České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Semestrální práce z předmětu X36ASS
Architektury softwarových systémů

Řešení lineárních elektrických sítí dokumentace

Jan Skalický

prosinec 2008

Obsah

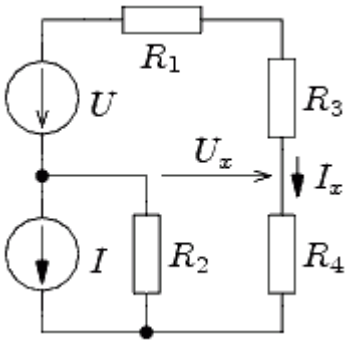
1. Zadání

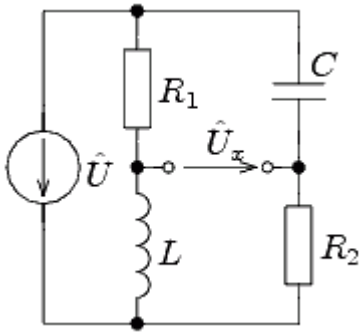
Navrhněte ve dvou architekturách program sloužící k řešení elektrických sítí s lineárními aktivními a pasivními dipóly, pracujících v ustáleném stacionárním nebo harmonickém stavu, zejména ke hledání neznámých obvodových veličin (napětí, proud). Jádro analytické simulace bude realizovat uživatelský vstup a výstup prostřednictvím konzole. Úkolem uživatele je popsat analyzovaný obvod jako soustavu součástek, připojených k označeným uzlům.

První architekturou bude implementace programu v jazyce Java. Druhou architekturou bude jeho implementace v C++.

1.1 Motivace

Motivací projektu je problém řešení elektrických sítí s lineárními aktivními a pasivními dipóly, pracujících v ustáleném stacionárním nebo harmonickém stavu. Příklady takových sítí jsou na následujících obrázcích:





1.2 Záměr

Záměrem projektu je jádro analytického řešícího softwaru. Vstupem uživatele bude popis obvodu ve tvaru soustavy součástek a jejich propojení. Výstupem programu bude nalezení neznámých obvodových veličin (napětí, proud, výkon).

1.3 Požadavky

1.3.1 Funkční požadavky

- Obvod může obsahovat řízené zdroje
- Zpracování obecných grafů (více) obvodů
- Správné rozpoznávání neurčitých stavů
- Stanovení jalových příkonů a účinníků
- Transformace pro počítání impedancí, zkratů, přenosů (a aproximací charakteristik)

1.3.2 Nefunkční požadavky

- Platformově nezávislý kód (portabilita)
- Rychlost výpočtu
- Možnost opravy vstupu
- Možnost rozšíření o GUI
- Výstup v úsporném formátu (pro zobrazování na malých displayích)

2. Popis a použití

Program Elnet slouží k řešení elektrických sítí s lineárními aktivními a pasivními dipóly, pracujících v ustáleném stacionárním nebo harmonickém stavu, zejména ke hledání neznámých obvodových veličin (napětí, proud). Díky jeho komplexnosti však umožňuje, s využitím jednoduchých transformací, stanovovat rovněž impedance linearních pasivních dipólů a počítat parametry náhradního zapojení lineárních aktivních dipólů. V extrémním případě lze s jeho pomocí stanovit rovněž hrubou frekvenční charakteristiku lineárního pasivního dipólu.

