FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV-Internet, počítačová síť

KIV/PT – Semestrální práce

Vypracovali: Pavel Habžanský, Jakub Mikeš

Standardní zadání semestrální práce pro PT 2016/2017

Zadání je určeno pro dva studenty. Práce zahrnuje dvě dílčí části - vytvoření funkčního programu diskrétní simulace a napsání strukturované dokumentace.

Zadání:

Firma KIV-Internet vlastní rozsáhlou počítačovou síť (v řádech tisíců uzlů). Počítačová síť slouží k přenosu dat mezi jednotlivými stroji a jejich přístupu k internetu. Síť využívá různé technologie od wifi po optické spoje. Kvalita a rychlost jednotlivých uzlů je dána typem spoje. V současné době se provoz po síti řídí protokolem OSPF, který ale není úplně optimálním řešením. Cílem Vaší práce je proto navrhnout protokol pro optimalizaci datových toků sítí.

Síť lze reprezentovat jako mesh topologii pomocí grafu. Získáme tím graf s ohodnocením jednotlivých hran (dvojice čísel), které nám určuje maximální propustnost a stabilitu daného spoje. Maximální propustnost udává, kolik můžeme spojem poslat Mbit/s (1s = nejmenší krok simulace propustnosti sítě). Tato hodnota bude v rozsahu 1 – 10000 Mbit/s. Druhá hodnota vyjadřuje spolehlivost a bude v rozsahu 0 - 1. Hodnota 1 udává maximální spolehlivost, kdy nedochází k žádné ztrátě packetů, jakákoliv hodnota menší než 1 udává, kdy začne docházet ke ztrátovosti 50% při zatížení. Například 0.9 znamená, že při vytížení linky nad 90% může dojít ke ztrátě 50% všech dat, které v dané vteřině přenášíte, tj. budou se muset odeslat znovu. Pravděpodobnost ztráty bude také 50% (realizujte generátorem náhodných čísel).

Na uzlech sítě jsou „chytré“ routery se síťovým smart-stackem (implementujete Vy), který dokáže uchovávat až 100Mb dat pro případ, že do uzlu přiteče více dat, než můžete aktuálně odeslat. Router má inteligentní routovací systémem, který se může měnit v čase (implementujete Vy). Routery mohou ovlivnit jak cestu, tak počet posílaných dat sousednímu uzlu. V případě přetečení stacku v cílovém uzlu dojde ke ztrátě celého balíku dat a musí se poslat znovu.

Vstupy:

Graf bude popsán souborem, který bude mít následující strukturu na každém řádku:
ID_UZLU_1 – ID_UZLU_2 – max_propustnost – chybovost
Pozor, řádky nemusí jít za sebou v pořadí uzlů!
Výsledný graf bude vždy souvislý.

Simulace:

Pro simulaci bude použit generovaný seznam požadavků ve tvaru:
ČAS – ID_ZDROJ – ID_CÍL – VELIKOST_DAT

Vytvoření funkčního programu:

Načtěte graf ze vstupního souboru a vytvořte strukturu pro „routery“ – uzly sítě **(10b.)**, zvolte a implementujte vhodné datové struktury pro reprezentaci vstupních dat, důsledně zvažujte paměťovou náročnost zvolených struktur a časovou náročnost algoritmů pro následovné výpočty **(10b.)**,
proveďte základní simulaci pro přenos dat, vypište, jak jsou data přenášena (jakou cestou/cestami) průběh simulace (všechny důležité hodnoty, včetně celkové ztrátovosti Vašeho řešení) zapisujte na obrazovku a do souboru **(10 b.)**.

Výše popsaná část bude váš minimální výstup při kontrolním cvičení cca v polovině semestru.

