FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Semestrální práce

Nástroj pro řešení úloh lineárního programování

Filip Nehéz

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Semestrální práce

Nástroj pro řešení úloh lineárního programování

Filip Nehéz

PLZEŇ 2024

Citace v seznamu literatury:
NEHÉZ, Filip. *Nástroj pro řešení úloh lineárního programování*. Plzeň, 2024. Semestrální práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

Obsah

Úvod 1

Tato semestrální práce se zaměřuje na návrh a implementaci programu, který řeší úlohy lineárního programování. Lineární programování je klíčovou oblastí matematické optimalizace, která nachází široké uplatnění v průmyslu, logistice, financích a dalších oblastech.

Optimalizace je nezbytná pro efektivní využití zdrojů a zlepšení rozhodovacích procesů. Lineární programování umožňuje nalézt nejlepší hodnotu účelové funkce při splnění řady omezujících podmínek. Program Nemesis je navržen tak, aby poskytoval efektivní a spolehlivé řešení těchto úloh.

Cílem této práce je vytvořit nástroj, který:

- Načte vstupní data ve formátu LP.
- Provede syntaktickou analýzu vstupních dat.
- Provede optimalizaci.
- Ošetří specifické scénáře, jako je neomezenost nebo neřešitelnost úlohy.
- Generuje výstup, který poskytuje uživateli přehledné a srozumitelné výsledky.

Zadání 2

Naprogramujte v jazyce ANSI C přenositelnou konzolovou aplikaci, která bude řešit úlohy lineárního programování zadané ve zjednodušeném formátu LP. Program bude spouštěn příkazem lp.exe s kombinací následujících argumentů – výrazy v lomených závorkách (<>), resp. hranatých závorkách ([]) označují povinné, resp. nepovinné argumenty:

<input-file> Soubor s popisem úlohy ve formátu LP. V případě, že uživatel zadá neexistující soubor, program vypíše chybové hlášení "Input file not found!\n"a vrátí hodnotu 1.

-o <path> Výstupní soubor s řešením úlohy. Pokud umístění neexistuje, bude vypsáno hlášení "Invalid output destination!\n"a program skončí s návratovou hodnotou 2. V případě, že uživatel tento přepínač nezadá, bude výsledek optimalizace vypsán na obrazovku. Do tohoto souboru neuvádějte chybová hlášení.

–output <path> Stejné jako v případě přepínače -o. Použití obou přepínačů -o a –output není chybou, program pak bude akceptovat poslední zadanou hodnotu.

V případě nalezení konečného optimálního řešení úlohy program vrátí hodnotu `EXIT_SUCCESS`. Chybové stavy týkající se zpracování vstupních souborů nebo samotného algoritmu optimalizace jsou popsány v dalších sekcích.

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť obsahuje všechny zdrojové soubory potřebné k přeložení programu, Makefile pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný Makefile a pro Windows Makefile.win) a dokumentaci ve formátu PDF vytvořenou v typografickém systému TEX (LATEX). Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

Analýza úlohy 3

3.1 Obecný popis úlohy

Úloha zadává návrh programu pro řešení problémů lineárního programování (LP), což je klíčová oblast matematické optimalizace. Lineární programování má široké využití v průmyslu, logistice, financích a dalších oblastech, kde je potřeba optimalizovat procesy za přísných omezení. Cílem je vytvořit nástroj, který umožní automatické řešení těchto problémů na základě vstupních dat ve specifickém formátu.

3.1.1 Lineární programování

Lineární programování je metoda optimalizace, jejímž cílem je nalézt nejlepší hodnotu účelové funkce, která je lineární, při splnění řady omezujících podmínek, jež jsou také vyjádřeny jako lineární rovnice nebo nerovnice. Typické prvky problému LP zahrnují:

- **Účelovou funkci:** Lineární výraz, který je potřeba maximalizovat nebo minimalizovat, například:

  $$z = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n,$$

  kde c_i jsou koeficienty a x_i rozhodovací proměnné.

- **Omezující podmínky:** Lineární rovnice nebo nerovnice, které vymezují přípustnou oblast řešení, např.:

  $$a_{11} x_1 + a_{12} x_2 \leq b_1, \quad x_1 \geq 0, \quad x_2 \geq 0.$$

- **Nezápornost proměnných:** Rozhodovací proměnné $x_1, x_2, \ldots, x_n$ bývají obvykle omezeny na kladné hodnoty.

