Algoritmos e Estruturas de Dados II
Aula 08 – Grafos: Busca em Largura

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

Grafos – Percursos

Como podemos navegar em um grafo?
Isto é, visitar seus vértices caminhando por meio das arestas, fazer buscas etc?

Grafos - Percursos

Há duas formas mais comuns de se fazer isso:

- Busca em Profundidade
- Busca em Largura

Grafos - Percursos

Há duas formas mais comuns de se fazer isso:

- Busca em Profundidade
- **Busca em Largura**

Grafos – Busca em Largura

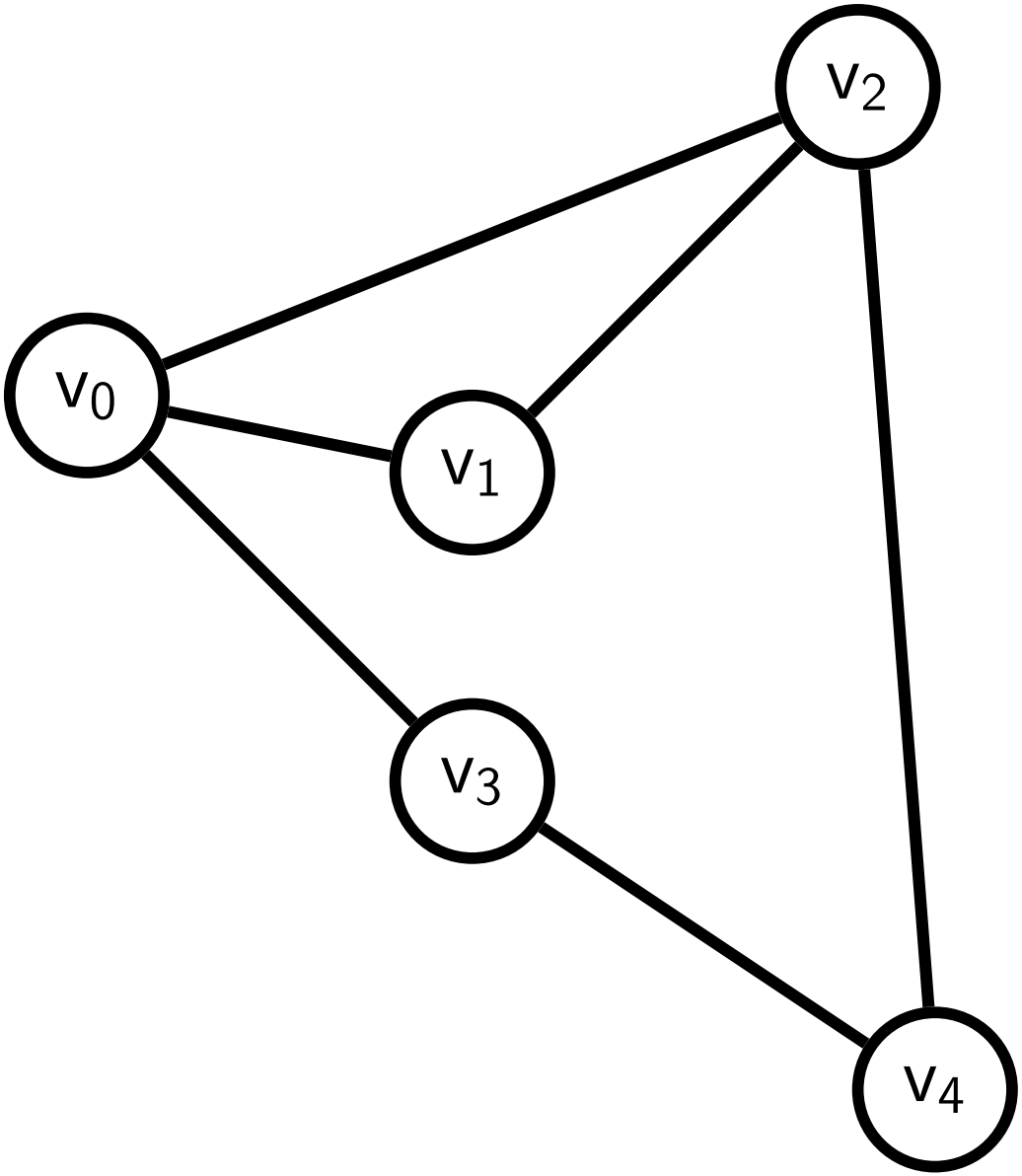
A Busca em Largura (ou do inglês *Breadth-First Search - BFS*) é um algoritmo ou estratégia que pode ser utilizada para realizar uma busca ou travessia estruturas como árvores e grafos.

Grafos – Busca em Largura

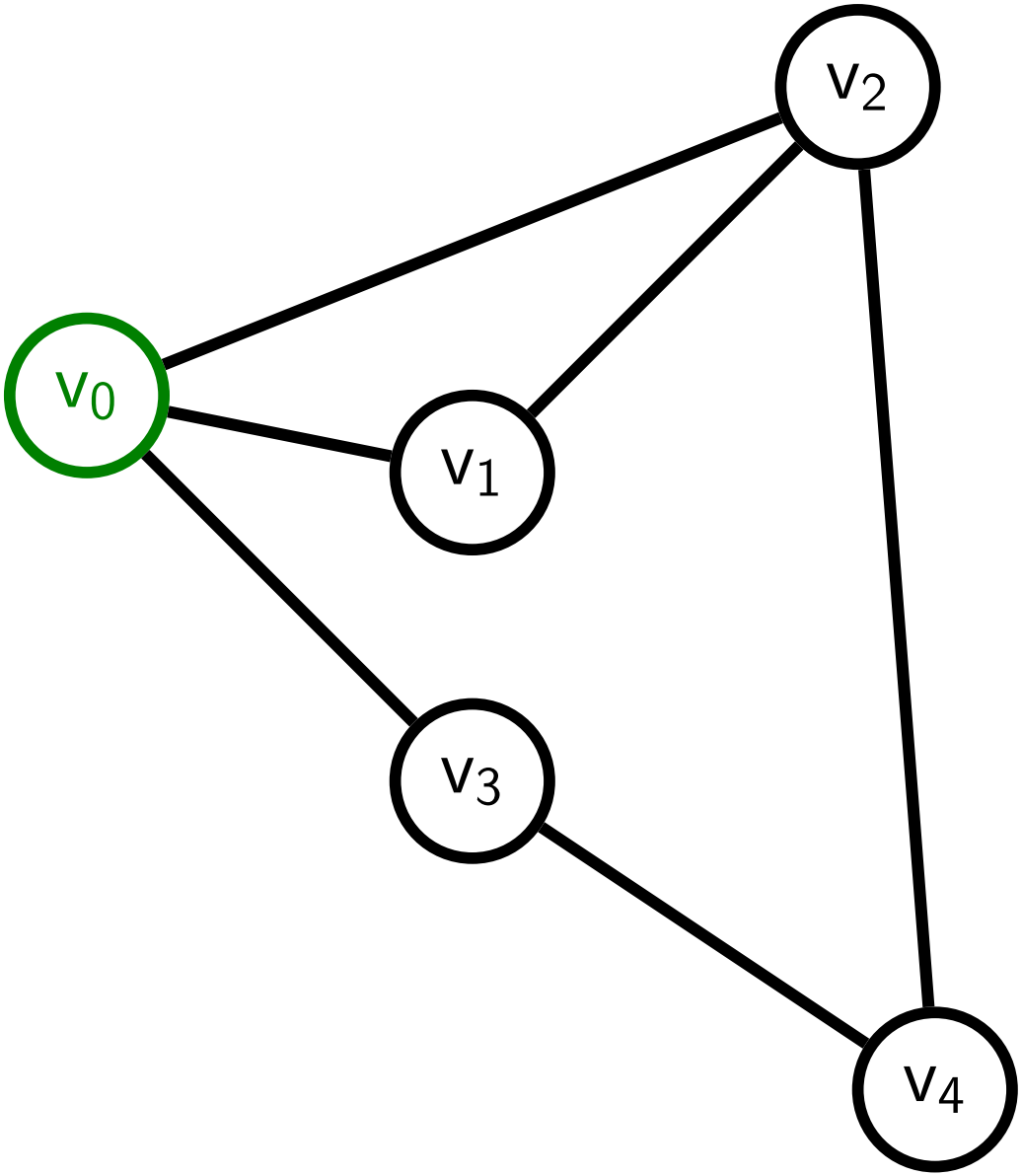
A Busca em Largura (ou do inglês *Breadth-First Search - BFS*) é um algoritmo ou estratégia que pode ser utilizada para realizar uma busca ou travessia estruturas como árvores e grafos.

Em grafos, o algoritmo começa a partir de um vértice e explora, inicialmente, todos os seus vizinhos, depois os vizinhos dos vizinhos e assim sucessivamente.

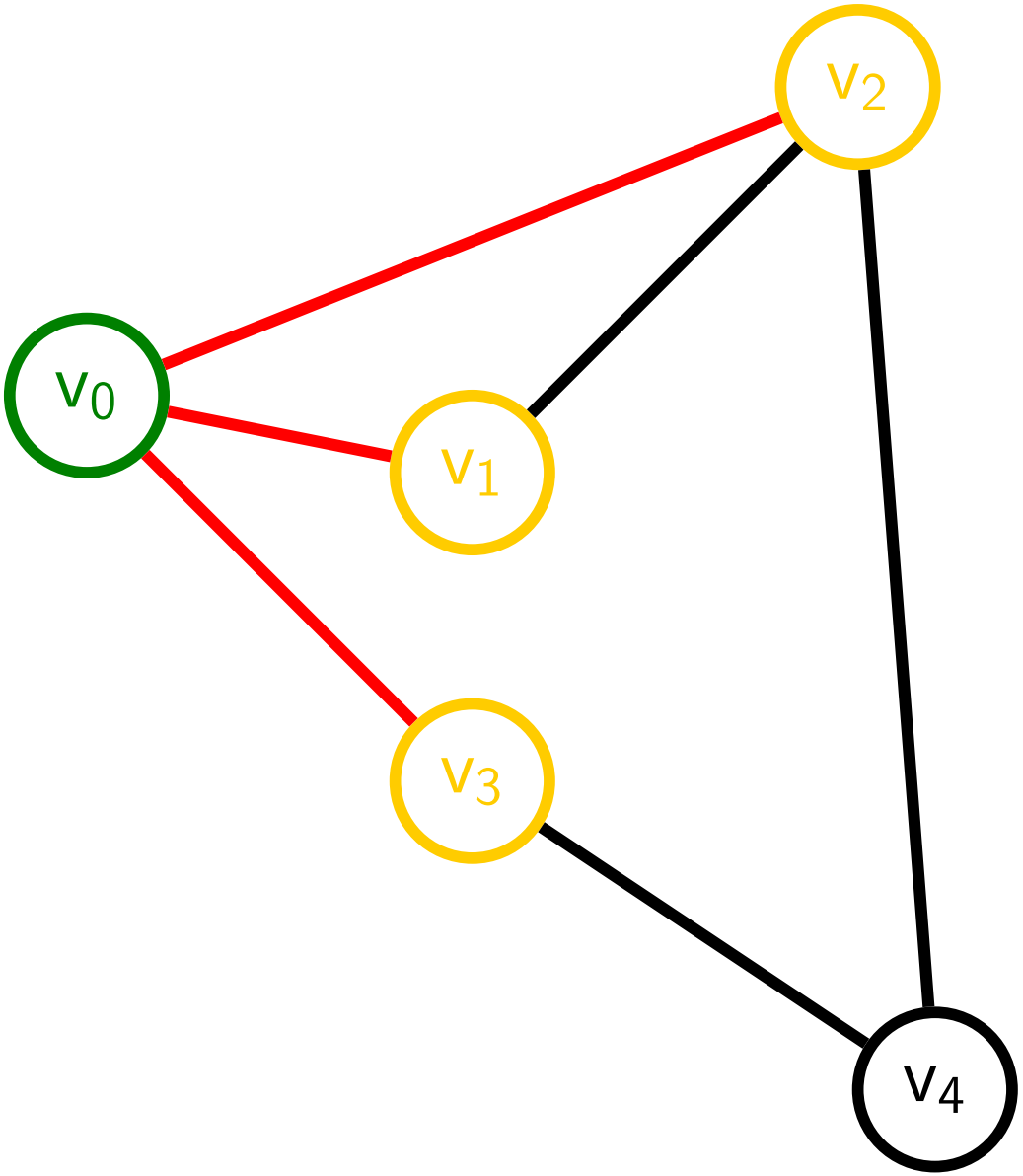
Busca em Largura - Ideia



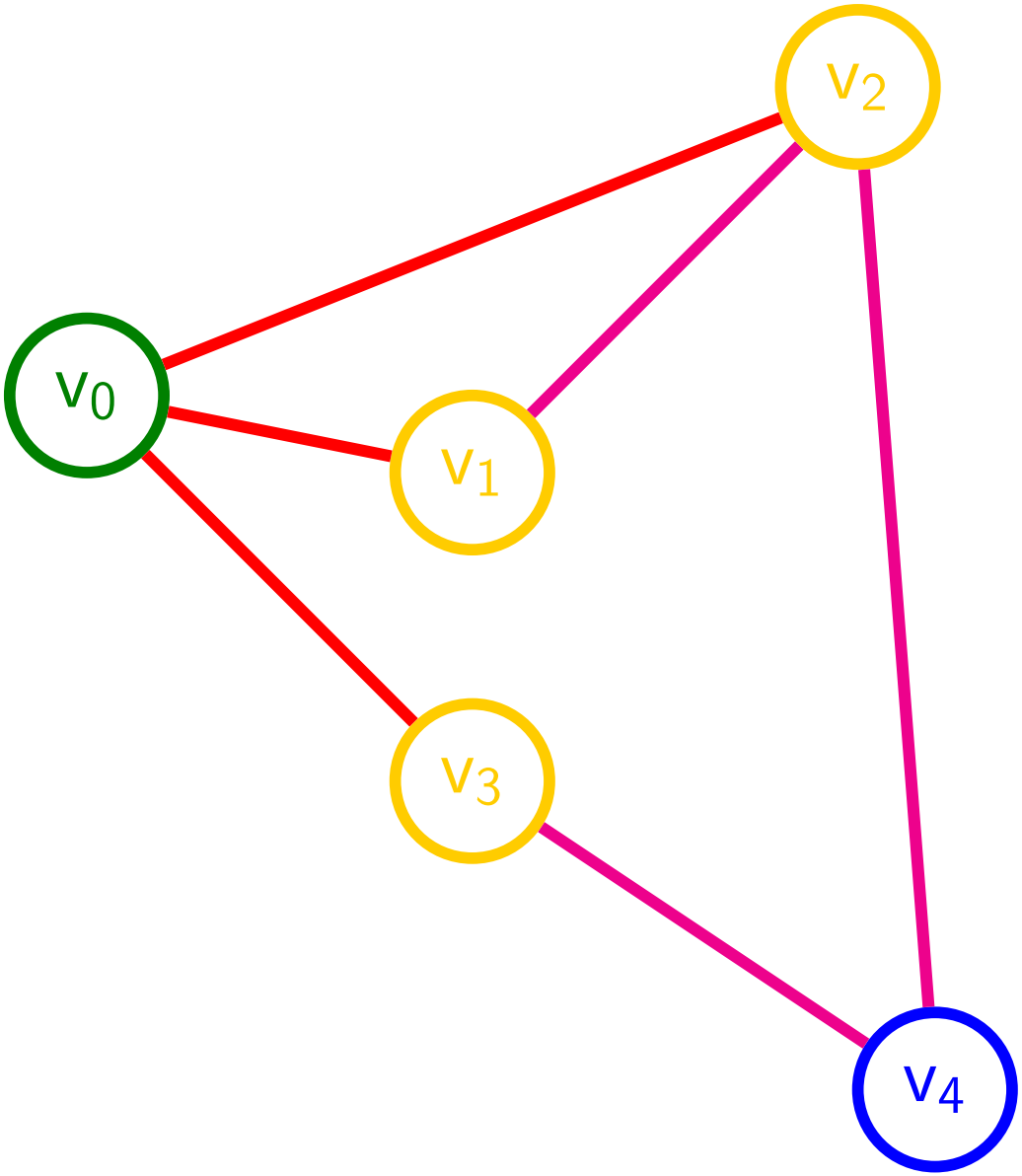
Busca em Largura - Ideia



Busca em Largura - Ideia



Busca em Largura - Ideia



Grafos – Busca em Largura

Em termos de implementação, **temos que visitar um dos vizinhos de cada vez**.

Grafos – Busca em Largura

Em termos de implementação, **temos que visitar um dos vizinhos de cada vez**.

Mas, diferentemente da Busca em Profundidade, **primeiro visitaremos todos os vizinho do nó atual para só depois visitarmos os vizinhos dos vizinhos**.

