UNIVERSIDADE FEDERAL DO TOCANTINS CAMPUS DE PALMAS
Curso: Bacharelado em Ciência da Computação
Disciplina: Processo e Análise de Algoritmos **Semestre:** 2022.1
Professor: Warley Gramacho **Alunos:** Carlos Eduardo Viana Lima e Neudison Nonato Maia Filho

Relatório Greedy Technique

1. Introdução:

Greedy Technique ou Algoritmo Guloso é um paradigma algorítmico que constrói uma solução passo por passo, sempre escolhendo o próximo passo que oferece o benefício mais óbvio e imediato. Portanto, os problemas em que a escolha localmente ótima também leva à solução global são mais adequados para o Greedy.

Para resolver um problema, um algoritmo guloso escolhe, em cada iteração, o objeto mais apetitoso que vê pela frente. Um algoritmo guloso é míope: ele toma decisões com base nas informações disponíveis na iteração corrente, sem olhar às consequências que essas decisões terão no futuro. Um algoritmo guloso jamais se arrepende ou volta atrás: as escolhas que faz em cada iteração são definitivas.

Greedy não é brute-force: Enquanto o Greedy seleciona a cada passo a melhor opção para aquele passo, no Brute-force o algoritmo seleciona uma opção de uma maneira mais simples, óbvia ou direta. E que repete essa tentativa até encontrar o resultado esperado.

Neste relatório ficaram marcados alguns exemplos de implementações acerca do Greedy Technique e seus resultados.

2. Huffman Coding:

Teremos como exemplo o uso de uma palavra: BCCABBDDAECCBBAEDDCC

Usando a Tabela ASCII é necessário atribuir 8 bits para cada letra, logo, realizando uma conta simples temos que: 8*20=160 bits. O tamanho dessa palavra seria de 160 bits, contudo, não precisamos ficar presos a Tabela ASCII, podemos criar nosso próprio código e assim reduzir o consumo de espaço.

Primeiramente vamos analisar qual a frequência das letras nessa palavra. Com isso em mente, agora precisamos definir quantos algarismos mínimos serão necessários para representar cada letra. Usando 3 bits podemos representar 27 letras diferentes, sendo assim temos a seguinte tabela:

A	3	000
B	5	001
C	6	010
D	4	011
E	2	100

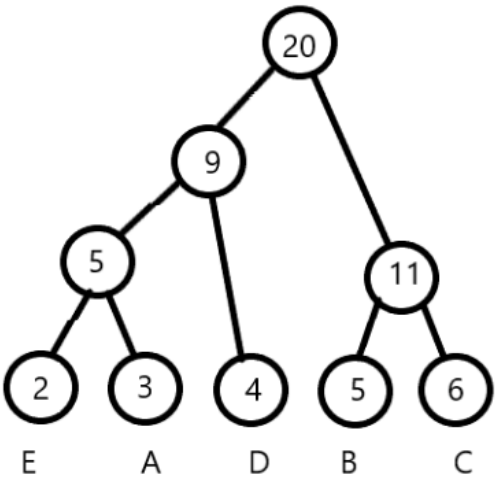
Agora temos 3 bits para cada letra, sendo assim: 3*20 = 60 bits. Entretanto, encontramos um problema nessa questão. Como a palavra está encriptografada é necessário passarmos também a tabela para que nosso destinatário possa entender a mensagem, e isso é feito da seguinte maneira:

Cada letra tem 8 bits então: 8*5= 40
Cada letra tem 1 código de 3 bits, logo: 3*5= 15

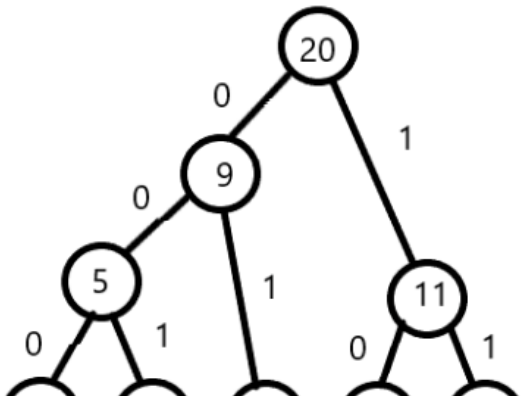
Total: 55 bits para a mesa de tradução.

Então, todo o código em si possui: 55+60 = 115 bits.

Porém, para o Huffman isso pode ser feito de outra forma ainda melhor! Ao invés de usarmos uma tabela, iremos usar uma árvore.



Os nós folhas representam as frequências de cada letra, seus pais são as combinações dos dois números de menor valor, com isso podemos atribuir a seguinte ordem de caminhos, esses que nos darão a nova ordem de códigos para cada letra dessa palavra:



A	3	001
B	5	10
C	6	11
D	4	01
E	2	000

Calculando a quantidades de bits temos:

A = 3*3 = 9
B = 5*2 = 10
C = 6*2 = 12
D = 4*2 = 8
E = 2*3 = 6

Total = 45 bits!

Entretanto, ainda devemos repassar o decodificador, seguindo a mesma lógica:

Cada letra tem 8 bit = 8*5 = 40 bits
Nosso novo código é vinculado as letras com 12 bits

No total temos: 52 bits

Logo 45+52= 97 bits!

De 160 para 97 bits, de fato uma redução estrondosa.

