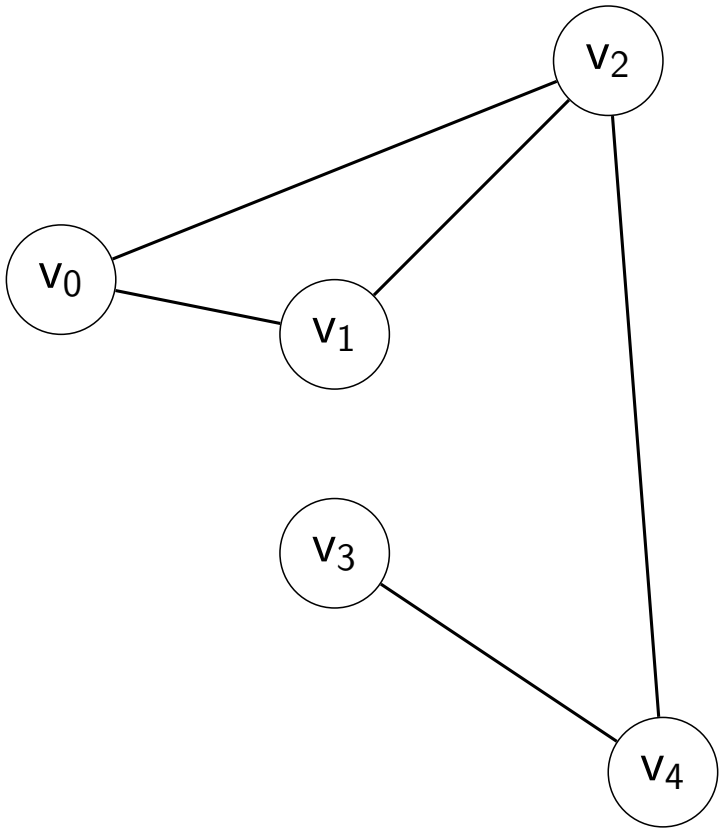
Algoritmos e Estruturas de Dados II Aula 03 – Grafos: Matriz de Adjacência

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

Grafos – Relembrando

- São representados como um conjunto de nós (vértices) conectados par a par por linhas (arestas)



Grafos - Relembrando

Podem ser representados utilizando:

Grafos - Relembrando

Podem ser representados utilizando:

- Matrizes de Adjacências

Grafos - Relembrando

Podem ser representados utilizando:

- Matrizes de Adjacências
- Listas de Adjacências

Grafos - Relembrando

Podem ser representados utilizando:

- **Matrizes de Adjacências**
- Listas de Adjacências

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta
- Remover uma Aresta

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta
- Remover uma Aresta
- Verificar se uma Aresta Existe

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta
- Remover uma Aresta
- Verificar se uma Aresta Existe
- Retornar o Número de Vértices

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta
- Remover uma Aresta
- Verificar se uma Aresta Existe
- Retornar o Número de Vértices
- Retornar o Número de Arestas

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta
- Remover uma Aresta
- Verificar se uma Aresta Existe
- Retornar o Número de Vértices
- Retornar o Número de Arestas
- Verifica se um Vértice Possui Vizinhos

Matriz de Adjacência

Definiremos as estruturas de dados para representar um grafo, bem como algoritmos para:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta
- Remover uma Aresta
- Verificar se uma Aresta Existe
- Retornar o Número de Vértices
- Retornar o Número de Arestas
- Verifica se um Vértice Possui Vizinhos
- Retornar o Grau de um Vértice

Matriz de Adjacência Binária

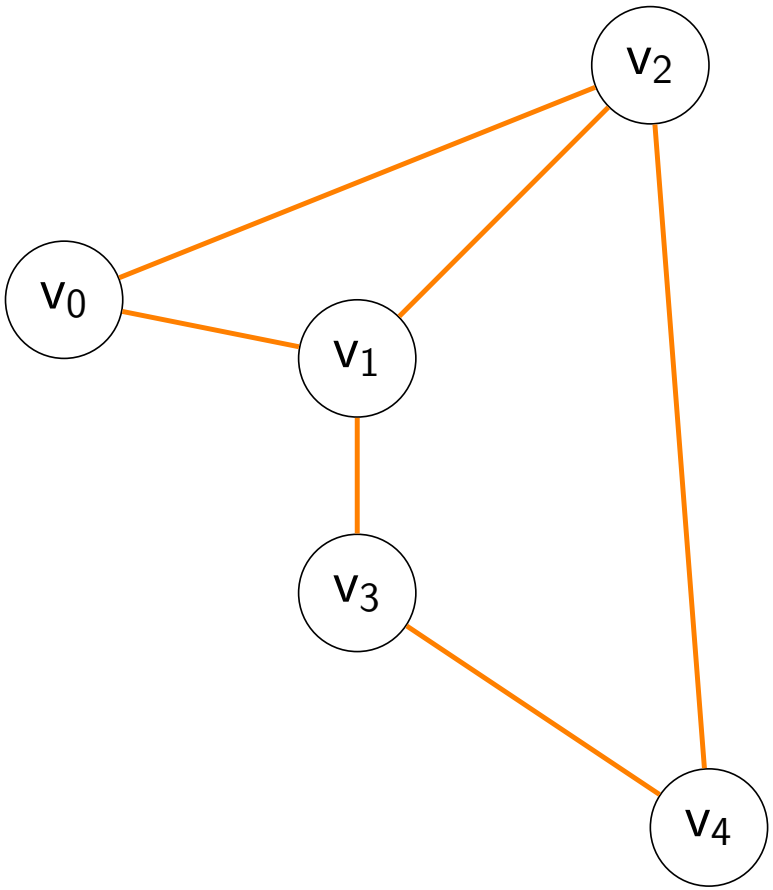
- Uma matriz de adjacências A de um grafo com n vértices é uma matriz $n \times n$ de bits, em que:

Matriz de Adjacência Binária

- Uma matriz de adjacências A de um grafo com n vértices é uma matriz $n \times n$ de bits, em que:
  - $A[i, j] = 1$ se houver uma aresta indo do vértice i para o vértice j no grafo.
  - $A[i, j] = 0$ se não houver aresta indo de i para j

Grafos – Matrizes de Adjacências

- Grafo não ponderado e não dirigido:



	v_0	v_1	v_2	v_3	v_4
v_0	0	1	1	0	0
v_1	1	0	1	1	0
v_2	1	1	0	0	1
v_3	0	1	0	0	1
v_4	0	0	1	1	0

Representação

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;
```

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

Representação

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;
```

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

Representação

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;
```

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

Representação

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;
```

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

Representação

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;
```

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

Representação

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;
```

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

Inicialização

```
bool inicializaGrafo(Grafo* g, int vertices){

}
```

Inicialização

```
bool inicializaGrafo(Grafo* g, int vertices){
  if (g==NULL || vertices<1) return false;
```

```
}
```

Inicialização

```
bool inicializaGrafo(Grafo* g, int vertices){
  if (g==NULL || vertices<1) return false;
  g->numVertices = vertices;
  g->numArestas = 0;


}
```

Inicialização

```
bool inicializaGrafo(Grafo* g, int vertices){
  if (g==NULL || vertices<1) return false;
  g->numVertices = vertices;
  g->numArestas = 0;
  int x, y;
  g->matriz = (bool**) malloc(sizeof(bool*)*vertices);




}
```

Inicialização

```
bool inicializaGrafo(Grafo* g, int vertices){
  if (g==NULL || vertices<1) return false;
  g->numVertices = vertices;
  g->numArestas = 0;
  int x, y;
  g->matriz = (bool**) malloc(sizeof(bool*)*vertices);
  for (x=0; x<vertices; x++){
    g->matriz[x] = (bool*)malloc(sizeof(bool)*vertices);




  }

}
```

Inicialização

```
bool inicializaGrafo(Grafo* g, int vertices){
  if (g==NULL || vertices<1) return false;
  g->numVertices = vertices;
  g->numArestas = 0;
  int x, y;
  g->matriz = (bool**) malloc(sizeof(bool*)*vertices);
  for (x=0; x<vertices; x++){
    g->matriz[x] = (bool*)malloc(sizeof(bool)*vertices);
    for (y=0; y<vertices; y++){
      g->matriz[x][y] = false;
    }
  }

}
```

Inicialização

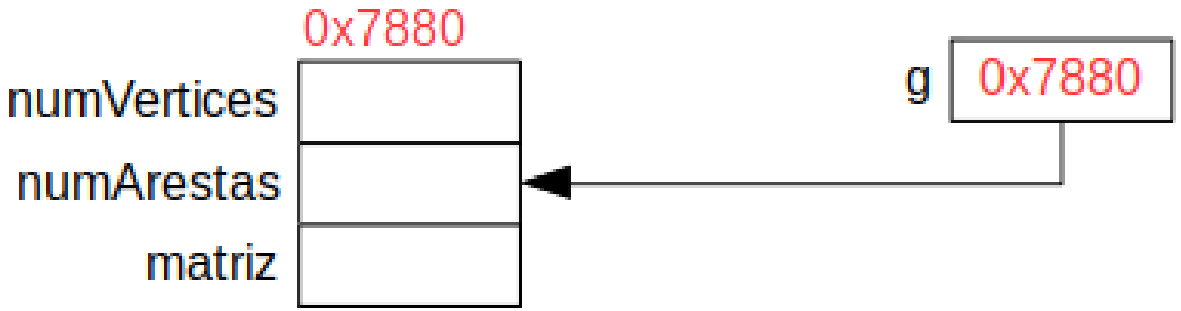
```
bool inicializaGrafo(Grafo* g, int vertices){
  if (g==NULL || vertices<1) return false;
  g->numVertices = vertices;
  g->numArestas = 0;
  int x, y;
  g->matriz = (bool**) malloc(sizeof(bool*)*vertices);
  for (x=0; x<vertices; x++){
    g->matriz[x] = (bool*)malloc(sizeof(bool)*vertices);
    for (y=0; y<vertices; y++){
      g->matriz[x][y] = false;
    }
  }
  return true;
}
```