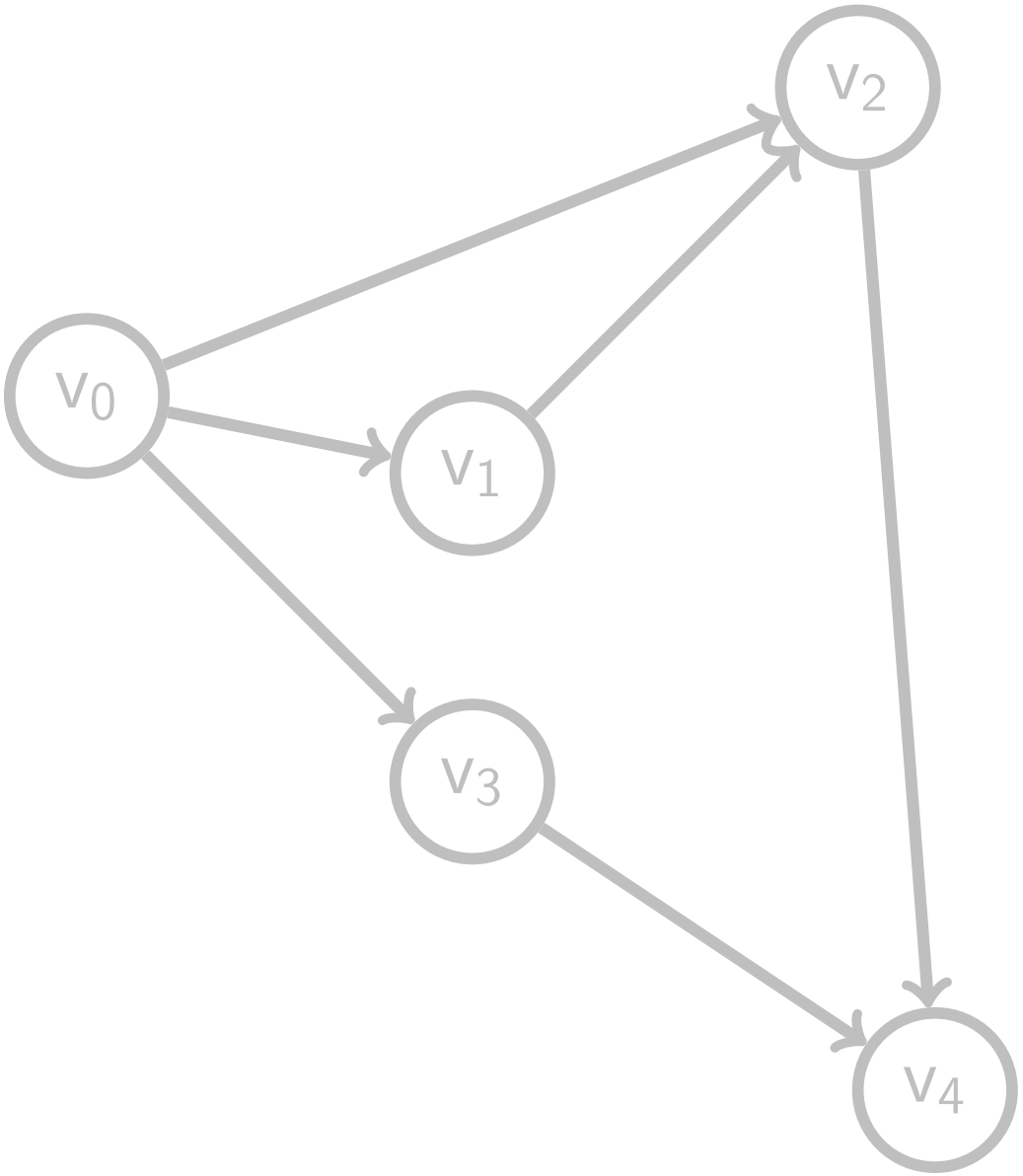
Grafos – Busca em Largura

Em termos de implementação, **temos que visitar um dos vizinhos de cada vez**.

Mas, diferentemente da Busca em Profundidade, **primeiro visitaremos todos os vizinho do nó atual para só depois visitarmos os vizinhos dos vizinhos**.

A cada vértice visitado, **colocaremos numa fila seus vizinhos**, para que sejam visitados apenas após todos aqueles que já estão na fila.

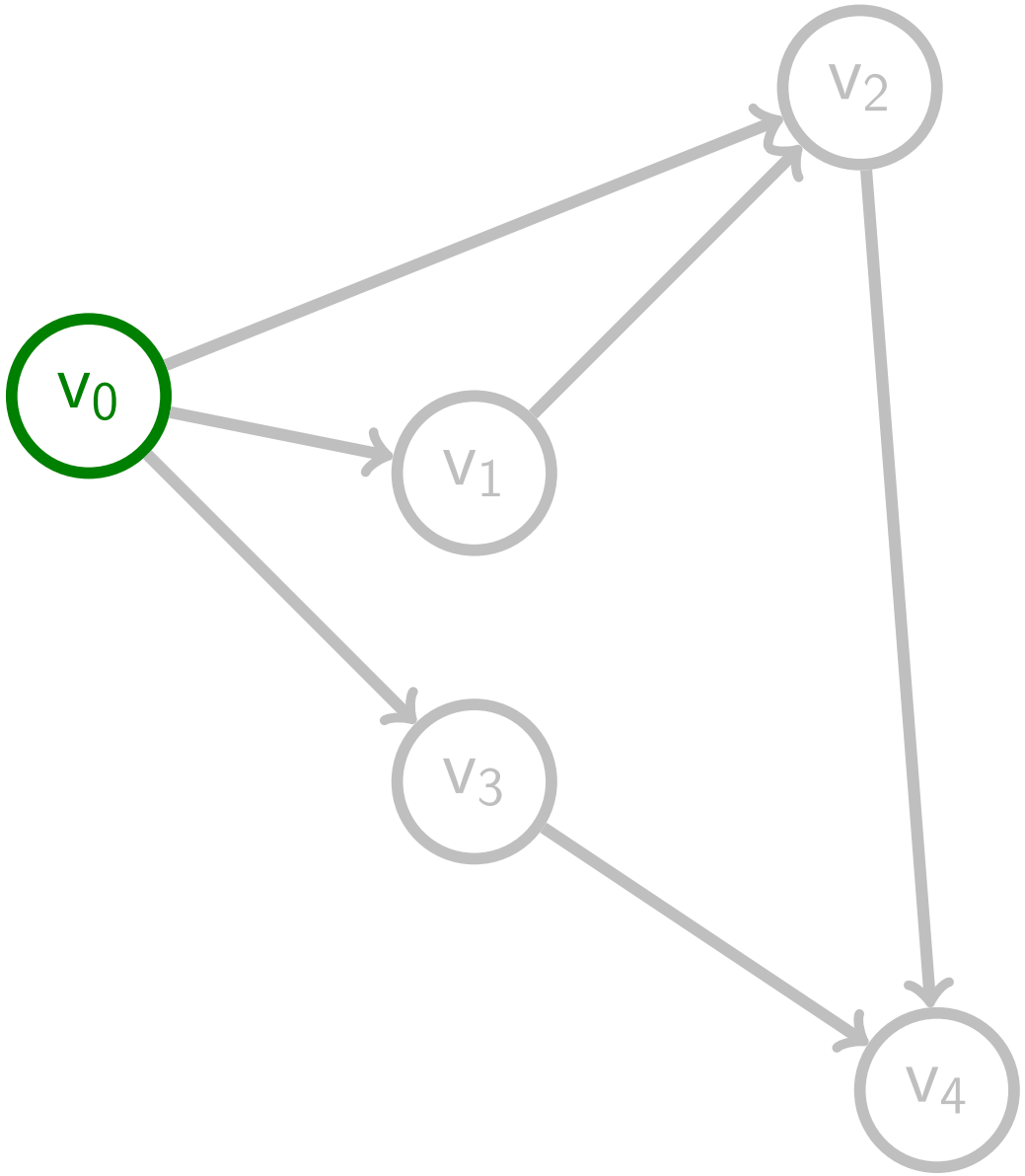
Busca em Largura



Fila:

Processados:

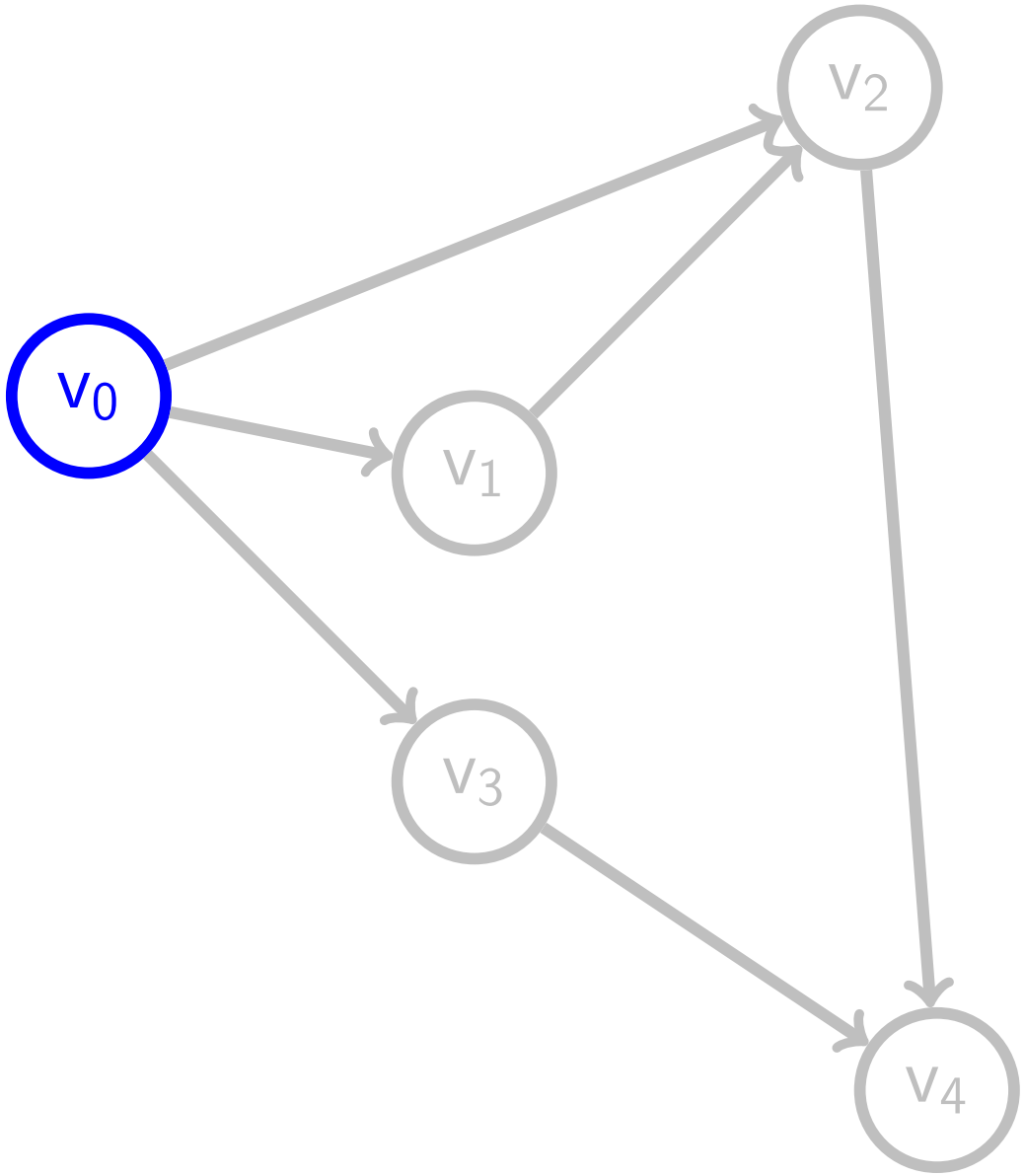
Busca em Largura




```
Fila:->v0

Processados:
```

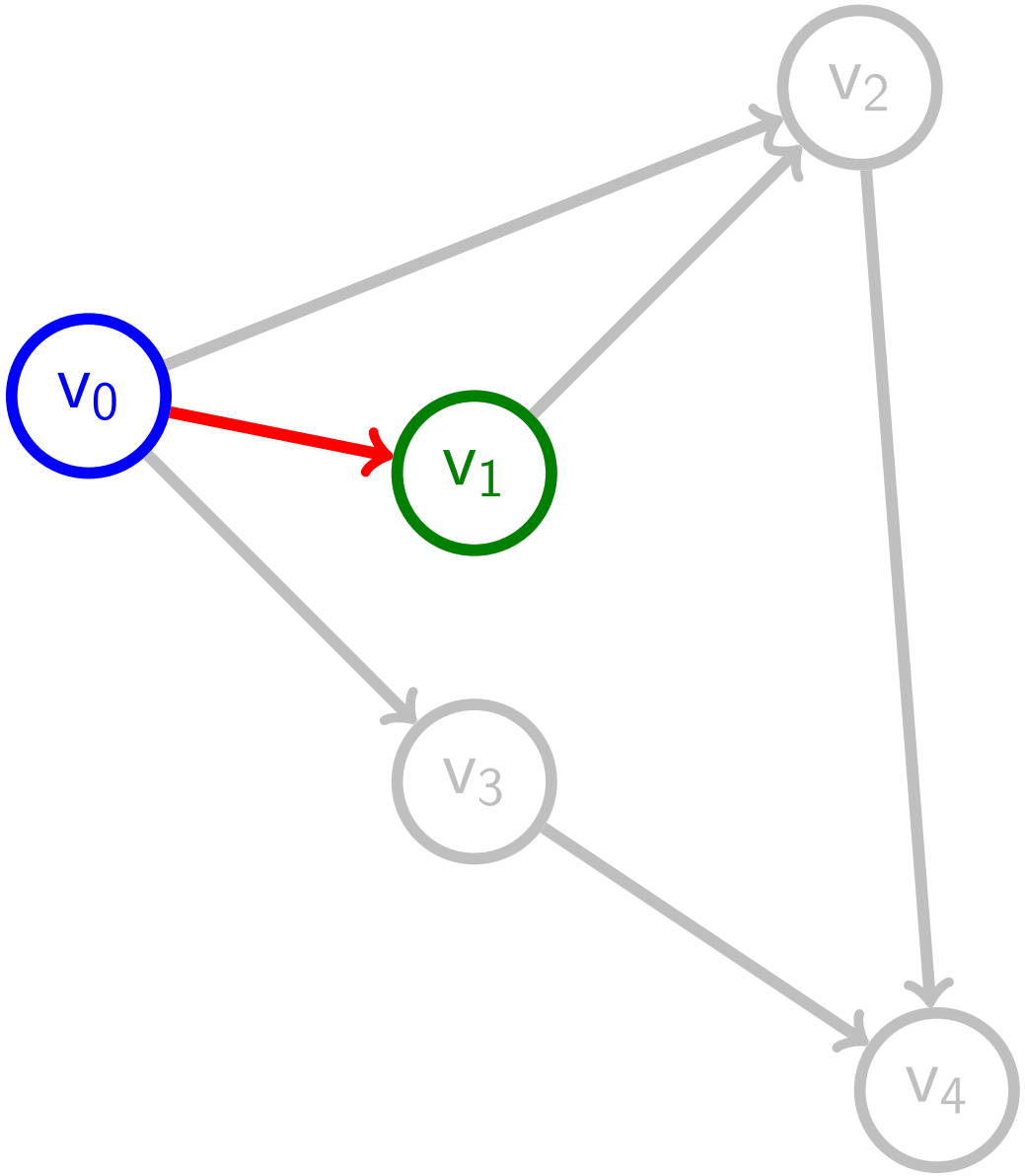
Busca em Largura



Fila:

Processados:
v0

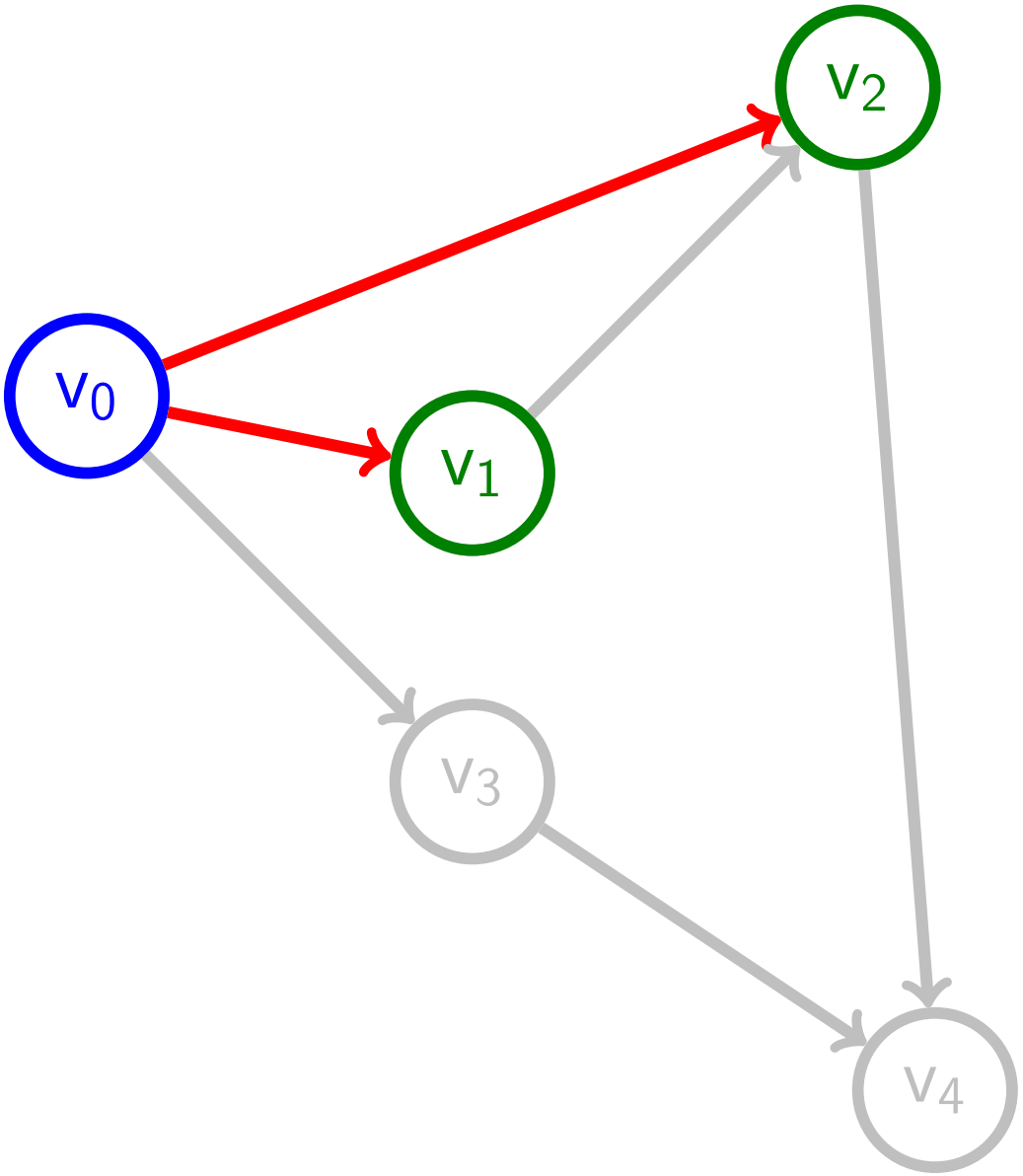
Busca em Largura




```
Fila:->v1

Processados:
v0
```

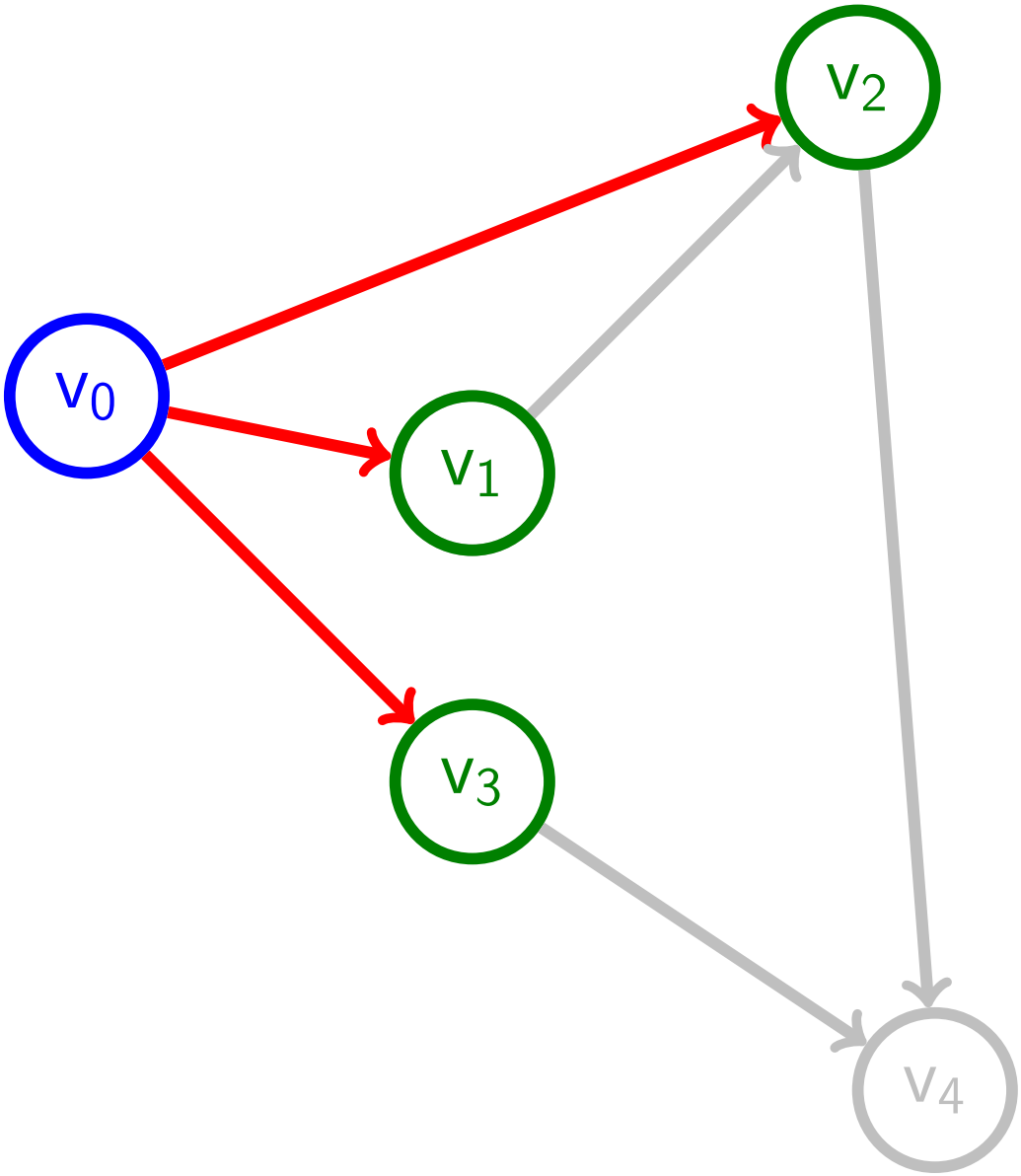
Busca em Largura




```
Fila:->v1->v2

Processados:
v0
```

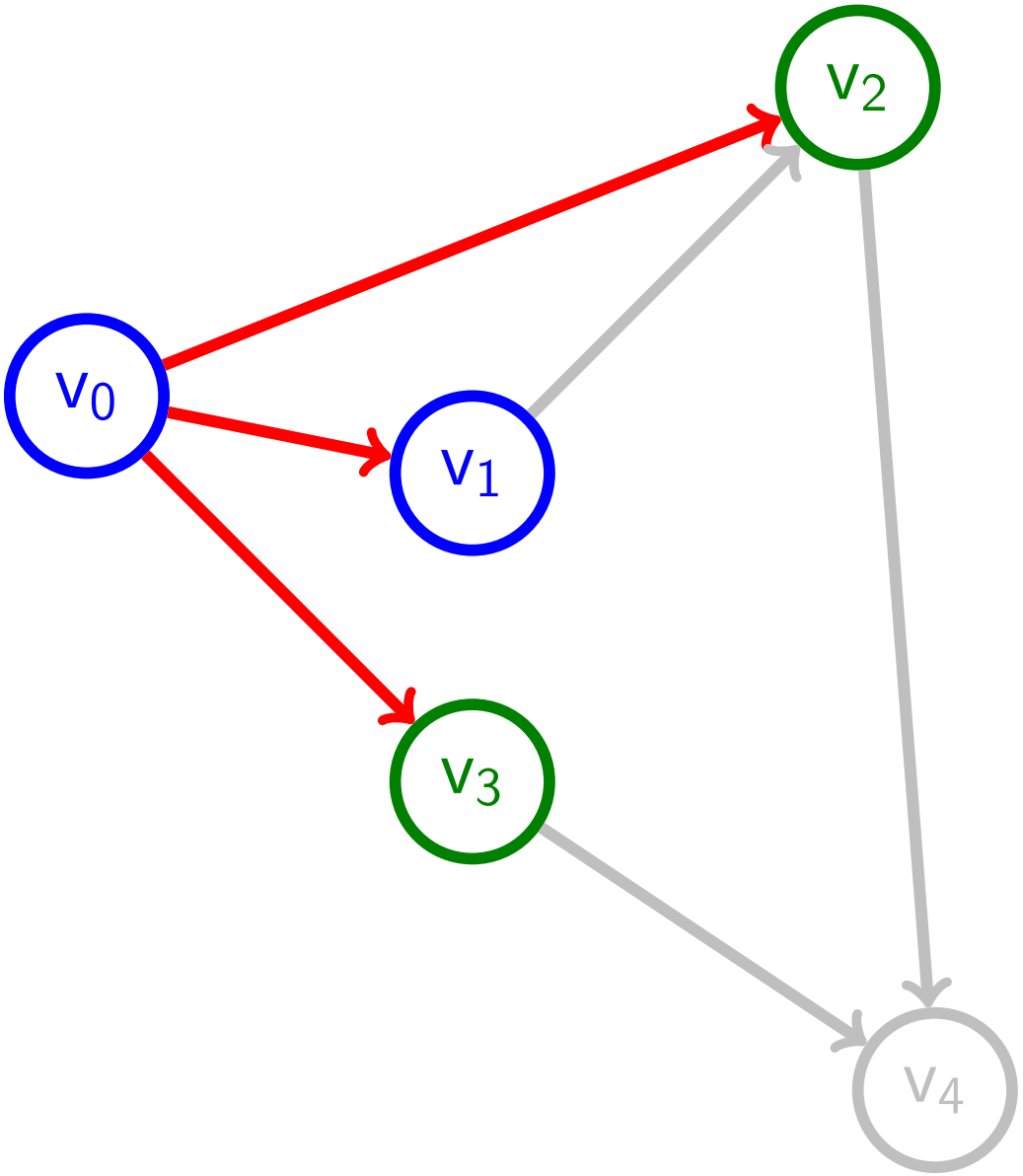
Busca em Largura




```
Fila:->v1->v2->v3

Processados:
v0
```

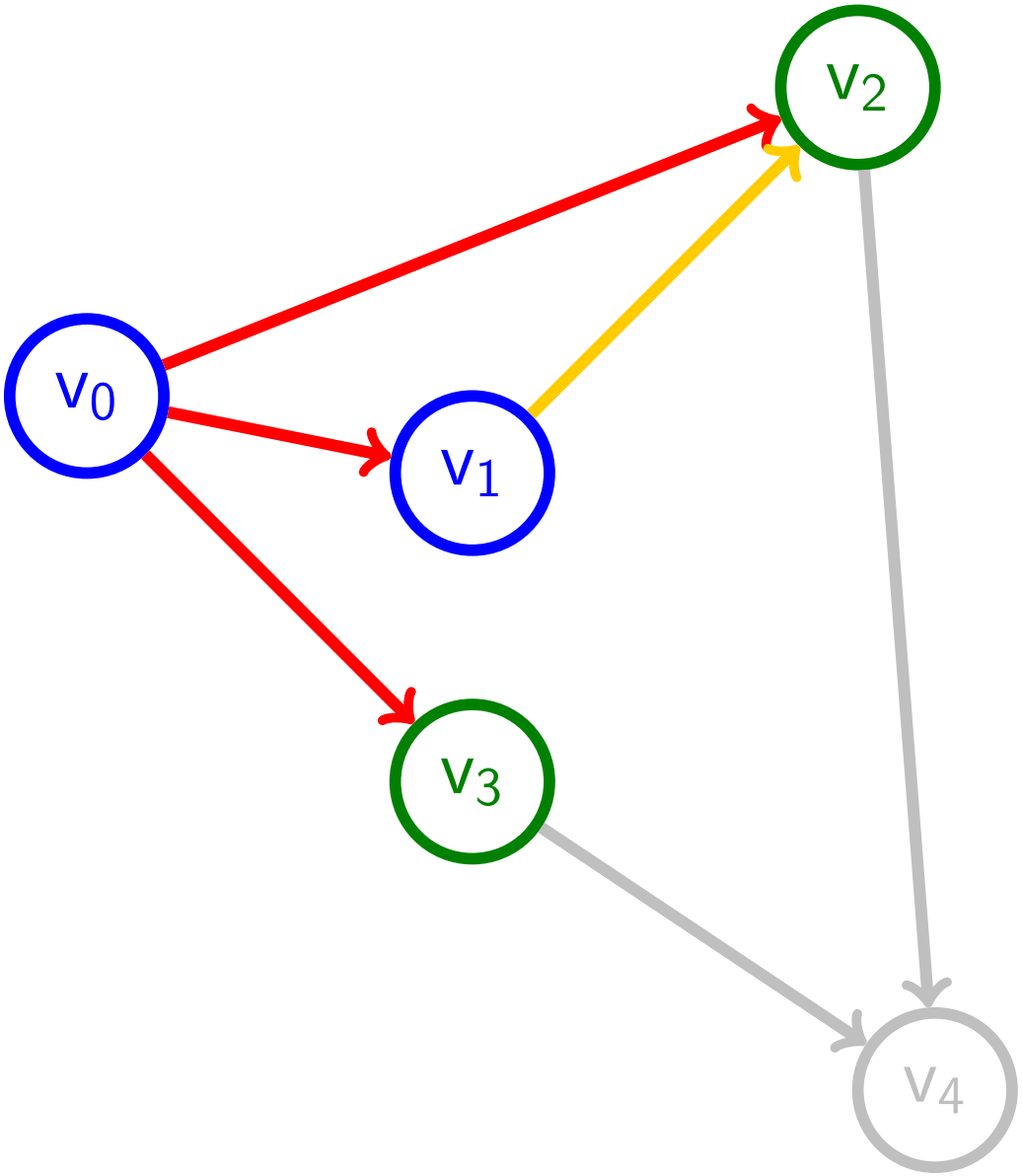
Busca em Largura




```
Fila:->v2->v3

Processados:
v0 v1
```

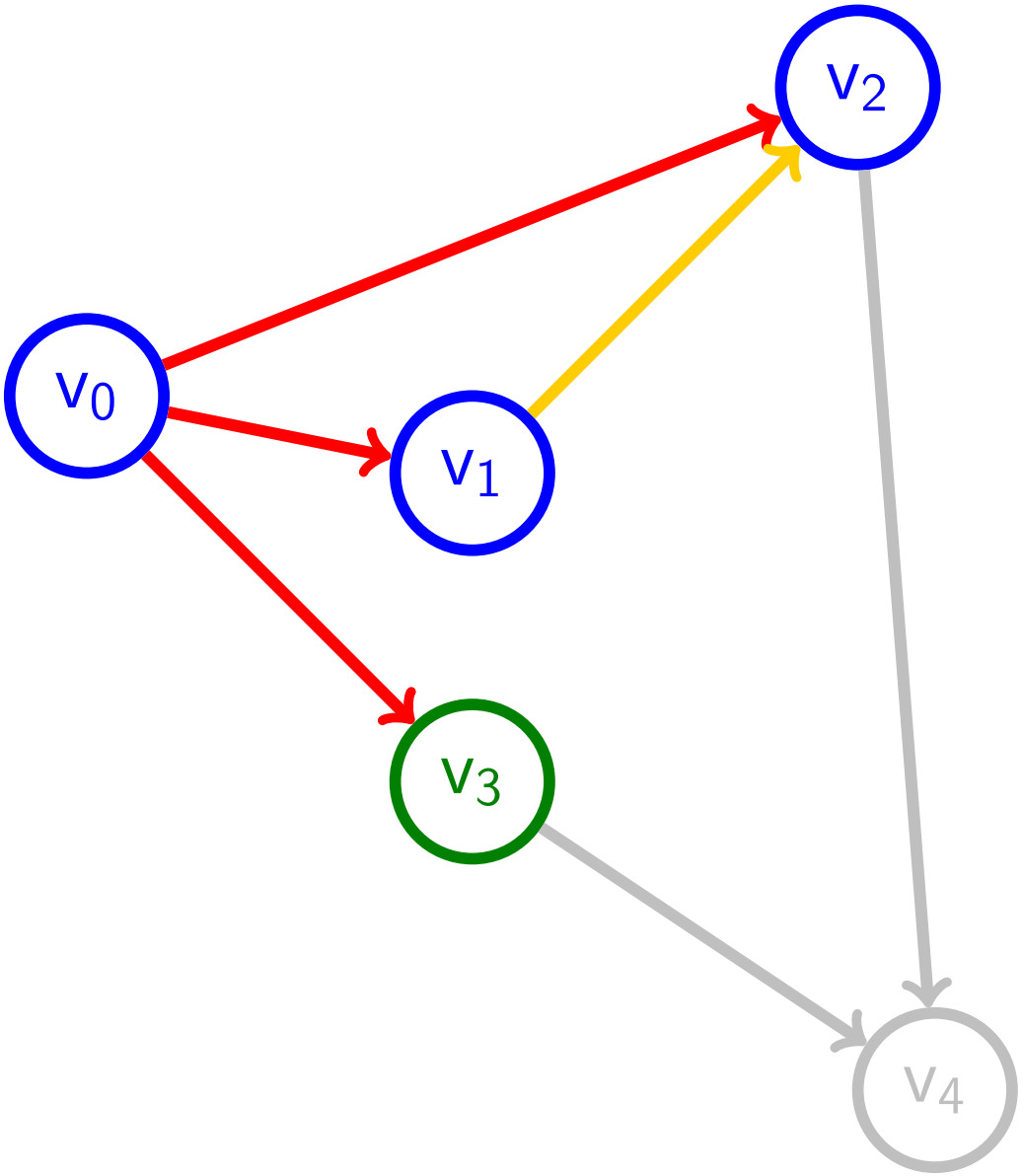
Busca em Largura




```
Fila:->v2->v3

Processados:
v0 v1
```

