CENTRO UNIVERSITÁRIO SERRA DOS ÓRGÃOS – UNIFESO
CENTRO DE CIÊNCIAS E TECNOLOGIA – CCT
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TÉCNICAS DE OTIMIZAÇÃO DE PROGRAMAÇÃO DINÂMICA

GABRIEL LAGOA DUARTE

Teresópolis
2017

CENTRO UNIVERSITÁRIO SERRA DOS ÓRGÃOS – UNIFESO
CENTRO DE CIÊNCIAS E TECNOLOGIA – CCT
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TÉCNICAS DE OTIMIZAÇÃO DE PROGRAMAÇÃO DINÂMICA

GABRIEL LAGOA DUARTE

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Serra dos Órgãos – UNIFESO como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Rafael Gomes Monteiro

Teresópolis
2017

CENTRO UNIVERSITÁRIO SERRA DOS ÓRGÃOS – UNIFESO
CENTRO DE CIÊNCIAS E TECNOLOGIA – CCT
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TÉCNICAS DE OTIMIZAÇÃO DE PROGRAMAÇÃO DINÂMICA

GABRIEL LAGOA DUARTE

Trabalho de Conclusão de Curso aprovado como requisito parcial para obtenção do título de Bacharel no Centro Universitário Serra dos Órgãos – UNIFESO pela banca examinadora:

_____

Nome do(a) Professor(a) Orientador(a)
(Presidente de Banca) por extenso - titulação abreviada

_____

Nome do(a) Convidado(a) por extenso - titulação abreviada

_____

Nome do(a) Convidado(a) por extenso - titulação abreviada

Teresópolis

dia de mês da defesa de 2017

Dedico esta monografia a minha família,
pelo apoio fornecido e aos meus amigos.

AGRADECIMENTOS

Texto dos agradecimentos.

Epígrafe

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

LISTA DE SIGLAS

ICT	Information and Communication Technology
IDE	Integrated Development Environment
LIS	Longest Increasing Subsequence

LISTA DE ALGORITMOS

RESUMO

Escrever um texto que contemple todo o conteúdo do trabalho, com espaçamento 1,5, justificado. Conforme as normas NBR 14724:2011 e NBR 6028:2003,da ABNT, o resumo é elemento obrigatório, constituído de parágrafo único; uma seqüência de frases concisas e objetivas e não de uma simples enumeração de tópicos, não ultrapassando 500 palavras, O resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. Deve-se usar o verbo na voz ativa e na terceira pessoa do singular. Devem ser seguido, logo abaixo, das palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores, que são palavras principais do texto, sendo de 3 a 5, separadas por ponto)

Palavras-chave: Palavra-chave 1. Palavra-chave 2. ...

ABSTRACT

Abstract text (maximum of 500 words).

Keywords: Keyword 1. Keyword 2. ...

SUMÁRIO

1 INTRODUÇÃO

Motivação: - Uma das áreas mais importantes da maratona de programação - Está em alta nas últimas competições

Justificativa: - Poucos trabalhos na área - Livros não abordam esse assunto

Objetivos - Gerar um trabalho que explique as principais técnicas de otimização utilizadas

Alterações necessárias no modelo LaTeX

- Adicionar uma lista de equações
- Tentar adicionar links no sumário
- Quadro != Tabela

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os conceitos que servem de insumo para a elaboração das etapas seguintes desse trabalho. Na seção 2.1 são demostrados alguns conceitos necessários para identificar e classificar a complexidade de um algoritmo. A seção 2.2 explica o básico sobre programação dinâmica, seus principais termos e o conceito de otimização, que servirá de base para a elaboração dos capítulos seguintes. Por fim, no capítulo 2.3 é feita uma análise de alguns trabalhos relacionados com o ensino de programação.

2.1 COMPLEXIDADE DE ALGORITMOS

Na ciência da computação, analisar um algoritmo está relacionado com a identificação da quantidade de recursos necessários para sua execução, podendo ser a quantidade de memória utilizada, largura de banda de comunicação, hardware do computador. Porém mais frequentemente a preocupação maior é em se medir o tempo computacional gasto para realizar determinado código (CORMEN *et al.*, 2009).

