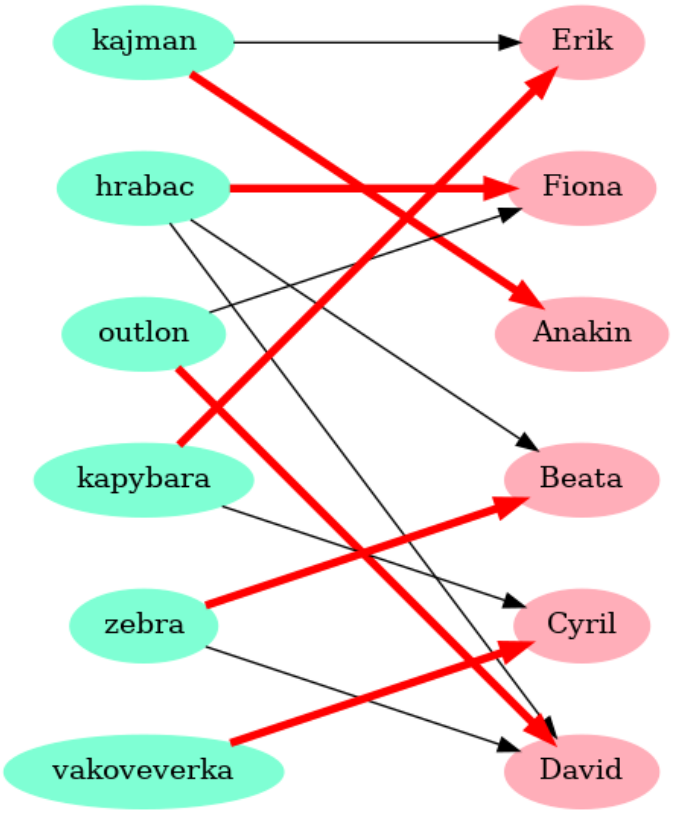
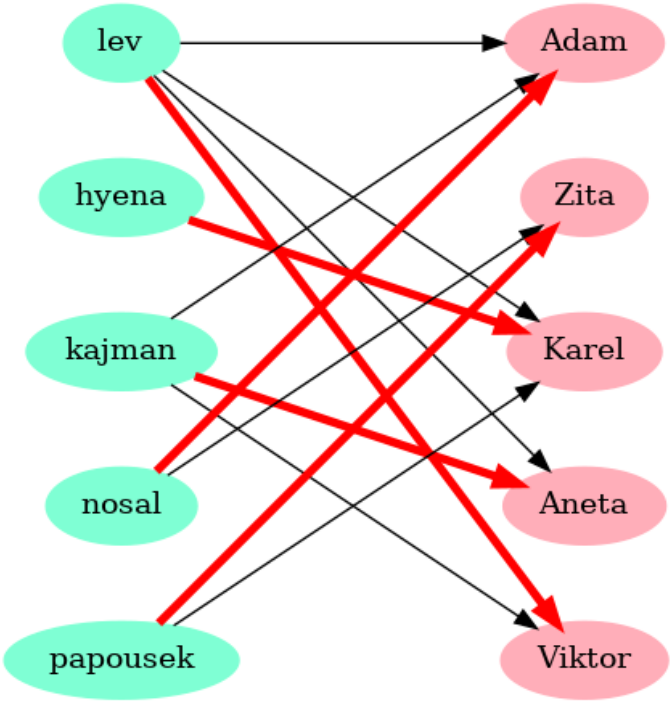
Řešení úlohy č. 2
Sponzoři

V úloze Sponzoří jste dostali zadané dvě množiny (množinu sponzorů a množinu zvířat), které jste mezi sebou měli pospojovat tak, aby co nejvíce zvířat mělo sponzora, za podmínky, že si každý sponzor může dovolit sponzorovat maximálně jedno zvíře. Tento problém není v teorii grafů nic nového, vstup si převedeme na tzv. bipartitní graf *(graf tvořený dvěma množinami (paritami) vrcholů, kde mezi vrcholy z jedné parity nevedou žádné hrany)*, kde jednu paritu bude tvořit množina zvířat a druhou množina sponzorů. Izolované vrcholy *(vrcholy, do kterých nevede žádná hrana)*, tedy v našem případě zvířata, která nechce sponzorovat žádný sponzor, z grafu rovnou můžeme odstranit. V tomto grafu pak hledáme tzv. maximální párování *(největší podmožinu hran takovou, že každý vrchol grafu bude patřit nejvýše do jedné hrany z této podmožiny)*.