Inicialização

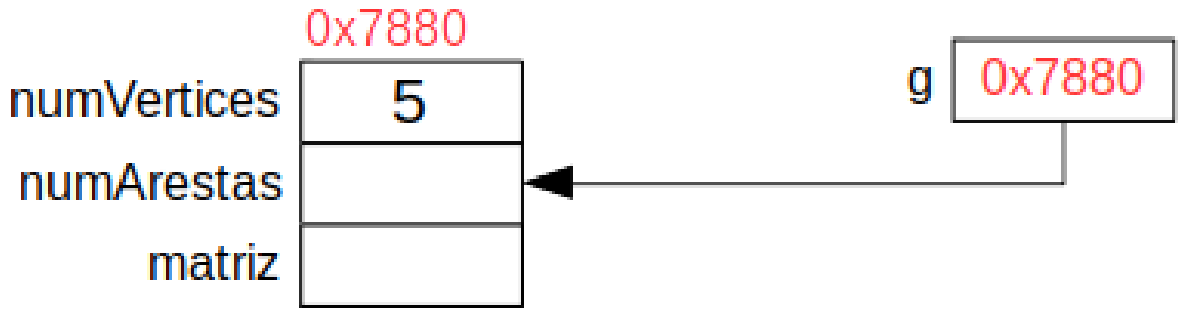
```
bool inicializaGrafo(Grafo* g, int vertices){
  if (g==NULL || vertices<1) return false;
  g->numVertices = vertices;
  g->numArestas = 0;
  int x, y;
  g->matriz = (bool**) malloc(sizeof(bool*)*vertices);
  for (x=0; x<vertices; x++){
    g->matriz[x] = (bool*)malloc(sizeof(bool)*vertices);
    for (y=0; y<vertices; y++){
      g->matriz[x][y] = false;
    }
  }
  return true;
}
```