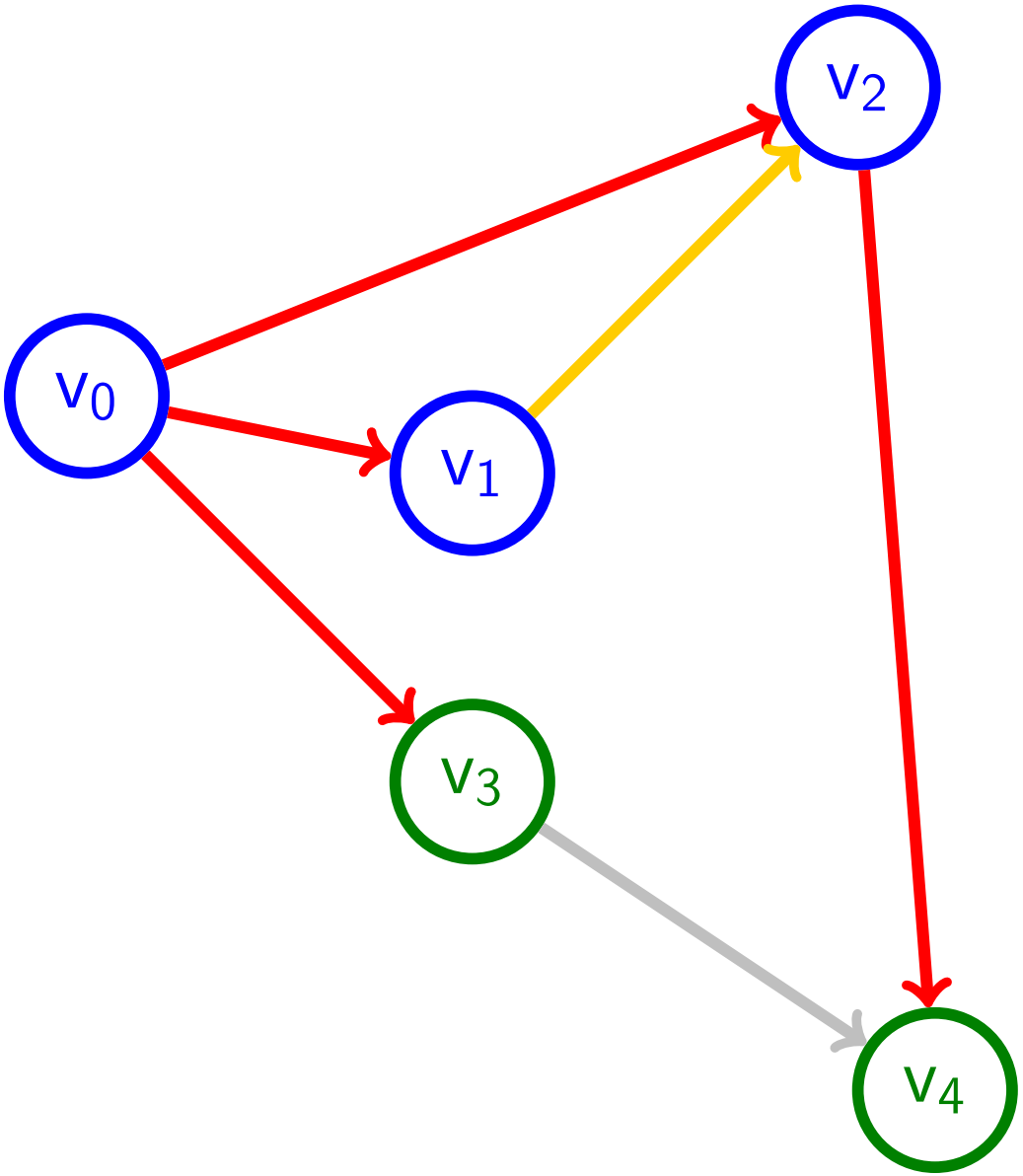
Busca em Largura




```
Fila:->v3

Processados:
v0 v1 v2
```

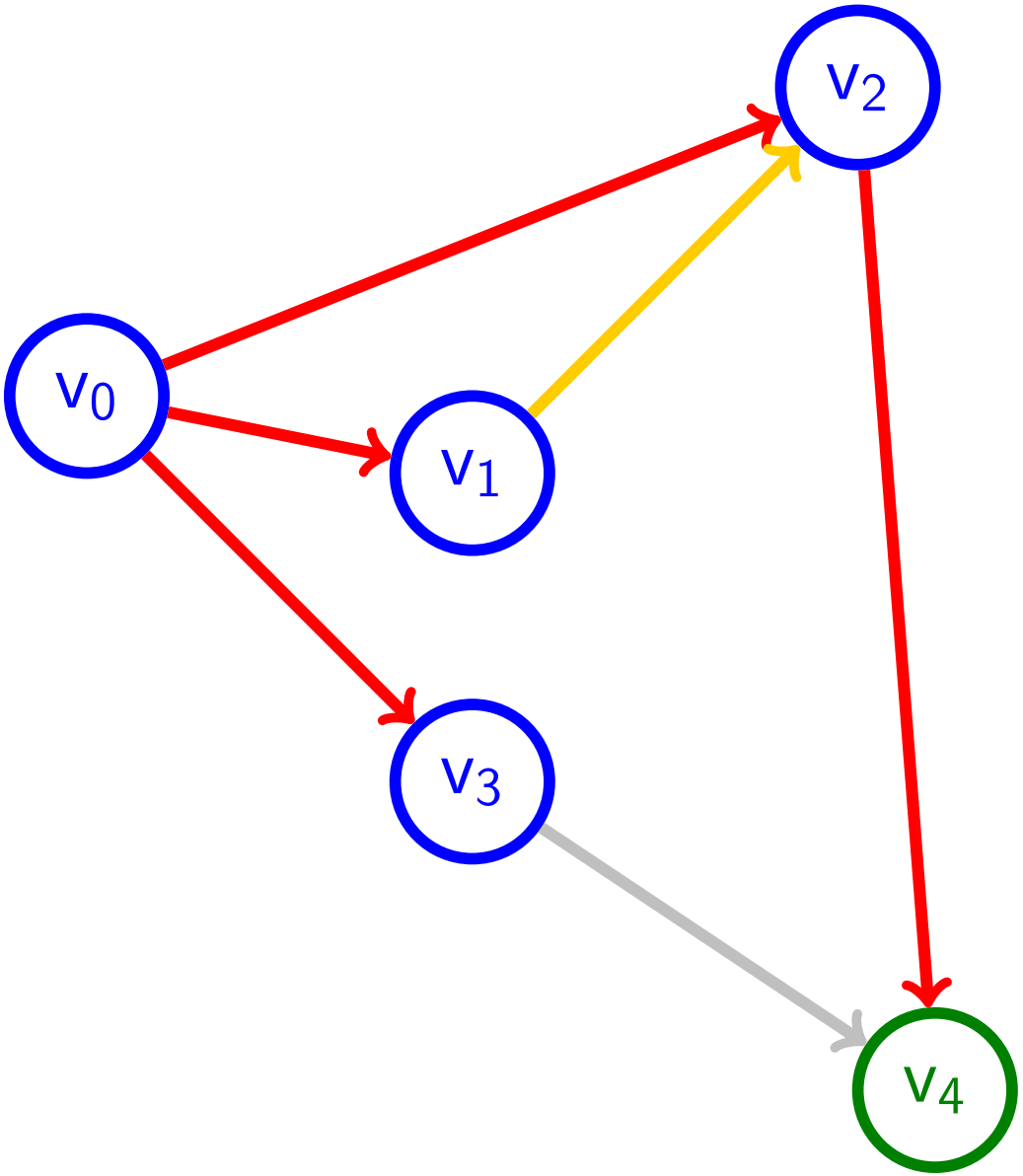
Busca em Largura




```
Fila:->v3->v4

Processados:
v0 v1 v2
```

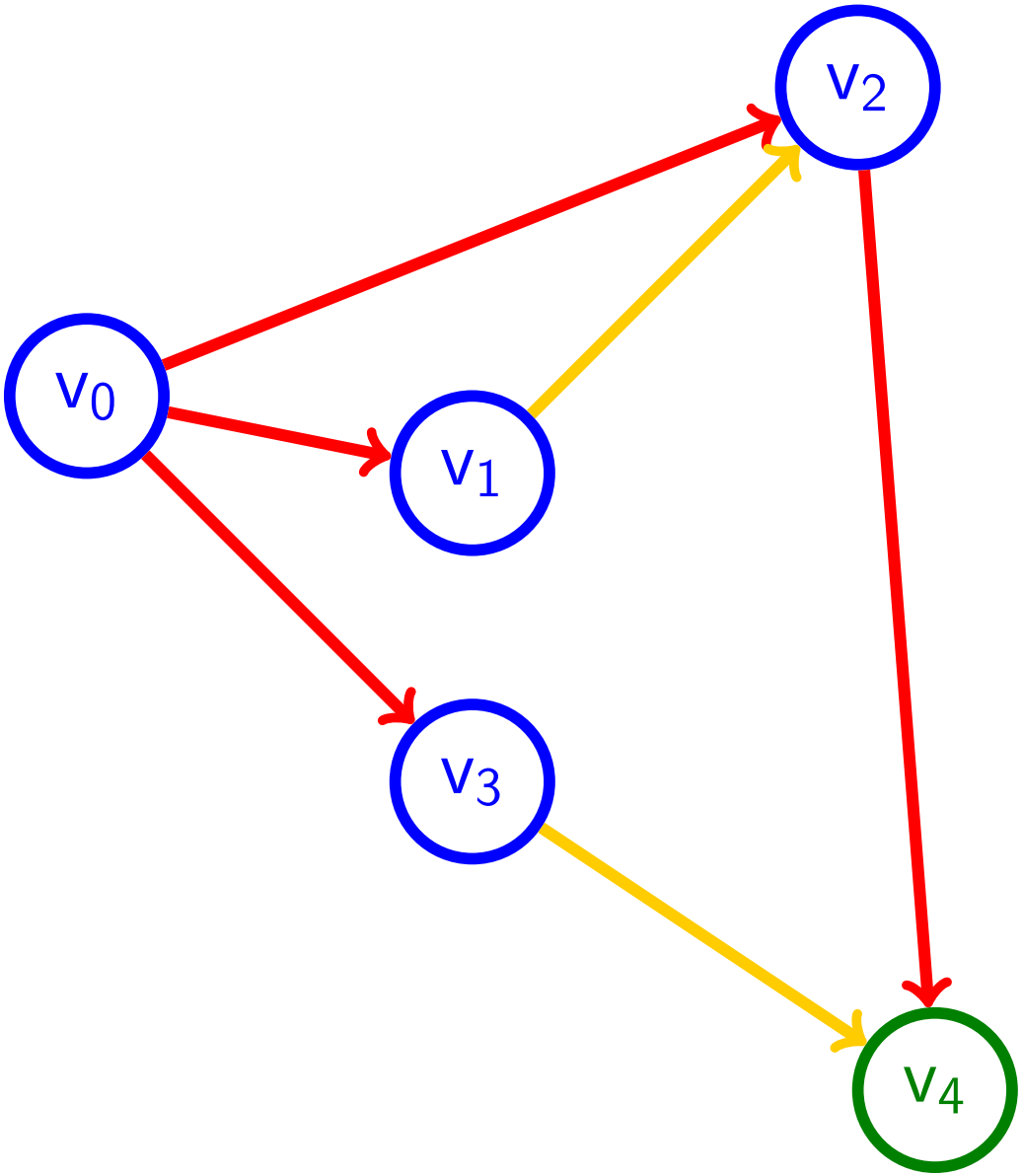
Busca em Largura




```
Fila:->v4

Processados:
v0 v1 v2 v3
```

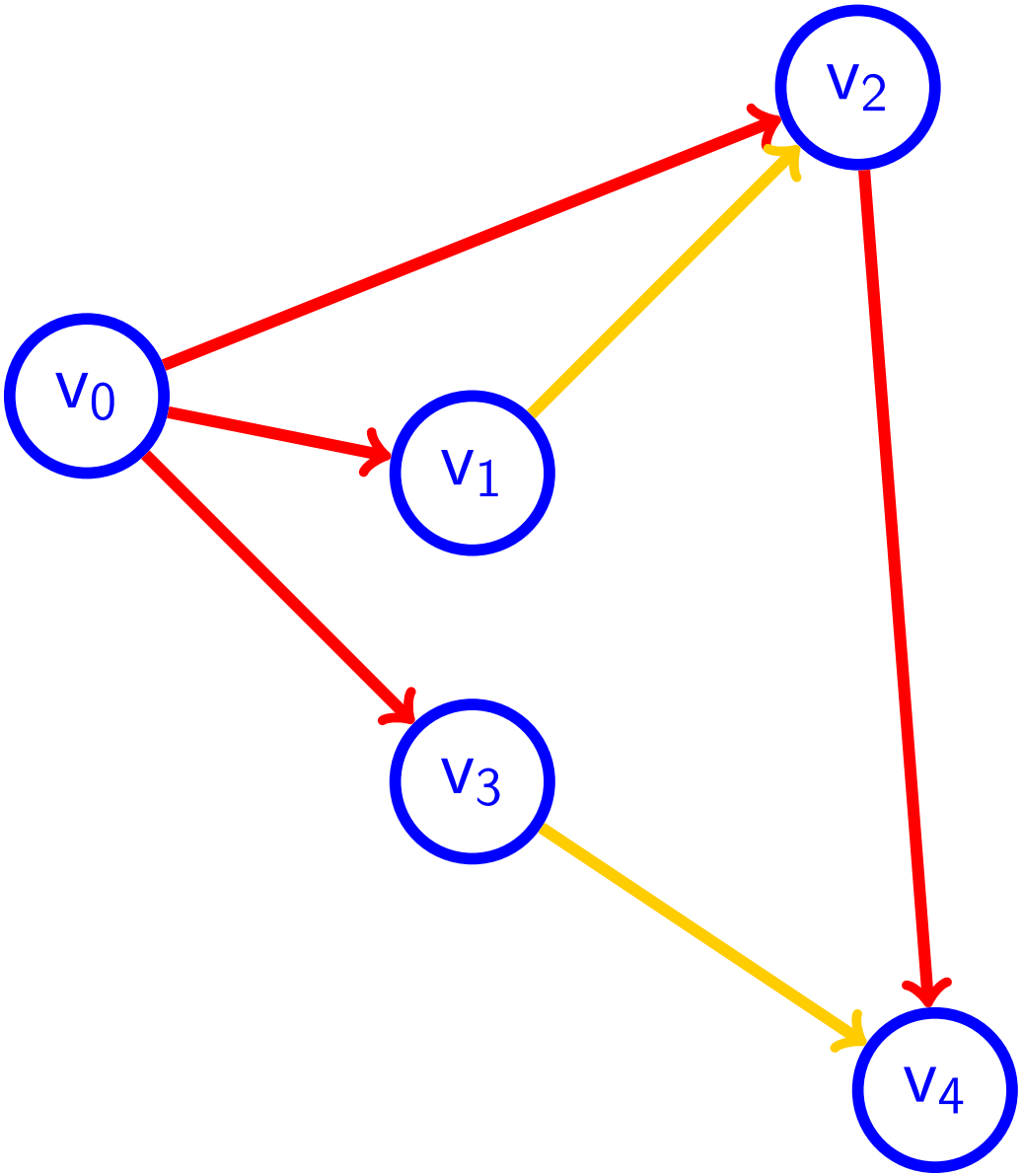
Busca em Largura




```
Fila:->v4

Processados:
v0 v1 v2 v3
```

