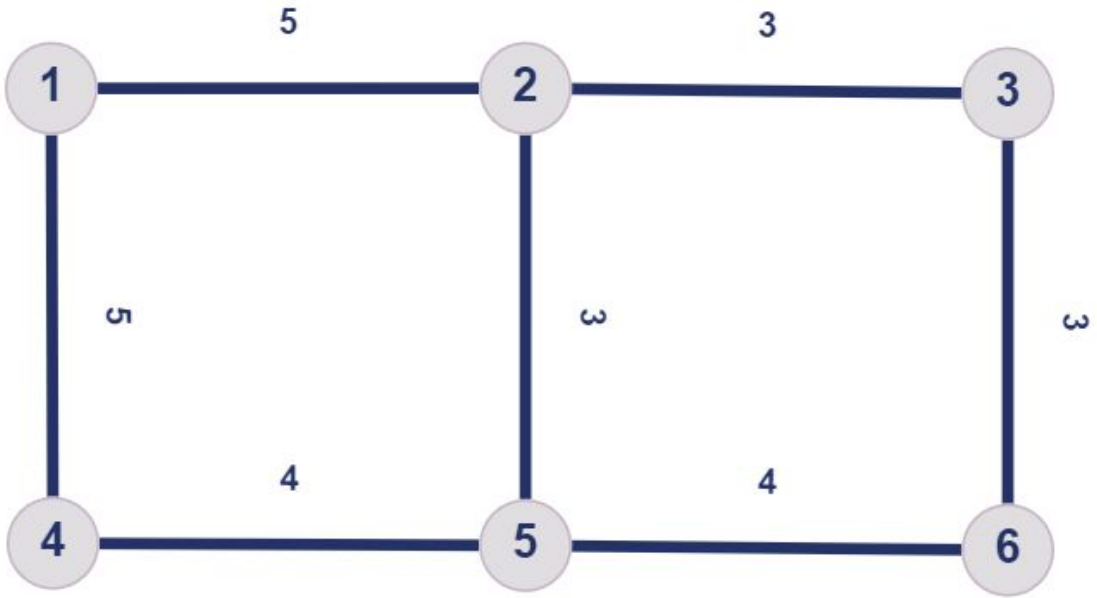
Resolução do problema EP05

Geraldo Rodrigues de Melo Neto
Gustavo Duarte Ventino
Maria Luisa Gabriel Domingues
Pedro de Araújo Ribeiro
Lucas Marques Pinho Tiago

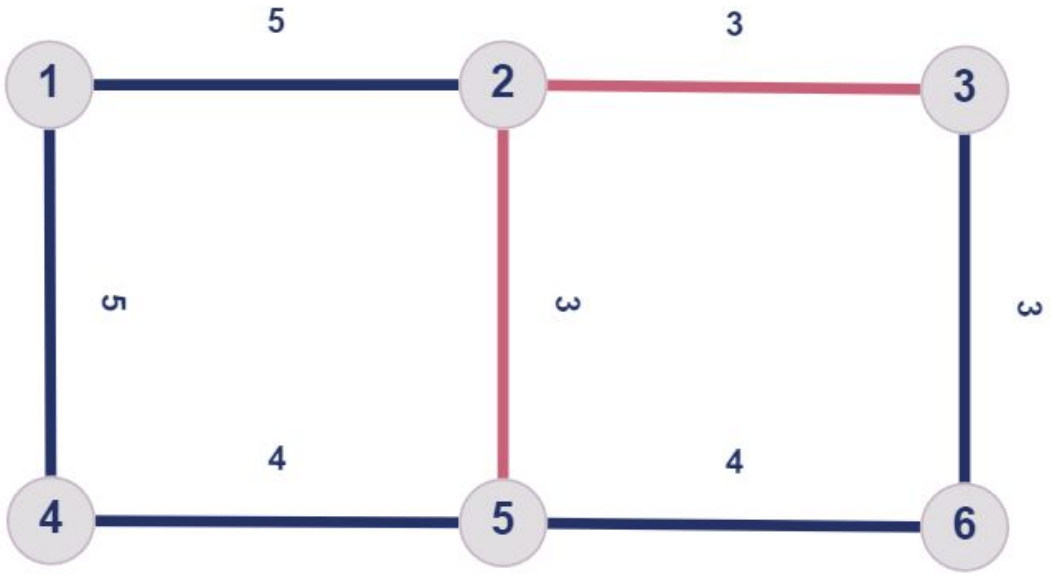
O Problema:

Em uma longa pista de corrida, temos um aumento corridas noturnas ilegais ocorrendo nela. Assim, temos que instalar câmeras de monitoramento nas pista de forma com que temos pelo menos uma câmera em cada percurso possível.

Exemplo:



Exemplo:



Entradas:

A entrada consiste em diversas verificações seguindo o modelo:

- Número de verificações **c**;
- Dois inteiros positivos **n** e **m**, onde **n** é o número de junções e **m** o número de ruas
- Próximas **m** linhas descrevem o caso a ser verificado que contém três inteiros positivos na qual:
  - Número da primeira junção;
  - Número da segunda junção;
  - Custo de se colocar uma câmera;

Observação: Cidades são numeradas de 1 a **n**.

Saídas:

A saída mostrará **c** linhas na qual cada uma representa uma verificação que consistem em um número inteiro positivo com o custo mínimo de se colocar câmeras na pista.

Nossa Resolução:

Decidimos resolver esse problema com Grafos pois podemos representar cada junção como vértices de um grafo e suas ruas como arestas, sendo o peso de cada aresta o custo para instalar a câmera.

Definimos dois TADs nos quais estão, respectivamente, classes presentes: uma para o grafo e outra para os vértices do grafo.

Nossa Resolução:

- Graph.hpp : contém, de atributos, um array com os vértices do grafo, além do número de vértices.
  - De métodos temos um método para a geração do grafo, um método de solução e impressão da solução.
- Vertex.hpp: contém, de atributos, um par de id de vértices (gerado após input) e o tamanho da aresta.
  - De métodos temos apenas print, que imprime o vértice dado.

```cpp
#include "Vertex.hpp"
#include <fstream>
#include <queue>
class Graph
{
public:
  std::vector<Vertex*> vertices;
  int size;

//Construtores:
  Graph();
  Graph(int size);

//Faz toda a leitura de entrada e cria o grafo:
  static Graph* readGraph();

//soluçaõ do problema, retorna o preço das cameras
  int solve();

//Imprime todo o grafo:
  void print();

//Imprime a resposta
  void printResul();
};
```

```cpp
#include <vector>
#include <string>
class Vertex
{
public:
  //Atributos
  std::pair<int,int> aresta;
  int size;

  //Construtores:
  Vertex();
  Vertex(int id1, int id2);
  Vertex(int id1, int id2, int size);

  //Printa as informacoes de id e valor do vertice atual
  void print();
};
```

Método para resolução do problema

Para resolver o problema, primeiro fizemos a leitura do grafo a partir da função readGraph(), e depois procuramos círculos dentro do grafo para encontrar cada percurso e contamos as menores arestas de cada círculo encontrado por meio do algoritmo de Kruskal.

A segunda parte da resolução está disponível a partir da função solve().

Leitura de dados:

```
12  // Faz toda a leitura de entrada e cria o grafo:
13  Graph * Graph::readGraph( ) {
14    int vertex = -1;
15    int n = 0, m = 0;
16    int v1 = 0, v2 = 0, size = 0;
17    Vertex *v;
18    Graph *g;
19    std::cin >> n;
20    std::cin >> m;
21    g = new Graph(n);
22    for(int i = 0; i < m; i++){
23      std::cin >> v1;
24      std::cin >> v2;
25      std::cin >> size;
26      v = new Vertex(v1-1,v2-1,size);
27      g->vertices.push_back(v);
28    }
29    //Ordena os vertices
30    std::sort(g->vertices.begin( ), g->vertices.end( ),
31              [](Vertex *e1, Vertex *e2)
32              { return e1->aresta.second > e2->aresta.second; });
33    std::stable_sort(g->vertices.begin( ), g->vertices.end( ),
34              [](Vertex *e1, Vertex *e2)
35              { return e1->aresta.first > e2->aresta.first; });
36    std::stable_sort(g->vertices.begin( ), g->vertices.end( ),
37              [](Vertex *e1, Vertex *e2)
38              { return e1->size > e2->size; });
39    return g;
40  }
```

Resolução:

```
42  // Resolução do problema
43  int Graph::solve() {
44    int vet[size];
45    int cnt = 0;
46    //inicionalizamos o vetor de "pais" com -1
47    //-1 indica que o vértice não foi inserido no grafo
48    for(int j = 0 ;j<size;j++){
49      vet[j]=-1;
50    }
51    for(auto i : vertices){
52      if((vet[i->aresta.first]==-1)&&(vet[i->aresta.second]==-1)){
53        vet[i->aresta.first]=i->aresta.first;
54        vet[i->aresta.second]=i->aresta.first;
55      }
56      else if(vet[i->aresta.first]==vet[i->aresta.second]){
57        cnt = cnt + i->size;
58      }
59      else if(vet[i->aresta.first]==-1){
60        vet[i->aresta.first]=vet[i->aresta.second];
61      }
62      else if(vet[i->aresta.second]==-1){
63        vet[i->aresta.second]=vet[i->aresta.first];
64      }
65      else{
66        int a,b;
67          a=vet[i->aresta.first];
68          b=vet[i->aresta.second];
69        for(int j = 0;j<size;j++)
70          if(vet[j]==a)
71            vet[j]=b;
72      }
73    }
74    return cnt;
75  }
```

FIM

Referências:

Repositório com os códigos fonte: https://replit.com/@Gezero/S06EP05

Imagens de grafos tiradas de: https://graphonline.ru/en/#