Vytvořte prostředí pro snadnou obsluhu programu (sledování uzlu, zrušení hrany – simulace výpadku) - **(6b.)**
umožněte sledování (za běhu simulace) aktuálního stavu sítě, vytížení sítě, atp. **(6b.)**
, vytvořte statistiky pro přenesená data sítí a uložte je do souboru – **(8b.)**,
vytvořte dokumentační komentáře ve zdrojovém textu programu a vygenerujte programovou dokumentaci (Javadoc) **(10b.)**,
vytvořte kvalitní dále rozšiřitelný kód – pro kontrolu použijte softwarový nástroj PMD (více na http://www.kiv.zcu.cz/~herout/pruzkumy/pmd/pmd.html), soubor s pravidly pdmrules.xml najdete na portálu v podmenu Samostatná práce **(10b.)**

- mínus 1 bod za vážnější chybu, při 6 a více chybách nutno opravit
- mínus 2 body za 10 a více drobných chyb

V rámci dokumentace:

- připojte zadání **(1b.),**
- popište analýzu problému **(6b.),**
- popište návrh programu (např. UML diagram) **(6b.),**
- vytvořte uživatelskou dokumentaci **(5b.),**
- zhodnoťte celou práci, vytvořte závěr **(2b.).**

Analýza problému

Jelikož jde podle zadání o počítačovou síť, evidentně se zde řeší grafový problém posílání datových balíčků mezi routery. Pro takový problém existuje několik možností řešení.

Reprezentace grafu

Pro reprezentaci grafu se nevíce nabízí reprezentace seznamem sousednosti nebo maticí sousednosti. Díky možnosti okamžitého přístupu k hraně mezi routery jsme se rozhodli pro matici sousednosti. Ta má samozřejmě rozměry počet_routerů * počet_routerů. Nicméně, abychom mohli přistoupit k hraně potřebujeme index routerů, ty udržujeme v tabulce, kde klíč je objektová reprezentace našeho routeru a hodnota je index daného routeru v matici sousednosti hran. Máme-li tedy cestu reprezentovanou s pomocí spojového seznamu routerů, můžeme jednoduše najít hranu mezi dvěma sousedícími routery.

Matice sousednosti nám také pomáhá poměrně jednoduše reprezentovat neorientovanost tohoto grafu. V případě, že vytvoříme hranu vedoucí z uzlu reprezentovaný v matici indexem 2 a uzlu s indexem 3, uložíme ji do matice na souřadnice [2,3]. Následně tak vytvoříme i druhou hranu, opačně orientovanou, a uložíme ji do matice na souřadnice [3,2].

Vstupní data

Vstupní data načítáme ze dvou souborů, jednak ze souboru hran, následně soubor požadavků. Nejprve načítáme soubor hran, z každého řádku (každý řádek ve formátu specifikovaný v zadání) v souboru vytvoříme hranu a přidáme ji do seznamu. Z tohoto seznamu následně vytváříme tabulku s routery a jejich indexy v matici sousednosti (viz **Reprezentace grafu**). Pak už jen z hran a uzlů vytvoříme matici sousednosti a vytvoříme graf právě z této matice sousednosti a tabulky indexů.

Požadavky načítáme z druhého souboru. Z každé řádky vytvoříme požadavek, který následně ukládáme do seznamu. Po načtení celého souboru předáme seznam grafu, který s ním dále pracuje.

Vyřizování požadavků

Jedna z možností by byla vyřízení všech požadavků najednou, nicméně to odporuje zadání, kde je specificky řečeno, že požadavky se vyskytují v různé časy. To však nevylučuje možnost vyřízení vícera požadavků najednou, protože některé mohou být poslány ve stejný čas. Tento problém je řešen tak, že každý datový baíček si pamatuje čas svého vyřízení.

Hledání cest

V zadání je specificky řečeno, že je možné v průběhu posílání datových balíčků měnit cestu. Jednou z možností je případ, kdy si každý balíček bude pamatovat všechny cesty do všech routerů. Tato možnost se jeví jako velmi neúsporná, jelikož by se takové cesty musely přepočítávat na každém uzlu zvlášť. Proto jsme se uchýlili k možnosti, kdy si každý router (uzel v grafu) pamatuje všechny cesty do všech ostatních routerů. Tyto cesty jsou vypočítány pomocí prohledávání grafu do hloubky (DFS) a jsou seřazeny sestupně podle součtů propustností spojů mezi routery.

K reprezentaci cesty byla vytvořena patřičná datová struktura pomocí spojového seznamu routerů.

Uložení cest
Jak bylo řečeno, každý router si pamatuje cesty do všech ostatních uzlů. Tyto cesty jsou uloženy na routeru v tabulce, kde klíčem je uzel jakožto cíl cesty a hodnotou je seznam cest, které do daného uzlu vedou. Koncepčně by se dalo říct, že jde o jednoduchou směrovací tabulku.