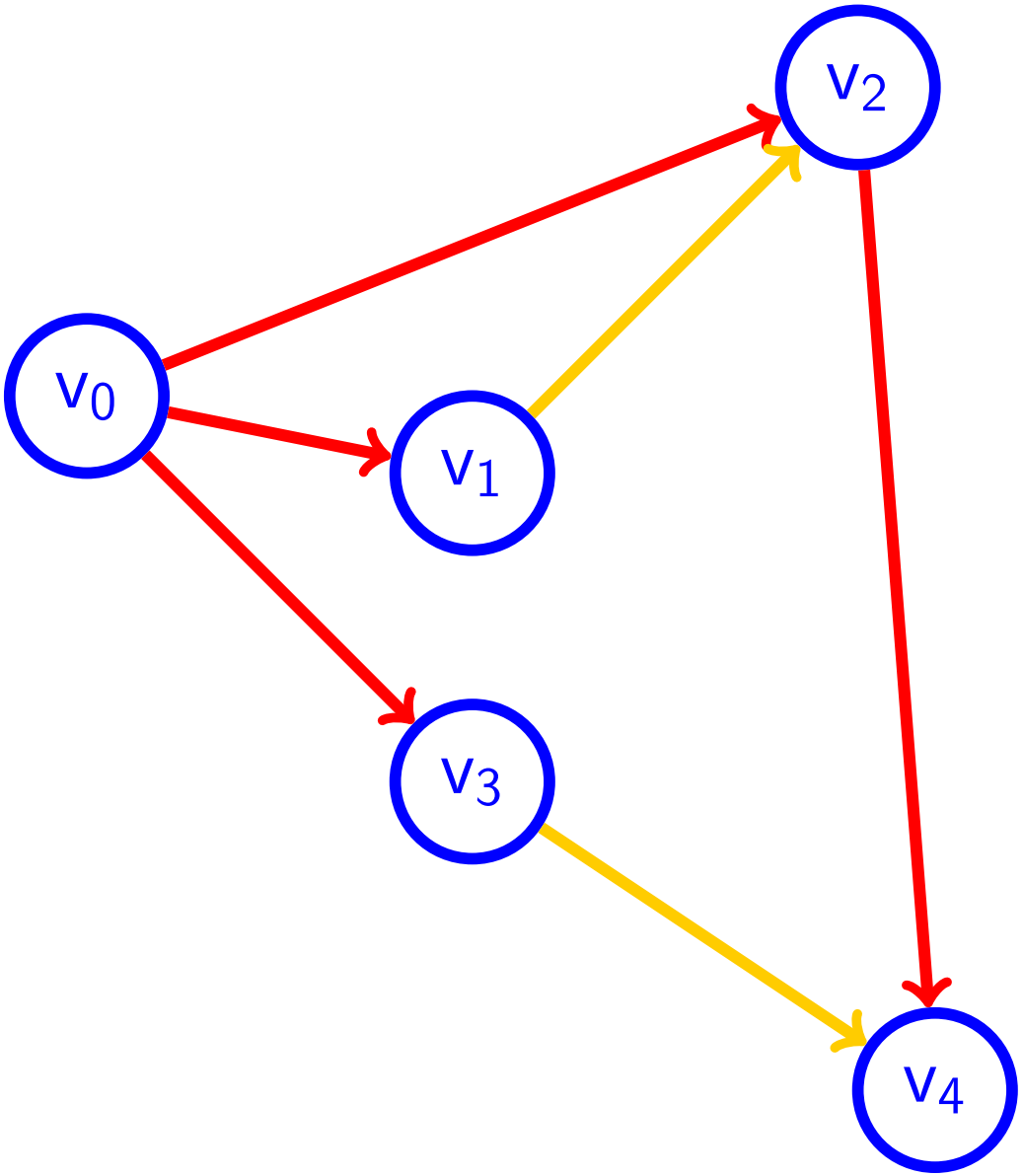
Busca em Largura



Fila:

Processados:
v0 v1 v2 v3 v4

Busca em Largura

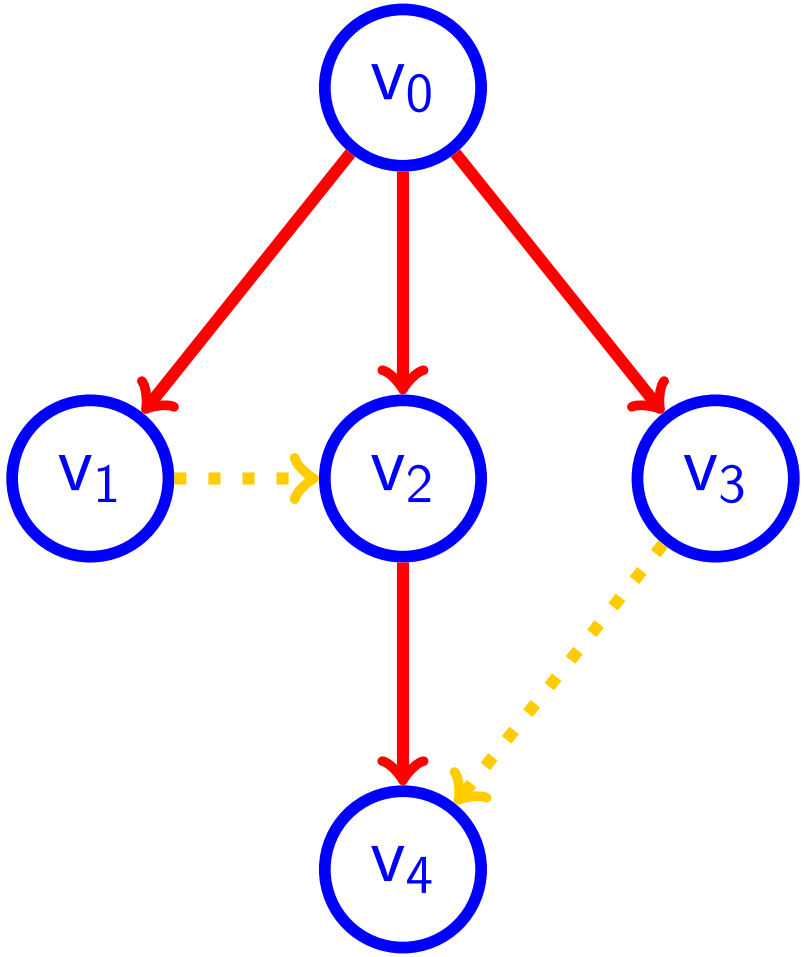


`Fila:`

`Processados:`
`v0 v1 v2 v3 v4`

A **árvore** produzida com as arestas a partir das quais visitamos cada vértice pela primeira vez permite a descoberta do **caminho de menor comprimento** entre a raiz (vértice inicial) e os demais.

Árvore da Busca em Largura



A **árvore** produzida com as arestas a partir das quais visitamos cada vértice pela primeira vez permite a descoberta do **caminho de menor comprimento** entre a raiz (vértice inicial) e os demais.

Busca em Largura - Fila

- Precisaremos **'enfileirar'** os vizinhos de cada vértice visitado.

Busca em Largura - Fila

- Precisaremos **'enfileirar'** os vizinhos de cada vértice visitado.
- Usar a pilha de recursão não funcionaria aqui! Precisamos de uma **fila**.

Busca em Largura - Fila

- Precisaremos **'enfileirar'** os vizinhos de cada vértice visitado.
- Usar a pilha de recursão não funcionaria aqui! Precisamos de uma **fila**.
- **Fila** é aquela estrutura de dados em que as **inserções ocorrem no final** e as **exclusões no início**.

Fila - Representação

```
typedef struct auxNo{
   int valor;
   struct auxNo* prox;
} No;

typedef struct{
   No* inicio;
   No* fim;
} Fila;
```

Fila - Inicialização

```
void inicializaFila(Fila* f){
   f->inicio = NULL;
   f->fim = NULL;
}
```

Fila - Inicialização

```
void inicializaFila(Fila* f){
   f->inicio = NULL;
   f->fim = NULL;
}
```

O(1)

Fila - Inicialização

```
void inicializaFila(Fila* f){
  f->inicio = NULL;
  f->fim = NULL;
}
```

O(1)

```
bool filaVazia(Fila* f){
  if (!f->inicio) return true;
  return false;
}
```

Fila - Inicialização

```
void inicializaFila(Fila* f){
  f->inicio = NULL;
  f->fim = NULL;
}
```

O(1)

```
bool filaVazia(Fila* f){
  if (!f->inicio) return true;
  return false;
}
```

O(1)

Fila - Inserção

```
void insereFila(Fila* f, int valor){
  No* novo = (No*) malloc(sizeof(No));
  novo->prox = NULL;
  novo->valor = valor;
  if (f->inicio == NULL){
    f->inicio = novo;
    f->fim = novo;
  }else{
    f->fim->prox = novo;
    f->fim = novo;
  }
}
```

Fila - Inserção

```
void insereFila(Fila* f, int valor){
  No* novo = (No*) malloc(sizeof(No));
  novo->prox = NULL;
  novo->valor = valor;
  if (f->inicio == NULL){
    f->inicio = novo;
    f->fim = novo;
  }else{
    f->fim->prox = novo;
    f->fim = novo;
  }
}
```

O(1)

Fila - Exclusão

```
int excluiFila(Fila* f){
   if (!f->inicio) return -1;
   No* atual = f->inicio;
   int valor = atual->valor;
   f->inicio = atual->prox;
   if (!f->inicio) f->fim = NULL;
   free(atual);
   return valor;
}
```

Fila - Exclusão

```
int excluiFila(Fila* f){
   if (!f->inicio) return -1;
   No* atual = f->inicio;
   int valor = atual->valor;
   f->inicio = atual->prox;
   if (!f->inicio) f->fim = NULL;
   free(atual);
   return valor;
}
```

O(1)

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){



}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;









}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f;




}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f;
  inicializaFila(&f);
}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;









}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;
  while( !filaVazia(&f) ){









  }
}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;
  while( !filaVazia(&f) ){
    atual = excluiFila(&f) ;




  }
}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;
  while( !filaVazia(&f) ){
    atual = excluiFila(&f) ;
    for (x=0;x<g->numVertices;x++)




  }
}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;
  while( !filaVazia(&f) ){
    atual = excluiFila(&f) ;
    for (x=0;x<g->numVertices;x++)
      if (!visitado[x] && g->matriz[atual][x]!=false){



      }
  }
}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;
  while( !filaVazia(&f) ){
    atual = excluiFila(&f) ;
    for (x=0;x<g->numVertices;x++)
      if (!visitado[x] && g->matriz[atual][x]!=false){
        insereFila(&f, x) ;
        visitado[x] = true;
      }
  }
}
```

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;
  while( !filaVazia(&f) ){
    atual = excluiFila(&f) ;
    for (x=0;x<g->numVertices;x++)
      if (!visitado[x] && g->matriz[atual][x]!=false){
        insereFila(&f, x) ;
        visitado[x] = true;
      }
  }
}
```

$O(V^2)$

Busca em Largura - Matrizes de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial);
  visitado[inicial] = true;
  while( !filaVazia(&f) ){
    atual = excluiFila(&f) ;
    for (x=0;x<g->numVertices;x++)
      if (!visitado[x] && g->matriz[atual][x]!= ARESTA_INVALIDA ){
        insereFila(&f, x) ;
        visitado[x] = true;
      }
  }
}
```

$O(V^2)$

Busca em Largura - Listas de Adjacências

```
void buscaEmLargura(Grafo* g, int inicial, bool* visitado){
  if (!g || inicial < 0 || inicial >= g->numVertices) return;
  int x, atual;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,inicial) ;
  visitado[inicial] = true;
  ...
```

Busca em Largura - Listas de Adjacências

```
...
ElemLista* end;
while( !filaVazia(&f) ){
   atual = excluiFila(&f);









   }
}
```

Busca em Largura - Listas de Adjacências

```
...
ElemLista* end;
while( !filaVazia(&f) ){
   atual = excluiFila(&f);
   end = g->A[atual];




   }
}
```

Busca em Largura - Listas de Adjacências

```
...
ElemLista* end;
while( !filaVazia(&f) ){
   atual = excluiFila(&f);
   end = g->A[atual];
   while (end){




      end = end->prox;
   }
}
}
```

Busca em Largura - Listas de Adjacências

```
...
ElemLista* end;
while( !filaVazia(&f) ){
   atual = excluiFila(&f);
   end = g->A[atual];
   while (end){
      x = end->vertice;
      if (!visitado[x]){
         insereFila(&f, x);
         visitado[x] = true;
      }
      end = end->prox;
   }
}
}
```

Busca em Largura - Listas de Adjacências

```
...
ElemLista* end;
while( !filaVazia(&f) ){
   atual = excluiFila(&f);
   end = g->A[atual];
   while (end){
      x = end->vertice;
      if (!visitado[x]){
         insereFila(&f, x);
         visitado[x] = true;
      }
      end = end->prox;
   }
}
}
```

$$O(V + E)$$

Busca em Largura - Listas de Adjacências

