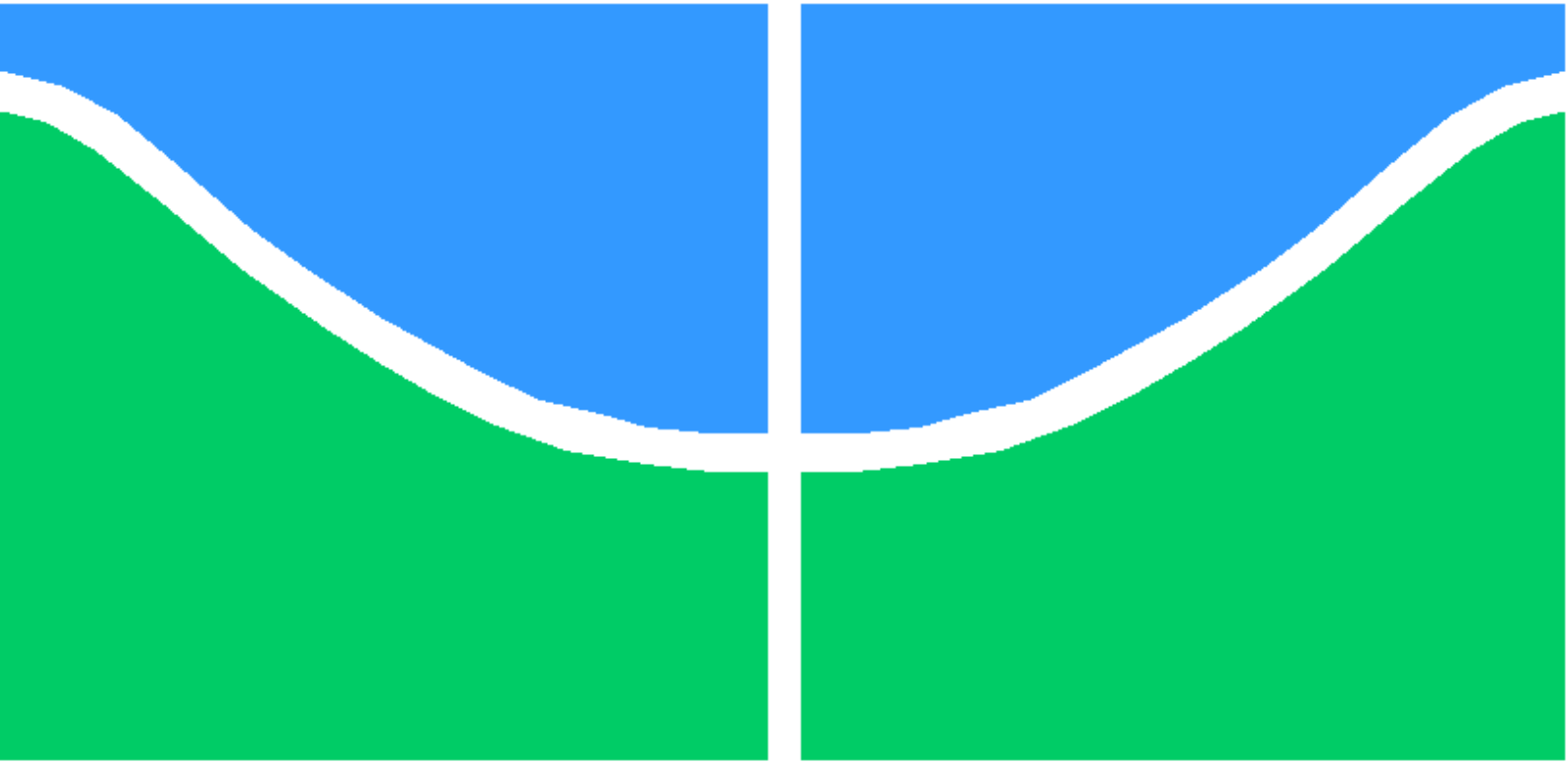


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Algoritmo para criação de rotas de compras econômicas

Autor: Winstein Caldeira Martins
Orientador: Prof. Doutor Nilton Correia da Silva

Brasília, DF
2015

Winstein Caldeira Martins

Algoritmo para criação de rotas de compras econômicas

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Doutor Nilton Correia da Silva

Brasília, DF

2015

Winstein Caldeira Martins
Algoritmo para criação de rotas de compras econômicas/ Winstein Caldeira Martins. – Brasília, DF, 2015-
58 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Doutor Nilton Correia da Silva

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

1. Algoritmo. 2. Rotas. I. Prof. Doutor Nilton Correia da Silva. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Algoritmo para criação de rotas de compras econômicas

CDU 02:141:005.6

Winstein Caldeira Martins

Algoritmo para criação de rotas de compras econômicas

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 11 de dezembro de 2015:

Prof. Doutor Nilton Correia da Silva
Orientador

Prof. Doutor Fabricio Ataides Braz
Convidado 1

Prof. Doutor Luiz Augusto Fontes Laranjeira
Convidado 2

Brasília, DF
2015

Agradecimentos

Agradeço primeiramente aos meus pais pela educação, direcionamento e, principalmente, apoio incondicional em todos os momentos da minha vida. Reconheço também a importância do meu amigo Tallys Martins, que teve participação na idealização do trabalho e que sempre compartilhei pensamentos para tomar as decisões pertinentes à este trabalho.

Sou muito grato ao professor orientador deste trabalho, Nilton Correia, que teve papel muito importante no amadurecimentos das ideias e direcionamento do trabalho. Tenho que agradecer também ao professor Edson Alves, que me apresentou diversos algoritmos e ajudou a construir a solução proposta.

Agradeço à Universidade de Brasília, em especial à Faculdade do Gama, que me proporcionou muito conhecimento durante a graduação, assim como todos os colegas que participaram da minha formação acadêmica de alguma forma.

"Que os vossos esforços desafiem as impossibilidades,
lembrai-vos de que as grandes coisas do homem
foram conquistadas do que parecia impossível."
(Charles Chaplin)

Resumo

Economizar ao realizar compras não é uma tarefa fácil, pois os preços dos estabelecimento comerciais são de difícil acesso. Mas, a partir do momento em que existe o acesso aos preços, é possível saber a maneira em que mais se economiza, realizando compras em locais diferentes. Esse é o problema que este trabalho busca solucionar, realizando comparações e estimando o custo da rota que deverá ser realizada. É proposta uma solução que utiliza algoritmo de Dijkstra e *Backtracking*, e é realizado um estudo sobre custo computacional e monetário para a execução do algoritmo.

Palavras-chaves: Dijkstra. *Backtracking*. Algoritmo.

Abstract

Saving money when shopping is not an easy task, because the products princes in stores is not easy to reach.Butm since there is access to prices, it is possible to find the best way to save money, shopping in different places.This paper looks foward to solve this problem, doing comparasions and estimating the cost of the result route. It is proposed a solution using Dijkstra‘s algorithm and backtracking, and a study of computational and monetary cost to run the algorithm was made.

Key-words: Dijkstra. Backtracking. Algorithm.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
AWS	Amazon *Web Services*
GHz	*Gigahertz*
h	Horas
HTTP	*Hypertext Transfer Protocol*
IOF	Imposto sobre operações financeiras
m	Meses
MVC	*Model-View-Controller*
min	Minutos
s	Segundos
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*

Lista de símbolos

$	Dólar americano
O	*Big* O
Ω	Ômega
R$	Real
Θ	Teta

Sumário

1 Introdução

1.1 Contexto

Todo e qualquer membro de uma sociedade é um consumidor ativo de bens e serviços (BARBOSA; CAMPBELL, 2006). O consumo está presente em toda e qualquer sociedade humana (BARBOSA, 2004).

> "A crença de que comprar em liquidações é algo vantajoso está embutida em nossa cultura de consumo. Tal afirmativa serve até mesmo para validar as justificativas para as compras. É comum utilizar-se de explicações do tipo "comprei porque estava na promoção"." (XAVIER, 2013)

Todo cidadão é um consumidor e todo bom consumidor busca economizar ao máximo, seja evitando compras desnecessárias ou buscando o menor preço. Buscar o menor preço para um determinado produto em lojas físicas é uma tarefa que requer certo esforço, pois é necessário consultar o preço em vários estabelecimentos um a um. Após encontrar o menor preço, o consumidor pode efetuar a compra consciente de que está economizando, ou pelo menos evitando gastar mais dinheiro.

Em junho de 1999 surgiu o BuscaPé, um site para comparação de preços de produtos em lojas virtuais. Através dele é possível saber o preço de determinado produto em várias lojas virtuais gratuitamente, facilitando a busca pelo menor preço por parte dos consumidores, que precisam apenas informar o produto desejado. "[...] o BuscaPé é uma empresa brasileira de comparação de preços e outras informações para a compra de produtos e serviços, com atuação na América Latina [...]" (ARRUDA; ROSSI; PENIDO, 2011).

Em junho de 2000, foi lançado o Bondfaro, uma ferramenta muito semelhante ao Buscapé, que oferece os mesmo serviços também de forma gratuita. Segundo Romero Rodrigues (2011), presidente do BuscaPé em 2011, citado por (ARRUDA; ROSSI; PENIDO, 2011), "Eles **(Bondfaro)**[1] tinham um modelo de negócio idêntico ao nosso e uma estratégia de seguidor bastante inteligente, que consistia em não arriscar nada e desenvolver o que nós tivéssemos desenvolvido e tivesse dado certo". Posteriormente outros comparadores de preços surgiram, como o Zoom, JáCotei e etc. Os comparadores têm em comum a comparação de preços de produtos em lojas virtuais, exclusivamente.

Neste cenário, as ferramentas para comparação de preços estimulam a concorrência entre as lojas, porém exclusivamente entre as lojas virtuais. As lojas físicas estavam

[1] Adaptado pelos autores.

excluídas, até que surgiram ferramentas que fazem comparação de preços de estabelecimentos físicos. O FaciLista e o MeuCarrinho são ferramentas que fazem comparações de produtos de supermercados. Com eles é possível saber o valor de diversos produtos em diversos supermercados, próximos ou não de onde o usuário está localizado. Esses aplicativos atingem uma fatia do mercado não explorada até então, e dão margem para o surgimento de novas ferramentas.

