fiks!

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FIT

Fiťácký **I**nformatický **K**orespondenční **S**eminář

Ročník 2023/24, 2. kolo

Co je to FIKS?

FIKS je Fiťácký Informatický Korespondenční Seminář pro středoškolské studenty pořádaný Fakultou informačních technologií ČVUT v Praze. Byl založen na podzim roku 2013 a nyní tak probíhá desátý ročník (samozřejmě číslujeme od nuly). Nabízí možnost potrápit tvůj mozek řešením algoritmických úloh různé obtížnosti, od snadných po zapeklité, na nichž se můžeš leccos nového naučit a podstatně se zdokonalit.

Jak to probíhá?

Jeden ročník se skládá z několika kol a následného soustředění pro nejlepší řešitele. V těchto kolech máš možnost v teple domova řešit zadané úlohy, a své řešení nám potom odešleš. My ti toto řešení opravíme, obodujeme a pošleme zpět, aby ses mohl poučit ze svých chyb. Spolu s tím zveřejníme vzorové řešení, které můžeš prostudovat a třeba se něco přiučit. Získané body se sčítají do konečného žebříčku, ze kterého vybereme ty nejlepší a pozveme je na již zmíněné soustředění.

Proč řešit FIKS?

Řešením každého problému, se kterým se potýkáme, se zdokonalujeme. Zde ti nabízíme možnost pořádně se zamyslet nad zajímavými algoritmickými problémy, vyzkoušet své algoritmické myšlení a programátorské dovednosti a naučit se spoustu nových věcí.

Také je to možnost seznámení s novými lidmi, které baví informatika, programování, matematika a přemýšlení vůbec. Těm nejlepším jsme schopni garantovat přijetí na FIT ČVUT bez přijímacích zkoušek.

Jak se můžu zapojit?

Začni nejprve tím, že se zaregistruješ na našich webových stránkách na adrese `https://fiks.fit.cvut.cz`. Potom si stáhni zadání úloh (nebo využij tuto brožurku), vyřeš je a své řešení nám tamtéž odevzdej.

Typy úloh

Celkem se ve FIKSu můžeš setkat se třemi typy úloh. O který typ úlohy se jedná, je vždy uvedeno u konkrétního zadání úlohy.

Nejčastěji se u nás potkáš s úlohami typu *Rozmysli, popiš a naprogramuj*. U každé úlohy tohoto typu se odevzdává jak popis algoritmu (s odhadem asymptotické složitosti), tak i zdrojový kód řešení problému v tebou zvoleném jazyce (jakýkoliv vyšší programovací jazyk dle tvé volby, například C, Java, Pascal, apod.).

Dalším typem jsou úlohy *Zamysli se*. Tyto úlohy jsou obvykle více teoretické a vyžadují, aby ses nad nimi důkladně zamyslel. Oproti předchozímu typu úloh nemusíš nic programovat, odevzdává se pouze slovní popis řešení problému.

Pokud nemáš rád teoretické úlohy a raději by sis procvičil/a své programátorské umění, pak pro je pro tebe určena kategorie *Odpověz Sfinze*. V úlohách tohoto typu

po tobě nechceme popis algoritmu, je však potřeba vyřešit daný problém a toto řešení pak precizně naprogramovat. Oproti ostatním typům úloh se navíc okamžitě dozvíš, zda je tvé řešení správné, protože ho můžeš okamžitě odevzdat do našeho vyhodnocovacího systému.

Další a podrobnější informace nalezneš na našich webových stránkách.

Fiťácký Informatický Korespondenční Semnář
Ročník 2023/24, 2. kolo

Začátek kola:	28.10.2023 00:00
Termín odevzdání	**8.12.2023 23:59**
Zveřejnění výsůedků:	8.1.2024 23:59
Odevzdávání:	Přes webové rozhraní na `https://fiks.fit.cvut.cz`
Další informace:	`https://fiks.fit.cvut.cz`
	`kontakt@fiks.fit.cvut.cz`
Discord:	`https://discord.com/invite/Pc2w7hp`

Chvíli poté, co poslední uzel potvrdil moji žádost o autorizaci, se na obvodu místnosti, hned vedle na prázdno se stále vysouvající a zasouvající CD mechaniky, začalo divně chovat světlo. Rozhraní prostoru uzlu a zdánlivé pryskyřice po obvodu přestalo čistě lámat obraz. Nejdříve vznikaly artefakty zřejmě způsobené trhlinami v optickém médiu, jako kdybych se díval do rozbitého zrcadla. Jednotlivé fragmenty se stále štěpily, až z rozraní vycházela pouze difundovaná záře, která postupně slábla a slábla, dokud nedosáhla téměř dokonalé černoty.

