Relatório Final MI - Algoritmos

Gustavo Henrique Bastos de Oliveira,[1]

[1]Engenharia da Computação – Universidade Estadual de Feira de Santana (UEFS)
Caixa Postal: 252-294 – CEP – 44.036.900 – Feira de Santana – BA – Brasil

ghboliveira@hotmail.com

1. Introdução

Diariamente muitas empresas trabalham com grandes volumes de dinheiro, esse volume fica retido nas empresas até alcançar um certo valor ou até um certo horário, quando uma dessas condições é alcançada essa quantia é enviada para um banco, por motivos de segurança. Essa coleta de dinheiro é geralmente realizada por empresas especializada em transporte de valores, onde agentes armados vão na empresa coletam os itens de valor e levam diretamente para o banco ou para algum outro posto de deposito. Para manter o nível de segurança o mais alto possível os agentes só descobrem o posto de coleta uma hora antes da coleta, esse tempo geralmente é utilizado para os agentes calcularem o caminho mais seguro entre a saída da garagem, coleta e o deposito, sabendo disso uma das empresas de transporte de valores chamado Forte Seguro decidiu torna o processo mais seguro, o presidente da empresa viu que poderiam ocorrer falhas neste processo e decidiu então que seria necessário um aplicativo que calculasse os caminhos entre estes pontos para que os agente possam escolher o caminho mais ideal e consequentemente o mais seguro. O presidente sabendo da capacidade da empresa GHSystem de desenvolvimento de aplicativos decidiu encomendar o aplicativo e notificou a empresa que seria necessário que o aplicativo funcionasse em um ambiente WEB, por motivos de segurança não explicou o por quê. Atendendo as necessidades da empresa Forte Seguro foi desenvolvido o aplicativo DistRotas onde é encontrado com facilidade e praticidade os caminhos entre os pontos cadastrados.

2. Fundamentação Teórica

2.1. Applet

Applets são plugins que são incorporados a softwares e geralmente possuem interface é dos tipos de APIs do java. Tem a capacidade de operar e/ou interagir com o programa em o qual foi encorporado. Applets também podem ser utilizados em navegadores de internet onde é executada pelo maquina virtual existente da maquina do cliente ou embutido no navegador.

2.2. ArrayList

Assim como os Applets, as ArrayLists são APIs java. ArrayList pode ser considerada uma coleção(objeto capaz de armazenar diversos tipos de objetos[Deitel, 2005, pág.673]) de qualquer tipo de coisa desde que seja um objeto. ArrayList é uma classe que implementa a interface List que definem um funcionamento básico para uma lista em java, como inserção, remoção, busca entre outros métodos básicos.

2.3. HashMap

O HashMap basicamente é uma espécie de lista onde cada objeto adicionado possui um valor e uma chave associada, permitindo buscas muito rápidas evitando o problema de ter que percorrer todos os itens para poder encontrar determinado objeto. Duas propriedades muito importantes dessa estrutura de dados é a capacidade inicial e o fator de carregamento. Quando um HashMap é instanciado o seu tamanho máximo é de apenas 16 itens, você pode definir no momento de instancia esse tamanho. A outra propriedade importante é o load fact(fator de carregamento), que momentos antes de atingir a capacidade definida atualiza esse valor para que novos itens possam ser inseridos, no caso padrão de 16 itens esse valor é aumentado para 32.

2.4. HashSet

Estruturas de dados do tipo Set são conhecidas por aceitar apenas valores únicos, ou seja em estruturas deste tipo não é possível a inserção de objetos já existentes. Dessas estruturas o HashMap é o mais rápido de todos com tempo de busca de O(1), ou seja independentemente da quantidade de itens inseridos ou da ordem de inserção o tempo de busca é o mesmo, mas isso acaba sendo um problema crítico de processamento em casos que envolvem milhões de itens, então esse tipo de estrutura é aconselhável apenas em casos que é necessário garantir a alta performance sem se importar com a ordem em que os objetos estão armazenados.