```
void buscaEmLarguraCompleta(Grafo* g){
  if (!g || g->numVertices<1) return;
  bool* visitado = malloc(sizeof(bool)*g->numVertices);
  int x;
  for (x=0;x<g->numVertices;x++) visitado[x] = false;
  for (x=0;x<g->numVertices;x++)
    if (!visitado[x]) buscaEmLargura(g, x, visitado);
  free(visitado);
}
```

$$O(V + E)$$

Busca em Largura

Para que serve a Busca em Largura?

Pode ser usada (ou adaptada) para resolver diferentes problemas:

Busca em Largura

Para que serve a Busca em Largura?

Pode ser usada (ou adaptada) para resolver diferentes problemas:

- Encontrar o **caminho de menor comprimento entre dois vértices** (em termos de número de arestas percorridas) - solução de labirinto

Busca em Largura

Para que serve a Busca em Largura?

Pode ser usada (ou adaptada) para resolver diferentes problemas:

- Encontrar o **caminho de menor comprimento entre dois vértices** (em termos de número de arestas percorridas) - solução de labirinto
- Encontrar componentes conexos

Busca em Largura

Para que serve a Busca em Largura?

Pode ser usada (ou adaptada) para resolver diferentes problemas:

- Encontrar o **caminho de menor comprimento entre dois vértices** (em termos de número de arestas percorridas) - solução de labirinto
- Encontrar componentes conexos
- Verificar se um grafo é bipartido

Busca em Largura

Para que serve a Busca em Largura?

Pode ser usada (ou adaptada) para resolver diferentes problemas:

- Encontrar o **caminho de menor comprimento entre dois vértices** (em termos de número de arestas percorridas) - solução de labirinto
- Encontrar componentes conexos
- Verificar se um grafo é bipartido
- Auxiliar no problema de fluxo máximo

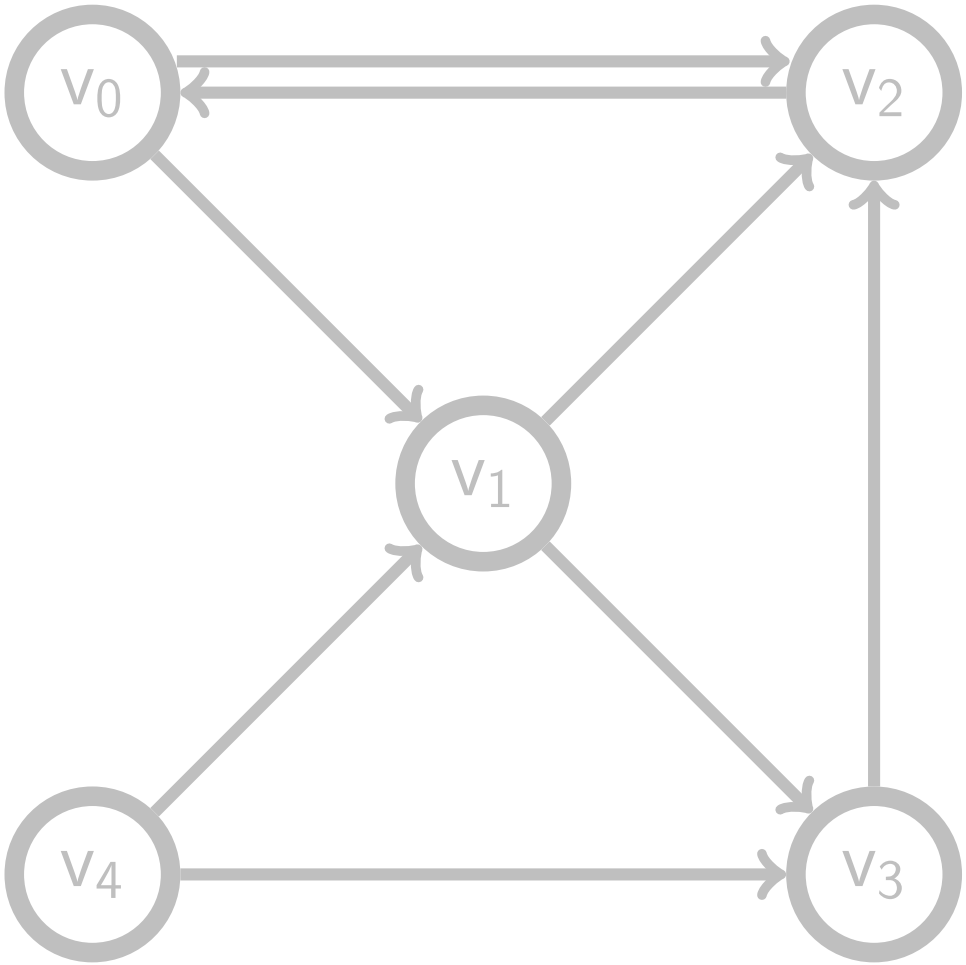
Busca em Largura

Para que serve a Busca em Largura?

Pode ser usada (ou adaptada) para resolver diferentes problemas:

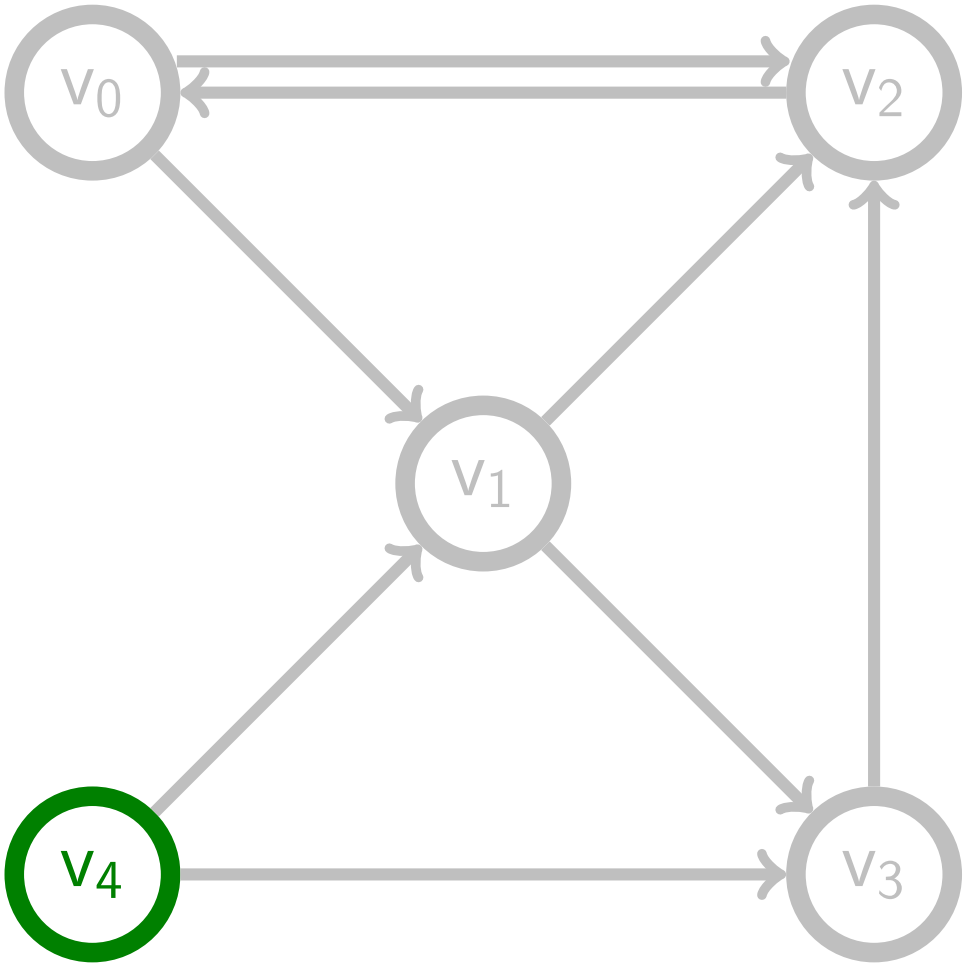
- Encontrar o **caminho de menor comprimento entre dois vértices** (em termos de número de arestas percorridas) - solução de labirinto
- Encontrar componentes conexos
- Verificar se um grafo é bipartido
- Auxiliar no problema de fluxo máximo
- ...

Encontrar uma Solução para um Labirinto



Início $v4$, destino $v0$

Encontrar uma Solução para um Labirinto

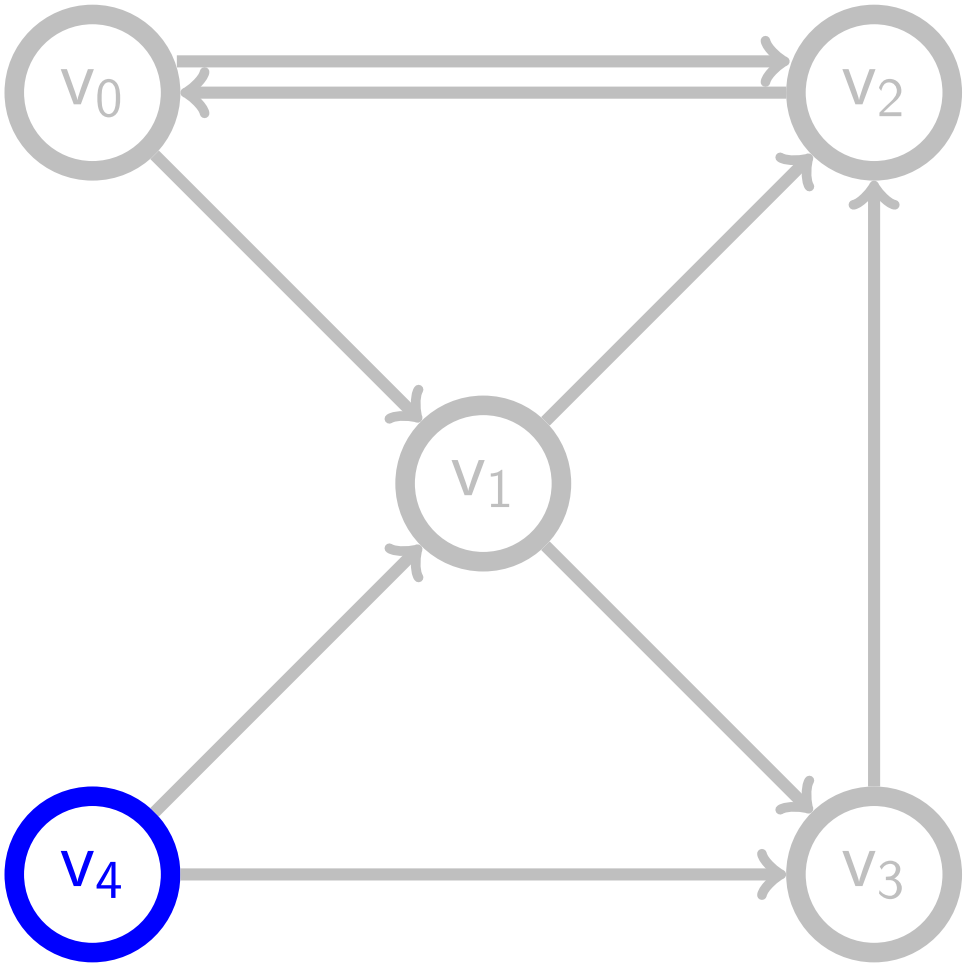


Início $v4$, destino $v0$

`Fila:->v4`

`Processados:`

Encontrar uma Solução para um Labirinto

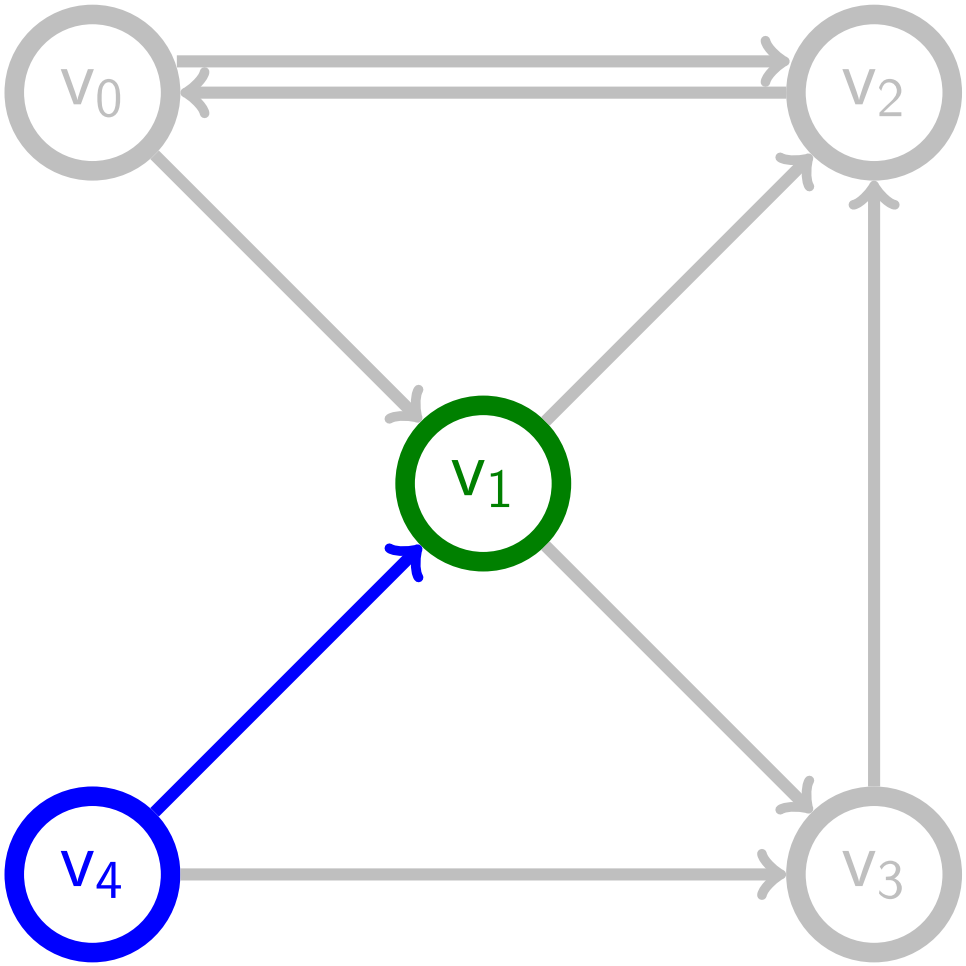


Início $v4$, destino $v0$

`Fila:`

`Processados:`
`v4`

Encontrar uma Solução para um Labirinto

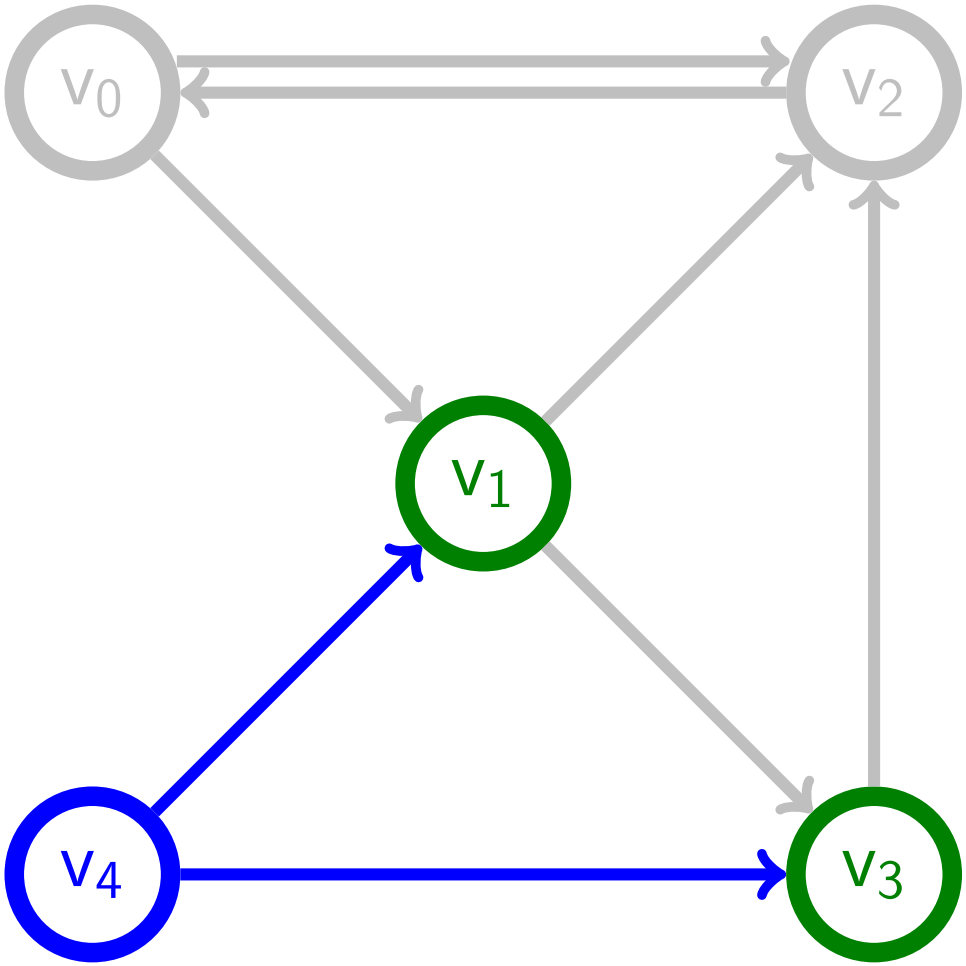


Início $v4$, destino $v0$

```
Fila:->v1

Processados:
v4
```

Encontrar uma Solução para um Labirinto

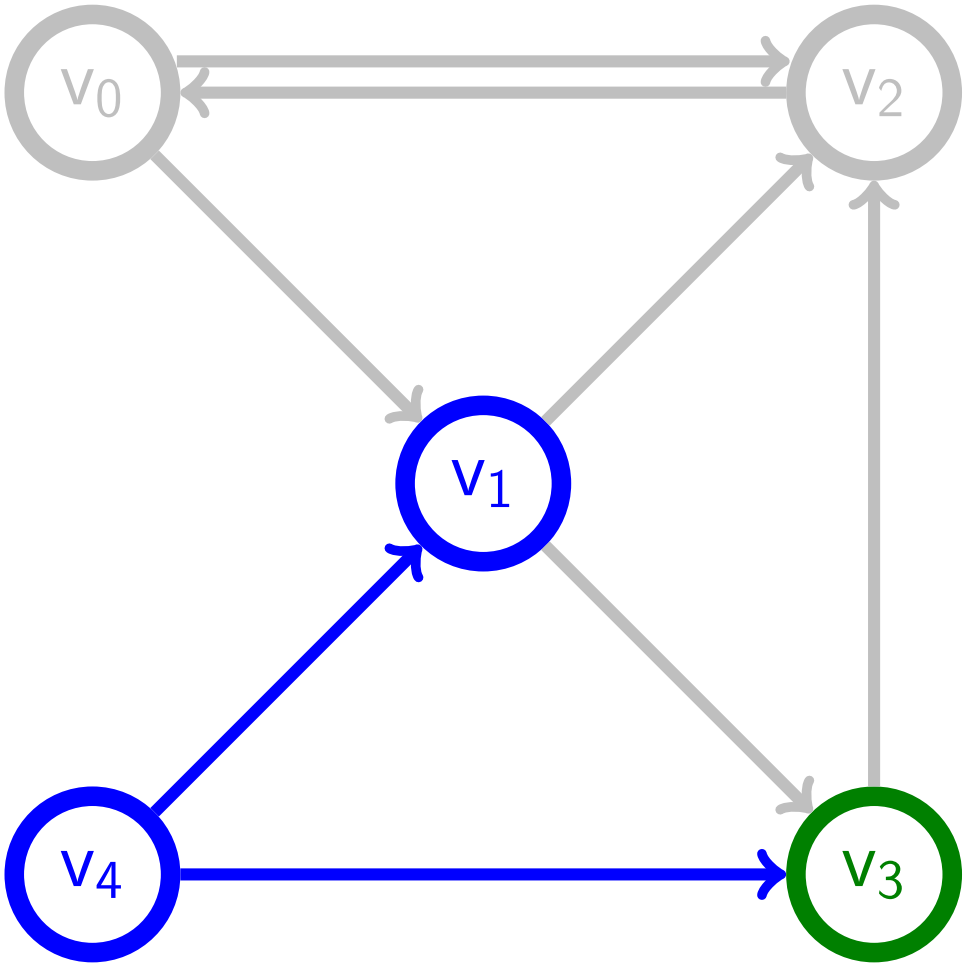


Início $v4$, destino $v0$