Z této opticky černé, logicky bílé díry vyšla postava s holí omotanou měděným drátem vypadajícím jako dlouhá cívka: "Vítej v Systému, Frodo. Říkají mi Gandalf. Bylo nám přáno, že injekce nakonec proběhla v pořádku. Snad i ten nedostatek paměti se nám měl přihodit, neboť ukázal tvé výtečné schopnosti, Frodo. S tvojí pomocí bychom konečně mohli uvést Systém do pořádku."

„Ale já jsem jenom pouhý středoškolák."

„Neboj, Frodo, nebudeš na tento úkol sám. Pomůžu ti toto břemeno nést."

Z tmavého portálu vyšly další tři postavy.

„Jmenuji se Aragorn a budu tvůj softwarový inženýr. Ochráním tě před každým nečitelným kódem. Moje IDE i unit testy budou tvým mečem i štítem, kterým se budeme bít s každým design patternem, plus minus jedničkovou chybou a kódem, který by sice měl podle dokumentace dělat nějakou věc, ale vlastně dělá něco úplně jiného."

„Jmenuji se Legolas a budu tvůj teoretický informatik. Každý problém, který se dá převést na grafovou úlohu, zasáhne asymptoticky optimální algoritmus z mého toulce. Mé důkazy jsou elegantní a čisté jako má elfská krev. V mnoha velkých bitvách, které se daly vyjádřit pomocí teorie her, jsem hrdinně dokázal, že máme výherní strategii, budeme-li táhnout první. Konkrétní strategii jsem však nenašel."

„Jmenuji se Gimli a budu tvůj hardwarerář. Nikdo neumí s železem pracovat tak, jak my trpaslíci. Ještě jsem nepotkal problém, který bych neumlátil svojí sekerou – jazykem C. Dokážu disassemblovat každou binárku – zvlášť když má příponu ELF. A netěžím kryptoměny! Nikdy jsem netěšil kryptoměny! A nikdy nebudu těžit kryptoměny! Slyšíš, Legolasi? Přestaň to o mě říkat!"

Úloha č. 1
Konfigurační řetězec

Rozmysli, popiš a naprogramuj! 10 b

Tato úloha se skládá ze dvou částí. Tvým úkolem je napsat program a zároveň zdůvodnit proč funguje.

„Vysvětlí mi konečně někdo, o co tady jde?!“

Gandalf se na Froda s pochopením otočil a začal: „Kdysi dávno se rektorát ČVUT rozhodl, že by měl vytvořit informační systém. Doba to byla těžká. Standardů bylo pomálu, architektury procesorů se měnily jak dneska JavaScriptové frameworky a většinu kódu psali fyzici. Bylo vybráno několik místností, ve kterých měla být zřízena potřebná infrastruktura – začalo se jim říkat uzly – a první inženýři začali dávat dohromady první kusy Systému nevědouce, co se z něj stane.

První software pro evidenci studentů byl napsaný v assembleru. Poté se k němu přidal systém na zkoušky v Algolu a systém na zápočty v Cobolu. Pro interakci mezi systémy někdo zbastlil pár řádků v Lispu a už to tak zůstalo. Pak ale bylo potřeba rozšířit systém pro evidenci studentů. Jelikož do assembleru nikdo nechtěl sahat, rozhodli se využít, co funguje, a dodatečné informace neukládat na magnetické pásce, ale skladovat je pomocí trochu Pascalovského kódu na disketě. Spojení těchto dat zařizoval drobný kód v ML. Docenti z FELu ale nadávali, že je to pomalé, tak si vytvořili alternativní agregátor ve Fortranu. A tak si každý napsal svůj skript na to, co zrovna potřeboval, nejlépe u toho použil výsledky cizího skriptu, a vesele ho přidal do systému.