1.2 Problema

Visto que o mercado de comparação de preços de lojas virtuais é bem explorado por diversas ferramentas e que o mercado de comparação de preços de estabelecimentos físicos não é, o problema abordado neste trabalho é a ausência de uma ferramenta capaz de realizar comparações de preços de produtos comercializados por estabelecimentos físicos, de maneira geral, não apenas supermercados.

A existência de ferramentas para comparação de preços unida da vontade de competir nesse meio gera a necessidade de oferecer uma ferramenta com um diferencial competitivo e, por se tratar de um produto de software, o diferencial a ser oferecido será uma inovação tecnológica.

> "Para se desenvolver um software de qualidade é preciso não só atender às normas técnicas e padrões estabelecidos, mas, também, ter a capacidade de incorporar características tecnológicas inovadoras, que não só satisfaçam as necessidades dos usuários, mas que sejam capazes de antevê-las. Um casamento harmonioso entre qualidade e inovação pode garantir a permanência de um produto competitivo no mercado." (FREIRE, 2002)

1.3 Objetivos

1.3.1 Objetivo geral

O objetivo geral deste trabalho de conclusão de curso é desenvolver o diferencial tecnológico citado na seção anterior, 1.2. Durante a execução do trabalho será desenvolvida uma tecnologia capaz de decidir qual é a melhor opcão de compra, do ponto de vista financeiro, para o usuário, ou seja, após o usuário informar os produtos de interesse e sua localização, o sistema calculará a combinação de compra que o usuário terá a maior economia, podendo realizar compras em mais de um estabelecimento e levando em consideração o custo de deslocamento do usuário.

Uma combinação de compra, que é a saída esperada, conterá os estabelecimentos e produtos que devem ser adquiridos em cada local, além de uma rota a ser percorrida para realizar as compras.

1.3.2 Objetivos específicos

Buscando alcançar e satisfazer o objetivo geral, foram definidos seis objetivos específicos, descritos a seguir:

- Caracterizar o problema
- Realizar estudo sobre técnicas e algoritmos capazes de solucionar o problema
- Implementar a solução
- Analisar a complexidade da solução implementada
- Realizar um estudo sobre o tempo de execução da solução em ambientes diversos
- Analisar os resultados do estudo

1.4 Justificativa

1.4.1 Importância do diferencial competitivo

Para uma pequena empresa entrar em um mercado com grandes concorrentes e obter sucesso, é necessário investir em diferenciais competitivos. Se a empresa atuar na área de desenvolvimento de software, o investimento em inovação tecnológica produz novidades que possuem valor e atraem usuários, provendo o sucesso da empresa. “O sucesso competitivo exige incessantes esforços inovativos, conferindo a essas atividades intenso dinamismo tecnológico.” (ROSELINO, 2006), ou seja, para alcançar o sucesso é necessário investir em inovação tecnológica.

Segundo Almeida e Frick, citado por (FREIRE, 2002), “Em países em desenvolvimento, como o Brasil, com empresas pequenas, sem poder de oligopólio e descapitalizadas, as atividades inovadoras são o principal fator de competitividade.”

Mesmo após obter sucesso, a inovação ainda é muito importante para manter e aumentar o sucesso. Muitas empresas que já obtiveram sucesso com o lançamento de um produto falham quando tentam competir em novos mercados ou tecnologias (BALACHANDRA; FRIAR, 1999). O que já foi novidade um dia pode se tornar obsoleto, ou seja, o investimento em inovações não deve parar.

Através da inovação é possível o acesso a novos mercados, o aumento dos lucros, a criação de empregos, fortalecimento da marca (BRANDÃO, 2006). A inovação é importante como fator sobrevivência e crescimento de uma organização (RAMACHANDRAN, 2012).

1.4.2 Motivação do autor

Tornei-me um entusiasta pela área de estudo e implementações de algoritmos, durante cursar uma disciplina[2] ofertada pela Faculdade do Gama. Apesar de não ser exatamente a área afim deste trabalho, para a resolução da problemática é necessária a implementação de algoritmos estudados na disciplina.

A complexidade do problema também é bastante motivadora, pois a solução não é evidente, trivial, e pode ser construída de diversas maneiras. Além disso, não é um exercício, preparado para uma maratona, é um problema real cuja solução pode trazer diversos benefícios aos usuários.

1.5 Método

O trabalho foi dividido e executado em etapas sequenciais, que buscam satisfazer os objetivos descritos incrementalmente.

1.5.1 Caracterização do problema

Esta etapa buscou descrever e delimitar o problema de forma clara, para que houvesse apenas uma interpretação. Todas as características foram levantadas e todas considerações foram feitas. Apesar de simples essa etapa tem grande importância, pois direcionou todos os esforços para a resolução da problemática exposta. Delimitou o escopo do trabalho, que é desenvolver uma solução.

1.5.2 Revisão bibliográfica

Foi realizado um estudo na literatura de como criar uma solução para o problema em questão. Vários algoritmos foram analisados, muitos deles até implementados, para facilitar a compreensão. Por não ser um problema simples, essa etapa demandou um grande esforço e boa parte do tempo disponível para execução do trabalho.

[2] Tópicos especiais em programação (Disciplina ministrada como preparação para maratonas de programação)

1.5.3 Construção da solução

Após identificar algoritmos e métodos potencialmente úteis, foi necessário criar de fato uma solução, que atendesse às restrições descritas. Após a implementação, foram criados cenários de teste, a fim de validar a solução proposta e verificar que todos os requisitos funcionais foram cumpridos.

1.5.4 Análise da solução

Validada a solução, foi necessário estudá-la para compreender suas necessidades computacionais, complexidade, tempo necessário para execução e limitações. Com o cálculo da complexidade ciclomática, surgiu a necessidade de calcular o tempo de execução do algoritmo em diferentes máquinas. Como não existia a possibilidade de executar o algoritmo nas máquinas, foi encontrado um meio teórico de criação da estimativa desejada.

1.5.5 Implementação do protótipo

Com a solução criada e analisada, o próximo passo é incorporá-la num sistema Web, integrado a softwares reais de mapeamento geográfico, para simular situações reais de uso e melhorar a visualização do algoritmo. Foram selecionadas tecnologias para a construção do protótipo funcional, algumas delas foram escolhidas por afinidade do autor, visto que o tempo para implementação era curto.

Essa é a etapa final, que consolida todo o trabalho.

1.6 Estrutura do trabalho

Este trabalho está organizado em cinco capítulos. Introdução para apresentar o problema e sua importância. Referencial teórico que contém informações sobre os conceitos utilizados para o desenvolvimento do trabalho. Problemática e solução que descreve o problema de maneira mais clara e detalhado, além de mostra como a solução foi construída. Resultados que apresenta informações sobre o algoritmo proposto. Conclusão que contém uma discussão sobre os resultados e sugestões de trabalhos futuros.

2 Referencial Teórico

2.1 Análise de complexidade de algoritmos

Ao projetar um algoritmo é necessário se preocupar com o desempenho do mesmo. Existem muitas maneiras de se resolver um problema, porém o desempenho das soluções deve ser levado em consideração para a escolha da solução que será implementada.

Para estudar a eficiência de algoritmos é pouco lógico observar recursos utilizados em computadores específicos, pois com os diferentes conjuntos de instruções, desempenho e arquitetura de processadores, é difícil comparar as medidas. A análise assintótica exprime o tempo de execução de algoritmos de maneira genérica, independente da linguagem de programação, particularidades de implementação ou computador utilizado através de funções matemáticas.

A análise leva em conta apenas o "fator de crescimento" das funções, desprezando valores pequenos do argumento, fatores multiplicativos e outros valores. A função n cresce na mesma velocidade das funções $2n$ e $10n$, por exemplo. Está disponível em (CORMEN et al., 2001) e (HOROWITZ; SAHNI, 1978) as ordens de complexidade mais utilizadas, assim como suas definições e aplicações; três delas são apresentadas a seguir.

2.1.1 Notação O

A notação *Big* O descreve o crescimento do número de operações realizadas de acordo com o aumento de elementos processados, de um algoritmo. Dadas duas funções não negativas, f e g, diz-se que f está na ordem O de g, representado por $O(g)$, se $f(n) \leq c.g(n)$, para c positivo e para grandes valores de n. Em outras palavras, as funções $f(n)$ e $g(n)$ tem o mesmo "fator de crescimento".

> "Quando diz-se que um algoritmo tem tempo de processamento $O(g(n))$, significa que o algoritmo executando o mesmo tipo de dado para valores crescentes de n, terá o tempo resultante sempre menor que algum tempo $c\ |g(n)|$. Quando se determina a ordem de complexidade de $f(n)$, tenta-se, sempre, obter o menor $g(n)$ de modo que $f(n) = O(g(n))$." (LORETO, 2000)

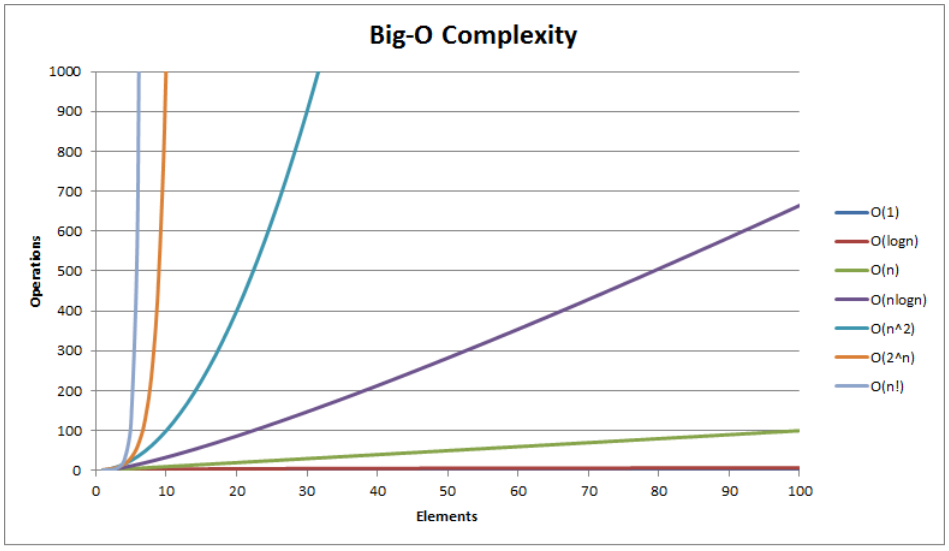


Figura 1: Representação gráfica do crescimento de funções. (BIG-O CHEAT SHEET, 2012)

2.1.2 Notação ômega

A notação ômega (Ω) descreve um limite inferior para o tempo de execução de certo algoritmo. Dadas duas funções não negativas f e g, diz-se que f está em ordem ômega de g, representado por $f = \Omega(g)$, se $f(n) \geq c \ . \ g(n)$, para c positivo e para grandes valores de n. Pode-se dizer que para executar $f(n)$ é necessário no mínimo $g(n)$.