```
Fila:->v1->v3
```

```
Processados:
v4
```

Encontrar uma Solução para um Labirinto

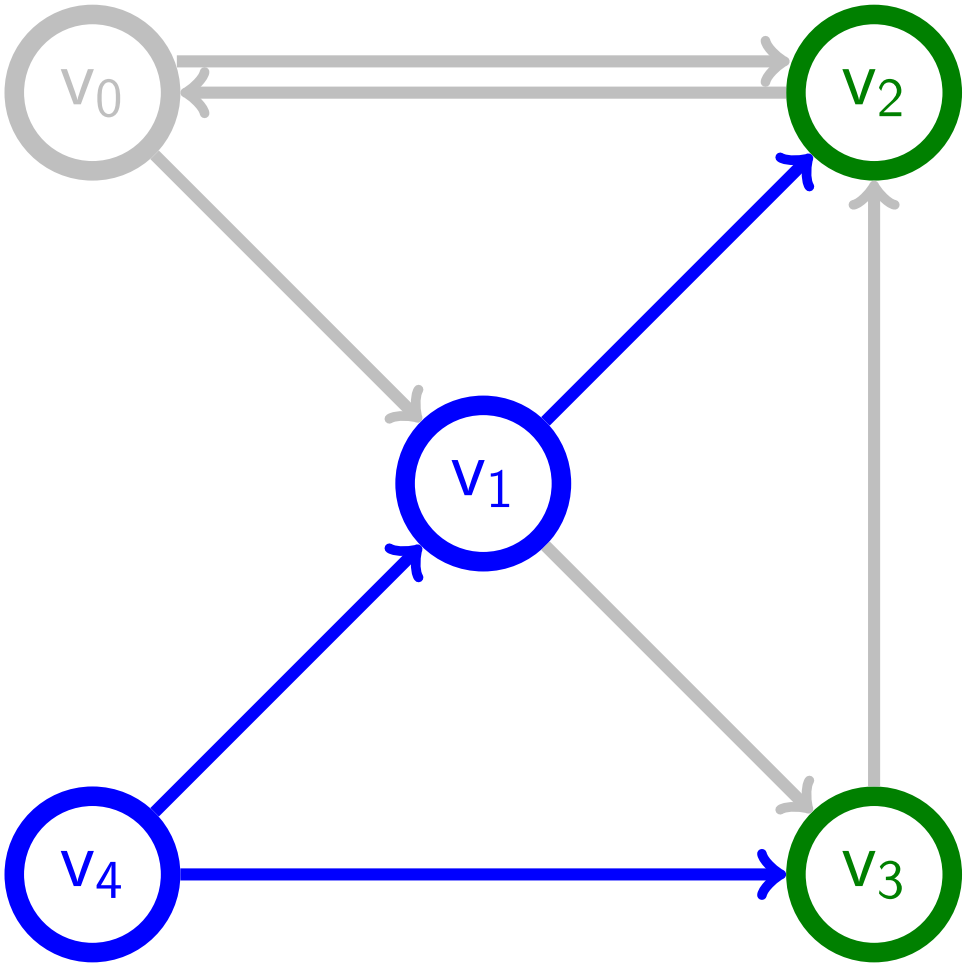


Início $v4$, destino $v0$

`Fila:->v3`

`Processados:`
`v4 v1`

Encontrar uma Solução para um Labirinto

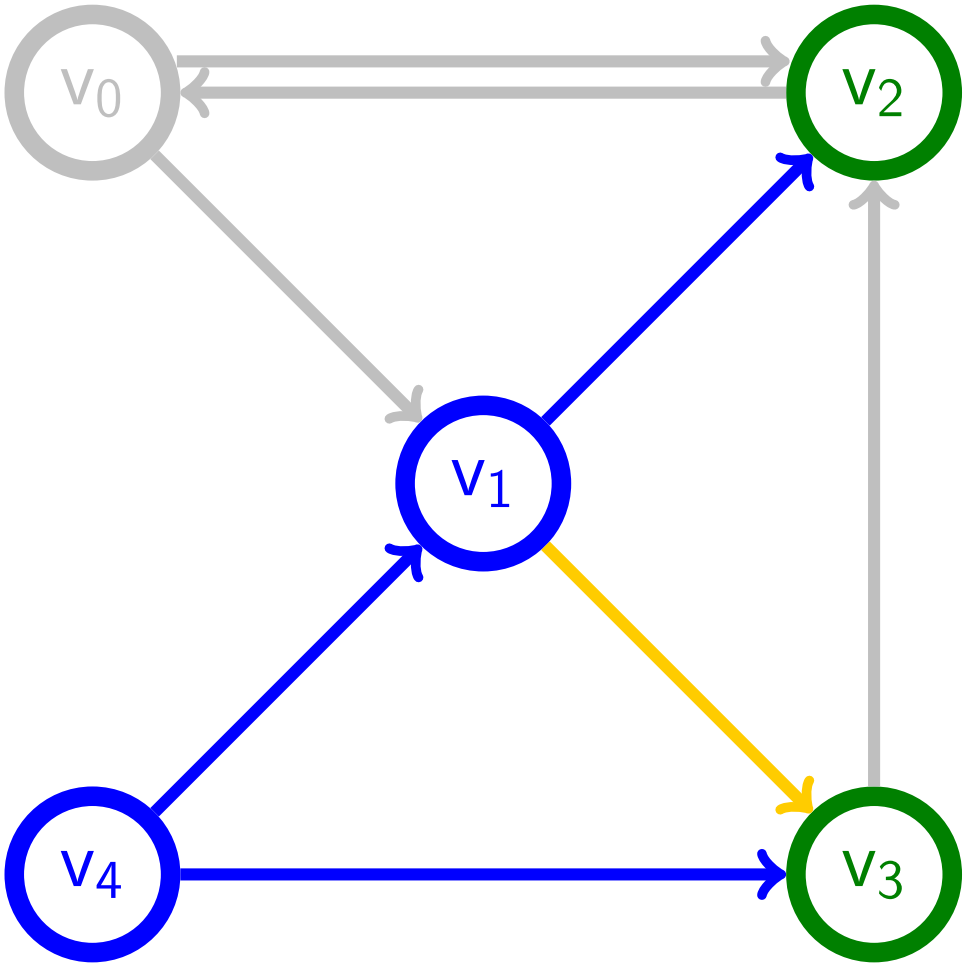


Início $v4$, destino $v0$

```
Fila:->v3->v2
```

```
Processados:
v4 v1
```

Encontrar uma Solução para um Labirinto

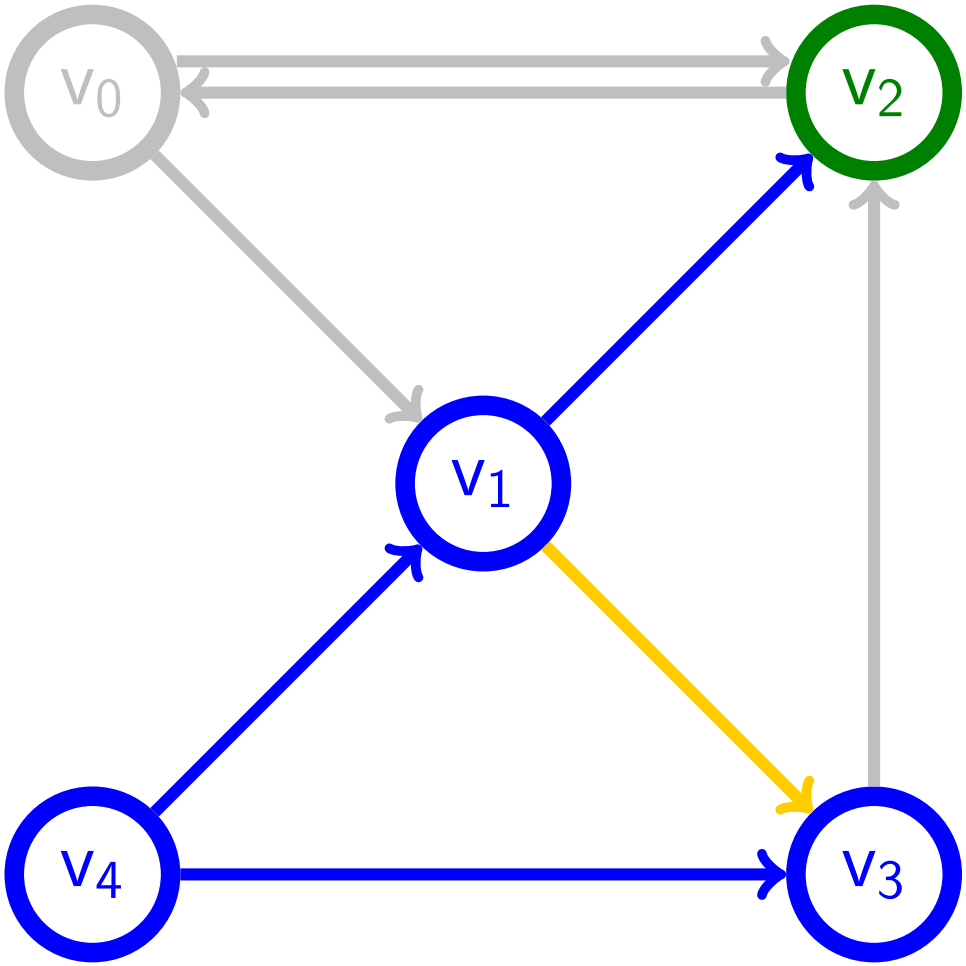


Início $v4$, destino $v0$

```
Fila:->v3->v2
```

```
Processados:
v4 v1
```

Encontrar uma Solução para um Labirinto

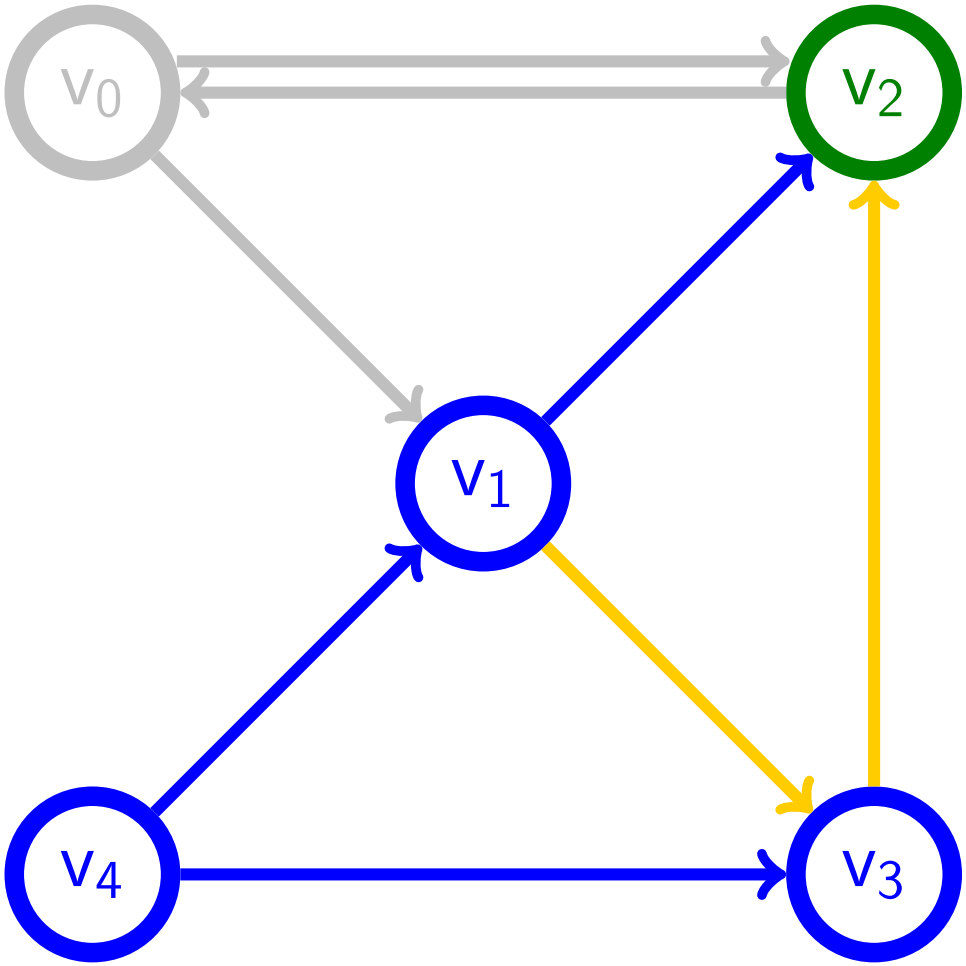


Início $v4$, destino $v0$

```
Fila:->v2
```

```
Processados:
v4 v1 v3
```

Encontrar uma Solução para um Labirinto

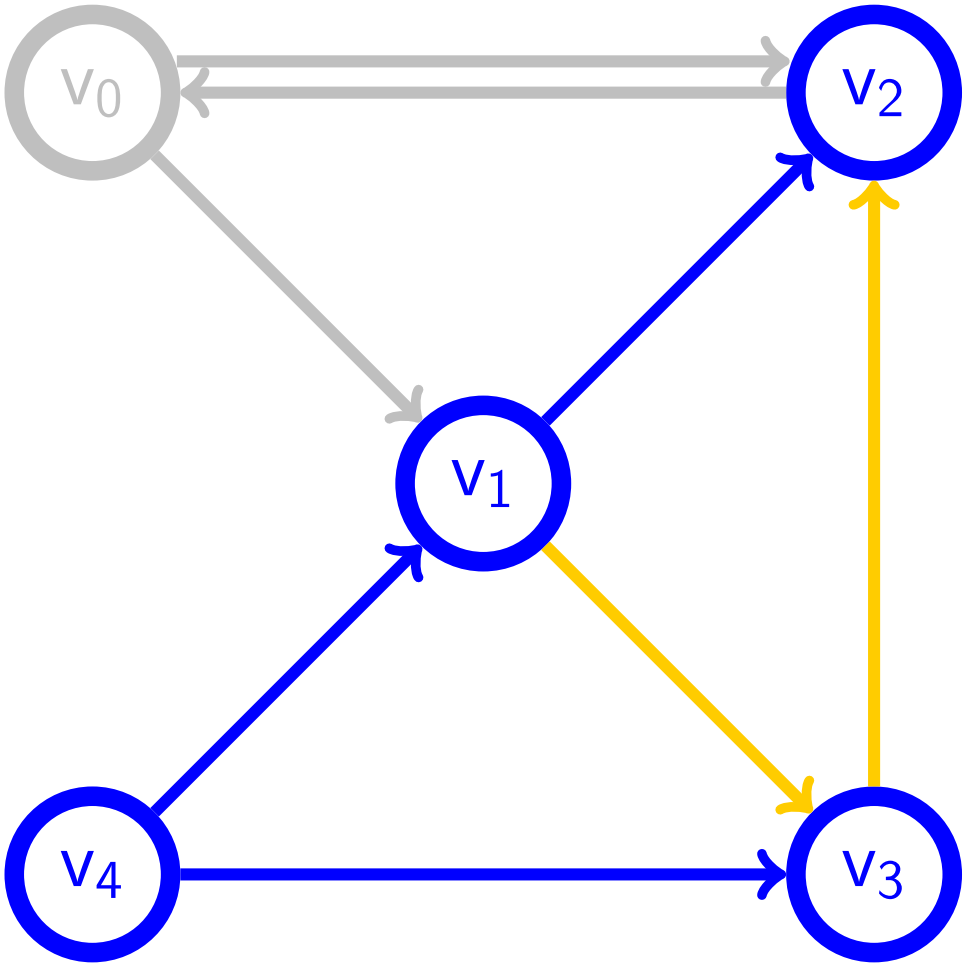


Início $v4$, destino $v0$