Systém rostl a uzelní inženýři měli více a více práce. Z uzlů je pomalu už nikdo skoro neviděl vycházet. Jenom někdo ze sekretariátu občas donesl řízek s chlebem (ekvivalent kebabu z minulého tisíciletí) k ventilační šachtě a zaklepal. A lidé si začali všímat, že přinesené dary často způsobily, že jejich problém byl do pár hodin vyřešen, zatímco maily kolegů, co řízky nenosili, zůstávaly nezodpovězené celé týdny. Po nějaké době tak oběti na oltář u ventilační šachty nosili téměř všichni. Požadavky ale byly natolik repetetivní, že uzelní inženýři napsali skripty, které požadavky řešily automaticky za ně – včetně konzumace obětí. Jakým způsobem to Systém dělal, nikdo neví.

Po letech ale přišla rekonstrukce a uzly bylo potřeba přesunout. Uzloví inženýři ale protestovali. Zamkli uzly, všechny maily z rektorátu přesměrovali do spamu a na maily ze stavební firmy nechali odpovídat Systém samotný. Jednoho dne ale přijeli dělníci a rozbourali zdi uzlu. Uvnitř ale našli jen prázdnou místnost, ze které byla cítit pryskyřice. Uzel ale fungoval dál. A když byla rekonstrukce hotová, uzelní inženýři byli vidět, jak vchází a vychází z chodby, kde kdysi bývaly dveře do uzlu a kde je dnes jenom učebna. Občas je někdo několik dnů neviděl a pak najenou vyšli z úplně jiné budovy.

Jak uzelní inženýři se Systémem splývali víc a víc, rozhodli se proto rovnou využít Systém k přesouvání mezi uzly. Napsali pár skriptů a začali se mezi uzly pohybovat, jako kdyby byli jenom packety v síti. Toto přesouvání funguje i dnes, jenom postupem času nabobtnalo a navázet spojení mezi uzly je obtížnější. Aragorn nad tím strávil věčnost. Zbytek ti vysvětlí on.“

Zadání

Na uzlu, se kterým se snažíme navázat spojení, běží n skriptů. K navázání spojení je potřeba odeslat uzlu konfigurační řetězec bitů délky m. Tento řetězec poté dostanou všechny skripty na vstupu. Jenomže jsme si všimli, že u většiny řetězců některé skripty spadnou. Vypozorovali jsme, že pokud změníme i-tý bit řetězce, jistá množina skriptů se začne chovat opačně – pokud padala, tak bude fungovat, a pokud fungovala, tak bude padat. Důležité je, že tato množina zůstane pro daný bit řetězce pořád stejná. Na začátku všechny skripty padají a výchozí konfigurační řetězec sestává ze

samých nul. Vaším úkolem je tedy nastavit konfigurační řetězec tak, aby žádný skript nepadal. Také musíte zjistit, kolik různých takových konfiguračních řetězců existuje.

Vstup

Na prvním řádku vstupu se nachází čísla m, n ($m, n \geq 1$) reprezentující délku konfiguračního řetězce a počet běžících skriptů.

Dále následuje m řádků, kde na každém se nachází nejprve číslo k_i určující pro i-tý bit počet skriptů, který ovlivňuje. Dále na tomto řádku následuje k_i čísel, které určují, které skripty jsou tímto bitem ovlivňovány.

Výstup

Na výstupu napište nejprve číslo x reprezentující počet možných konfiguračních řetězců. Pokud je $x > 0$, vypište na další řádek nějaké řešení v podobě konfiguračního řetězce (tedy m bitů).

Ukázka

Vstup	Výstup

`2 2`
`1 1`
`1 2`
	`1`
	`11`

Máme konfigurační řetězec délky $m = 2$ a $n = 2$ skriptů. První bit ovlivňuje první skript a druhý bit ovlivňuje druhý skript. Pro konfigurační řetězec jsou čtyři možnosti: `00`, `01`, `10`, `11`. Pro první řetězec budou oba skripty padat, pro druhý a třetí zůstane jeden nefunkční a pro čtvrtý řetězec oba skripty fungují. Je tedy jediné řešení `11`.

Vstup	Výstup

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

Zde máme $n = 3$ skripty a délka konfiguračního řetězce je $m = 2$. Zde je ale vidět, že žádná z možností `00`, `01`, `10`, `11` nezabere jako konfigurační řetězec. Pro řetězec `00` budou všechny tři skripty padat. Pro řetězec `01` bude padat první skript, pro řetězec `10` bude padat druhý skript a pro řetězec `11` bude padat třetí skript.

