ESTRUTURAS DE DADOS

Grafos (implementação)

Roteiro

- **Estrutura do Nó**
- **Tipo Abstrato de Dados**
- **Detalhes de Implementação**

Estrutura do Nó

```
class Vertex {
 public:
  Vertex(){
    this->nome = "";
  }
  Vertex(std::string nome){
    this->nome  = nome;
  }
  std::string getNome() const {
    return nome;
  }
 private :
  std::string nome;
};
```

Usaremos matrizes de adjacências, então a representação das informações é simples.

Tipo Abstrato de Dados

Implementaremos uma estrutura minimalista.

- **Internamente, representaremos da seguinte forma:**

```
class Graph {
 private:
  int NULL_EDGE; // Constante para aresta nula.
  int maxVertices; // Número máximo de vértices.
  int numVertices; // Número de vértices adicionados.
  Vertex* vertices; // Array com todos os vértices.
  int** edges; // Matriz de adjacências
  bool* marks; // marks[i] marca se vertices[i] foi usado.
  int getIndex(Vertex);
```

A interface pública contém:

```
 public:
  Graph(int max = 50, int null = 0);  // construtor
  ~Graph(); // destrutor

  void addVertex(Vertex);
  void addEdge(Vertex, Vertex, int);

  int getWeight(Vertex, Vertex);
  void getAdjacents(Vertex, Queue&);
  void clearMarks();
  void markVertex(Vertex);
  bool isMarked(Vertex);
  void printMatrix();
};
```

Usaremos Queue para criar uma lista de adjacentes.

Detalhes de Implementação

```
Graph::Graph(int max, int null_edge) {
  NULL_EDGE   = null_edge;
  maxVertices = max;
  numVertices = 0;
  vertices = new Vertex[maxVertices];
  marks = new bool[maxVertices];
  edges = new int* [maxVertices];

  // Criando matriz de adjacências
  for (int row = 0; row < maxVertices; row++)
    edges[row] = new int[maxVertices];
  // Populando matriz de adjacências com valor nulo
  for (int row = 0; row < maxVertices; row++)
    for (int col = 0; col < maxVertices; col++)
      edges[row][col] = NULL_EDGE;
}
```

A inicialização criará as estruturas de vértices e adjacências.

Note que criamos um vetor bidimensional no construtor. Nesse caso, o destrutor precisa desalocar todas as linhas, de uma por uma.

```
Graph::~Graph() {
  delete [] vertices;
  delete [] marks;
  for (int row = 0; row < maxVertices; row++){
    delete [] edges[row];
  }
  delete [] edges;
}
```

Em geral, verificamos se quantidade de new é a mesma da de delete.

Em vários lugares, precisaremos do índice de um determinado vértice.

- O método privado **getIndex** cumpre esse papel.

```
int Graph::getIndex(Vertex vertex) {
  int index = 0;
  while (!(vertex.getNome() == vertices[index].getNome())){
    index++;
  }
  return index;
}
```

- Note que fazemos apenas uma busca sequencial, outras opções poderiam ser mais eficientes. Entretanto, complicariam o código.

```
void Graph::addVertex(Vertex vertex){
  vertices[numVertices] = vertex;
  numVertices++;
}
void Graph::addEdge(Vertex fromVertex,
                    Vertex toVertex,
                    int weight){
  int row = getIndex(fromVertex);
  int col = getIndex(toVertex);

  edges[row][col] = weight;
  // Remover se grafo direcionado.
  edges[col][row] = weight;
}
```

Para adicionar vértices e arestas, basta acessar o vetor ou a matriz correspondente.

No método getWeight, iremos simplesmente retornar o valor que está na matriz de adjacência.

- **Uma outra ideia seria lançar um erro caso o valor na matriz fosse igual a NULL_EDGE.**

```
int Graph::getWeight(Vertex fromVertex,
                     Vertex toVertex){
  int row = getIndex(fromVertex);
  int col = getIndex(toVertex);
  return edges[row][col];
}
```

Para obter a lista de adjacentes a um dado vértice, usaremos por comodidade uma estrutura que já aprendemos em aulas anteriores, a fila.

```
void Graph::getAdjacents(Vertex vertex,
                         Queue &adjVertices){
  int fromIndex;
  int toIndex;
  fromIndex = getIndex(vertex);
  for (toIndex = 0; toIndex < numVertices; toIndex++)
    if (edges[fromIndex][toIndex] != NULL_EDGE)
      // Uma cópia do elemento é adicionada no array.
      adjVertices.enqueue(vertices[toIndex]);
}
```

Alguns métodos gerenciam marcações nos vértices.

- São úteis para vários algoritmos de busca que precisam verificar se um determinado vértice já foi visitado.

```
void Graph::clearMarks(){
  for (int i = 0; i < numVertices; i++)
    marks[i] = false;
}
void Graph::markVertex(Vertex vertex){
  int index = getIndex(vertex);
  marks[index] = true;
}
bool Graph::isMarked(Vertex vertex){
  int index = getIndex(vertex);
  return marks[index];
}
```

Um último método para imprimir a matriz de adjacências na saída padrão.

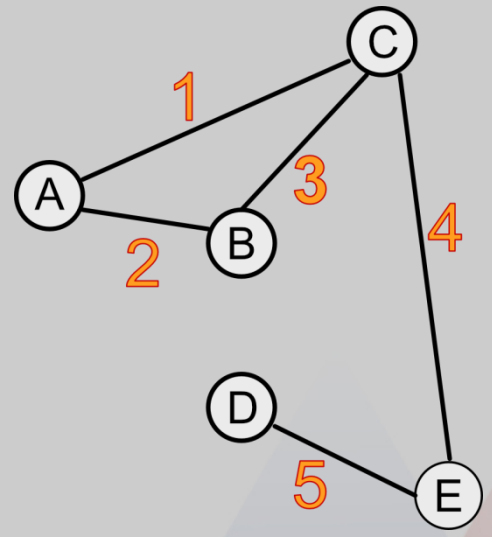
```
void Graph::printMatrix(){
  for (int i = 0; i < numVertices; i++) {
    for (int j = 0; j < numVertices; j++) {
      std::cout << edges[i][j] << ",";
    }
    std::cout << std::endl;
  }
}
```

Para usar a estrutura, basta:

```
int main() {
  Graph graph;
  Vertex a = Vertex("A"); graph.addVertex(a);
  Vertex b = Vertex("B"); graph.addVertex(b);
  Vertex c = Vertex("C"); graph.addVertex(c);
  Vertex d = Vertex("D"); graph.addVertex(d);
  Vertex e = Vertex("E"); graph.addVertex(e);

  graph.addEdge(a, b, 2);  graph.addEdge(a, c, 1);
  graph.addEdge(b, c, 3);  graph.addEdge(c, e, 4);
  graph.addEdge(d, e, 5);

  graph.printMatrix();
  std::cout << std::endl;
}
```



ESTRUTURAS DE DADOS

Grafos (implementação)