PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

APLICAÇÃO PARA DEFINIR ESCALONAMENTO DE COMPRAS DE VACINAS, USANDO PROGRAMAÇÃO DINÂMICA PARA REDUZIR CUSTOS FINANCEIROS.

Caio Bueno de Oliveira

GOIÂNIA

2021

Caio Bueno de Oliveira

APLICAÇÃO PARA DEFINIR ESCALONAMENTO DE COMPRAS DE VACINAS, USANDO PROGRAMAÇÃO DINÂMICA PARA REDUZIR CUSTOS FINANCEIROS.

Trabalho de Conclusão de Curso apresentado à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(a): Prof. Me. Alexandre Ribeiro

GOIÂNIA

2021

Caio Bueno de Oliveira

APLICAÇÃO PARA DEFINIR ESCALONAMENTO DE COMPRAS DE VACINAS, USANDO PROGRAMAÇÃO DINÂMICA PARA REDUZIR CUSTOS FINANCEIROS.

Esse Trabalho de Conclusão de Curso foi julgado adequado para obtenção do grau de Bacharel em Engenharia da Computação e aprovado em sua forma final pela Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, em 05 de dezembro de 2019.

Profa. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

Orientador: Prof. Me. Alexandre Ribeiro

Prof. André Luiz Alves

Prof. Me. Max Gontijo de Oliveira

GOIÂNIA

2021

AGRADECIMENTOS

Agradeço, primeiramente e acima de tudo, a Deus por ter conduzido meu conhecimento e dado forças para continuar até a finalização do trabalho.

Aos meus familiares por todo o apoio durante o curso e durante o processo de elaboração do trabalho final.

Ao meu professor, orientador e amigo Alexandre Ribeiro, por toda a motivação pela computação, inspiração, conselhos, ensinamentos e conhecimentos transmitidos ao longo do curso.

Ao meu colega Adolfo Schneider pela colaboração no projeto e na idealização dos códigos.

A todos os professores da universidade que tiveram parte na minha construção de conhecimento ao longo do curso através de esclarecimentos, recomendações e apoio na computação.

RESUMO

O estudo elabora uma técnica de programação dinâmica para calcular o escalonamento que produz o menor custo de compra e armazenamento de vacinas. Ao utilizar-se desta técnica, é possível garantir uma solução ótima para definir as datas de compra de vacinas em qualquer clínica ou hospital. Ao longo do trabalho, foi demonstrada também a técnica de *backtracking* usando recursão e o quão inferior essa é computacionalmente nesta situação. O objetivo do problema é, não apenas criar um algoritmo situacional, mas também um que seja genérico- que possa ser utilizado em diversas situações e intervalos de tempo, sem a perda dos tantos benefícios da técnica de programação dinâmica no processo-. A partir dessa generalização do código, uma solução com dados de entrada configuráveis foi criada.

Palavras-chave: Programação Dinâmica. Generalização. Escalonamento de compra. Redução de Custos.

ABSTRACT

A dynamic programming technique to calculate the lowest costing scheduling for vaccine purchase and storage, was studied. When utilizing this technique, it is possible to guarantee the optimal solution to determine the dates of purchase of vaccines in any clinic or hospital. In this paper, it was also demonstrated the backtracking recursive technique and how inferior it is computationally in this situation. The objective of the situation problem is not only to create a situational algorithm but also a generic one that can be used in more diverse situations and time intervals, without any loss of the many benefits of the dynamic programming technique. Due to the generalization of the algorythm, a solution with configurable input data was created.

Keywords: Dynamic Programming. Generalization. Purchase Scheduling. Cost Reduction.

SUMÁRIO

LISTA DE TABELAS

1 INTRODUÇÃO

No capítulo de desenvolvimento, esse trabalho explica brevemente algumas características da distribuição e armazenamento de vacinas no país e a importância desse processo. Nessa perspectiva, com o aumento da demanda de vacinas, é necessário um maior controle sobre as compras pelos hospitais e clínicas de vacinação. Dessa forma, é um processo, desde a compra e o envio até seu armazenamento e, por fim, sua administração ao paciente. Nesse sentido, por ser um termolábil, todas as vacinas devem passar pela cadeia do frio para garantir a segurança do produto e prolongar sua vida útil. Analisando os custos gerados por todo esse processo, vê-se a possibilidade da optimização destes, diminuindo o capital usado e aumentando a efetividade da vacinação e armazenamento.

Demonstra-se que, durante a compra dos lotes de vacinas pelo estabelecimento (clínica de vacinação, hospital, etc.), há algumas variáveis que podem afetar no custo final da vacinação, como por exemplo o próprio preço da unidade de vacina, a demanda, o custo de entrega e o custo de armazenamento.

Para isso, a computação pode ser usada como ferramenta de apoio. Nesse propósito, ao efetuar os cálculos de compras de vacinas, é possível criar um algoritmo que calcule o melhor escalonamento para a aquisição destas, com a maior eficiência e menor custo possível.

Dentro da computação, existem várias técnicas de tomadas de decisão, porém cada técnica é aplicada em um tipo de situação. Assim, há a busca de resultados diferentemente ou seguindo outros caminhos para chegar à solução. As técnicas que mais se assemelham com a solução do problema que este trabalho analisa são: Técnica de Backtracking Recursivo e Técnica de Programação Dinâmica. Frente a isso, é necessário apenas verificar se o problema se encaixa na solução de Programação Dinâmica.

Esse estudo introduz comparativamente a técnica de backtracking com recursão e a técnica de programação dinâmica, possibilitando a análise da diferença de custo computacional entre elas e demonstrando a superioridade da utilização da programação dinâmica nesse caso de uso. Faz-se a apresentação do algoritmo problema da Mochila Binária, para que seja utilizado, na situação problema tratada no projeto, obtendo então os resultados esperados.

Caso a situação problema cumpra os requisitos para um problema de optimização com sobreposição de subproblemas, poderá ser utilizada a técnica de programação dinâmica para

sua resolução. Com a comprovação de uma subestrutura ótima, é garantida a possibilidade da criação de uma solução ótima, sendo assim, pode-se aplicar a técnica de programação dinâmica.