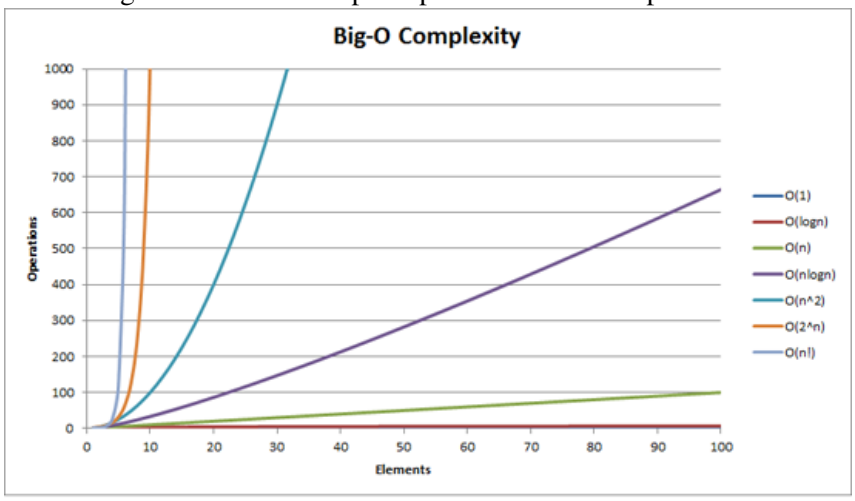
Quando é feita a análise de complexidade, é possível identificar qual classe um determinado algoritmo pertence. A tabela 1 lista algumas classes de forma ordenada, sendo a primeira função a melhor possível e a última a pior. Após classificar um algoritmo, é possível escolher entre diversas soluções para um mesmo problema, qual é a mais adequada no momento e poder saber antes de executar quanto tempo e memória o algoritmo irá gastar quando o tamanho da entrada for n, onde n corresponde, geralmente a quantidade de elementos que devem processados. Na figura 1 é possível observar o comportamento de algumas funções na medida que a quantidade de elementos aumenta.

Tabela 1: Principais classes de funções para analisar algoritmos

Notação	**Exemplo de algoritmos**
$O(1)$	Determinar se um número é par ou ímpar
$O(logn)$	Busca binária
$O(\sqrt{n})$	Determinar se um número é primo
$O(n)$	Procurar um elemento em um *array* não ordenado
$O(n*logn)$	*Merge sort*[1]
$O(n^2)$	*Bubble sort*[2]
$O(n^3)$	*Floyd-Warshall*[3]
$O(n^c)$	Encontrar o maior emparelhamento em um grafo
$O(c^n)_{c>1}$	Resolver o problema do caixeiro viajante[4]com programação dinâmica
$O(n!)$	Resolver o problema do caixeiro viajante com força bruta

Fonte: Próprio Autor.

Figura 1: Gráfico das principais classes de complexidade



Fonte: PERRETT, 2010

2.2 PROGRAMAÇÃO DINÂMICA

Programação dinâmica é uma técnica que combina soluções de subproblemas, da mesma maneira que a divisão e conquista, que divide o problema em subproblemas, resolve cada

[1]Ver mais em: http://quiz.geeksforgeeks.org/merge-sort/
[2]Ver mais em: http://quiz.geeksforgeeks.org/bubble-sort/
[3]Ver mais em: http://www.geeksforgeeks.org/dynamic-programming-set-16-floyd-warshall-algorithm/
[4]Ver mais em: http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/

um recursivamente e depois é feita a junção das soluções para resolver o problema original. Porém este método é normalmente utilizado quando os subproblemas se sobrepõem, ou seja, um mesmo estado é encontrado diversas vezes na etapa de divisão. Portanto, se fosse aplicado um algoritmo ingênuo de divisão e conquista, um mesmo estado seria resolvido várias vezes, aumentando o custo computacional do algoritmo (CORMEN *et al.*, 2009).

Para resolver o problema de sobreposição, a técnica de programação dinâmica salva a resposta de todos os estados que vão sendo encontrados. Assim, no momento que se deparar com algo que já foi resolvido ela simplesmente retorna o valor que já estava armazenado.