3. Ovládání a tipy

Protože se jedná vlastně o jádro simulačního programu pro lineární ustálené elektrické sítě, je uživatelský vstup a výstup realizován prostřednictvím konzole. Úkolem uživatele je popsat analyzovaný obvod jako soustavu součástek, připojených k označeným uzlům (v tomto případě se za uzel považuje spojení 2 a více součástek). Program zpracovává následující typy součástek:

- propojka (typ 0)
- zdroj napětí (typ 1)
- zdroj proudu (typ 2)
- rezistor (typ 3) ideální odporník[1]
- induktor (typ 4) ideální cívka[1]
- kapacitor (typ 5) ideální kondenzátor[1]
- impedance (typ 6) obecná hodn. impedance (volitelný úhel složek)
- řízené napětí (typ 7) proudem nebo napětím řízený zdroj napětí
- řízený proud (typ 8) napětím nebo proudem řízený zdroj proudu

Po úvodním dotazu na počet součástek obvodu (Number of parts) a úhlové frekvenci všech zdrojů (Source frequency [Hz]) nastává vlastní zadávání parametrů jednotlivých prvků. Při analýze harmonického obvodu jsou Všechny vstupní komplexní veličiny (hodnoty nezávislých, řízených zdrojů a obecné impedance) zadávány ve tvaru fázorů (v pořadí absolutní hodnota, úhel). Pro všechny prvky je nutné uvést mezi kterými uzly obvodu se nacházejí. Pro jednotlivé typy pak:

1 reálné součástky lze nahradit vhodným zapojením ideálních prvků

typ	Veličina	Jednotka
1	napětí zdroje[2]	[V]
2	proud zdroje[2]	[A]
3	rezistance	[Ω]
4	indukčnost	[H]
5	kapacita	[F]
6	impedance[2]	[Ω]
7	napětí zdroje[2]	způsob řízení[3] a řídící prvek
8	proud zdroje[2]	způsob řízení[3] a řídící prvek

Pokud je na začátku zvolen stacionární stav (úhlová frekvence je 0), pak je dotaz na úhel zdrojů vynechán. Pokud je zjištěna přítomnost jalové součástky ve stejnosměrného obvodu, je zobrazeno varování a dojde k její náhradě „inertním“ zdrojem, aby nebyla porušena konzistence sítě. K podobné situaci dojde při pokusu o vložení kondenzátoru s nulovou kapacitou do obvodu střídavého. Rovněž není přípustné, aby jeden prvek měl oba póly připojené do stejného uzlu. V takovém případě, dojde k záměně jednoho uzlu za nejbližší neobsazený.

Uzly obvodu není nutné definovat vzestupně a rovněž není nutné obsadit všechny mezi prvním a posledním použitým. Jelikož jsou uzly i součástky číslovány od nuly, vyplatí se začít definicí zdroje se záporným pólem připojeným do uzlu 0 a pokračovat definicí zbylých součástek.

Po zadání potřebných parametrů všech prvků sítě dojde k její analýze a vyřešené veličiny (napětí, proudy a výkony všech součástek) jsou vypsány do konzole. V případě analýzy obvodu ve stacionárním stavu jsou veličiny zobrazeny ve formě reálných čísel, v případě harmonické analýzy (pokud úhlová frekvence nebyla nulová) v komplexním tvaru složkovém i exponencielním (vhodné např. pro rychlé odečítání činného i zdánlivého výkonu). Všechny výsledky jsou zobrazeny v základních jednotkách pro danou veličinu.

3.1 Tipy

Pro stanovení impedance/admitance linearního pasivního dipólu proveďte lehkou transformaci, spočívající v přidání proudového/napěťového zdroje s proudem -1A/-1V na svorky analyzovaného dipólu. Po vyřešení obvodu stačí odečíst velikost napětí/proudu na svorkách zdroje/zdrojem, které je rovno hledané impedanci/admitanci.

Pro stanovení zkratového proudu linearního aktivního dipólu vřaďte na jeho výstupní svorky propojku. Po vyřešení obvodu stačí odečíst velikost proudu propojkou.

Pro určení napětí mezi dvěma uzly, které nejsou propojeny žádnou součástkou (a tudíž se jejich napětí standardně nezobrazuje v množině výstupních napětí a muselo by se počítat slučováním napětí ve vhodně zvolené smyčce), vřaďte mezi takovéto uzly nulový kondenzátor (nebo nulový proudový zdroj), což je opak propojky. Nedojde tak k ovlivnění obvodu a požadované napětí se po vyřešení obvodu objeví v podobě napětí tohoto nově vloženého inertního prvku.