Dělení datových balíčků
Zadání specifikuje možnost rozdělit datový balíček na menší porce v případě, že spoj mezi dvěma uzly nemůže přenést dané množství dat. Tento případ je řešen tak, že datový balíček je rozdělen na dvě poloviny, jedna bude putovat dále, druhá se uloží do SmartStacku na daném routeru.

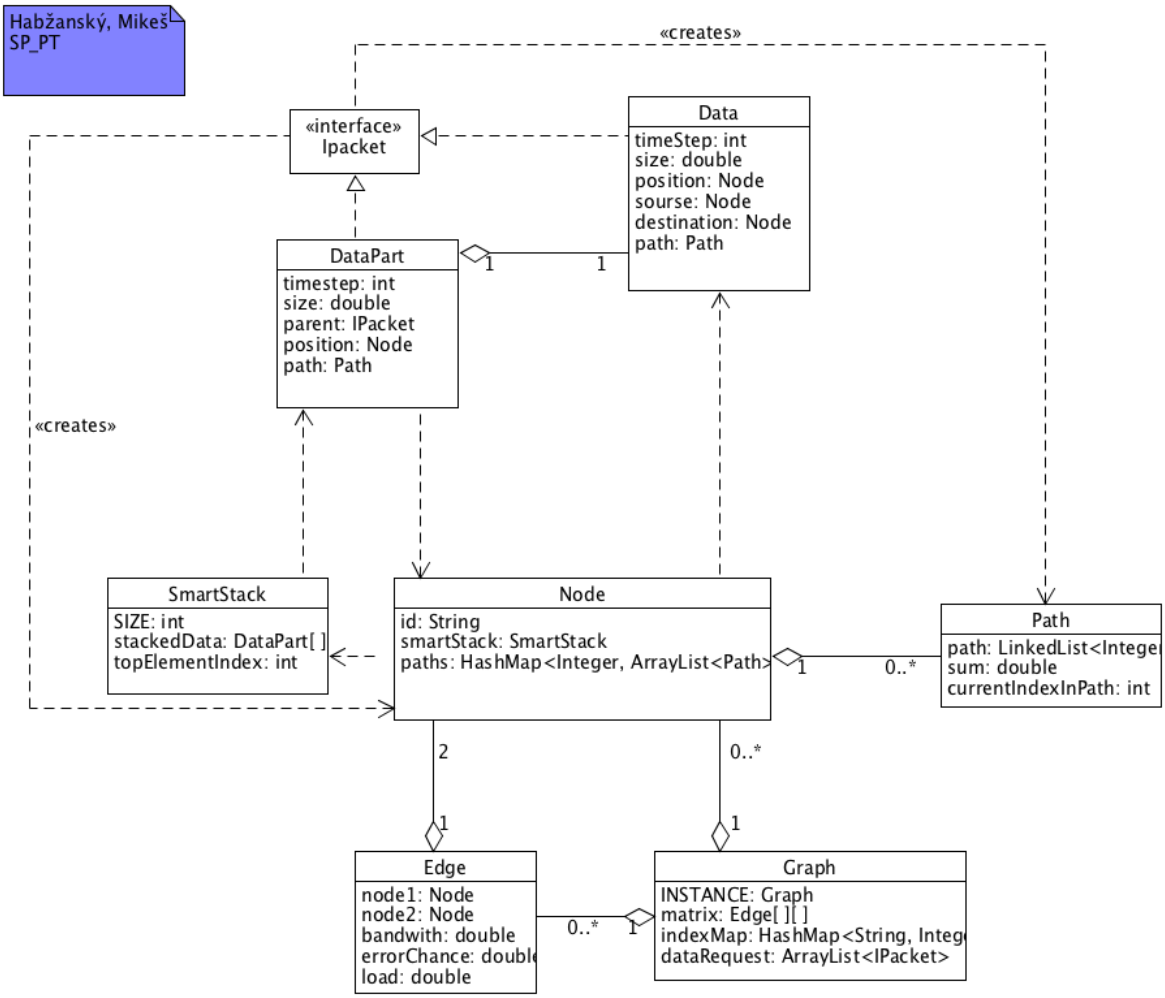
Polovina, která se uloží do SmartStacku je uložena do seznamu pro požadavky je mu nastaven čas o jedna vyšší, než se právě vyřizuje (bude tedy řešen v dalším kroku vyřizování požadavků). Tento balíček si pamatuje svého „rodiče“, od kterého byl původně oddělen a samozřejmě také původní cíl cesty.