2.5. Grafos

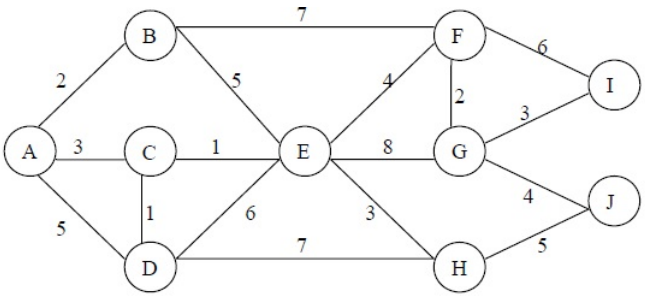
Grafos são estruturas de dados semelhantes a árvores(em Lafore (2004, pág.561)árvores são estrutura de dados que combinam o melhor de arrays e listas encadeadas) pois possuem referências entre seus nós seguindo uma hierarquia da raiz para as folhas, mas já em grafos não existe essa hierarquia pois podem haver ligações entre todos vértices(são o mesmo que os nó's em listas), na árvore só existe duas ligações entre nós, isso não ocorre com os grafos. Assim pode-se entender que todo tipo de árvore é um grafo, mas nem todo grafo é uma árvore. Uma outra diferença com as árvores é que em grafos podem haver vértices separados, quando isso ocorre o grafo é chamado de grafo desconexo. Apesar de poderem ser relacionados dessa forma árvores e grafos são usados de maneiras diferentes em programação. Grafos são utilizados geralmente seguindo um problema físico ou abstrato, como aviões em um sistema de aeroporto onde os vértices representam os aviões e as arestas(é um caminho até o outro vértice) a rota que o avião NÃO deve seguir, em um sistema bem elaborado dois vértices nunca se encontrarão. Além destes tipos de grafos também existe os grafos ponderados que são grafos que contém arestas com valores que significam o custo de um determinado vértice a outro vértice, como por exemplo, em sistemas de GPS dois aparelhos são considerados como vértices, onde o peso da aresta significa o custo para um aparelho encontrar o outro, esse custo pode ser o tempo, distância entre outros tipos de anotações que podem ser adotadas.

2.6. Algoritmo de Dijkstra

Talvez um dos problemas mais encontrados em grafos ponderados são o de menor caminho, onde é escolhido um ponto inicial e um ponto de destino e é necessário encontrar neste grafo o caminho onde possamos alcançar esses pontos com o menor custo possível. São problemas comumente encontrado em empresas de telefonia, onde tem um grafo

composto por portes(vértices no grafo) e linhas(arestas), onde devemos encontrar o menor custo para implantação de uma nova linha de cabos, iniciando em um poste x até um poste y. A solução adotada para este tipo de problema foi o algoritmo de Dijkstra, esse algoritmo foi concebido pelo holandês Edsger Dijsktra em 1959, esse algoritmo é baseado na representação de matriz de adjacência de um determinado grafo. Esse algoritmo não só encontra o menor caminho entre dois pontos, mas entre todos os pontos do grafo partindo de um determinado ponto.

Segue abaixo um exemplo de um grafo ponderado com a sua respectiva matriz de adjacência, onde fica o caminho necessário para partir de um ponto e chegar aos outros pontos do grafo, no exemplo a matriz foi criada partindo do ponto A.



	A	B	C	D	E	F	G	H	I	J
A	**0**	**2A**	3A	5A	∞	∞	∞	∞	∞	∞
AB			**3A**	5A	7B	9B	∞	∞	∞	∞
ABC				**4C**	4C	9B	∞	∞	∞	∞
ABCD					**4C**	9B	∞	11D	∞	∞
ABCDE						8E	12E	**7E**	∞	∞
ABCDEH						**8E**	12E		∞	12H
ABCDEHF							**10F**		14F	12H
ABCDEHFG									13G	**12H**
ABCDEHFGJ									**13G**	
ABCDEHFGJI										