Obrázek 2.1 Maximální párování (červeně)

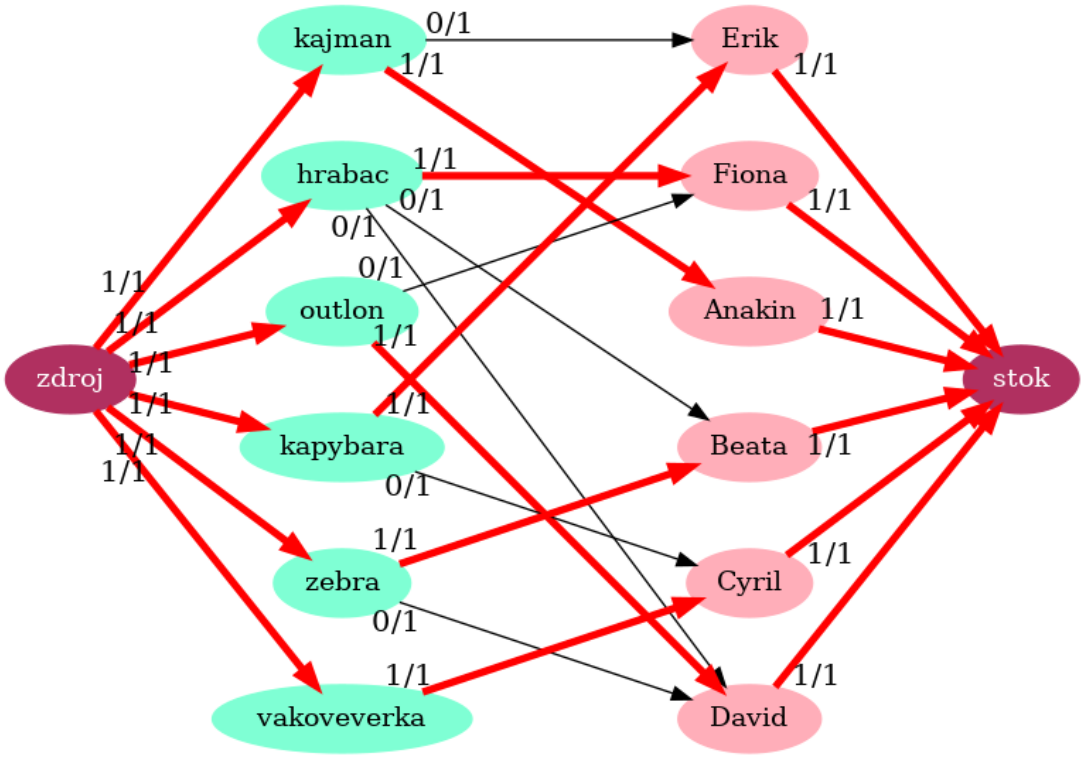
Hodně z vás se úlohu snažilo řešit pomocí nějakého hladového algoritmu *(druh algoritmu, kdy v každé iteraci vybereme možnost, která se aktuálně jeví jako optimální, a doufáme, že tak dojdeme i k celkově optimálnímu řešení)*. Například vás mohlo napadnout postupně vybírat sponzory a zvířata tak, aby daný sponzor vždy chtěl sponzorovat co nejméně zvířat a zároveň jím vybrané zvíře chtělo sponzorovat co nejméně ostatních sponzorů. I když takový algoritmus vypadá velmi nadějně a pro spoustu vstupů vrátí správný výsledek, tak bohužel, hladový přístup nám u této úlohy, možná trochu proti intuici, nepomůže. Vždycky budeme schopni najít nějaký vstup, pro který nedostaneme správnou odpověď. Nemáme žádný důkaz, proč by takový algoritmus měl být korektní. Tento konkrétní algoritmus nám maximální párování nemusí vrátit například pro takovýto vstup:



Obrázek 2.2 Vstup, pro který hladový algoritmus nezafunguje

Jako první můžeme vybrat dvojici nosál-Zita, tím však pro papouška i hyenu zbyde už jen jeden potenciální sponzor, Karel, a maximální párování tak už nedostaneme.

A jak se tedy taková úloha dá řešit správně a rychleji než brute-forcem všech možností? Graf si můžeme převést na tzv. síť. Přidáme mu zdrojový a stokový vrchol, zdrojový vrchol spojíme hranou se všemi vrcholy z první parity, stokový vrchol se všemi vrcholy z druhé parity a hrany zorientujeme směrem od zdrojového vrcholu do stokového vrcholu. Všem hranám nastavíme nějakou maximální kapacitu, v našem případě kapacitu 1. Následně nás bude zajímat maximální tok v této síti. Tok v síti si můžeme představit jako vodu tekoucí vodovodním potrubím. Ze zdrojového vrcholu do sítě pustíme vodu a zajímá nás, kolik jí doteče do stokového vrcholu, za podmínky, že každou trubkou (hranou) může protéct maximálně tolik vody, kolik je její kapacita.



