Documentação de Funções

Os códigos contém as funções comuns de um grafo (criaGrafo, insereAresta), por isso não serão descritas.

Foram utilizadas structs para agrupar as informações de um grafo e de uma fila. Estão da seguinte forma:

```
typedef struct {
    int nVertices, ehPonderado;
    int **pesos, **arestas;
    int *grau, *cor, *dist, *anterior;
} Grafo;

typedef struct Fila{
    int vertice;
    struct Fila *prox;
} Fila;
```

`*cor`, `*dist` e `*anterior` são atributos adicionados ao grafo para guardar informações importantes para a busca em largura. Também foram utilizados alguns #define para tornar a essa busca possível:

```
#define WHITE 0
#define GRAY 1
#define BLACK 2
```

A busca em largura foi feita por meio de iterações registrando em uma fila os vértices adjacentes ao atual para analisar posteriormente. Os vértices não visitados são marcados como `WHITE`, os agendados na fila são marcados como `GRAY` e os já visitados são marcados como `BLACK`. Assim nenhum vértice entre duas vezes na fila e os vértices que são adjacentes de mais de um outro vértice são mapeados a partir da primeira rota até eles que foi encontrada.

A partir das distâncias mapeadas, uma outra busca semelhante a busca em profundidade é utilizada para montar as rotas da busca em largura.

As questões que usaram filas (Torre de Hanoi e Viagem de Arthur) implementam as seguintes funções de fila:

1. `Fila *enqueue(Fila *fila, int vertice);`
   Adiciona um item no final da fila e retorna a fila modificada.
2. `Fila *dequeue(Fila **fila);`
   Retira o item do início da fila e o retorna.
3. `void showFila(Fila *fila);`

Apresenta todas as informações contidas numa fila por meio de recursão.

serão citadas nas lista de funções de cada uma, mas não serão descritas novamente.

1. Torre de Hanoi

Para a torre de hanoi foram utilizados dois `#define` a mais para representar a quantidade de discos e de pinos:

```
#define DISCOS 4
#define PINOS 3
```

A seguir as funções utilizadas na torre de hanoi:

1. `int ** buscaLargura(Grafo **gr, int origem, int destino, int *visitados, int **rotaFinal, int passos);`
   A partir da origem informada, mapeia a distância de cada vértice em relação à origem de acordo com o algoritmo apresentado no relatório.
2. `void printEstado(int *atual);`
   Exibe para o usuário no formato "(%d,%d,%d,%d)" o estado `*atual`.
3. `int contem(int *lista, int tam, int num);`
   Retorna 1 se `num` estiver contido em `*lista` e 0 caso contrário.

Para montar o grafo

4. `void montaHanoi(Grafo **gr, int **estados, int tamanho);`
   Testa todas as combinações de estados possíveis e insere arestas entre os estados que retornem 1 na função `comparaEstado`.
5. `int comparaEstado(int *estado1, int *estado2, int tamanho);`
   Utiliza as 4 funções seguintes para informar se existe aresta entre `*estado1` e `*estado2` de acordo com as regras apresentadas no relatório. Retorna 1 caso haja aresta e 0 no caso contrário.

6. `int discoEmBaixo(int *estado, int posDisco, int tamanho);`
   Retorna 1 caso haja um disco abaixo do `posDisco`. Não é permitido colocar um disco em cima de outro disco menor. logo, não podem haver discos com o mesmo valor de `posDisco` nas posições mais à direita.
7. `int discoEmCima(int *estado, int posDisco, int tamanho);`
   Retorna 1 caso haja um disco acima do `posDisco`. Discos mais a direita são menores. portante ficam acima dos mais a esquerda que têm o mesmo valor.

8. `int posDiff(int *vetor1, int *vetor2, int tamanho);`
   Retorna um inteiro informando a primeira posição que tem valores diferentes em `*vetor1` e `*vetor2`.
9. `int contEquals(int *vetor1, int *vetor2, int tamanho);`
   Retorna um inteiro informando quantos números são iguais entre `*vetor1` e `*vetor2`.

Para registrar a rota

10. `int ** montaRota(Grafo **gr, int origem, int destino, int *visitados, int **rotaFinal, int passos);`
    Dado um ponto de origem no grafo da torre de hanoi e um ponto de destino, retorna a melhor e a pior rota em `**rotaFinal` por meio de uma busca em profundidade levando em consideração as distância de cada vértice em relação ao estado (1,1,1,1) mapeadas pela função de busca em largura. Só adiciona na rota vértices com distância maior ou igual a do vértice atual, dessa forma as rotas comprem o requisito de não considerar jogadas anteriores.
11. `int length(int *lista, int tam);`
    Retorna o tamanho de uma ponteiro de inteiros. Considera apenas os valores antes do primeiro 0 da lista, caso haja. `tam` é o tamanho máximo possível.
12. `void copyList(int **copia, int *original, int tam, int local);`
    Copia os valores de um ponteiro de inteiros para a primeira posição de um ponteiro de ponteiros inteiros. Se fez necessário usar ponteiro de ponteiro para guardar as rotas das buscas. A variável `local` diz em qual posição de `**copia` o ponteiro que irá receber `*original`. Foi convencionado que a maior rota fica na posição 0 e a maior rota na posição 1.
13. `Fila *enqueue(Fila *fila, int vertice);`
14. `Fila *dequeue(Fila **fila);`
15. `void showFila(Fila *fila);`

