UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Denisson Augusto Bastos Leal

SISTEMA DE FORMULAÇÃO DE RAÇÃO

Juazeiro - BA
2014

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Denisson Augusto Bastos Leal

SISTEMA DE FORMULAÇÃO DE RAÇÃO

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Engenharia da Computação pela Universidade Federal do Vale do São Francisco.

Orientador: Dr. Brauliro Gonçalves Leal
Coorientador: Dra Debora Cristine de Oliveira Carvalho
Dr. Fábio Nunes Lista

Juazeiro - BA
2014

Denisson Augusto Bastos Leal
Sistema de Formulação de Ração/ Denisson Augusto Bastos Leal. – Juazeiro - BA, 7 de março de 2014
Orientador: Dr. Brauliro Gonçalves Leal

Trabalho de Conclusão de Curso – Universidade Federal do Vale do São Francisco
Colegiado de Engenharia da Computação, 7 de março de 2014.

1. Formulação. 2. Ração. I. Universidade Federal do Vale do São Francisco. II. Leal, Brauliro Gonçalves. III. Título

CDU 02:141:005.7

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Denisson Augusto Bastos Leal

SISTEMA DE FORMULAÇÃO DE RAÇÃO

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação, pela Universidade Federal do Vale do São Francisco.

Dr. Brauliro Gonçalves Leal
Orientador

Me. Jorge Luis Cavalcanti Ramos
Convidado 1

Me. Rômulo Calado Pantaleão Camara
Convidado 2

Aprovado pelo Colegiado de Eng. _____ em _____/_____/2014

Resumo

Este trabalho apresenta uma revisão bibliográfica de nutrição animal, ressaltando pontos chaves para formulação de ração, e de programação linear, discutindo de forma genérica até chegar no algoritmo Simplex. Em seguida apresenta os materiais que serão utilizados na fase de desenvolvimento, como tabelas nutricionais para gado de corte e de leite, cavalos, caprinos, aves e suínos, e um projeto de software para um sistema para desktop feito em Java que, a partir das características dos animais e dos alimentos disponíveis com seus respectivos preços, encontra suas necessidades nutricionais tabeladas e utiliza o simplex para formular uma ração completa com menor custo possível. Como trabalhos futuros o software será feito, testado e implantado deixando margem para ampliação da quantidade de suas tabelas, podendo acrescentar necessidades de outros animais.

Palavras-chaves: nutrição. formulação de ração. programação linear. simplex.

Abstract

This is the english abstract.

Key-words: latex. abntex. text editoration.

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

A minha família, por todo amor, carinho e exemplo que me deram. Meus amigos Henrique Menezes, Núbia Araújo, Cintia Araujo e Artenia Almeida. Meus professores, em especial Brauliro Leal, pelo apoio e paciência.

"Aquele que diz que pode e aquele que diz
que não pode tem ambos razão".
Confúcio, filósofo chinês (551 a.C. - 479 a.C.)

Lista de abreviaturas e siglas

TI	Tecnologia da Informação
NRC	*National Research Council*
NAS	*National Academy of Sciences*
MS	Matéria Seca
PB	Proteína Bruta
NDT	Nutrientes Digestíveis Totais
PDR	Proteína Degradável no Rúmen
Ca	Cálcio
P	Fósforo
PL	Programação Linear
IDE	Ambiente de Desenvolvimento Integrado
UML	*Unified Modeling Language* ou Linguagem de Modelagem Unificada

Sumário

Lista de ilustrações

Lista de tabelas

1 Introdução

Com o aumento da renda da população brasileira e o crescimento da população mundial, a exigência do setor produtivo de carnes cresce proporcionalmente. A perspectiva é que o avanço na produção de frango seja de 56,1%, a bovina de 32,3% e a suína 22% em um período de dez anos (AVESUI. . . , 2012). Como a zona rural vem diminuindo, resta como opção viável otimizar na produção, como abater o animal mais novo e bem nutrido. Nesse contexto, a zootecnia atua como a ciência que estuda as melhorias na produção animal, visando sempre produzir o máximo, no menor tempo, com o menor gasto e considerando ainda o bem estar do animal.

Um problema muito comum na zootecnia é o balanceamento de nutrientes para um melhor aproveitamento/desenvolvimento do animal sem déficit e nem excesso nutricional, em outras palavras, formar uma ração completa com os alimentos disponíveis. Cada animal possui uma tabela com suas necessidades e ela é alterada para cada fase que o animal se encontra, em contra partida, cada alimento possui uma relação de nutrientes fornecidos. A Seção 2.3.1 mostra como é feito o cálculo manual e da uma ideia da dificuldade, falta de precisão e vulnerabilidade a erros desse método.

O Simplex é um algoritmo de programação linear que resolve o problema de maximizar ou minimizar uma equação, tendo uma série de restrições. O Capítulo 3 mostra detalhadamente o funcionamento do método e apresenta alguns algoritmos que resolvem o problema.

O setor de Tecnologia da Informação (TI) tem ajudado o desenvolvimento da agroindústria. A manipulação de grandes volumes de dados e a automatização de tarefas manuais ajuda a agilizar e dar mais confiança aos processos.

O chamado método computacional consistem em utilizar programação linear para balancear os nutrientes de uma ração de uma forma que supra as necessidades de um certo animal. Além de utilizar o computador para realizar os cálculos, ainda possui a vantagem de minimizar o custo.

1.1 Objetivo Geral

Este trabalho tem como objetivo modelar e desenvolver um sistema que, dado um animal e os alimentos disponíveis com seus respectivos preços, faça um balanceamento de nutrientes, com o menor custo possível.

1.2 Objetivo Específico

Para atingir o objetivo geral as etapas abaixo devem ser cumpridas:

- Buscar tabelas nutricionais dos animais para alimentar a base de dados;
- Implementar o algoritmo Simplex para formulação de menor custo;
- Estruturar dados da forma mais genérica possível, para possibilitar inserção de novas tabelas nutricionais;
- Montar uma interface simples e amigável, para que os usuários consigam usar sem dificuldades.

2 Nutrição Animal

O balanço nutricional consiste em estabelecer ganhos e perdas de massa corporal de acordo com a quantidade ingerida e eliminada de uma substância (ANDRIGUETTO, 1981).

2.1 Exigências nutricionais

Cada animal possui necessidades nutricionais diferentes, além disso há fatores que também devem ser considerados, tais como a idade, estágio de produção, sexo, raça e condições ambientais, desta forma para cada diferente conjunto de características há uma exigência nutricional que quando seguida obtém um programa de nutrição eficiente (NRC, 2001).

A Tabela 1 mostra uma tabela de exigências nutricionais para frango de corte macho, de desempenho médio e com 1 a 7 dias vida.

Tabela 1 – Exigências nutricionais para frango de corte macho, de desempenho médio e com 1 a 7 dias vida.

Nutrientes	Quantidade
Energia Metabolizável	2.950 kcal/kg
Proteína	22,20 %
Cálcio	0,920 %
Fósforo Disponível	0,470 %
Fósforo Digestível	0,395 %
Sódio	0,220 %
Lisina Dig.	1,310 %
Metionina Dig.	0,511 %
Metionina + Cistina Dig	0,944 %
Treonina Dig	0,852 %
Triptofano Dig	0,223 %
Arginina Dig	1,415 %
Glicina + Serina Dig	1,926 %
Valina Dig	1,009 %
Isoleucina Dig	0,878 %

Fonte: Rostagno et al. (2011)

Existe dois diferentes tipos de sistemas gástricos: o monogástrico que possui apenas um estomago que é responsável pelo processo de quebra do alimento e o ruminante que

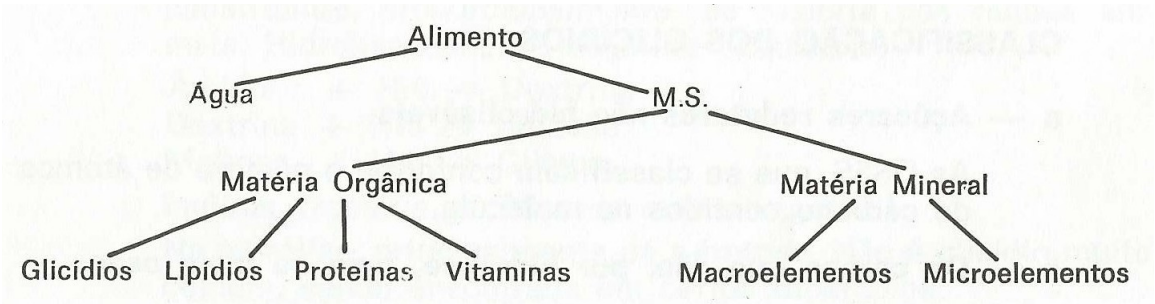
possui vários estômagos cada um com funções digestivas diferentes, por essa razão são chamados também de poligástrico.

2.2 Composição dos alimentos

Jacouot et al. (1958) definem alimento como "uma substância que, consumida por um individuo, é capaz de contribuir para assegurar o ciclo regular de sua vida e a sobrevivência da espécie à qual pertence".