Vstup	Výstup

`3 2`
`1 1`
`2 1 2`
`2 1 2`
	`2`
	`010`

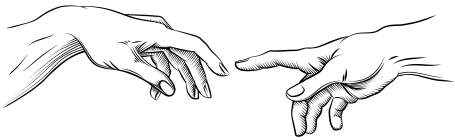
Zde máme $n = 2$ skripty a délku konfiguračního řetězce $m = 3$. Řešení jsou celkem 2: `010` a `001`.

Bodové podmínky

- Za vyřešení problému jsou 2 **body**.

- Za vyřešení problému polynomiálním algoritmem (v m a n) bez počítání počtu možných řešení je 7 **bodů** (tedy stačí umět říct zda řešení existuje či nikoliv a nějaké vypsat, pokud existuje).
- Za plné vyřešení problému polynomiálním algoritmem je 10 **bodů**.

Úloha č. 2
Navazování spojení



Zamysli se! 10 b

Tato úloha je čistě teoretická, tvým úkolem zde není napsat program. Namísto toho si dej záležet na kvalitním slovním popisu, kde mimo jiné jasně zdůvodníš, proč tvůj postup skutečně bude fungovat.

„Dobrá práce, Frodo. Teď, když máme algoritmus na navazování spojení mezi uzly, tak mezi nimi můžeme cestovat mnohem snáze. To se nám teď bude velmi hodit. Víš, Frodo, v Systému se teď dějí strašné věci. Nikdo tomu pořádně nerozumí. Proto moudrý teoretický informatik doc. Elrond, dávný uzelný inženýr, ustanovil Společenstvo paketů – z každé informatické rasy vybral nejzkušenější jedince, kteří mají za cíl problém zdebugovat. A když Správce potkal tebe, vycítil, že jsi ve Společenstvu potřeba. Nyní je ale již čas vyrazit.“

Aragorn spustil můj algoritmus na spočítání konfiguračního řetězce, Gimli zapojil všechny kabely potřebné k zahájení přesunu a Legolas mezitím přemýšlel nad důkazem, že hardwaráři (a hlavně Gimli) by se dali aproximovat stavovým automatem. Gandalf se mezitím shýbl nade mnou a podal mi čepici a boty s podpatkem. „Co to je?“ zeptal jsem se. „Enkapsulace. Je to tvoje nová hlavička a patička, abys mohl být odeslán pomocí síťového protokolu. Hlavně si hlídej v hlavičce hodnotu TTL. Udává tvůj Time To Live. Každým přesunem se sníží o jedna. Je to proto, aby se packet nemohl v Systému zacyklit. To by byl problém.“

Když Aragorn s Gimlim připravili vše potřebné pro přesun a Legolas přidal do svého modelu Gimliho zásobník, Gandalf odeslal konfigurační řetězec a na rozhraní pryskyřice začaly vznikat optické artefakty podobné těm, které jsem viděl před příchodem Společenstva. Tentokrát ale nedocházelo k tmavnutí. Naopak. Prostor přede mnou byl stále světlejší a světlejší, až přes zář nebylo na pryskyřici vidět nic jiného.

Aragorn a Gimli do této bíle, logicky černé díry vešli. Gandalf mi pokynul a já jsem také vykročil. Přes zář jsem nic neviděl. Jenom jsem šel dál a dál. Až jsem se najednou ocitl v místnosti velmi podobné té předchozí – dalším uzlu.

Proces se několikrát opakoval, párkrát jsme se rozdělili a v některých uzlech se potkali. Až Gandalf prohlásil: „Vše tak strašně dlouho trvá. Pokud máme zdebugovat náš problém, je třeba, abychom se po Systému mohli pohybovat rychleji. Postavme dostatek spojení, abychom se byli schopní dostat z libovolného uzlu do libovolného jiného bez stavění dalších. Legolasi, to bys mohl zvládnout vymyslet.“

Legolas, očividně rozladěn tím, že musel ke svému zásobníkovému automatu přidat pásku, aby konečně Gimliho vymodeloval, se pustil do práce.

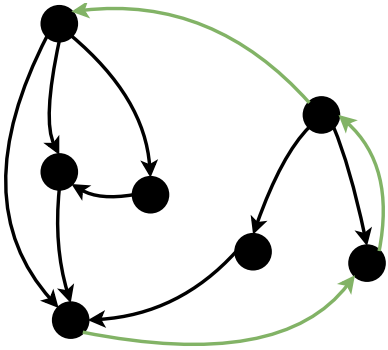
Všiml si, že mezi všemi dosud navázanými jednosměrnými spojeními se dosud nevytvořil žádný cyklus.