> "A notação ômega (Ω), que descreve um limite assintótico inferior sobre $f(n)$, é usada para estimar o tempo de processamento de um algoritmo. Em alguns casos, o tempo de execução de um algoritmo, $f(n)$, é tal que $f(n) = \Omega(g(n))$ e $f(n) = \mathrm{O}(g(n))$. Para essas circunstâncias usa-se a notação Θ (Teta)."[1] (LORETO, 2000)

2.1.3 Notação teta

A notação *Big* O tem o conceito de "$f \leq g$" e a notação ômega tem o conceito de "$f \geq g$". Já a notação teta tem o conceito de "$f = g$". Dadas duas funções, f e g, diz-se que f e g estão na mesma ordem teta, representado por $f = \Theta(g)$, se $f = \mathrm{O}(g)$ e $f = \Omega(g)$. As funções n^2, $1000n^2$ e $n^2 + 50n$ pertencem a ordem $\Theta(n^2)$, por exemplo.

Ao usar a notação teta, é possível afirmar que tem-se um limite assintótico estreito para o tempo de execução do algoritmo. É um limite estreito porque o tempo de execução é limitado por fator acima e abaixo. "Se $f(n) = \Theta(g(n))$ então $f(n)$ é limitado superiormente e inferiormente por $g(n)$" (LORETO, 2000).

[1] Adaptado pelo autor.

2.2 Grafos

Existe na matemática um ramo que estuda as relações entre elementos de um determinado conjunto que faz o uso de estruturas denominadas grafos, G(V,A), onde V é a representação de um conjunto não vazio dos vértices de um grafo G, e A é um conjunto de pares de arestas deste grafo.

"Grafos são representações abstratas que descrevem organização de sistemas de transporte, circuitos elétricos, interações humanas e redes de telecomunicações". [2] (SKIENA; REVILLA, 2003)

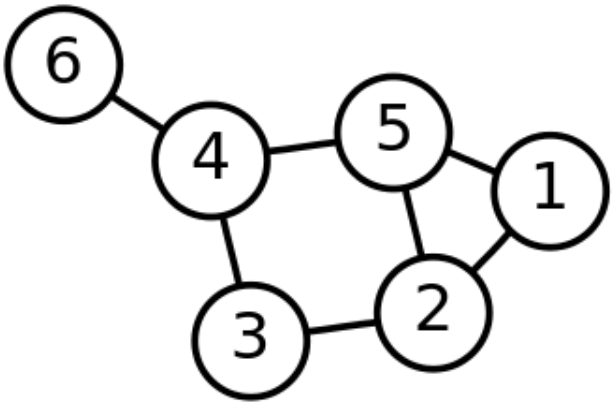


Figura 2: Representação de um grafo. (Giga Mundo, 2009)

2.2.1 Definições

A representação em grafo de cada aplicação varia de acordo com o contexto. As arestas podem ou não ter direção, pode ser que um vértice tenha uma aresta ligando-o a si próprio (*loop*), ou ainda que as arestas tenham um peso ou custo associado. Podemos também classificar um grafo em direcionado, quando as arestas tem um sentido - representado por uma seta - e não direcionado, quando o caminho das arestas pode ser feito nos dois sentidos.

Desta forma, quando uma aresta conecta dois vértices, esses são denominados incidentes à aresta e a quantidade de arestas ligadas a um vértice é denominada valência, onde os *loops* são contados duas vezes. No grafo de exemplo da figura 2 os vértices 1 e 3 possuem uma valência de 2, os vértices 2, 4 e 5 têm a valência de 3 e o vértice 6 tem a valência de 1.

No contexto dos grafos temos também um termo chamado adjacência. Dois vértices são considerados adjacentes se existe uma aresta que liga os dois. Novamente, no grafo da imagem 2 os vértices 1 e 2 são adjacentes, mas os vértices 2 e 4 não são. Assim, o conjunto de vértices ligados a um determinado ponto são os vizinhos deste ponto.

[2] Tradução livre feita pelo autor.

2.2.2 Representação

A representação gráfica mostra os vértices representados por círculos e as arestas como linhas conectando os vértices como pode ser visto na imagem 2. No entanto, esta representação é bem diferente da estrutura abstrata utilizada na computação e matemática. Por exemplo, um grafo simples pode ser representado matematicamente como um conjunto de vértices V = { 1, 2, 3, 4, 5, 6, } e um conjunto de arestas E = { {1,2}, {1,5}, {2,3}, {2,5}, {3,4}, {4,5}, {4,6} }.

Na computação, um grafo também pode ser representado em diferentes formas, cada uma com diferentes abordagens para satisfazer algoritmos e problemas específicos.

2.2.2.1 Lista de arestas

Uma maneira simples de se representar um grafo é com uma lista de a arestas. Para representar uma aresta utiliza-se um par de valores indicando os vértices de origem e destino. Caso a aresta tenha um peso, basta adicionar um terceiro elemento indicando-o. Como cada aresta possui apenas dois ou três números, o espaço total para uma lista de arestas é em função do número de arestas |E|.

A = [[1,2], [1,5], [2,3], [2,5], [3,4], [4,5], [4,6]] [3]

Listas de arestas são simples, mas se quisermos descobrir se há uma ligação entre dois vértices em particular, precisamos procurá-la na lista de arestas. Se as arestas aparecerem na lista sem uma ordem em particular, a questão torna-se uma busca linear por uma lista de E arestas. Mas para um caso de uma lista de arestas ordenadas podem ser utilizadas técnicas para deixar a busca com complexidade O(log E).

2.2.2.2 Matriz de adjacências

Esta é uma das formas mais comuns e a mais completa de se representar um grafo. Dado um grafo G com n vértices, podemos representá-lo em uma matriz A_{n*n}. Os elementos de A dependem das propriedades do grafo que se deseja representar e guardam basicamente as informações de adjacência entre os vértices. No caso de um grafo não direcionado, simples e sem pesos nas arestas, então a matriz guarda apenas o valor 1 caso o haja adjacência entre os vértices V_i e V_j e 0 caso não, o que resulta em uma matriz simétrica ao longo da diagonal principal.

[3] Lista de arestas do grafo da imagem 2.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 3 & 0 & 0 & 0 & 0 & 3 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 5 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 & 3 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 & 0 & 4 & 0 & 0 \end{bmatrix}$$

Para grafos com pesos e as arestas guardam os valores dos pesos e para grafos direcionados a matriz perde a simetria conforme a direção das arestas.

Com uma matriz de adjacência, podemos descobrir se há ligação entre dois vértices em tempo constante O(1). No entanto, ela é uma representação que ocupa muito recurso computacional, pois guarda as informações de todas as combinações O(n^2) o que é ruim para grafos esparsos, com poucas arestas. E também, caso o objetivo seja descobrir quais vértices são adjacentes a um vértice V_i é necessário percorrer todos os elementos da lista i para verificar seus valores.

2.2.2.3 Lista de adjacências

Esta representação combina a matriz de adjacências com a lista de arestas. Para cada vértice V_i do grafo, é armazenada uma lista com os vértices adjacentes a i. Para adicionar o peso da aresta, basta adicionar um par de elementos para cada item da lista.

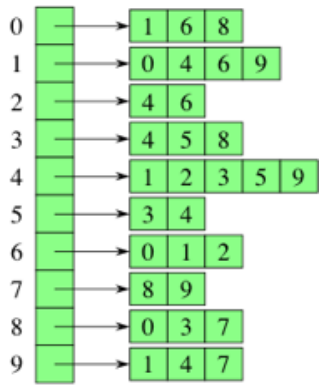


Figura 3: Representação de uma lista de adjacências. (KHAN ACADEMY, 2015)

Desta forma, para saber a ligação entre dois vértices, basta acessar a lista de adjacências do vértice i desejado e percorrer a lista de forma que, no pior caso, o tempo

para processar esta busca é O(d), onde d é a valência[4] do vértice i.

2.3 Algoritmo de Dijkstra

2.3.1 Problema do caminho mínimo

O problema do caminho mínimo é um dos clássicos da ciência da computação pois está associado a várias aplicações práticas, como roteamento, fluxo em redes e telecomunicações. O problema consiste na travessia de um grafo entre dois vértices, buscando minimizar o custo, que é dado pela soma dos pesos de cada aresta percorrida.

> "O problema do caminho mínimo é um dos problemas fundamentais de otimização em grafos. [...] Algoritmos para esse problema estão sendo estudados há muito tempo, entretanto ainda são feitos avanços em algoritmos na teoria do problema do caminho mínimo."[5] (CHERKASSKY; GOLDBERG; RADZIK, 1996)

Sendo u e v dois vértices de um grafo $G = (V, A)$, o menor caminho entre u e v é uma sequência de arestas que unem u a v acumulando o menor comprimento, passando por vértices distintos. É necessário que haja uma conexão entre os vértices u e v, para que exista um caminho mínimo, ou seja, v deve ser sucessor de u (GOLDBARG; LUNA, 2005). "Na maioria dos casos, existe mais de um caminho entre dois nós específicos; com isso, o problema do caminho mínimo consiste em encontrar o caminho com menor custo dentre todos os possíveis." (HERNANDES; BERTON; CASTANHO, 2009)

2.3.2 Definição

Em 1959, o holandês Edsger Dijkstra anunciou um algoritmo para resolver um problema conhecido como *single-source shortest path* (problema do caminho mínimo), cujo objetivo era encontrar o menor caminho entre dois vértices de um grafo ponderado (ZHANG et al., 2014). Esse algoritmo é capaz de encontrar, por exemplo, a menor distância para uma viagem que atravessa várias cidades com diferentes caminhos para se chegar a um destino. Dado um vértice de início S, ele encontra o menor caminho a partir de S para todos os outros vértices no grafo, inclusive o destino T (SKIENA; REVILLA, 2003).

Se as arestas são positivas, o algoritmo de Dijkstra tem bom desempenho. Uma implementação do algoritmo é executada num tempo O($m + n\ log\ n$), onde n e m são número de vértices e arestas, respectivamente (CHERKASSKY; GOLDBERG; RADZIK, 1996).