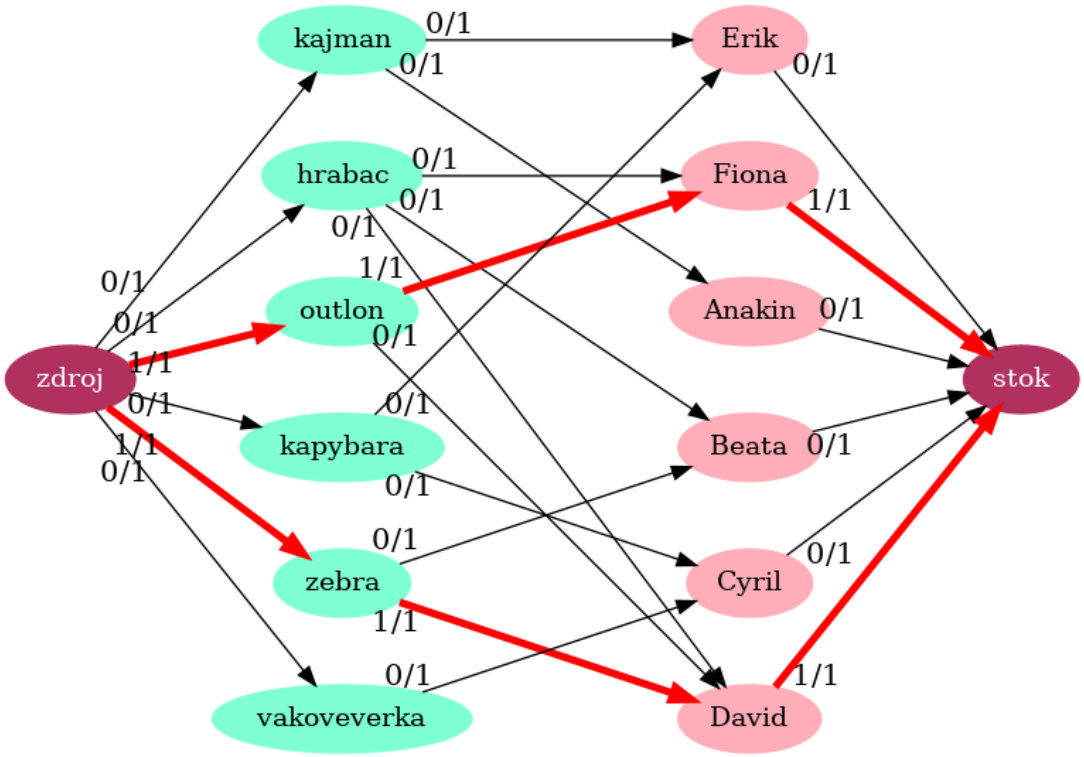
Obrázek 2.3 Maximální tok v síti

Jakmile v naší síti najdeme maximální tok, vytvoříme množinu hran X, do které dáme ty hrany mezi první a druhou paritou, po kterých teče jednotkový tok. Díky tomu, že do každého vrcholu z první parity vede ze zdroje pouze jedna hrana a z každého vrcholu z druhé parity vede pouze jedna hrana do stoku, musí každým vrcholem v obou paritách protékat tok maximálně 1. Díky tomu víme, že se nám v X neopakují žádné vrcholy a splňujeme tak podmínku pro párování. Jelikož si velikost toku a velikost jemu příslušejícího párování odpovídají, je toto párování maximální možné. Větší tok a tudíž i větší párování nemůžeme dostat.

