Aluno: Lucas Block Villatore

Descrição do problema

Caminhada máxima: Toda manhã Sr. Gump sai para caminhar por sua cidade. Ele não gosta de passar mais de uma vez por cada lugar. Mas ele gosta de demorar em seu passeio. Dado um conjunto de lugares (vértices), as ligações entre estes lugares (arestas) e os custos (tempo) de percorrer cada uma destas ligações, devemos encontrar um trajeto para o Sr. Gump que seja o mais longo possível, sem repetir lugares.

Modelagem

Nesse problema, precisamos buscar o maior caminho possível com algumas limitações impostas. Para descobrir esse caminho, podemos utilizar backtrack na busca em profundidade para escolher o melhor caminho possível. Para cada chamada recursiva da busca em profundidade, verificamos se o caminho que estamos criando seguem algumas propriedades:

- É a primeira vez que visitamos o vértice;
- Se a aresta {u, v} não foi processada;
- Se o caminho não foi finalizado, i.e., verificar se o caminho voltou para o vértice 1;
- Se existe caminho um caminho possível para a raíz.

Detalhes da implementação

Nesse trabalho, para representar os caminhos possíveis, foi utilizado um grafo como estrutura de dados. Para representar esse grafo, utilizei uma matriz de adjacência Mn*n, onde n representa o número de vértices dado na leitura dos dados.

A implementação pode ser separada nos 4 tópicos a seguir:

- Leitura
- Geração da árvore de caminhos minimos
- Buscar pelo custo e caminho máximo
- Mostrar resultados na tela

Leitura

Nessa parte, é construído a estrutura do Grafo. Para cada valor lido i * j, é atualizado no grafo de adjacência tanto na posição i * j quanto na posiçao j * i para facilitar a manipulação.

Geração da árvore de caminhos mínimos

A partir do grafo gerado, o algoritmo realizará uma busca em profundidade para gerar uma árvore de caminhos mínimos. A cada caminho mínimo gerado, a busca em profundidade armazenará na váriavel *caminhos* todo o caminho percorrido até chegar novamente ao vértice inicial.

A busca em profundidade irá percorrer o próximo caminho se:

- O caminho é válido, i.e, não tentar ir até um vértice que já foi visitado

- Aresta *{v, u}* não tenha sido processada, i.e, não tiver no vetor de caminhos percorridos [..., v, u, ...]
- O caminho não estiver finalizado, i.e, não ter voltado para o vértice inicial.

Busca pelo custo e caminho máximo

Nessa parte, o algoritmo vai fazer uma busca simples na árvore gerada pela etapa anterior. O algoritmo percorre o caminho que foi utilizado na busca em profundidade e armazena o custo desse caminho. O caminho que tiver o maior custo será retornado para o programa principal.

Mostrar resultados na tela

O algoritmo mostrará 3 informações:

- Na saída de erro, quantos nós na árvore de caminhos mínimos foram gerados
- Na saída padrão
  - O custo do caminho
  - O caminho percorrido.

Compilação

```
$ make
```

Vai gerar o um executável chamado *caminhada*

Como executar

```
$ ./caminhada < entradas/10-vertices-1
```

```
$ 92
$ 1 8 1
```

Exemplos utilizados para teste

Entrada:

```
./caminhada < entradas/10-vertices-1
```

Saída:

```
92
1 8 1
```

Entrada:

```
$ ./caminhada < entradas/10-vertices-2
```

Saída:

```
94
1 5 1
```

Entrada:

```
$ ./caminhada < entradas/10-vertices-3
```

Saída:

```
52
1 10 1
```

Entrada:

```
$ ./caminhada < entradas/10-vertices-4
```

Saída:

```
207
1 8 5 4 9 6 3 10 1
```

Entrada:

```
$ ./caminhada < entradas/10-vertices-5
```

Saída:

```
185
1 5 3 6 9 4 7 2 8 1
```

Entrada:

```
$ ./caminhada < entradas/10-vertices-6
```

Saída:

```
335
1 4 5 3 10 6 7 9 8 1
```

Entrada:

```
$ ./caminhada < entradas/40-vertices-1
```

Saída:

```
802
1 20 5 21 2 19 32 17 18 15 26 31 28 4 24 16 3 39 8 10 38 40 9 25 12 11 37 29 36
27 1
```

Entrada:

```
$ ./caminhada < entradas/50-vertices-1
```

Saída:

```
88
1 18 1
```