[4] Valência: número de vértices ligados àquele vértice em questão.
[5] Tradução livre feita pelo autor.

2.3.3 Funcionamento

Segundo (DENG et al., 2012), para se obter o caminho mínimo de um vértice pertencente à um grafo utilizando o algoritmo de Dijkstra, deve-se executar os passos descritos a seguir. Assumindo que o vértice no início do caminho é o vértice de origem e que a distância de um vértice qualquer X será a distância do vértice de origem ao vértice X. O algoritmo atribuirá distância iniciais e tentará aprimorar a cada passo.

1. Atribuir um valor de distância a todos os vértices: zero para o vértice de origem e infinito para todos os outros vértices;
2. Marcar todos os vértices como não visitados. Estabelecer o vértice inicial (origem) como atual.
3. Para o vértice atual, considerar todos os vértices adjacentes e calcular uma tentativa de distância para cada um deles. Por exemplo, se o vértice atual A tem a distância de 6, e uma aresta o conecta a outro vértice B tem distância 2, a distância para B passando por A será $6 + 2 = 8$. Se essa distância é menor que a distância registrada anteriormente, sobrescreva a distância;
4. Após calcular a distância para todos os vértices adjacentes ao vértice atual, marque-os como visitados. Um vértice que já foi visitado não será checado novamente; a distância registrada já é a mínima;
5. Se todos os vértices já foram visitados, parar. Caso contrário, estabelecer o vértice com menor distância para vértice inicial como vértice atual e retroceder ao passo 3.

A figura 4 ilustra a execução do algoritmo de Dijkstra. Os pesos das arestas estão indicados próximos às arestas e o valor da distância no momento da iteração está dentro do vértice. 99 representa infinito. Os vértices marcados como visitados estão grifados com cor verde. (SIBEYN, 2002)

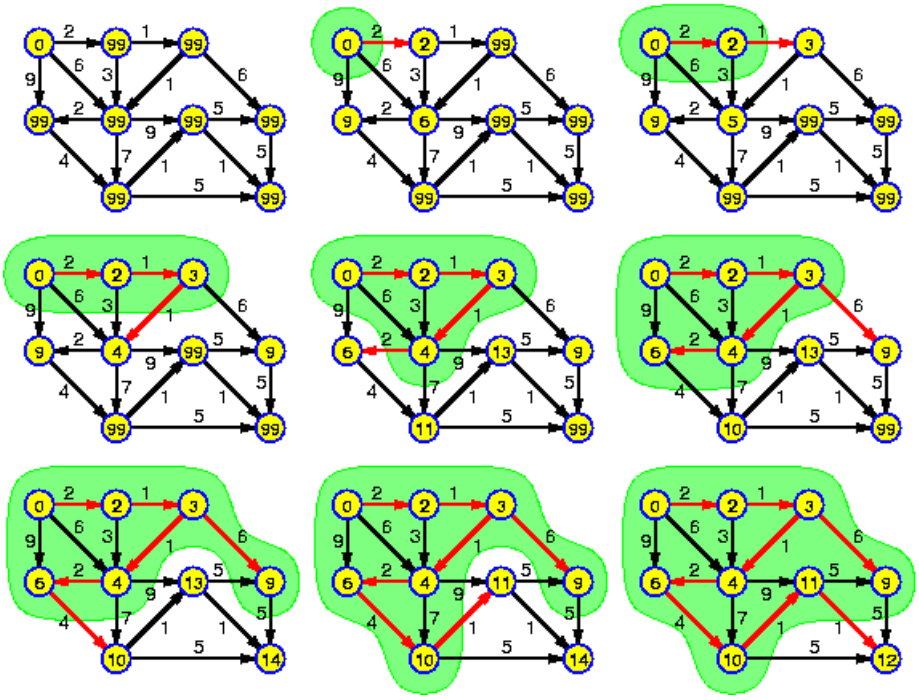


Figura 4: Execução do algoritmo de Dijkstra (SIBEYN, 2002). Adaptado.

Em (CORMEN et al., 2001) encontram-se mais informações sobre o algoritmo.

2.4 *Backtracking*

Em muitos problemas do mundo real uma solução pode ser encontrada por uma busca exaustiva através de uma quantidade numerosa, porém limitada, de possíveis soluções. Entretanto, é necessário desenvolver uma técnica de busca sistemática, que pode diminuir o espaço de busca de possibilidades em um muito menor. *Backtracking* é um método de busca exaustiva organizado, que, na maioria dos casos, pode evitar a busca em todas as possíveis soluções. Geralmente é utilizado para resolver problemas onde um grande número de soluções finitas devem ser inspecionadas (ALSUWAIYEL, 2003).

Backtracking é um método sistemático para iterar todas as possíveis configurações em um espaço de busca. É uma técnica/algoritmo genérico que deve ser personalizado para cada cenário de aplicação (SKIENA; REVILLA, 2003). O algoritmo é apropriado para resolver problemas de otimização combinatória de larga escala (DAI; LIU, 2013).

Se o algoritmo de *Backtracking* é utilizado para encontrar todas as soluções de um problema, ou uma de suas soluções ótimas, ele não encerrará enquanto não retroceder para a raiz da árvore do espaço de soluções e, simultaneamente, explorar todas as sub árvores dos nó raiz. Quando o algoritmo é utilizado para encontrar qualquer solução possível, ele encerrará assim que encontrar uma solução para o problema (WANG, 2001).

2.4.1 Funcionamento

Nesta seção, o algoritmo genérico é descrito como um método de busca sistemático que pode ser aplicado a classe de problemas cuja solução consistem em um vetor (x_1, x_2, ..., x_i), satisfazendo algumas restrições. i é um inteiro entre 0 e n, onde n é uma constante que depende da formulação do problema (DAI; LIU, 2013).

Cada x_i no vetor de soluções pertence a um conjunto finito linearmente ordenado. O algoritmo é iniciado com o vetor de soluções vazio e, então, escolhe o menor elemento de X_1, chamado x_1. Se (x_1) é uma solução parcial, o algoritmo procede e escolhe o elemento mínimo de X_2, chamada x_2. Se (x_1, x_2) é uma solução parcial, o menor elemento de X_3 é incluído; caso contrário x_2 passa a ser o próximo elemento de X_2. Suponde que o algoritmo explorou a solução parcial (x_1, x_2, ..., x_j). Ele, então, investigará o vetor v = (x_1, x_2, ..., x_j, x_{j+1}). Existem os três casos a seguir (DAI; LIU, 2013):

1. Se v representa uma solução final para o problema, o algoritmo o armazena como uma solução e então se encerra, caso seja necessário apenas uma solução, ou continua para explorar outras soluções;

2. Se v representa uma solução parcial, o algoritmo avança e escolhe o elemento mínimo do conjunto X_{j+2};

3. Se v não é uma solução final e nem parcial, existem dois "sub casos":

    a) Se existem outros elementos para escolher no conjunto X_{j+1}, o algoritmo estabelece x_{j+1} como o próximo elemento de X_{j+1};

    b) Se não existem elementos para escolher no conjunto X_{j+1}, o algoritmo retrocede estabelecendo x_j como o próximo elemento de X_j. Se novamente não existem elementos no conjunto X_j, o algoritmo retrocede estabelecendo x_{j-1} como o próximo membro de X_{j-1}, e assim por diante.

Em (SKIENA; REVILLA, 2003) e (WANG, 2001) encontram-se mais informações sobre o algoritmo.

3 Problemática e solução

3.1 Caracterização do problema

Este trabalho busca resolver o problema de roteirização de compras de produtos em diversos estabelecimentos comerciais. Existem várias lojas e um usuário, interessado em adquirir produtos e economizar ao máximo percorrendo estabelecimentos e realizando compras, levando em consideração o custo de locomoção.

No protótipo implementado para resolver o problema, a localização e os estabelecimentos são pontos, representados por suas coordenadas geográficas (latitude e longitude), em um mapa. As coordenadas possibilitam o cálculo de distância linear entre duas lojas. Utilizando um sistema auxiliar de mapeamento geográfico[1], que fornece as distâncias entre dois pontos considerando as vias públicas, é possível obter um valor de distância que será efetivamento percorrido pelo usuário.

Existem vários estabelecimentos com seus produtos e preços. Para construção do protótipo, é considerado que todos os estabelecimentos comercializam todos os produtos. Por não ter acesso à uma lista de produtos, foi criada uma lista para todos os estabelecimentos, com variação apenas do preço.

Um exemplo é apresentado a seguir, com base na tabela 1.

Produto / Estabelecimento	**Estabelecimento A**	**Estabelecimento B**	**Estabelecimento C**
Produto 1	R$ 1,50	R$ 2,00	R$ 2,50
Produto 2	R$ 10,20	R$ 8,50	R$ 11,50
Produto 3	R$ 5,20	R$ 5,50	R$ 5,00
Produto 4	R$ 30,40	R$ 28,50	R$ 27,00
Produto 5	R$ 40, 00	R$ 40,00	R$ 40,00
Produto 6	R$ 70, 80	R$ 70,00	R$ 65,90
Total	R$ 158,10	R$ 154,50	R$ 151,90

Tabela 1: Exemplo de situação de compra. Fonte: Autor.

Partindo do pré suposto que todos os produtos devem ser adquiridos e analisando apenas os preços nos estabelecimentos, sem considerar o custo de locomoção, a melhor opção é adquirir os produtos de menor preço em cada estabelecimento. Assim, o produto 1 seria comprado no estabelecimento A, o produto 2 no estabelecimento B, e o restante no estabelecimento C.

[1] *Google Maps* ou *Open Street Map*, por exemplo.