Popis úkolu

Budete pracovat s orientovaným grafem. V orientovaném grafu jsou hrany mezi vrcholy vždy jednosměrné. Pokud tedy existuje hrana z prvního vrcholu do druhého, umíme cestovat pouze tímto směrem. Abychom se uměli přímo vrátit, museli bychom přidat další, přesně opačně orientovanou hranu.

Vaším cílem je umět se po grafu libovolně pohybovat. Chcete, aby vždy existovala cesta mezi každými dvěma vrcholy, a to oběma směry. Toho umíte dosáhnout přidáváním nových hran do

grafu. Graf se kterým budete pracovat nebude obsahovat žádné cykly. V orientovaném grafu to znamená, že v jakémkoli vrcholu začnete a jakýmikoli hranami se budete pohybovat, nikdy neskončíte v počátečním vrcholu.



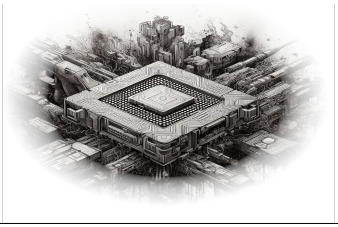
Na obrázku můžete vidět příklad doplnění hran do acyklického orientovaného grafu tak, že existuje cesta tam i zpět mezi každým párem vrcholů.

Navrhněte strategii přidávání hran, která splní zadanou podmínku. Popište, které hrany budete do grafu přidávat a odhadněte jejich počet. Popište, na čem tento počet závisí. Při návrhu strategie se soustřeďte na co nejmenší počet přidaných hran.

Rozšíření zadání

Omezení na pouze acyklické grafy je dost přísné. Upravte tedy vaši strategii, aby fungovala i v případě, že orientovaný graf není acyklický. Popište změny ve vaší strategii a odhadněte, jak se změní vaše tvrzení o počtu přidaných hran a vlastnostech grafu na kterých tento počet závisí ve srovnání se základním zadáním.

Úloha č. 3
Bitva v paměti



Odpověz sfinze! 10 b

Tato úloha je vyhodnocována automaticky. Je potřeba, aby výstup programu ***přesně*** *korespondoval se specifikací výstupu níže. Jak odevzdávat tento typ úloh se můžeš dočíst na webových stránkách FIKSu pod záložkou „Jak řešit FIKS“.*

Se seznamem všech potřebných spojení jsme se dali do práce. Cestovali z uzlu do uzlu a vytvářeli vše potřebné. S Gimlim a Gandalfem jsme v jednom uzlu čekali na ostatní. Konečně se před námi vytvořila černá, logicky bílá díra.

„No konečně, že jste tady,“ zvolal Gandalf. Místo Aragorna a Legolase ale z černé, logicky bílé díry vyskočila vyhublá postava v šedém děravém plášti. Pod pláštěm bylo staré triko s nějakým kódem, z pod kterého lezly žebra. Některá ovšem chyběla a byla nahrazena žebry procesorového chladiče. Ruce volně splývaly podél těla a místo některých prstů byly šroubováky, kleště a drát, který připomínal oko pájky. V oblasti hlavy byla velká díra, neboť místo pravé mozkové hemisféry byla zasazena patice k procesoru. Levé oko vypadalo spíše jako rozbitá čočka kamery a místo pusy byla disketová mechanika, ze které koukaly zbytky řízku.

Gandalf pozvedl svoji hůl a mocným zmáčknutím tlačítka vyslal k postavě magnetický impulz. Postava se podlomila v kolenech a s rachotem spadla na zem a roztříštila se. Z jednotlivých úlomků se rozeběhly bugy – malé kusy PCB, které vypadaly jak části všemožných konektorů. Hlavně ISA, PCI a USB. Bugy se rozutekly po místnosti a vložily se do nepoužitých portů vší elektroniky, která se v místnosti nacházela.

„Rychle, nesmí se dostat k paměti,“ pronesl klidně Gandalf. Bylo bohužel už pozdě. Když se podíval do paměti uzlu, byla plná náhodného kódu, který nešel vymazat. „Je potřeba vytvořit vlastní, který náhodný kód odstraní.“

Tvým cílem bude napsat program, který je nejen schopen přežít, ale i zničit ostatní programy – budeš tedy bojovat jak s předpřipravenými programy, tak s programy svých kolegů. Nebude to ale tak jednoduché, jak by se mohlo zdát. Tvůj program bude běžet na počítači, který má několik specifických vlastností.