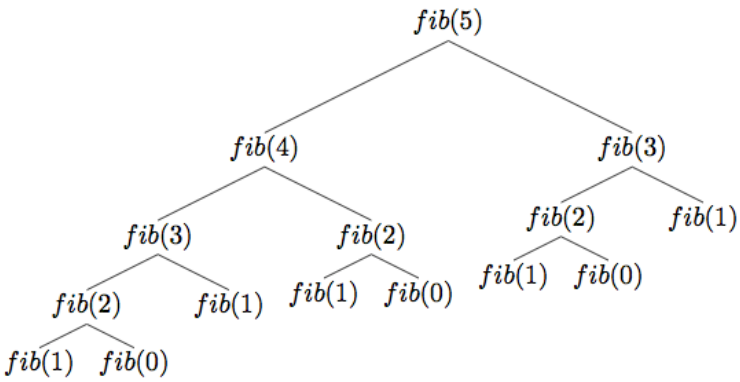
A sequência de Fibonacci é um exemplo de fácil entendimento de quando é necessário a utilização de programação dinâmica. Esta é uma sequência de números inteiros que tem seu início com 0 e 1, os termos subsequentes são uma soma dos dois últimos números. A sequência recebeu o nome do matemático Leonardo de Pisa, mais conhecido como Fibonacci, que no ano de 1202 descreveu o crescimento da população de coelhos utilizando esta sequência (HOM, 2013). Os primeiros termos são:

$$0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,... \tag{1}$$

podendo ser representada através da seguinte recorrência, onde $fib(i)$ representa i-ésimo termo da sequência:

$$fib(i) = \begin{cases} i & \text{se } i \leq 1, \\ fib(i-1) + fib(i-2) & \text{se } i > 1. \end{cases} \tag{2}$$

Figura 2: Árvore de recursão do Fibonacci de 5



Fonte: SCHWARTZ, 2011

Na figura 2 é apresentada a árvore de recursão gerada ao utilizar a equação 2 para o

cálculo do $fib(5)$. Através dela, é fácil ver que diversos estados estão se repetindo, por exemplo: $fib(2)$ aparece três vezes e sempre que é encontrado ele é divido no $fib(1)$ e $fib(0)$, assim deixando a complexidade deste algoritmo em $O(2^N)$. Ao aplicar programação dinâmica neste algoritmo é possível reduzir a complexidade para $O(N)$, pois cada estado será expandido uma única vez.

O código a seguir mostra como seria a implementação da função sem a utilização de programação dinâmica.

Algoritmo 1: Implementação Fibonacci sem programação dinâmica

```
1  int fib(int i){
2      if(i <= 1)
3          return i;
4      return fib(i - 1) + fib(i - 2);
5  }
```

Para otimizar o código e utilizar programação dinâmica basta incluir uma tabela que salva todos os estados. Sua inclusão faz uma alteração mínima no código, como é mostrado no algoritmo 2.

Algoritmo 2: Implementação Fibonacci com programação dinâmica

```
1  #define MAX 20
2  int tabela[MAX + 1];
3
4  int fib(int i){
5      if(tabela[i] != -1)
6          return tabela[i];
7      if(i <= 1)
8          return tabela[i] = i;
9      return tabela[i] = fib(i - 1) + fib(i - 2);
10 }
```

Para mais informações sobre programação dinâmica e suas técnicas, o site *TopCoder*[5] possui um artigo amplo com vários problemas e dicas para soluções. Ele divide sua explicação em teoria e prática, começando nos tópicos mais simples e indo até alguns mais avançados.

2.2.1 Otimizações

Ao utilizar programação dinâmica para otimizar um problema, normalmente ocorre uma queda drástica na classe de complexidade associada a solução, como é o caso da sequência de Fibonacci, discutida na seção 2.2, onde foi possível sair de uma complexidade exponencial para

[5]https://www.topcoder.com/community/data-science/data-science-tutorials/

uma linear. Apesar de parecer uma ótima forma de solucionar um problema, às vezes apenas aplicar programação dinâmica não é suficiente, e existem casos onde é possível e necessário otimizar ainda mais.