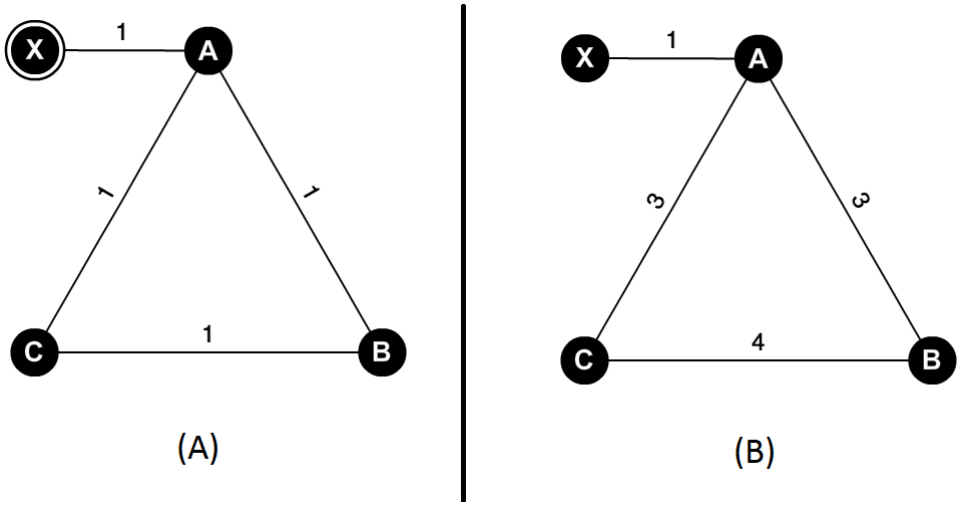


Figura 5: Grafos que representam a localização e distâncias do usuário (vértice X) e estabelecimentos A, B e C (vértices A, B e C, respectivamente). Fonte: Autor.

Levando em consideração o grafo A da figura 5, é necessário calcular o valor para percorrer o caminho que levará a maior economia possível. Neste cenário, a melhor opção seria passar por todos os estabelecimentos, adquirindo o produto 1 no estabelecimento A, o produto 2 no estabelecimento B e o restante dos produtos no estabelecimento C. Seria gasto R$ 147,90 em produtos e R$ 5,00 em locomoção, totalizando R$ 152,90.

Levando em consideração o grafo B da figura 5, a melhor opção seria comprar todos os produtos em A, gastando R$ 158,10 em produtos e R$ 2,00 em locomoção, totalizando R$ 160,10. Se a compra for realizada em mais de um local seria possível economizar no valor gasto com produtos, porém o valor de locomoção faz com que a opção não seja a melhor.

3.2 Solução adotada

Para a construção da solução para o problema descrito na seção anterior, foram utilizados os dois algoritmos descritos no capítulo 2, Dijkstra e *backtracking*. Por se tratar de um algoritmo de busca de possibilidades, *backtracking* foi utilizado para criar todas as possíveis configurações de compra, para assim, determinar qual é a melhor. Após construir as combinações de compra, é necessário calcular o custo da rota a ser percorrida. Para esse cálculo foi utilizado o algoritmo de Dijkstra, que calcula a menor distância de um vértice à todos os outros de um grafo.

Tendo como insumo os estabelecimentos envolvidos e suas listas de preços, um grafo que tem como vértices os estabelecimentos e a localização do usuário, a lista de produtos desejados, a solução cria um combinação de compra, calcula o valor da rota, e armazena a melhor combinação, executando os seguintes passos:

1. Estabelecer melhor opção com valor infinito.
2. Criar uma combinação de compra válida.
3. Identificar estabelecimentos envolvidos na combinação de compra.
4. Cacular o custo da menor rota que passar por todos os estabelecimentos envolvidos e retorna para o local de origem (localização do usuário).
5. Armazenar a combinação de compra caso seu custo seja menor do que a total já armazenado.
6. Retornar para o item 2 até que de acabem as possibilidades de compra.

O código fonte do algoritmo criado pode ser consultado no Anexo A.

Após executados os passos, a combinação de menor valor será encontrada. A tabela 2 representa de maneira mais clara, a resolução do problema exemplo citado na seção 3.1.

Nº	**Produto 1**	**Produto 2**	**Produto 3**	**Produto 4**	**Produto 5**	**Produto 6**
1	A	A	A	A	A	A
2	A	A	A	A	A	B
3	A	A	A	A	A	C
4	A	A	A	A	B	A
5	A	A	A	A	B	B
6	A	A	A	A	B	C
7	A	A	A	A	C	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮
300	B	A	C	A	A	C
301	B	A	C	A	B	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮
729	C	C	C	C	C	C

Tabela 2: Todas as possibilidades de compra para o problema apresentado na seção 3.1. As letras A, B e C representam estabelecimentos. Fonte: Autor.

A cada combinação de compra criada é necessário identificar a quantidade de estabelecimento envolvidos e calcular a menor rota que se inicia e termina na localização do usuário, passando pelos estabelecimentos identificados. São criadas E^P combinações, E é o número de estabelecimentos e P o número de produtos. No exemplo descrito acima são criadas $3^6 = 729$ combinações.

Por criar todas as possíveis soluções, o algoritmo tem um desempenho considerado ruim, pois possui complexidade alta. O comportamento do algoritmo é representado pela função $E + PE^{P+1}logE$, tendo, assim, a complexidade O($PE^{P+1}logE$). E e P representam a quantidade de estabelecimentos e produtos, respectivamente. A tabela 3 apresenta uma

simulação do crescimento de possibilidades com o aumento do número de estabelecimentos e produtos.

	Nº de produtos				
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**
1	1	1	1	1	1
2	2	32	1024	$3,28.10^4$	$1,05.10^6$
3	3	243	$5,9.10^4$	$1,43.10^7$	$3,49.10^9$
4	4	1024	$1,05.10^6$	$1,07.10^9$	$1,1.10^{12}$
5	5	3125	$9,77.10^6$	$3,05.10^{10}$	$9,54.10^{13}$

Tabela 3: Quantidade de possíveis soluções. Fonte: Autor.

Utilizando a função que descreve o comportamento do algoritmo, é possível calcular a quantidade de operações que serão realizadas durante a execução, para determinados números de estabelecimentos e produtos.

	Nº de produtos					
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**	**25**
1	1	1	1	1	1	1
2	4	99	6168	$2,95.10^5$	$1,26.10^7$	$5,05.10^8$
3	8	1743	$8,45.10^4$	$3,08.10^8$	$9,98.10^{10}$	$3,03.10^{13}$
4	14	12335	$2,52.10^7$	$3,87.10^{10}$	$5,29.10^{13}$	$6,77.10^{16}$
5	23	54613	$3,41.10^8$	$1,59.10^{12}$	$6,66.10^{19}$	$2,6.10^{19}$

Tabela 4: Quantidade de operações realizadas durante a execução do algoritmo. Fonte: Autor.

4 Resultados

4.1 Tempo de execução da solução

Amazon *Web Services*[1] é um serviço de computação em nuvem disponibilizado pela Amazon.com[2]. São oferecidos diversos serviços e a locação de servidores de aplicação é um deles. A tabela 5 apresenta uma breve descrição de pequena parte das máquinas disponíveis.

Servidores Amazon *Web Services* **Tipo de instância**	**Núcleos**	**Processador**	**Velocidade de *clock* (GHz)**
t2.micro	1	Família Intel Xeon	Até 3.3
t2.medium	2	Família Intel Xeon	Até 3.3
m4.xlarge	4	Intel Xeon E5-2676 v3	2.4
m4.2xlarge	8	Intel Xeon E5-2676 v3	2.4
m4.4xlarge	16	Intel Xeon E5-2676 v3	2.4
m4.10xlarge	40	Intel Xeon E5-2676 v3	2.4

Tabela 5: Descrição dos servidores disponibilizados por AWS. Fonte: (Amazon Web Services, 2015). Adaptado.

Levando em consideração que um processador de 1GHz pode executar um bilhão de instruções por segundo, é possível estimar o tempo de execução do algoritmo nos servidores da Amazon. São apresentados, nas tabelas a seguir, os tempos de resposta do algoritmo nas diferentes instâncias dos servidores da Amazon.

Instância: t2.micro			**Núcleos: 1**		***Clock*: 3.3**	
	Nº de produtos					
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**	**25**
1	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s
2	0,000 s	0,000 s	0,000 s	0,000 s	0,003 s	0,153 s
3	0,000 s	0,000 s	0,000 s	0,093 s	30,242 s	2 h 33 min
4	0,000 s	0,000 s	0,007 s	11,727 s	4 h 26 min	7 m 27 dias
5	0,000 s	0,000 s	0,103 s	8 min	23 dias	254 anos

Tabela 6: Tempo necessário para execução do algoritmo na máquina t2.micro. Fonte: Autor.

[1] Disponível em: <https://aws.amazon.com/pt/>.
[2] Disponível em: <http://www.amazon.com/>.

Instância: t2.medium			**Núcleos: 2**		***Clock*: 3.3**	
	Nº de produtos					
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**	**25**
1	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s
2	0,000 s	0,000 s	0,000 s	0,000 s	0,001 s	0,077 s
3	0,000 s	0,000 s	0,000 s	0,093 s	15,121 s	1 h 17 min
4	0,000 s	0,000 s	0,003 s	5,864 s	2 h 13min	3 m 28 dias
5	0,000 s	0,000 s	0,052 s	4 min	11 dias 12h	127 anos

Tabela 7: Tempo necessário para execução do algoritmo na máquina t2.medium. Fonte: Autor.

Instância: m4.xlarge			**Núcleos: 4**		***Clock*: 2.4**	
	Nº de produtos					
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**	**25**
1	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s
2	0,000 s	0,000 s	0,000 s	0,000 s	0,001 s	0,053 s
3	0,000 s	0,000 s	0,000 s	0,032 s	10,396 s	53 min
4	0,000 s	0,000 s	0,002 s	4,031 s	1h 32 min	2 m 22 dias
5	0,000 s	0,000 s	0,036 s	2 min 45 s	8 dias 7h	87 anos

Tabela 8: Tempo necessário para execução do algoritmo na máquina m4.xlarge. Fonte: Autor.

Instância: m4.2xlarge			**Núcleos: 8**		***Clock*: 2.4**	
	Nº de produtos					
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**	**25**
1	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s
2	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,027 s
3	0,000 s	0,000 s	0,000 s	0,016 s	5,192 s	27 min
4	0,000 s	0,000 s	0,001 s	2,015 s	46 min	1 m 11 dias
5	0,000 s	0,000 s	0,018 s	1 min 23 s	4 dias 4h	44 anos

Tabela 9: Tempo necessário para execução do algoritmo na máquina m4.2xlarge. Fonte: Autor.

Instância: m4.4xlarge			**Núcleos: 16**		***Clock*: 2.4**	
	Nº de produtos					
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**	**25**
1	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s
2	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,014 s
3	0,000 s	0,000 s	0,000 s	0,008 s	2,571 s	14 min
4	0,000 s	0,000 s	0,000 s	1,008 s	23 min	21 dias
5	0,000 s	0,000 s	0,009 s	33 s	2 dias 2h	22 anos

Tabela 10: Tempo necessário para execução do algoritmo na máquina m4.4xlarge. Fonte: Autor.