2 veličiny zadávané fázorem
3 napětím, proudem

Program je díky své komplexnosti schopen řešit více nezávislých obvodů v 1 kroku, stačí definovat oba současně do jednoho zadání s nepropojenými částmi (resp. propojenými maximálně jedním prvek (a to nikoliv proudovým zdrojem), který se však na výsledku činnosti obvodu nijak neprojeví).

4. Znaménková konvence

Kladný smysl obvodových veličin je volen stejně pro spotřebiče i pro zdroje, sice od uzlu s vyšším indexem k uzlu s nižším indexem (tomu je nutné přizpůsobit hodnoty veličin zdrojů – šipka zadávané veličiny musí směřovat do nižšího uzlu).

Příkony prvků spotřebovávajících el. energii pak vycházejí kladné a u prvků generujících záporné. Jalový příkon spotřebiče indukčního charakteru je v souladu s konvencí kladný, u kapacitního charakteru záporný.

5. Neurčité stavy a přesnost

Neurčitými stavy jsou myšlena zapojení, která nemají na teoretické úrovni smysluplné řešení. Jedná se o zkratované nebo paralelně zapojené zdroje napětí, nezapojené nebo sériově zapojené zdroje proudu pod. Tyto situace, při kterých nelze stanovit obvodové veličiny tak, aby byl výsledek konzistentní vzhledem k zapojení sítě, signalizuje program hláškou „no definite solution for this circuit“ a pozastavením činnosti.

Přesnost výpočtů a číselného výstupu programu je omezena přesností datového typu double pro plovoucí desetinnou čárku. Výsledky nejsou, ani při výpisu do konzole, jakkoliv zaokrouhlovány (ačkoliv někdy může být patrné, že místo 4.99999 patří 5).

Výstupní rozsah úhlů komplexních čísel je -π až +π, přičemž -π je použito k indikaci čísla s oběma nulovými složkami.

6. Vývojová dokumentace

Schéma činnosti programu je následující:

1) vstup dat
2) vytvoření datových struktur (graf obvodu)
3) analýza grafu
    3.1) hledání optimálního úplného stromu (kostry)
    3.2) hledání nezávislých smyček -> plnění matice
    3.3) hledání nezávislých řezů -> plnění matice
4) doplnění vznikající matice definicemi součástek
5) řešení heterogenní soustavy s komplexní maticí
    5.1) Gaussova eliminace
    5.2) Jordanova eliminace
    5.3) převod na jednotkovou matici
6) vyjmutí výsledků obvodových veličin, spočtení příkonů
7) výstup dat

6.1 Popis užitých tříd a jejich metod

- algebra — maximum, minimum a signum ze 2 čísel
- CCircuit — popis obvodu + analyzační algoritmy
- CComplex — komplexní číslo, operace s ním
- CDipole — vlastnosti součástky obvodu
- CEdge — hrana grafu
- CMatrix — komplexní matice, řešení
- CRoute — směry incidentních hran grafu
- CVertex — vrchol grafu
- main — I/O z konzole, inicializace obvodu

6.2 Analýza grafu

Nejzajímavější část programu je analýza grafu, proto ji popíši podrobněji.