No artigo de Cechin (2019), foi demonstrado um cenário de um hospital específico, em que foi utilizada a técnica de programação dinâmica para otimizar as compras de vacinas, chegando a atingir uma economia do valor final. Sendo assim, a proposta do projeto é criar um algoritmo de programação dinâmica generalizado, para que sirva para qualquer estabelecimento que distribua vacinas e tenha previsão de demandas, tendo os parâmetros de entrada configuráveis pelo usuário.

No capítulo de Proposta do Projeto, este trabalho analisa o algoritmo criado para solucionar a situação de forma ótima e, utilizando este, obtém resultados nas instâncias simuladas e os analisa. Garante-se então que a solução criada a partir de uma subestrutura ótima é, então ótima, atendendo às expectativas de resultados.

2 ARMAZENAMENTO DE VACINAS

2.1 Situação atual mundial

A demanda de vacinas ao redor do mundo é crescente e, com isso, no momento atual tem-se o maior nível de acessibilidade a dados sobre a distribuição dessas já apresentado até então, em todos os continentes do mundo. Levando em conta o banco de dados da *Our World in Data* da universidade de Oxford, pode-se analisar a situação do Brasil em comparação aos outros países líderes de vacinação ao redor do mundo.

De acordo com a Universidade de Oxford, até o dia 17 de novembro de 2021, o Brasil possui 60,4% da sua população completamente vacinada, representando um longo caminho percorrido, principalmente considerando a situação atual. Entretanto, esse é um desafio que não terá fim. Como qualquer vírus, vacinas de reforço são necessárias e vacinas novas podem ser constantemente criadas e distribuídas para lidar com as variantes que surgem ao longo do tempo. Levando isso em conta, fica claro como o transporte e armazenamento das vacinas será crucial para tempos atuais e futuros, independentemente da situação mundial. Logo, deve-se planejar e otimizar itens para o aumento da eficiência.

2.2 Armazenamento seguro: importância do monitoramento da temperatura de vacinas

O controle das condições térmicas das vacinas é rigoroso e sua segurança depende de todos os locais onde os produtos são mantidos durante a cadeia do frio. Os produtos Termolábeis, que são aqueles que exigem um controle rigoroso de temperatura, devem ser submetidos a essa cadeia para prolongar sua duração por meio da conservação por resfriamento. As vacinas requerem esse processo de controle desde sua saída da fábrica até seu destino final- como postos de saúde, hospitais, farmácias ou clínicas de vacinação-. Desse jeito, a cadeia do frio é executada desde o recebimento do produto pelo hospital ou clínica de vacinação até a administração no paciente. Em resumo, trata-se de um ciclo de conservação que necessita constantemente de manutenção e monitoramento, garantindo assim a preservação da qualidade e eficácia do produto. Nessa colocação, é crucial acrescentar que A Agência Nacional de Vigilância Sanitária (Anvisa) é a responsável por estabelecer os critérios de conservação de produtos termolábeis (ALMEIDA, 2020).

A seguir, são apresentados alguns insumos que exigem o controle de temperatura para a sua distribuição, desde o produtor até o cliente final.

- Medicamentos;

- Vacinas;
- Hemoderivados;
- Hemocomponentes;
- Órgãos doados para transplante;
- Leite materno;
- Alimentos.

Tratando-se de produtos da área da saúde, essa tarefa torna-se essencial, não somente para a proteção do produto em si, mas também dos usuários para os quais a vacina é destinada, os quais buscam se proteger da doença. Sendo assim, um investimento inicial é necessário para a aquisição de sistemas de monitoramento que garantam os padrões de temperatura de acordo com a determinação do fabricante em todas as etapas da cadeia do frio (ALMEIDA, 2020).

Para a adequada conservação dos imunobiológicos, a cadeia de frio é composta das seguintes etapas:

- fabricante de vacinas;
- distribuição/transporte;
- chegada da vacina no local de armazenamento do fornecedor;
- geladeiras e câmaras de estocagem;
- administração ao usuário/paciente.

2.3 Rede de Frio

O Programa Nacional de Imunizações conta com uma Rede Nacional constituída por uma estrutura física: a Rede de Frio. Desse modo, ocorre a viabilização do processo logístico, objetivando a promoção da garantia da qualidade dos imunobiológicos adquiridos e ofertados à população. Visando a manutenção adequada da cadeia do frio, a rede do frio é um sistema que opera por meio de normatização, planejamento, avaliação e financiamento. A estrutura da Rede de Frio permeia as três esferas de gestão, organizando-se em instâncias com fluxos de armazenamento e distribuição. Assim, compõem o Sistema as seguintes instâncias: (MINISTÉRIO DA SAÚDE, 2017)

- Nacional;
- Estadual;
- Regional (conforme estrutura do estado);
- Municipal;
- Local.

2.4 Cadeia do Frio

É o processo logístico da Rede de Frio para conservação dos imunobiológicos, desde o laboratório produtor até o usuário e inclui as etapas de recebimento, armazenamento, distribuição e transporte. Portanto, assegura a preservação do produto e suas características originais. (MINISTÉRIO DA SAÚDE, 2017)

Tabela 1 - Período de armazenamento dos imunobiológicos nas instâncias da Rede do Frio

Centrais Estaduais	Centrais Regionais	Sala de Vacinação
6 a 12 meses	3 a 6 meses	1 mês

Fonte: Ministério da Saúde. Manual da Rede do Frio. 2017.

3 PROBLEMA A SER TRATADO

3.1 Artigo: Programação dinâmica aplicada à redução de custos nas compras de vacinas de um hospital

No artigo de Cechin, Biasuz, Falavigna e Corso (2019), há uma proposta interessantíssima de aplicar a programação dinâmica para a redução de custos na compra de vacinas por um hospital específico, com dados específicos e constantes. Com base nele, viu-se então que a programação dinâmica é uma ferramenta da computação viável para resolver tal problema.

É dito no citado artigo que, dado um período de meses pré-determinado (no caso foi mostrado um exemplo de 4 meses), é possível calcular a melhor compra de vacinas para cada mês, levando em conta os seguintes fatores:

- Preço das vacinas por unidade em determinado mês;
- Custo de entrega;
- Custo de armazenamento.

Todos esses fatores são levados em conta na definição do algoritmo, aumentando a complexidade e incidindo sobre o preço final da compra de vacinas de maneira direta.