Instância: m4.10xlarge			**Núcleos: 40**		***Clock*: 2.4**	
	Nº de produtos					
Nº de estabelecimentos	**1**	**5**	**10**	**15**	**20**	**25**
1	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s
2	0,000 s	0,000 s	0,000 s	0,000 s	0,000 s	0,005 s
3	0,000 s	0,000 s	0,000 s	0,003 s	1,040 s	5 min
4	0,000 s	0,000 s	0,000 s	0,403 s	9 min	8 dias
5	0,000 s	0,000 s	0,004 s	21 s	20 h	9 anos

Tabela 11: Tempo necessário para execução do algoritmo na máquina m4.10xlarge. Fonte: Autor.

4.2 Protótipo

O algoritmo que busca uma solução para o problema proposto está incorporado em um protótipo funcional, que permite selecionar produtos, informar uma localização e encontrar a melhor maneira de realizar a compra. Existe um limite de até quatro estabelecimentos para a realização das comparações.

O protótipo é um sistema Web, construído seguindo o padrão MVC, escrito em linguagem Python. Foi utilizado o *framework* Django, que estimula o desenvolvimento Web rápido e limpo. O SGBD selecionado para o protótipo foi o MariaDB, um sistema de banco de dados relacional muito semelhante ao MySQL.

MVC é um padrão de projeto que ajuda a separar manipulação de dados e regras de negócio. Pode ser dividido em três elementos: *Model*, *View* e *Controller*. *Model* representa dados e regras de acesso e atualização desses dados. *View* renderiza o conteúdo de uma *Model* e necessita ser atualizada sempre que uma *Model* sofre alteração. E por fim, *Controller* traduz as interações que o usuário tem com uma *View* para ações que a *Model* irá executar. (DEY, 2011)

Pyhton é uma linguagem pequena e fácil de aprender, com recursos surpreendentes. É uma linguagem de interpretação de scripts orientada a objetos. Ela tem propriedades que a fazem especial para computação científica. (DAY, 2014)

Django é um framework Web que facilita para o desenvolvedor Python criar aplicações Web mais rapidamente e com menos código. Django traz para o programador Python benefícios semelhantes aos trazidos por Ruby on Rails para desenvolvedores Ruby. (TAFT, 2006)

> "MariaDB é um servidor de banco de dados que oferece a funcionalidade e substituição para o MySQL. [...] Além das funcionalidades básicas do MySQL, MariaDB oferece um rico conjunto de aprimoramentos de recursos, incluindo mecanismos de armazenamento alternativo, otimizações de servidores e patches." (MariaDB Foundation, 2015)

Como é necessário informações sobre vias e estradas púbicas, foi utilizado o *Google Maps* como sistema de mapeamento geográfico. "Existem muitos sistemas de mapas, como *Yahoo Maps* e *Bing Maps*, mas o mais popular é o *Google Maps*. Na verdade, é a API mais popular da internet."[3] (SVENNERBERG, 2010).

O sistema de mapas fornecerá informações para a construção do grafo que será utilizado pelo algoritmo. Como o sistema é caixa preta, ou seja, não é possível saber detalhes sobre as vias, existe apenas a possibilidade de saber a distância entre pontos, o grafo a ser criado não será fiel a realidade, será um grafo completo, onde todos os vértices são adjacentes a todos os outros vértices. Essa consideração não traz prejuízos para a execução da solução.

Existem várias APIs do Google Maps e, para a construção do protótipo, foram utilizadas três delas: *Google Maps Directions API*, *Google Maps Geocoding API* e *Google Maps Embed API*.

Google Maps Directions API é um serviço que calcula rotas entre localizações através de requisições HTTP. É possível procurar por instruções para alguns modos de transporte (a pé, bicicleta ou carro) e também é possível especificar pontos intermediários da rota. (Google Developers, 2015)

Geocoding é o processo de converter endereços em coordenadas geográficas. *Reverse geocoding* é o processo de converter coordenadas geográficas em endereços que um homem possa ler. *Google Maps Geocoding API* é um serviço que fornece acesso a esses processos via requisição HTTP. (Google Developers, 2015)

[3] Tradução livre feita pelo autor.

Com *Google Maps Embed API* é possível adicionar uma instância do Google Maps a um sistema próprio, sem a necessidade de escrever nenhum código (Google Developers, 2015).

Muitos usuário do Google Earth desejavam a integração com o Google Maps e, em 26 de abril, o Google integrou o *Earth* ao *Maps*, disponibilizando uma visão real da Terra ao *Maps* (Academic OneFile, 2010). Segundo (SILVA; NAZARENO, 2009), com base em um estudo realizado no município de Goiânia-GO, as imagens da cidade disponíveis no *Google Earth* têm 90% de nível de confiança. (OLIVEIRA et al., 2009) realizaram um estudo com treze pontos localizados na cidade de São Leopoldo-RS, e concluíram que a precisão do *Google Earth* é compatível com a escala 1:15.000.

> "A precisão das imagens do software varia de acordo com a região estudada. Em grandes capitais as imagens são de maior qualidade e atualizadas constantemente. [...] Nos trabalhos pesquisados na literatura, pode-se constatar que numa classificação quanto a precisão, o *Google Earth* se enquadrou em escalas variando de 1:2000 à 1:30.000" (ALENCAR; SANTOS, 2013).

O Anexo B apresenta o protótipo, contendo descrição e telas.

5 Conclusão

O locais de compra possuem uma grande quantidade de produtos, que despertam o interesse dos usuários. Os estabelecimentos têm preços variados entre si, dificultando a escolha do usuário de qual é a melhor opção de compra, em qual mercado deve ir, quanto será gasto com o deslocamento até os locais alvo e etc.

Assim foi proposto o desenvolvimento de uma tecnologia capaz de tomar essa decisão para o usuário, buscando encontrar a combinação de compras que gera o menor gasto, levando em conta o valor gasto para visitar os locais.

A solução criada possui realiza um grande número de operações até encontrar a melhor opção, o que torna sua complexidade muito elevada. Foi realizada uma análise em sistemas que comparam preços de um produto ou serviço e que são muito utilizadas pela sociedade, a fim de determinar se o tempo utilizado pelo algoritmo proposto é aceitável ou não para o usuário final.

Sistema	**Tempo de resposta (s)**
Submarino Viagens	9.343
Skyscanner	5.327
Booking	18.063
Buscapé	18.641
Zoom	17.245
Hoteis.com	24.629
Média	**15.541**

Tabela 12: Tempo de resposta de sistemas que buscam preços de produtos ou serviços. Fonte: Autor.

Para a coleta das medidas descritas acima foi utilizada um ferramenta chada *Web Page Test*[1], que faz algumas análises de sistemas disponíveis na *Web*. Um dos resultados apresentados pela ferramenta é o tempo gasto para o sistema processar a solicitação e apresentar um resposta.

Se o tempo de resposta for maior que 15 segundos, o sistema tem que ser arquitetado para "liberar" o usuário física e mentalmente, para que ele possa fazer outras atividades e olhar o resultado quando for conveniente (MILLER, 1968).

Para uma requisição que tem uma resposta visual, como um gráfico, esquema ou grafo, a resposta deve iniciar em uma prazo de dois segundos e ser completamente exibida em no máximo dez segundos (MILLER, 1968).

[1] Disponível em: <http://www.webpagetest.org/>.

Com base na tabela e nos trechos do texto de Miller, é possível afirmar que, com certos limites, o algoritmo tem desempenho dentro do esperado. Para a comparação com apenas um ou dois estabelecimento, o número de produtos pode crescer bastante. Já para maiores quantidades de itens de compra, é necessário mais cuidado. Para análise com três locais, o número máximo de produtos está entre vinte e vinte e cinco, e assim por diante.

5.1 Trabalhos futuros

Este trabalho gerou um algoritmo com uma complexidade muito alta, que torna inviável sua utilização com grande número de produtos. Por esse motivo, pode-se realizar otimizações para diminuir o tempo necessário para executar o algoritmo. A utilização de programação dinâmica, por exemplo, pode diminuir o tempo gasto para executar o algoritmo de Dijkstra.

Seria interessante buscar novas maneiras de resolver o mesmo problema, utilizando algum outro algoritmo exato, como algoritmos genéticos, ou utilizando heurísticas, para obter uma solução ótima ou próxima dela.

A produção de um sistema que mantenha as informações sobre estabelecimentos e preços também é importante. Após desenvolvê-lo realizar a integração com o algoritmo proposta, a fim de incorporar o diferencial competitivo.

Referências

Academic OneFile. Google brings earth view to google maps 403811. *eWeek*, 2010. Disponível em: <http://go-galegroup.ez54.periodicos.capes.gov.br/ps/i.do?id=GALE%7CA224886098&v=2.1&u=capes&it=r&p=AONE&sw=w&asid=ce980dc5b11c63cde408df56d7e0543f>. Citado na página 45.

ALENCAR, C. M. S.; SANTOS, P. L. V. A. da C. Precisão dos dados cartográficos disponíveis na web através da imagem do google earth. *Encontro Internacional Dados, Tecnologia e Informação*, 2013. Disponível em: <http://gpnti.marilia.unesp.br:8085/index.php/DTI/DTI/paper/viewFile/279/103>. Citado na página 45.

ALSUWAIYEL, M. H. *Algorithms Design Techniques and Analysis*. [S.l.]: World Scientific Publishing, 2003. Citado na página 35.

Amazon Web Services. *Instâncias do Amazon EC2*. 2015. Disponível em: <https://aws.amazon.com/pt/ec2/instance-types/>. Citado 2 vezes nas páginas 15 e 41.

ARRUDA, C.; ROSSI, A.; PENIDO, E. Buscapé: Do empreendedorismo à inovação aberta. *Fundação Dom Cabral*, 2011. Disponível em: <http://acervo.ci.fdc.org.br/AcervoDigital/Casos/Casos%202010/CF1005.pdf>. Citado na página 23.

BALACHANDRA, R.; FRIAR, J. H. Managing new product development processes the right way. *Inf. Knowl. Syst. Manag.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 1, n. 1, p. 33–43, jan. 1999. ISSN 1389-1995. Disponível em: <http://dl.acm.org/citation.cfm?id=1234016.1234020>. Citado na página 25.

