CAIXEIRO VIAJANTE

1- Algoritmos:
Os algoritmos implementados são: um para a busca do caminho ótimo utilizando força bruta, buscando e expandindo uma árvore para encontrando o menor caminho dentro dela e um algoritmo que busca apenas pelo menor caminho diretamente em uma árvore, optando apenas pela melhor escolha indo de uma cidade a outra. A seguir será explicado a forma de implementação, modo de funcionamento e tempos de execução, quantidade de memória utilizados e desvio de acerto para o algoritmo aproximativo.

1.1- Força bruta:
O algoritmo de força bruta executa N vezes (sendo N o tamanho do grafo) para formar uma árvore contendo todas as possibilidades de caminhos para o caixeiro viajante. Assim que conclui, verifica-se qual é o menor caminho que cobre todas as cidades saindo de uma cidade X e voltando para a mesma.

1.2- Aproximativo:
O algoritmo aproximativo é muito mais simples, ele busca apenas o menor caminho entre um pai e o filho, expandindo em uma lista contendo todos os nodos de menor caminho. Sera dado um exemplo mais a frente.

2- Entradas do programa e como são fornecidos o grafos utilizados:
Ao iniciar o programa, o usuário deve informar a quantidade de cidades, estas cidades são classificadas como nodos de um grafo.
O algoritmo irá gerar utilizando a função Rand() as distancias das arestas para uma entrada (x,y). Os valores de -1 significam que de uma cidade a outra não possui ligamento.

Exemplo base: Utilizarei o grafo não dirigido da figura 1 para exemplificar a execução do algoritmo aproximativo.

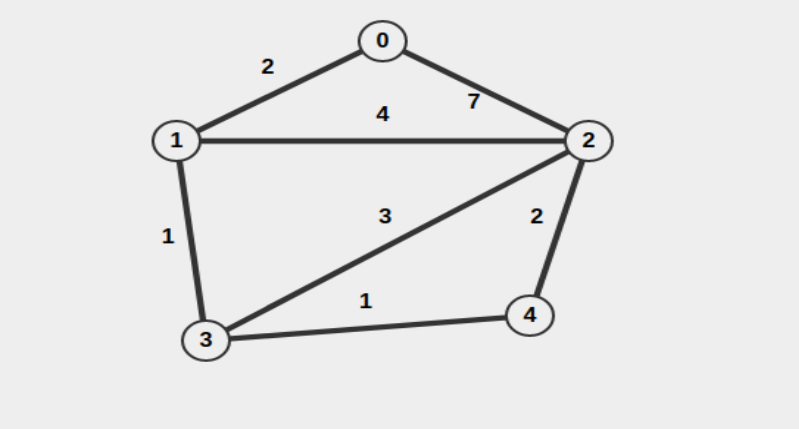

Figura 1: Grafo não dirigido

A partir deste grafo, é construído a matriz de adjacência:

Adjacency Matrix					
	0	**1**	**2**	**3**	**4**
0	0	2	7	0	0
1	2	0	4	1	0
2	7	4	0	3	2
3	0	1	3	0	1
4	0	0	2	1	0

Figura 2: matriz de adjacência

Funcionamento:
Os algoritmos percorrem a matriz do grafo iniciando pelo nodo definido como o inicial e terminando no mesmo nodo. O nodo inciais é são informados pelo usuário, exceto no caso de força bruta, onde o nodo inicial é escolhido a partir de uma analise para saber qual é a melhor escolha para cobrir todos os vértices tendo o menor custo.

Caso o nodo não seja encontrado, será retornado uma mensagem de Erro dizendo que não foi encontrado o nodo final, pois ele não possui conexão alguma com os demais nodos.

Ao rodar o programa, será mostrado no final o tempo de execução de cada algoritmo, o custo total de cada um e os nodos que foram percorridos para chegar ao resultado final.