Narozdíl od běžného programování tu je pár drobných rozdílů.

- Paměť pro data a kód je společná. To znamená, že můžeš přepsat svůj vlastní kód. Stačí zapsat do paměti na správné místo správné byty.
- Na počítači běží několik (až 8) procesů, ty ale mají společnou paměť. To znamená, že můžeš přepsat kód jiného procesu pokud zjistíš, kde běží.
- Paměť je omezená. Tvůj kód se navíc musí vejít do 256 instrukcí.
- Kromě paměti (která je sdílená pro všechny programy) má každý program své vlastní registry. Registr je taková proměnná, která je ale čistě pro tvůj program a nikdo jiný ji nemůže číst ani do ní zapisovat. Ve všech registrech je na začátku nula. V prvním registru (R0) je vždy nula a

není povolené do něj zapisovat. Registry nemají jména, ale čísla, indexuje se od nuly. Registrů je 6 a mají velikost 32 bitů.

- Chování při přetečení (tedy pokusu uložit větší číslo, než se vejde) jak paměti tak registrů není definované – může se stát cokoliv.

Tvůj program běží, dokud nenastane některé z následujícího:

- Vykonal jsi neplatnou instrukci. Tedy něco, co není v tabulce níže. Toto se může snadno stát, pokud jsi neopatrně zapisoval do paměti a špatně sis přepsal instrukce.
- Do neplatných instrukcí spadá i přístup mimo rozsah paměti.
- Vykonal jsi instrukci „bomba“ ve chvíli, kdy zbývala na odpočtu nula.
- Zacyklil jsi se.

Tvým cílem je běžet déle, než programy ostatních. (Tedy jednou z možných metod přežití může být zkusit zničit ostatní programy.)

Registry

Číslo	Popis
0	Vždy obsahuje nulu. Do něj nelze zapisovat.
1	Druhý registr.
2	Třetí registr.
3	Čtvrtý registr.
4	Pátý registr.
5	Šestý registr.

Paměť

Paměť má předem neznámý rozsah. Lze ji indexovat klasicky jako u normálního pole, tedy lze číst například z `mem[10]` nebo zapisovat na `mem[11]`.

- Vždy obsahuje na začátku $256 * 4$ bytů, které „nepatří“ žádnému hráči. Najdeš tam také speciální adresy, ze kterých jde jenom číst:

Adresa	Popis
42 (0x2a)	Obsahuje adresu nejbližšího programu[1].
43 (0x2b)	Obsahuje adresu druhého nejbližšího programu.

- Poté pro každého hráče obsahuje dalších $256 * 4$ bytů, kde se nachází jeho počáteční program.
- Nevíš, v jakém pořadí jsi ani kolik programů běží - tedy jestli jsou před tebou nějaké programy, nebo ne.

[1] Počítáno od místa, kde zrovna vykonáváš kód.

Instrukce

Každá instrukce je 4 byty (32 bitů) dlouhá. První byte je kód instrukce, další tři jsou argumenty.

Všechna čísla ve sloupečku „Kód“ uvedená v této tabulce jsou hexadecimální. Tedy například číslo 69 je ve skutečnosti `0x69`, což je v desítkové soustavě 105. Dvojice čísel (v tomto případě tedy třeba 69 je jeden byte = 8 bitů.)

Ve sloupečku kód je předpis instrukce. Když je někde číslo, je pevně dané. Otazníky znamená, že hodnota daných bitů může být libovolná a nezáleží na ní. Zápis typu `reg1(4b)` znamená, že následují 4 bity, které určují registr, který instrukce použije. Zápis typu `imm(16b)` znamená, že následuje 16 bitů, které určují číslo (vyloženě číselný argument), které instrukce použije.

Výrazem `pc` se myslí adresa právě vykonávané instrukce. Tato hodnota se inkrementuje po každém vykonání instrukce, pokud neproběhla speciální instrukce, která hodnotu `pc` mění - například skoky.

Neboj se, pokud to zatím nedává smysl, z příkladů bude vše jasné.