Como mostra a Figura 1, o alimento é composto por água (umidade) e matéria seca(MS), no qual o seu valor nutritivo depende, que por sua vez é dividida em matéria orgânica (glicídios, lipídios, proteínas e vitaminas) e matéria mineral (macroelementos e microelementos) (ANDRIGUETTO, 1981).

Figura 1 – Esquema da composição química dos alimentos



Fonte: Andriguetto (1981)

Para caracterização de um alimento é utilizado a tabela nutricional que contém as principais informações da sua composição, como a Tabela 2 que mostra a composição nutricional do milho.

Tabela 2 – Informações nutricionais do milho

MS(Kg)	PB(%)	NDT(%)	PDR(%)	Ca(%)	P(%)
72.5	5.8	60.9	2.3	0.02	0.18

Fonte: NRC (2007)

2.3 Formulação de Rações

Como não existe um alimento naturalmente completo, ou seja, com todos os nutrientes necessários para uma dieta, há necessidade de formular ração composta de

vários alimentos de forma que supra as necessidades do animal (ANDRIGUETTO, 1981).

O termo ração é empregado para o total de alimentos consumidos em um período de 24 horas e quando associado com o termo balanceado diz que está nutricionalmente adequado para atender às necessidades do animal que o consome nesse período (ANDRIGUETTO, 1981).

No século XIX surgiram os primeiros trabalhos relacionados com o balanceamento de rações. Thaer foi o pioneiro ao dar um tratamento científico na alimentação de rebanhos e publicou, em 1810, a primeira tabela conhecida como Equivalentes Feno (CULLISON, 1975).

Em 1959, o *National Research Council* (NRC) da *National Academy of Sciences* (NAS) publicaram um conjunto abrangente de tabelas contendo as exigências nutricionais de animais e a composição de alimentos (SAKOMURA; ROSTAGNO, 2007). No Brasil, a Universidade Federal de Viçosa publicou, no ano 2000, a primeira Edição das Tabelas Brasileiras para Aves e Suínos (ROSTAGNO et al., 2011). Periodicamente, essas publicações têm sido revisadas e atualizadas, servindo de base para nutricionistas da área.

A formulação de rações é feita em quatro etapas

- Caracterizar o animal

  Consiste em classificar o animal de acordo com fatores que influenciem na sua dieta, como idade e peso.

- Definir as exigências

  As exigências dos animais podem ser consultadas em tabelas, como as do *National Research Council* (NRC) e as Tabelas Brasileiras para Aves e Suínos, publicadas pela Universidade Federal de Viçosa (SAKOMURA; ROSTAGNO, 2007).

  Essas tabelas costumam ter como entrada as características do animal e como resposta tem as exigências de cada nutriente.

- Seleção de alimentos

  Os alimentos e ingredientes devem ser escolhidos e combinados de tal maneira que a ração seja nutricionalmente equilibrada (SAKOMURA; ROSTAGNO, 2007).

  As informações nutricionais dos alimentos podem ser encontradas nos NRCs, em trabalhos científicos, entre outros.

- Utilizar um método para formulação

  Deve ser escolhido um método de acordo com a conveniência e com a precisão desejada, adiante está descrito as características de alguns dos métodos mais utilizados.

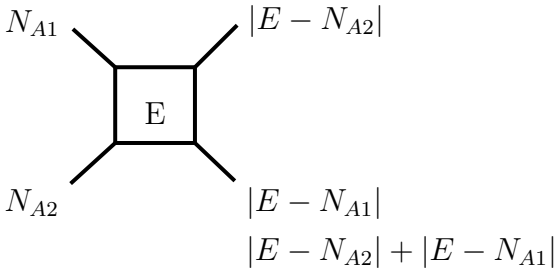
2.3.1 Métodos Manuais

Existem vários métodos manuais para formular rações, os mais conhecidos são Método da Tentativa, Quadrado de Pearson e Equações Algébricas.

O Método da Tentativa consiste em aumentar ou diminuir a quantidade dos alimentos até que as exigências do animal sejam atendidas. Este método possui base empírica e é utilizado por quem tem experiência.

O Quadrado de Pearson é um método que considera apenas um determinado nutriente e pode ser utilizado dois alimentos ou duas misturas de alimentos. A Figura 2 mostra como é feito o cálculo, finalizando pelas Equações 2.1 e 2.2.

Figura 2 – Quadrado de Pearson.



Fonte: Produzido pelo próprio autor.

$$\text{Alimento 1} = \frac{|E - N_{A1}|}{|E - N_{A2}| + |E - N_{A1}|}100 \tag{2.1}$$

$$\text{Alimento 2} = \frac{|E - N_{A2}|}{|E - N_{A2}| + |E - N_{A1}|}100 \tag{2.2}$$

Onde E é a exigência de um certo nutriente, N_{A1} é a concentração do nutriente no alimento 1 e N_{A2} é a concentração do nutriente no alimento 2.

O método das Equações Algébricas, semelhante ao Quadrado de Pearson, é relativamente simples, consiste em montar uma equação de quantidade e uma equação considerando um determinado nutriente e em seguida resolver o sistema linear, a Equação 2.3 mostra o procedimento geral, onde A_1 é a quantidade do alimento 1, A_2 é a quantidade do alimento 2, Z é a folga, E é a exigência de um certo nutriente, N_{A1} é a concentração do nutriente no alimento 1 e N_{A2} é a concentração do nutriente no alimento 2.

$$\begin{array}{rcrcrcl} A_1 & + & A_2 & + & Z & = & 100 \\ A_1N_{A1} & + & A_2N_{A2} & & & = & E \end{array} \tag{2.3}$$

2.3.2 Método Computacional

O principal problema dos métodos manuais é a limitação da quantidade de itens. Umas das formas de resolver esse problema é utilizar um método chamado de Programação Linear (PL), pois além de fazer o balanceamento considerando vários nutrientes, ele ainda tem a vantagem de encontrar a ração de menor custo (SAKOMURA; ROSTAGNO, 2007).

Segundo Sakomura e Rostagno (2007), "para se calcular rações utilizando essa metodologia, são necessárias as seguintes informações: preços dos alimentos, alimentos disponíveis, composição dos alimentos e exigências nutricionais dos animais".

Devido a complexidade dos cálculos da Programação Linear e a possibilidade de automatizar o processo, esse método é usado através de programas de computador.O uso do software abre a possibilidade de usar muitas restrições nutricionais e a quantidade de alimentos que for necessário. Além de que, o único conhecimento necessário para ser usuário desse tipo de software é sobre nutrição, ou seja, o usuário não precisa entender os passos da PL.

3 Programação Linear

Existem vários problemas que consistem em encontrar o menor custo ou o maior lucro levando em consideração séries de restrições e recursos limitados. Se for possível especificar o objetivo como uma função linear de algumas variáveis e especificar as restrições dos recursos como igualdades ou desigualdades sobre essas variáveis, então teremos um problema de programação linear (CORMEN, 2002).

De uma forma geral o problema consiste em uma equação, de n variáveis, que queremos maximizar ou minimizar

$$a_0x_0 + a_1x_1 + ... + a_nx_n \tag{3.1}$$

e que está sujeito a m equações de restrição

$$\begin{array}{cccccc} a_{00}x_0 & +a_{01}x_1 & ... & a_{0n}x_n & \leq & c_0 \\ a_{10}x_0 & +a_{11}x_1 & ... & a_{1n}x_n & \leq & c_1 \\ \vdots & & \ddots & & & \\ a_{m0}x_0 & +a_{m1}x_1 & ... & a_{mn}x_n & \leq & c_m \end{array} \tag{3.2}$$

onde

$$x_0, x_1, x_2, ..., x_n \geq 0 \tag{3.3}$$

As equações 3.1, 3.2 e 3.3 são conhecidas como a Forma Padrão de um problema de programação linear (CORMEN, 2002). Para um caso bidimensional, as restrições são retas que limitam o plano xy formando assim uma região de interseção, como mostra a Figura 3.

3.1 Forma Padrão e Forma Relaxada

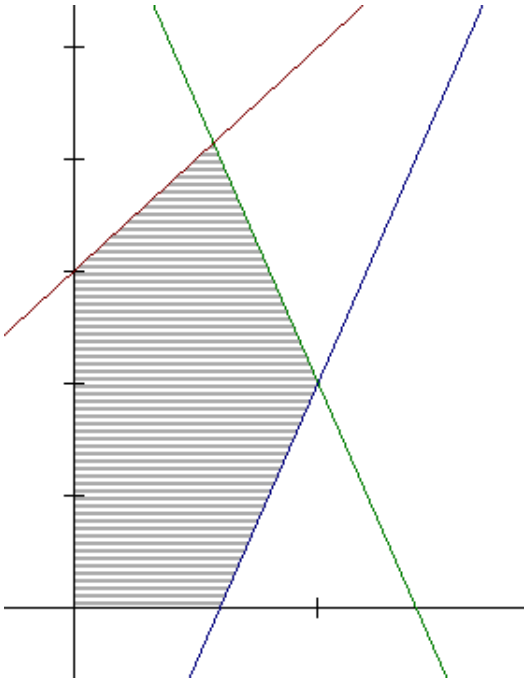
A forma relaxada é um modelo de entrada para que o algoritmo facilmente identifique as equações de restrição e qual deve ser minimizado ou maximizado. A conversão de um problema qualquer é feita primeiramente para a forma padrão, para em seguida encontrar a sua forma relaxada.