$O(V^2)$

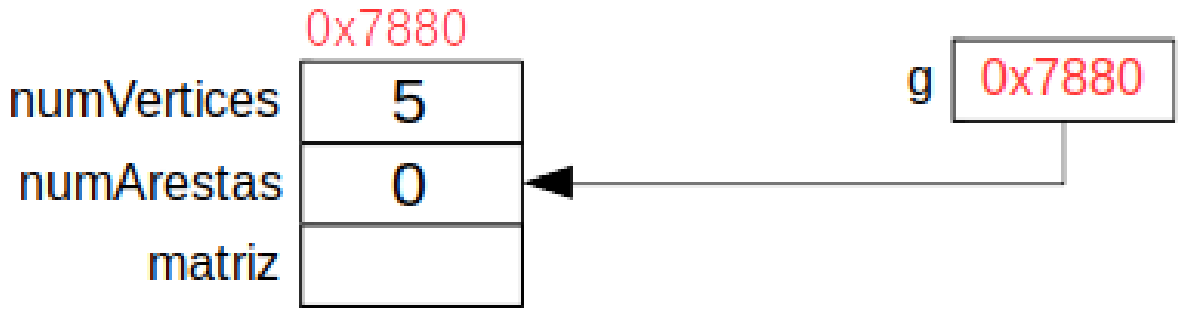
Inicialização



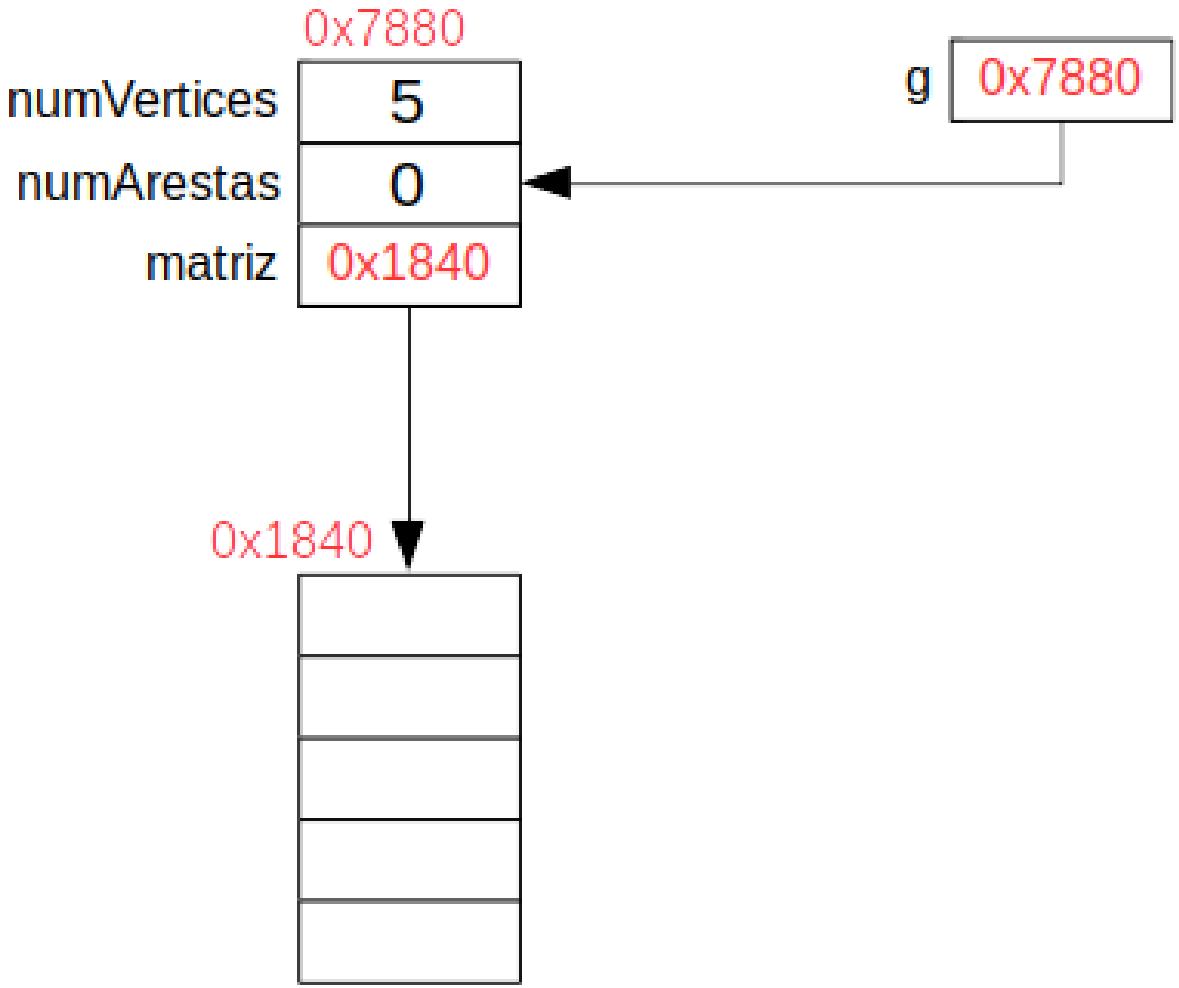
Inicialização



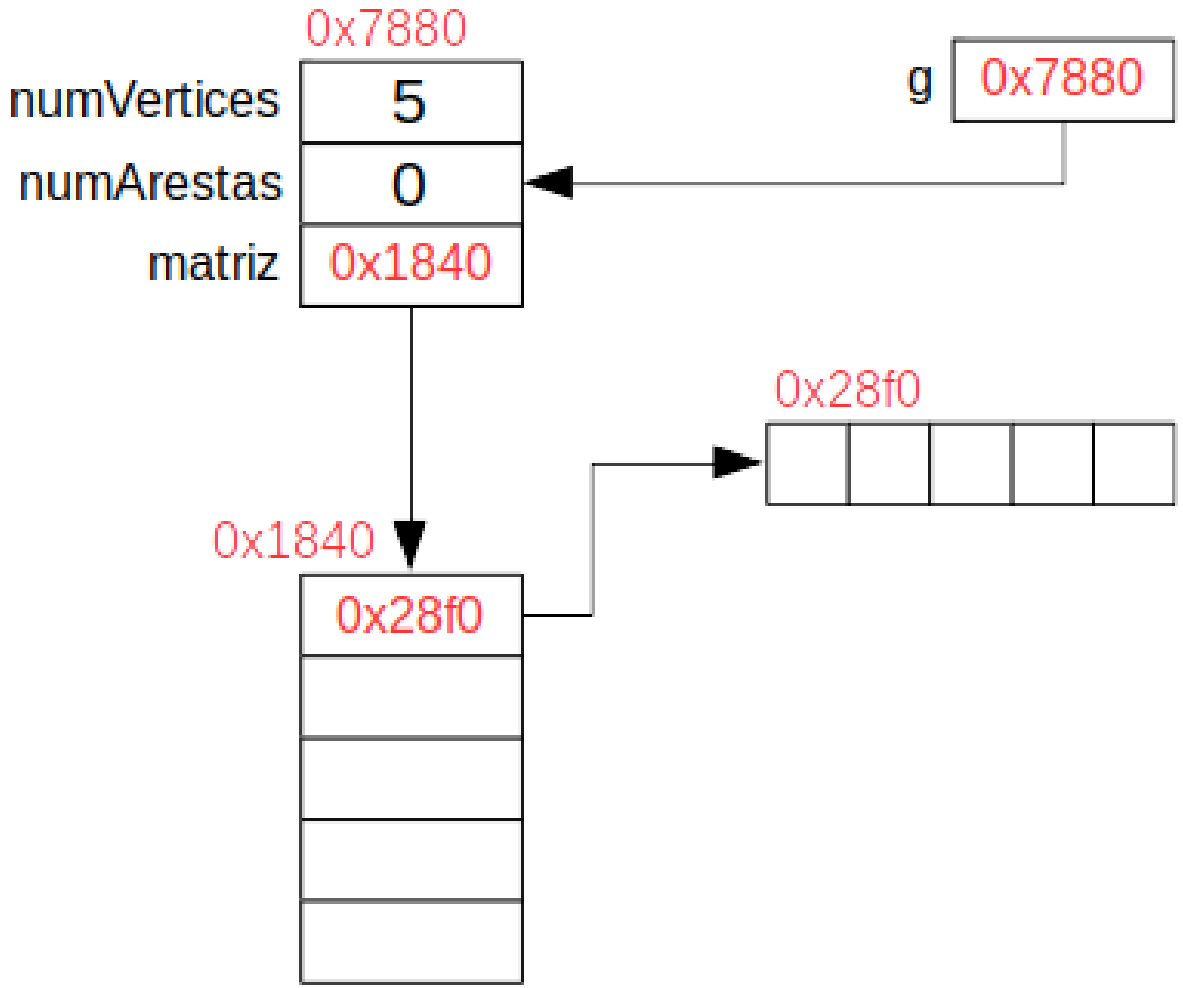
Inicialização



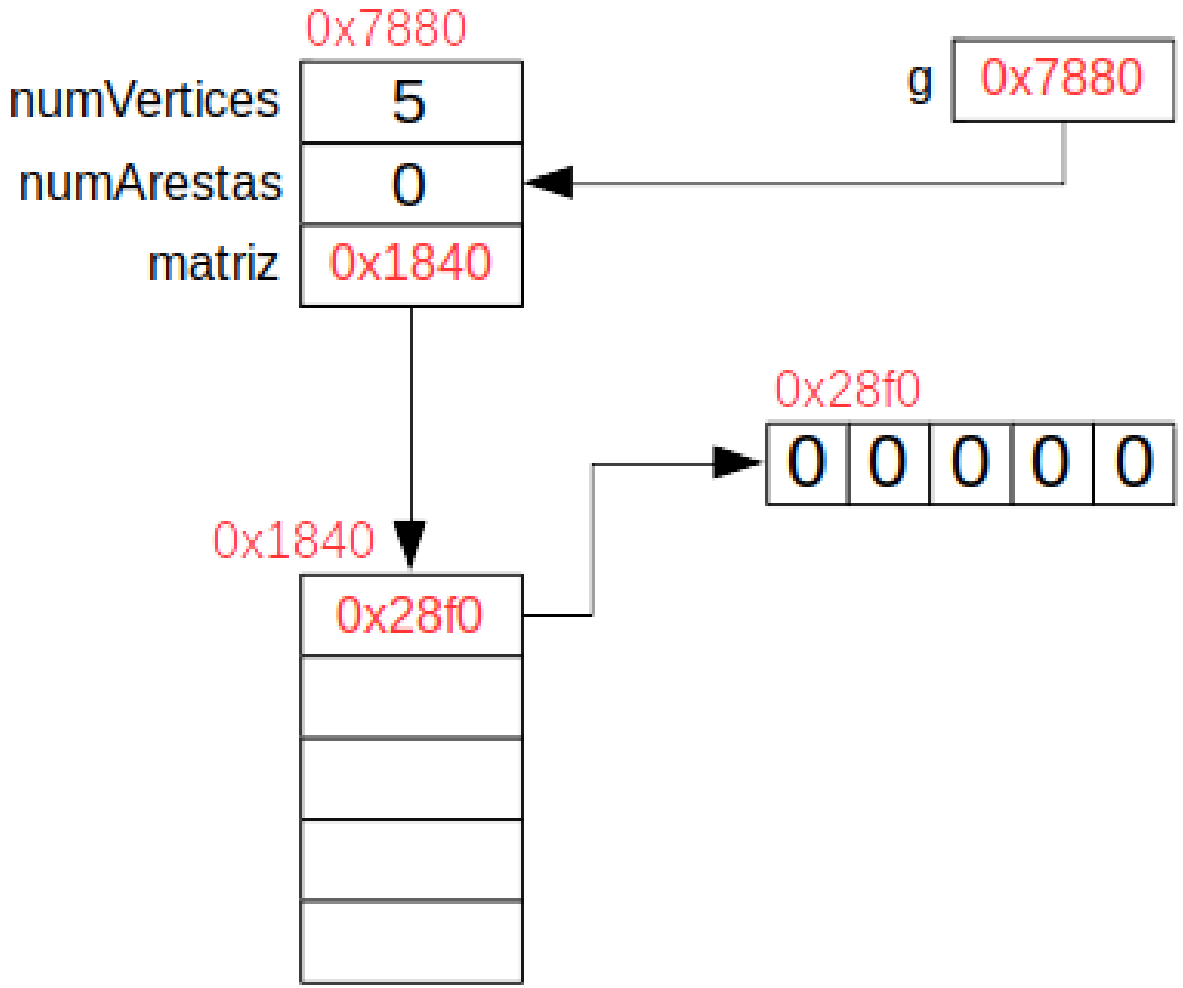
Inicialização



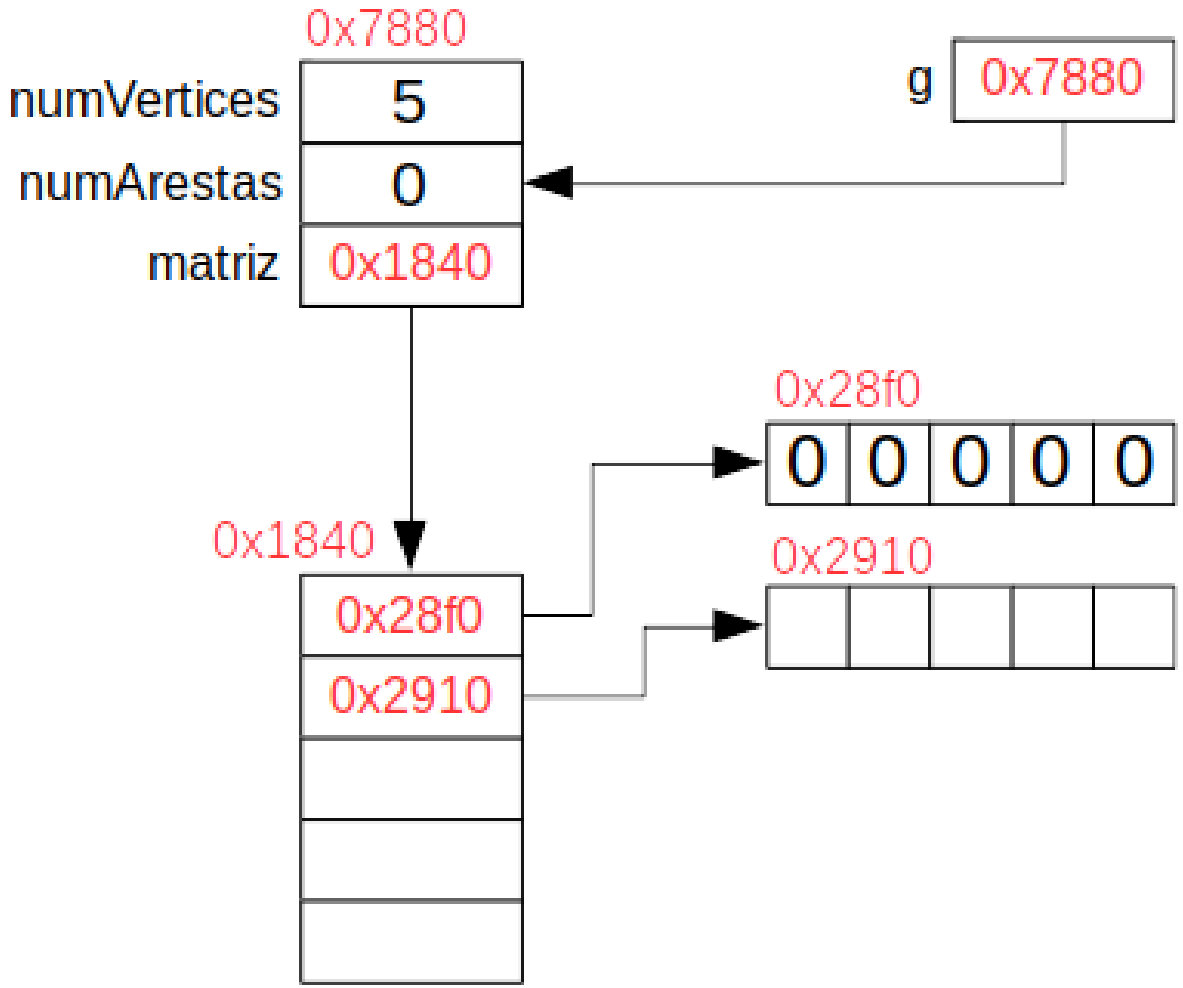
Inicialização



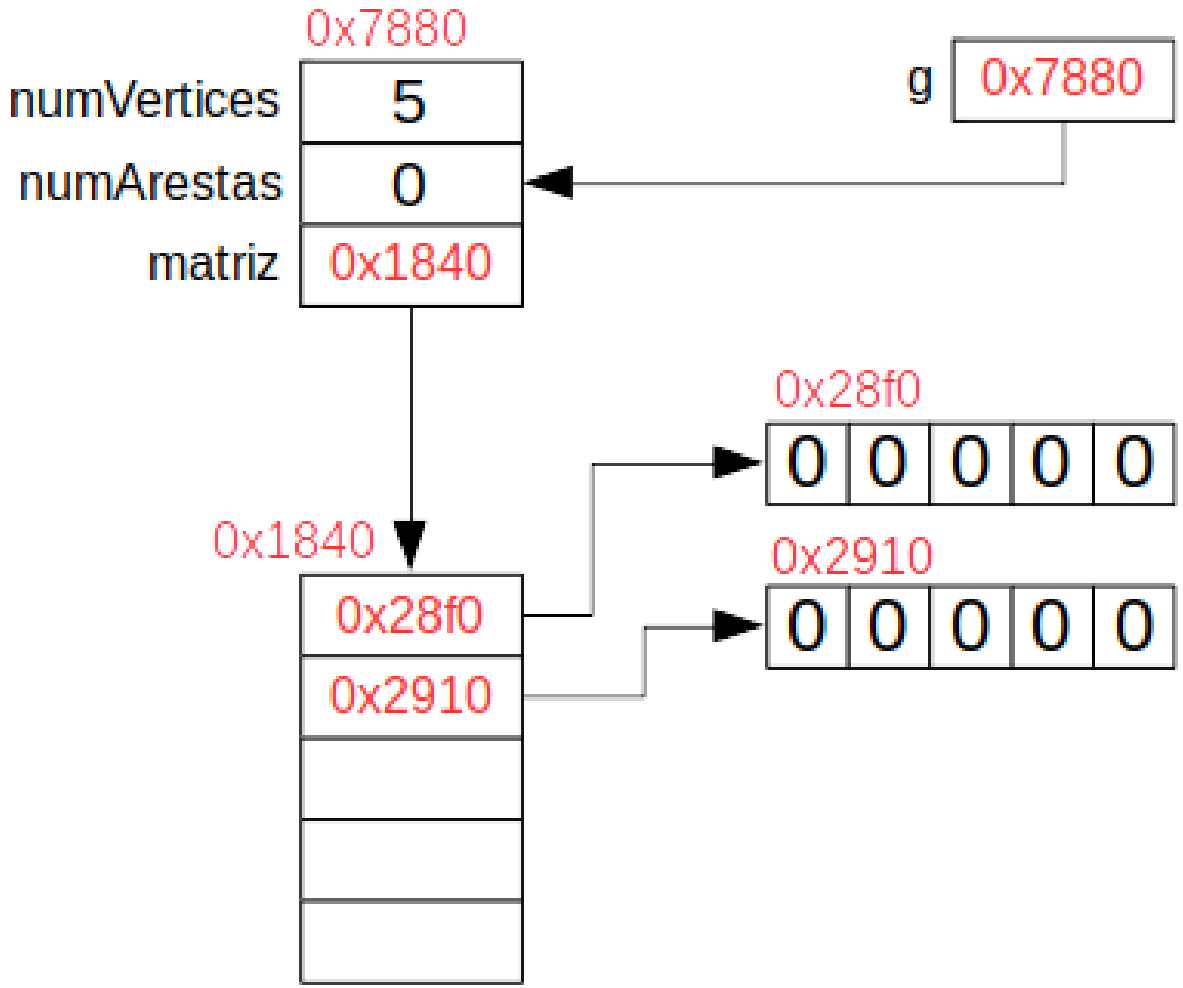
Inicialização



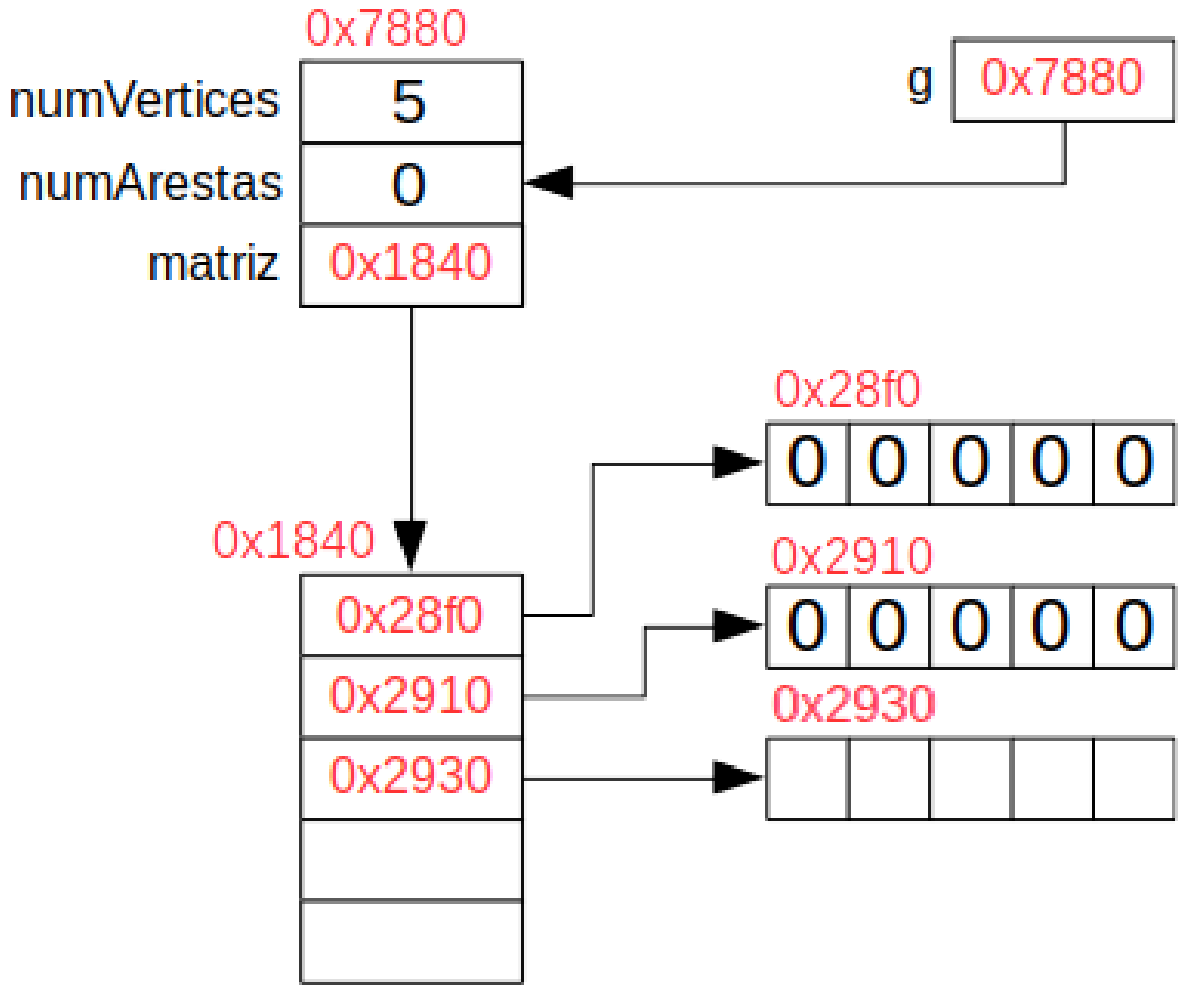
Inicialização



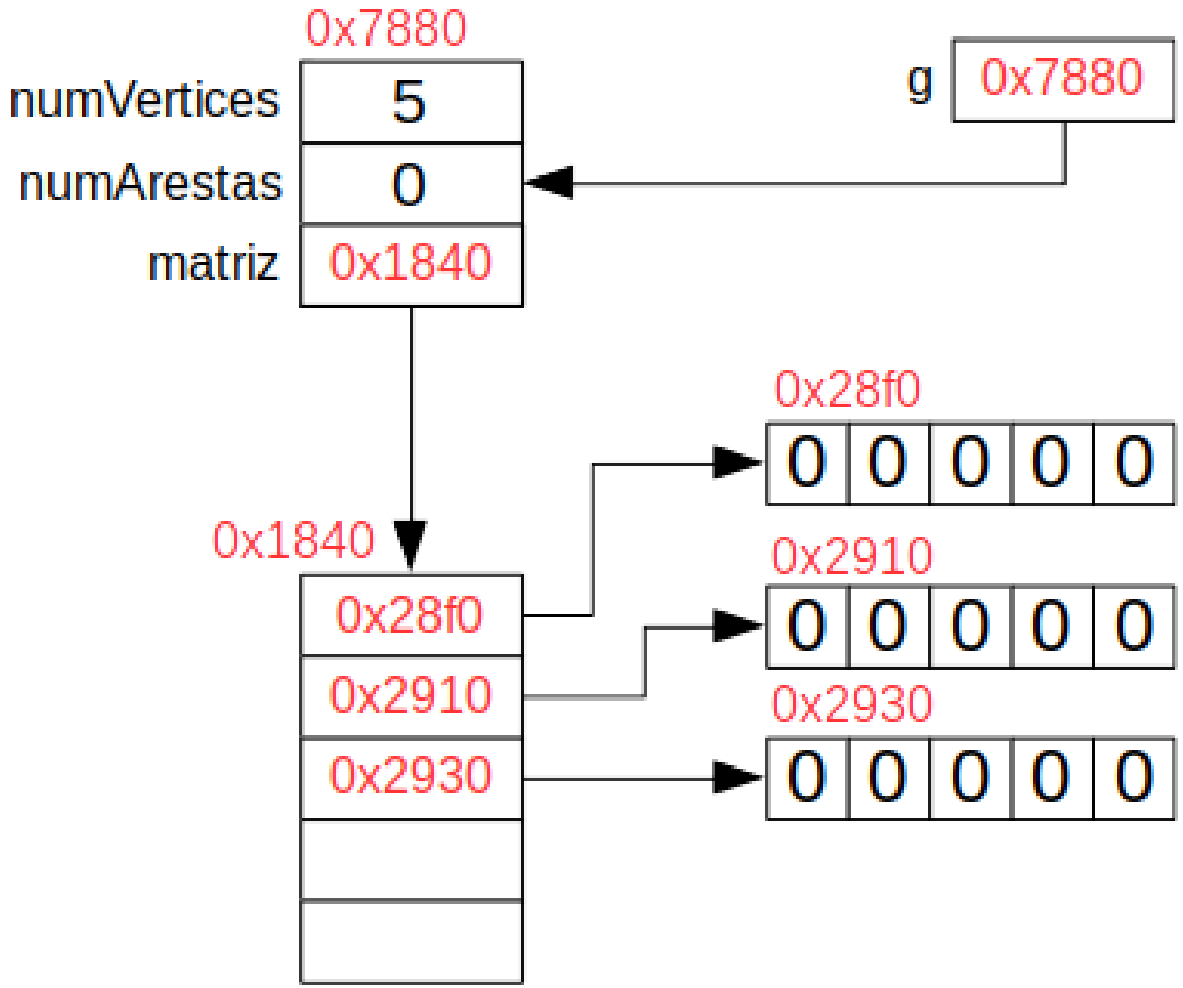
Inicialização



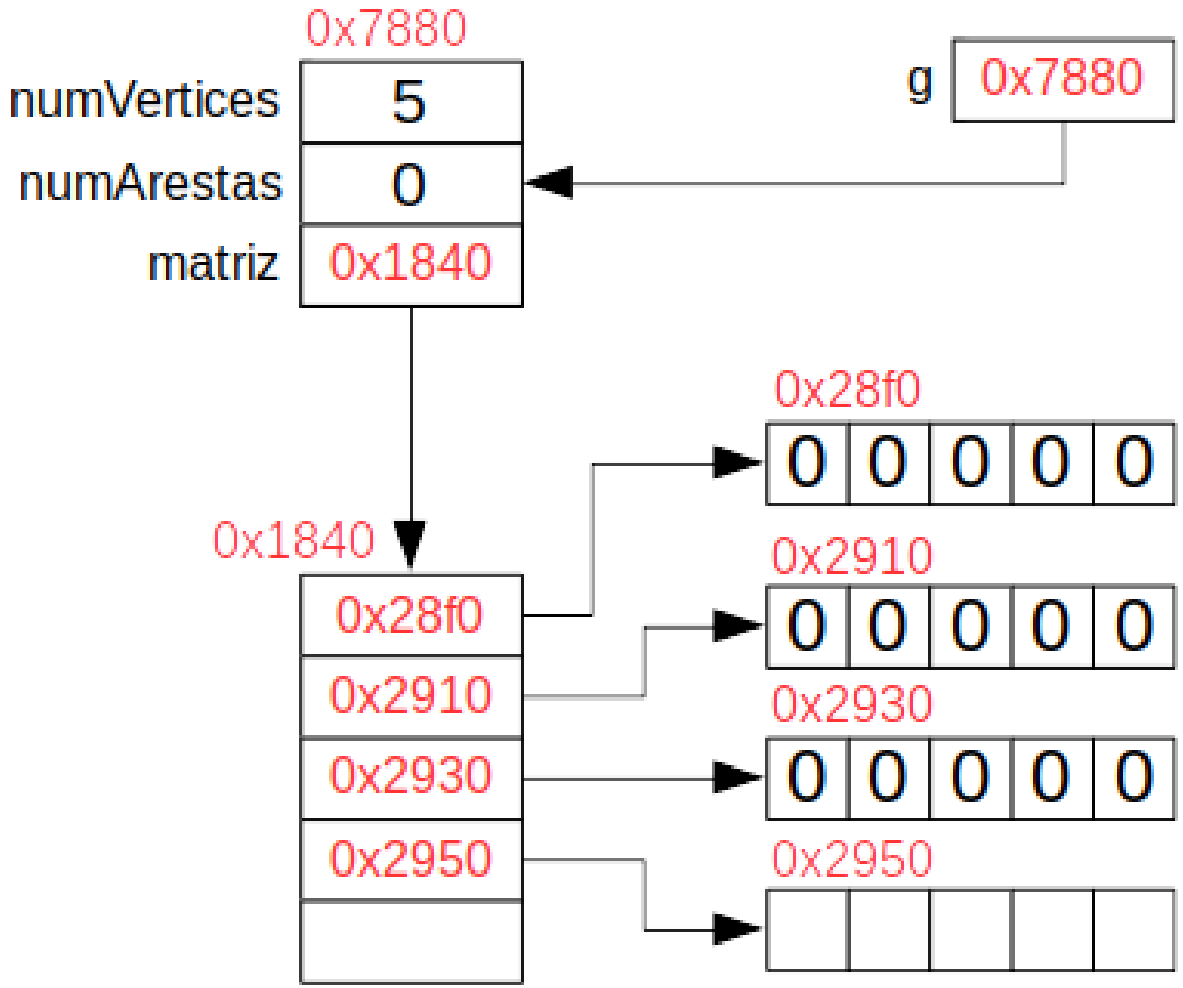
Inicialização



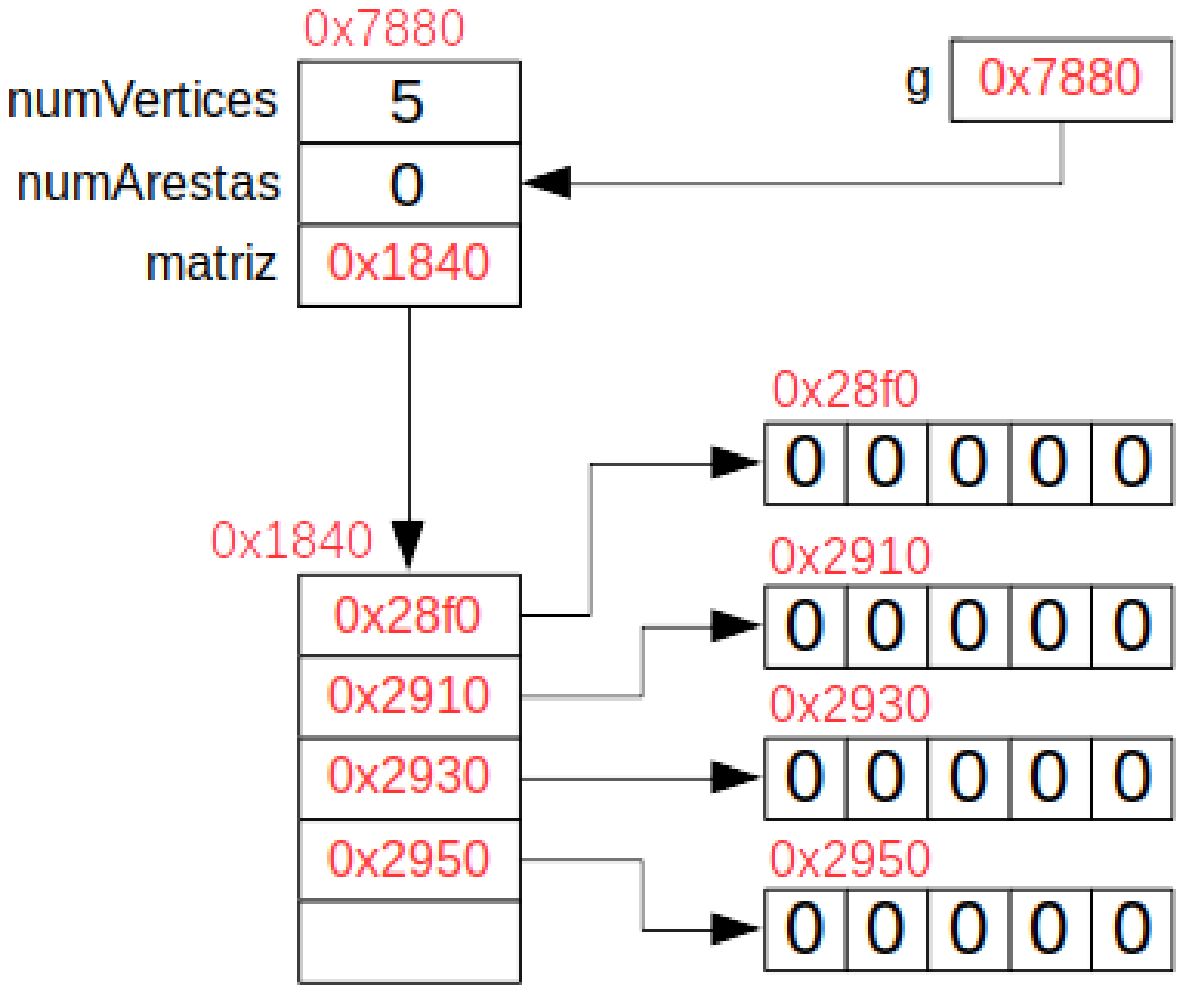
Inicialização



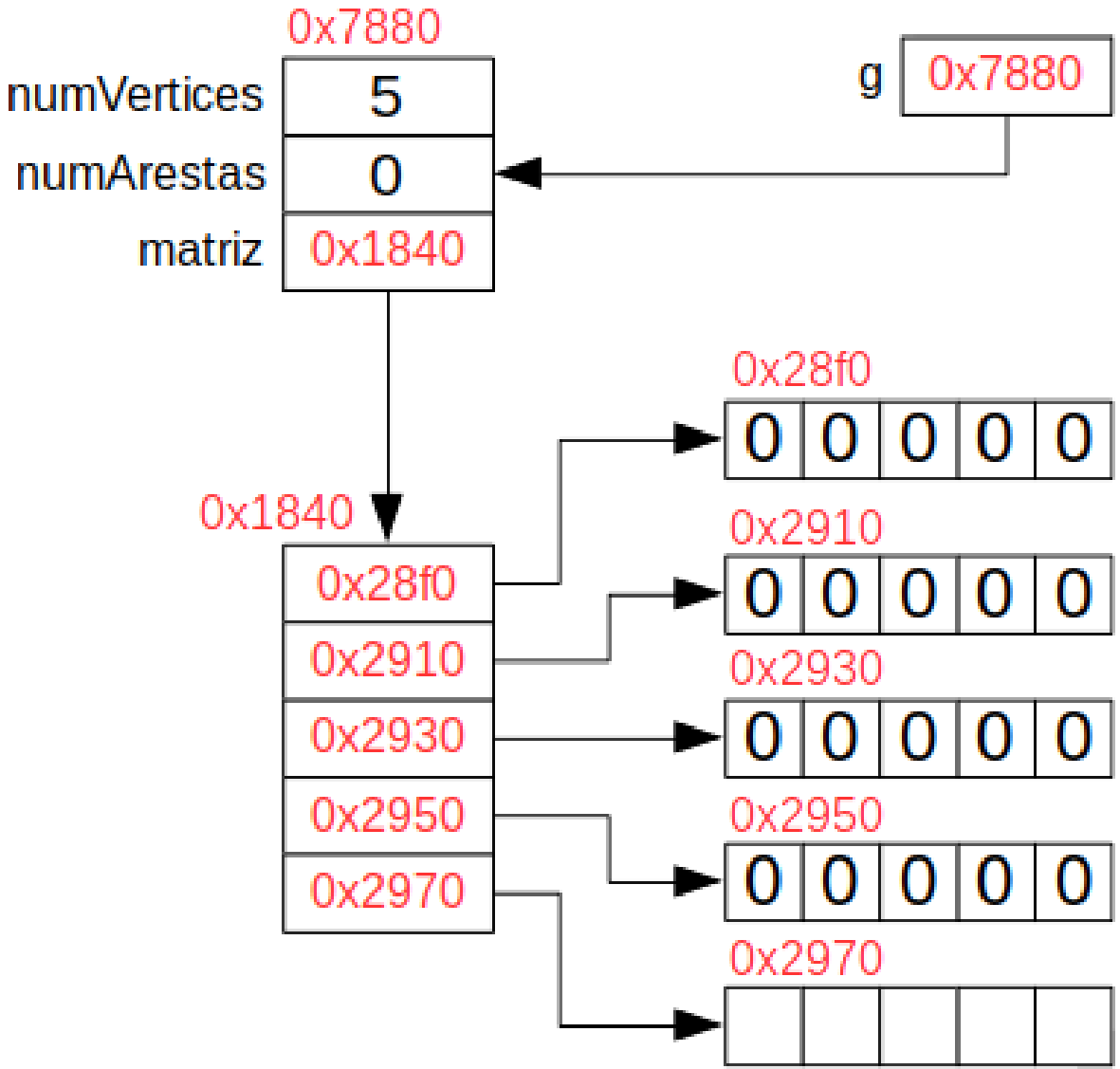
Inicialização



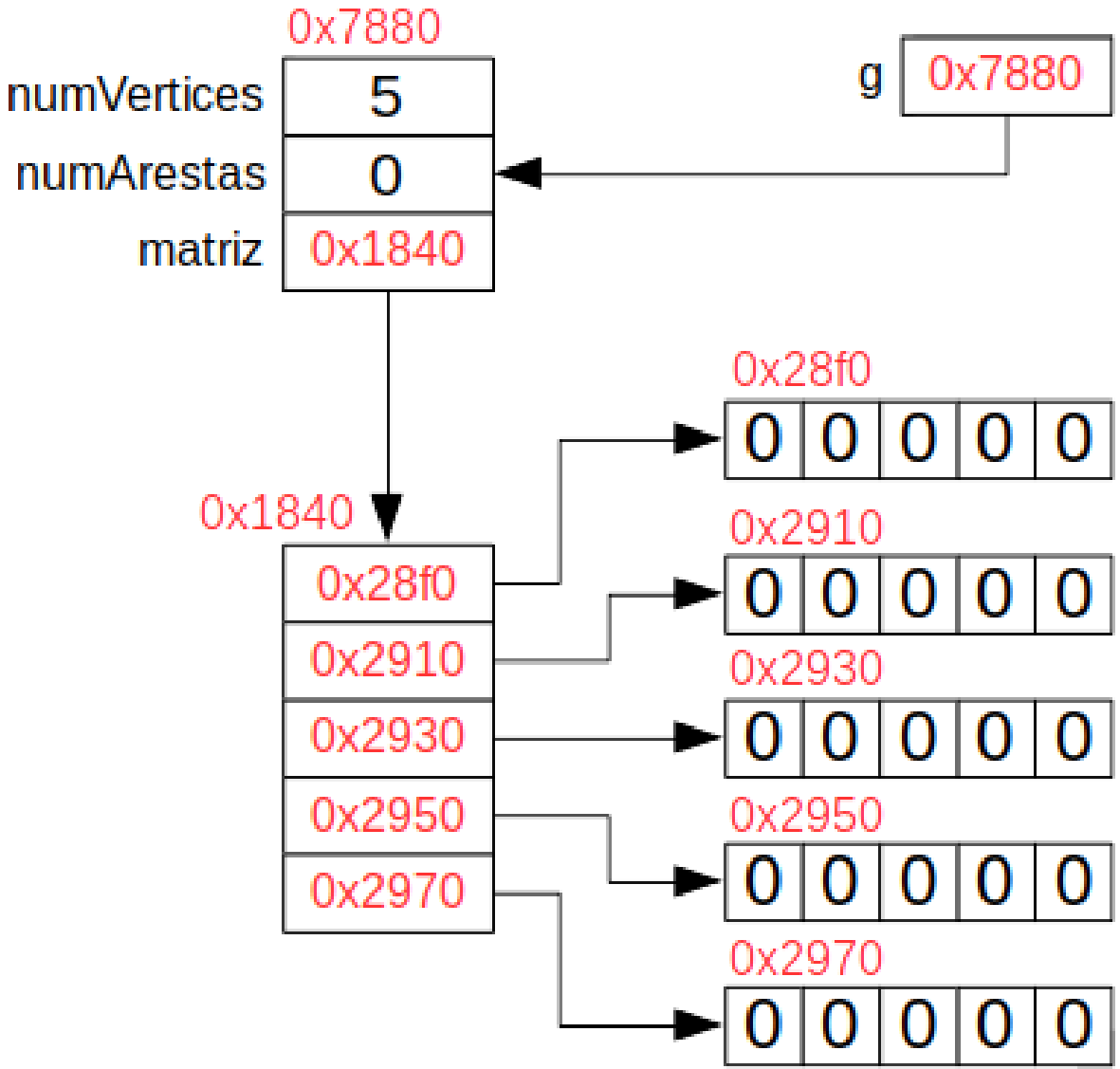
Inicialização