6.2.1 Data grafu

graf je uložen v objektové struktuře:

- pole vrcholů (CVertex nodes) zastupující uzly obvodu
  - pole směrů (CRoute routes) k sousedním vrcholům přes incidentní hrany
  - počet směrů (int routecnt)
  - počet obsažených hran kostry (int bones)
  - značka pro trasovací účely (int tag)
- pole hran (CEdge branches) zastupující větve obvodu
  - indexy vrcholů (int vex1, vex2), které spojuje
  - je součástí kostry? (boolean bone)
  - značka pro trasovací účely (int tag)

6.2.2 Algoritmy pro analýzu grafu

jsou instančními metodami třídy CCircuit (pracují s objekty nodes, branches):

- int skeleton()

  nalezení a označení kostry grafu tak, aby měla pokud možno co nejvíc listů[4] (vrcholů s právě jednou hranou z kostry); je zde počítáno i s možností nesouvislého grafu popisujícího více nezávislých obvodů v jednom zadání; metoda vrací počet segmentů

- int bones()

  spočtení obsažených hran kostry pro každý vrchol grafu; metoda vrací počet nalezených listů

4 optimalizace pro následné hledání nezávislých řezů – v případě listu je určení řezu triviální

- void getLoop()

  nalezení nezávislé smyčky obsahující nezávislou hranu v parametru a naplnění adekvátního řádku matice její rovnicí (+ uplatnění znaménkové konvence)

- void getCut()

  nalezení nezávislého řezu obsahujího hranu kostry v paramatru a naplnění adekvátního řádku matice jeho rovnicí (+ uplatnění znaménkové konvence)[5]

6.3 Implementované architektury

V souladu se zadáním byly použity 2 implementační architektury lišící se v použitých programovacích jazycích. Těmi byly rozšířené imperativní jazyky Java a C++.

6.3.1 C++

Výsledkem je offline konzolová aplikace. Použitým toolchainem (nástrojovým řetězcem) byly nástroje kompilátoru gcc a vývojového balíku GNU Tools na OS GNU/Linux. Byl použit objektový i strukturální přístup jazyka C. Správa paměti je vlastní dynamická alokace. Přesnost výpočtů plovoucí čárky je závislá pouze na cílové architektuře CPU a je tedy dána při překladu.

Celkově byl problém dekomponován do 4 tříd a 4 struktur. Data grafu schématu obvodu popisuje odstavec 6.2.1.

6.3.2 Java

Výsledkem je opět offline konzolová aplikace. Použitým toolchainem bylo IDE (integrované vývojové prostředí) Borland Jbuilder na OS Solaris. Byl použit striktně objektový přístup v rámci možností jazyka. Správa paměti je realizována pomocí zabudovaného Garbage Collectoru. Přesnost výpočtů plovoucí čárky je dána běhovým prostředím Javy (JRE – Java Runtime Environment) a je tedy platformově závislá.

Oproti implementaci v C++ byla vzhledem ke striktně objektovému přístupu použita pozměněná dekompozice – do 7 tříd (vrcholy, hrany a směry jsou jednoduché třídy; tento rozdíl přesně zachycuje class diagram níže). Odlišná je rovněž alokace polí objektů (alokace pointerů místo volání konstruktorů v C++). Z hlediska programátorského aspektu bylo nutné také omezení v některých konstruktech jazyka (for, goto, aj.)

5 tato metoda pracuje tak, že si rozdělí kostru na 2 části podél hrany kostry obsažené v řezu a každý vrchol grafu si označí (tag) podle toho, patří-li na jednu stranu od zadané hrany nebo na druhou. poté pro všechny nezávislé hrany zjišťuje, zda-li přímo nespojují 2 vrcholy z odlišných částí. není těžké ukázat, že právě takové hrany jsou součástí hledaného řezu.

