Resolução do problema EP04

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

O Problema:

É necessário analisar a contagem de bolsas de petróleo distribuídas em uma determinada área representada por uma matriz de j linhas por k colunas onde cada coordenada pode ou não conter petróleo. Se duas coordenadas vizinhas (cima, baixo, direita, esquerda e diagonais) possuem petróleo elas contam como uma única bolsa.

Entradas:

A entrada consiste em, nessa ordem:

- Quantidade de linhas e de colunas da matriz, nessa ordem.
- N (sendo N = j x k) caracteres representativos da presença de petróleo, sendo * para coordenadas sem e @ para coordenadas com.
- 0 0 para encerrar o programa ou as dimensões da próxima matriz.

Saídas:

O programa deve imprimir a quantidade de bolsas de petróleo de todas as matrizes lidas em ordem.

Nossa Resolução:

Decidimos resolver esse problema com Grafos pois as relações de vizinhança entre as coordenadas da matriz podem ser expressas como as adjacências de um grafo.

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 e arestas.
  - De métodos, contém Getters e Setters básicos para cada atributo, o método de geração do grafo e dois métodos referentes a busca e contagem de bolsas de petróleo.
- Vertex.hpp: contém, de atributos, um array de vértices adjacentes, o valor do vértice (dado no input) e seu id.
  - De métodos, contém Getters e Setters básicos para cada atributo, além de métodos como addToAdjacency(), que inserem vértices no array de adjacência, gerando novas arestas.

```cpp
#include <vector>
class Vertex
{
private:
  int id;
  int value;
  std::vector<Vertex*> adjacency;
public:
  //Construtores:
  Vertex();
  Vertex(int id, int value);
  //Getters:
  int getId();
  int getValue();
  std::vector<Vertex*> getAdjacency();
  //Setters:
  void setId(int id);
  void setValue(int value);
  //Adiciona o vertice v na lista de adjacencia
  void addToAdjacency(Vertex *v);
  //Printa as informacoes de id e valor do vertice atual
  void print();
  //Printa a lista de adjacencia do vertice atual
  void printAdjacency();
};
```

```cpp
#include "Vertex.hpp"

class Graph
{
private:
  std::vector<Vertex*> vertices;
  int size;
  int edges;
public:
//Construtores:
  Graph();
  Graph(int size, int edges);
//Faz toda a leitura de entrada e cria o grafo:
  static Graph* readGraph(int j, int k);
//Adiciona novo vertice na lista de vertices:
  void addVertex(Vertex* v);
//Setter de arestas
  void setEdges(int edges);
//Conta a quantidade de vértices de valor 1
  int busca();
//Busca vértices adjacentes ao id que possuem valor 1
  void contaAdj(int id,bool vet[]);
//Dado um id retorna o vertice naquela posição:
  Vertex* getVertex(int id);
//Imprime todo o grafo:
  void print();
};
```

Método para resolução do problema:

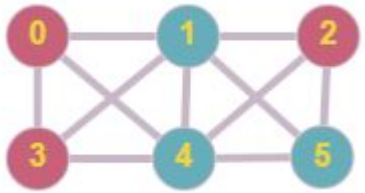
Optamos por dividir o problema em duas etapas, a leitura do grafo e atribuição das relações de adjacência, em seguida a contagem de bolsas de petróleo tendo em consideração que duas coordenadas adjacentes marcadas contam como uma única bolsa.

Primeiro lê-se todos os valores dos caracteres

```
34  Graph* Graph::readGraph(int j, int k)
35  {
36    int n = 0, r=0, cnt = 0 , m = 0;
37    char value;
38    Vertex *v;
39    Graph *g;
40
41    n = j*k;
42
43    g = new Graph(n,m);
44
45    for (int i = 0; i < n; i++) {
46      std::cin >> value;
47      if(value=='*'){
48        v = new Vertex(i,0);
49        g->addVertex(v);
50      }
51      else if(value=='@'){
52        v = new Vertex(i,1);
53        g->addVertex(v);
54      }
55    }
```

Método para resolução do problema:

Depois, verificamos para cada vértice sua posição relativa aos limites das dimensões da matriz e baseado nisso atribuímos as adjacências.




```
56    for (int i = 0; i < j; i++)
57    {
58      for (int f = 0; f < k; f++){
59        r = (i*k)+f;
60      if(f!=0){
61        g->getVertex(r)->addToAdjacency(g->getVertex(r-1));
62        cnt++;
63      }
64      if(f!=k-1){
65        g->getVertex(r)->addToAdjacency(g->getVertex(r+1));
66        cnt++;
67      }
68      if(i!=0){
69        g->getVertex(r)->addToAdjacency(g->getVertex(r-k));
70        cnt++;
71      }
72      if(i!=(j-1)){
73        g->getVertex(r)->addToAdjacency(g->getVertex(r+k));
74        cnt++;
75      }
76      if((f!=0)&&(i!=0)){
77        g->getVertex(r)->addToAdjacency(g->getVertex(r-1-k));
78        cnt++;
79      }
80      if((f!=0)&&(i!=j-1)){
81        g->getVertex(r)->addToAdjacency(g->getVertex(r-1+k));
82        cnt++;
83      }
```

Método para resolução do problema:

Em seguida começamos a contagem de vértices com valor 1, se encontrado é chamada a função que verifica de alguma das adjacências também possui valor 1

```
int Graph::busca(){
  int cnt = 0;
  bool vis[size];
  for(int i=0; i<size;i++){
    vis[i] = false;
  }

  for(int i=0;i<size;i++){
    if(vertices[i]->getValue()!=0){
      if(!vis[i]){
        cnt++;}
      vis[i]=true;
      contaAdj(i, vis);
    }
  }
  return cnt;
}
```

Método para resolução do problema:

Ao encontrar algum vizinho que também possui valor 1, sua posição é marcada como visitada e a função é chamada recursivamente para este vizinho até que não hajam mais coordenadas adjacentes de valor 1.

```
118 void Graph::contaAdj(int id, bool vet[]){
119   for(auto i : vertices[id]->getAdjacency()){
120     if((i->getValue()!=0)&&(!vet[i->getId()])){
121       vet[i->getId()]=true;
122       contaAdj(i->getId(),vet);}
123   }
124 }
```

Método para resolução do problema:

Todo o processo anterior é repetido na main até que seja lido o input 0 0 e em seguida a lista de contagens é impressa

```cpp
#include <iostream>
#include "Graph.hpp"

int main()
{
  std::vector<int> vet;
  int cnt, j, k;
  std::cin >> j; //linhas
  std::cin >> k; //colunas
  while((k!=0)&&(j!=0)){
  Graph* graph = Graph::readGraph(j, k);
  cnt = graph->busca();
    vet.push_back(cnt);
  std::cin >> j;
  std::cin >> k;
  }
  for(auto i : vet)
    std::cout << i << std::endl;
  return 0;
}
```

Referências:

Imagem do grafo tirada de: https://graphonline.ru/en/#

Repositório com os códigos fonte: https://replit.com/@PedroRibeiroA12/TG-EP04#main.cpp