A conversão para forma padrão, que consiste em, primeiramente, retirar as variáveis que não possuem restrições de não negatividade. Então uma variável x_j, que não possui restrição, deve ser substituída por $x'_j - x''_j$, onde $x'_j, x''_j \geq 0$ (CORMEN, 2002).

Em seguida é necessário retirar as restrições de igualdade. Para isso basta substituir

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \tag{3.4}$$

Figura 3 – Interseção de restrições.



Fonte: Produzido pelo próprio autor

por

$$\begin{array}{lll} \sum_{j=1}^{n} & a_{ij}x_j & \geq \quad b_i \\ \sum_{j=1}^{n} & a_{ij}x_j & \leq \quad b_i \end{array} \tag{3.5}$$

Note que a única forma de satisfazer as duas equações é estando numa condição de igualdade (CORMEN, 2002).

Em seguida, as restrições de maior que ou igual serão convertidas em menor que ou igual. Para isso basta multiplicar por -1, ou seja, quando temos

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i \tag{3.6}$$

transformamos na equivalência

$$\sum_{j=1}^{n} -a_{ij}x_j \leq -b_i \tag{3.7}$$

Para resolver de maneira eficiente um programa linear com o algoritmo simplex, ele deve ser expressado de forma que as únicas restrições de desigualdade são as de não negatividade, chamada de forma relaxada (CORMEN, 2002). Com base nisso, as restrições na forma

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \tag{3.8}$$

devem ser convertidas para restrições de igualdade

$$\sum_{j=1}^{n} a_{ij}x_j - b_i = x_{n+i} \tag{3.9}$$

onde $x_{n+i} \geq 0$. x_{n+i} é chamada de variável relaxada porque ela mede a folga entre os lados esquerdo e direito da equação (CORMEN, 2002).

3.2 Algoritmo Simplex

Para descrever o algoritmo simplex a definição de alguns conceitos torna-se necessária. Dado uma equação na forma relaxada, as *variáveis básicas* são as variáveis do lado esquerdo da igualdade, enquanto as *variáveis não básicas* são as do lado direito. Cada iteração do algoritmo simplex consiste em definir as variáveis não básicas como 0 e calcular as variáveis básicas. Definimos este como processo de *encontrar solução básica*. O *valor objetivo* é o resultado da equação que vamos maximizar ou minimizar.

O Cormen (2002) explica que "Para alcançar um aumento no valor objetivo, escolhemos uma variável não básica tal que, se fossemos aumentar o valor dessa variável desde 0, então o valor objetivo também aumentaria. A quantidade pela qual aumentamos a variável é limitada pelas outras restrições. Em particular, nós aumentamos até alguma variável básica se tornar 0. Em seguida, reescrevemos a forma relaxada, trocando as funções dessa variável básica e da variável não básica escolhida." Esse processo simplesmente converte uma forma relaxada em uma forma relaxada equivalente até a solução ótima se tornar óbvia.

A Figura 4 mostra um algoritmo de criação de pivô, que consiste em escolher a variável de entrada não básica x_e e a variável de saída básica x_l e trocar suas funções (CORMEN, 2002).

O algoritmo Simplex da Figura 6 é uma implementação alto nível dos passos descritos a cima, que recebe como entrada um programa linear em forma padrão e tem como saída uma solução ótima. Ele usa o algoritmo da Figura 5 que determina se o programa linear é impossível ou retorna sua forma relaxada, caso seja possível (CORMEN, 2002).

Figura 4 – Algoritmo de criação de pivô

```
PIVOT(N, B, A, b, c, v, l, e)
1  ▷ Calcular os coeficientes da equação para a nova variável básica x_e.
2  b̂_e ← b_l/a_le
3  for cada j ∈ N − {e}
4      do â_ej ← a_lj/a_le
5  â_el ← 1/a_le
6  ▷ Calcular os coeficientes das restrições restantes.
7  for cada i ← B − {l}
8      do b̂_i ← b_i − a_ie b̂_e
9          for cada j ∈ N − {e}
10             do â_ij ← a_ij − a_ie â_ej
11             â_ij ← a_ij − a_ie â_el
12 ▷ Calcular a função de objetivo.
13 v̂ ← v + c_e b̂_e
14 for cada j ∈ N − {e}
15     do ĉ_j ← c_j − c_e â_ej
16 ĉ_l ← − c_e â_el
17 ▷ Calcular novos conjuntos de variáveis básicas e não básicas.
18 N̂ = N − {e} ∪ {l}
19 B̂ = B − {l} ∪ {e}
20 return (N̂, B̂, Â, b̂, ĉ, v̂)
```

Fonte: Cormen (2002, p. 630)

Figura 5 – Algoritmo para encontrar solução básica inicial

```
INITIALIZE-SIMPLEX(A, b, c)
 1  seja l o índice do mínimo b_i
 2  if b_l ≥ 0                    ▷ A solução básica inicial é possível?
 3     then return ({1, 2, ..., n}, {n + 1, n + 2, ..., n + m}, A, b, c, 0)
 4  formar L_aux adicionando −x_0 ao lado esquerdo de cada equação
        e definindo a função de objetivo como −x_0
 5  seja (N, B, A, b, c, v) a forma relaxada resultante para L_aux
 6  ▷ L_aux tem n + 1 variáveis não básicas e m variáveis básicas.
 7  (N, B, A, b, c, v) ← PIVOT(N, B, A, b, c, v, l, 0)
 8  ▷ A solução básica é agora possível para L_aux.
 9  repetir o loop while das linhas 2 a 11 de SIMPLEX até encontrar uma
        solução ótima para L_aux
10  if a solução básica define x̄_0 = 0
11     then return a forma final relaxada com x_0 removido e
                a função de objetivo original restaurada
12  else return "impossível"
```

Fonte: Cormen (2002, p. 644)

Figura 6 – Algoritmo Simplex

```
SIMPLEX(A, b, c)
 1  (N, B, A, b, c, v) ← INITIALIZE-SIMPLEX(A, b, c)
 2  while algum índice j ∈ N tem c_j > 0
 3      do escolher um índice e ∈ N para o qual c_e > 0
 4         for cada índice i ∈ B
 5             do if a_ie > 0
 6                    then Δ_i ← b_i/a_ie
 7                    else Δ_i ← ∞
 8         escolher um índice l ∈ B que minimize Δ_i
 9         if Δ_l = ∞
10            then return "ilimitado"
11            else (N, B, A, b, c, v) ← PIVOT(N, B, A, b, c, v, l, e)
12  for i ← 1 to n
13      do if i ∈ B
14            then x̄_i ← b_i
15            else x̄_i ← 0
16  return (x̄_1, x̄_2, ..., x̄_n)
```

Fonte: Cormen (2002, p. 631)

4 Materiais e Métodos

4.1 Tabelas de Exigências Nutricionais

A Tabela 3 lista as tabelas de exigências nutricionais, informando a fonte e os animais que ela abrange.

Tabela 3 – Relação de tabelas nutricionais

Fonte	Animais
NRC (2000)	Gado de corte
NRC (2001)	Gado de leite
NRC (2007)	Cavalos e caprinos
Rostagno et al. (2011)	Aves e suínos

Fonte: Produzido pelo próprio autor

Essas tabelas serão utilizadas durante o processo de definição de exigências, como descrito na Seção 2.3. Onde as características do animal serão utilizadas para encontrar suas necessidades nutricionais.

4.2 Linguagem Java

Java é uma linguagem de programação de alto nível, que possui como uma de suas principais características a portabilidade (DEITEL, 2005). A sua orientação a objetos, permite uma fácil manutenção e abstração com o objeto real.

O conjunto de bibliotecas do Java permite resolver facilmente diversos tipos de problemas, deixando o foco para o algoritmo principal. Dentre vários pacotes, os principais são: *XStream* e *xmlpull* para converter xml ou json em objetos e vise-versa; *FileWriter* para manipulação de arquivos; *JExcelAPI* para escrever arquivos xls e *Swing* para interface gráfica.

4.3 Classe Simplex

A classe Simplex a ser utilizada faz parte de um conjunto de códigos disponibilizados em (SEDGEWICK; WAYNE, 2013), pois é uma implementação em java bastante utilizada e testada. A código fonte está disponível no Anexo A.