Inicialização



Inicialização



Imprimindo um Grafo

```
void exibeGrafo(Grafo* g){

}
```

Imprimindo um Grafo

```
void exibeGrafo(Grafo* g){
  if (!g) return;
```

```
}
```

Imprimindo um Grafo

```
void exibeGrafo(Grafo* g){
  if (!g) return;
  int x, y;
  printf("Imprimindo grafo
                (vertices: %i; arestas: %i).\n",
                    g->numVertices, g->numArestas);
```

Imprimindo um Grafo

```
void exibeGrafo(Grafo* g){
  if (!g) return;
  int x, y;
  printf("Imprimindo grafo
                (vertices: %i; arestas: %i).\n",
                    g->numVertices, g->numArestas);
  for (x=0;x<g->numVertices;x++) printf("\t%5i",x);
  printf("\n");
```

```
}
```

Imprimindo um Grafo

```
void exibeGrafo(Grafo* g){
  if (!g) return;
  int x, y;
  printf("Imprimindo grafo
                (vertices: %i; arestas: %i).\n",
                    g->numVertices, g->numArestas);
  for (x=0;x<g->numVertices;x++) printf("\t%5i",x);
  printf("\n");
  for (x=0;x<g->numVertices;x++){
    printf("%i",x);



  }
}
```

Imprimindo um Grafo

```
void exibeGrafo(Grafo* g){
  if (!g) return;
  int x, y;
  printf("Imprimindo grafo
                (vertices: %i; arestas: %i).\n",
                    g->numVertices, g->numArestas);
  for (x=0;x<g->numVertices;x++) printf("\t%5i",x);
  printf("\n");
  for (x=0;x<g->numVertices;x++){
    printf("%i",x);
    for (y=0;y<g->numVertices;y++)
