MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - PICOS
CURSO BACHARELADO EM SISTEMAS DE INFORMAÇÃO
DISCIPLINA: ESTRUTURAS DE DADOS II
PROFESSOR(A): JULIANA

ALUNO: ARMANDO LUZ BORGES

RELATÓRIO

Referente ao terceiro trabalho da disciplina

Picos - PI

2023

Relatório referente ao terceiro trabalho da disciplina de Estruturas de dados II

Armando Luz Borges

Resumo:
Este relatório explora aplicações de grafos em diversas áreas e a análise de caminhos entre vértices, apresentando soluções para encontrar trajetos eficientes em grafos, incluindo a Torre de Hanoi, usando para isso os algoritmos de Dijkstra e Bellman-Ford. Além disso, também é introduzido um método para determinar o caminho mais confiável em grafos ponderados por probabilidades. Os resultados são destacados com base nos testes, mostrando a funcionalidade correta dos algoritmos em diferentes contextos. Por fim, conclui-se que o algoritmo de Dijkstra foi mais eficiente devido à ausência de arestas negativas e a solução de confiabilidade se mostrou eficaz na escolha de caminhos confiáveis.

1. Introdução

Os grafos, como estruturas de dados fundamentais na teoria dos grafos, têm uma ampla gama de aplicações em diversos campos, incluindo redes de comunicação, logística, sistemas de transporte, análise de dados e muito mais. Um aspecto essencial desses grafos é a análise de caminhos, que busca determinar rotas mais eficientes, confiáveis ou ótimas entre dois pontos/vértices.

Neste relatório, será apresentada uma abordagem para encontrar o caminho mais rápido entre dois vértices de um grafo simples que representa os possíveis movimentos do jogo lógico “Torre de Hanoi”, utilizando para isso dois algoritmos populares, o de Dijkstra e o de Bellman-Ford. Além disso, será apresentada uma solução eficiente para encontrar o trajeto mais confiável em um grafo ponderado com probabilidades que representam o nível de confiança do caminho.

O restante do relatório está organizado da seguinte forma: A seção 2 descreve as funções utilizadas e a lógica por trás de cada uma, abordando desde a sua finalidade até os parâmetros que elas exigem. Na seção 3 são apresentados os testes realizados com a finalidade de assegurar o funcionamento dos algoritmos e a integridade dos dados, além de medir o tempo de busca do caminho mais curto/confiável. Por fim, na seção 4, são feitas conclusões breves a respeito dos três algoritmos.

2. Funções utilizadas

Nesta seção são apresentadas as funções utilizadas na implementação de cada algoritmo, bem como a descrição de suas funcionalidades e parâmetros que exigem. Além disso, vale ressaltar que algumas das funções descritas abaixo são comuns em alguns dos algoritmos. Portanto, estas serão descritas apenas uma vez, pois implementam exatamente a mesma lógica.

- **findConfigIndex**
  Esta função é responsável por encontrar o índice de uma determinada configuração em um vetor de configurações, representado por uma matriz de caracteres. Essa função é usada para mapear as configurações (vértices) de início e fim, representadas como uma

string, para o índice correspondente na matriz de configurações. A função recebe um vetor de strings (matriz de caracteres) representando as possíveis configurações (vértices) do grafo, além de receber também uma string (vetor de caracteres), que seria o vértice que se deseja encontrar. A função retorna um inteiro representando o índice da configuração.

- **minDistance**
  Esta função é responsável por encontrar o índice do vértice não visitado com a menor distância acumulada a partir de um conjunto de vértices. Essa função foi utilizada nos algoritmos de Dijkstra e Bellman-Ford. A função recebe um vetor contendo as distâncias acumuladas a partir de um vértice (configuração) de origem até cada vértice. Além disso, também recebe um vetor de booleanos que indica os vértices que já foram visitados ou processados pelo algoritmo. A função retorna um inteiro que representa o índice do vértice com a menor distância.

- **dijkstra**
  Esta função implementa o algoritmo de Dijkstra para encontrar o menor caminho entre duas configurações (vértices) em um grafo. A função recebe o grafo representado como uma matriz de adjacência, composta pelos valores inteiros 0 e 1, de tamanho n *x* n. Além disso, também recebe um vetor de configurações (vértices) representados por um vetor de strings, um ponto (configuração) de partida e outro de chegada, ambos representados por strings. Esta função é auxiliada pelas duas descritas acima. Nos últimos passos de sua execução, o algoritmo exibe o número de passos necessários para concluir o menor caminho, bem como uma descrição do caminho em si. A função não possui retornos.