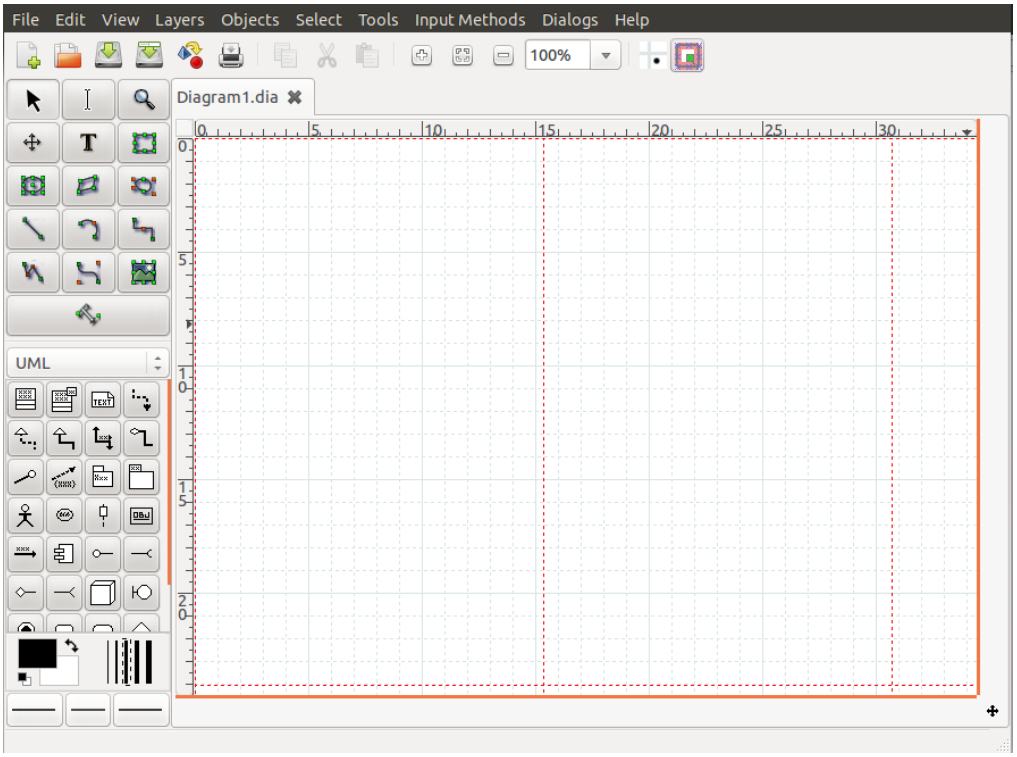
4.4 Ferramentas Utilizadas

Nessa seção é apresentada a principal ferramenta utilizada na documentação e a que será desenvolvimento.

4.4.1 Editor de Diagramas DIA

O DIA é um editor de diagramas simples e multiplataforma, que da suporte a toda UML 2.0 e foi utilizada para documentação do projeto. A Figura 7 mostra a tela inicial do software.

Figura 7 – Editor de Diagramas Dia

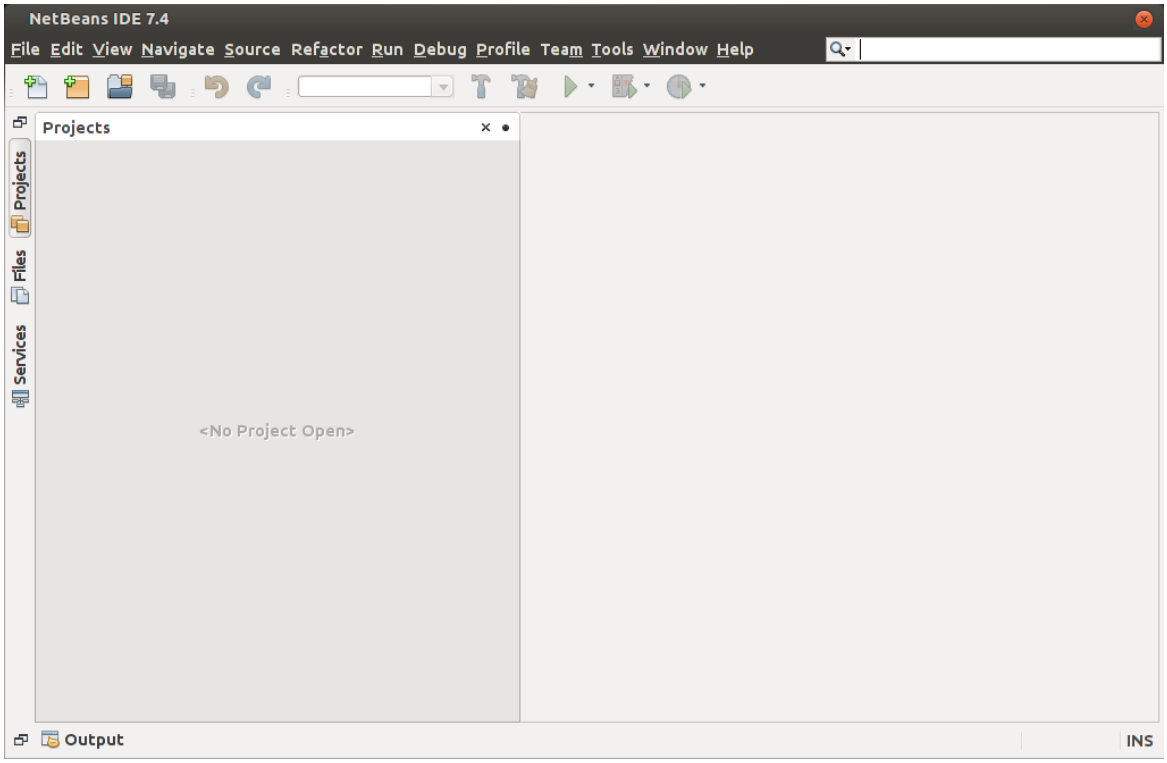


Fonte: Produzido pelo próprio autor

4.4.2 IDE NetBeans

NetBeans é uma IDE multiplataforma e com suporte a várias linguagens, possui a característica de ser simples de usar e montar interfaces. Nesse projeto foi utilizada a versão 7.3.1. A Figura 8 mostra a tela inicial do software.

Figura 8 – IDE NetBeans



Fonte: Produzido pelo próprio autor

5 Planejamento de Software

Uma linguagem visual muito utilizada para modelar sistemas computacionais é a Linguagem de Modelagem Unificada (UML), que tem como objetivo definir características do software, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e as necessidades físicas do equipamento que o software deverá ser implantado (GUEDES, 2004).

A engenharia de software é a área voltada para o planejamento dos softwares, na qual está presente em toda documentação da especificação, do desenvolvimento e da manutenção, onde visa organização, produtividade e qualidade.

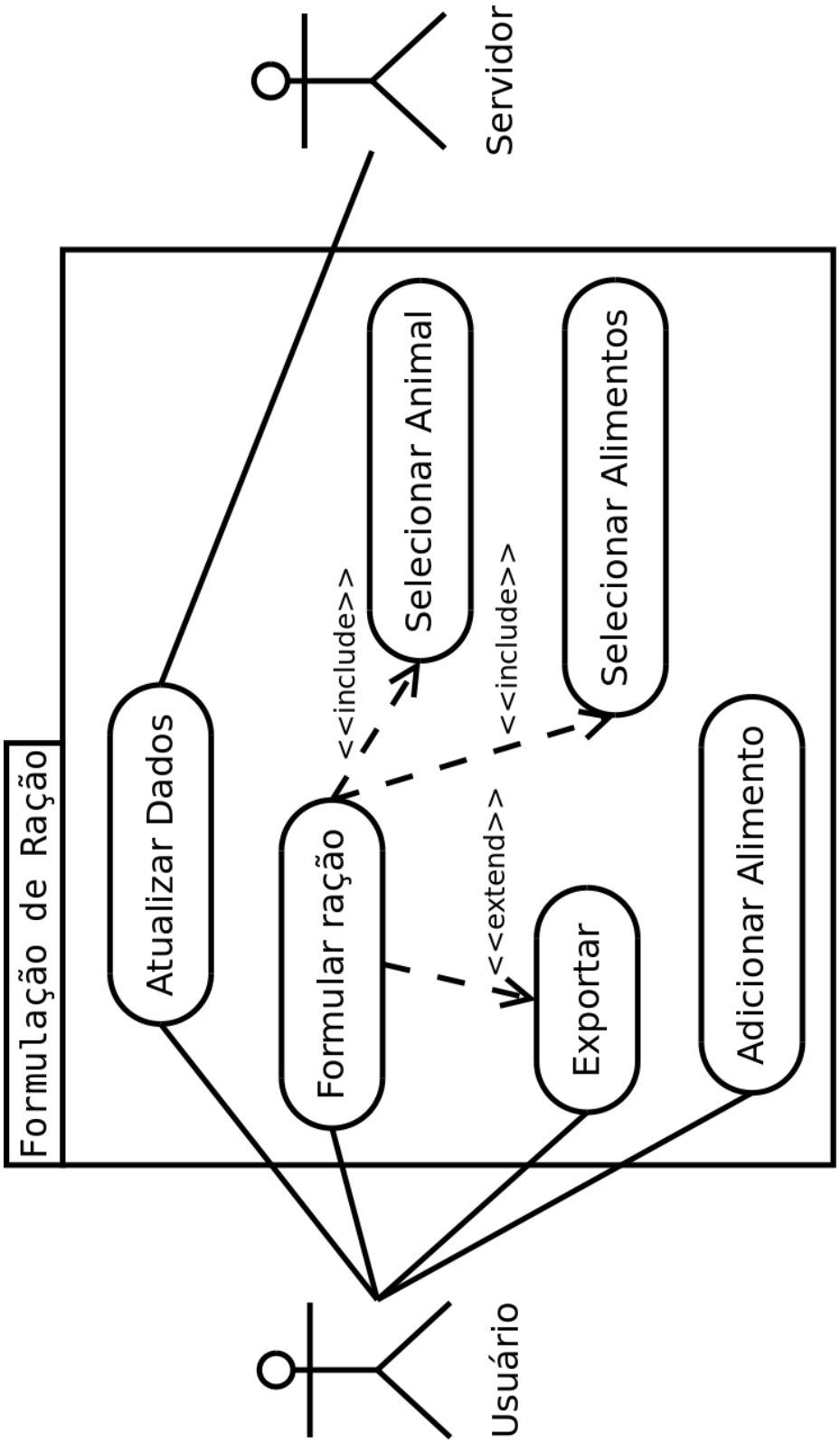
5.1 Caso de Uso

A Figura 9 mostra o Diagrama de Casos de Uso para o sistema de formulação de ração. Para cada um dos seis casos de uso, é listada a seguir uma descrição detalhada e um Diagrama de Atividades.