6.3.3 Porovnání architektur

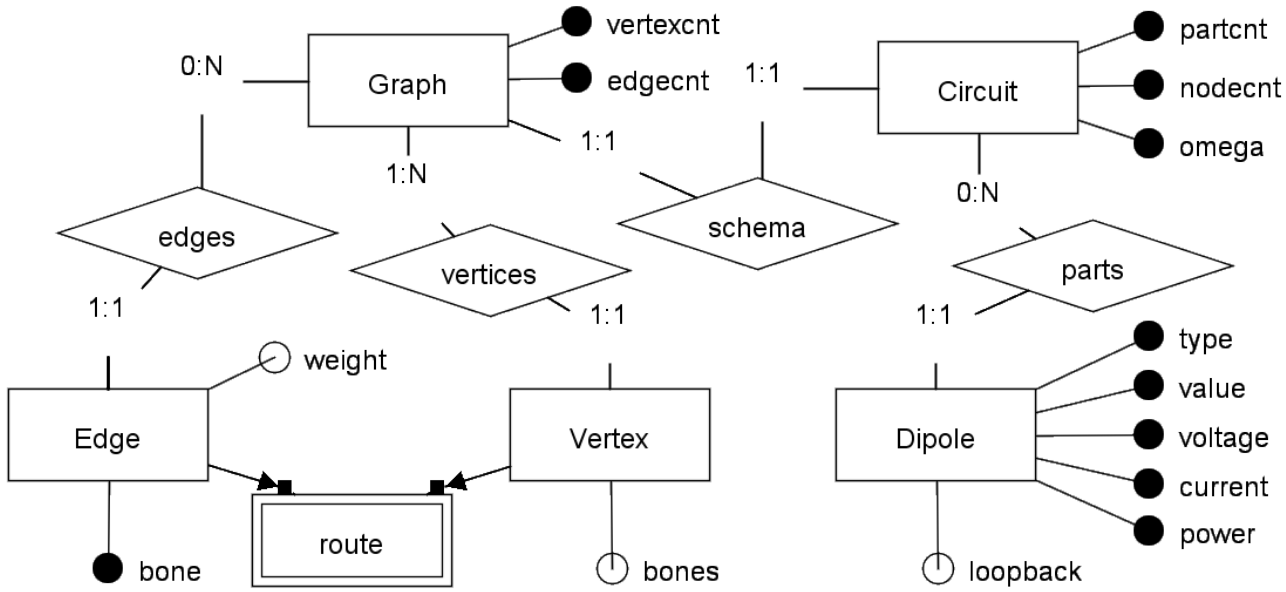
Postatné rozdíly obou implementačních architektur porovnává následující tabulka:

C++	Java
použití struktur (lépe optimalizovatelné)	všechno je třída (více systémové, režie)
rychlejší běh, ale je nutno rekompilovat pro platformu	rychlejší běh, ale je nutno rekompilovat pro platformu
vlastní alokace paměti (předvídatelné chování)	vlastní alokace paměti (předvídatelné chování)

6.4 E-R diagramy

E-R diagramy popisují organizaci dat v databázi, která je v našem případě realizována strukturami programovacích jazyků a uložena v paměti v době běhu.