      printf("\t%5i",g->matriz[x][y]);
    printf("\n");
  }
}
```

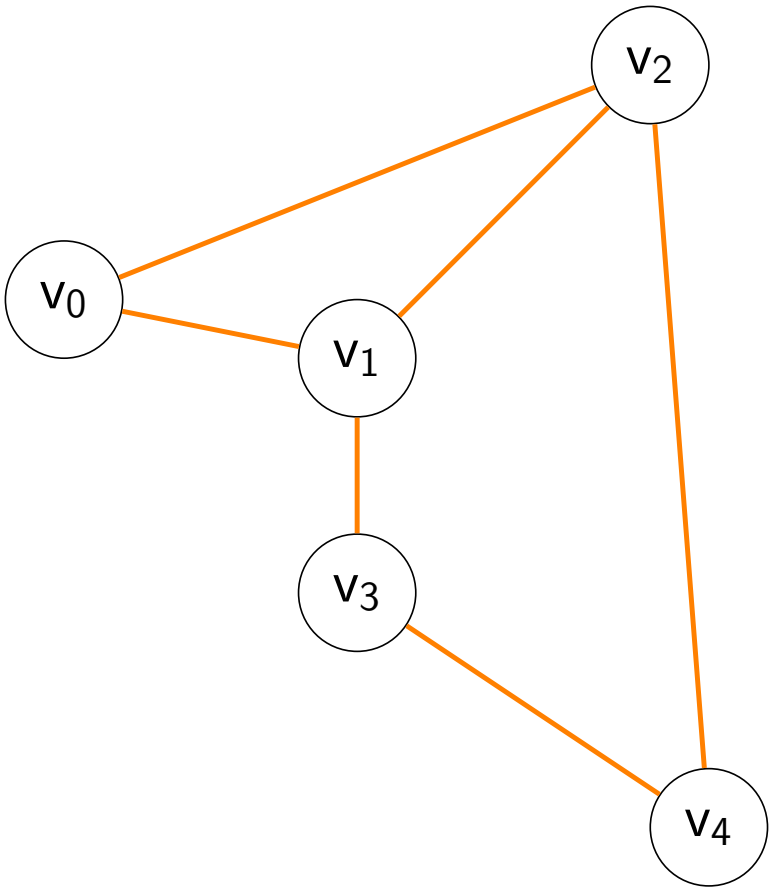
Imprimindo um Grafo

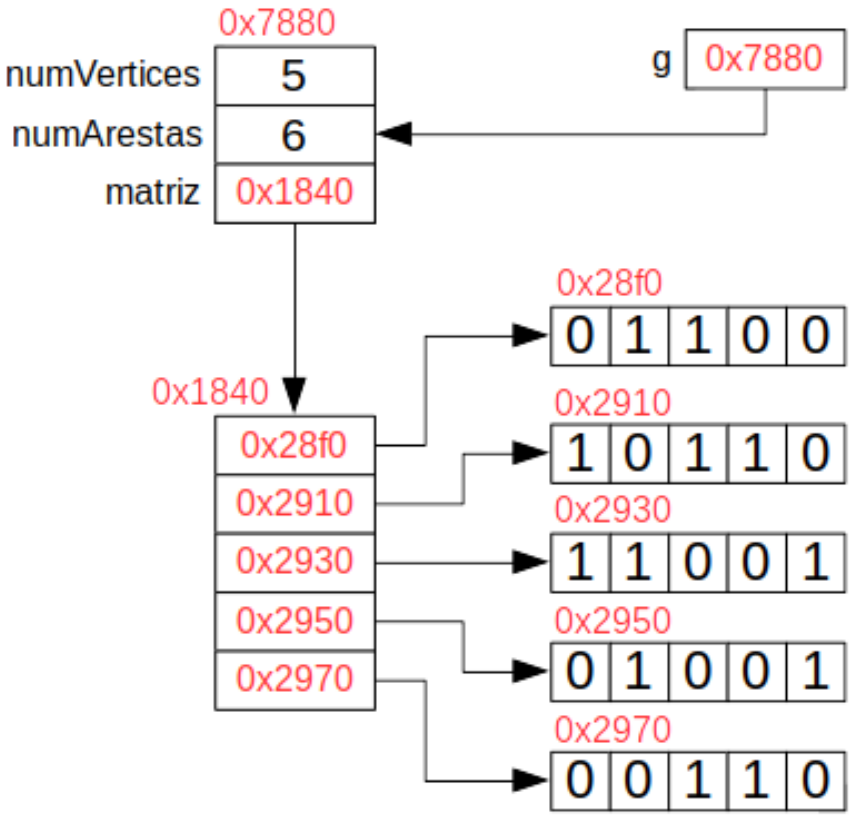
```
void exibeGrafo(Grafo* g){
  if (!g) return;
  int x, y;
  printf("Imprimindo grafo
                (vertices: %i; arestas: %i).\n",
                    g->numVertices, g->numArestas);
  for (x=0;x<g->numVertices;x++) printf("\t%5i",x);
  printf("\n");
  for (x=0;x<g->numVertices;x++){
    printf("%i",x);
    for (y=0;y<g->numVertices;y++)
      printf("\t%5i",g->matriz[x][y]);
    printf("\n");
  }
}
```

$O(V^2)$

Imprimindo um Grafo

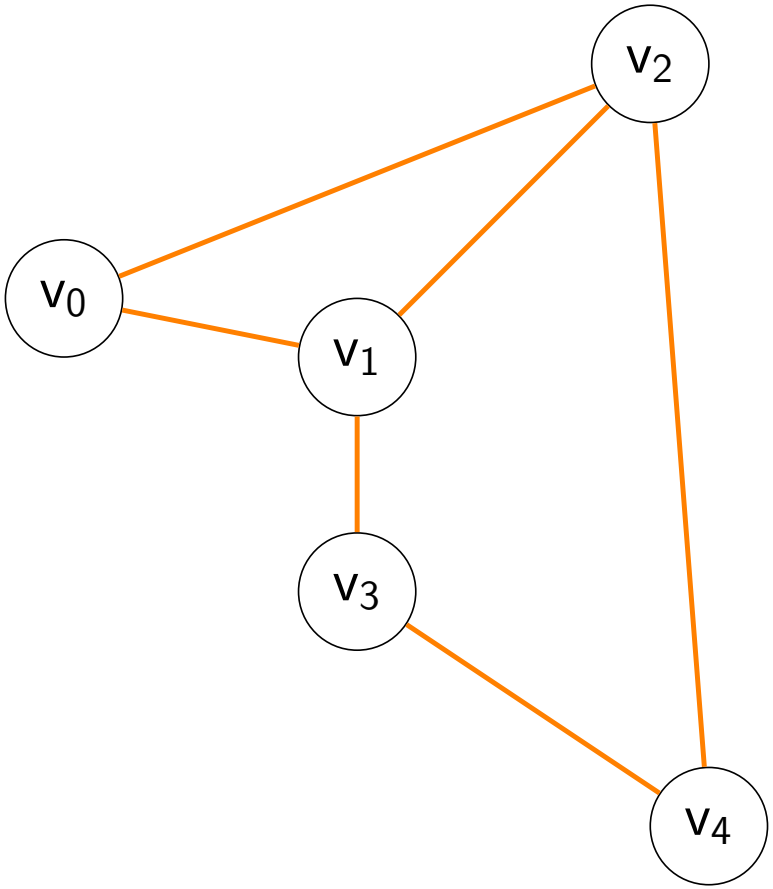
Grafo não ponderado e não dirigido:





Imprimindo um Grafo

Grafo não ponderado e não dirigido:




```
Imprimindo grafo (vertices: 5; arestas: 6).
      0     1     2     3     4
0     0     1     1     0     0
1     1     0     1     1     0
2     1     1     0     0     1
3     0     1     0     0     1
4     0     0     1     1     0
```

Liberando a Memória de um Grafo

```
bool liberaGrafo(Grafo* g){




}
```

Liberando a Memória de um Grafo

```
bool liberaGrafo(Grafo* g){
  if (g==NULL) return false;




}
```

Liberando a Memória de um Grafo

```
bool liberaGrafo(Grafo* g){
  if (g==NULL) return false;
  int x;
  for (x=0; x<g->numVertices; x++)
    free(g->matriz[x]);




}
```

Liberando a Memória de um Grafo

```
bool liberaGrafo(Grafo* g){
  if (g==NULL) return false;
  int x;
  for (x=0; x<g->numVertices; x++)
    free(g->matriz[x]);
  free(g->matriz);




}
```

Liberando a Memória de um Grafo

```
bool liberaGrafo(Grafo* g){
  if (g==NULL) return false;
  int x;
  for (x=0; x<g->numVertices; x++)
    free(g->matriz[x]);
  free(g->matriz);
  g->numVertices = 0;
  g->numArestas = 0;
  g->matriz = NULL;

}
```

Liberando a Memória de um Grafo

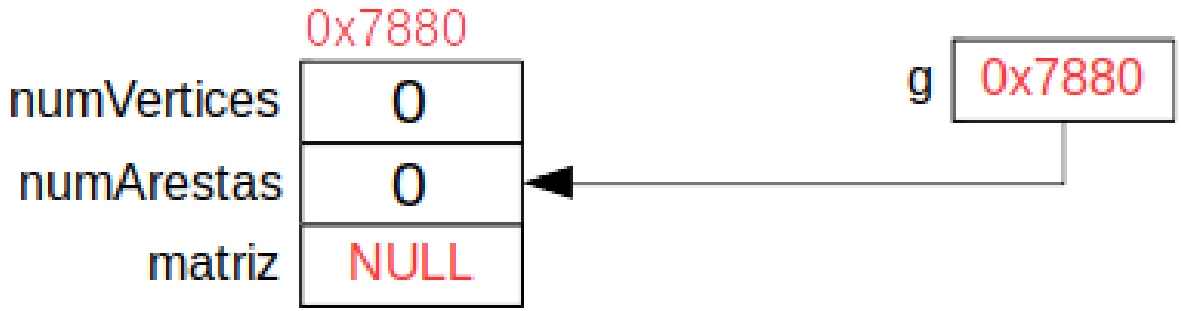
```
bool liberaGrafo(Grafo* g){
  if (g==NULL) return false;
  int x;
  for (x=0; x<g->numVertices; x++)
    free(g->matriz[x]);
  free(g->matriz);
  g->numVertices = 0;
  g->numArestas = 0;
  g->matriz = NULL;
  return true;
}
```

Liberando a Memória de um Grafo

```
bool liberaGrafo(Grafo* g){
  if (g==NULL) return false;
  int x;
  for (x=0; x<g->numVertices; x++)
    free(g->matriz[x]);
  free(g->matriz);
  g->numVertices = 0;
  g->numArestas = 0;
  g->matriz = NULL;
  return true;
}
```

$O(V)$

Liberando a Memória de um Grafo



Inserindo uma Aresta

```
bool insereAresta(Grafo* g, int v1, int v2){




}
```

Inserindo uma Aresta

```
bool insereAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      v1 == v2) return false;









}
```

Inserindo uma Aresta

```
bool insereAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      v1 == v2) return false;
  if (g->matriz[v1][v2] == false){




  }

}
```

Inserindo uma Aresta

```
bool insereAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      v1 == v2) return false;
  if (g->matriz[v1][v2] == false){
      g->matriz[v1][v2] = true;
      g->matriz[v2][v1] = true;

  }

}
```

Inserindo uma Aresta

```
bool insereAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      v1 == v2) return false;
  if (g->matriz[v1][v2] == false){
      g->matriz[v1][v2] = true;
      g->matriz[v2][v1] = true;
      g->numArestas++;
  }

}
```

Inserindo uma Aresta

```
bool insereAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      v1 == v2) return false;
  if (g->matriz[v1][v2] == false){
      g->matriz[v1][v2] = true;
      g->matriz[v2][v1] = true;
      g->numArestas++;
  }
  return true;
}
```

Inserindo uma Aresta

```
bool insereAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      v1 == v2) return false;
  if (g->matriz[v1][v2] == false){
      g->matriz[v1][v2] = true;
      g->matriz[v2][v1] = true;
      g->numArestas++;
  }
  return true;
}
```

O(1)

Removendo uma Aresta

```
bool removeAresta(Grafo* g, int v1, int v2){



}
```

Removendo uma Aresta

```
bool removeAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;




}
```

Removendo uma Aresta

```
bool removeAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;
  g->matriz[v1][v2] = false;
  g->matriz[v2][v1] = false;


}
```

Removendo uma Aresta

```
bool removeAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;
  g->matriz[v1][v2] = false;
  g->matriz[v2][v1] = false;
  g->numArestas--;

}
```

Removendo uma Aresta

```
bool removeAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;
  g->matriz[v1][v2] = false;
  g->matriz[v2][v1] = false;
  g->numArestas--;
  return true;
}
```

Removendo uma Aresta

```
bool removeAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;
  g->matriz[v1][v2] = false;
  g->matriz[v2][v1] = false;
  g->numArestas--;
  return true;
}
```

O(1)

Verificando a Existência de uma Aresta

```
bool arestaExiste(Grafo* g, int v1, int v2){



}
```

Verificando a Existência de uma Aresta

```
bool arestaExiste(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;

}
```

Verificando a Existência de uma Aresta

```
bool arestaExiste(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;
  return true;
}
```

Verificando a Existência de uma Aresta

```
bool arestaExiste(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;
  return true;
}
```

O(1)

Número de Vértices

```
int numeroDeVertices(Grafo* g){


}
```

Número de Vértices

```
int numeroDeVertices(Grafo* g){
  if (g!=NULL) return g->numVertices;

}
```

Número de Vértices

```
int numeroDeVertices(Grafo* g){
  if (g!=NULL) return g->numVertices;
  else return -1;
}
```

Número de Vértices

```
int numeroDeVertices(Grafo* g){
  if (g!=NULL) return g->numVertices;
  else return -1;
}
```

O(1)

Número de Aresta

```
int numeroDeArestas(Grafo* g){


}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;

}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}


int numeroDeArestas2(Grafo* g){




}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}


int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;









}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}


int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)



}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}


int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)
    for (y=x+1; y<g->numVertices; y++)


}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}


int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)
    for (y=x+1; y<g->numVertices; y++)
      if (g->matriz[x][y]) arestas++;

}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}
```

```
int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)
    for (y=x+1; y<g->numVertices; y++)
      if (g->matriz[x][y]) arestas++;
  return arestas;
}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}
```

O(1)

```
int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)
    for (y=x+1; y<g->numVertices; y++)
      if (g->matriz[x][y]) arestas++;
  return arestas;
}
```

Número de Aresta

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}
```

$O(1)$

```
int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)
    for (y=x+1; y<g->numVertices; y++)
      if (g->matriz[x][y]) arestas++;
  return arestas;
}
```

$O(V^2)$

Verificar se um Vértice Possui Vizinhos

```
bool possuiVizinhos(Grafo* g, int v){



}
```

Verificar se um Vértice Possui Vizinhos

```
bool possuiVizinhos(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return false;



}
```

Verificar se um Vértice Possui Vizinhos

```
bool possuiVizinhos(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return false;
  int x;
  for (x=0;x<g->numVertices;x++)
    if (g->matriz[v][x]) return true;

}
```

Verificar se um Vértice Possui Vizinhos

```
bool possuiVizinhos(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return false;
  int x;
  for (x=0;x<g->numVertices;x++)
    if (g->matriz[v][x]) return true;
  return false;
}
```

Verificar se um Vértice Possui Vizinhos

```
bool possuiVizinhos(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return false;
  int x;
  for (x=0;x<g->numVertices;x++)
    if (g->matriz[v][x]) return true;
  return false;
}
```

$O(V)$

Grau de um Vértice

```
int retornaGrauDoVertice(Grafo* g, int v){



}
```

Grau de um Vértice

```
int retornaGrauDoVertice(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return -1;




}
```

Grau de um Vértice

```
int retornaGrauDoVertice(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return -1;
  int x, grau = 0;
  for (x=0;x<g->numVertices;x++)
    if (g->matriz[v][x]) grau++;

}
```

Grau de um Vértice

```
int retornaGrauDoVertice(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return -1;
  int x, grau = 0;
  for (x=0;x<g->numVertices;x++)
    if (g->matriz[v][x]) grau++;
  return grau;
}
```

Grau de um Vértice