Figure 1. Grafo conexo ponderado com matriz de adjacência gerado pelo algoritmo de Dijkstra

Suponha o seguinte grafo:

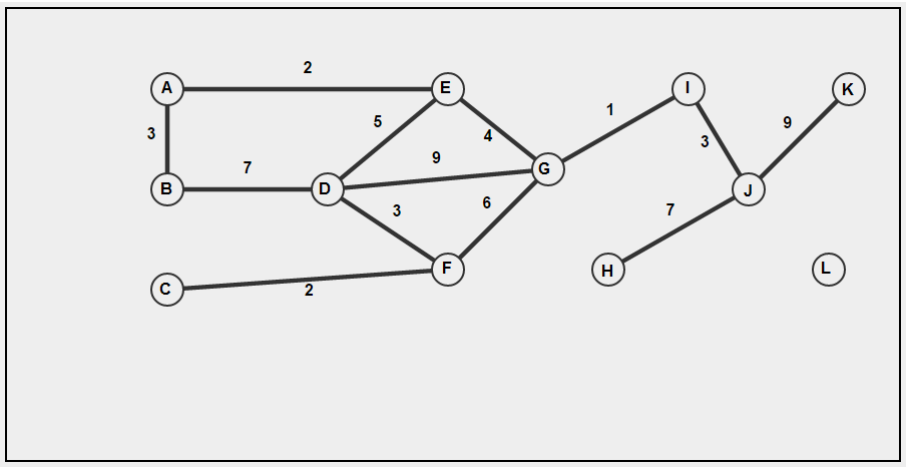


Figure 2. Grafo exemplo de diversos problemas.

Neste grafo pode-se observar diversos problemas que podem ser encontrados em grafos, como vértices desconexos(L) e caminhos com custos iguais(D - G, D - E - G, entre os demais). Se quisermos sair do vértice B para o vértice D, o algoritmo basicamente irá começar do vértice B e irá mapear todos os caminhos a partir deste vértice.

3. Materiais e Métodos

3.1. Os menores caminhos

Com o problema do menor caminho o algoritmo de Dijkstra atende a necessidade de encontrar o menor caminho de um ponto até qualquer outro ponto do grafo, mas com a necessidade de encontrar não só o menor tamanho, mas todos os menores caminhos, então seguindo o algoritmo foi necessário realizar algumas adaptações no código para a detecção de todos os menores caminhos. Foi criado uma classe Dijkstra com construtor padrão que recebe o grafo ao qual será usado o algoritmo.

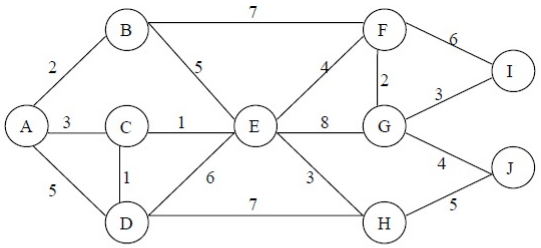
No momento a execução do algoritmo primeiro é necessário utilizar o objeto instanciado e executar o método executar passando o ponto inicial(método void), depois executar o método getCaminho passando null no primeiro parâmetro (por ser um método recursivo esse parâmetro apenas é usado entre chamadas recursivas) e no segundo o ponto de destino, esse método retorna uma lista de listas de vértices, basicamente uma lista com todos os caminhos possíveis entre os pontos definidos. Como pode-se vê abaixo:

```
Dijkstra operador = new Dijkstra(grafo);
operador.executar(inicio);
List<List<Vertice>> caminhos = operador.getCaminho(null, destino);
```

Figure 3. Instanciando um operador Dijkstra e encontrando o menor caminho no grafo ”grafo”, do ponto ”inicio” ao ponto ”destino”.

Simulando o algoritmo no seguinte grafo, começando do vértice A, irá adicionar todos os vértices adjacentes e calcular o custo entre estes vértices, após ter calculado todos os custos do vértice A, irá calcular os valores a partir dos vértices adjacentes e marcar o vértice A como visitado. Fazer isso até que todos os vértices sejam visitado, restando a seguinte matriz.

Seguindo a ordem da linha de cima para baixo, pode-se observar cada passo executado pelo algoritmo.



	A	B	C	D	E	F	G	H	I	J
A	**0**	**2A**	3A	5A	∞	∞	∞	∞	∞	∞
AB			**3A**	5A	7B	9B	∞	∞	∞	∞
ABC				**4C**	4C	9B	∞	∞	∞	∞
ABCD					**4C**	9B	∞	11D	∞	∞
ABCDE						8E	12E	**7E**	∞	∞
ABCDEH						**8E**	12E		∞	12H
ABCDEHF							**10F**		14F	12H
ABCDEHFG									13G	**12H**
ABCDEHFGJ									**13G**	
ABCDEHFGJI										

Figure 4. Grafo exemplo.

4. Resultados

Com a conclusão do aplicativo foram alcançados os seguintes resultados.

4.1. Applet

Com a necessidade da aplicação rodar em um ambiente Web todas as funções necessárias foram implementadas, gerando uma interface prática e amigável ao usuário. O desenvolvimento do aplicativo foi como o esperado, mas a execução do applet no navegador Web foi difícil, pois a tecnologia applet irá ser descontinuada até o final do ano pela Oracle(empresa responsável pelo Java), mas alguns navegadores não possuem mais suporte ao plugin. Outro problema é que para o java aceitar rodar o plugin no navegador é necessário gerar um certificado digital que dá autenticidade ao plugin, mas por ser de difícil de criar foi necessário adicionar o link onde será executado o applet na lista de exceções java. Para adicionar é necessário ir nas configurações java/segurança/editar lista de exceções e adicionar o link do applet.(para obter o link, é necessário abrir o arquivo html que tem que ficar uma pasta acima do arquivo java, geralmente na pasta bin da aplicação).

A tela inicial é composta pelos botões Cadastra Ponto, Cadastra Aresta, Remove Ponto, Remove Aresta, Escolher Rota e Mostrar Grafo.

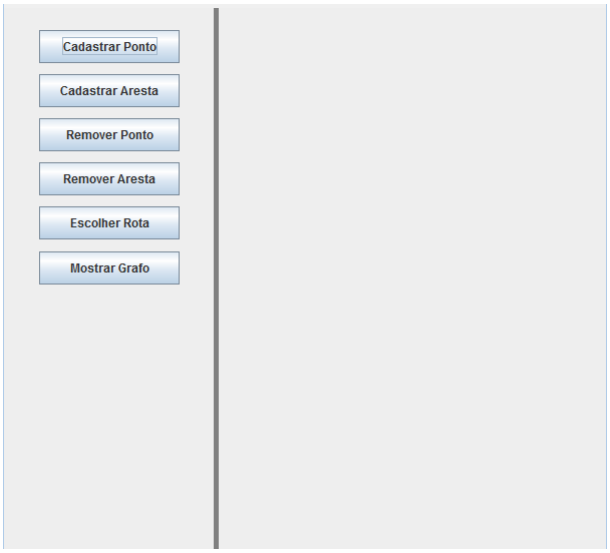


Figure 5. Tela inicial da aplicação.

Clicando no botão Cadastrar Ponto o usuário irá ver a seguinte tela de cadastro:

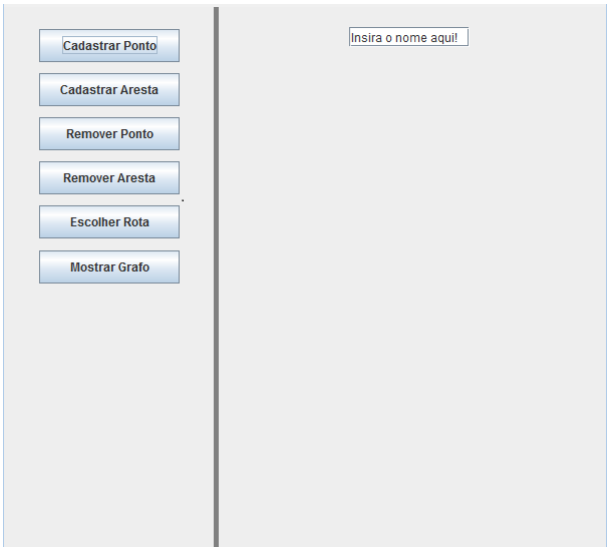


Figure 6. Tela cadastro de pontos.

Nesta tela o usuário irá poder definir o nome e cadastrar apenas um ponto por vez, e não é possível cadastrar pontos com nomes iguais.

Clicando no botão Cadastrar Aresta o usuário irá ver a seguinte tela de cadastro:

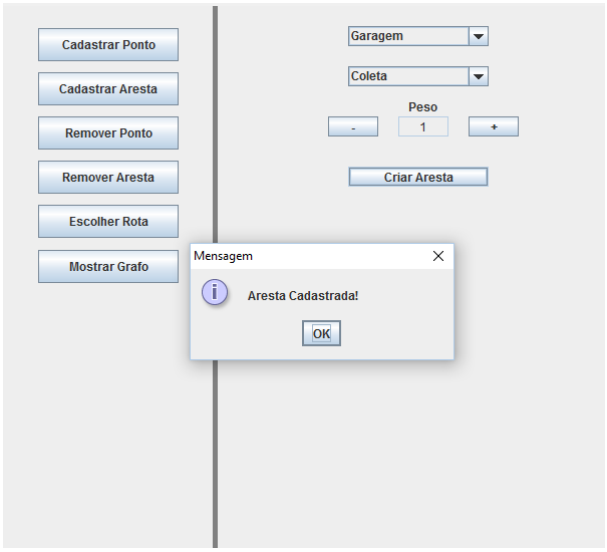


Figure 7. Tela cadastro de arestas.

Nesta tela o usuário pode cadastrar uma aresta entre os pontos escolhidos com um peso definido. Não é permitido a inserção de mais que uma aresta entre os pontos, pois uma aresta entre {a,v} é considerada a mesma entre {v,a} e também não é permitido arestas como o inicio e fim iguais.

Passando para o botão remover ponto, o usuário irá ver a seguinte tela:

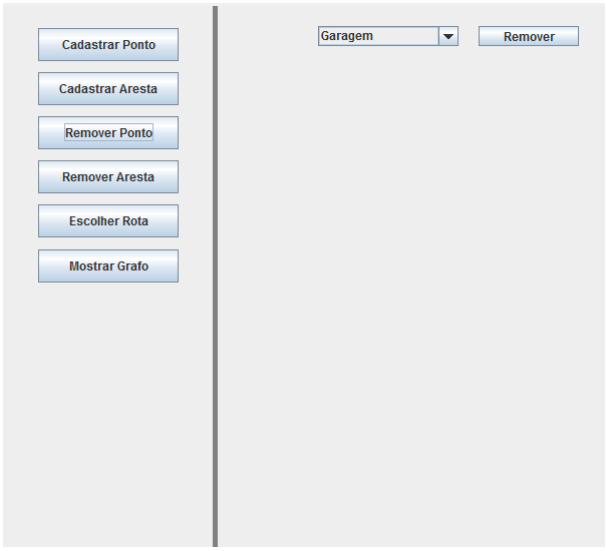


Figure 8. Tela remover ponto.

Nesta tela o usuário pode selecionar entre os pontos cadastrados, clicando em remover para excluir o ponto selecionado, retirando todas as aresta incidentes a este ponto.

Assim como o botão Remover Ponto o botão Remover Aresta permite a remoção das arestas cadastradas anteriormente.

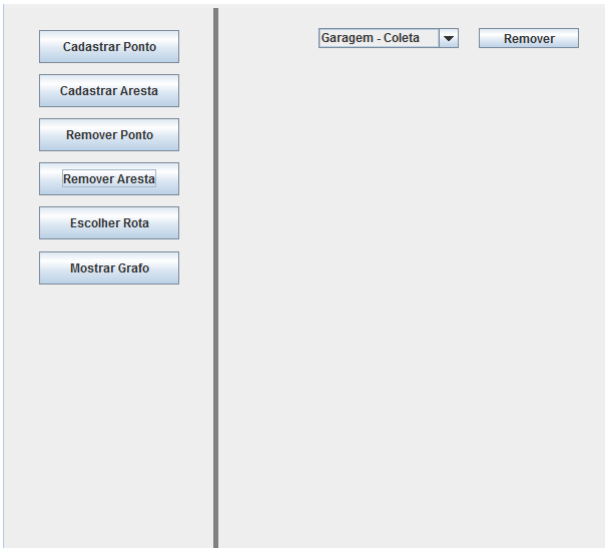


Figure 9. Tela remover aresta.

Já no botão Escolher Rota, o usuário irá ter a seguinte tela:

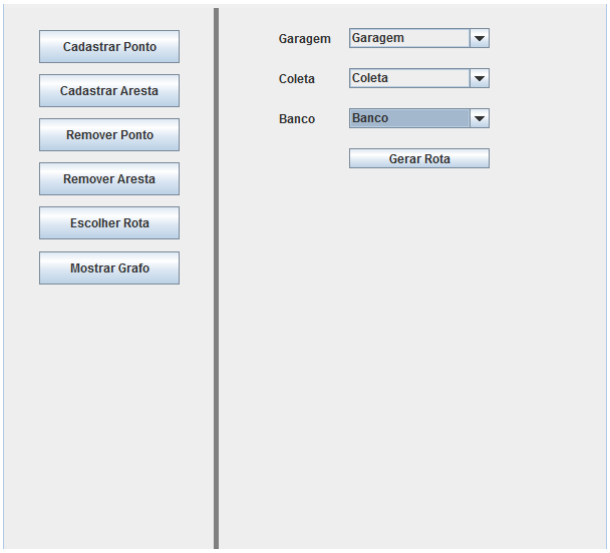


Figure 10. Tela para a escolha dos pontos da rota.

Nesta tela o usuário pode fazer a escolha entre os pontos, como garagem, coleta e banco. Depois de cadastrar os pontos o usuário pode encontrar o menor caminho entre estes pontos, o caminho vai ser visualizado a partir de um grafo, como o seguinte exemplo:

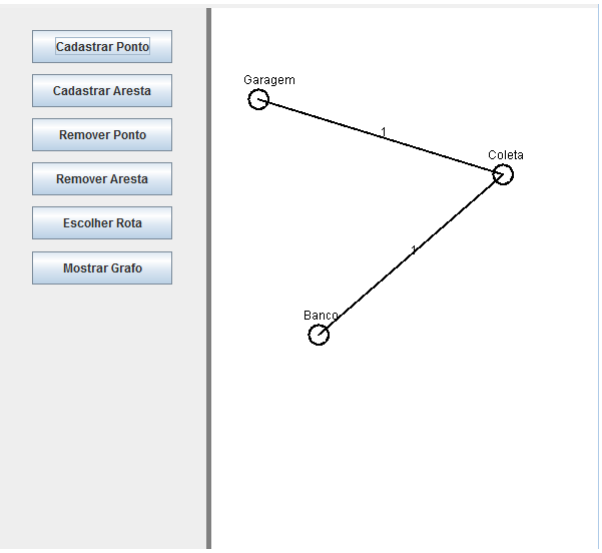


Figure 11. Grafo menor caminho.

Nesta tela é exibido todos os menores caminhos da garagem até o ponto de coleta e do ponto de coleta ao banco.