Para entrada de dados

16. `int ehIgual(int *estado1, int *estado2);`
    Retorna 1 caso os estados informados sejam iguais e 0 caso contrário.
17. `int traduzEstado(int **estados, int *estado);`
    Informa em qual posição do vetor (+1) está localizado o estado informado pelo usuário.
18. `int * entrada();`
    Recebe como entrada do usuário 4 inteiros representando um estado do jogo.

2. Tubo de Formigas

A struct do grafo do tubo de formigas é um pouco mais simples do que o usado nas outras questões. Ficou da seguinte forma:

```
typedef struct {
    int nVertices, ehPonderado;
    int *pesos, *arestas;
} Grafo;
```

Isso porque não foi usada busca em largura, então aqueles ponteiros adicionais são desnecessário. O `*grau` também foi excluído porque o grau de todos os vértices nesse problema é 1, tornando este vetor desnecessário. Além disso a grau sempre ser 1 também levou a possibilidade de `*pesos` e `*arestas` serem transformados em ponteiros normais (antes eram ponteiro de ponteiro) porque nunca irá ocorrer de um vértice guardar mais de uma adjacência. Ponteiros são suficientes. Também não foi utilizado filas neste código.

As funções utilizadas neste código foram as seguintes:

1. `void geraEstados(int **estados);`
   Adiciona todos os estado possíveis para o problema em `**estados`.
2. `int ** montaRota(Grafo **gr, int origem, int destino, int *visitados, int **rotaFinal, int passos);`
   Executa uma busca em profundidade para montar a rota até o destino, copiando para `**rotaFinal` o menor caminho encontrado que contenha o destino. No entanto, no escopo deste exercício existe apenas uma rota possível de qualquer estado até o destino, logo, a busca é apenas uma rescursão que se repete até chegar ao destino.

Para montar o grafo

3. `void montaTubo(Grafo **gr, int **estados, int tam);`
   Itera na lista de `**estados` e busca para qual estado o atual aponta. Quando encontra, adiciona no vetor pelo método `inserirAresta()` atual.
4. `int * proxEstado(int *estado);`
   Utiliza as 4 funções seguintes para verificar qual será o estado seguinte segundo as regras de movimentação das formigas apresentadas no relatório e retorna o novo estado.
5. `int temColisao(int *estado);`
   Retorna 1 se houverem pares de +1 e -1 dentro de `estado`.
6. `int temSaida(int *estado);`
   Retorna 1 se `estado[0]== -1` ou `estado[tamanho-1]== 1`, e 0 caso contrário.
7. `int * resolveSaida(int *estado);`
   Retorna um estado com menos posição removendo o começo caso `estado[0] == -1` e/ou o final caso `estado[tamanho-1] == 1`

8. `int * resolveColisao(int *estado);`
   Executa as multiplicações por -1 nos pares de posição onde +1 antecede -1. Retorna o estado modificado com as colisões executadas.

Auxiliares

9. `void copyList(int **copia, int *original, int tam);`
   Copia os valores de um ponteiro de inteiros para a primeira posição de um ponteiro de ponteiros inteiros. Se fez necessário usar ponteiro de ponteiro para guardar as rotas das buscas.
10. `int ehIgual(int *estado1, int *estado2);`
    Retorna 1 caso os estados informados sejam iguais e 0 caso contrário.
11. `int tamEstado(int *tamEstado);`
    Retorna o tamanho de um estado informado pelo usuário.
12. `int contem(int *lista, int tam, int num);`
    Retorna 1 se `num` estiver contido em `*lista` e 0 caso contrário.
13. `int length(int *lista, int tam);`
    Retorna o tamanho de uma ponteiro de inteiros. Considera apenas os valores antes do primeiro 0 da lista, caso haja.
14. `void printEstado(int *estado);`
    Imprime para o usuário o estado passado por parâmetro no formato “(%d,...,%d) para o usuário. Funciona para qualquer tamanho de estado.

Entrada

15. `int traduzEstado(int **estados, int *estado);`
    Informa em qual posição do vetor (+1) está localizado o estado informado pelo usuário.
16. `int * entrada();`
    Recebe como entrada do usuário 4 inteiros representando um estado do jogo.

3. Viagem de Arthur

Funções da Viagem de Arthur.