3.1.2 Praktické aspekty úlohy

Úloha není pouze akademická, ale má významné praktické uplatnění. Příkladem může být:

- Optimalizace výroby v podniku, kde jsou omezené zdroje, jako pracovní čas, materiály nebo energie.
- Plánování logistiky, zahrnující minimalizaci přepravních nákladů při splnění požadavků na dodávky.
- Finanční optimalizace, např. maximalizace výnosů investičního portfolia při omezeném riziku.

3.2 Analýza specifického zadání

V této úloze je úkolem vytvořit program, který:

1. **Načte vstupní data ve formátu LP:** Tato data zahrnují účelovou funkci, omezující podmínky, omezení proměnných a další klíčové prvky.
2. **Vyřeší optimalizační problém:** Použije se algoritmus, jako je simplexová metoda, pro nalezení optimálního řešení.
3. **Zpracuje specifické scénáře:**
   - Identifikace neomezeného řešení (např. účelová funkce může růst do nekonečna).
   - Zjištění neexistence přípustného řešení (např. podmínky nemají průnik).
   - Výpočet optimální hodnoty účelové funkce a odpovídajících hodnot proměnných.
4. **Generuje výstup:** Výstup může být zobrazen na obrazovce nebo uložen do souboru.

3.2.1 Syntaktická analýza vstupního souboru

Syntaktická analýza vstupního souboru je klíčovou částí programu, která zajišťuje správné načtení a interpretaci vstupních dat ve formátu LP. Tento proces zahrnuje několik kroků, které jsou nezbytné pro úspěšné zpracování vstupního souboru a přípravu dat pro optimalizační algoritmus. V následujících odstavcích provedu úvahu o způsobech provedení syntaktické analýzy.

Syntaktická analýza tokenizací. Tokenizace je proces, při kterém je vstupní text rozdělen na jednotlivé tokeny, což jsou základní stavební kameny, jako jsou klíčová slova, operátory, proměnné a čísla. Tento krok je důležitý pro identifikaci a klasifikaci jednotlivých částí vstupního souboru. Tokenizace však může být složitá, pokud nejsou tokeny jednoznačně odděleny mezerami nebo jinými oddělovači.

Shunting Yard algoritmus. Shunting Yard algoritmus, vyvinutý Edsgerem Dijkstrou, je efektivní metoda pro převod infixových výrazů na postfixové (reverzní polskou notaci, RPN). Tento algoritmus je vhodný pro syntaktickou analýzu matematických výrazů, protože dokáže správně zpracovat operátory s různou prioritou a závorky.

Porovnání tokenizace a Shunting Yard algoritmu. Při porovnání tokenizace a Shunting Yard algoritmu je zřejmé, že Shunting Yard algoritmus nabízí několik výhod. Zatímco tokenizace je základním krokem pro rozdělení vstupního textu, Shunting Yard algoritmus poskytuje strukturovaný přístup k syntaktické analýze matematických výrazů. Tento algoritmus zajišťuje správné pořadí operací a zpracování závorek, což je klíčové pro správnou interpretaci a vyhodnocení výrazů.

Výběr Shunting Yard algoritmu. Na základě výše uvedeného porovnání jsem se rozhodl použít Shunting Yard algoritmus pro syntaktická analýza vstupního souboru. Tento algoritmus nám umožňuje efektivně a správně zpracovat matematické výrazy, které jsou součástí účelové funkce a omezujících podmínek. Ovšem základy tokenizace se využijí i při tomto postupu syntaktické analýzy.

3.2.2 Optimalizační proces

Optimalizační proces je klíčovou součástí programu, která se zaměřuje na nalezení nejlepšího možného řešení dané úlohy lineárního programování. Tento proces zahrnuje několik metod a technik, které jsou použity k dosažení optimálního výsledku. V následujících odstavcích provedu úvahu o způsobech provedení optimalizace.