Para utilizar uma técnica de otimização de programação dinâmica, alguns critérios em relação a função de recorrência devem ser correspondidos. Cada técnica tem uma abordagem que possibilita a resolução de um conjunto de problemas que compartilham certas propriedades.

A tabela 2 ilustra algumas formas otimizações existentes, que possibilitam tanto na redução de espaço, quanto de tempo. Estas técnicas serão discutidas no capítulo 5.

Tabela 2: Otimizações de programação dinâmica

Nome/Tipo	**Característica**
Redução de espaço	Reduz a quantidade de memória necessária quando um estado depende de uma quantidade fixa de outros estados
Estrutura de dados	Reduz a complexidade de tempo com o auxílio de uma estrutura de dados que consegue responder a consultas do tipo mínimo ou máximo em um intervalo de uma *array*
Divide and Conquer	Realiza divisão e conquista para encontrar o ponto ótimo necessário para se resolver o estado atual, reduzindo a complexidade temporal
Knuth Optimization	Utiliza informações de onde estava a solução ótima de um estado anterior para diminuir o espaço de busca dos outros, assim reduzindo a complexidade temporal
Convex Hull Trick	Através de conceitos geométricos, essa técnica possibilita a redução da complexidade temporal

Fonte: Próprio Autor.

2.3 ENSINO DE ALGORITMOS

O ensino de algoritmos, por se tratar de assuntos que normalmente não são tão simples, com um fundamentação matemática extensa, não é uma tarefa fácil, portanto foi feita uma busca na literatura de trabalhos que têm por objetivo criar uma metodologia de ensino. Assim servindo de modelo para o que está aqui sendo desenvolvido.

Szlávi e Zsakó (2003) apresentaram diversas metodologias relacionadas ao ensino de programação. O autor discuti sobre alguns modelos e exemplifica quando e para qual nível de estudante cada método será mais proveitoso ao ser aplicado. Estes métodos determinam a forma

de estruturar o curso que deseja ser ensinado e a maneira de explicar os conteúdos.

Uma ampla pesquisa na literatura com o foco na parte educacional do estudo de programação foi feita. Diversos métodos e tópicos foram identificados e analisados para poder ser realizado uma classificação, e assim, auxiliar os professores a identificar em seus alunos características comuns e padrões, que poderão ser contornados com base no que já foi realizado e está documentado na literatura, é mostrado por Robins, Rountree (2003).

Pears *et al.* (2007) desenvolveu um *survey* que reúne algumas formas da literatura de ensinar a introdução de programação. Além disso, os trabalhos reunidos, foram classificados e agrupados pela forma de ensino e pelos métodos aplicados.

Zsakó e Nóra (2008) realizaram uma análise nos principais métodos e aplicações que auxiliam no aprendizado e no ensino dos tópicos de ICT (do inglês, *Information and Communication Technology*). Para cada método é exemplificado o seu funcionamento, como realizar a sua aplicação e para qual nível de estudante ele é mais apropriado.

No trabalho desenvolvido por Papp-varga, Szlávi e Zsakó (2008), foi feita uma análise semelhante a que foi realizada no trabalho apresentado na subseção anterior. Porém, o foco deste é o ensino de uma linguagem de programação, portanto os métodos apresentados demostram os passos ideais para transmitir os conceitos da linguagem proposta.

Radošević, Orehovački e Lovrenčić (2009), se propõem em criar uma ferramenta que facilite o aprendizado de linguagens de programação básicas, como C++, ao invés da utilização de IDE (do inglês, *Integrated Development Environment*). A finalidade desta ferramenta é ajudar os estudantes em não cometer erros que são comuns a quem está iniciando. Além disso, oferecer uma forma mais simples do professor auxiliar seus alunos.

Vihavainen, Paksula e Luukkainen (2011), discutem como ensinar o básico de programação para quem está começando. O autor propõe um modelo de ensino e faz a aplicação deste em uma turma de um curso de ciência da computação. Ao final do trabalho é feito um comparativo entre uma turma que utilizou da metodologia proposta e uma que não usou, e os resultados foram positivos, mostrando que a quantidade de evasão e reprovação foram reduzidos.