Kód	Instrukce	Popis	Příklad
`69 ??(8b) ??(8b) ??(8b)`	`NOP`	Nic neudělá.	`69 de ad be`
`01 reg1(4b) reg2(4b) imm(16b)`	`ADD`	`reg1 += reg2 + imm`	`01 12 00 04` - k hodnotě prvního registru přičte hodnotu registru 2 plus 4 a výsledek uloží do prvního registru.
`02 reg1(4b) reg2(4b) imm(16b)`	`SUB`	`reg1 -= reg2 + imm`	`02 34 00 A4` - od hodnoty třetího registru odečte (hodnotu registru 4 plus `0xA4` (tedy 164)) a výsledek uloží do třetího registru.
`03 reg1(4b) reg2(4b) imm(16b)`	`MUL`	`reg1 *= reg2 + imm`	`03 50 FF EE` - hodnotu pátého registru vynásobí (hodnotou registru 0 plus `0xFFEE`) a výsledek uloží do páteho registru.
`05 reg1(4b) reg2(4b) imm(16b)`	`LOAD`	`reg1 = mem[reg2+imm]`	`05 13 00 00` - hodnotu prvního registru nastaví na hodnotu v paměti na adrese (hodnota třetího registru plus 0).
`06 reg1(4b) reg2(4b) imm(16b)`	`STORE`	`mem[reg2+imm] = reg1`	`06 04 00 0A` - hodnotu v paměti na adrese (hodnota čtvrtého registru plus 10) nastaví na hodnotu nultého registru (tedy zapíše nulu).
`07 reg1(4b) reg2(4b) imm(16b)`	`MOV`	`reg1 = reg2 + imm`	`07 12 01 28` - hodnotu prvního registru nastaví na hodnotu druhého registru plus `0x128`.
`10 reg1(4b) reg2(4b) imm(16b)`	`JUMP`	`if (reg1 == reg2) pc += imm`	`10 12 00 04` - pokud je hodnota prvního registru rovna hodnotě druhého registru, skočí o čtyři instrukce dopředu.
`11 reg1(4b) reg2(4b) imm(16b)`	`REVJUMP`	`if (reg1 == reg2) pc -= imm`	`11 00 00 01` - pokud je hodnota nultého registru rovna hodnotě nultého registru, skočí o 1 instrukci dozadu.

12 reg1(4b) reg2(4b) imm(16b)	LTJUMP	`if (reg1 < reg2) pc += imm`	12 12 00 04 - pokud je hodnota prvního registru menší než hodnota druhého registru, skočí o čtyři instrukce dopředu.
13 reg1(4b) reg2(4b) imm(16b)	REVLTJUMP	`if (reg1 < reg2) pc -= imm`	13 50 00 08 - pokud je hodnota páteho registru menší než hodnota nultého registru, skočí o osm instrukcí dozadu.
14 reg1(4b) reg2(4b) imm(16b)	NEQJUMP	`if (reg1 != reg2) pc += imm`	14 12 0A 00 - pokud je hodnota prvního registru nerovna hodnotě druhého registru, skočí o `0xA00` instrukce dopředu.
15 reg1(4b) reg2(4b) imm(16b)	REVNEQJUMP	`if (reg1 != reg2) pc -= imm`	15 12 00 04 - pokud je hodnota prvního registru nerovna hodnotě druhého registru, skočí o čtyři instrukce dozadu.
20 reg1(4b) ??(4b) imm(16b)	SETIMMLOW	`reg1[low] = imm`	20 1F 00 04 - nastaví dolních 16 bitů prvního registru na hodnotu 4.
21 reg1(4b) ??(4b) imm(16b)	SETIMMHIGH	`reg1[high] = imm`	21 1C 00 04 - nastaví horních 16 bitů prvního registru na hodnotu 4.
42 imm1(8b) imm2(16b)	TELEPORT	Zmrazí program. Pokud jiný program také spustí instrukci TELEPORT, tyto dva programy si okamžitě prohodí místa. Pokud žádný program TELEPORT nezavolá po dobu imm1 instrukcí, program skočí o imm2 instrukcí dopředu.	42 10 00 00 - zmrzne program na 16 instrukcí. Pokud nikdo nezavolá teleport, skočí o 0 dopředu (tedy na tu samou instrukci a zavolá teleport znovu).
50 imm(16b) ??(8b)	BOMB	Každým spuštěním se hodnota imm sníží o 1 – dojde k přepsání instrukce v paměti. Pokud byla hodnota 0, nelze snížit a program bude ukončen.	50 00 00 CD - program zanikne hned po spuštění této instrukce.