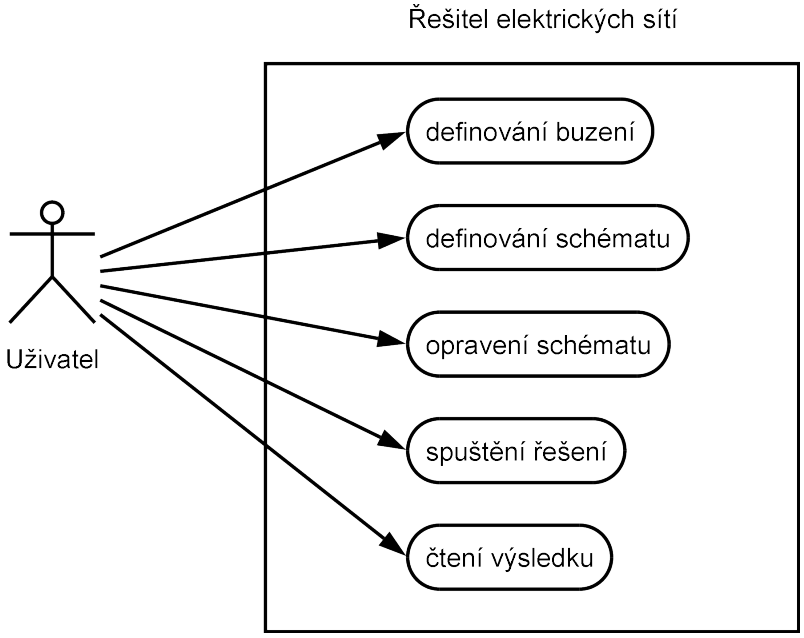
6.4.1 E-R model grafu schématu



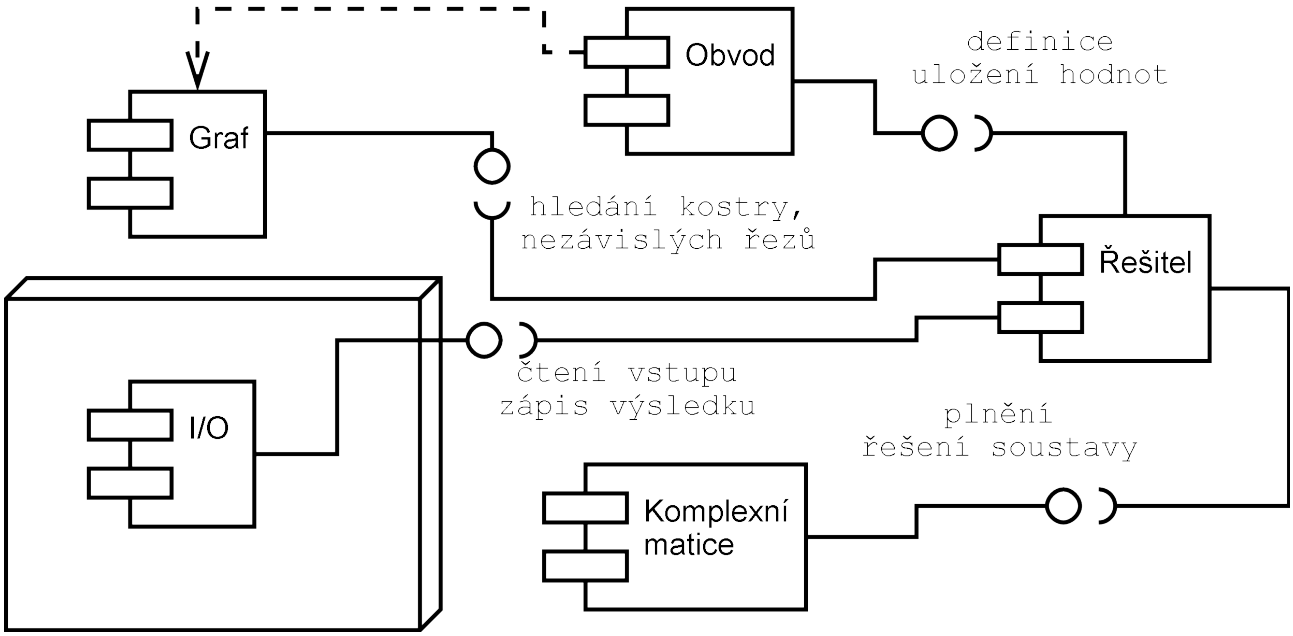
6.5 UML diagramy

UML je unifikovaný jazyk pro modelování softwarových systémů. Standardizované diagramy nevyžadují další komentář. Vytvořeny byly ty diagramy, které mají význam pro „stand-alone“ program v souladu s požadavky zadání.