Apesar de existirem diversos trabalhos que tratam sobre o ensino de algoritmos e programação, todos os encontrados têm um foco para um nível básico de conhecimento, levando em conta alunos que estão iniciando nesta área, portanto, não será aplicado nenhum deste diretamente. No capítulo 4 será discutido a metodologia elaborada para a construção do trabalho.

3 HISTÓRICO E TRABALHOS RELACIONADOS

Capítulo destino a fazer o levanto sobre programação dinâmica, desde seu surgimento até os dias atuais, mostrando os trabalhos na área e onde tem sido utilizada.

Citar os trabalhos, dizendo o que cada um fez.

O "porém": nada focado nas otimizações X, Y e Z - onde é usado além das maratonas

3.1 TRABALHOS RELACIONADOS

Nesta seção são apresentados alguns trabalhos que tem um objetivo similar ao deste projeto. Em todos os encontrados a intenção do autor foi criar um conteúdo teórico sobre algoritmos e programação. Apesar de apenas um trabalho possuir conteúdo relacionado a programação dinâmica, todos eles são úteis para a elaboração da metologia a ser desenvolvida.

Tommasini (201-) em seu trabalho visa ensinar programação dinâmica para quem está iniciando nesta área. Sua maior motivação foi a falta de um bom material didático sobre esse tema. Seu trabalho tem um foco muito didático, apresentando diversas técnicas, problemas com soluções e propondo vários exercícios para o leitor praticar o que foi ensinado. Ao final do trabalho é apresentado uma lista de problemas de maratonas de programação sobre os assuntos desenvolvidos no texto.

No trabalho proposto por Dalalio (2013), foi realizado um estudo de algoritmos e estruturas de dados para a resolução de problemas relacionados a *String Matching*[1]. Seu texto é bem didático, mostrando diversos problemas e as diversas formas de resolução, apresentando a complexidade e seu código. Ao final diversos problemas são propostos para que o leitor os resolva utilizando os conceitos elaborados no texto.

Couto (2016) desenvolveu um trabalho voltado ao ensino de algoritmos, seu foco principal são os algoritmos aplicados em sequências de caracteres. Seu texto ficou dividido entre diversos capítulos, onde cada um deles apresentava uma estrutura de dados diferente, assim podendo mostrar ao leitor quando é melhor utilizar uma em relação as outras.

[1] https://en.wikipedia.org/wiki/String_searching_algorithm

4 METODOLOGIA

Explicar a linguagem que será utilizada (tanto do texto corrido, quanto a linguagem de programação, pseudocódigo, etc)

5 DESENVOLVIMENTO (ALTERAR NOME)

Aqui entra o desenvolvimento real do trabalho com as principais técnicas de otimização

5.1 REDUÇÃO DE ESPAÇO

Dentre as otimizações disponíveis, as que envolvem redução de memória são as mais simples de serem aplicadas, seu uso pode ser facilmente entendido no problema da mochila [1]. Esse problema deseja maximar o valor dos itens colocados em uma mochila, onde estes possuem um valor e um peso associado, enquanto a mochila possui uma capacidade máxima de peso. Além disso, nenhum item pode ser dividido.

$$dp[i][j] = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0, \\ max(valor[i-1] + dp[i-1][j - peso[i-1]], dp[i-1][j]) & \text{se } peso[i-1] \leq j, \\ dp[i-1][j] & \text{se } peso[i-1] > j \end{cases} \tag{1}$$

O problema da mochila pode ser resolvido através da relação de recorrência apresentada acima, onde $dp[i][j]$ representa o valor máximo que pode se conseguir ao colocar os i-ésimos primeiros itens em uma mochila de capacidade j. Os vetores $valor$ e $peso$, representam o valor e peso associado a cada um dos n itens, respectivamente. A resposta para o problema estará em $dp[n][capacidade]$.

Analisando a complexidade da equação 1 é fácil ver que será necessário $O(n * capacidade)$, tanto de memória, quanto de tempo. Porém, é notório que para solucionar a linha i da matriz de programação dinâmica, só são necessárias as respostas que já foram calculadas na linha $i-1$, portanto podemos trabalhar apenas com duas linhas consecutivas da matriz, sempre alternando entre linha par e ímpar.

$$dp[i\&1][j] = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0, \\ max(valor[i-1] + dp[\sim i\&1][j - peso[i-1]], dp[\sim i\&1][j]) & \text{se } peso[i-1] \leq j, \\ dp[\sim i\&1][j] & \text{se } peso[i-1] > j \end{cases} \tag{2}$$

A equação 2 demostra como reduzir a memória. Os valores que serão utilizados nas linhas da DP serão sempre 0 ou 1, assim o total de memória necessária é de $2 * capacidade$, deixando com uma complexidade de $O(capacidade)$. A resposta para o problema da mochila

[1] http://www.geeksforgeeks.org/dynamic-programming-set-10-0-1-knapsack-problem/

utilizando esta relação estará em $dp[n\&1][capacidade]$.

O seguinte código mostra a implementação do problema da mochila com memória linear.

Algoritmo 1: Implementação Mochila

```
1
2  int mochila(){
3      int valor[] = {60, 100, 120};
4      int peso[]  = {10, 20, 30};
5      int capacidade = 50, n = 3;
6
7      int dp[2][capacidade + 1];
8      for(int i = 0; i <= n; i++){
9          for(int j = 0; j <= capacidade; j++){
10             if(!i || !j)
11                 dp[i&1][j] = 0;
12             else if(peso[i - 1] <= j)
13                 dp[i&1][j] = max(valor[i-1] + dp[~i&1][j-peso[i-1]],
14                                  dp[~i&1][j]);
15             else
16                 dp[i&1][j] = dp[~si&1][j];
17         }
18     }
19     return dp[n&1][capacidade];
20 }
```

5.2 ESTRUTURA DE DADOS RMQ

Um dos problemas clássicos de programação dinâmica é o LIS (do inglês, *Longest Increasing Subsequence*). Neste, o objetivo é encontrar a maior subsequência de um *array* onde todos os elementos estão ordenados de forma crescente.

Imaginemos o seguinte *array*:

$$2, 5, 3, 7, 11, 8$$

Uma das solução para este conjunto é a subsequência $2, 3, 7, 8$, que possui tamanho quatro.

Para resolver este problema pode ser usado programação dinâmica com a seguinte recorrência:

$$dp[i] = \begin{cases} 1 & \text{se } i = 0 \\ max(dp[j]+1)_{0 \leq j < i} & \text{se } i \neq 0 \text{ e } v[j] \leq v[i] \end{cases} \quad (3)$$

A equação 3, resolve o problema do LIS, com a complexidade $O(n^2)$ de tempo e $O(n)$ de memória, porém é notório que quando está sendo calculado a $dp[i]$, ou seja, a maior LIS que termina no i-ésimo índice, está sendo percorrido todos elementos do *array* que estão a esquerda e que possuem um valor menor ou igual que o elemento atual. Em outras palavras, queremos o maior elemento da dp que seu valor está no intervalo $[0..v[i]]$.

A partir destas observações podemos melhorar a complexidade dessa solução com o auxílio de uma estrutura de dados que consegue consultar máximo ou mínimo de intervalo de um *array* de forma mais eficiente. Pode ser ela uma *SegmentTree*[2] ou *BIT*[3], com isso é possível remover a busca linear no *array* pelo maior elemento e trocar por uma busca logarítmica, deixando a solução final com $O(n * logn)$.

O código a seguir mostra uma maneira de implementar o LIS utilizando *BIT*.

Algoritmo 2: Implementação LIS

```
1
2  #define MAXN 20
3  int v[] = {2, 5, 3, 7, 11, 8};
4  int bit[MAXN];
5  int dp[MAXN];
6
7  void update(int x, int v){
8      for(; x < MAXN; x+=x&-x)
9          bit[x] = max(bit[x], v);
10 }
11
12 int get(int x){
13     int ans = 0;
14     for(; x; x-=x&-x)
15         ans = max(ans, bit[x]);
16     return ans;
17 }
18
19 int lis(int n){
20     int ans = 0;
21
22     for(int i = 0; i < n; i++){
```

[2]http://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/
[3]http://www.geeksforgeeks.org/binary-indexed-tree-or-fenwick-tree-2/

```
23          dp[i] = 1+get(v[i]);
24          update(v[i], dp[i]);
25          ans = max(ans, dp[i]);
26      }
27
28      return ans;
29  }
```

