = 0;                           { už tímto vrcholem prošel signál doleva a doprava? }
begin
   if x=1 then                                { začínáme posílat }
      begin x := 0; l := 1; r := 1; end
   else if (S[2].l=1) and (S[1].r=1) then     { signály se v tomto vrcholu potkaly }
      begin y := 1; stop; end
   else if (S[2].l=1) and (l=0) then l := 1   { předáme signál doleva }
   else if (S[1].r=1) and (r=0) then r := 1   { předáme signál doprava }
   else stop;                                 { nic se neděje => můžeme končit }
end.
```

P-III-4 Policie zasahuje

Program: `policie.pas` / `policie.c` / `policie.cpp`
Vstup: `policie.in`
Výstup: `policie.out`

I v tomto kole se na vás radní Stínové Prahy obracejí s dalším, ještě naléhavějším, problémem. Ve městě se totiž usídlila mafie a její řádění překročilo únosnou mez. Proto byla městská policie pověřena učinit řádění mafie přítrž. Jak už to ale bývá, není dostatek důkazů o činnosti mafie, a tak se policisté rozhodli nějakou dobu sledovat, jak se mafiáni mezi sebou stýkají. Mafiáni jsou však prohnaní a nechodí jen po ulicích, ale využívají ke svým přesunům i kanalizační systém města. Do kanalizace je tedy třeba rozestavit policejní hlídky tak, aby bylo zamezeno tajným kontaktům mezi mafiány. Přesněji, je potřeba, aby na každé cestě mezi domy dvou mafiánů byla alespoň jedna policejní hlídka.

Tento nelehký úkol naštěstí zjednodušuje fakt, že kanalizačním systémem Stínové Prahy lze mezi každými dvěma domy projít právě jedním způsobem. Takže pokud se má mafián dostat kanalizací z jednoho místa na druhé, má jen jedinou možnost, kudy kanalizací projít (pokud nechce jít žádným místem dvakrát). Speciálně to tedy znamená, že kanalizace má „acyklickou“ strukturu a je souvislá, jako např. kanalizační systém na obrázku.

Vaším úkolem je napsat program, který pro daný popis kanalizačního systému a seznam domů ve vlastnictví mafiánů určí minimální počet hlídek, které je nutné do kanalizačního systému rozmístit tak, aby na každé cestě mezi dvěma domy mafiánů byla alespoň jedna hlídka. Hlídky lze umísťovat pouze do míst větvení, tj. hlídka nemůže být umístěna uprostřed stoky. Speciálně hlídka, která je umístěna ve větvení, kam je napojen dům některého z mafiánů, odděluje tento dům od všech ostatních domů.

Vstup:

Na prvním řádku vstupního souboru `policie.in` jsou dvě celá čísla n ($3 \leq n \leq 100\,000$) a p ($2 \leq p < n$) oddělená jednou mezerou. Číslo n udává počet větvení v kanalizačním systému – *větvením* rozumíme buď slepý konec nějaké stoky (napojený na dům, který ale nemusí patřit mafiánovi) nebo křižovatku, ze které vedou alespoň dvě stoky. Kanalizační systém města je pak tvořen $n - 1$ stokami, z nichž každá spojuje dvě větvení. Číslo p udává počet domů, které jsou ve vlastnictví mafiánů.

Větvení v kanalizaci jsou očíslována přirozenými čísly od 1 do n. Domy jsou do kanalizace připojeny jen v místech větvení. Na každém z následujících $n - 1$ řádků vstupního souboru jsou dvě celá čísla a_i a b_i ($1 \leq a_i, b_i \leq n$), která určují větvení spojená i-tou stokou.

Posledních p řádků vstupního souboru obsahuje vždy jedno celé číslo od 1 do n. Tato čísla jsou navzájem různá a určují čísla větvení v kanalizaci, kam jsou napojeny domy mafiánů.

Výstup:

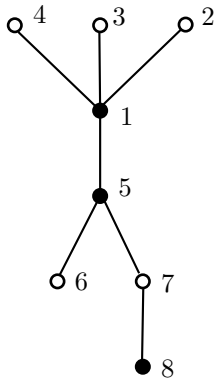
Váš program má do výstupního souboru `policie.out` vypsat nejmenší možný počet míst, která je třeba obsadit hlídkou, aby na cestě mezi každými dvěma domy mafiánů byla alespoň jedna hlídka.

Příklad:

`policie.in`
```
8 5
1 2
1 3
1 4
1 5
5 6
5 7
7 8
2
3
4
6
7
```

`policie.out`
```
2
```

(Hlídky je možné umístit na větvení s čísly 1 a 5.)



P-III-5 Rybka

Program: `rybka.pas` / `rybka.c` / `rybka.cpp`
Vstup: `rybka.in`
Výstup: `rybka.out`

Za horkých bezmračných letních dní se voda v rybníce Blaťáku někdy skoro vaří. Slunce nemilosrdně žhne a malé rybky se spěchají skrýt do hlubších a chladnějších částí rybníka. Jen rybka Julka se opozdila za ostatními a už skoro umdlévá.