```
Fila:->v2
```

```
Processados:
v4 v1 v3
```

Encontrar uma Solução para um Labirinto

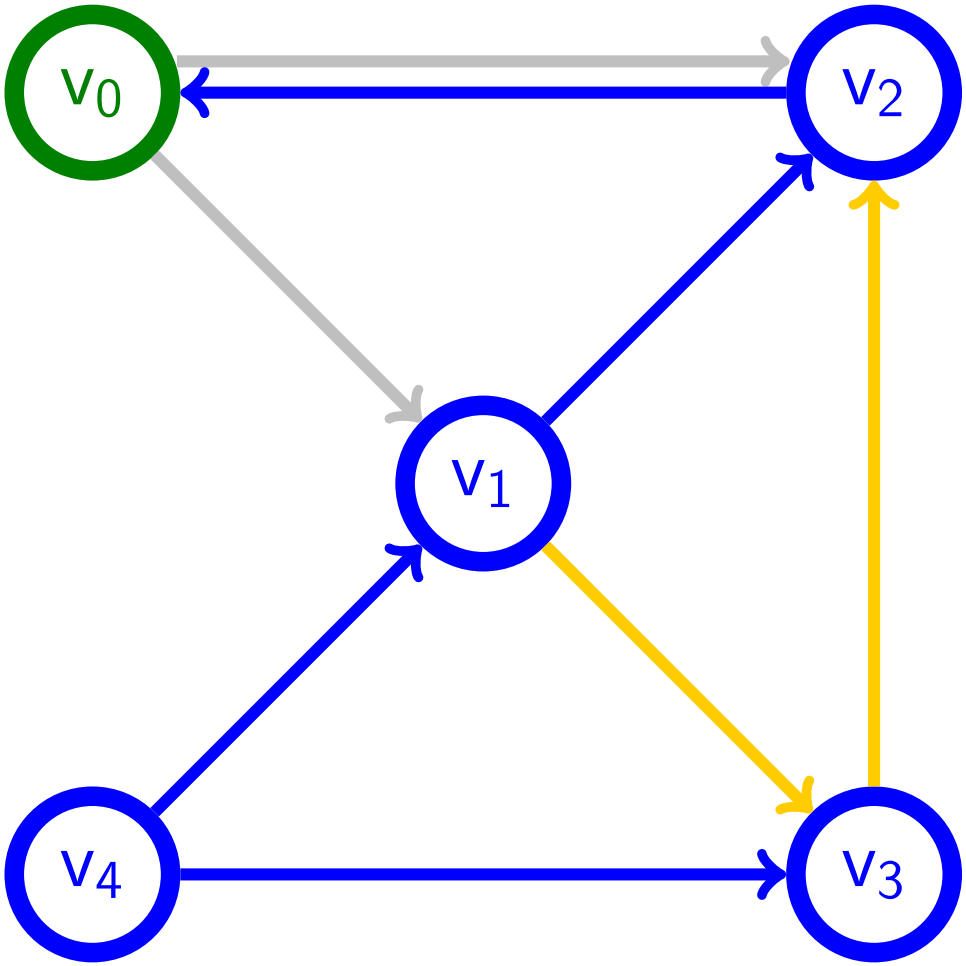


Início $v4$, destino $v0$

`Fila:`

`Processados:`
`v4 v1 v3 v2`

Encontrar uma Solução para um Labirinto

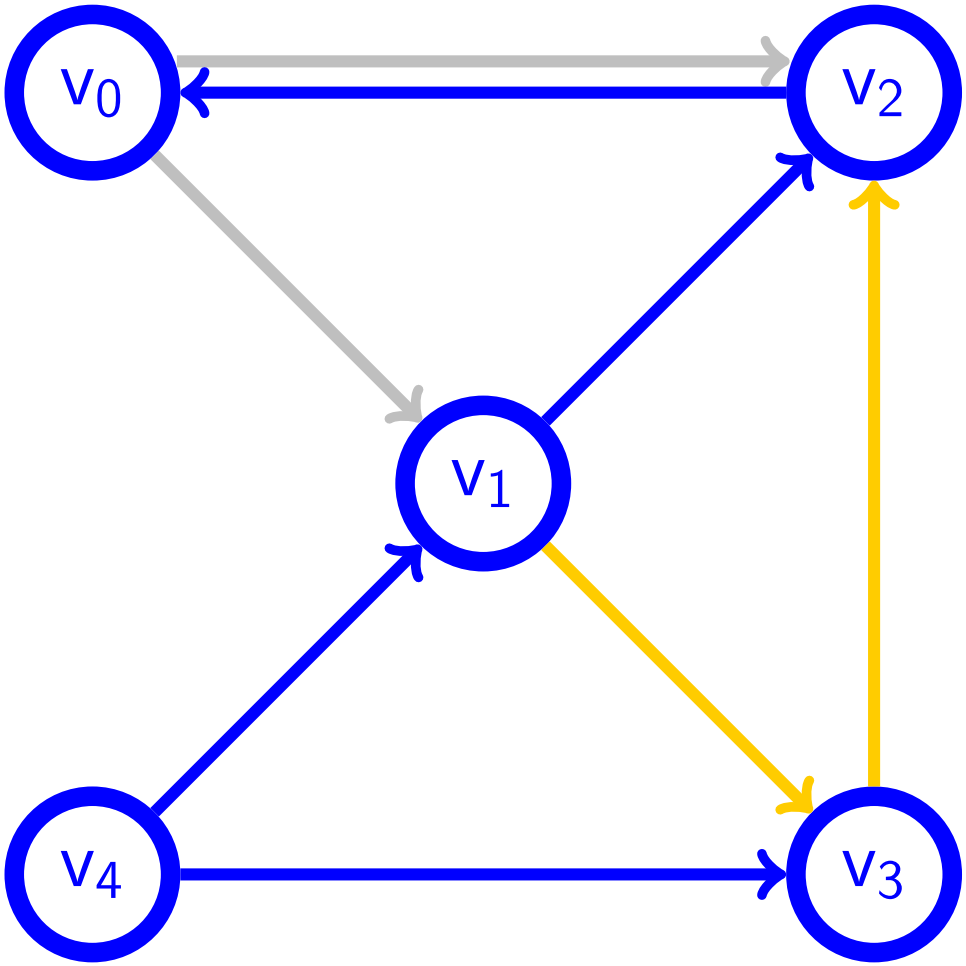


Início $v4$, destino $v0$

`Fila:->v0`

`Processados:`
`v4 v1 v3 v2`

Encontrar uma Solução para um Labirinto

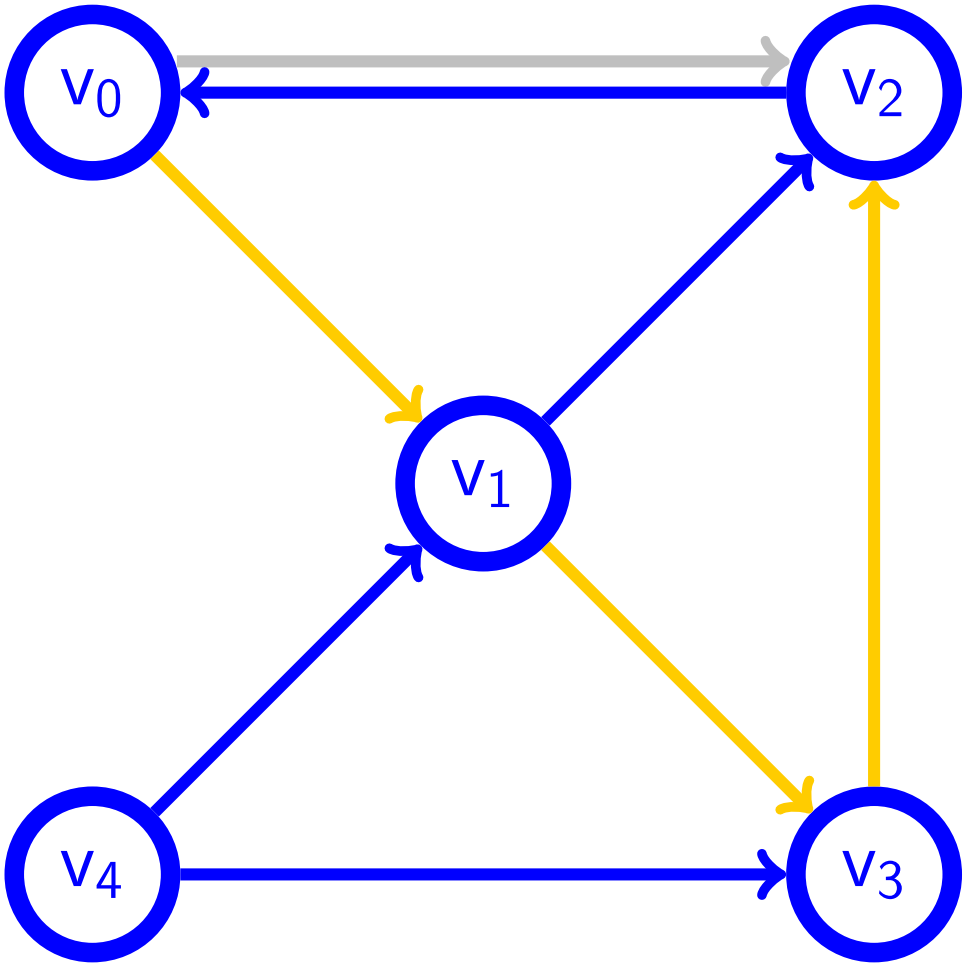


Início $v4$, destino $v0$

`Fila:`

`Processados:`
`v4 v1 v3 v2 v0`

Encontrar uma Solução para um Labirinto

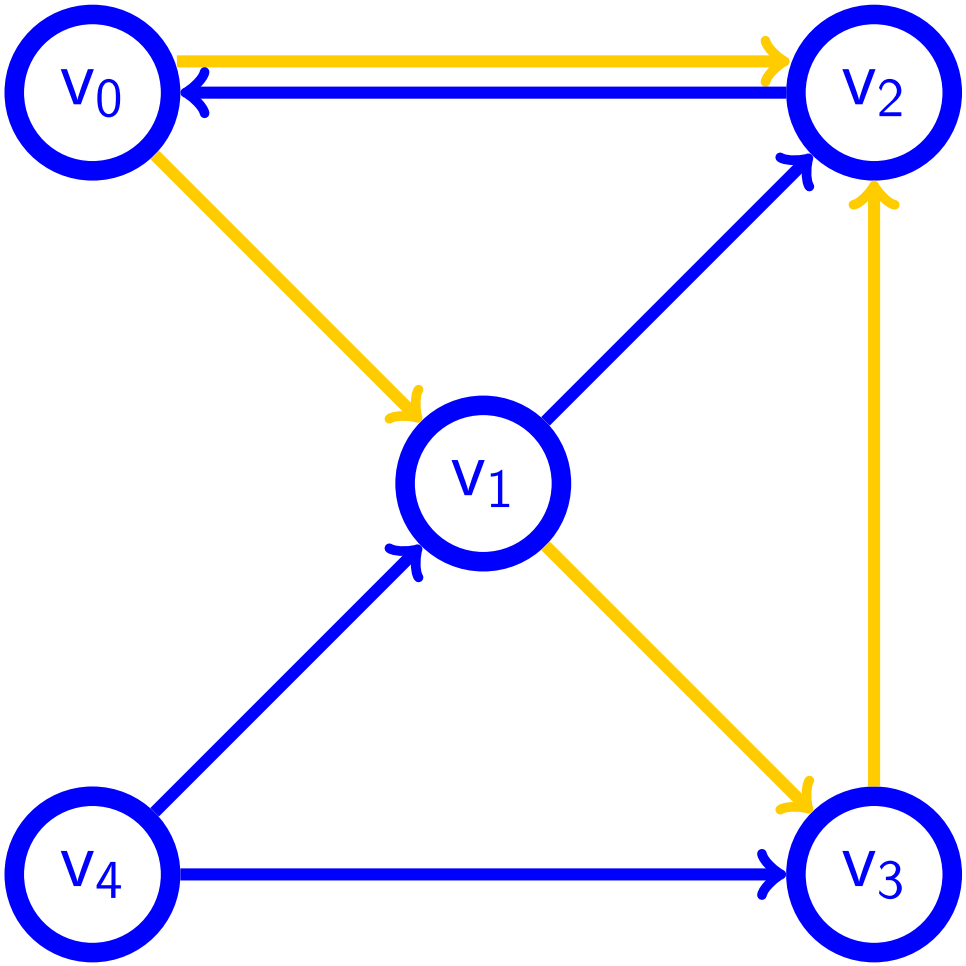


Início $v4$, destino $v0$

`Fila:`

`Processados:`
`v4 v1 v3 v2 v0`

Encontrar uma Solução para um Labirinto

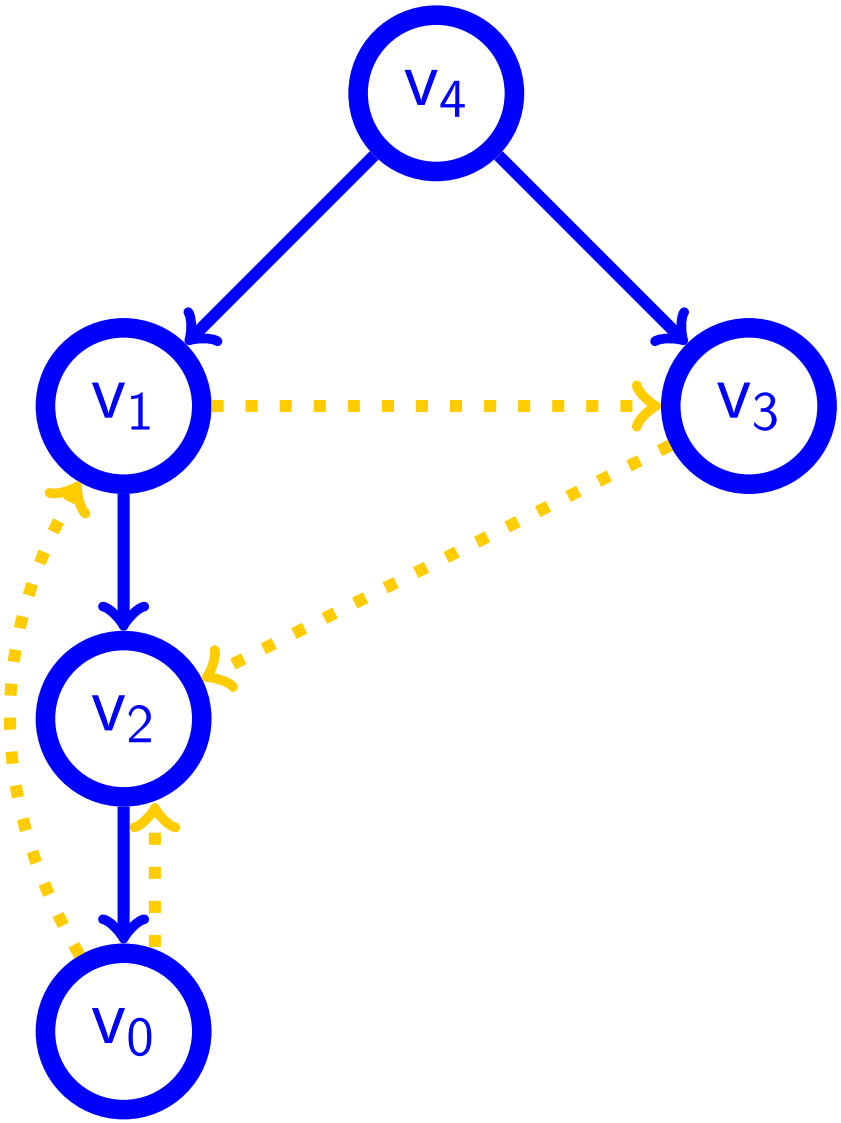


Início $v4$, destino $v0$

`Fila:`

`Processados:`
`v4 v1 v3 v2 v0`

Árvore da Busca



Resumo da Busca em Largura (iniciada em $v4$)

Nó	Ant.	Dist.
0	2	3
1	4	1
2	1	2
3	4	1
4	-1	0

Busca em Largura - Resolvendo um Labirinto

```
int BFSCaminho(Grafo* g, int origem, int destino){
  if (!g || g->numVertices<1 || origem<0 || destino<0 ||
     origem>=g->numVertices || destino>=g->numVertices) return -1;
  if (origem==destino) return 0;
  bool visitado[g->numVertices];
  int distancia[g->numVertices];
  int anterior[g->numVertices];
  int x, atual;
  for (x=0;x<g->numVertices;x++) {
    visitado[x] = false; distancia[x] = -1; anterior[x] = -1;
  }
  distancia[origem] = 0;
  visitado[origem] = true;
  Fila f ;
  inicializaFila(&f) ;
  insereFila(&f,origem) ;
  ElemLista* end;
  ...
```

Busca em Largura - Resolvendo um Labirinto

```
  while(! filaVazia(&f) ){
    atual = excluiFila(&f) ;
    end = g->A[atual];
    while (end){
      x = end->vertice;
      if (!visitado[x]){
        distancia[x] = distancia[atual]+1;
        anterior[x] = atual;
        if (x==destino) return distancia[destino];
        visitado[x] = true;
        insereFila(&f, x) ;
      }
      end = end->prox;
    }
  }
  return distancia[destino];
}
```

Busca em Largura - Resolvendo um Labirinto

```
  while(! filaVazia(&f) ){
    atual = excluiFila(&f) ;
    end = g->A[atual];
    while (end){
      x = end->vertice;
      if (!visitado[x]){
        distancia[x] = distancia[atual]+1;
        anterior[x] = atual;
        if (x==destino) return distancia[destino];
        visitado[x] = true;
        insereFila(&f, x) ;
      }
      end = end->prox;
    }
  }
  return distancia[destino];
}
```

$$O(V + E)$$

Algoritmos e Estruturas de Dados II
Aula 08 – Grafos: Busca em Largura

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