BARBOSA, L. *Sociedade de Consumo*. Rio de Janeiro: Zahar, 2004. Citado na página 23.

BARBOSA, L.; CAMPBELL, C. *Cultura, consumo e identidade*. 1ª edição. ed. Rio de Janeiro: Editora FGV, 2006. Citado na página 23.

BIG-O CHEAT SHEET. *Big-O Complexity Chart*. 2012. Disponível em: <http://bigocheatsheet.com/>. Citado 2 vezes nas páginas 13 e 29.

BRANDÃO, V. *Brasil Inovador: o Desafio Empreendedor: 40 Histórias de Sucesso de Empresas Que Investem em Inovação*. [S.l.]: Finep, 2006. Citado na página 26.

CHERKASSKY, B. V.; GOLDBERG, A. V.; RADZIK, T. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, Springer-Verlag, v. 73, n. 2, p. 129–174, 1996. Disponível em: <http://link.springer.com/article/10.1007%2FBF02592101>. Citado na página 33.

CORMEN, T. H. et al. *Introduction to Algorithms*. 3ª. ed. [S.l.]: MIT Press, 2001. Citado 2 vezes nas páginas 28 e 35.

DAI, Q.; LIU, Z. Modenpbt: A modified backtracking ensemble pruning algorithm. *Applied Soft Computing*, v. 13, p. 4292–4302, 2013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494613002123>. Citado 2 vezes nas páginas 35 e 36.

DAY, C. Python power. *Computing in Science and Engineering*, v. 16, n. 1, p. 88 –88, 2014. Disponível em: <http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=6756874>. Citado na página 44.

DENG, Y. et al. Fuzzy dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing*, v. 12, p. 1231–1237, 2012. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494611004376>. Citado na página 34.

DEY, T. A comparative analysis on modeling and implementing with mvc architecture. *International Journal of Computer Applications*, 2011. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.1591&rep=rep1&type=pdf>. Citado na página 43.

FREIRE, E. *Inovação e Competitividade: O Desafio a ser Enfrentado pela Indústria de Software*. Dissertação (Mestrado) — Universidade Estadual de Campinas, 2002. Disponível em: <http://www.bibliotecadigital.unicamp.br/document/?code=vtls000242713>. Citado 2 vezes nas páginas 24 e 25.

Giga Mundo. *Busca em árvores ou grafos*. 2009. Disponível em: <http://computacao.gigamundo.com/inteligencia-artificial/busca-em-arvores-ou-grafos/>. Citado 2 vezes nas páginas 13 e 30.

GOLDBARG, M. C.; LUNA, H. P. L. *Otimização Combinatória e Programação Linear - Modelos e Algoritmos*. [S.l.]: Editora Campus, 2005. Citado na página 33.

Google Developers. *Google Maps APIs*. 2015. Disponível em: <https://developers.google.com/maps/>. Citado 2 vezes nas páginas 44 e 45.

HERNANDES, F.; BERTON, L.; CASTANHO, M. J. d. P. C. O problema de caminho mínimo com incertezas e restrições de tempo. *Pesquisa Operacional*, scielo, v. 29, p. 471 – 488, 08 2009. ISSN 0101-7438. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382009000200012&nrm=iso>. Citado na página 33.

HOROWITZ, E.; SAHNI, S. *Fundamentals Of Computer Algorithms*. [S.l.]: Computer Science Press, 1978. Citado na página 28.

KHAN ACADEMY. *Representando grafos*. 2015. Disponível em: <https://s3.amazonaws.com/ka-cs-algorithms/adjacency_list.png>. Citado 2 vezes nas páginas 13 e 32.

LORETO, A. B. *Cálculo da complexidade exata de algoritmos do tipo divisão-e-conquista através das equações características*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2000. Disponível em: <http://hdl.handle.net/10183/2133>. Citado 2 vezes nas páginas 28 e 29.

MariaDB Foundation. *MariaDB*. 2015. Disponível em: <https://mariadb.org/pt-br/>. Citado na página 44.

MILLER, R. B. Response time in man-computer conversational transactions. *Internation Business Machines Corporation*, p. 267–277, 1968. Disponível em: <http://dl.acm.org/citation.cfm?id=1476628>. Citado na página 46.

OLIVEIRA, M. Z. de et al. Imagens do google earth para fins de planejamento ambiental: uma análise de exatidão para o município de são leopoldo/rs. *Simpósio Brasileiro de Sensoriamento Remoto*, p. 1835–1842, 2009. Disponível em: <http://marte.sid.inpe.br/col/dpi.inpe.br/sbsr@80/2008/11.10.17.37/doc/1835-1842.pdf>. Citado na página 45.

RAMACHANDRAN, R. Enabling dispersed innovation — how the united states can uutilize its long tail of talent. *International Journal of Innovation and Technology Management*, v. 09, n. 01, p. 1250007, 2012. Disponível em: <http://www.worldscientific.com/doi/abs/10.1142/S0219877012500071>. Citado na página 26.

ROSELINO, J. E. de S. *A industria de software : o "modelo brasileiro"em perspectiva comparada*. Tese (Doutorado) — Universidade Estadual de Campinas, Campinas, 2006. Disponível em: <http://www.bibliotecadigital.unicamp.br/document/?code=vtls000380464>. Citado na página 25.

SIBEYN, J. *Graph Algorithms*. 2002. Disponível em: <http://users.informatik.uni-halle.de/~jopsi/dssea/chap8.shtml>. Citado 3 vezes nas páginas 13, 34 e 35.

SILVA, L. A. e; NAZARENO, N. R. X. de. Análise do padrão de exatidão cartográfica da imagem do google earth tendo como área de estudo a imagem da cidade de goiânia. *Simpósio Brasileiro de Sensoriamento Remoto*, p. 1723–1730, 2009. Disponível em: <http://marte.sid.inpe.br/col/dpi.inpe.br/sbsr@80/2008/11.14.15.58/doc/1723-1730.pdf>. Citado na página 45.

SKIENA, S. S.; REVILLA, M. A. *Programming Challenges: The Programming Contest Training Manual*. [S.l.]: Springer, 2003. Citado 4 vezes nas páginas 30, 33, 35 e 36.

SVENNERBERG. *Beginning Google Maps API 3*. [S.l.]: Apress, 2010. Citado na página 44.

TAFT, D. Django: Python on a plane. *eWeek*, 2006. Disponível em: <http://go-galegroup.ez54.periodicos.capes.gov.br/ps/i.do?id=GALE%7CA150558878&v=2.1&u=capes&it=r&p=AONE&sw=w&asid=71ce6e06022fb73e66be0ed73cc46f9e>. Citado na página 44.

WANG, X. D. *Computer Algorithms Design and Analysis*. Pequim, China: Publishing House of Electronics Industr, 2001. Citado 2 vezes nas páginas 35 e 36.

XAVIER, A. N. O poder da narrativa e sua eficácia simbólica no campo do consumo contemporâneo. 2013. Disponível em: <http://pucposcom-rj.com.br/wp-content/uploads/2013/11/Adriana-Nogueira-Xavier.pdf>. Citado na página 23.

ZHANG, X. et al. An improved physarum polycephalum algorithm for the shortest path problem. *The Scientific World Journal*, 2014. Disponível em: <http://www.hindawi.com/journals/tswj/2014/487069/>. Citado na página 33.

Anexos

ANEXO A – Código Fonte

O algoritmo construído neste trabalho de conclusão de curso foi uma combinação de *Backtracking* com algoritmo de Dijkstra. Os algoritmos foram implementados em linguagem Python. A seguir é apresentado o código fonte do algoritmo de Dijkstra.

```
def _popmin(self, pqueue):
        """ remove the nearest vertex

        arguments:
                pqueue: dict
        """
        lowest = MAX_INT
        keylowest = None
        for key in pqueue:
                if pqueue[key] < lowest:
                        lowest = pqueue[key]
                        keylowest = key
        del pqueue[keylowest]
        return keylowest

def _dijkstra(self, graph, start):
        """ calculate the sorthest path using Dijkstra's algorithm

        arguments:
                graph: dict of dict
                start: int
        """
        pqueue = {}
        dist = {}
        pred = {}

        for v in graph:
                dist[v] = MAX_INT
                pred[v] = -1
        dist[start] = 0
        for v in graph:
                pqueue[v] = dist[v]

        while pqueue:
```

```
                u = self._popmin(pqueue)
                for v in graph[u].keys():
                        w = graph[u][v]
                        newdist = dist[u] + w
                        if (newdist < dist[v]):
                                pqueue[v] = newdist
                                dist[v] = newdist
                                pred[v] = u

        return dist
```

A implementação de Backtracking utilizada é exibida abaixo.

```
def _distance_value(self, graph, source, stores):
        """ calculate the cost to go from source to all stores
        and return to source

        arugments:
                graph: dict of dict
                source: int
                stores: list of SimpleStore
        """
        vertex = source
        total = 0

        for store in stores:
                min_dist = MAX_INT
                found = False

                dist = self._dijkstra(graph, vertex)

                for s in self.choosen_stores.keys():
                        if dist[s] < min_dist:
                                vertex = s
                                min_dist = dist[s]
                                found = True

                if found:
                        total += min_dist
                        self.visiting_cand.append(vertex)
                        del self.choosen_stores[vertex]

        return total + dist[source]
```

```
def _backtracking(self, kp, p, ks, s, stores, products, graph):
        """ creates all possible shopping combinations

        arugments:
                kp -- int
                p -- int
                ks -- int
                s -- int
                stores -- list of SimpleStore
                products -- list of int
                graph -- dict of dict
        """
        if kp == p:
                total = 0
                self.choosen_stores = {}
                self.visiting_cand = []
                for i, x in enumerate(self.cands):
                        total += stores[x].prices[products[i]]
                        self.choosen_stores[stores[x].id] = True

                products_value = total
                total += self._distance_value(graph, 0, stores)

                if total < self.min_total:
                        self.products_value = products_value
                        self.min_total = total
                        self.best_choice = list(self.cands)
                        self.visiting_order = list(
                        self.visiting_cand)
        else:
                for j in xrange(ks, s):
                        self.cands.append(j)
                        kp += 1

                        self._backtracking(kp, p, ks, s, stores,
                        products, graph)

                        self.cands.pop()
                        kp -= 1
```

ANEXO B – Protótipo

Durante a execução do trabalho foi implementado um protótipo funcional, para facilitar a criação de diferentes situações de compra. Foram inseridos setecentos e oitenta e três produtos no banco de dados da aplicação e criadas vinte e três lojas, em diferentes locais. A localização das lojas foi extraída do Google Maps.

