RELATÓRIO TÉCNICO
TRABALHO FINAL EDA - Turma 2/2015

Github:
https://github.com/julioxavierr/distanciaMin.git

Alunos:
Júlio César Xavier Portela de Souza - 14/0024140
Marlon Mendes Ciriático Guimarães - 14/0063919

Descrição do algoritmo (Algoritmo de Dijkstra):

Algoritmo concebido pelo holandês Edsger Dijkstra, é provavelmente o algoritmo mais conhecido no tratamento de grafos. Assemelha-se ao Best First Search, no entanto é um algoritmo guloso, ou seja, toma a decisão que parece ótima no momento.

Este algoritmo não encontra apenas o caminho mais curto entre duas cidades A e B, mas sim entre uma cidade (chamada origem) e todas as outras.

Partindo do princípio que todos as arestas têm um peso positivo, o algoritmo considera um conjunto S de menores caminhos, partindo de um vértice inicial A. A cada passo do algoritmo busca-se nas adjacências dos vértices pertencentes a S aquele vértice com menor distância relativa a A e adiciona-o a S e, então, repetindo os passos até que todos os vértices alcançáveis por A estejam em S. Arestas que ligam vértices já pertencentes a S são desconsideradas.

Descrição dos dados utilizados:

Como fonte de dados, foram utilizadas vinte e sete cidades e municípios do estado de Goiás, além das distâncias entre as que eram ligadas diretamente por uma rodovia.

Para uso desses dados no código, foi utilizada uma lista de adjacência.

Dificuldades encontradas:

i. Cidades sem caminhos?

Houve uma dificuldade inicial ao selecionar e obter dados das cidades, resolvido após algum tempo de pesquisa. Existe um caminho possível para todas as cidades.

ii. Problemas com recursos computacionais?

Não.

iii. Estouro de memória? Em que casos?

Não.

Análise de casos:

Sistema Operacional utilizado:
Ubuntu 14.04

Hardware utilizado:
Notebook Dell Inspiron 15R 5537
8192 MB, DDR 3

i. Piores e melhores casos.

Para calcular a complexidade assintótica do algoritmo de achar o menor caminho entre duas cidades, considere o código abaixo:

```
int shortestDistance(Dijkstra *dijkstra, int target) {

   int vertex = -1, distance = -1;                                  O(1)

   while(1) {                                                       O(V)

      findClosestVertexAvailable(dijkstra, &vertex, &distance);     O(V)

      if(distance == INFINITY)                                      O(1)
         break;

      relaxVertex(dijkstra, vertex);                        O(size(adjList[vertex]));

      dijkstra->visited[vertex] = 1;                        O(1)
   }

   return dijkstra->minDistance[target];                    O(1)
}
```

V	**Conjunto de vértices (cidades)**
E	**Conjunto de arestas (estradas, caminhos)**
G(V, E)	**Grafo com vértices no conjunto V e arestas no conjunto E**
size(adjList[vertex])	**Tamanho da lista de adjacência do vértice "vertex"**
|C|	**Número de elementos do conjunto C**

A condição de parada deste loop é quando **distance** assume o valor **INFINITY**. Isto acontece quando o vetor de visitados (dijkstra->visited) está totalmente preenchido com valores 1, ou seja, todo os vértices foram visitados. Marcar um vértice como participado pode acontecer, no máximo, |V| vezes, tem-se então que o loop é executado |V| vezes.

|V| * findClosestVertexAvailable(dijkstra, &vertex, &distance) +
|V| * 1
E = soma de size(adjList[i]) , para todo 0 <= i < |V|
|V| * 1

Os valores |V| * 1 representam as operações constante do **if** e da atribuição de variável.
A função findClosestVertexAvailable(...) faz uma busca linear no array **minDistance**, que tem tamanho **|V|** (número de vértices), sempre passando por todos os elementos, portanto: findClosestVertexAvailable(dijkstra, &vertex, &distance) é **O(|V|)**,

A função relaxVertex(dijkstra, vertex) verifica todos os vértices diretamente conectados com o vértice **vertex**, então tem complexidade proporcional ao número de vértices. No código, o grafo é

representado como uma lista de adjacência, e podemos definir que o número de vértices diretamente ligados com um determinado vértice é o tamanho de sua lista. Como **size(adjList[vertex])** representa o tamanho da lista do vértice vertex, e esta operação é feita para todos os vértices, então temos que a soma dos tamanhos das listas de todo os vértices é igual ao número de arestas no grafo.