4. `void montaGrafo(Grafo **gr);`
   Recebe do usuário uma entrada formatada da seguinte forma para montar um grafo: A primeira linha da entrada recebe dois inteiros V e A representando o número de vértices e o número de arestas, respectivamente. As A linhas seguintes são compostas por três inteiros O, D, P representando a origem, destino e peso, respectivamente.

5. `int buscaProfundidade(Grafo **gr, int origem, int *visitados, int **rotaFinal, int passos, int dinheiro);`
   Monta todas as rotas possíveis levando em conta o dinheiro disponível e vértices já visitados por meio de iteração de todos os elementos adjacentes ao vértice atual e busca recursivas em cada um. A rota é acompanhada pelo vetor de visitados que são guardados em `rotaFinal` sempre que a lista de visitados se tornar maior que a rota já guardada. `rotaFinal` é um ponteiro de ponteiro para que seu valor continue preservado mesmo depois de voltar da recursão. Retorna o dinheiro que sobrou considerando a maior rota encontrada.
6. `int buscaLargura(Grafo **gr, int origem, int **rotaFinal, int dinheiro);`
   A partir da origem informada, mapeia a distância de cada vértice em relação à origem de acordo com o algoritmo apresentado no relatório. Retorna o dinheiro que sobrou considerando a maior rota encontrada.

Para registrar a rota

7. `void montaRota(Grafo **gr, int origem, int **rotaFinal, int final);`
   Faz uma iteração a partir do vértice `final` de `**gr` até chegar na `origem`. A cada iteração o vértice é adicionado no vetor de `**rotaFinal` e segue para o vértice contido em `(*gr)->anterior[final-1]`.
8. `int contem(int *lista, int tam, int num);`
   Retorna 1 se num estiver contido em lista e 0 caso contrário.
9. `int length(int *lista);`
   Retorna o tamanho de uma ponteiro de inteiros. Considera apenas os valores antes do primeiro 0 da lista, caso haja.
10. `void append(int **copia, int valor, int local);`
    Adiciona `valor` no final do vetor `**copia[local]`.
11. `void copyList(int **copia, int *original);`
    Copia os valores de um ponteiro de inteiros para a primeira posição de um ponteiro de ponteiros inteiros. Se fez necessário usar ponteiro de ponteiro para guardar as rotas das buscas.
12. `Fila *enqueue(Fila *fila, int vertice);`
13. `Fila *dequeue(Fila **fila);`
14. `void showFila(Fila *fila);`
15. `long getMicrotime();`
    Retorna tempo em microsegundos.

4. Lista de grafos utilizados para os testes da 5ª questão

Como a proposta da questão era o de uma viagem em um país, elaborei gráficos baseados em 10 países diferentes. Estão listado a seguir:

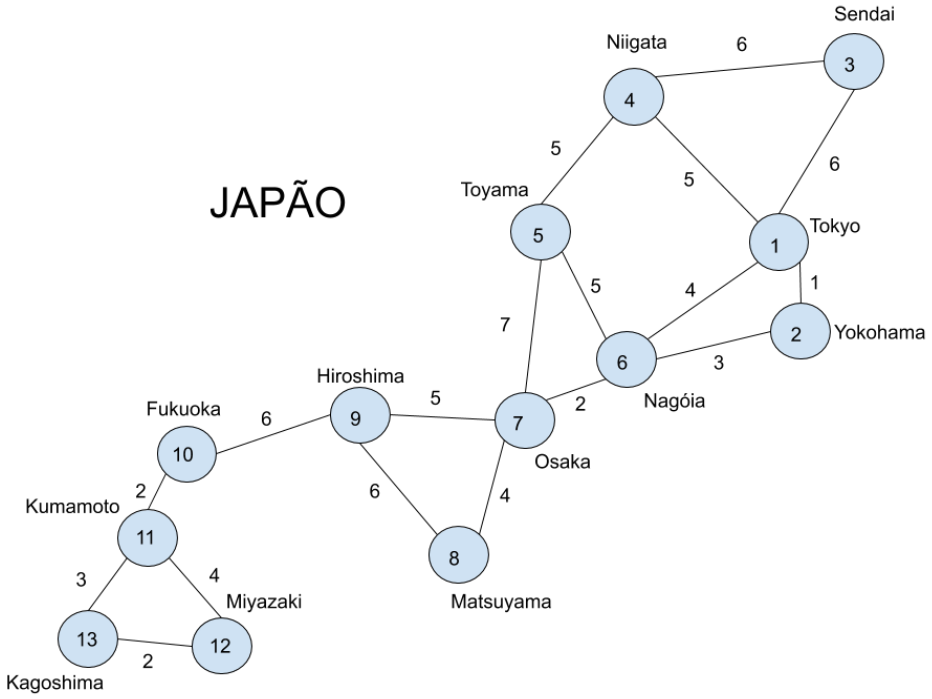

Figura 1 - Grafo representando o Japão.