Seguindo o exemplo do grafo não dirigido citado acima na figura 1 e 2, constrói-se a árvore de execução onde queremos sair do nó 0 e chegar novamente ao nó 0 passando por todos os nodos em busca da menor distancia entre estes dois nós

Iremos partir do nó zero, buscando sempre o menor caminho, somando e expandindo a arvore olhando sempre pro caminho do filho. Como na imagem a seguir:

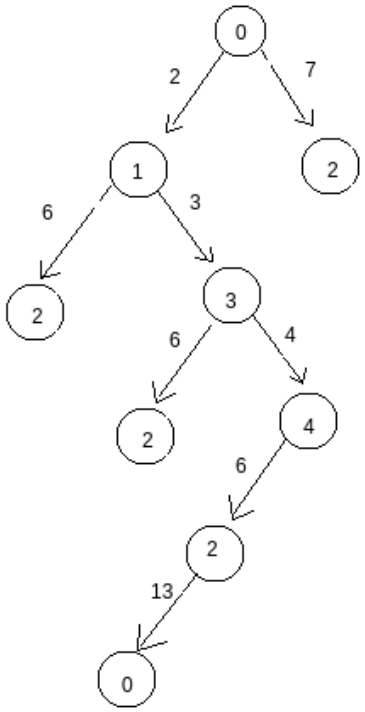

Figura 3: Arvore aproximativa expandida

A seguir um gráfico que expressa o tempo de execução dos dois algoritmos em milissegundos para diversos tamanhos de grafo:

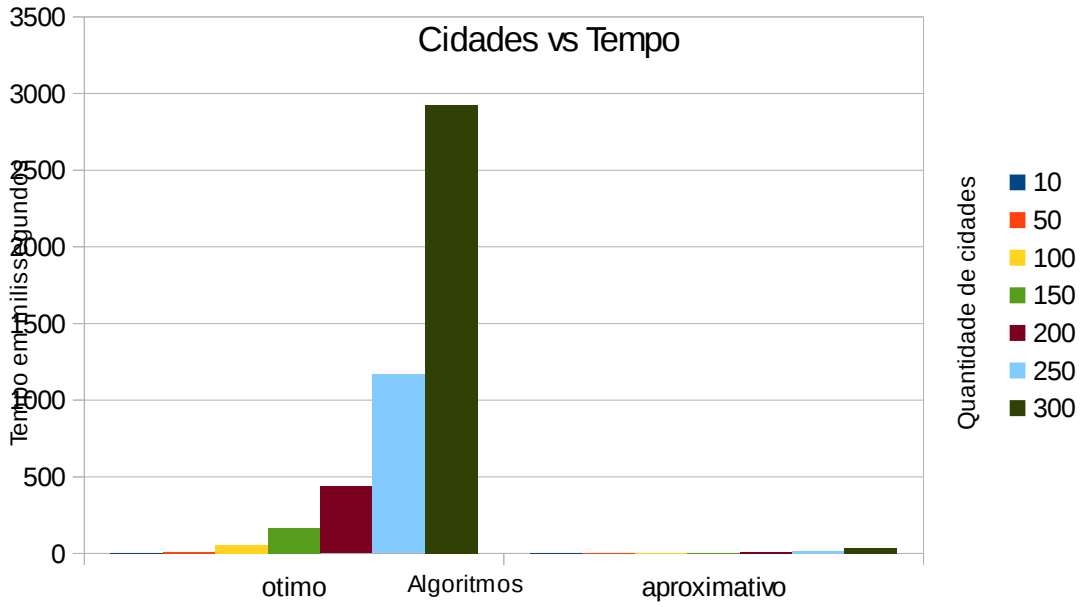

Gráfico: Gráfico do tempo de execução dos algoritmos em relação a quantidade de nodos.

Conclusões:

O algoritmo aproximativo implementado, não retorna sempre o menor custo, mas possui uma velocidade de execução muito menor que a do algoritmo ótimo, o problema é que sua resposta não é tão boa quanto no algoritmo ótimo, mas em certas vezes acaba retornando o melhor caminho.
Mas em questão de tempo e espaço consumido pelo algoritmo aproximativo, ele se torna eficaz, pois consome muito menos memória e tempo como mostrado no gráfico acima.
Nesta questão, caso queira encontrar uma resposta 100% correta, deve-se optar pelo algoritmo ótimo, pois mesmo demorando bem mais que o aproximativo, ele está sempre correto e com o menor custo entre o nodo inicial e final. Mas além do problema de tempo de execução, deve-se tomar cuidado pois na execução do algoritmo ótimo gasta-se muito mais memória, já que ele precisa guardar muitas informações dos nós expandidos.