```
int retornaGrauDoVertice(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return -1;
  int x, grau = 0;
  for (x=0;x<g->numVertices;x++)
    if (g->matriz[v][x]) grau++;
  return grau;
}
```

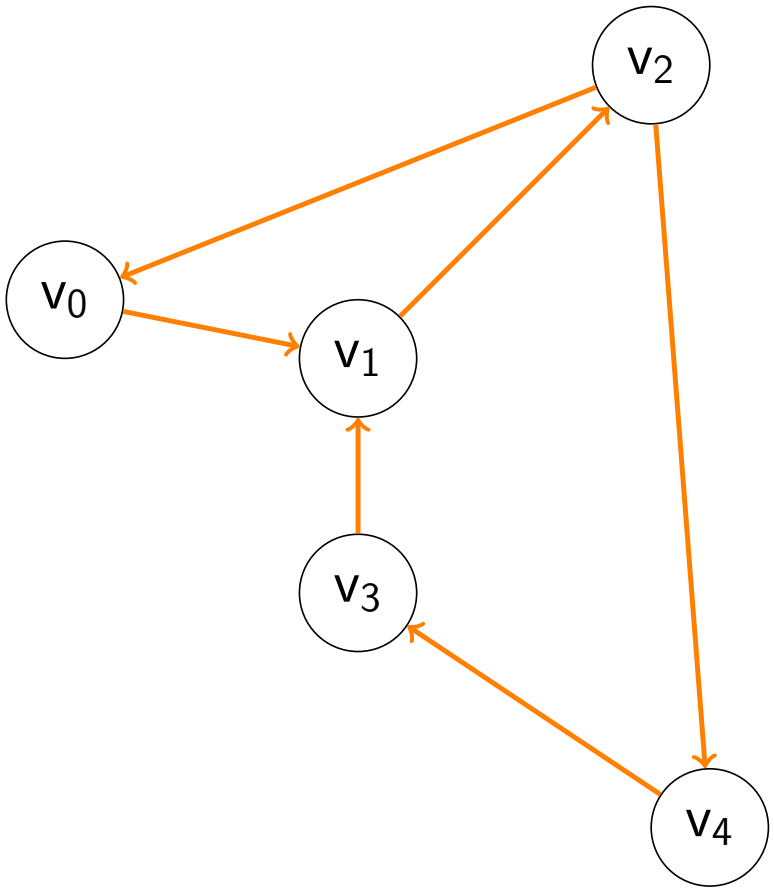
$O(V)$

Grafos – Matrizes de Adjacências

- E se o grafo for dirigido?

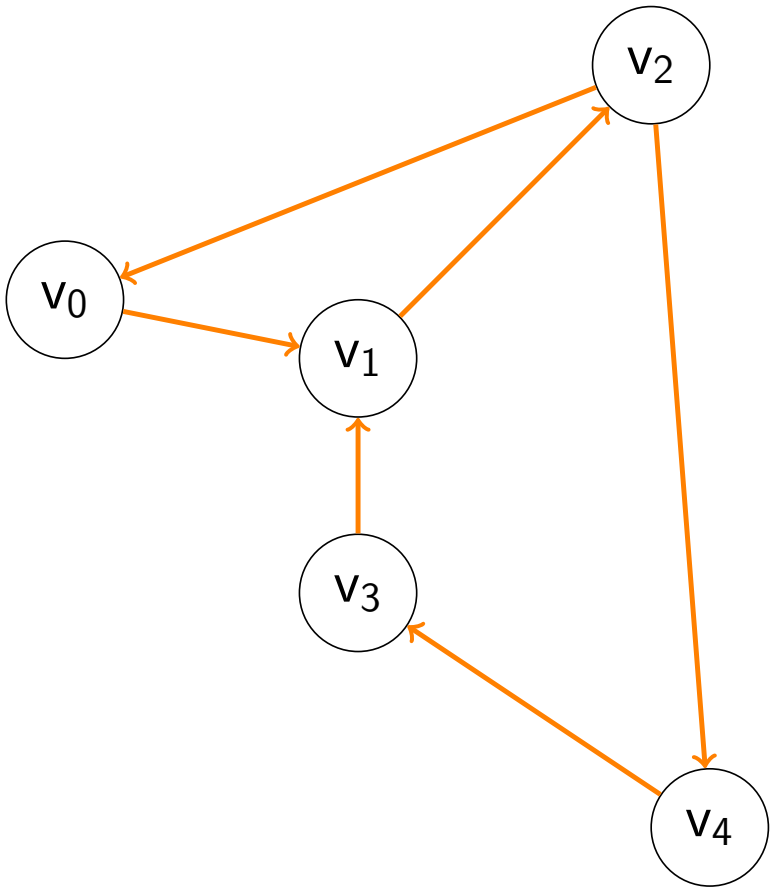
Grafos – Matrizes de Adjacências

- E se o grafo for dirigido?



Grafos – Matrizes de Adjacências

- E se o grafo for dirigido?

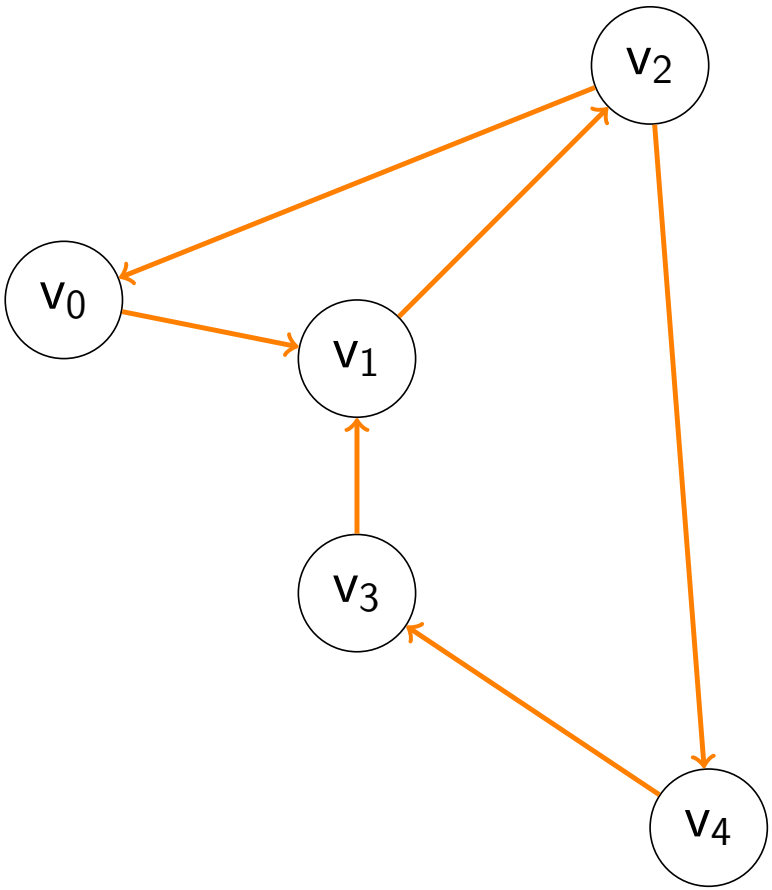


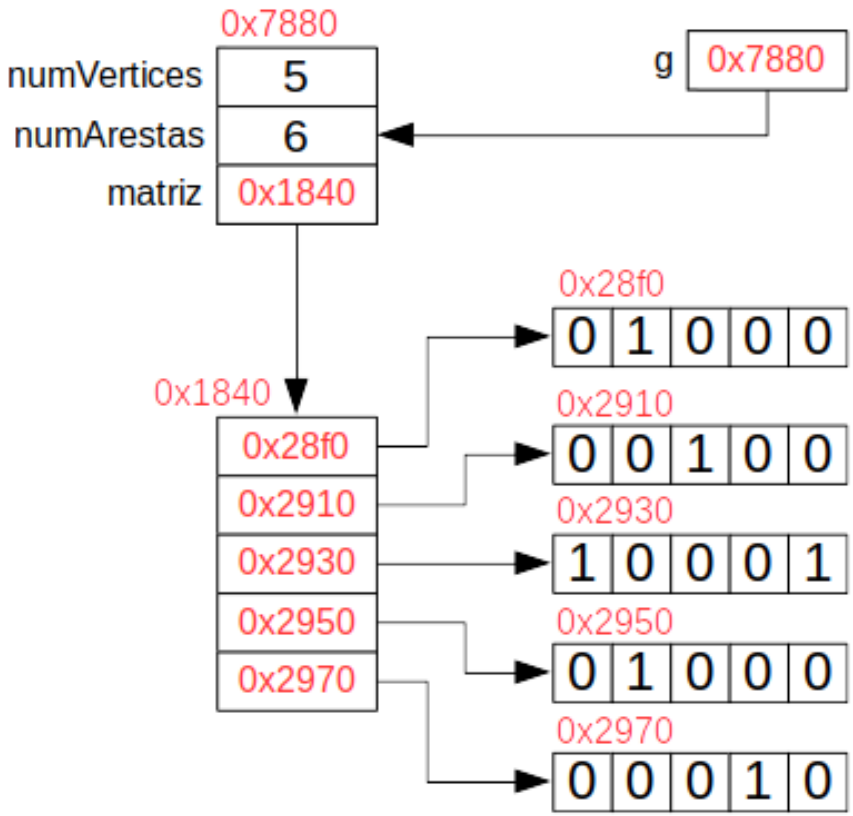
A matriz não será simétrica.

	v_0	v_1	v_2	v_3	v_4
v_0	0	1	0	0	0
v_1	0	0	1	0	0
v_2	1	0	0	0	1
v_3	0	1	0	0	0
v_4	0	0	0	1	0

Grafos – Matrizes de Adjacências

- E se o grafo for dirigido?





Matrizes de Adjacências - Grafo Dirigido

A estrutura de dados será a mesma, haverá pequenas mudanças em algumas funções:

Matrizes de Adjacências - Grafo Dirigido

A estrutura de dados será a mesma, haverá pequenas mudanças em algumas funções:

- Inicializar um Grafo
- Imprimir um Grafo
- Liberar a Memória de um Grafo
- Inserir uma Aresta
- Remover uma Aresta
- Verificar se uma Aresta Existe
- Retornar o Número de Vértices
- Retornar o Número de Arestas*
- Verificar se um Vértice Possui Vizinhos
- Retornar o Grau de um Vértice

Inserindo uma Aresta - Digrafo

```
bool insereAresta(Grafo* g, int v1, int v2){



}
```

Inserindo uma Aresta - Digrafo

```
bool insereAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices
      || v1 == v2) return false;
  if (g->matriz[v1][v2] == false){
      g->matriz[v1][v2] = true;
      g->matriz[v2][v1] = true;
      g->numArestas++;
  }
  return true;
}
```

O(1)

Removendo uma Aresta - Digrafo

```
bool removeAresta(Grafo* g, int v1, int v2){




}
```

Removendo uma Aresta - Digrafo

```
bool removeAresta(Grafo* g, int v1, int v2){
  if (!g || v1 < 0 || v2 < 0 ||
      v1 >= g->numVertices || v2 >= g->numVertices ||
      g->matriz[v1][v2] == false) return false;
  g->matriz[v1][v2] = false;
  g->matriz[v2][v1] = false;
  g->numArestas--;
  return true;
}
```

O(1)

Número de Aresta - Digrafo

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}
```

O(1)

```
int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)
    for (y=x+1; y<g->numVertices; y++)
      if (g->matriz[x][y]) arestas++;
  return arestas;
}
```

$O(V^2)$

Número de Aresta - Digrafo

```
int numeroDeArestas(Grafo* g){
  if (g!=NULL) return g->numArestas;
  else return -1;
}
```

O(1)

```
int numeroDeArestas2(Grafo* g){
  if (g==NULL) return -1;
  int x, y, arestas = 0;
  for (x=0; x<g->numVertices; x++)
    for (y=0; y<g->numVertices; y++)
      if (g->matriz[x][y]) arestas++;
  return arestas;
}
```

$O(V^2)$

Grau de um Vértice - Digrafo

```
int retornaGrauDoVertice(Grafo* g, int v){




}
```

Grau de um Vértice - Digrafo

```
int retornaGrauDoVertice(Grafo* g, int v){
  if (!g || v < 0 || v >= g->numVertices) return -1;
  int x, grau = 0;
  for (x=0;x<g->numVertices;x++){
     if (g->matriz[v][x]) grau++;
     if (g->matriz[x][v]) grau++;
  }
  return grau;
}
```

$O(V)$

Grafos – Matrizes de Adjacências

E se o grafo for ponderado?

Grafos – Matrizes de Adjacências

E se o grafo for ponderado?

Trataremos isso na próxima aula!

Algoritmos e Estruturas de Dados II
Aula 03 – Grafos: Matriz de Adjacência

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri