Trabalho de Grafos

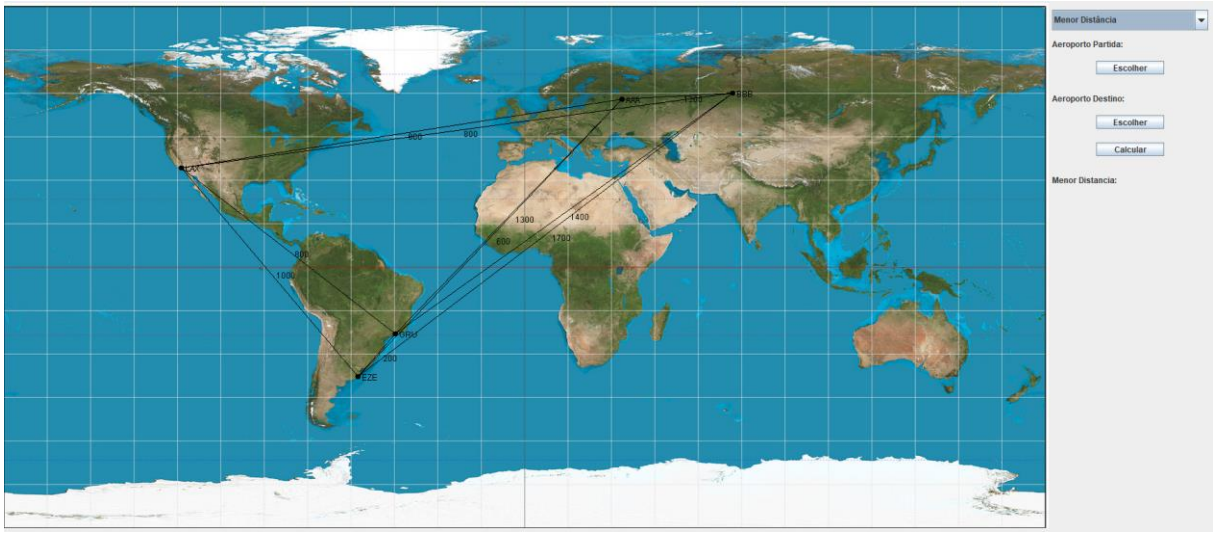
Rodrigo Padilha Fonseca

PUC Minas

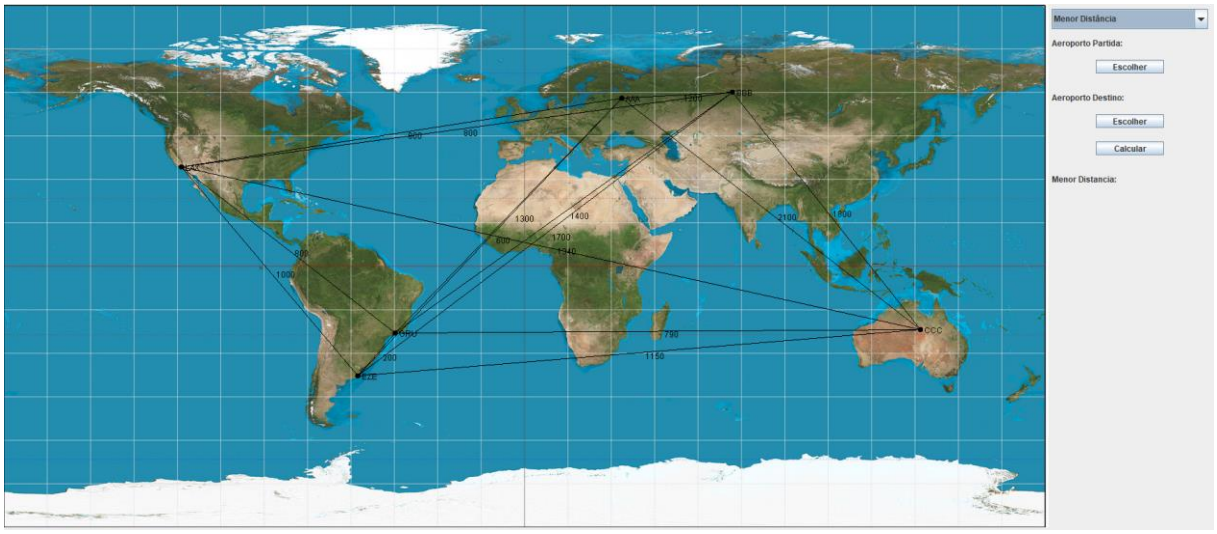
Belo Horizonte, Brasil

1. Mapa e Interface

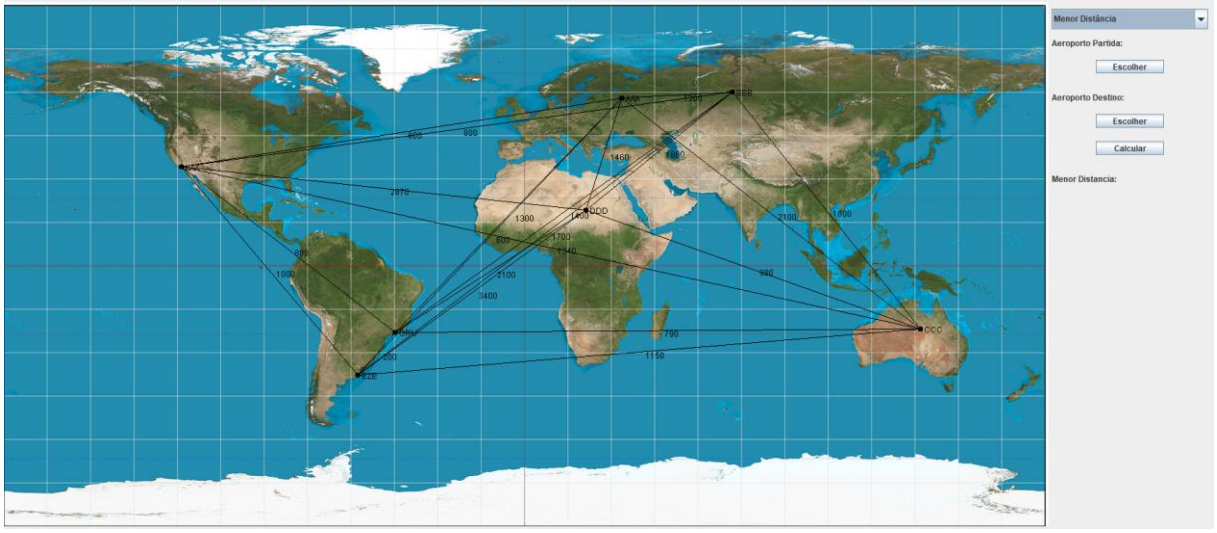
Para a construção da interface foi utilizada a biblioteca gráfica Java Swing. O mapa é uma projeção cilíndrica equidistante, e carregado na divisão da parte esquerda, e os outros componentes, como botões e labels no outro lado. Para preencher o mapa, são lidos os arquivos “aeroportos.txt”, que contém a quantidade dos aeroportos e seus nomes e posições latitudinais e longitudinais, e o arquivo “rotas.txt”, com a quantidade de rotas, os aeroportos e a tarifa. Os arquivos são lidos, e as posições convertidas em coordenadas. Para inserir os aeroportos e as rotas no mapa, foi utilizada a classe Graphics, que nos permite desenhar em uma imagem.
Os aeroportos e as rotas foram salvos como objetos, sendo que a classe Aeroporto contém como atributos o nome, e as posições x e y. Já as rotas possuem os aeroportos, preço e altura do voo. Ao ler os arquivos, foi instanciado um vetor com os aeroportos, um com as rotas, e criada uma matriz de adjacências de tarifas, e os aeroportos foram desenhados em suas posições como círculos, e as rotas como um segmento de reta entre os aeroportos utilizando a cor preta.
Para testar, foram utilizadas grafos completos de 5, 6, 7 e 8 vértices, e os tempos calculados em milissegundos utilizando a função currentTimeMillis().
A maioria das questões foi feita utilizando um grafo em que os vértices eram os aeroportos e as arestas os vôos,sendo a única exceção o problema de encontrar as alturas dos vôos.
Grafos utilizados para os testes:
1:



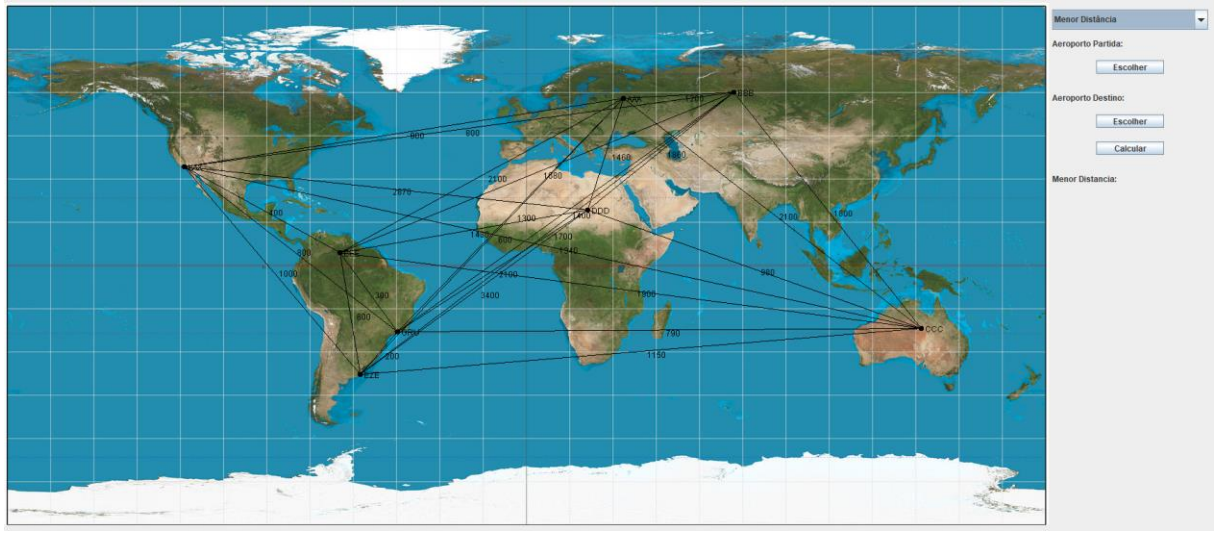
2:


Menor Distância
Aeroporto Partida:
Escolher
Aeroporto Destino:
Escolher
Calcular
Menor Distancia:

3:


Menor Distância
Aeroporto Partida:
Escolher
Aeroporto Destino:
Escolher
Calcular
Menor Distancia:

4:


Menor Distância
Aeroporto Partida:
Escolher
Aeroporto Destino:
Escolher
Calcular
Menor Distancia:

2. Menor Distância

O problema proposto é encontrar a menor distância entre dois aeroportos. Para fazer isso, foi instanciada uma matriz de adjacências com as distâncias calculadas entre os pontos que representam os dois aeroportos. Com essa matriz pronta, foi utilizada uma adaptação do algoritmo de Dijkstra. Depois de calculada, a menor distância é mostrada em um componente, e o caminho correspondente pintado de amarelo no mapa.

Tempos de execução:
1: Partida: EZE
Destino: BBB
Tempo: 1
2: Partida: GRU
Destino: CCC
Tempo: 1
3: Partida: AAA
Destino: DDD
Tempo: 1
4: Partida: LAX
Destino: EEE
Tempo: 1

3. Menor Tarifa

O problema proposto é bem semelhante ao anterior, com a diferença de que nesse ao invés de ser calculada a distância teremos que descobrir a menor tarifa entre dois aeroportos. Para isso, foi utilizada a matriz de adjacências com as tarifas que foi instanciada quando a interface estava sendo construída e uma adaptação do algoritmo de Dijkstra. Depois de calculada, o valor da menor tarifa é mostrado em um componente, e o menor caminho correspondente a essa tarifa é pintado de amarelo no mapa.

Tempos de execução:
1: Partida: EZE
Destino: BBB
Tempo: 1
2: Partida: GRU
Destino: CCC
Tempo: 1
3: Partida: AAA
Destino: DDD
Tempo: 1
4: Partida: LAX
Destino: EEE
Tempo: 1

4. Viagem ao Redor do Mundo

O problema proposto é encontrar a menor tarifa para uma viagem ao redor do mundo, passando por todos os aeroportos. Isso corresponde a encontrar a menor tarifa entre os caminhos hamiltonianos. Depois de calculado, o valor da menor tarifa é mostrado em um componente, e o menor caminho correspondente a essa tarifa é pintado de amarelo no mapa. Como não possui solução ótima, foram implementados dois algoritmos diferentes.

4.1. Força Bruta

Para encontrar os caminhos hamiltonianos, o algoritmo de força bruta executa uma adaptação de buscas em profundidade a partir de todos os vértices, retornando a tarifa para comparação apenas quando todos os vértices, que correspondem aos aeroportos, já tiverem sido visitados. Depois de todos serem percorridos, a menor tarifa terá sido descoberta, e então mostrada em um componente da interface. No mapa, o caminho correspondente a essa menor tarifa também será colorido de amarelo.

Tempos de execução:
1: Partida: AAA
Tempo: 1
2: Partida: BBB
Tempo: 1
3: Partida: CCC
Tempo: 2
4: Partida: EEE
Tempo: 9

4.2. Heurística – Vizinho mais próximo

Para uma solução mais rápida, foi implementada uma solução baseada no algoritmo do vizinho mais próximo, que adiciona o aeroporto de partida, e depois passa de vizinho a vizinho pelas arestas que possuem a menor tarifa, e tenta retornar ao vértice inicial quando todos já tiverem sido visitados.

Tempos de execução:
1: Partida: AAA
Tempo: 1
2: Partida: BBB
Tempo: 1
3: Partida: CCC
Tempo: 1
4: Partida: DDD
Tempo:

5. Subconjunto de Trechos

O problema proposto é encontrar o menor subconjunto de aeroportos a partir dos quais é possível ir de qualquer lugar para qualquer destino com o menor preço. Isso é o mesmo que encontrar a árvore geradora mínima do grafo, já que ela possuirá os vértices e as rotas necessárias para ir de qualquer ponto a qualquer outro vértice com o menor valor total. Depois de calculada, o valor da menor tarifa é mostrado em um componente, e o menor caminho correspondente a essa tarifa é pintado de amarelo no mapa. Como esse problema não possui solução ótima, também foi implementado um algoritmo de força bruta, e um utilizando uma heurística.

5.1. Força Bruta

O algoritmo de força bruta executado percorre recursivamente todas as árvores possíveis no grafo, e compara o valor da tarifa com o menor valor já encontrado até o momento caso todos os vértices tenham sido visitados.

Tempos de execução:
1: Tempo: 5
2: Tempo: 150
3: Tempo: 4830
4: Tempo: 549118

5.2. Heurística – Algoritmo de Prim

Para uma solução mais rápida desse problema, foi utilizado o algoritmo de Prim, que inicia a árvore com um vértice aleatório, que no caso da adaptação feita corresponde ao aeroporto de índice 0, e a partir dele adiciona os vizinhos do conjunto que possuem a menor tarifa. Para encontrar essa menor tarifa, a cada iteração todos os aeroportos que já foram adicionados são percorridos. Com a menor tarifa descoberta, o aeroporto correspondente é adicionado, e a tarifa aumentada.

Tempos de execução:
1: Tempo: 1
2: Tempo: 1
3: Tempo: 1
4: Tempo: 1

6. Altura dos vôos

O problema proposto é encontrar alturas diferentes para aeroportos que se cruzam, com o somatório dessas alturas sendo o menor valor possível. Para resolver isso, foi criado um novo grafo em que os vértices são os vôos, e as arestas os vôos que se cruzam. Depois disso, é encontrado o conjunto independente máximo do grafo, que corresponde ao maior conjunto de vértices que não possuem limitações entre si quanto à altura de vôo, e esses são retirados do grafo. O processo é repetido até o grafo estar vazio, com a altura do vôo iniciando em 10000 no primeiro conjunto e sendo aumentada em 1000 a cada iteração, e no final o valor dos vôos é adicionado no mapa, com diferentes cores e posições de acordo com o algoritmo executado (força bruta ou heurística). Como encontrar um conjunto independente máximo não possui solução ótima, para a parte de

encontrar os conjuntos independentes máximos foi implementada uma versão que utiliza força bruta e uma heurística.

6.1. Força Bruta

O algoritmo de força bruta implementado procura todos os conjuntos independentes do grafo dos vôos que foi construído, e depois disso analisa o somatório de cada um. Caso seja menor que o menor somatório encontrado até agora, o menor somatório é redefinido. Depois de tudo, a altura das rotas é alterada.
Tempos de execução:
Tempo em grafo 4-completo: 331
1(5-completo): Mais de 10 minutos

6.2. Heurística – Menor grau

Ao invés de verificar todos os conjuntos, foi encontrado apenas um utilizando a heurística de retirar sempre os vértices com menor grau. Para isso, foi percorrida a matriz de adjacência dos vôos. O resto do procedimento foi como está descrito em 6.
Tempos de execução:
1: Tempo: 1
2: Tempo: 1
3: Tempo: 1
4: Tempo: 1

Referências

Slides disponibilizados no SGA

https://stackoverflow.com/questions/23192573/java-proper-line-intersection-check