3.2 Generalizar o Algoritmo

Levando em conta que o artigo citado tem como proposta generalizar a entrada de dados e usar a mesma técnica de programação dinâmica para resolver o problema, disponibilizar uma solução de otimização de recursos para outras empresas que possuem problemas similares de armazenamento de vacinas e tenham previsão da demanda dos meses seguintes e dos custos é assertivo. Partindo desse princípio, é possível criar um algoritmo cujos dados de entrada podem variar de acordo com as demandas e custos de cada momento e de cada usuário, possibilitando então que o funcionário da própria clínica insira os dados, como: Período em meses, Demanda de cada mês, Custo da vacina de cada mês, Custo de Entrega em cada mês, Custo de Armazenamento mensal, Limite de Estoque.

Com isso, o funcionário da clínica pode, de forma independente, refazer o cálculo no momento adequado, alterando os dados de entrada de acordo com as circunstâncias. Caso algum dos parâmetros citados acima sofra alguma alteração, pode-se recalcular o escalonamento de compras de vacinas novamente para a optimização do valor.

Reduzindo os custos da compra de vacinas, possibilita-se que mais unidades possam ser compradas, aumentando a eficiência da vacinação no local e facilitando que mais pessoas sejam imunizadas.

4 PROGRAMAÇÃO DINÂMICA

Para que a técnica de programação dinâmica seja melhor conceituada é importante que seja entendido antes o conceito de *backtracking*. Nessa análise, será exposto a seguir como o algoritmo funciona, como pode ser usado e porque é importante.

4.1 Backtracking

Um algoritmo *backtracking* é um método recursivo que utiliza a abordagem do problema através da força bruta. Ele constrói soluções utilizáveis para um problema de optimização combinatória passo a passo. Considerando todas as soluções viáveis, ele sempre encontra a solução ótima. Assim, caracteriza-se como um método de busca exaustiva. (KREHER; STINSON, 1999)

O nome *backtracking* se dá pelo comportamento do algoritmo que, quando a solução atual não é adequada, retrocede seus passos anteriores para tentar outra. Diferentemente da técnica de programação dinâmica, que busca uma solução ótima, o *backtracking* busca solucionar problemas que possuem diversas soluções.

Os computadores modernos são tão rápidos que, em alguns casos, a força bruta é uma solução efetiva para o problema- como a contagem de itens em um conjunto é um problema mais facilmente resolvido construindo-o do que utilizando parâmetros combinatórios sofisticados. Isto é, respeitando um certo limite de número de itens gerados para que o tempo de execução seja aceitável. (SKIENA; REVILLA, 2006)

Um computador pessoal, atualmente, tem em média um *clock* de 1 *gigahertz*, fato que possibilita um bilhão de operações por segundo. Sendo assim, pode-se esperar que a máquina consiga buscar, entre alguns milhões de itens por segundo, uma informação. Porém, em escala, um milhão provavelmente significa todos os arranjos de 10 ou 11 itens aproximadamente ou um milhão de subconjuntos seria em torno de 20 combinações. O *backtracking* é então utilizado em problemas de larga escala ou busca exaustiva, pois ao podar o espaço de busca garantindo apenas elementos que realmente importam, é possível utilizar a técnica da forma mais poderosa possível. (SKIENA; REVILLA, 2006)

Porém, essa técnica pode não representar a melhor solução em muitos casos. Um algoritmo que utiliza a técnica de *backtracking* pode apresentar um grande problema de recálculo, fazendo com que a complexidade do algoritmo não se torne eficiente em muitos casos. Um algoritmo de *backtracking* para resolver um arranjo de n chega a uma complexidade de $O(2^n)$.

4.2 Conceituando a Programação dinâmica

A técnica de programação dinâmica é um método tanto de optimização matemática quanto de programação computacional. Em ambos os contextos, ele serve para simplificar um problema complexo, dividindo-o em subproblemas mais simplificados.

Através da combinação de soluções, pode-se resolver subproblemas separadamente através de recursão. Porém, aplica-se a programação dinâmica quando os problemas se sobrepõem. Sendo assim, gravando as respostas em uma tabela, podemos evitar recálculos desnecessários ao resolver o problema, armazenando os dados em uma matriz.

4.3 Quando é viável aplicar programação dinâmica em um problema

Em uma situação problema que pode ser resolvida ou ter sua solução melhorada através da computação, pode-se ter muitos métodos que realizam as mesmas funções. Aqui, temos as situações em que certamente a técnica de programação dinâmica é melhor aplicada e produz resultados próximos aos desejados. Consistentemente sabendo disso, pode-se pensar em soluções de programação dinâmica para outros problemas que atendem a estes aspectos que segundo Cormen (2019) são:

> Em geral, aplicamos a programação dinâmica a problemas de otimização. Tais problemas podem ter muitas soluções possíveis. Cada solução tem um valor e desejamos encontrar uma solução com o valor ótimo (mínimo ou máximo). Denominamos tal solução "uma" solução ótima para o problema, ao contrário de "a" solução ótima, já que podem existir várias soluções que alcançam o valor ótimo.

4.4 Subestrutura ótima

Caracterizar a estrutura de uma solução ótima é o primeiro passo para resolver um problema de otimização utilizando a programação dinâmica. "Subestrutura ótima: soluções ótimas para um problema incorporam soluções ótimas para subproblemas relacionados, que podemos resolver independentemente." (CORMEN, 2009, p. 303). Se a solução ótima para um problema *P*, de tamanho *n*, pode ser calculada pela observação das soluções ótimas de subproblemas [p1, p2, ...] com tamanho menor que *n*, então o problema *P* é aceito como um problema que possui subestrutura ótima.

Segundo Cormen (2009), sabe-se que para elaborar uma solução de problema utilizando a programação dinâmica, segue-se uma sequência de quatro etapas. São essas:

- A caracterização de uma subestrutura ótima;
- A definição de seu valor recursivamente;
- O cálculo deste valor;
- A construção da solução ótima a partir destes cálculos.

Segundo Cormen (2009), para definir uma subestrutura ótima é necessário seguir os seguintes passos:

- A solução do problema deve consistir em tomadas de decisã e essa decisão produz um ou mais subproblemas a serem resolvidos;
- É necessário supor que para o dado problema exista uma solução ótima;
- Determinar quais subproblemas decorrem a partir da solução ótima e caracterizar melhor o espaço de subproblemas resultante;
- Demonstrar que, dentro de uma solução ótima para o problema, todas as soluções de subproblemas usadas dentro deste são ótimas.

Na técnica de programação dinâmica, é frequentemente usada a subestrutura ótima de baixo pra cima (*bottom-up*). Ou seja, primeiro encontra-se soluções ótimas para subproblemas e, a partir da resolução desses, encontra-se a solução ótima para o problema geral. Isso acarreta em escolher entre os subproblemas, selecionando quais serão utilizados na solução final. (CORMEN, 2009)

4.5 Subproblemas Sobrepostos

A programação dinâmica é aplicável em um problema de optimização apenas se esse elemento existir. Ou seja, um algoritmo recursivo resolveria os mesmos subproblemas, porém repetidas vezes ao invés de sempre gerar novos subproblemas. O número total de subproblemas distintos é um polinômio no tamanho de entrada. Logo, dizer que um problema de optimização possui subproblemas sobrepostos significa que um algoritmo recursivo reexamina o mesmo problema repetidamente. Os algoritmos de programação dinâmica, então, tiram proveito disso, resolvendo cada subproblema apenas uma vez e armazenando os resultados em uma tabela, que será reexaminada sempre que necessário com um tempo busca constante.

Algoritmos considerados corretos e eficientes são conhecidos por muitas questões de grafos importantes, como caminho mais curto, árvore geradora mínima, e coincidências, as quais são tratadas com *backtracking*. Algoritmos gulosos cão conhecidos por apresentarem a solução mais “natural” de cada problema, porém muitas vezes esta forma é errada, e na ausência de provas para esta solução, ela muito provavelmente falhará. A programação dinâmica então,

proporciona uma forma de design de um algoritmo customizado que ao mesmo tempo pode analisar todas as possibilidades enquanto armazena resultados e evita recálculos. (SKIENA; REVILLA, 2006)

O *backtracking* é um procedimento que atende às expectativas, sendo esse uma busca em profundidade. Esse método é eficiente, pois marca todos os vértices para não revisitá-los. Porém, é computacionalmente caro por buscar em todos os caminhos soluções possíveis. Sendo assim, na programação dinâmica pode-se implementar eficientemente um algoritmo recursivo guardando resultados parciais. O artifício é enxergar onde o programa recursivo repetitivamente computa os mesmos subproblemas várias vezes. Dessa maneira, guardar os resultados em uma tabela ao invés de recalculá-los, levará a um algoritmo eficiente. (SKIENA; REVILLA, 2006)

4.6 Problema da Mochila

Suponha que um alpinista deve selecionar os itens que vai carregar na mochila. Esse alpinista deve considerar a capacidade da mochila conjuntamente aos vários itens disponíveis. Cada item possui um valor e o alpinista busca maximizar o valor de utilidade total de sua carga. Considere a capacidade da mochila como C e o número de itens como n que variam de 0 até $n – 1$. Cada objeto i tem peso $W[i]$ e valor $V[i]$.

Exemplo: Uma mochila com capacidade $C = 10$ e 3 objetos enumerados {0, 1, 2}, cujos pesos são {7, 5, 4} e valores {4, 3, 2}, respectivamente. As combinações possíveis onde os itens pesam 10 ou menos serão:

Objetos Escolhidos:

{0} -> Valor Total = 4, Peso Total = 7;

{1} -> Valor Total = 3, Peso Total = 5;

{2} -> Valor Total = 2, Peso Total = 4;

{1,2} -> Valor Total = 5, Peso Total = 9;

Podemos afirmar que a solução ótima é o subconjunto {1,2}.

Esse exemplo é chamado de Mochila Binária, em que cada item pode apenas ser levado ou não ser levado e apresenta a propriedade da subestrutura ótima. Nesse caso, ele considera a carga mais valiosa que pesa no máximo W quilos e, se for removido o item j da carga da mochila, o resto deve ser a carga mais valiosa que pese $W – wj$ que ele possa levar nos $n – 1$ itens originais, excluindo o j. (CORMEN, 2009)

O valor máximo pode ser obtido ao calcular o máximo de 2 valores:

- O valor máximo obtido adicionando $n - 1$ itens ;
- O valor do item n mais o valor máximo obtido adicionando $n - 1$ itens sem exceder o valor de peso máximo suportado.

Subestrutura ótima. Em uma instância com os parâmetros (n, C, W, V), tratando-se de um objeto i qualquer, tem-se 2 casos distintos:

1) Objeto i pertence a um subconjunto ótimo, logo $V[i]$ + valor da solução ótima mas com capacidade $C - W[i]$.
2) Objeto i não pertence a um subconjunto ótimo, logo a solução ótima é a solução de um novo problema sem o item i.

(TOMMASINI, 2014)

Função recursiva. Obedecendo a propriedade da subestrutura ótima, define-se a função *Mochila*, a qual recebe os objetos em qualquer ordem e enumerados de 0 a $n - 1$. Utilizaremos 2 parâmetros, os quais são o*bjetoAtual* e *weight*, tendo em vista que *Mochila*(*objetoAtual*,*weight*) será o maior valor possível obtido com os itens até $n - 1$ usando weight como a capacidade disponível da mochila.

Serão utilizados os vetores W para armazenar os pesos dos objetos, V para armazenar os valores

Se $W[1..n]$ e $V[1..n]$ são vetores numéricos e X é um subconjunto do intervalo $1..n$, denotaremos por $W(X)$ e $V(X)$ as somas $\sum_{i \in X} W[i]$ e $\sum_{i \in X} V[i]$ respectivamente. Deve-se encontrar um subconjunto X do intervalo $1..n$ tal que $W(X) \leq c$ e $V(X) \geq V(Y)$ para todo subconjunto Y de $1..n$ tal que $W(Y) \leq c$ (FEOFILOFF, 2020).

Algoritmo 1 –Algoritmo da mochila binária utilizando recursão

Algoritmo MOCHILA(*objetoAtual*, *weight*)

```
1.  se objetoAtual = n então
2.     devolva 0
3.  se memoriza[objetoAtual][weight] já foi calculado então
4.     devolva memoriza[objetoAtual][weight]
5.  r1 = infinito, r2 = infinito
6.  //Tentar colocar o objeto de índice objetoAtual na mochila se ainda tiver espaço para ele
7.  se W[objetoAtual] ≤ weight então
8.     r1 = V[objetoAtual] + MOCHILA(objetoAtual + 1, weight – W[objetoAtual]);
9.  //Agora calcular a melhor resposta sem usar objetoAtual, simplesmente ignorando-o:
```

10. *r*2 = MOCHILA(*objetoAtual* + 1, *weight*)
11. *resposta* = max(*r*1, *r*2)
12. *memoriza*[*objetoAtual*][*weight*] = *resposta*
13. **devolva** resposta

Tem-se aqui um código que utiliza a técnica de recursão para calcular o melhor valor possível a ser carregado pela mochila. Nesse contexto, o código é considerado uma solução válida para o problema, mas não ótima. Todas as vezes que a função for acionada, ela será empilhada gerando muita ocupação de memória.

Na primeira e segunda linha, foi descrita a condição de finalização, a qual significa basicamente que o algoritmo chegou ao n, que é o último objeto a ser testado. Na linha 3 e 4, há um tratamento para prevenir que a função recalcule uma posição. No entanto, ele apenas evita um recálculo literal de determinada posição. Na linha 5, duas variáveis de resposta irão receber um valor infinito para que, a partir daí, elas possam ser minimizadas. Na linha 7, é testado se o objeto cabe no espaço remanescente da mochila. Em caso positivo, na linha 8 a primeira variável de resposta recebe o valor obtido pela aquisição do objeto atual na mochila e da recursão que irá calcular os próximos objetos e valores. É possível visualizar que, na linha 10 do pseudo-código, a resposta atual será calculada empilhando a função e, nesse ponto, temos um recálculo lógico da resposta. Isso ocorre porque toda vez que a comparação de máximo for executada, o programa irá calcular e verificar entre o valor atual com o objeto atual ou o valor sem o objeto. O maior problema desse tipo de abordagem é o número de vezes que o mesmo problema é resolvido, gerando então um algoritmo de complexidade O(2n). Em seguida, a resposta final recebe o maior valor apresentado pelas respostas 1 e 2 e, então, é devolvido.

Algoritmo 2 –Algoritmo da mochila binária utilizando programação dinâmica

1. **leia** *n*
2. **leia** *C*
3. **para** *objetoAtual* = 1 **enquanto** *objetoAtual* ≤ *n* **faça**
4. | **leia** *weight*[*objetoAtual*]
5. | **leia** *Valor*[*objetoAtual*]
6. **fim**
7. *resposta = 0*
8. **para** *objetoAtual* = 1 **enquanto** *objetoAtual* ≤ *n* **faça**
9. | **para** *Weight* = 0 **enquanto** *Weight* ≤ *C* **faça**
10. | | *memoriza*[*objetoAtual*][*Weight*] = *-infinito*
11. | **fim**
12. **fim**
13. *memoriza*[*0*][*0*] = 0
14. **para** *objetoAtual* = 1 **enquanto** *objetoAtual* ≤ *n* **faça**

15. | **para** *Weight* = 0 **enquanto** *Weight* ≤ *C* **faça**
16. | | *memoriza*[*objetoAtual*][*Weight*] = *memoriza*[*objetoAtual* - 1][*Weight*]
17. | | **se** *Weight* ≥ *weight*[*objetoAtual*] **então**
18. | | | *memoriza*[*objetoAtual*][*Weight*] = max(*memoriza*[*objetoAtual*][*Weight*], *memoriza*[*objetoAtual* - 1][*Weight* - *weight*[*objetoAtual*]] + *Valor*[*objetoAtual*]);
19. | | **fim**
20. | | *resposta* = max(*resposta*, *memoriza*[*objetoAtual*][*Weight*])
21. | **fim**
22. **fim**
23. **devolva** resposta

Nas duas primeiras linhas são lidos os valores de quantidade de objetos e de capacidade máxima. Em seguida, nas linhas 3 até 6, são preenchidos os vetores de peso e de valor para cada objeto. Na linha 7, a resposta recebe 0. Nas linhas 8 a 12, a matriz memoriza que tem comprimento C e a altura n recebe infinito em todas as posições. Na linha 13, a matriz recebe 0 na posição [0][0]. Na linha 14, pode ser demonstrada a complexidade do algoritmo em si. Nesse ponto, serão tomadas as decisões dinamicamente. Sendo assim, nota-se nos dois comandos para quê a complexidade do algoritmo é de ordem n×C ou, simplificando, pode-se dizer que é O(n2). Sendo assim, ele irá na linha 16 receber o objeto anterior memorizado. Logo em seguida, na linha 17 e com a condição do objeto caber no espaço remanescente da mochila, ele irá fazer a comparação da linha 18. A maximização de valor da linha 18 se dá entre o objeto atual e o objeto na posição [objetoAtual – 1][Weight - weight[objetoAtual] + o valor do objeto atual. E, por fim, na linha 20 a resposta recebe o máximo entre a resposta anterior ou a posição atual da matriz.

Tabela 2 - Peso e Valor dos Itens do Exemplo

	1	2	3	4
W	4	2	1	3
V	500	400	300	450

Fonte: Elaborado pelo autor

Tabela 3 - Matriz de execução da solução de programação dinâmica

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	700	700	800
4	0	300	400	700	700	850

Fonte: Elaborado pelo autor

Pode-se notar que, ao utilizar da técnica de programação dinâmica, todos os dados necessários ficam armazenados na tabela e nesse algoritmo em si, já é reaproveitado o último valor na linha 13, em que ele já é salvo e será comparado com o próximo valor calculado. Dessa forma, quando requer-se o valor anterior para que seja comparado com o novo valor calculado (na linha 15), seu acesso é instantâneo. Isso acarreta na redução da complexidade do algoritmo que, utilizando a recursão, seria de ordem $O(2^n)$ para um de ordem $O(n^2)$.

4.7 Passos para construir um algoritmo de Programação Dinâmica

Passo 1: Identificar o subproblema em palavras

Iniciar a escrita do código antes de pensar criticamente sobre o problema é uma abordagem ineficiente. Uma boa maneira de assimilar o problema de forma compreensível antes de escrever o código é fraseá-lo em palavras, descrevendo assim o subproblema identificado dentro do problema original. Para organizar as ideias, é recomendado escrever, em um papel ou algo do gênero, as informações necessárias para a resolução do problema e escrever o subproblema com essa questão já em mente.

Passo 2: Escrever o problema como uma decisão matemática recorrente

O motivo desse passo é que a recorrência matemática ou decisão repetitiva serão eventualmente o que será colocado em código. E, além disso, tendo escrito o subproblema em palavras anteriormente, já é fornecida uma base para que o problema seja escrito matematicamente. Ou seja, se o que for escrito no primeiro passo se mostrar como algo difícil de mentalizar matematicamente, então provavelmente o subproblema está errado. E, algumas perguntas podem convencionalmente ajudar neste passo, como por exemplo:

- Qual decisão está sendo tomada repetidamente?
- Se o algoritmo está no estágio i, qual informação é necessária para a decisão que será tomada no estágio i+1?

Passo 3: Identificar a subestrutura ótima

Após ter escrito em palavras o problema e notado em que trecho ocorre a repetição de tomada de decisão, é possível então calcular sua subestrutura ótima, escrevendo matematicamente. Assumindo que todas as decisões tomadas até então foram ótimas, é possível discernir como a próxima decisão a ser tomada será ótima, tornando então possível transformar isso em uma expressão matemática.

Passo 4: Codificar o algoritmo salvando os dados em uma tabela diretamente

Tendo feito os 3 passos anteriores, é possível escrever o algoritmo utilizando a programação dinâmica diretamente, pois já há informações necessárias para criar a tabela e utilizar dela mesma para que os próximos cálculos sejam optimizados.

5 PROPOSTA DO TRABALHO

5.1 Artigo estudado sobre uso da programação dinâmica para reduzir custos na compra de vacinas

Após estudar o algoritmo da mochila-binária, foram notadas semelhanças matemáticas que tornam possível a modelagem desse algoritmo para sua utilização no problema de escalonamento de compra de vacinas.

Iniciando em um estágio e de um tempo t qualquer, um estado de entrada S_0 e um estado de saída S_t e uma variável de decisão X_t que poderá influenciar na saída e também no custo, tem-se um custo de estágio $f_t(e_{t-1},x_t)$. Sob influência também da demanda d_t e a transformação φ_t do estágio t, sendo que φ_t irá expressar as saídas como função das entradas. A equação de custo $f_t(e_{t-1},x_t)$ irá medir a eficiência da transformação das entradas e saídas. Logo, temos $S_t = \varphi_t (S_{t-1}, x_t)$.

As equações de continuidade de fluxo utilizadas foram as seguintes:

$$e_t = e_{t-1} + x_t - d_t \quad (1)$$

max ou min $f_t(e_{t-1},x_t)$ = max ou min [(custo de estado S_{t-1}) + (custo viável de decisão x_t) + (custo de estado S_{t+1})] (2)

(Cechin, 2019)

5.2 Instância Analisada pelo Artigo Estudado

No trabalho de Cechin (2019), foram apresentados parâmetros adquiridos a partir de dados de um hospital. Nesse cenário, foram analisados os custos de compra e estoque do medicamento por um período de 4 meses.

Foram abordados os seguintes dados: O tamanho dos lotes é predeterminado, sendo de 5000 unidades e pode-se apenas comprar múltiplos deste. O estoque inicial contém 1 lote (5000 unidades) e o limite máximo de estoque é de 60000 unidades. O período tratado é de 4 meses, mesmo as vacinas possuindo validade superior a isso e não é permitido que sobrem vacinas ao final do período. As demandas dos 4 meses são respectivamente 4, 2, 3 e 2 lotes (20000, 10000, 15000 e 10000 unidades). Os valores que atuam sobre o custo e podem sofrer alterações ao longo dos meses são: O custo de entrega, que no primeiro mês é de R$ 1100,00 e nos meses seguintes de R$ 1200,00; o custo de armazenamento, que é de 1,85% do valor total mensal nos

2 primeiros meses e de 2% nos meses seguintes e o preço por unidade de vacina, que é de R$ 35,00 nos 2 primeiros meses e de R$ 40,00 nos 2 meses restantes.

5.3 Cenário esperado sem o uso da técnica

Foi feito um cálculo utilizando os mesmos parâmetros da instância do artigo. Logo, foram compradas as quantidades de vacinas de acordo com a demanda, questão que seria provável caso não fosse usado o algoritmo para calcular automaticamente. Seguem os resultados:

Tabela 4 - Cenário do Artigo Estudado sem o uso da técnica de programação dinâmica

Mês	Estoque Inicial	Quantidade Comprada	Custo do Estágio
1	5000	15000	R$ 529337
2	0	10000	R$ 351200
3	0	15000	R$ 601200
4	0	10000	R$ 401200
		Custo Total:	R$ 1.882.937,50

Fonte: Cechin, 2019

5.4 Proposta de Algoritmo utilizando a técnica de programação dinâmica generalizada

Para solucionar o problema tratado nesse trabalho, foi desenvolvido e testado um código que atende às expectativas de redução de custos de dinheiro. Utilizando a programação dinâmica, é possível ver, em forma de algoritmo, como a solução para o escalonamento e compra de vacinas funciona. Sendo assim, segue-se o algoritmo:

Algoritmo 3 –Algoritmo para definição de escalonamento de vacinas usando programação dinâmica.

Algoritmo VACINAOTIMA()

```
1.  linha = n
2.  coluna = demandaTotal
3.  memoriza[linha+1][coluna+1]
4.  para mês = 0 enquanto mês ≤ linha faça
5.  |  para estoque = 0 enquanto estoque ≤ coluna faça
6.  |  |  memoriza[mês][estoque] = infinito
7.  |  fim
8.  fim
9.  para estoque = 0 enquanto estoque ≤ coluna faça
10. |  se estoque ≤ demanda[n – 1] então
11. |  |  paraComprar = demanda[n – 1] – estoque
12. |  |  memoriza[n][estoque] = custoDaCompra(n, paraComprar)
13. |  fim
14. fim
15. para mês = n -1 enquanto mês > 0 incrementando -1 faça
16. |  para estoque = 0 enquanto estoque ≤ coluna faça
```

17. | | **para** *paraComprar* = 0 **enquanto** *paraComprar* + *estoque* ≤ *SufixoSomaDemanda*[*mês* -1] **faça**
18. | | | *novoEstoque* = (*paraComprar* + *estoque*) - *demanda*[*mês* – 1]
19. | | | **se** *novoEstoque* > 0 **então continue**
20. | | | *custoParcial* = *custoDaCompra*(*mês*, *paraComprar*) + *custoDoArmazenamento*(*paraComprar* + *estoque*, *mês*)
21. | | | **se** *memoriza*[*mês*][*estoque*] > *custoParcial* + *memoriza*[*mês*+1][novoE*stoque*] **então**
22. | | | | *memoriza*[*mês*][*estoque*] = *custoParcial* + *memoriza*[*mês*+1][novoE*stoque*]
23. | | | **fim**
24. | | **fim**
25. | **fim**
26. **fim**
27. **devolva** *memoriza*[1][*estoque*]

No pseudocódigo do algoritmo trabalhado, temos que na linha 1 e 2 o valor de linhas da matriz será *n* e o valor de colunas será o tamanho da demanda total em lotes, mas para que a técnica de programação dinâmica funcione precisamos que a matriz possua uma linha e coluna extras como descrito na linha 3. Temos então que, da linha 4 a 8, há uma repetição para preencher a matriz com infinito em todos os espaços. Da linha 9 até a 14, há uma repetição que testa se o estoque é menor ou igual a demanda no mês atual na linha 10. Caso positivo, na linha 11 a quantidade para ser comprada será calculada subtraindo o estoque atual da demanda atual. Então, o custo da compra será calculado com uma função que aplica as variáveis requisitadas como preço atual da vacina, preço de entrega etc. Da linha 15 a 26 inicia-se efetivamente a seção onde será feito o cálculo principal. É possível notar como as 3 estruturas de repetição juntas tem uma complexidade, respectivamente, de *n*×*coluna*×*possíveis lotes a comprar*, podendo ser simplificada para $O(n^3)$. Na linha 18, o novo valor de estoque será calculado baseado no que será comprado somado ao estoque atual, subtraído da demanda. Caso seja maior que 0, uma variável de custo temporária irá receber o custo total do mês que se dá pelo custo da compra e armazenamento (ambos calculados por funções). E, por fim, na linha 21 é testado se o valor é ótimo, comparando o custo da posição atual da matriz com o custo temporário calculado somado ao custo do mês anterior calculado (levando em conta que o cálculo é feito com os meses em ordem invertida. Logo, o mês anterior calculado seria o mês seguinte cronologicamente). Na linha 21, é devolvida a resposta do cálculo que estará na posição [1][*estoque*] na matriz.

5.5 Subestrutura ótima alcançada no código

Como visto anteriormente no conceito de subestrutura ótima, é possível analisar a estrutura desse código e confirmar a existência de uma subestrutura ótima na resolução desse problema.

A solução do problema consiste em tomadas de decisão no início de cada mês e essa decisão produz um ou mais subproblemas a serem resolvidos. São esses as próximas tomadas de decisão acarretadas aos meses seguintes.

Foi suposto que para o dado problema exista uma solução ótima, como o intuito do trabalho, caso contrário não seria possível iniciar a pesquisa.

Se provado que o problema tem subestrutura ótima, então pode-se escrever o algoritmo e garante-se que ele está correto.

Ao determinar os subproblemas que decorrem a partir da solução ótima, foi possível caracterizar melhor o espaço de subproblemas resultante, sendo este o conjunto de meses seguintes aos quais serão definidas soluções, assumindo que todas as decisões anteriores foram ótimas.

Por último percebe-se que, dentro de uma solução ótima para o problema, todas as soluções de subproblemas usadas dentro deste são ótimas.

Logo, sendo k o número de lotes comprados e e o estágio atual, temos:

$$\text{CUSTO}(1,1..k\text{-}1,e)+\text{custo}(k,e')+\text{CUSTO}(e',k+1...\text{N},0) \qquad (3)$$

5.6 Resultados – Instância do Artigo Original utilizando a técnica de programação dinâmica

Utilizando a técnica de programação dinâmica, foi possível calcular os valores ótimos de compra previamente. Portanto, atingiu-se uma certa economia do custo final. Seguem os resultados abaixo:

Tabela 5 - Resultados da instância do Artigo Estudado utilizando a técnica de programação dinâmica

Mês	Estoque Inicial	Quantidade Comprada	Custo da Compra	Custo da Entrega	Custo de Armazenamento	Custo do Estágio
1	5000	15000	R$525.000	R$1.100	R$12.950	R$539.050
2	0	35000	R$1.225.000	R$1.200	R$22.662	R$1.248.862
3	25000	0	0	0	R$20.000	R$20.000
4	10000	0	0	0	R$8.000	R$8.000
					Custo Total:	R$1.807.912

Fonte: Elaborado pelo autor

Resultado: R$1807912,00

Compra ótima do mês 1:15000 unidades (3 lotes);

Compra ótima do mês 2:35000 unidades (7 lotes);

Compra ótima do mês 3:0;

Compra ótima do mês 4:0.

5.7 Comparação de resultados

Tendo em vista a redução do valor total no intervalo de 4 meses- de R$ 1.882.937,50 para R$ 1.807.912,00- houve uma economia de R$ 75.025,00. Esse feito acarretou em uma porcentagem de 4,14% de diferença no valor total final.

5.8 Simulação de Instância

Agora, utilizando dados criados como exemplo, pode-se criar outra instância a ser analisada. Utilizando: limite máximo de estoque de 100000 unidades, lotes novamente de 5000 unidades e um período de 5 meses; um estoque inicial de 15000 unidades e demandas mensais de 4, 2, 3, 2, 4 lotes (20000, 10000, 15000, 10000 e 20000 unidades), respectivamente; com o custo de entrega de R$ 1100,00 no primeiro mês, R$ 1200,00 nos meses 2 a 4 e R$ 1100,00 novamente no último mês; custo de armazenamento de 2,5% nos 3 primeiros meses e de 1,85% nos 2 meses restantes e custos das vacinas de R$ 40,00 nos 2 primeiros meses e de R$ 35,00 nos 2 meses restantes. Aqui, novamente, tem-se os resultados previstos nessa instância caso a técnica de programação dinâmica não fosse utilizada. Ao comprar os lotes apenas de acordo com a demanda, temos:

Tabela 6 - Resultados da instância simulada sem utilizar a técnica de programação dinâmica

Mês	Estoque Inicial	Quantidade Comprada	Custo do Estágio
1	5000	15000	R$ 821.100
2	0	10000	R$ 411.200
3	0	15000	R$ 616.200
4	0	10000	R$ 357.675
5	0	20000	R$ 714.050
		Custo Total:	R$ 2.920.225

Ao utilizar a técnica de programação dinâmica para calcular a compra ótima de cada mês, obteve-se os seguintes valores:

Tabela 7 - Resultados da instância simulada utilizando a técnica de programação dinâmica

Mês	Estoque Inicial	Quantidade Comprada	Custo da Compra	Custo da Entrega	Custo de Armazenamento	Custo do Estágio
1	15000	30000	R$1.201.100	R$1.100	R$45.000	R$1.246.100
2	25000	0	0	0	R$25.000	R$25.000
3	15000	0	0	0	R$15.000	R$15.000
4	0	30000	R$1.051.200	R$1.200	R$19.425	R$1.070.625
5	20000	0	0	0	R$12.950	R$12.950
					Custo Total:	R$2.827.825

Como resultado, houve uma redução do custo total final de R$ 2.920.225 para R$ 2.827.825,00 . Logo, representando uma redução de custos de 3,26%.

5.9 Discussão

Foi constatado que a técnica de programação dinâmica generalizada trabalhada no projeto está correta, pois ao possuir uma subestrutura ótima, é possível criar uma solução ótima para o problema. Portando, utilizando a solução ótima encontrada, pode-se prever o melhor escalonamento de compras para os meses planejados e, assim, reduzir os custos.

6 CONCLUSÃO

Foi abordada a generalização de uma técnica de programação dinâmica para o escalonamento da compra de vacinas, de tal forma que, devido os dados de entrada serem configuráveis, o próprio usuário pode utilizar do algoritmo para efetuar os cálculos e conseguir uma economia de gastos no estabelecimento.

Foi mostrada como a distribuição de vacinas funciona na rede do frio e a importância do armazenamento em temperatura adequada. Essas questões, junto ao custo da vacina e ao custo da entrega, são fatores importantes no cálculo de valores da situação problema.

Foi constatada a eficácia da técnica de programação dinâmica e a sua superioridade sobre a técnica de backtracking nessa situação, através do exemplo do problema da mochila booleana - o qual se apresenta muito semelhante à solução necessária do problema tratado no projeto.

Por fim, foi demonstrado o algoritmo para a solução do problema em forma de pseudo-código, com a sua explicação detalhada e com a sua complexidade. Utilizando esses recursos, possibilitou-se a simulação das instâncias de situações utilizando o código e, dessa forma, demonstrando os resultados obtidos com o experimento.

7 REFERÊNCIAS

ALMEIDA, L. Monitoramento e armazenamento de vacinas: diretrizes e métodos adequados. **Nexxto**. 27 maio 2020. Disponível em: <https://nexxto.com/monitoramento-e-armazenamento-de-vacinas-diretrizes-e-metodos-adequados/>. Acesso em: 17 nov. 2021.

ALMEIDA, L. Cadeia do Frio: logística e controle de temperatura. **Nexxto**. 21 novembro 2019. Disponível em: <https://nexxto.com/cadeia-do-frio-logistica-e-controle-de-temperatura//>. Acesso em: 17 nov. 2021.

BRASIL. **Ministério da Saúde**. Secretaria de Políticas de Saúde. Manual de rede de frio do Programa Nacional de Imunizações. Brasília, DF, 2017.

BRASIL. **Ministério da Saúde**. Secretaria de Políticas de Saúde. Do laboratório ao braço: caminho das vacinas Covid-19 tem controle de qualidade rígido e operação logística em tempo recorde. 20 ago. 2021. Disponível em: < https://www.gov.br/saude/pt-br/assuntos/noticias/do-laboratorio-ao-braco-caminho-das-vacinas-covid-19-tem-controle-de-qualidade-rigido-e-operacao-logistica-em-tempo-recorde>. Acesso em: 24 nov. 2021.

CECHIN, R.B.; BIASUZ, R.; FALAVIGNA, A. CORSO, L.L. Programação dinâmica aplicada à redução de custos nas compras de vacinas de um hospital. **Portal de Revistas da USP**. 22 dez. 2019. Disponível em: <https://www.revistas.usp.br/rmrp/article/view/155752/158298>. Acesso em: 24 nov. 2021.

CORMEN, T.H. et al. Algoritmos**: Teoria e Prática**. Tradução de Arlete Marques. . **ed. Rio de Janeiro: Elsevier**, 2009.

FEOFILOFF, P. Mochila booleana. **Instituto de Matemática e Estatística da USP**. 21 nov. 2020. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html>. Acesso em: 4 dez. 2021.

KAFKES, A. Demystifying Dynamic Programming. **FreeCodeCamp**. 31 jul. 2017. Disponível em: <https://www.freecodecamp.org/news/demystifying-dynamic-programming-3efafb8d4296/>. Acesso em: 24 nov. 2021.

KREHER, D. L.; STINSON, D. R. Combinatorial algorithms: generation, enumeration, and search. **Boca Raton, Florida: CRC Press**, 1999. (Discrete Mathematics and Its Applications). ISBN 978-0849339882.

MARQUES, F.P. O Problema da Mochila Compartimentada. 2000. 10.11606/d.55.2000.tde-14012005-165350. Dissertação (Mestrado) - **USP, São Carlos**.

MINARDI, R. Programação dinâmica. **Online Bioinfo**. 4 set. 2019. Disponível em: <https://homepages.dcc.ufmg.br/~raquelcm/onlinebioinfo/index.php?alias=glossario_programacao_dinamica>. Acesso em: 5 out. 2021.

SKIENA, S. S.; REVILLA, M. A. Programming challenges: The programming contest training manual. **New York: Springer Science & Business Media**, 2006.

TOMMASINI, S **Programação Dinâmica**. 2014. 7278080. TCC - USP, São Paulo

UNIVERSIDADE DE OXFORD. **Our World in Data**. 2021. Disponível em: <https://ourworldindata.org/covid-vaccinations?country=BRA>. Acesso em: 17 nov. 2021.