Například, následující kód se zacyklí:

```
69 00 00 00  ; NOP
11 00 00 01  ; REVJUMP reg0, reg0, 0x0001
```

Odevzdávání

Odevzdávat budeš zdrojový kód svého programu ve formátu značeném výše. Jak sis už mohl všimnout, `;` je komentář, vše až do konce řádku se ignoruje. Prázdné řádky a mezery se také ignorují. Na každém

jiném řádku je potřeba mít přesně 8 číslic, které dohromady dají (hexadecimálně) 32 bitů každé instrukce.

Odevzdávání této úlohy je složitější kvůli způsobu vyhodnocování. Jelikož je úloha komplexní, je zapotřebí dát účastníkovi podrobnou zpětnou vazbu. Podrobné informace o běhu tvého programu půjde zjistit na fiks.soptik.tech.

Na odevzdávací web se registruješ tím, že odešleš soubor, který bude mít na prvním řádku `email:<fiks email>` a na druhém `heslo:<moje heslo>`. Email musí být totožný jako ten, pod kterým máš zaregistrovaný svůj FIKS účet. **Nastav si jiné heslo, než používáš kdekoliv jinde!**

Po odevzdání je možné se svým emailem a zvoleným heslem přihlásit na web fiks.soptik.tech, kde si můžeš zobrazit výsledky svých odevzdání a zároveň stáhnout záznam toho, co se stalo. U „ostrých“ sfinga odevzdání budeš mít k dispozici pouze omezený log, který bude zobrazovat pouze tvoje akce – je to z toho důvodu, abys nemohl tak jednoduše kopírovat kód ostatních programů.

Web fiks.soptik.tech po přihlášení umožní i nahrát program přímo. Takovýto program bude spuštěn s jednoduchým protivníkem, který se stále jenom cyklí. Dostaneš kompletní výpis celého stavu prostředí, abys mohl debugovat svoje programy.

Pokud chceš debugovací prostředí ve kterém dělá protivník něco jiného, přepiš jeho kód sám :)

Pokud bys měl s odevzdáním nebo webem jakékoliv problémy, ozvi se na FIKS discordu nebo na `petr.stastny@fit.cvut.cz`.

Hodnocení

Body se udělují následovně (uvidíš je na hodnotícím webu):

- 0.01 bodu za úspěšné nastavení hesla
- celkem 1 bod za to, že odevzdáš validní program
- celkem 2 body za to, že porazíš protivníka, který se pouze cyklí.
- celkem 3-5 bodů za to, že porazíš protivníka, který na tebe bude aktivně útočit. Můžeš dostat i nižší počet bodů, pokud se mu zvládneš dost dlouho bránit, ale nepodaří se ti ho porazit.

Pokud dosáhneš alespoň 2 bodů, budeš zařazen do bitvy s ostatními účastníky.

Jednou týdně proběhne souboj mezi všemi účastníky, jejichž poslední odevzdaný program získal alespoň 2 body. Pokud se souboje budou účastnit alespoň 4 účastníci, dostaneš až 2 extra body v závislosti na svém výkonu. (Tyto body z různých týdnů se nesčítají.)

Po uzavření úlohy proběhne finální souboj, na základě kterého bude rozděleno až 5 bodů.

Maximum bodů, kolik můžeš získat, je 10.

Získané body uvidíš na portálu fiks.soptik.tech, do sfingy budou nahrané později po konci kola.

Příklad

Následující program spočítá faktoriál čísla 4 (vstup je uložen v registru 1, výstup bude v registru 2).

```
; reg1 = 4
; reg2 = 1
; while (reg1 != 0) {
;   reg2 *= reg1
;   reg1--
; }
; ; reg2 = 24
```

```
; Move 4 to register 1
07 10 00 04 ; reg1 = reg0 + 4 (reg 0 is always zero)
            ; This could also be achieved by using SETIMMLOW.
; Move 1 to register 2
07 20 00 01 ; reg2 = reg0 + 1
; Loop start
; if reg1 is zero, jump to end
10 10 00 04 ; if (reg1 == reg0) pc += 4 (skip the whole loop here)

  ; reg2 *= reg1
  03 21 00 00 ; reg2 *= reg1
  ; reg1--
  02 10 00 01 ; reg1 -= 1
  ; Jump to loop start
  11 00 00 03 ; If reg0 is equal to reg0, jump three instructions backwards
     ; (to loop start)
; LOOP END

69 00 00 00 ; NOP

; reg2 is now 24
```