- **bellmanFord**
  Esta função implementa o algoritmo de Bellman-Ford, que também é utilizado para encontrar o menor caminho entre dois pontos de um grafo. A função também recebe uma matriz de adjacência composta pelos valores inteiros 0 e 1, de tamanho n *x* n, representando o grafo, além de receber também um vetor de configurações (vértices) representados por um vetor de strings, um ponto (configuração) de partida e outro de chegada, ambos representados por strings. Esta função é auxiliada somente pela função **findConfigIndex** para mapear os pontos de início e fim. A função não possui retornos, e nas últimas etapas de sua execução, ela mostra o número de movimentos necessários para percorrer o caminho mais curto encontrado, bem como o próprio caminho.

- **logMaxProb**
  Esta função é responsável por calcular o logaritmo negativo da probabilidade máxima. A função recebe um valor double representando a probabilidade e também retorna um valor double representando o logaritmo negativo dessa probabilidade.

- **mostReliablePath**
  Recebe uma estrutura representando um grafo, e dois inteiros representando os pontos de partida e de destino. A função implementa um algoritmo baseado no algoritmo de Dijkstra para encontrar o caminho mais confiável entre dois vértices em um grafo direcionado ponderado, onde as arestas têm associações de

confiabilidade. A função não possui retornos, e em suas últimas etapas de execução, ela exibe o caminho mais confiável e sua respectiva probabilidade.

3. Resultados dos testes

Nesta seção serão apresentados os testes realizados com o intuito de avaliar o funcionamento dos algoritmos, bem como também checar a sua eficiência e eficácia. Todas as informações a respeito dos resultados podem ser observados através das imagens de teste, inclusive o tempo de execução em microssegundos dos algoritmos.

Algoritmo de Dijkstra

O padrão de configuração dos vértices seguidos é descrito nas imagens abaixo. É possível constatar que o resultado é coerente confirmando com os desenhos do grafo modelado em seguida. Os pontos em azul representam os pontos de partida e de chegada, enquanto que os pontos em vermelho representam o caminho mais curto.

Teste 1:

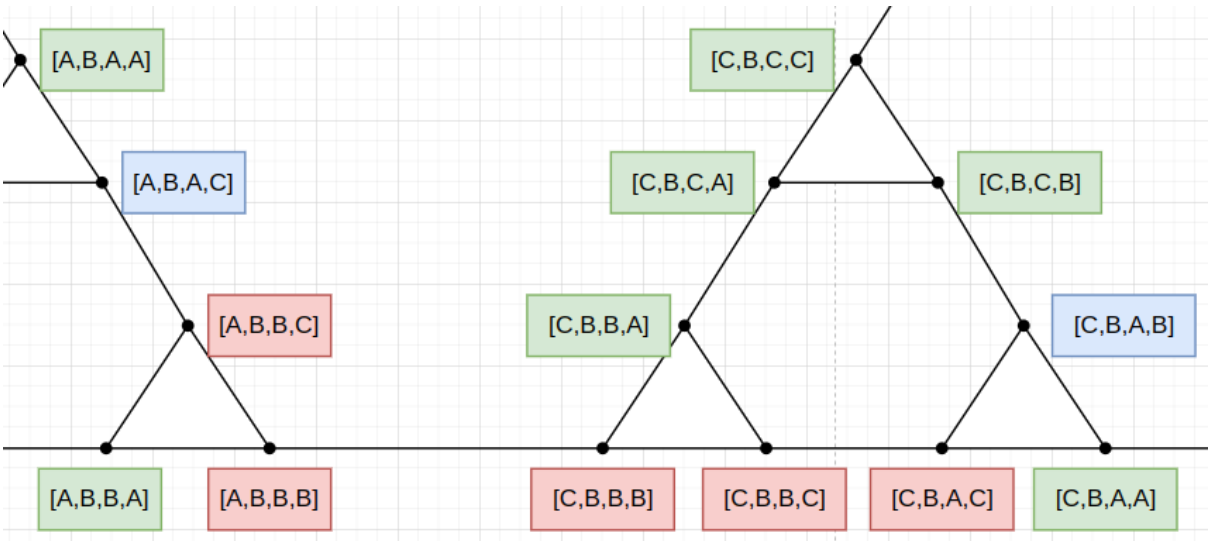
```
----------------------------------------------------------Torre de Hanoi----------------------------------------------------------
As jogadas sao descritas seguindo este padrao: '[A,A,A,A]'
Este padrao indica que os 4 discos estao no poste A, sendo que a esquerda ficam os maiores discos e a direita os menores.

1. Pesquisa do menor caminho com Dijkistra.
2. Pesquisa do menor caminho com Bellman.
3. Sair.
-> 1

Insira um ponto de partida no seguinte formato: [A,A,A,A]
-> [A,B,A,C]

Insira um ponto de destino no seguinte formato: [A,A,A,A]
-> [C,B,A,B]

Tempo de busca: 215.00 microsegundos
Menor número de movimentos de [A,B,A,C] até [C,B,A,B]: 6
Caminho percorrido: [C,B,A,B] <- [C,B,A,C] <- [C,B,B,C] <- [C,B,B,B] <- [A,B,B,B] <- [A,B,B,C] <- [A,B,A,C]
```



Teste 2:

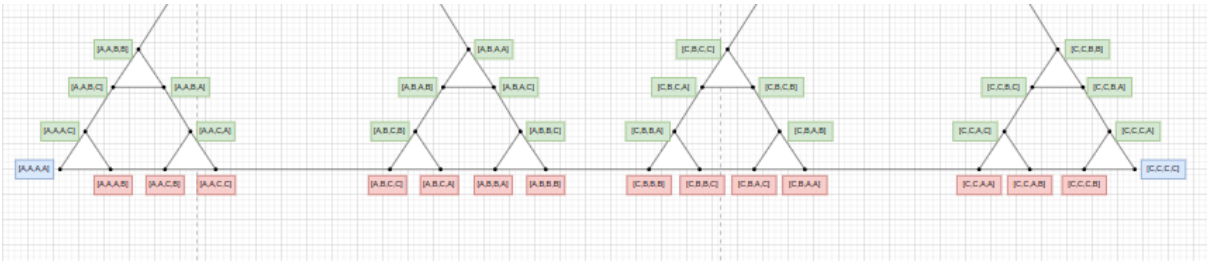
```
----------------------------------------------------------Torre de Hanoi----------------------------------------------------------
As jogadas sao descritas seguindo este padrao: '[A,A,A,A]'
Este padrao indica que os 4 discos estao no poste A, sendo que a esquerda ficam os maiores discos e a direita os menores.

1. Pesquisa do menor caminho com Dijkistra.
2. Pesquisa do menor caminho com Bellman.
3. Sair.
-> 1

Insira um ponto de partida no seguinte formato: [A,A,A,A]
-> [A,A,A,A]

Insira um ponto de destino no seguinte formato: [A,A,A,A]
-> [C,C,C,C]

Tempo de busca: 199.00 microsegundos
Menor número de movimentos de [A,A,A,A] até [C,C,C,C]: 15
Caminho percorrido: [C,C,C,C] <- [C,C,C,B] <- [C,C,A,B] <- [C,C,A,A] <- [C,B,A,A] <- [C,B,A,C] <- [C,B,B,C] <- [C,B,B,B] <- [A,B,B,B]
 <- [A,B,B,A] <- [A,B,C,A] <- [A,B,C,C] <- [A,A,C,C] <- [A,A,C,B] <- [A,A,A,B] <- [A,A,A,A]
```



Com base nos exemplos vistos acima, fica evidente o funcionamento adequado do algoritmo, visto que ele de fato busca o caminho mais curto de acordo com o grafo modelado.

Algoritmo de Ford-Moore-Bellman

Os padrões de configuração aqui também seguem a mesma estrutura que o algoritmo de Dijkstra.

Teste 1:

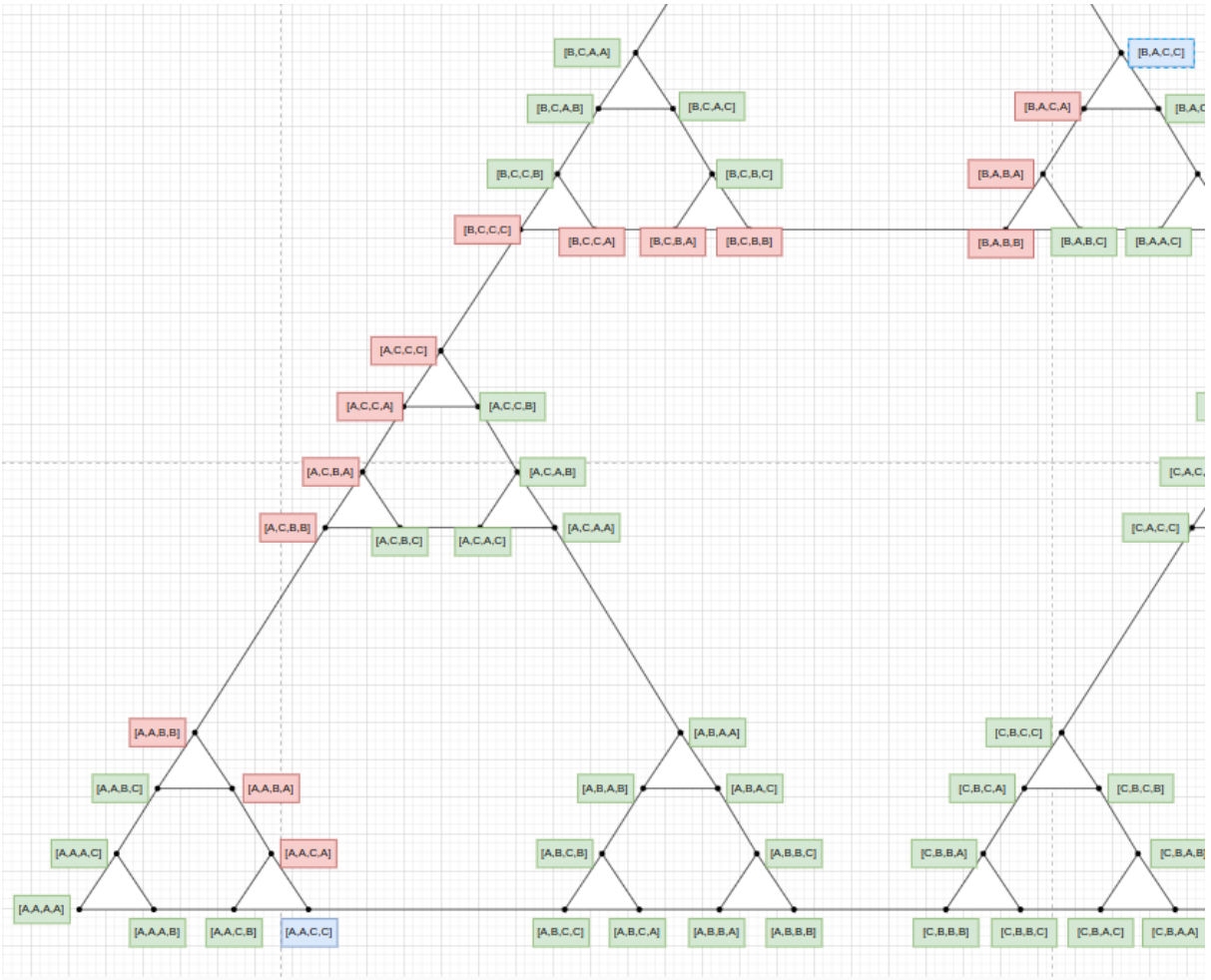
```
----------------------------------------------------------Torre de Hanoi----------------------------------------------------------
As jogadas sao descritas seguindo este padrao: '[A,A,A,A]'
Este padrao indica que os 4 discos estao no poste A, sendo que a esquerda ficam os maiores discos e a direita os menores.

1. Pesquisa do menor caminho com Dijkistra.
2. Pesquisa do menor caminho com Bellman.
3. Sair.
-> 2

Insira um ponto de partida no seguinte formato: [A,A,A,A]
-> [B,A,C,C]

Insira um ponto de destino no seguinte formato: [A,A,A,A]
-> [A,A,C,C]

Tempo de busca: 4114.00 microsegundos
Menor número de movimentos de [B,A,C,C] até [A,A,C,C]: 15
Caminho percorrido: [A,A,C,C] <- [A,A,C,A] <- [A,A,B,A] <- [A,A,B,B] <- [A,C,B,B] <- [A,C,B,A] <- [A,C,C,A] <- [A,C,C,C] <- [B,C,C,C]
 <- [B,C,C,A] <- [B,C,B,A] <- [B,C,B,B] <- [B,A,B,B] <- [B,A,B,A] <- [B,A,C,A] <- [B,A,C,C]
```