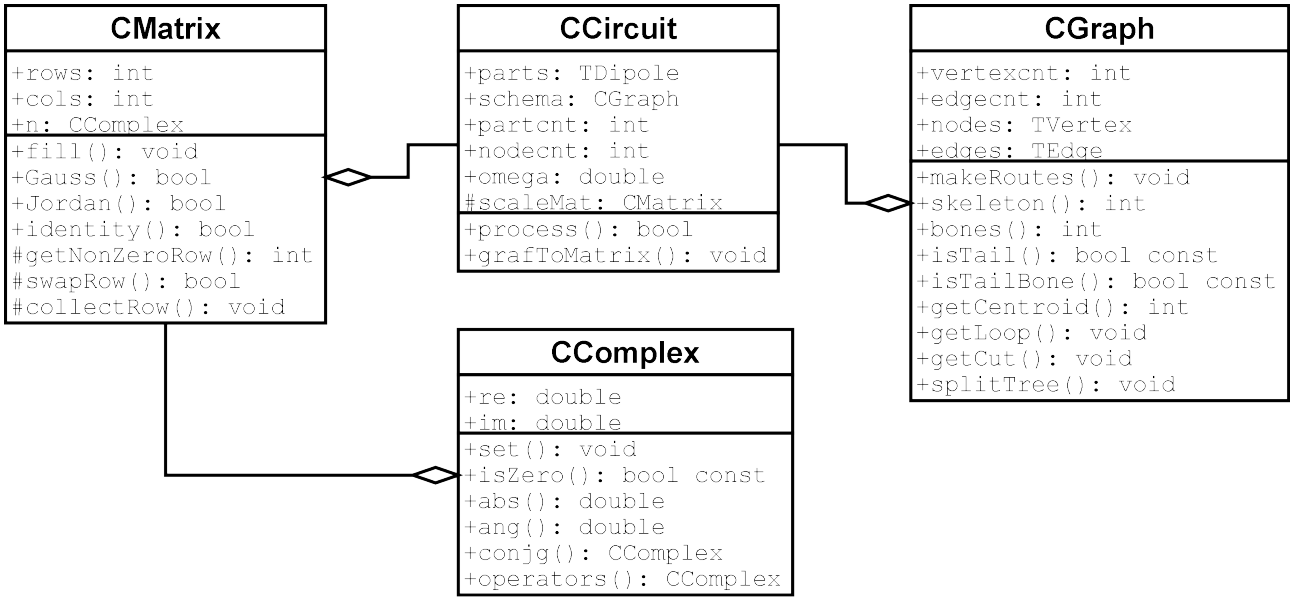
6.5.1 Use case diagram



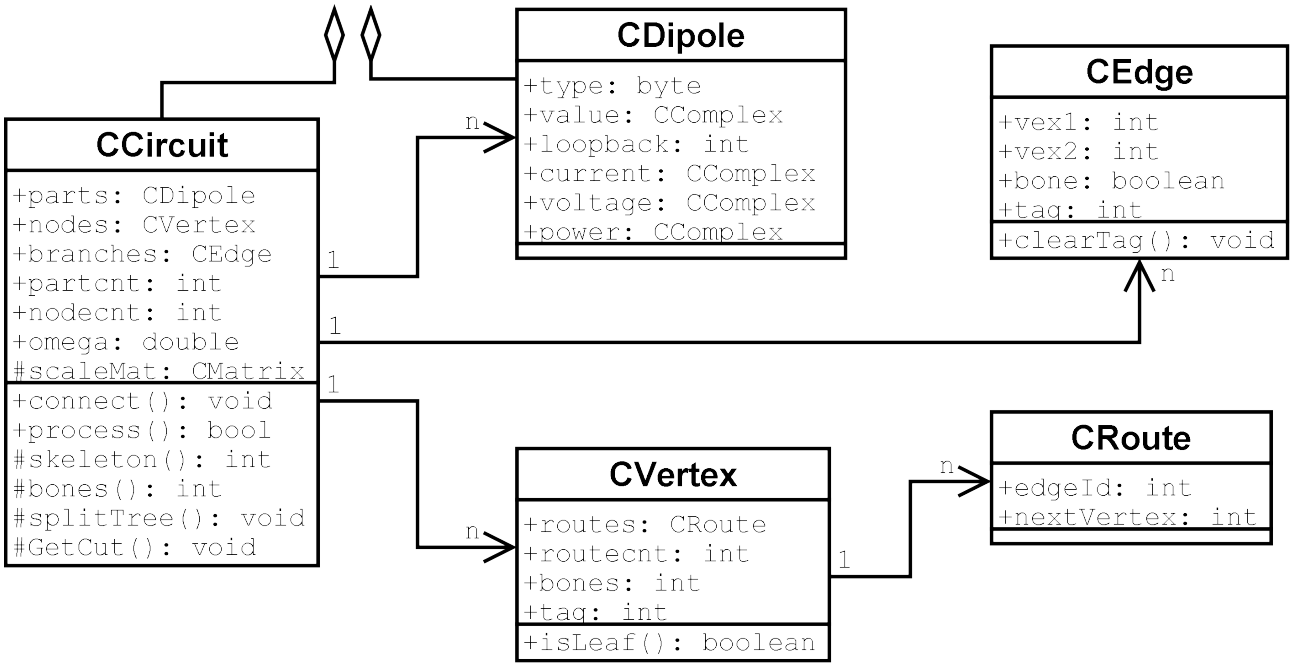
6.5.2 Diagram komponent



6.5.3 Class diagram

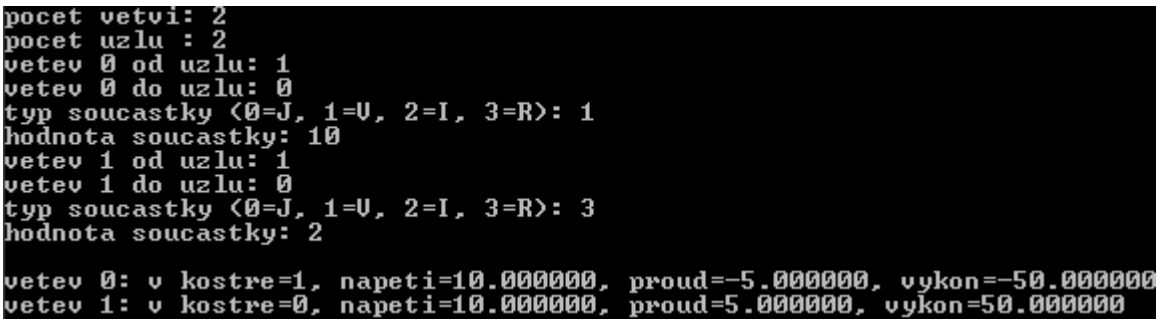


6.5.4 Class diagram – CGraph v Javě



7. Uživatelské rozhraní

Ukázková konzolová aplikace obsahuje testovací main() s následujcím příkladovým promptem:



8. Zhodnocení požadavků

Všechny funkční i nefunkční požadavky byly splněny. Obvod může obsahovat řízené zdroje i více nezávislých komponent grafu schématu. Správně rozpoznává neurčité stavy (indikací na výstupu) a umožňuje stanovovat některé další velčiny z obvodových.

Jedná se o kód, který není závislý na specifikách platformy (OS, arch.), ale pouze programovacího jazyka, takže je dobře portabilní. Vlastní výpočet je poměrně rychlý. Program lze rozšířit o GUI dle libosti a upravený main() provádí výstup v úsporném formátu, což bylo odzkoušeno na grafickém kalkulátoru Casio Algebra FX 2.0+.

9. Závěr

Zdá se, že program pracuje spolehlivě; byl několikrát testován pro různé typy obvodů a všechny výsledky vždy souhlasily se správným řešením. Důraz při výběru zpracovávaného tématu byl kladen i na jeho využitelnost v praxi. Míním, že nejvíce se mi povedlo vymyslet princip činnosti metody pro hledání nezávislých řezů. Někdo by naopak mohl programu vytknout provedení uživatelského rozhraní, leč opakuji, jedná se vlastně o jádro a návrh interfacu nebyl účelem práce.