ESTRUTURAS DE DADOS

PageRank (implementação)

Roteiro

- **Noções Preliminares**
- **Detalhes de Implementação**

Noções Preliminares

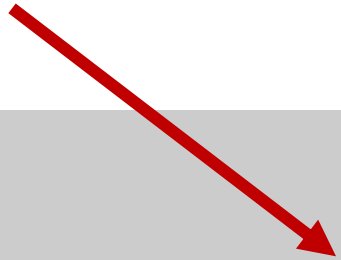
Vamos utilizar o nosso código implementado com a representação por matriz de adjacências.

O código foi feito para ser usado com grafos não direcionados. Entretanto, a mudança para grafos direcionados é simples.

- **Precisamos apenas modificar o preenchimento da matriz no momento da inserção das arestas.**

```
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;
}
```

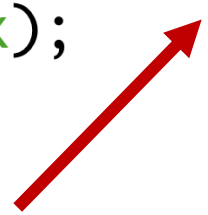


Linha Comentada

O tipo abstrato de dados também mudou.
Acrescentamos o método getPageRanks.

```
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();
 void getPageRanks(float*);
```



Novo método

Quem invoca o método getPageRanks é responsável por saber o tamanho do vetor.

```
int main() {
  Graph graph;
  Vertex a = Vertex("a"); Vertex b = Vertex("b");
  Vertex c = Vertex("c"); Vertex d = Vertex("d");
  graph.addVertex(a); graph.addVertex(b);
  graph.addVertex(c); graph.addVertex(d);
  graph.addEdge(a, c, 1); graph.addEdge(a, b, 1);
  graph.addEdge(b, d, 2); graph.addEdge(c, a, 3);
  graph.addEdge(c, b, 5); graph.addEdge(c, d, 3);

  float* pageRanks = new float[4];
  graph.getPageRanks(pageRanks);

  for (int i = 0; i < 4; i++){
    std::cout << pageRanks[i] << " , ";
  }
```

Detalhes de Implementação

Primeiro, computamos os graus de saída dos vértices.

```
void Graph::getPageRanks(float* pageRanks){
  // Computando graus de saída
  int* outputDegree = new int[numVertices];
  for (int i = 0; i < numVertices; i++) {
    outputDegree[i] = 0;
    for (int j = 0; j < numVertices; j++) {
      if (edges[i][j] != NULL_EDGE) {
        outputDegree[i] += 1;
      }
    }
  }
```

O segundo passo será a inicialização dos PageRanks.

- **Todos terão o mesmo PageRank inicialmente.**
- **A soma dos PageRanks deverá ser 1 (um).**

```
// Computando pageRanks.
float* pr_previous = new float[numVertices];
float* pr          = new float[numVertices];

// Inicializando
for (int i = 0; i < numVertices; i++) {
  pr_previous[i] = 1.0/numVertices;
}
```

No passo iterativo, atualizamos os pesos.

```
float d = .85;
for (int numIter = 0; numIter < 100; numIter++){
  // Passo Iterativo
  for (int i = 0; i< numVertices; i++){
    pr[i] = 0;
    for (int j = 0; j < numVertices; j++){
      if (edges[j][i] != NULL_EDGE){
        pr[i] += pr_previous[j]/outputDegree[j];
      }
    }
    pr[i] = (1-d)/numVertices + d*pr[i];
  }
  for (int i = 0; i< numVertices; i++){
    pr_previous[i] = pr[i];
  }
}
```

Por fim, retornamos os valores para quem chamou a função e desalocamos a memória que alocamos dinamicamente.

```
for (int i = 0; i< numVertices; i++){
  pageRanks[i] = pr[i];
}

delete [] pr_previous;
delete [] pr;
delete [] outputDegree;
```

ESTRUTURAS DE DADOS

PageRank (implementação)