Entrada do teste:

```
14 18
13 12 2
13 11 3
12 11 4
11 10 2
10 9 6
9 8 6
9 7 5
8 7 4
7 6 2
7 5 7
5 4 5
6 5 5
6 1 4
6 2 3
2 1 1
1 4 5
1 3 6
4 3 6
```

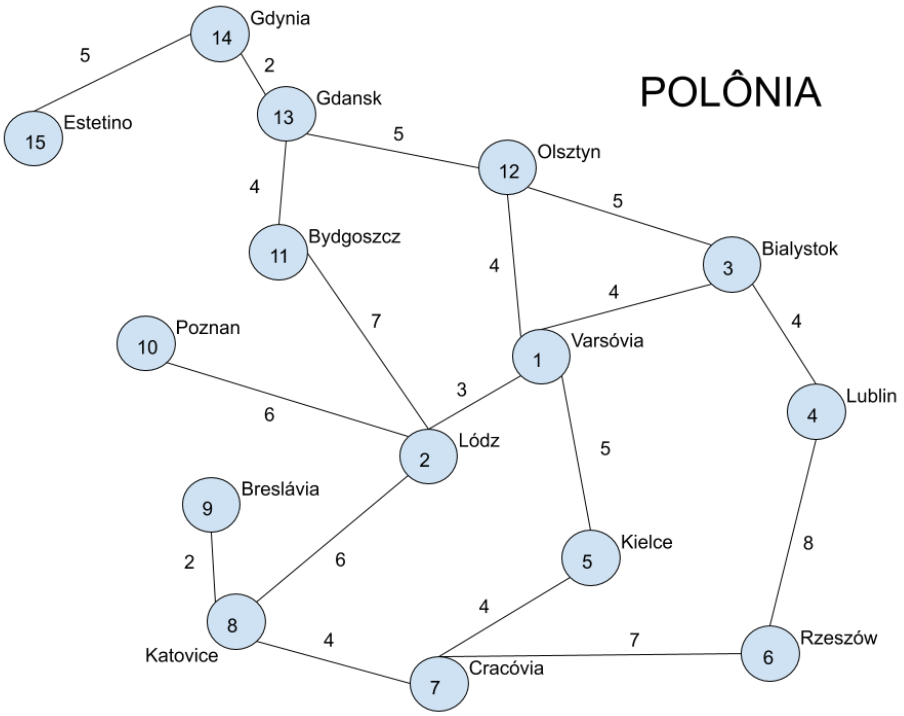

Figura 2 - Grafo representando a Polônia.

Entrada do teste:

15 18
9 8 2
8 7 4
8 2 6
7 5 4
7 6 7
2 10 8
2 11 7
2 1 3
5 1 5
6 4 8
11 13 4
1 12 4
1 3 4
4 3 4
13 14 2
12 13 5
12 3 5
14 15 5