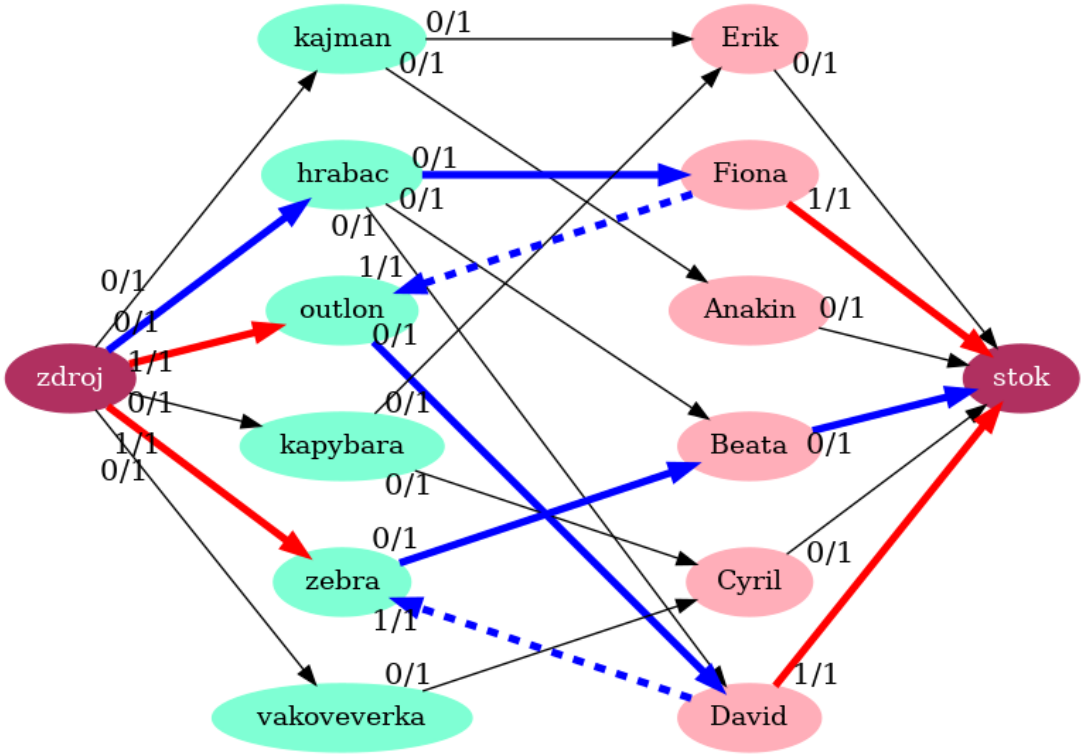
A jak tedy takový maximální tok najdeme? Například pomocí Ford-Fulkersonova algoritmu:

1. Všem hranám nastav nulový tok
2. Dokud existuje nějaká zlepšující cesta, tak podél ní vylepši aktuální tok
3. Vrať získaný tok jako maximální

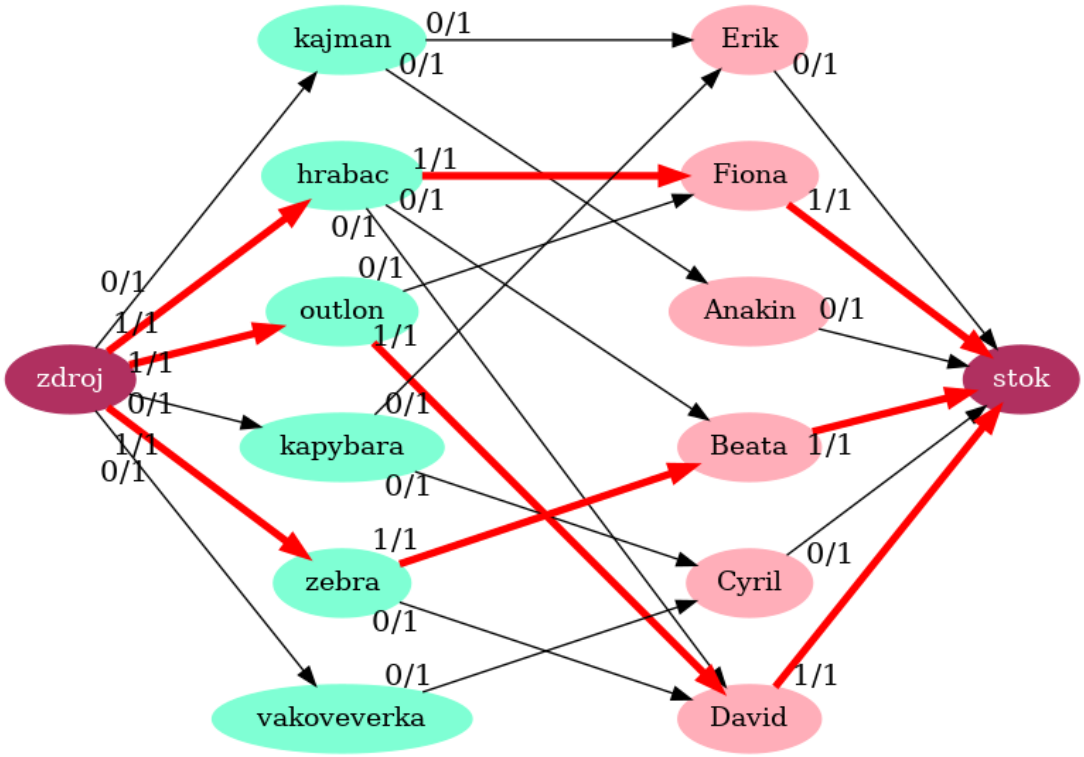
Zlepšující cesta je v našem případě taková cesta ze zdroje do stoku, která může obsahovat hrany vedoucí po i proti směru, a kde na hranách po směru teče tok 0 a na hranách proti směru tok 1. Tok podél ní vylepšíme tak, že hranám po směru zvýšíme tok na 1 a hranám proti směru snížíme tok na 0. Tím vždy vylepšíme aktuální tok (a také párování) o 1.