3. Prism and Kruskal:

A ideia por trás do algoritmo de Prim é simples, como uma árvore geradora significa que todos os vértices devem estar conectados, portanto, os dois subconjuntos disjuntos de

vértices devem ser conectados para formar uma a árvore. Eles devem ser conectados com a borda de peso mínimo para torná-lo uma árvore de abrangência mínima. O Kruskal vem com a mesma proposta, porém com uma alteração que será apresentada a seguir.

Prism:

- Necessita que os pontos já estejam ligados, e não necessita ir apenas pelos novos pontos ligados.
- Não funciona em grafos não conectados!

Kruskal:

- Ele sempre vai selecionar a aresta de menor valor, ao menos que a aresta de valor mínimo crie um ciclo.

3.1. Exemplos:

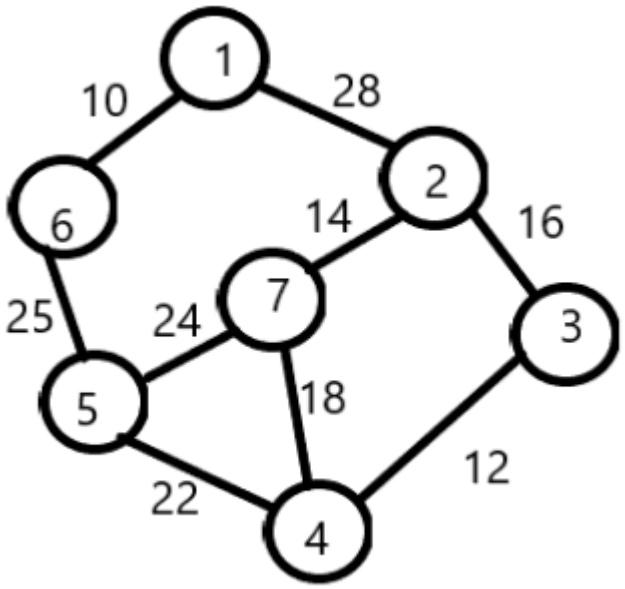


Imagem 3

Temos o determinado grafo acima cada caminho tem um peso, um valor, o Prism irá seguir os caminhos de menor custo que estejam ligados aos vértices, nesse caso:

1, 6, 5, 4, 3, 2, 7

Entretanto, caso tenhamos um grafo cuja suas arestas não estejam todas ligadas vamos ter o problema de que o Prism não irá percorrer por todo o grafo da devida maneira.

Já o Kruskal irá pegar as arestas de menor valor, não importando se estão conectadas ou não:

(1,6), (3,4), (2,7), (2,3)... A próxima aresta deveria ser a (7,4) entretanto isso iria ocasionar em um ciclo, nesse caso a aresta é ignorada e continuamos normalmente;

(1,6), (3,4), (2,7), (2,3), (5,4), (6,5).

4. Dijkstra:

Dado um grafo e um vértice fonte, ele encontra os caminhos mais curtos da fonte para todos os vértices no grafo dado. O algoritmo de Dijkstra é muito semelhante ao algoritmo de Prim para a árvore geradora mínima. Mantemos dois conjuntos, um conjunto contém vértices incluídos na árvore do caminho mais curto, outro conjunto inclui vértices ainda não incluídos na árvore do caminho mais curto. A cada passo do algoritmo, encontramos um vértice que está no outro conjunto (conjunto de ainda não incluído) e tem uma distância mínima da fonte.

4.1. Exemplos:

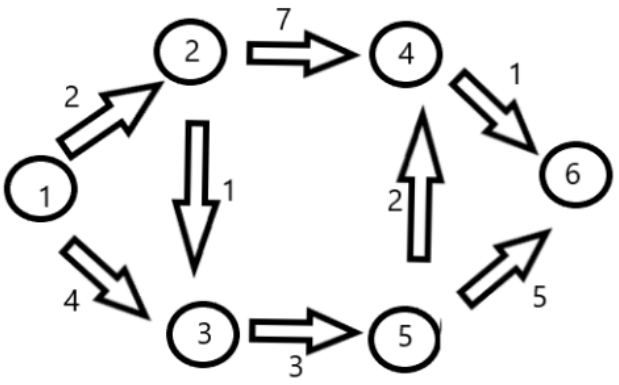


Imagem 4

Iniciando pelo vértice 1 temos que ele se liga para 2 com peso 2 e para 3 com peso 4, por não ter conhecimento dos demais vértices a distância até eles é infinita, contudo, indo pelo menor peso (2) agora temos conhecimento que a partir do 2 é peso 7 para o 4 e 1 para o 3. Logo o menor peso é 1 (para o 3), contudo já gastamos 2 vindo para o vértice dois, logo o valor total é 3 do vértice 1 até o 3.

O Dijkstra se mantém nessa lógica até percorrer todos os vértices, é muito utilizado em roteadores na atualidade.

5. Considerações Finais:

Embora a falha do Greed Method ser que as vezes ele falha em encontrar soluções globais por seu foco e não considerar todos os dados, ele ainda é um método interessante de se

utilizar. Sob as circunstâncias certas e com regras(passo-a-passo) bem definidas ele se mostra extremamente efetivo..

Alguns dos exemplos mostrados, como por exemplo o de Huffman, feitos para demonstrar a capacidade do algoritmo mostra suas capacidades. Não obstante, pode-se usar o algoritmo de Dijkstra para trabalhar em áreas como Redes de Computadores. Ou caso queira investir em bolsas de valores utilizando de um algoritmo que te aponte os melhores investimentos também pode-se implementar o Greed Method. Um algoritmo que se usado de forma certa, traz muitos benefícios.