Simplexová metoda. Simplexová metoda je jednou z nejznámějších a nejpoužívanějších metod pro řešení úloh lineárního programování. Tato metoda iterativně prochází vrcholy mnohostěnu, který představuje množinu přípustných řešení, a hledá optimální řešení na hranicích tohoto mnohostěnu. Simplexová metoda je efektivní a často se používá v praxi, ale může selhat v případě degenerovaných úloh.

Interiérové metody. Interiérové metody, také známé jako metody vnitřního bodu, jsou alternativou k simplexové metodě. Tyto metody procházejí vnitřkem množiny přípustných řešení a hledají optimální řešení pomocí iterativních kroků. Interiérové

metody jsou často rychlejší než simplexová metoda pro velké úlohy, ale mohou být složitější na implementaci.

Metoda velkého M (The Big M Method). Metoda velkého M je varianta simplexové metody, která se používá pro řešení úloh s umělými proměnnými. Tato metoda přidává velké penalizační hodnoty (M) k umělým proměnným v účelové funkci, což zajišťuje, že umělé proměnné budou v optimálním řešení nulové. Metoda velkého M je efektivní a snadno implementovatelná, což ji činí vhodnou volbou pro mnoho praktických úloh.

Výběr metody velkého M. Na základě porovnání různých optimalizačních metod jsem se rozhodl použít metodu velkého M pro řešení zadané optimalizační úlohy. Tato metoda nabízí jednoduchou implementaci a efektivní řešení úloh s umělými proměnnými. Použití metody velkého M zajišťuje, že náš program bude schopen rychle a správně nalézt optimální řešení pro širokou škálu úloh lineárního programování a určit řešitelnost úlohy.

Popis implementace 4

Tato kapitola poskytuje podrobný popis implementace programu Nemesis, který řeší úlohy lineárního programování pomocí simplexové metody s velkým M. Program je napsán v jazyce ANSI C a skládá se z několika modulů, které spolupracují na načtení, zpracování a vyřešení optimalizační úlohy.

4.1 Struktura programu

Program je rozdělen do několika souborů, z nichž každý má specifickou funkci:

- `main.c`: Hlavní vstupní bod programu.
- `parser.c` a `parser.h`: Modul pro parsování vstupního souboru a přípravu dat pro optimalizační algoritmus.
- `simplex.c` a `simplex.h`: Implementace simplexové metody s velkým M.
- `matrix.c` a `matrix.h`: Modul pro práci s maticemi.
- `queue.c` a `queue.h`: Implementace frontové struktury.
- `stack.c` a `stack.h`: Implementace zásobníkové struktury.

4.2 Hlavní modul

Soubor `main.c` obsahuje hlavní funkci programu, která provádí následující kroky:

1. Parsování argumentů příkazové řádky pomocí funkce `args_parser`.
2. Načtení a parsování vstupního souboru pomocí funkce `input_parser`.
3. Použití simplexového algoritmu k vyřešení optimalizační úlohy pomocí funkce `simplex`.
4. Výpis výsledků na standardní výstup nebo do souboru pomocí funkce `printResults`.

4.3 Modul pro syntaktickou analýzu

Modul `parser.c` je zodpovědný za syntaktickou analýzu vstupního souboru ve formátu LP a přípravu dat pro optimalizační algoritmus. Tento modul zahrnuje několik klíčových kroků a struktur, které jsou nezbytné pro správné načtení a interpretaci vstupních dat.

4.3.1 Klíčové struktury

- `struct problem_data`: Tato struktura obsahuje veškerá data potřebná pro řešení optimalizační úlohy, včetně účelové funkce, podmínek, omezení a výsledků.
- `struct rpn_item`: Tato struktura reprezentuje prvek v reverzní polské notaci (RPN). Prvek může být číslo, proměnná nebo operátor.
- `struct evaluation_expression`: Tato struktura slouží k vyhodnocení matematických výrazů. Obsahuje konstantu a koeficienty proměnných.

4.3.2 Shunting Yard algoritmus

Shunting Yard algoritmus, vyvinutý Edsgerem Dijkstrou, je klíčovou součástí modulu pro syntaktickou analýzu. Tento algoritmus je použit pro převod infixových výrazů na postfixové (reverzní polskou notaci, RPN), což usnadňuje jejich vyhodnocení. Algoritmus zajišťuje správné pořadí operací a zpracování závorek.