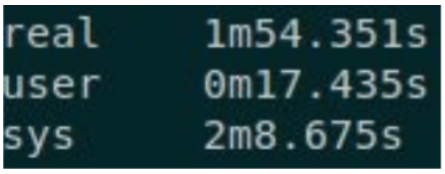
O(|V|) * (O(|V|) + O(1) + O(1) + O(|E|) =
O($|V|^2$ + |E|);

Observe que, o caso médio deste algoritmo é igual ao pior e melhor casos. Para as mesmas variáveis de entrada, número de vértices e arestas, estes tempos continuam os mesmos, pois a condição de parada do loop é igual, e as complexidades das funções findClosestVertexAvailable e relaxVertex permanecem iguais.

ii. Caso médio (considere todas as origens possíveis neste cálculo).

A presente implementação do algoritmo Dijkstra garante que, independente do início e destino, o tempo de execução será o mesmo para um mesmo grafo, pois os casos pior, médio e melhor são assintoticamente iguais.

Através de um script de bash o programa foi executado aproximadamente 50 mil vezes, a imagem a seguir resume os resultados obtidos.



A partir do tempo de execução real (1m54.351s) pode-se alcançar um valor médio próximo do tempo de execução do programa quando executado apenas uma vez.

Número de execuções	50.000
Tempo total de execução	104.351 segundos
Média	0,00208702 segundos

Gráfico de soluções: Distâncias Mínimas partindo de diferentes cidades.

Tabela de distâncias (em KM) usada no algoritmo

	GOIÂNIA	APARECIDA DE GOIÂNIA	HIDROLÂNDIA	BELA VISTA	PIRACANJUBA	SENADOR CANEDO	BONFINÓPOLIS	GOIANÁPOLIS	SILVÂNIA	ANÁPOLIS	ABADIÂNIA	ALEXÂNIA	CORUMBÁ	COCALZINHO	PIRENÓPOLIS	TEREZÓPOLIS	NERÓPOLIS	GOIANIRA	TRINDADE	ABADIA DE GOIÁS	ARAGOIÂNIA	GUAPÓ	SANTA BÁRBARA	ANICUNS	INHUMAS	VIANÓPOLIS	LUZIÂNIA
GOIÂNIA	0	21	0	0	0	23	0	49	0	0	0	0	0	0	0	33	37	28	0	22	0	0	0	0	0	0	0
APARECIDA DE GOIÂNIA	21	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0
HIDROLÂNDIA	0	20	0	38	51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BELA VISTA	0	0	38	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PIRACANJUBA	0	0	51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SENADOR CANEDO	23	0	0	36	0	0	34	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BONFINÓPOLIS	0	0	0	0	0	34	0	29	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GOIANÁPOLIS	49	0	0	0	0	36	29	0	0	25	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0
SILVÂNIA	0	0	0	0	0	0	46	0	0	69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	79	0
ANÁPOLIS	0	0	0	0	0	0	0	25	69	0	43	0	53	0	63	27	0	0	0	0	0	0	0	0	0	0	0
ABADIÂNIA	0	0	0	0	0	0	0	0	0	43	0	27	51	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ALEXÂNIA	0	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95
CORUMBÁ	0	0	0	0	0	0	0	0	0	53	51	0	0	20	20	0	0	0	0	0	0	0	0	0	0	0	0
COCALZINHO	0	0	0	0	0	0	0	0	0	0	0	0	20	0	40	0	0	0	0	0	0	0	0	0	0	0	0
PIRENÓPOLIS	0	0	0	0	0	0	0	0	0	63	0	0	20	40	0	0	0	0	0	0	0	0	0	0	0	0	0
TEREZÓPOLIS	33	0	0	0	0	0	0	17	0	27	0	0	0	0	0	0	52	0	0	0	0	0	0	0	0	0	0
NERÓPOLIS	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	52	0	0	0	0	0	0	0	0	34	0	0
GOIANIRA	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0	38	0	19	0	0
TRINDADE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	16	0	36	27	0	0	0	0
ABADIA DE GOIÁS	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	18	19	0	0	0	0	0
ARAGOIÂNIA	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	19	0	0	0	0	0
GUAPÓ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	19	19	0	0	0	0	0	0
SANTA BÁRBARA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	27	0	0	0	0	34	55	0	0
ANICUNS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34	0	73	0	0
INHUMAS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34	19	0	0	0	0	55	73	0	0	0
VIANÓPOLIS	0	0	0	0	0	0	0	0	79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	105
LUZIÂNIA	0	0	0	0	0	0	0	0	0	0	0	95	0	0	0	0	0	0	0	0	0	0	0	0	0	105	0

Como executar o programa:

1 - Execute o arquivo '**CaminhoMinimo**' presente na pasta principal usando o Terminal.

```
$ ./CaminhoMinimo
```

OU

2 - Compile o arquivo '**CaminhoMinimo.c**' presente na pasta principal usando o Terminal e em seguida execute-o.

```
$ gcc -o CaminhoMinimo CaminhoMinimo.c
$ ./CaminhoMinimo
```