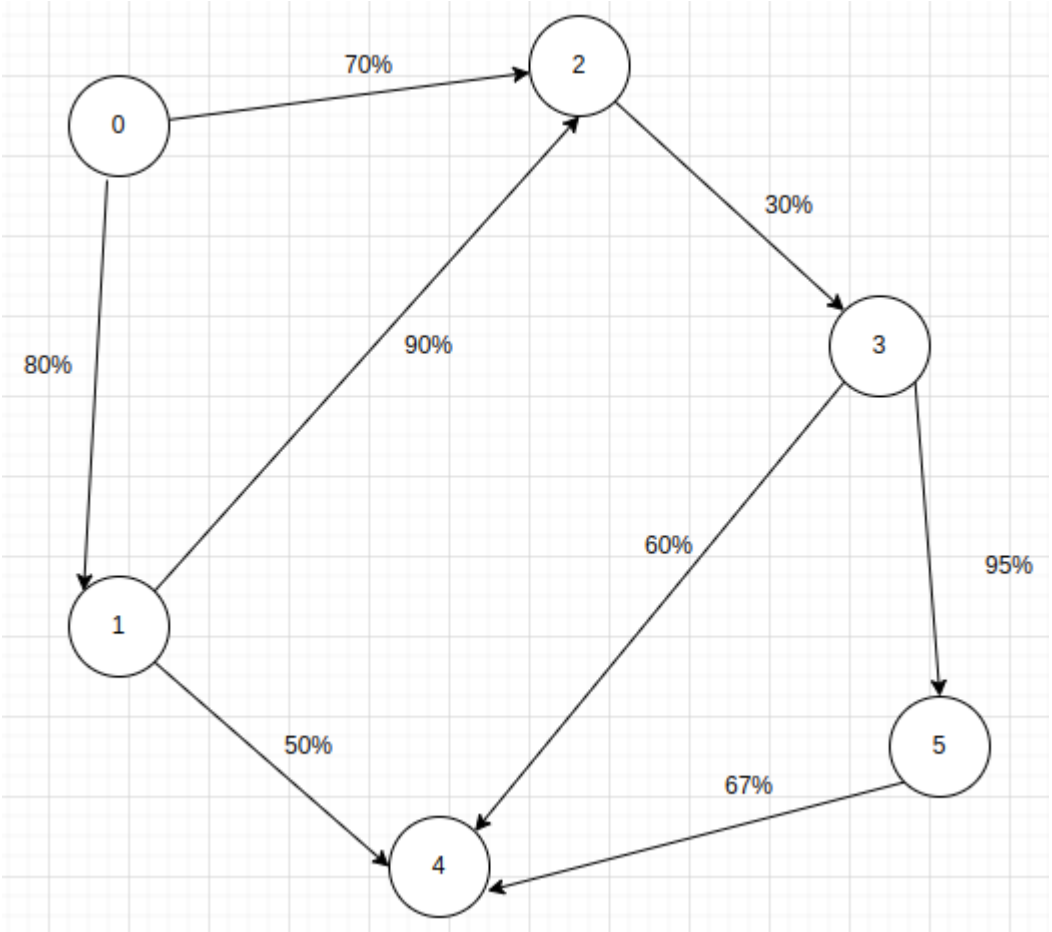
Com base neste único exemplo, é possível observar que o algoritmo de Ford-Moore-Bellman funciona corretamente, exibindo o menor caminho entre dois pontos distantes no grafo.

Algoritmo para o grafo de confiabilidade

Este algoritmo corresponde à solução da terceira questão do trabalho. Esta implementação é baseada no algoritmo de Dijkstra e, como o problema trata de valores de confiabilidade no intervalo entre 0 e 1, a solução propõe uma forma eficiente de tratar valores específicos que se caracterizam por serem muito baixos, próximos a zero. A ideia por trás disso, é trazer uma representação mais simples destes valores ao invés de realizar a multiplicação direta, a fim de tornar os cálculos de confiabilidade mais simples, evitando estouro de memória e outros problemas com arredondamento de valores. Abaixo segue um teste realizado no código.

Teste 1:

```
1. Procurar menor caminho.
2. Sair.
-> 1
Digite o vértice de origem: 3
Digite o vértice de destino: 4
Caminho mais confiável de 3 até 4 com probabilidade 0.6365
Caminho percorrido: 4 <- 5 <- 3
```



Como pode ser observado, o algoritmo não escolhe necessariamente o caminho direto do vértice 3 para o 4, na verdade ele busca corretamente o caminho mais confiável, mesmo que tenha de passar por outros vértices para isso.

4. Conclusão

Como pôde ser visto nos resultados, os algoritmos obtiveram sucesso em suas implementações. Vale observar que o algoritmo de Dijkstra obteve um desempenho significativamente maior que o de Bellman, visto que este último é mais eficiente quando se tem arestas com valor negativo. Como a Torre de Hanói não possui casos em que há arestas com valores negativos, o algoritmo de Dijkstra acaba se saindo melhor. Ademais, com relação ao último algoritmo, os resultados foram satisfatórios, visto que, no teste realizado, embora houvesse um caminho direto para o vértice de destino, existia um caminho alternativo que passava por outro vértice, mas que possuía uma maior confiabilidade, sendo este o caminho indicado pelo algoritmo.