Taková choulostivá malá rybka, jako je Julka, snáší jen určitý rozsah teplot, řekněme t_1 až t_2 (včetně krajních hodnot). Ráno mají různé části rybníka různou teplotu a jakmile vyjde slunce, všechny části rybníka se ohřívají stejně rychle, a to o 1 stupeň za 1 časovou jednotku. Rybník Blaťák je obdélníkový a je rozdělen na $m \times n$ stejně velkých čtvercových polí. Pole na pozici $[i, j]$ má ráno teplotu $T[i, j]$ stupňů ($1 \leq i \leq m$, $1 \leq j \leq n$). Julka je ráno na pozici $[J_x, J_y]$ a potřebovala by se dostat do bezpečí na pozici $[C_x, C_y]$. Julka se za každou časovou jednotku posune na jedno ze čtyř přes hranu sousedících čtvercových polí nebo zůstane stát. Pak se vždy teplota všech polí zvýší o 1 stupeň.

Samozřejmě se rybička cestou nesmí přehřát ani nachladit, tj. pole, kde se Julka vyskytuje, musí mít teplotu T, $t_1 \leq T \leq t_2$. Tato teplota musí být dodržena jak v okamžiku, když Julka na pole vplouvá, tak když jej opouští (mezi čímž se teplota zvýšila alespoň o 1 stupeň). Vyjímku tvoří jen cílové pole, na které smí vplout nezávisle na jeho teplotě a tím se dostat do bezpečí. Rybka Julka nesmí samozřejmě na své cestě opustit rybník.

Napište program, který dostane na vstupu Julčin teplotní rozsah $\langle t_1, t_2 \rangle$ ($0 \leq t_1 < t_2 \leq 10^6$), velikost Blaťáku $m \times n$ ($1 \leq m, n \leq 1\,000$), počáteční a cílovou pozici Julky $[J_x, J_y]$ a $[C_x, C_y]$ ($1 \leq J_x, C_x \leq m, 1 \leq J_y, C_y \leq n$) a počáteční teplotu každého pole rybníka $T[i, j]$ ($0 \leq T[i, j] \leq 10^6$) a najde pro naši malou rybku cestu do bezpečí respektující teplotní omezení či zjistí, že taková cesta neexistuje. Nalezená cesta má být nejrychlejší možná, tj. má trvat co nejméně časových jednotek. Pokud existuje více nejrychlejších cest, program může vypsat libovolnou z nich. Můžete předpokládat, že všechny teploty t_1, t_2 a $T[i, j]$ jsou celá čísla. Navíc také můžete předpokládat, že rybka je na počátku v poli s pro ni přijatelnou teplotou a že počáteční pole je různé od cílového.

Vstup:

Na prvním řádku vstupního souboru `rybka.in` jsou čtyři celá čísla m, n, t_1 a t_2 oddělená mezerami, na druhém řádku jsou pak souřadnice J_x, J_y, C_x a C_y. Všechna čísla m, n, t_1, t_2, J_x, J_y, C_x a C_y splňují výše uvedená omezení. Rybník Blaťák je orientován tak, že pole se souřadnicemi $[i, j]$ sousedí severní hranou s polem se souřadnicemi $[i, j-1]$, jižní hranou s polem se souřadnicemi $[i, j+1]$, západní hranou s polem $[i-1, j]$ a východní hranou s polem $[i+1, j]$. Posledních n řádků vstupního souboru popisuje počáteční teploty jednotlivých částí rybníka, na řádku $j+2$ se tedy nacházejí čísla $T[1, j], T[2, j], T[3, j], \ldots, T[m, j]$ vzájemně oddělená mezerami.

Výstup:

Do souboru `rybka.out` vypište Julčinu cestu jako posloupnost pokynů pro pohyb rybky oddělených mezerami. Pokyn je buď jeden znak `S`, `J`, `V` nebo `Z`, který určuje směr pohybu rybky, nebo kladné číslo udávající počet časových jednotek, po které se rybka nepohybuje. Jednotlivé pokyny jsou odděleny jednou mezerou. Výstupní soubor nesmí obsahovat dvě čísla za sebou a poslední pokyn, který obsahuje, musí být pokynem k pohybu rybky. V případě, že cesta neexistuje, vypište do výstupního souboru řádek s textem „`Chudak Julka!`“.

Příklad:

`rybka.in`
```
3 4 10 20
2 2 3 4
9 7 13
0 18 15
1 15 19
2 3 0
```

`rybka.out` *(jedna z více možností)*
```
V S 1 Z Z 5 J J J V V
```

`rybka.in`
```
3 2 20 30
1 1 3 1
25 13 25
27 24 0
```

`rybka.out`
```
Chudak Julka!
```