Obrázek 2.4 Aktuální tok



Obrázek 2.5 Zlepšující cesta (modře)



Obrázek 2.6 Tok vylepšený podél zlepšující cesty

Podrobný důkaz korektnosti a konečnosti Ford-Fulkersonova algoritmu najdete v Průvodci labyrintem algoritmů.

Nejvíce paměti v našem programu spotřebujeme na uložení sítě. Tam ukládáme $\|V\|$ vrcholů našeho bipartitního grafu, zdrojový a stokový vrchol, hrany ze zdrojového a do stokového vrcholu, kterých máme $\|V\|$, a $\|E\|$ hran našeho bipartitního grafu. Každou hranu si v našem programu ukládáme dvakrát (po směru a proti směru). Paměťová složitost je tedy $O(\|V\| + 2 + 2 \cdot (\|V\| + \|E\|)) = O(\|V\| + \|E\|) = O(\|E\|)$, jelikož díky počátečnímu odstranění izolovaných vrcholů víme, že $\|E\| \geq \frac{\|V\|}{2}$, a tedy $\|V\| = O(\|E\|)$. Počet sponzorů na vstupu označíme $\|S\|$ a počet zvířat $\|Z\|$. Víme, že $\|V\| \leq \|S\| + \|Z\|$ a $\|E\| \leq \|S\| \cdot \|Z\|$. Horní meze dostaneme, kdybychom při tvorbě bipartitního grafu neodstraňovali žádné izolované vrcholy a ze všech zvířat by v bipartitním grafu vedla hrana do všech sponzorů. Paměťovou složitost tedy můžeme ekvivalentně vyjádřit jako $O(\|S\| \cdot \|Z\|)$.

Pro načtení a uložení vstupu dostaneme časovou složitost $O(\|Z\| + \|V\| + \|E\|) = O(\|Z\| + \|E\|)$, kdy nejdříve načítáme všechna zvířata na vstupu (i izolované vrcholy) a pak nám opět nejvíce času zabere pro každého sponzora proiterovat a uložit zvířata, která chce sponzorovat. Všechna data si ukládáme do nafukovacích polí nebo do hašovacích tabulek, vkládání a přístup k prvkům přes klíč/index tedy máme amortizovaně konstantní.

Jelikož v každé iteraci zvětšíme tok o 1, je počet iterací Ford-Fulkersonova algoritmu roven velikosti maximálního toku, tedy $O(\|V\|)$. Když pro hledání zlepšující cesty využijeme BFS nebo DFS se složitostí $O(\|V\| + \|E\|)$, tak celkově pro celý Ford-Fulkersonův algoritmus dostaneme složitost $O(\|V\| \cdot (\|V\| + \|E\|)) = O(\|V\|^2 + \|V\| \cdot \|E\|) = O(\|V\| \cdot \|E\|)$. Opět využíváme, že $\|V\| = O(\|E\|)$.

Seřazení potenciálně sponzorovaných zvířat pro vypsání ve správném pořadí nám zabere $O(\|V\| \cdot \log \|V\|)$.

Iterace přes tato zvířata a hledání párování, tedy hledání sousední hrany s jednotkovým tokem, nám zabere $O(\|V\| + \|E\|) = O(\|E\|)$, jelikož žádnou hranu neprocházíme víckrát.

Dostáváme tak celkovou časovou složitost programu $O(\|Z\| + \|E\| + \|V\| \cdot \|E\| + \|V\| \cdot \log \|V\| + \|E\|) = O(\|Z\| + \|V\| \cdot \|E\|)$.

Existují i efektivnější přístupy, hodně z vás úlohu řešilo například pomocí Hopcroft-Karpova algoritmu, který v každém kroku algoritmu najde více různých zlepšujících cest a párování tak umí najít dokonce v čase $O(\sqrt{\|V\|} \cdot \|E\|)$.