4.3.3 Klíčové kroky modulu

- **Syntaktická analýza argumentů**: Funkce `args_parser` zpracovává argumenty příkazové řádky a identifikuje vstupní a výstupní soubory.
- **Čtení vstupního souboru**: Funkce `input_parser` čte vstupní soubor a extrahuje relevantní sekce, jako jsou účelová funkce, podmínky a omezení.
- **Tokenizace**: Funkce `input_parser` rozdělí vstupní sekce na jednotlivé komponenty. Např. levá a pravá strana rovnice,
- **Syntaktická analýza výrazu**: Tokenizované výrazy jsou převedeny do formátu, který může být vyhodnocen simplexovým algoritmem, pomocí Shunting Yard algoritmu.
- **Vyhodnocení výrazu v RPN**: Funkce `rpn_evaluate` vyhodnocuje výrazy v reverzní polské notaci a převádí je na strukturu `evaluation_expression`.

4.3.4 Detailní popis funkcí

- `args_parser`: Tato funkce zpracovává argumenty příkazové řádky a identifikuje vstupní a výstupní soubory. Kontroluje správnost argumentů a vrací odpovídající kód chyby v případě nesprávného vstupu.
- `input_parser`: Tato funkce čte vstupní soubor a extrahuje relevantní sekce, jako jsou účelová funkce, podmínky a omezení. Data jsou ukládána do struktury `problem_data`.
- `prepare_expression`: Tato funkce připravuje matematické výrazy k parsování do reverzní polské notace (RPN). Kontroluje správnost závorek a nahrazuje názvy proměnných jejich indexy.
- `parse_to_rpn`: Tato funkce převádí infixové výrazy na RPN pomocí Shunting Yard algoritmu. Výsledkem je fronta `queue` obsahující prvky `rpn_item`.
- `rpn_evaluate`: Tato funkce vyhodnocuje výrazy v RPN a převádí je na strukturu `evaluation_expression`. Používá zásobníkovou strukturu `stack` pro dočasné ukládání mezivýsledků.

4.3.5 Pseudokód Shunting Yard algoritmu

Níže je uveden pseudokód Shunting Yard algoritmu, který je použit v modulu pro syntaktickou analýzu:

Zdrojový kód 4.1: Pseudokód algoritmu Shunting Yard

```
1. Pro každý token ve vstupním výrazu:
    a. Pokud je token číslo nebo proměnná, přidej ho do výstupní fronty.
    b. Pokud je token operátor:
        i. Dokud je na vrcholu zásobníku operátor s vyšší nebo stejnou prioritou, přesuň ho do výstupní fronty.
        ii. Přidej aktuální operátor na zásobník.
    c. Pokud je token levá závorka, přidej ji na zásobník.
    d. Pokud je token pravá závorka:
        i. Dokud není na vrcholu zásobníku levá závorka, přesuň operátory ze zásobníku do výstupní fronty.
        ii. Odstraň levou závorku ze zásobníku.
2. Přesuň všechny zbývající operátory ze zásobníku do výstupní fronty.
```

4.4 Modul pro simplexovou metodu

Modul `simplex.c` obsahuje implementaci simplexové metody s velkým M. Tento modul je klíčový pro řešení úloh lineárního programování. Níže je uveden detailní popis hlavních algoritmů použitých v tomto modulu.

4.4.1 Inicializace simplexové tabulky

Inicializace simplexové tabulky je prvním krokem simplexové metody. Tento krok zahrnuje následující kroky:

1. **Příprava matice omezení**: Matice omezení je sestavena z koeficientů omezení a přidání umělých proměnných.
2. **Příprava účelové funkce**: Účelová funkce je upravena tak, aby zahrnovala velké penalizační hodnoty (M) pro umělé proměnné.
3. **Sestavení počáteční simplexové tabulky**: Počáteční simplexová tabulka je sestavena z matice omezení a účelové funkce.

4.4.2 Pivotování

Pivotování je klíčový krok simplexové metody, který umožňuje přechod mezi vrcholy přípustné oblasti. Tento krok zahrnuje následující kroky:

1. **Výběr pivotního sloupce**: Pivotní sloupec je vybrán při testu optimality, který je popsán v další sekci.
2. **Výběr pivotního řádku**: Pivotní řádek je vybrán na základě nejmenšího kladného podílu pravé strany a hodnoty pivotního sloupce.
3. **Provádění pivotních operací**: Pivotní operace zahrnují dělení pivotního řádku pivotním prvkem a následné úpravy ostatních řádků tak, aby se pivotní prvek stal jedničkou a ostatní prvky v pivotním sloupci se staly nulami. Tedy od každého nepivotního řádku je odečten příslušný násobek pivotního řádku.

4.4.3 Test optimality

Test optimality je prováděn pro určení, zda je řešení aktuální iterace simplexové tabulky optimální, nebo jej lze dále vylepšit. Pokud algoritmus testu optimality odhalí, že aktuální řešení není optimální, současně určuje sloupec, který je pro optimalizaci nejvhodnější.

Tetsovací hodnota se vypočítá jako rozdíl koeficientu příslušné proměnné sloupce a součtu násobků koeficientu bázové proměnné s hodnotou v daném sloupci a příslušném řádku.

Testovací hodnota se počítá pro každý sloupec simplexové tabulky. Pokud je některá z testovacích hodnot pozitivní, aktuální řešení není optimální.

Nejvhodnější sloupec pro optimalizaci je ten s největší pozitivní testovací hodnotou.

4.4.4 Ukončení algoritmu

Algoritmus simplexové metody je ukončen, když je dosaženo optimálního řešení a zároveň je báze validní, nebo byl-li přesažen maximální počet iterací simplexové tabulky. Validní bází problému se rozumí báze bez umělých proměnných.

Algoritmus je ukončen, pokud by se v průběhu mělo dělit nulou, s návratovou hodnotou úlohy neexistujícího řešení.

Pokud při pivotizaci nelze vybrat pivotní sloupec, je algoritmus ukončen s návratovou hodnotou úlohy s neomezeným řešením.

Pokud se po ukončení simplexového algoritmu nalézají v bázi umělé proměnné, je toto řešení považováno za neplatné a je navrácena hodnota úlohy s neexistujícím řešením.

4.4.5 Pseudokód algoritmu simplexové metody

Zdrojový kód 4.2: Pseudokód algoritmu simplexové tabulky

```
1. Inicializace simplexové tabulky
2. Opakuj, dokud (není dosaženo optimálního řešení nebo v bá
   zi řešení stále figurují umělé proměnné) a zároveň nebyl p
   řesažen maximální počet iterací taabulky:
        a. Vyber pivotní sloupec
        b. Vyber pivotní řádek
        c. Proveď pivotní operace
        d. Urči novou bázi.
3. Kontrola báze řešení a ukončení algoritmu.
```

Uživatelská příručka 5

Tato kapitola poskytuje návod k použití programu Nemesis, včetně instrukcí pro přeložení a sestavení programu na různých operačních systémech, formátu vstupního souboru a ukázek běhu programu. Program Nemesis je navržen pro řešení optimalizačních úloh pomocí metody velkého M.

5.1 Přeložení a sestavení programu

V této sekci jsou uvedeny kroky pro přeložení a sestavení programu Nemesis na operačních systémech Linux a Windows. Pro každý operační systém jsou poskytnuty specifické instrukce a požadavky.

5.1.1 Linux OS s použitím GCC

Pro přeložení a sestavení programu Nemesis na Linuxu s použitím GCC postupujte podle následujících kroků:

1. Ujistěte se, že máte nainstalovaný GCC. Můžete to ověřit příkazem "`gcc --version`".
2. Otevřete terminál a přejděte do adresáře s projektem Nemesis.
3. Spusťte příkaz "`make`" pro přeložení a sestavení programu. Tento příkaz použije "`Makefile`", který je součástí projektu, a vytvoří spustitelný soubor "`lp.exe`" v adresáři "`build`", který je zkopírován i do kořenového adresáře.

5.1.2 Windows OS s použitím MSVC

Pro přeložení a sestavení programu Nemesis na Windows s použitím Microsoft Visual C/C++ (MSVC) postupujte podle následujících kroků:

1. Ujistěte se, že máte nainstalovaný Microsoft Visual Studio s podporou pro C/C++.

2. Otevřete Developer Command Prompt for Visual Studio a přejděte do adresáře s projektem Nemesis.

3. Spusťte příkaz "`nmake -f Makefile.win`" pro přeložení a sestavení programu. Tento příkaz použije "`Makefile.win`", který je součástí projektu, a vytvoří spustitelný soubor "`lp.exe`" v aktuálním adresáři.

5.2 Formát vstupního souboru

Pro zachycení optimalizačního modelu bude program používat redukovanou a zobecněnou verzi formátu LP, který je popsán na webu `https://docs.gurobi.com/projects/optimizer/en/current/reference/fileformats/modelformats.html`. Až na návěstí End není pořadí jednotlivých sekcí fixní. Vstupní soubory mohou obsahovat následující sekce:

Maximize/Minimize. Výraz uvozující řádek se zápisem optimalizované účelové funkce. Program je schopen zpracovat standardní operátory +, −, ∗, = nebo závorky (), [] a {}. Oproti originální verzi ovšem nevyžaduje, aby jednotlivé operandy a operátory byly v matematických výrazech striktně odděleny mezerou (to platí i v ostatních sekcích souboru). Názvy proměnných tedy dříve uvedené operátory a závorky obsahovat nesmí. Při násobení není nutné použití operátoru ∗, například "2.5z" značí 2,5 krát z, zatímco "$z2$" je pouze název proměnné.

Subject To. Sekce obsahující seznam podmínek ve formátu "<název>: <výraz>". Navíc oproti účelové funkci mohou podmínky obsahovat porovnávací operátory <= a >=.

Bounds. Omezení hodnot rozhodovacích proměnných. V této sekci jsou povoleny pouze porovnávací operátory uvedené výše.

Generals. Obsahuje seznam použitých rozhodovacích proměnných oddělených znakem mezery.

End. Uvozuje konec souboru, tzn. že se vyskytuje vždy jako poslední a sekce uvedené za ním jsou syntaktickou chybou.

5.2.1 Příklad vstupního souboru

Zdrojový kód 5.1: Příklad vstupního souboru ve formátu LP

```
\ pořadí není fixní
Subject To
ca: (—2var_x + 1 * var_y) >= 2
cb: var_x — [2var_y] >= 2
Minimize
var_x — var_y
Generals
var_x var_y
Bounds
var_x >= 0
var_y >= 0
End
```

5.3 Použití programu

Program `lp.exe` je konzolová aplikace, která řeší úlohy lineárního programování zadané ve zjednodušeném formátu LP.

5.3.1 Spuštění programu

Program lze spustit z příkazové řádky s následujícími argumenty:

`lp.exe <input-file> [-o <output-file>] [--output <output-file>]`

- `<input-file>`: Povinný argument. Soubor s popisem úlohy ve formátu LP.
- `-o <output-file>`: Nepovinný argument. Určuje výstupní soubor, do kterého bude zapsáno řešení úlohy. Pokud tento argument není zadán, výsledek optimalizace bude vypsán na obrazovku.
- `--output <output-file>`: Nepovinný argument. Stejné jako v případě přepínače `-o`. Použití obou přepínačů `-o` a `--output` není chybou, program pak bude akceptovat poslední zadanou hodnotu.

Příklad spuštění

```
lp.exe data.lp -o vysledek.txt
```

Tento příkaz spustí program `lp.exe` s vstupním souborem `data.lp` a výstup bude zapsán do souboru `vysledek.txt`.

```
lp.exe data.lp --output vysledek.txt
```

Tento příkaz má stejný efekt jako předchozí, protože přepínače `-o` a `--output` jsou ekvivalentní.

```
lp.exe data.lp
```

Tento příkaz spustí program `lp.exe` s vstupním souborem `data.lp` a výsledek optimalizace bude vypsán na obrazovku. Varování a chybová hlášení jsou vždy vypisována na konzoli.

5.3.2 Možné návratové hodnoty a varování

Návratové hodnoty

- `EXIT_SUCCESS (0)`: Program úspěšně dokončil svou činnost.
- `EXIT_INVALID_INPUT_FILE (1)`: Vstupní soubor nebyl nalezen.
- `EXIT_INVALID_OUTPUT_FILE (2)`: Neplatný výstupní soubor.
- `EXIT_SYNTAX_ERROR (11)`: Chyba syntaxe ve vstupním souboru.
- `EXIT_MALLOC_ERROR (1001)`: Chyba při alokaci paměti.
- `EXIT_UNKNOWN_VAR (10)`: Neznámá proměnná ve vstupním souboru. Na konzoli je vypsán i její název.
- `EXIT_OBJECTIVE_UNBOUNDED (20)`: Cílová funkce je neomezená.
- `EXIT_OBJECTIVE_INFEASIBLE (21)`: Neexistuje žádné přípustné řešení.

Varování

Za běhu programu se může objevit následující varování:

- Nepoužitá proměnná:

```
Warning: unused variable '<název proměnné>'!
```

5.4 Ukázky běhu programu

5.4.1 Úloha s konečným řešením

Výpis 5.2: Ukázka běhu programu - konečné řešení

```
filipn@GG:/usr/Nemesis$ cat ./test.lp
\ uloha z uvodu zadani
Subject To \ poradi neni fixni
vyroba: x_1 + 2x_2 <= 8
baleni: 2 *x_1 + 1 * x_2 <= 6
Generals
x_2 x_1 x_3
Bounds
0 <= x_1
0 <= x_2
Maximize
3x_1+ 2 * x_2 \ ucelova funkce
End
filipn@GG:/usr/Nemesis$ ./lp.exe test.lp
Warning: unused variable 'x_3'!
x_2 = 3.333333
x_1 = 1.333333
filipn@GG:/usr/Nemesis$ echo $?
0
```

5.4.2 Úloha s neomezeným řešením

Výpis 5.3: Ukázka běhu programu - neomezené řešení

```
filipn@WP:/usr/Nemesis$ cat ./test.lp
\ uloha z uvodu zadani, ale s otocenymi operatory v Subject
   To
Subject To \ poradi neni fixni
vyroba: x_1 + 2x_2 >= 8
baleni: 2 *x_1 + 1 * x_2 >= 6
Generals
x_2 x_1 x_3
Bounds
0 <= x_1
0 <= x_2
Maximize
3x_1+ 2 * x_2 \ ucelova funkce
End
filipn@WP:/usr/Nemesis$ ./lp.exe test.lp
Warning: unused variable 'x_3'!
Objective function is unbounded.
```

```
filipn@WP:/usr/Nemesis$ echo $?
20
```

5.4.3 Úloha bez řešení

Výpis 5.4: Ukázka běhu programu - bez řešení

```
filipn@EZ:/usr/Nemesis$ cat ./test.lp
\ poradi neni fixni
Subject To
ca: (-2var_x + 1 * var_y) >= 2
cb: var_x - [2var_y] >= 2
Minimize
var_x - var_y
Generals
var_x var_y
Bounds
var_x >= 0
var_y >= 0
End
filipn@EZ:/usr/Nemesis$ ./lp.exe test.lp
No feasible solution exists.
filipn@EZ:/usr/Nemesis$ echo $?
21
```

Závěr 6

Cílem této semestrální práce bylo navrhnout a implementovat program, který řeší úlohy lineárního programování. Program byl úspěšně navržen a implementován v jazyce ANSI C, přičemž byly využity různé algoritmy a datové struktury pro efektivní zpracování a řešení optimalizačních úloh.

Během vývoje programu jsem se zaměřil na několik klíčových aspektů:

- Správné načtení a syntaktická analýza vstupních dat ve formátu LP.
- Implementace simplexové metody s velkým M pro nalezení optimálního řešení.
- Ošetření specifických scénářů, jako je neomezenost nebo neřešitelnost úlohy.
- Generování výstupu, který poskytuje uživateli přehledné a srozumitelné výsledky.

Program Nemesis byl testován na různých úlohách lineárního programování a prokázal svou schopnost efektivně a správně řešit širokou škálu optimalizačních problémů. Během testování jsem narazil na několik výzev, včetně specifického problému s nástrojem Valgrind, který hlásil chybu "impossible happened".

Celkově lze říci, že cíle semestrální práce byly splněny a program Nemesis představuje užitečný nástroj pro řešení úloh lineárního programování. V budoucnu by bylo možné program dále rozšířit o další optimalizační metody a vylepšit jeho uživatelské rozhraní pro ještě lepší uživatelský zážitek.

Seznam výpisů