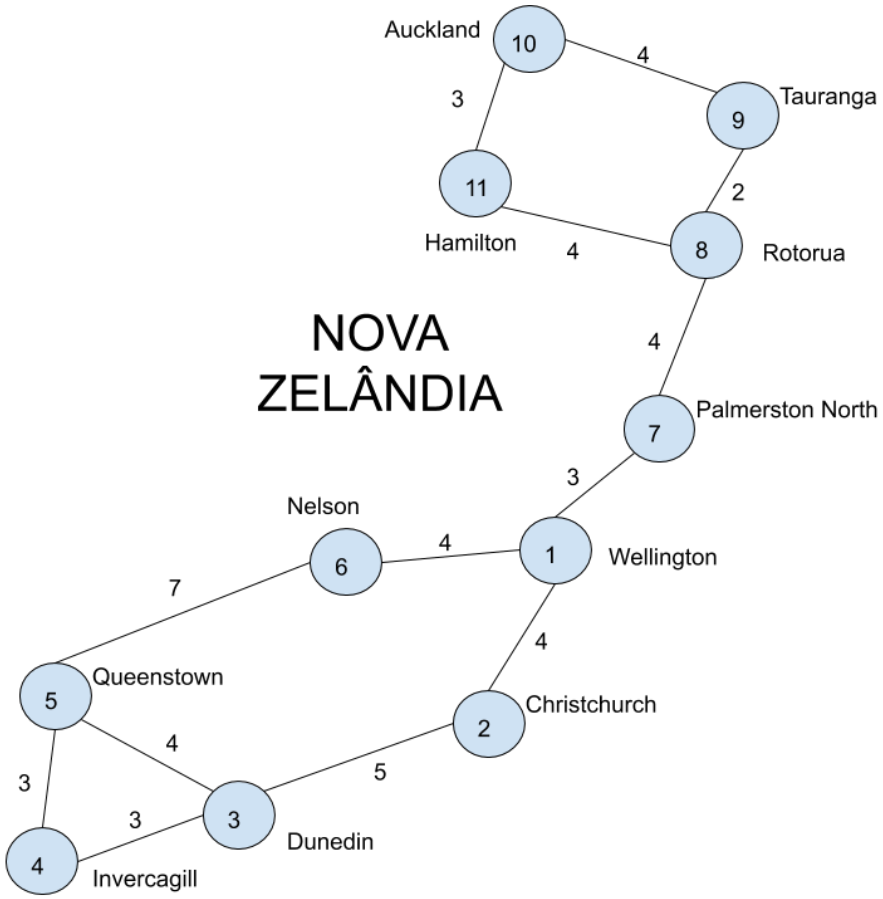

Figura 3 - Grafo representando a Nova Zelândia.

Entrada do teste:

11 13
1 2 4
1 6 4
1 7 3
2 3 5
6 5 7
3 5 4
3 4 3
5 4 3
7 8 4
8 11 4
8 9 2
11 10 3
9 10 4

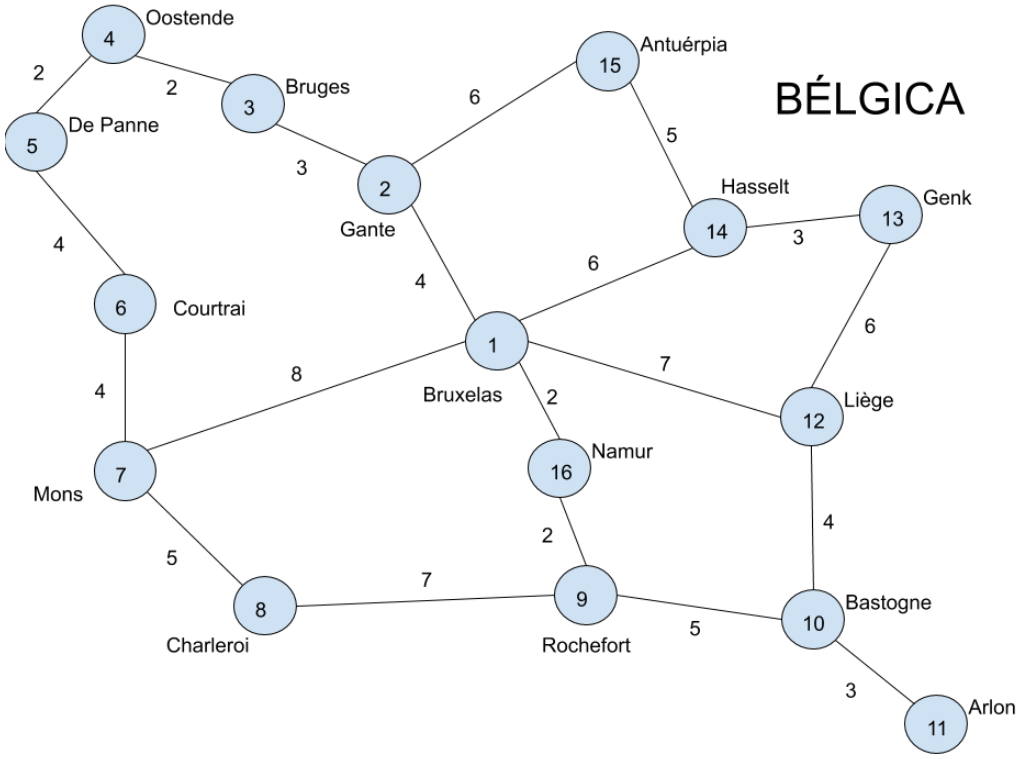

Figura 4 - Grafo representando a Bélgica.

Entrada do teste:

16 20
11 10 3
10 9 5
10 12 4
9 16 2
9 8 7
16 1 2
8 7 5
12 1 7
7 1 8
12 13 6
13 14 3
14 1 6
14 15 5
15 2 6
7 6 4
6 5 4
5 4 2
4 3 2
3 2 3
2 1 4