Caso de uso 1 - Atualizar dados

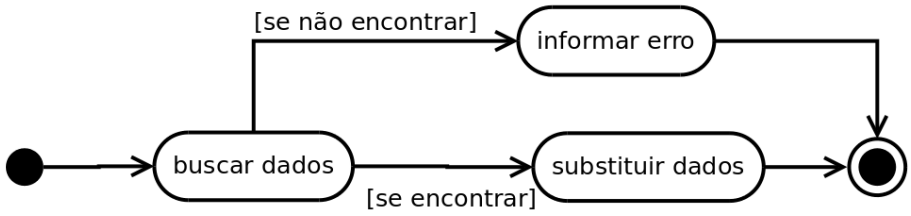
- Atores:
  - Usuário;
  - Servidor.
- Pré-condição:
  - Está conectado a Internet.
- Fluxo de evento primário:
  - O usuário deve clicar em "Atualizar Dados";
  - O sistema busca os dados no servidor;
  - Os dados antigos são substituídos.
- Fluxo de evento secundário:
  - Caso encontre algum problema informe ao usuário.
- Pós-condição:
  - Os dados devem permanecer salvos localmente.

Figura 9 – Diagrama de Casos de Uso



Fonte: Produzido pelo próprio autor

Figura 10 – Diagrama de Atividades - Atualizar dados



Fonte: Produzido pelo próprio autor

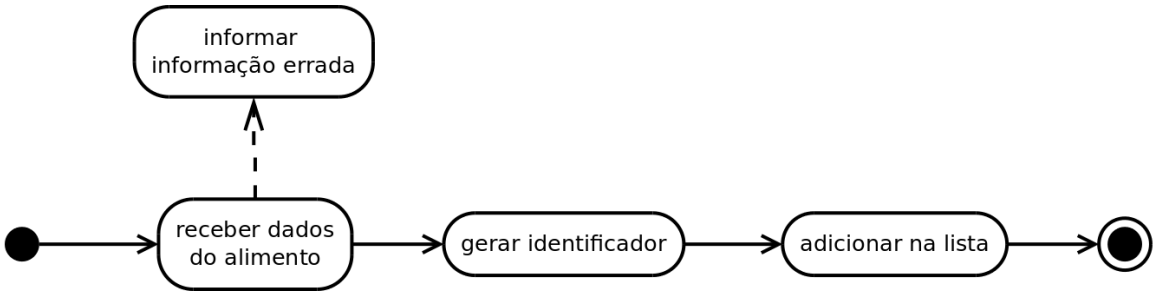
Caso de uso 2 - Adicionar Alimento

- Atores:
  - Usuário.
- Pré-condição:
- Fluxo de evento primário:
  - O usuário deve clicar em "Adicionar Alimento";
  - O Usuário informa os dados do alimento e confirma;
  - É gerado um identificador para diferenciar dos demais;
  - O sistema atualiza a lista de alimentos.
- Fluxo de evento secundário:
  - Se for inserido valores inválidos o sistema informa e espera uma correção;
  - A operação pode ser cancelada.
- Pós-condição:
  - Alimento salvo e disponível na lista.

Caso de uso 3 - Formular Ração

- Atores:
  - Usuário.
- Pré-condição:
  - Possuir dados de alimentos e animais.

Figura 11 – Diagrama de Atividades - Adicionar Alimento



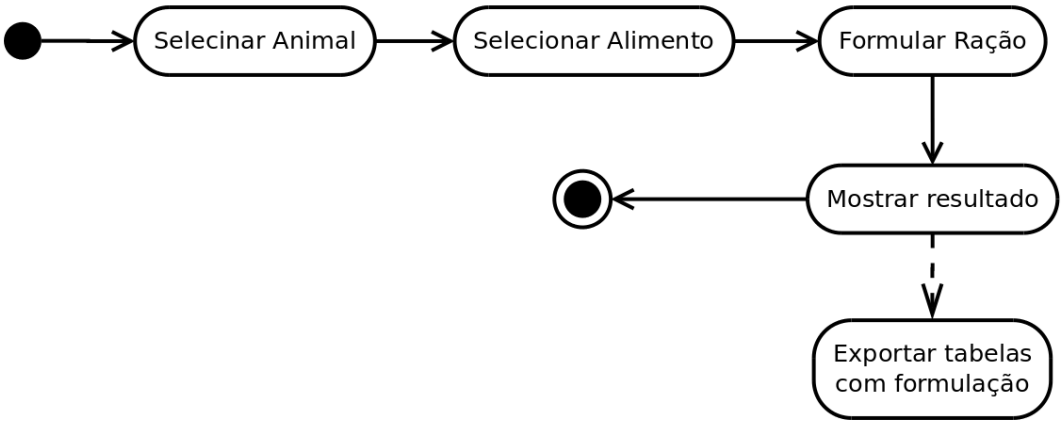
Fonte: Produzido pelo próprio autor

- Fluxo de evento primário:
  - Começa com o início do sistema ou quando o usuário clicar em "Novo";
  - Usa Selecionar Animal;
  - Usa Selecionar Alimentos;
  - Mostra a ração formulada.
- Fluxo de evento secundário:
  - No final o usuário pode usar Exportar;
  - Pode voltar em qualquer etapa;
  - No final o usuário pode informar quantos quilogramas de ração e o sistema deve calcular com base nas porcentagens já encontrada.
- Pós-condição:
  - Deve aparecer uma ração formulada.

Caso de uso 4 - Selecionar Animal

- Atores:
  - Usuário.
- Pré-condição:
  - O usuário está formulando ração.
- Fluxo de evento primário:
  - O usuário informa qual o animal está trabalhando;

Figura 12 – Diagrama de Atividades - Formular Ração



Fonte: Produzido pelo próprio autor

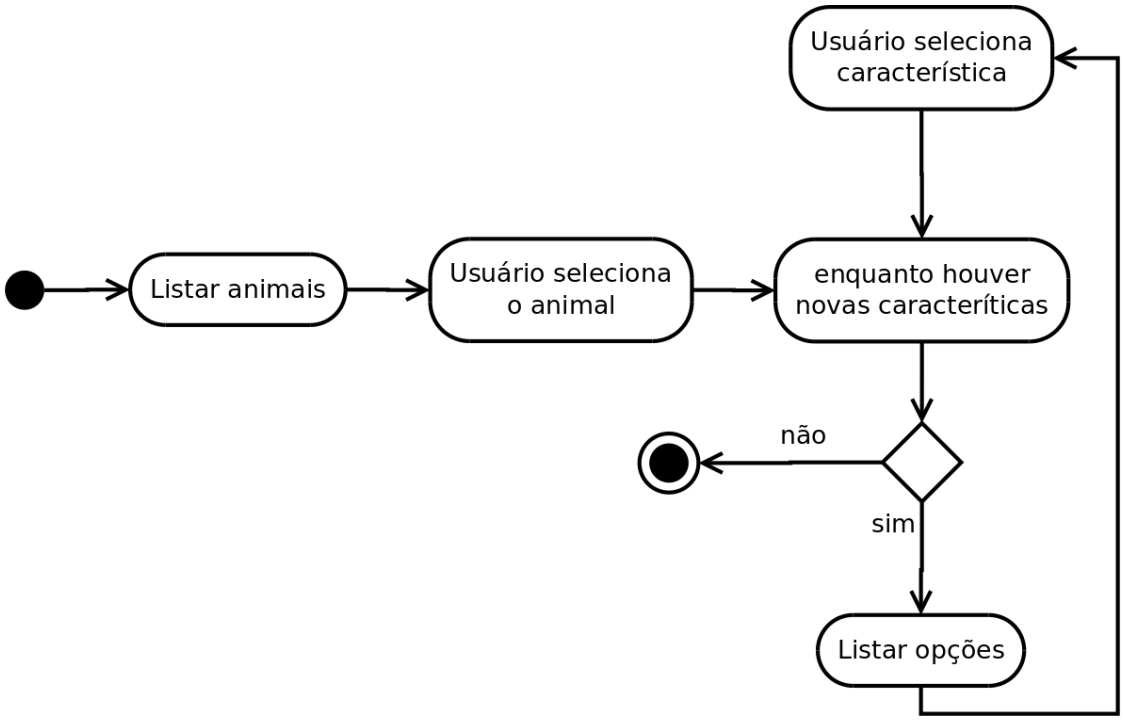
  - O sistema solicita as características especificas para o animal;
  - O usuário informa as características;
  - Vai para próxima etapa.

- Fluxo de evento secundário:
  - O animal pode ser redefinido e o sistema deve seguir do passo 2.
- Pós-condição:
  - Animal selecionado.

Caso de uso 5 - Selecionar Alimentos

- Atores:
  - Usuário.
- Pré-condição:
  - O usuário está formulando ração.
- Fluxo de evento primário:
  - O sistema deve listar todos os alimentos disponíveis com suas principais características;
  - O usuário informa quais devem ser utilizados na formulação;

Figura 13 – Diagrama de Atividades - Selecionar Animal



Fonte: Produzido pelo próprio autor

  - Vai para próxima etapa.

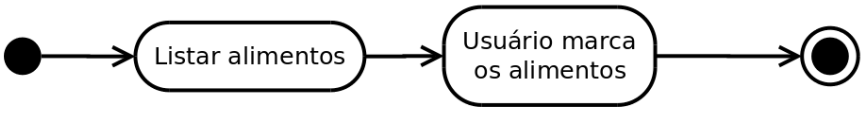
- Fluxo de evento secundário:

  - Se um novo alimento for inserido deve ser acrescentado na lista;
  - O usuário pode voltar para seleção do animal.

- Pós-condição:

  - Alimentos selecionados selecionado.

Figura 14 – Diagrama de Atividades - Selecionar Alimentos

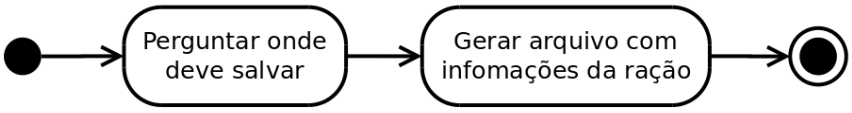


Fonte: Produzido pelo próprio autor

Caso de uso 6 - Exportar

- Atores:
  - Usuário.
- Pré-condição:
  - Ração deve está formulada.
- Fluxo de evento primário:
  - O usuário clica em Exportar;
  - O usuário informa onde deve ser salvo;
  - O sistema gera um arquivo com os dados da ração.
- Fluxo de evento secundário:
- Pós-condição:
  - Arquivo com os dados da ração é gerado.

Figura 15 – Diagrama de Atividades - Exportar



Fonte: Produzido pelo próprio autor

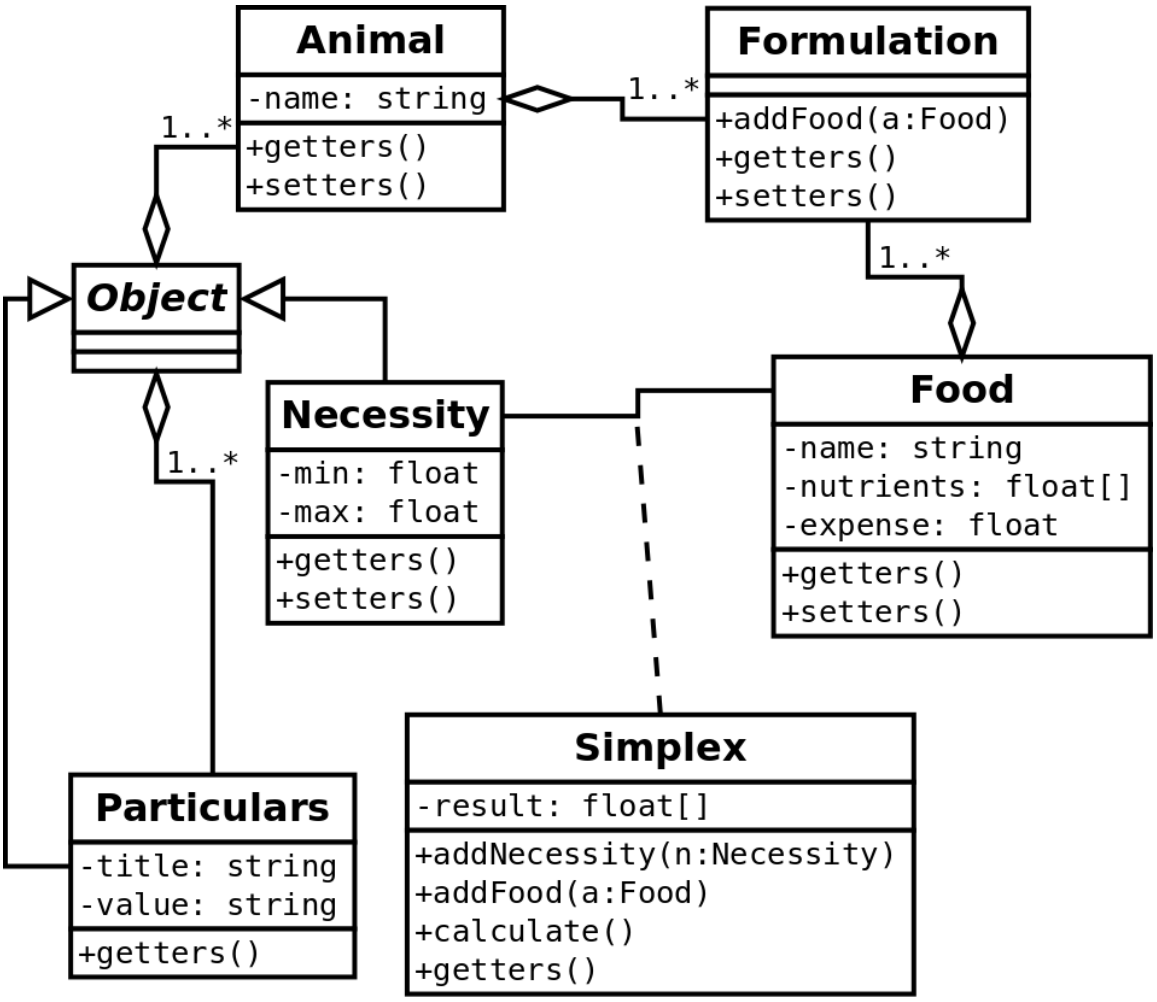
5.2 Diagrama de Classes

A Figura 16 mostra o Diagrama de Classes do software de formulação de ação, no qual os métodos *getters* e *setters* representam o conjunto dos métodos para acessar os atributos.

5.3 Estrutura dos Dados

Os dados estão organizados em uma variação de Árvore B+, onde a raiz é o animal, as folhas são as necessidades do animal e os demais nós são suas características. A navegação na Árvore se dar por escola do animal e de suas características. A Figura 17 mostra como os dados estarão estruturados durante a execução do programa.

Figura 16 – Diagrama de Classes



Fonte: Produzido pelo próprio autor

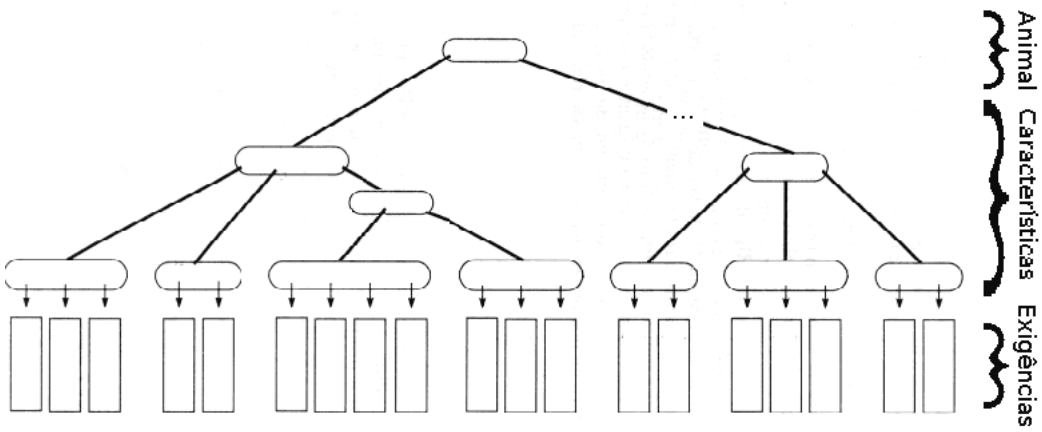
5.4 Protótipo de Interface

A Figura 18 mostra um protótipo de tela inicial do software, na qual tem as opções de escolha do animal e de suas respectivas características, na qual aparecerá de forma dinâmica, e quando concluído basta clicar no botão *Próximo*. A tela seguinte, Figura 19, contém uma tabela com todos os alimentos cadastrados e seus nutrientes, onde o usuário deve selecionar os que quer usar e clicar no botão *Calcular*.

Quando a ração for finalizada a tela da Figura 20 vai apresentar o resultado com a quantidade de cada alimento e de cada nutriente.

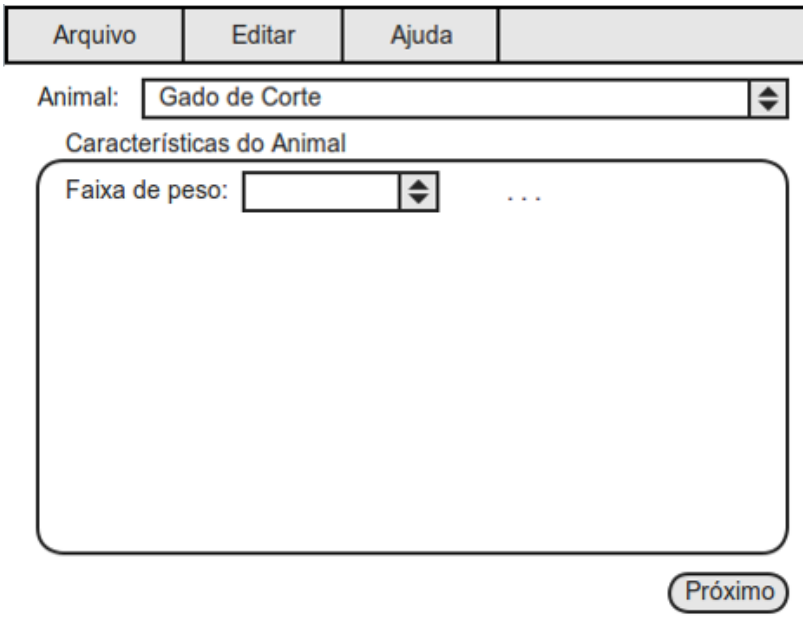
A opção *Arquivo* do menu superior contém as subopções de *Novo*, *Abrir*, *Salvar*, *Salvar como* e *Fechar*, que é o conjunto para manipulação do arquivo que será salvo no disco. *Editar* possui as subopções de *Editar exigências* e *Editar alimentos*, essas edições serão mantidas apenas no arquivo que está sendo editado, ou seja, para uma nova formulação os

Figura 17 – Organização dos dados



Fonte: Produzido pelo próprio autor

Figura 18 – Tela de escolha do animal



Fonte: Produzido pelo próprio autor

dados antigos que serão carregados. O *Manual* e o *Sobre* fazem parte da opção *Ajuda*.

Figura 19 – Tela de escolha dos alimentos

Fonte: Produzido pelo próprio autor

Figura 20 – Tela que mostra resultado da formulação

Fonte: Produzido pelo próprio autor

6 Conclusão e Trabalhos Futuros

Neste trabalho foi apresentado um sistema para formulação de ração de custo mínimo. Inicialmente foi apresentado conceitos teóricos sobre nutrição de animais e os conhecimentos necessários para o balanceamento nutricional. Em seguida foi discutido o algoritmo Simplex que maximiza ou minimiza uma equação a partir de restrições.

Foi apresentado a documentação inicial do software, começando pelo caso de uso, sua descrição e o diagrama de atividade, em seguida o diagrama de classes e um protótipo de interface.

Com certa frequência são feitos trabalhos com nutrição animal, sendo criada tabelas para animais ainda não estudados e também adicionadas outras para casos mais específicos como uma diferente região, clima ou uma fase que se encontra. Esse trabalho pode ser expandido para outros animais, sendo necessário apenas suas tabelas com as devidas necessidades nutricionais, mas inicialmente é aplicado apenas para os animais relacionados na Tabela 3.

O cronograma mostrado na Tabela 4 indica as etapas concluídas(⊗) e as etapas a serem executadas(◯) no desenvolvimento do sistema de acordo com o discutido na Seção 5. A enumeração a seguir indica a etapas correspondentes ao cronograma:

1. Levantamento bibliográfico e estudo teórico.
2. Coleta de dados.
3. Planejamento de Software.
4. Defesa do Trabalho de Conclusão de Curso I.
5. Desenvolvimento do Software.
6. Aplicação de testes.
7. Implantação.
8. Defesa do Trabalho de Conclusão de Curso II.

Tabela 4 – Cronograma de atividades mostrando as etapas concluídas e as etapas a serem executadas.

Atividade	2013			2014						
	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul
1.	⊗	⊗								
2.			⊗							
3.				⊗	⊗					
4.						⊗				
5.						◯	◯	◯		
6.								◯	◯	
7.										◯
8.										◯

Fonte: Produzido pelo próprio autor

Referências

ANDRIGUETTO, J. M. *Nutrição animal*. São Paulo: Nobel, 1981. Citado 3 vezes nas páginas 23, 24 e 25.

AVESUI: Produção de frango é a que mais deve crescer no Brasil na próxima década. 2012. Avicultura Industrial. Disponível em: <http://www.aviculturaindustrial.com.br/noticia/avesui-producao-de-frango-e-a-que-mais-deve-crescer-no-brasil-na-proxima-decada/20120402164314_G_456>. Acesso em: 05 mar. 2014. Citado na página 21.

CORMEN, T. H. *Algoritmos: teoria e prática*. Rio de Janeiro: Campus, 2002. Citado 5 vezes nas páginas 29, 30, 31, 32 e 33.

CULLISON, A. E. *Feeds and Feeding*. Athens, Georgia: A Prentice Hall Company, 1975. Citado na página 25.

DEITEL, H. M. *Java*: como programar. São Paulo: Pearson Prentice Hall, 2005. Citado na página 35.

GUEDES, G. T. A. *UML*: Uma abordagem prática. São Paulo: Novatec, 2004. Citado na página 39.

JACOUOT, R. et al. *Nutrition Animale*. J.B. Bailliere: Paris, 1958. Citado na página 24.

NRC: Nutrient requirements of beef cattle. 7. ed. Washinton: D.C., 2000. Citado na página 35.

NRC: Nutrient requeriments of dairy cattle. 7. ed. Washinton: D.C., 2001. Citado 2 vezes nas páginas 23 e 35.

NRC: Nutrient requirements of horses. 6. ed. Washinton: D.C., 2007. Citado 2 vezes nas páginas 24 e 35.

ROSTAGNO, H. S. et al. *Tabelas Brasileiras para Aves e Suínos*: Composição de alimentos e exigências nutricionais. 3. ed. Viçosa, MG: [s.n.], 2011. Citado 3 vezes nas páginas 23, 25 e 35.

SAKOMURA, N. K.; ROSTAGNO, H. S. *Métodos de pesquisa em nutrição de monogástricos*. Jaboticabal: Funep, 2007. Citado 3 vezes nas páginas 25, 26 e 27.

SEDGEWICK, R.; WAYNE, K. *Algorithms*. 2013. Disponível em: <http://algs4.cs.princeton.edu/65reductions/>. Acesso em: 23 dez. 2013. Citado na página 35.

ANEXO A – Código Fonte da Classe Simplex

```
1  /*************************************************************************
2   *  Compilation:  javac Simplex.java
3   *  Execution:    java Simplex
4   *
5   *  Given an M-by-N matrix A, an M-length vector b, and an
6   *  N-length vector c, solve the  LP { max cx : Ax <= b, x >= 0 }.
7   *  Assumes that b >= 0 so that x = 0 is a basic feasible solution.
8   *
9   *  Creates an (M+1)-by-(N+M+1) simplex tableaux with the
10  *  RHS in column M+N, the objective function in row M, and
11  *  slack variables in columns M through M+N-1.
12  *
13  *************************************************************************/
14
15 public class Simplex {
16     private static final double EPSILON = 1.0E-10;
17     private double[][] a;   // tableaux
18     private int M;          // number of constraints
19     private int N;          // number of original variables
20
21     private int[] basis;    // basis[i] = basic variable corresponding to
           row i
22                             // only needed to print out solution, not book
23
24     // sets up the simplex tableaux
25     public Simplex(double[][] A, double[] b, double[] c) {
26         M = b.length;
27         N = c.length;
28         a = new double[M+1][N+M+1];
29         for (int i = 0; i < M; i++)
30             for (int j = 0; j < N; j++)
31                 a[i][j] = A[i][j];
32         for (int i = 0; i < M; i++) a[i][N+i] = 1.0;
33         for (int j = 0; j < N; j++) a[M][j]   = c[j];
34         for (int i = 0; i < M; i++) a[i][M+N] = b[i];
35
36         basis = new int[M];
37         for (int i = 0; i < M; i++) basis[i] = N + i;
38
39         solve();
```

```
40
41          // check optimality conditions
42          assert check(A, b, c);
43      }
44
45      // run simplex algorithm starting from initial BFS
46      private void solve() {
47          while (true) {
48
49              // find entering column q
50              int q = bland();
51              if (q == -1) break;  // optimal
52
53              // find leaving row p
54              int p = minRatioRule(q);
55              if (p == -1) throw new ArithmeticException("Linear program is
                    unbounded");
56
57              // pivot
58              pivot(p, q);
59
60              // update basis
61              basis[p] = q;
62          }
63      }
64
65      // lowest index of a non-basic column with a positive cost
66      private int bland() {
67          for (int j = 0; j < M + N; j++)
68              if (a[M][j] > 0) return j;
69          return -1;  // optimal
70      }
71
72     // index of a non-basic column with most positive cost
73      private int dantzig() {
74          int q = 0;
75          for (int j = 1; j < M + N; j++)
76              if (a[M][j] > a[M][q]) q = j;
77
78          if (a[M][q] <= 0) return -1;  // optimal
79          else return q;
80      }
81
82      // find row p using min ratio rule (-1 if no such row)
83      private int minRatioRule(int q) {
84          int p = -1;
85          for (int i = 0; i < M; i++) {
```

```
86              if (a[i][q] <= 0) continue;
87              else if (p == -1) p = i;
88              else if ((a[i][M+N] / a[i][q]) < (a[p][M+N] / a[p][q])) p = i;
89          }
90          return p;
91      }
92
93      // pivot on entry (p, q) using Gauss-Jordan elimination
94      private void pivot(int p, int q) {
95
96          // everything but row p and column q
97          for (int i = 0; i <= M; i++)
98              for (int j = 0; j <= M + N; j++)
99                  if (i != p && j != q) a[i][j] -= a[p][j] * a[i][q] / a[p][q
                        ];
100
101         // zero out column q
102         for (int i = 0; i <= M; i++)
103             if (i != p) a[i][q] = 0.0;
104
105         // scale row p
106         for (int j = 0; j <= M + N; j++)
107             if (j != q) a[p][j] /= a[p][q];
108         a[p][q] = 1.0;
109     }
110
111     // return optimal objective value
112     public double value() {
113         return -a[M][M+N];
114     }
115
116     // return primal solution vector
117     public double[] primal() {
118         double[] x = new double[N];
119         for (int i = 0; i < M; i++)
120             if (basis[i] < N) x[basis[i]] = a[i][M+N];
121         return x;
122     }
123
124     // return dual solution vector
125     public double[] dual() {
126         double[] y = new double[M];
127         for (int i = 0; i < M; i++)
128             y[i] = -a[M][N+i];
129         return y;
130     }
131
```

```
132
133     // is the solution primal feasible?
134     private boolean isPrimalFeasible(double[][] A, double[] b) {
135         double[] x = primal();
136
137         // check that x >= 0
138         for (int j = 0; j < x.length; j++) {
139             if (x[j] < 0.0) {
140                 StdOut.println("x[" + j + "] = " + x[j] + " is negative");
141                 return false;
142             }
143         }
144
145         // check that Ax <= b
146         for (int i = 0; i < M; i++) {
147             double sum = 0.0;
148             for (int j = 0; j < N; j++) {
149                 sum += A[i][j] * x[j];
150             }
151             if (sum > b[i] + EPSILON) {
152                 StdOut.println("not primal feasible");
153                 StdOut.println("b[" + i + "] = " + b[i] + ", sum = " + sum)
                        ;
154                 return false;
155             }
156         }
157         return true;
158     }
159
160     // is the solution dual feasible?
161     private boolean isDualFeasible(double[][] A, double[] c) {
162         double[] y = dual();
163
164         // check that y >= 0
165         for (int i = 0; i < y.length; i++) {
166             if (y[i] < 0.0) {
167                 StdOut.println("y[" + i + "] = " + y[i] + " is negative");
168                 return false;
169             }
170         }
171
172         // check that yA >= c
173         for (int j = 0; j < N; j++) {
174             double sum = 0.0;
175             for (int i = 0; i < M; i++) {
176                 sum += A[i][j] * y[i];
177             }
```

```
178             if (sum < c[j] - EPSILON) {
179                 StdOut.println("not dual feasible");
180                 StdOut.println("c[" + j + "] = " + c[j] + ", sum = " + sum)
                        ;
181                 return false;
182             }
183         }
184         return true;
185     }
186 
187     // check that optimal value = cx = yb
188     private boolean isOptimal(double[] b, double[] c) {
189         double[] x = primal();
190         double[] y = dual();
191         double value = value();
192 
193         // check that value = cx = yb
194         double value1 = 0.0;
195         for (int j = 0; j < x.length; j++)
196             value1 += c[j] * x[j];
197         double value2 = 0.0;
198         for (int i = 0; i < y.length; i++)
199             value2 += y[i] * b[i];
200         if (Math.abs(value - value1) > EPSILON || Math.abs(value - value2)
               > EPSILON) {
201             StdOut.println("value = " + value + ", cx = " + value1 + ", yb
                    = " + value2);
202             return false;
203         }
204 
205         return true;
206     }
207 
208     private boolean check(double[][]A, double[] b, double[] c) {
209         return isPrimalFeasible(A, b) && isDualFeasible(A, c) && isOptimal(
               b, c);
210     }
211 
212     // print tableaux
213     public void show() {
214         StdOut.println("M = " + M);
215         StdOut.println("N = " + N);
216         for (int i = 0; i <= M; i++) {
217             for (int j = 0; j <= M + N; j++) {
218                 StdOut.printf("%7.2f ", a[i][j]);
219             }
220             StdOut.println();
```

```
221         }
222         StdOut.println("value = " + value());
223         for (int i = 0; i < M; i++)
224             if (basis[i] < N) StdOut.println("x_" + basis[i] + " = " + a[i
                ][M+N]);
225         StdOut.println();
226     }
227
228
229     public static void test(double[][] A, double[] b, double[] c) {
230         Simplex lp = new Simplex(A, b, c);
231         StdOut.println("value = " + lp.value());
232         double[] x = lp.primal();
233         for (int i = 0; i < x.length; i++)
234             StdOut.println("x[" + i + "] = " + x[i]);
235         double[] y = lp.dual();
236         for (int j = 0; j < y.length; j++)
237             StdOut.println("y[" + j + "] = " + y[j]);
238     }
239
240     public static void test1() {
241         double[][] A = {
242             { -1,  1,  0 },
243             {  1,  4,  0 },
244             {  2,  1,  0 },
245             {  3, -4,  0 },
246             {  0,  0,  1 },
247         };
248         double[] c = { 1, 1, 1 };
249         double[] b = { 5, 45, 27, 24, 4 };
250         test(A, b, c);
251     }
252
253     // x0 = 12, x1 = 28, opt = 800
254     public static void test2() {
255         double[] c = {  13.0,  23.0 };
256         double[] b = { 480.0, 160.0, 1190.0 };
257         double[][] A = {
258             {  5.0, 15.0 },
259             {  4.0,  4.0 },
260             { 35.0, 20.0 },
261         };
262         test(A, b, c);
263     }
264
265     // unbounded
266     public static void test3() {
```

```
267         double[] c = { 2.0, 3.0, -1.0, -12.0 };
268         double[] b = {  3.0,    2.0 };
269         double[][] A = {
270             { -2.0, -9.0,  1.0,  9.0 },
271             {  1.0,  1.0, -1.0, -2.0 },
272         };
273         test(A, b, c);
274     }
275
276     // degenerate - cycles if you choose most positive objective function
        coefficient
277     public static void test4() {
278         double[] c = { 10.0, -57.0, -9.0, -24.0 };
279         double[] b = {  0.0,   0.0,  1.0 };
280         double[][] A = {
281             { 0.5, -5.5, -2.5, 9.0 },
282             { 0.5, -1.5, -0.5, 1.0 },
283             { 1.0,  0.0,  0.0, 0.0 },
284         };
285         test(A, b, c);
286     }
287
288    public static void test5() {
289         double[] c = { 3., 1., 2.0 };
290         double[] b = {  30.0, 24.0, 36. };
291         double[][] A = {
292             { 1., 1., 3. },
293             { 2., 2., 5. },
294             { 4., 1., 2. }
295         };
296         test(A, b, c);
297     }
298
299
300     // test client
301     public static void main(String[] args) {
302
303        /* try                                { test1();             }
304         catch (ArithmeticException e) { e.printStackTrace(); }
305         StdOut.println("-----------------------------------");
306
307         try                                { test2();             }
308         catch (ArithmeticException e) { e.printStackTrace(); }
309         StdOut.println("-----------------------------------");
310
311         try                                { test3();             }
312         catch (ArithmeticException e) { e.printStackTrace(); }
```

```
313         StdOut.println("-----------------------------------");
314
315         try                                { test4();           }
316         catch (ArithmeticException e) { e.printStackTrace(); }
317         StdOut.println("-----------------------------------");*/
318
319         try                                { test5();           }
320         catch (ArithmeticException e) { e.printStackTrace(); }
321         StdOut.println("-----------------------------------");
322
323
324         int M = Integer.parseInt(args[0]);
325         int N = Integer.parseInt(args[1]);
326         double[] c = new double[N];
327         double[] b = new double[M];
328         double[][] A = new double[M][N];
329         for (int j = 0; j < N; j++)
330             c[j] = StdRandom.uniform(1000);
331         for (int i = 0; i < M; i++)
332             b[i] = StdRandom.uniform(1000);
333         for (int i = 0; i < M; i++)
334             for (int j = 0; j < N; j++)
335                 A[i][j] = StdRandom.uniform(100);
336         Simplex lp = new Simplex(A, b, c);
337         StdOut.println(lp.value());
338     }
339
340 }
```