5.3 DIVIDE AND CONQUER OPTIMIZATION

Problemas:

- Kattis Branch Assignment WF 2016

5.4 KNUTH OPTIMIZATION

Problemas:

- Codechef = CHEFAOR
- URI 2475

5.5 CONVEX HULL TRICK

Problemas:

- URI 2481

6 CONSIDERAÇÕES FINAIS

Considerações finais do trabalho, com conclusão e trabalhos futuros

6.1 CONCLUSÕES

6.2 TRABALHOS FUTUROS

REFERÊNCIAS

CORMEN, T. H. *et al.* **Introduction to Algorithms**. Favoritenstrasse 9/4th Floor/1863: The MIT Press, 2009.

HOM, E. J. **What is the Fibonacci Sequence?** 2013. Disponível em: <http://www.livescience.com/37470-fibonacci-sequence.html>. Acesso em: 02 abr. 2017.

PAPP-VARGA, Z.; SZLáVI, P.; ZSAKó, L. Ict teaching methods – programming languages. In: **Annales Mathematicae et Informaticae 35**. [s.n.], 2008. p. 163–172. Disponível em: <https://www.researchgate.net/publication/228955901>.

PEARS, A. *et al.* A survey of literature on the teaching of introductory programming. In: **Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education**. New York, NY, USA: ACM, 2007. (ITiCSE-WGR '07), p. 204–223. Disponível em: <http://doi.acm.org/10.1145/1345443.1345441>.

PERRETT, D. **CompSci 101 - Big-O Notation**. 2010. Disponível em: <http://www.daveperrett.com/articles/2010/12/07/comp-sci-101-big-o-notation/>. Acesso em: 23 abr. 2017.

RADOšEVIć, D.; OREHOVAčKI, T.; LOVRENčIć, A. New approaches and tools in teaching programming. 2009. Disponível em: <https://www.researchgate.net/publication/224930648>.

ROBINS, A.; ROUNTREE, J.; ROUNTREE, N. Learning and teaching programming: A review and discussion. **Computer Science Education**, v. 13, n. 2, p. 137–172, 2003. Disponível em: <http://www.tandfonline.com/doi/abs/10.1076/csed.13.2.137.14200>.

SCHWARTZ, H. R. **Memoization using Closures**. 2011. Disponível em: <https://harryrschwartz.com/2011/01/06/memoization-using-closures.html>. Acesso em: 02 abr. 2017.

SZLáVI, P.; ZSAKó, L. Methods of teaching programming. 2003. Disponível em: <https://www.researchgate.net/publication/235925815>.

VIHAVAINEN, A.; PAKSULA, M.; LUUKKAINEN, M. Extreme apprenticeship method in teaching programming for beginners. In: **Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education**. New York, NY, USA: ACM, 2011. (SIGCSE '11), p. 93–98. ISBN 978-1-4503-0500-6. Disponível em: <http://doi.acm.org/10.1145/1953163.1953196>.

ZSAKó, L.; NóRA, C. Ict teaching methods applications. In: **Mittermeir R., Syslo M. (eds.) Informatics Education contributing across the curriculum, proc. of selected papers - 3rd ISSEP conference**. [s.n.], 2008. p. 47–53. Disponível em: <https://www.researchgate.net/publication/235925718>.

FUNDAÇÃO EDUCACIONAL SERRA DOS ÓRGÃOS – FESO
CENTRO UNIVERSITÁRIO SERRA DOS ÓRGÃOS – UNIFESO
CENTRO DE CIÊNCIAS E TECNOLOGIA – CCT
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DECLARAÇÃO DE PRÓPRIA AUTORIA

Teresópolis, XX/XX/2017

Eu, Gabriel Lagoa Duarte, declaro para fins de conclusão do Curso de Bacharelado em Ciência da Computação do UNIFESO, que este Trabalho de Conclusão de Curso é de minha própria autoria, estando ciente das consequências disciplinares a que estarei sujeito caso seja comprovada fraude ou má-fé. Sem mais, subscrevo-me,

Atenciosamente,

_____

Gabriel Lagoa Duarte