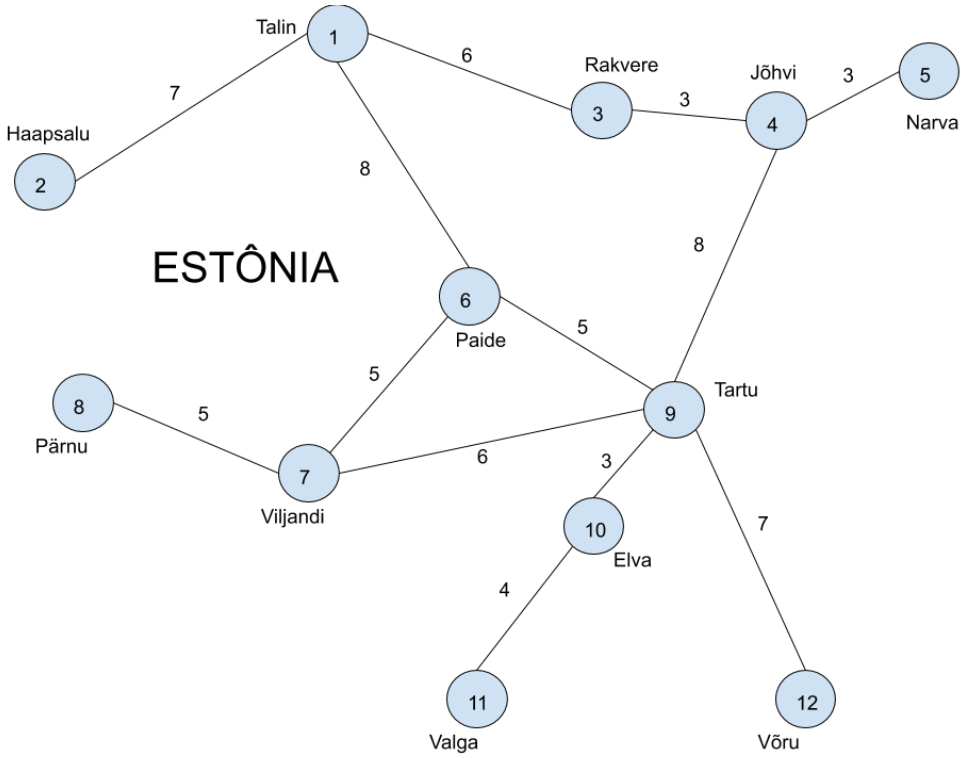

Figura 5 - Grafo representando a Estônia.

Entrada do teste:

12 13
1 2 7
1 6 8
1 3 6
6 7 5
6 9 5
7 8 5
7 9 6
3 4 3
4 5 3
4 9 8
9 10 3
9 12 7
10 11 4

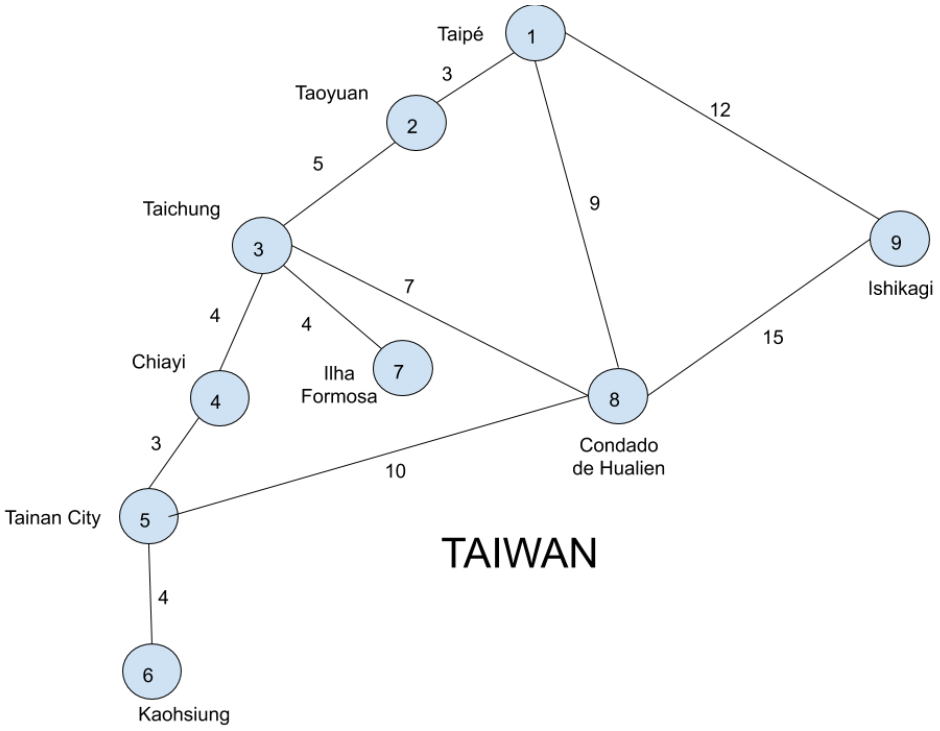

Figura 6 - Grafo representando Taiwan.

Entrada do teste:

9 11
1 2 3
1 9 12
1 8 9
2 3 5
9 8 15
8 5 10
3 8 7
3 7 4
3 4 4
4 5 3
5 6 4

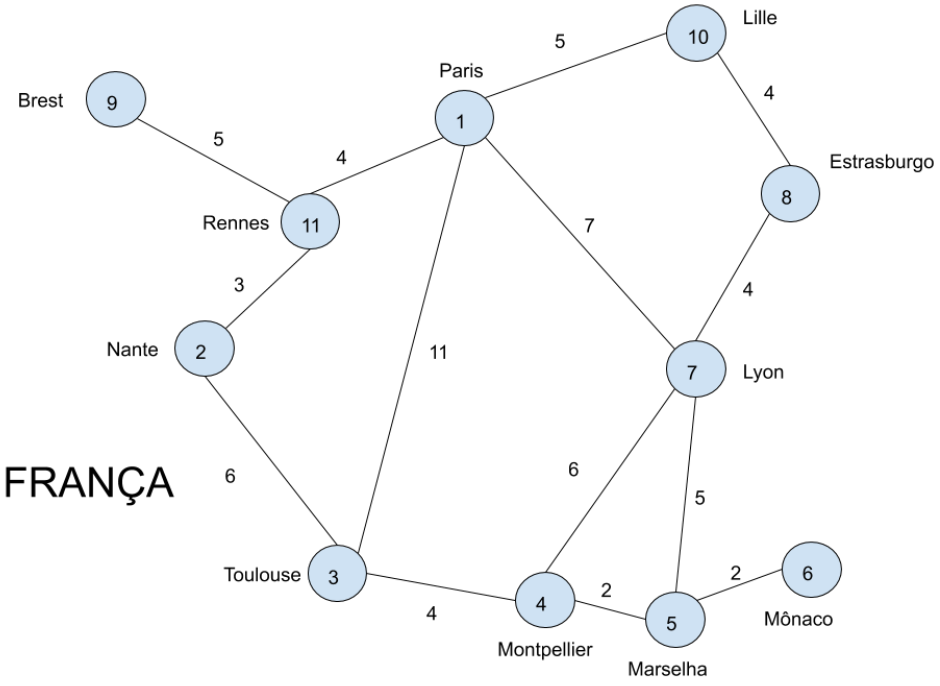

Figura 7 - Grafo representando a França.

Entrada do teste:

11 14
1 11 4
1 3 11
1 7 7
1 10 5
11 9 5
11 2 3
3 2 6
3 4 4
7 4 6
7 5 5
7 8 4
10 8 4
4 5 2
5 6 2

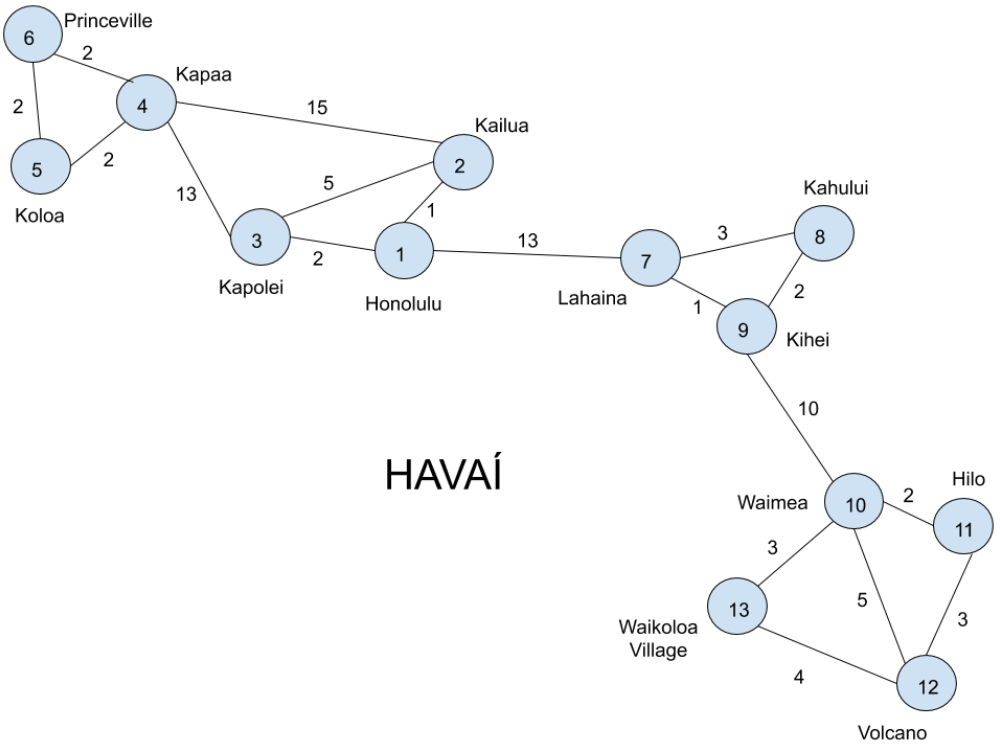

Figura 8 - Grafo representando a França.

Entrada do teste:

13 18
1 2 1
1 3 2
1 7 13
2 4 15
2 3 5
3 4 13
7 8 3
7 9 1
4 5 2
4 6 2
8 9 2
5 6 2
9 10 10
10 11 2
10 12 5
10 13 3
11 12 3
12 13 4

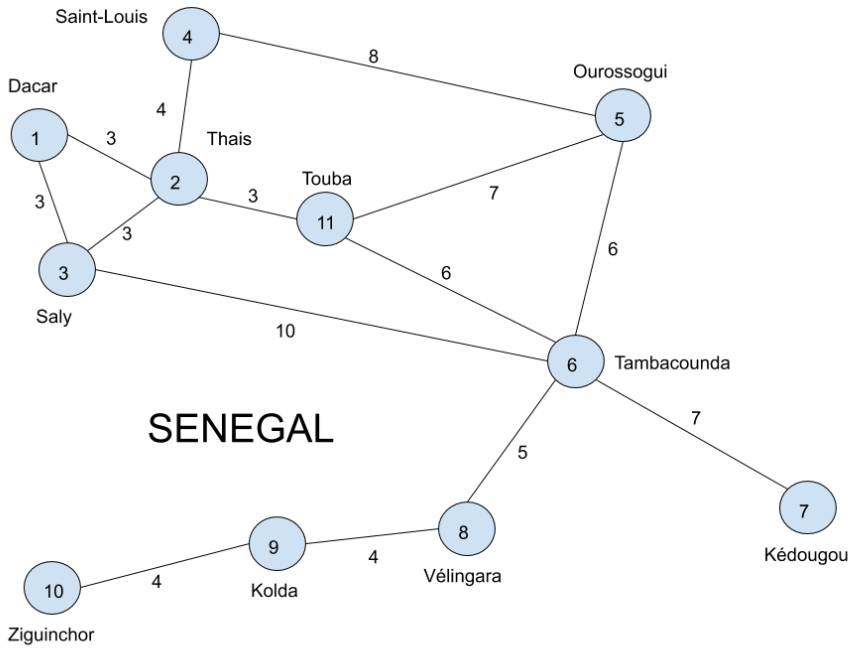

Figura 9 - Grafo representando o Senegal.

Entrada do teste:

11 14
1 2 3
1 3 3
2 3 3
2 4 4
2 11 3
3 6 10
4 5 8
11 5 7
11 6 6
5 6 6
6 7 7
6 8 5
8 9 4
9 10 4

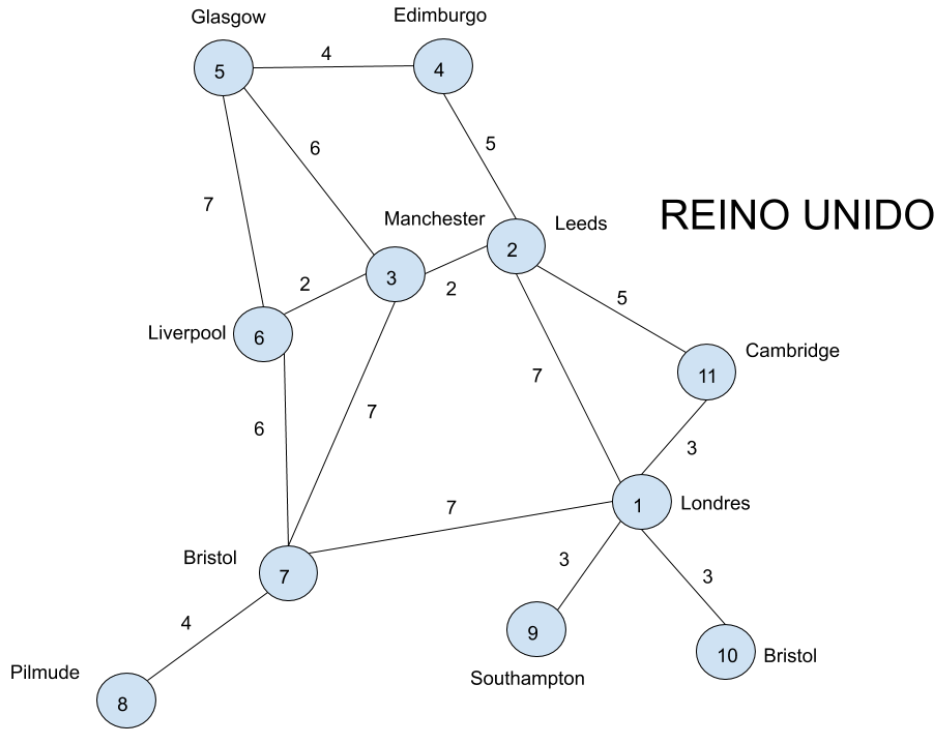

Figura 10 - Grafo representando o Senegal.

Entrada do teste:

11 15
1 7 7
1 9 3
1 10 3
1 11 3
1 2 7
7 8 4
7 6 6
7 3 7
11 2 5
6 5 7
6 3 2
2 3 2
2 4 5
5 3 5
5 4 4