Ao utilizar o protótipo, o primeiro passo é informar o CEP de origem e os produtos de interesse do usuário. O CEP é utilizado para se obter uma localização, que representa o local onde o usuário está, para possibilitar o cálculo da rota. Após a inserção dos dados, o sistema apresenta os preços dos produtos de interesse em todos os estabelecimentos cadastrados, ordenados pelo menor preço. Além dos preços, é apresentada a distância, em linha reta, do usuário até cada uma das lojas. Para o cálculo final, é necessário apenas informar o consumo médio do veículo, para possibilitar a estimativa do custo de locomoção. Como resultado final, é apresentada a rota, utilizando o Google Maps, e uma lista que contém os produtos que devem ser adquiridos em cada uma das lojas que serão visitadas, acompanhados dos preços de cada produto, o total gasto em produtos e o total gasto em locomoção.

As três figuras a seguir são as telas do protótipo.

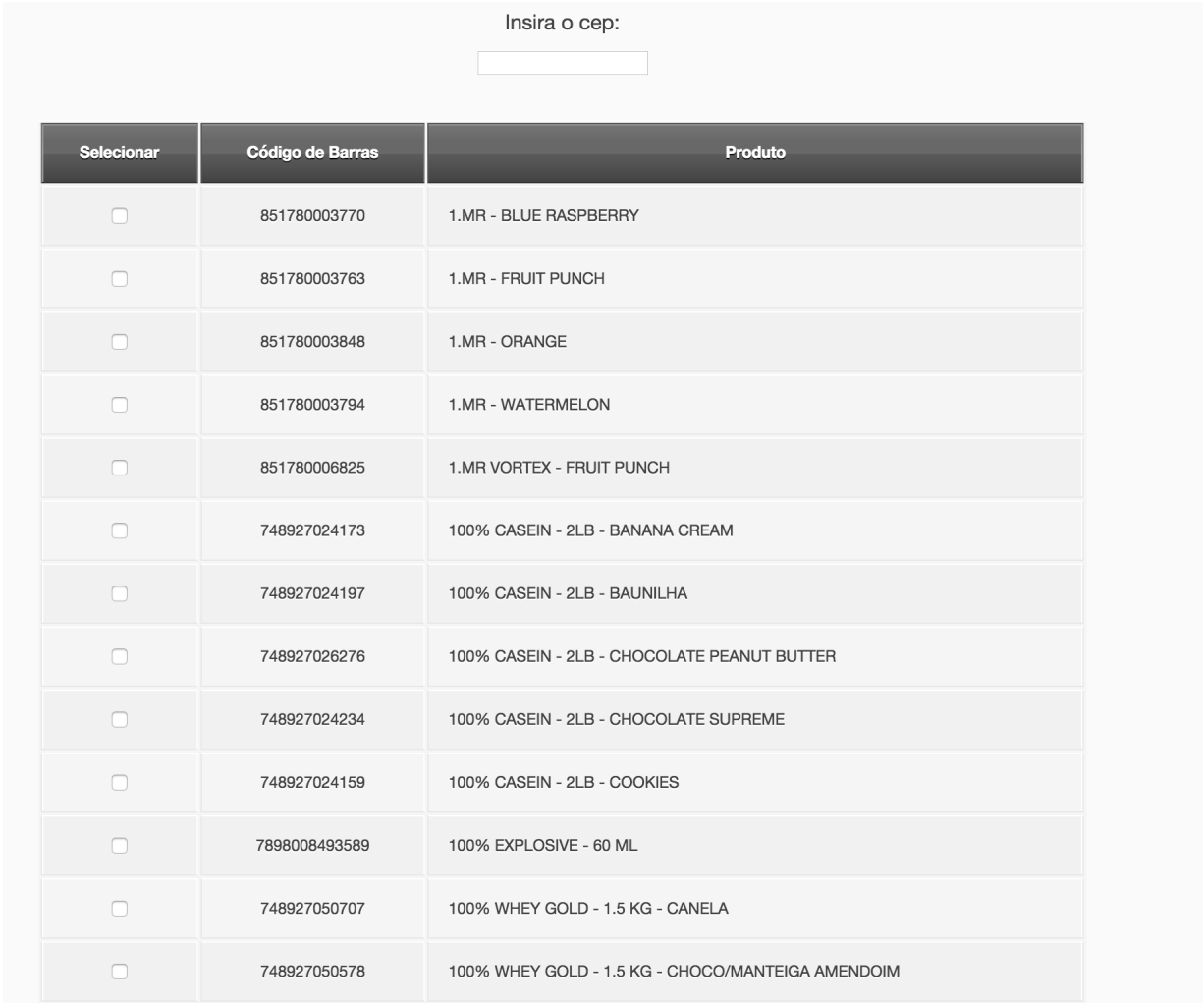


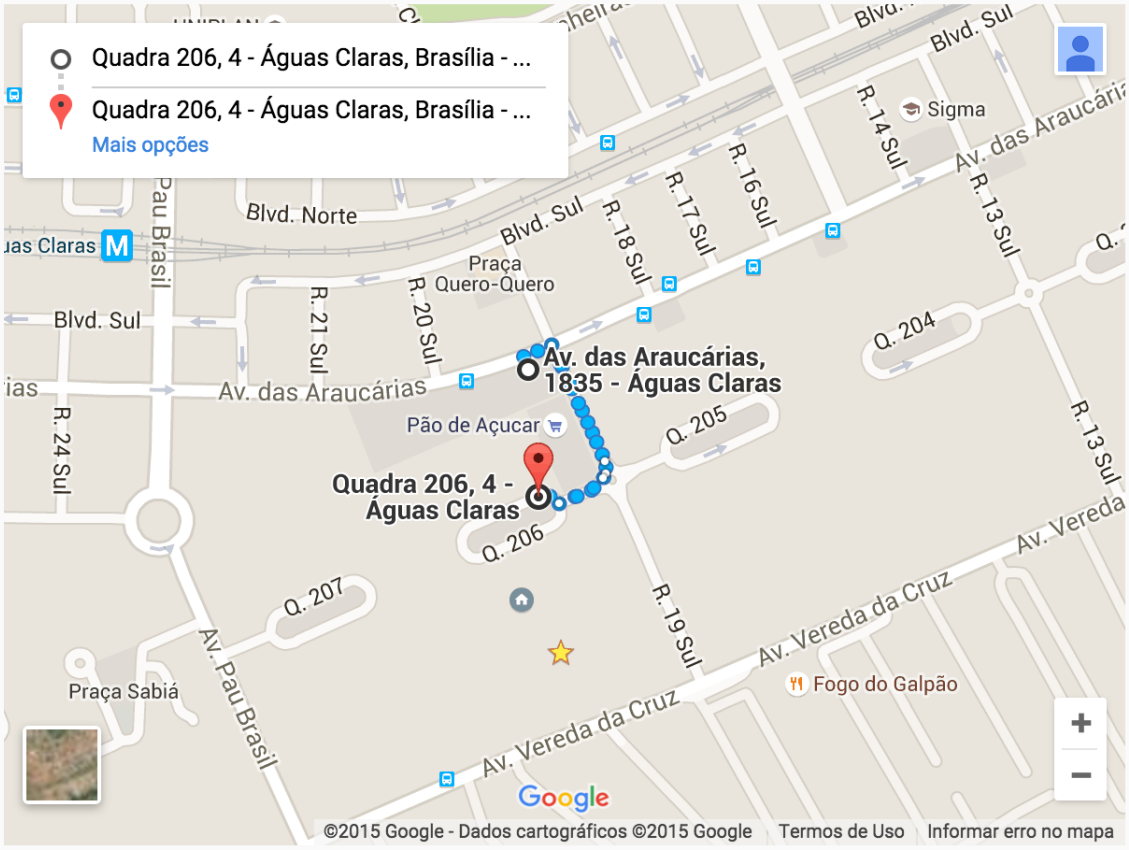
Figura 6: Tela inicial do protótipo. Fonte: Autor.

Média de consumo do veículo (km/l):

Produto	Work Out	Body Building	Body House	Lion Nutri	Up Grade	Prothus Suplementos	Invicta Suplementos	Corpo Atletico	Max Form	Mundo dos Suplementos	Nutricorpus	Pump Suplementos	Max Form
1.MR - ORANGE	R$ 183.47	R$ 233.28	R$ 200.00	R$ 188.32	R$ 175.53	R$ 184.65	R$ 221.60	R$ 205.11	R$ 216.60	R$ 191.54	R$ 213.45	R$ 232.84	R$ 137.19
100% WHEY GOLD - 5LB - CHOCOLATE COCONUT	R$ 276.95	R$ 340.51	R$ 349.90	R$ 323.19	R$ 349.02	R$ 384.10	R$ 314.45	R$ 327.03	R$ 451.70	R$ 377.24	R$ 371.86	R$ 354.81	R$ 293.98
3 WHEY ULTRA IPC 900G - CHOCOLATE	R$ 112.61	R$ 115.87	R$ 139.90	R$ 157.38	R$ 142.48	R$ 132.06	R$ 146.99	R$ 150.50	R$ 185.52	R$ 108.57	R$ 139.15	R$ 118.11	R$ 178.59
Total	R$ 573.03	R$ 689.66	R$ 689.80	R$ 668.89	R$ 667.03	R$ 700.81	R$ 683.04	R$ 682.64	R$ 853.82	R$ 677.35	R$ 724.46	R$ 705.76	R$ 609.76
Distância	0.16 km	0.75 km	0.88 km	10.10 km	11.35 km	11.64 km	12.34 km	13.83 km	13.90 km	14.83 km	16.67 km	16.99 km	17.63 km

Calcular melhor rota

Figura 7: Tela que contém os preços do produtos selecionados. Fonte: Autor.



Work Out

1.MR - ORANGE -- R$ 183.47

100% WHEY GOLD - 5LB - CHOCOLATE COCONUT -- R$ 276.95

3 WHEY ULTRA IPC 900G - CHOCOLATE -- R$ 112.61

Total em produtos: R$ 573.03

Total em locomoção: R$ 1.35

Total: R$ 574.38

Figura 8: Tela final, que contém a rota e os produtos que devem ser adquiridos em cada um dos estabelecimentos.Fonte: Autor.