Dynamická změna cesty
Jak bylo zmíněno v předchozí části, při větším vytížení (nebo malé propustnosti) spoje se datový balíček dělí na dvě části. Nicméně i po rozdělení se může stát, že nebude možné daný balíček poslat. V takovém případě je potřeba změnit cestu, vybíráme tedy z routeru cestu druhou v pořadí (první v pořadí jsme vyhradili pro požadavek, který má start v daném routeru).

Spojování dat
Jak bylo řečeno, datové balíčky se dělí na menší porce, pokud uzel nemůže přenést celý packet. Tyto data je následně potřeba spojit dohromady až dorazí do cíle. Původní myšlenka byla taková, že datový segment se připojí ke svému rodiči až dorazí do cíle. Nicméně uchýlili jsme se k myšlence, že se k němu připojí jakmile se opět sejdou na stejném routeru. To potenciálně může způsobit menší počet datových balíčků obýhajících v síti. Nicméně také však díky tomu může dojít k častějšímu dělení datových balíčků.

UML diagram

Na následujícím UML diagramu jsou detailněji popsány vztahy mezi jednotlivými komponentami:



Výsledky PMD

PMD na začátku hlásilo mnoho chyb, většinou chybějící gettery/settery, používání příkazu if s tělem o jedné řádce bez složených závorek a typování generických typů přímo na vlastní třídu místo rozhraní. Všechny tyto problémy byly opraveny a jejich výpis byl zkrácen na:

/Users/PavelHabzansky/IdeaProjects/PT_Semestralka/src/Graph.java:17: class 'Graph' je prilis slozita (Cyclomatic Complexity of 3 (Highest = 16)).

/Users/PavelHabzansky/IdeaProjects/PT_Semestralka/src/Graph.java:244: Ruznych cest pruchodu metodou 'sendDataPackets()' je 391, doporuceny limit je 200 (NPath complexity).

/Users/PavelHabzansky/IdeaProjects/PT_Semestralka/src/Graph.java:244: method 'sendDataPackets' je prilis slozita (Cyclomatic Complexity of 16).

Uživatelská dokumentace

Před spuštěním programu se ujistěte, že máte nainstalovanou Java JRE v aktuální verzi (http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html). Do libovolné složky na disku uložte soubor **PT_Semetralka.jar** a k němu dva textové soubory, jeden s hranami, druhý s požadavky.

Formát souboru s hranami:

```
id0 id1 100 0.8
id1 id2 300 0.9
id2 id3 1000 0.5
id3 id4 400 0.8
id4 id5 180 0.75
id5 id0 200 0.85
id1 id5 300 0.9
id1 id4 150 0.64
id2 id4 800 0.3
```

Formát souboru s požadavky:

```
1 id0 id3 400
3 id4 id2 300
2 id5 id1 550
5 id3 id1 400
7 id5 id2 800
8 id5 id1 260
4 id5 id4 700
```

Program se spouští z příkazové řádky příkazem **java -jar PT_Semestralka.jar <soubor_hrany> <soubor_pozadavky>** a následným stiskem klávesy Enter.

Na terminál se promítne matice reprezentující graf a případné zprávy o příchodu balíčku dat do cíle nebo jeho selhání a návratu do seznamu požadavků.

Při doražení balíčku dat do cíle se zapíše stručný údaj o jeho cestě do souboru **log.txt.**

Závěr

Podařilo se i přes menší potíže s implementací vytvořit program splňující kritéria definovaná zadáním. S prací jsme relativně spokojeni a byla i přes občasné klopýtnutí zvládnutelná a napsaná v rozšířitelném formátu. Kvalita kódu se projevila ve chvílích, kdy byla potřeba změnit implementace některých principů, které tak ve finále zabraly pouze pár řádek a žádné drastické změny nebyly nutné.