Já no ultimo botão, Mostrar Rota o usuário irá vê a seguinte tela:

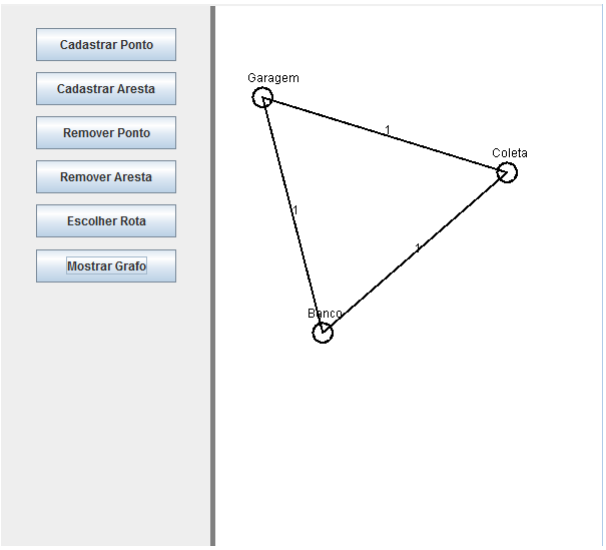


Figure 12. Pontos e arestas cadastradas.

4.2. Testes

No desenvolvimento do algoritmo de menor caminho foram executados 49 testes unitários com 6 grafos, onde todos os testes passaram da forma esperada, encontrando os caminhos possíveis esperados.

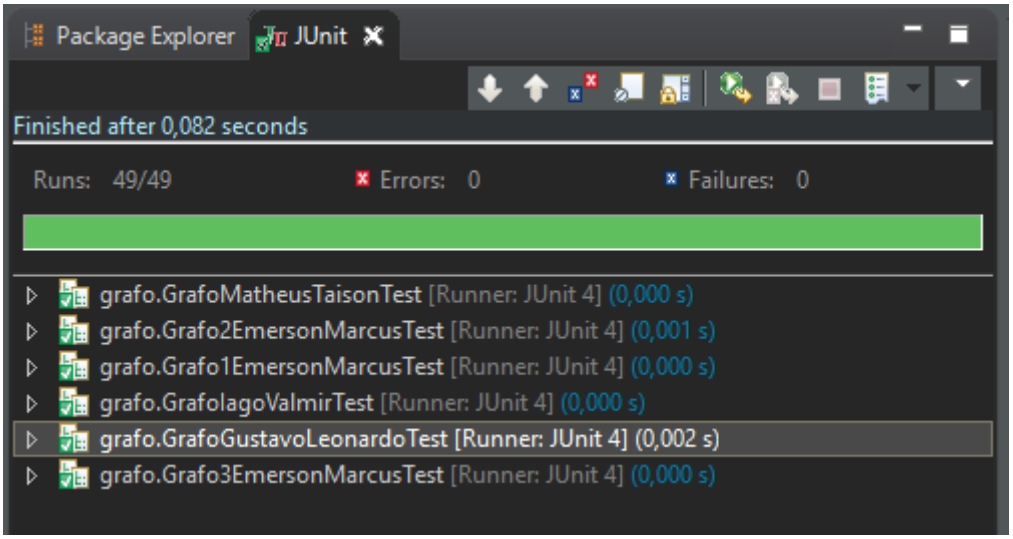


Figure 13. 49 testes executados com 6 grafos diferentes.

5. Conclusão

Partindo do problema proposto a solução atende ao exigido, desde a multiplataforma como a execução em navegadores Web ou até como aplicativo em sistemas que contém instalado a máquina virtual java. A interface foi desenvolvida com o pensamento em uma utilização fácil utilizando apenas o mouse. O algoritmo de menor caminho funcionou em todos os testes, retornando os caminhos possíveis.

6. Referências

Lafore, Robert, and Mitchell Waite. Data structures & algorithms in Java. Sams, 2003. Deitel, Paul, and Harvey Deitel. Java How to program. Prentice Hall Press, 2011. Tahchiev, Petar, et al. JUnit in action. Manning Publications Co., 2010.