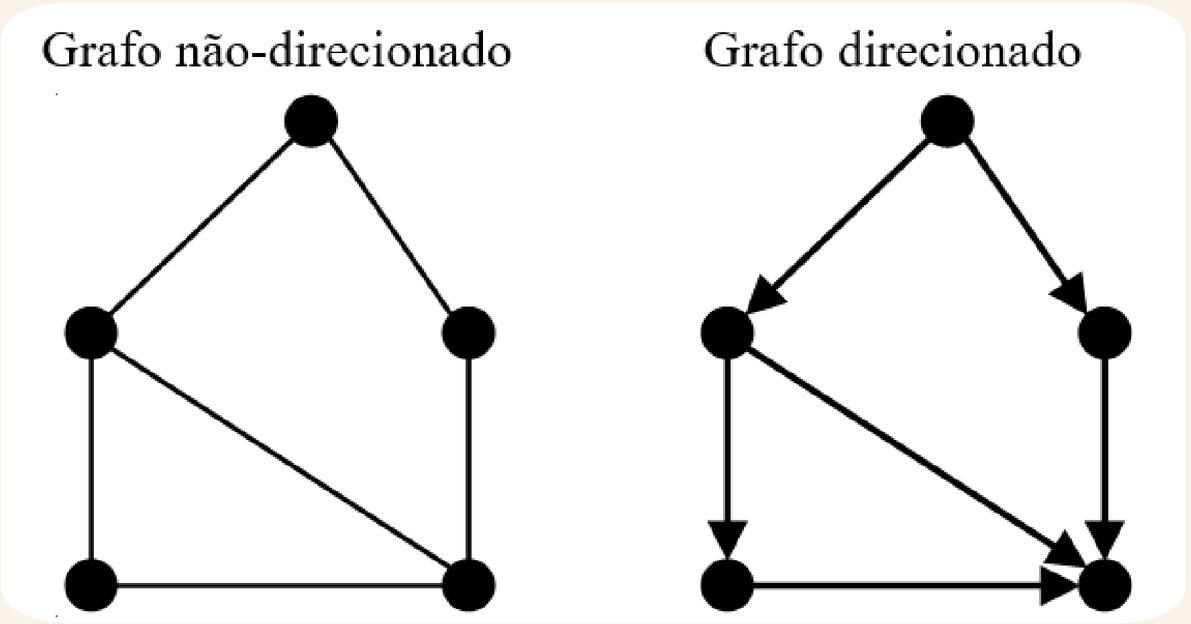
Busca em Grafos sem Pesos

LUCAS DOGO
MARIA EDUARDA PIRES
MATHEUS BARBOSA
TIAGO COSTA
RYAN RODRIGUES

AEDS III

01 - Introdução

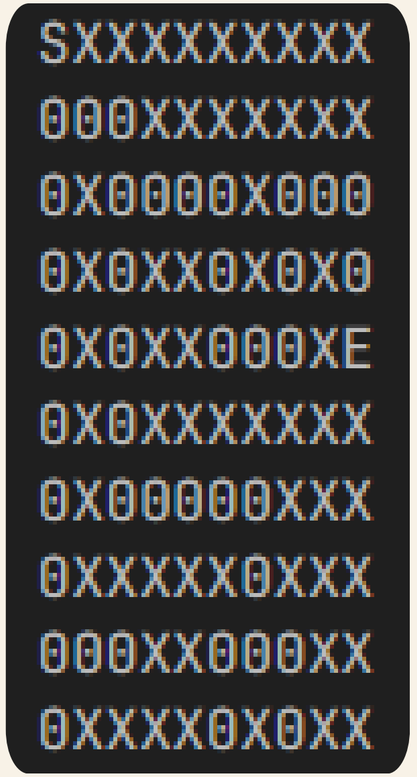
- Objetivo do trabalho: Compreender e implementar algoritmos de busca em grafos sem pesos, praticando a modelagem de grafos.
- Contextualização da Teoria dos Grafos: Importância em diversas áreas como Ciência da Computação, Biologia e Engenharia.



01 - Introdução

Labirinto

- Descrição do problema: Transformar um labirinto em um grafo para encontrar o caminho entre entrada e saída.
- Estrutura do labirinto: Representação dos elementos (E-S-X-0).



01 - Introdução

Labirinto

- A saída desejada consiste em um passo a passo com os vértices percorridos, onde o primeiro elemento representa a linha do vértice no arquivo original, e o segundo elemento representa a coluna.

```
0,0
1,0
1,1
1,2
2,2
2,1
2,0
3,0
```

02 - Estruturas

- No e Aresta: Descrição das estruturas utilizadas para representar vértices e arestas.
- Pilha: Funcionalidades e operações da estrutura de pilha implementada.

```c
typedef struct No {
    int linha;
    int coluna;
    bool visitado;
    struct Aresta* listaAdjacencia;
} No;

typedef struct Aresta {
    No* destino;
    struct Aresta* proximaAresta;
} Aresta;
```

```c
typedef struct Pilha {
    No* items[LINHAS * COLUNAS];
    int topo;
} Pilha;
```

02 - Estruturas

- Fila: Funcionalidades e operações da estrutura de fila implementada.
- Grafo: Manipulação de vértices para criação e uso do grafo.

```c
typedef struct Fila {
    struct {
        No* filho;
        No* pai; // Adicionamos um ponteiro para o nó pai
    } items[LINHAS * COLUNAS];
    int frente;
    int fim;
} Fila;
```

```c
void adicionarAresta(Aresta** listaAdjacencia, No* destino);
void criarGrafo(char labirinto[LINHAS][COLUNAS], No* nos[LINHAS][COLUNAS]);
```

02 - Estruturas

Criar Grafo

```c
void criarGrafo(char labirinto[LINHAS][COLUNAS], No* nos[LINHAS][COLUNAS]) {
    for (int i = 0; i < LINHAS; i++) {
        for (int j = 0; j < COLUNAS; j++) {
            if (labirinto[i][j] != PAREDE) {
                No* atual = nos[i][j];
                atual->linha = i;
                atual->coluna = j;
                atual->visitado = false;
                atual->listaAdjacencia = NULL;

                // Adicionar arestas para células adjacentes válidas
                if (i > 0 && labirinto[i - 1][j] != PAREDE) {
                    adicionarAresta(&(atual->listaAdjacencia), nos[i - 1][j]);
                }
                if (i < LINHAS - 1 && labirinto[i + 1][j] != PAREDE) {
                    adicionarAresta(&(atual->listaAdjacencia), nos[i + 1][j]);
                }
                if (j > 0 && labirinto[i][j - 1] != PAREDE) {
                    adicionarAresta(&(atual->listaAdjacencia), nos[i][j - 1]);
                }
                if (j < COLUNAS - 1 && labirinto[i][j + 1] != PAREDE) {
                    adicionarAresta(&(atual->listaAdjacencia), nos[i][j + 1]);
                }
            }
        }
    }
}
```

03 - Algoritmos

- Foram empregados os algoritmos clássicos de busca em grafos: Busca Por Profundidade e Busca Por Largura.

- Ambos os algoritmos exploram o grafo de maneira sistemática, mas com diferentes estratégias, proporcionando diferentes resultados e desempenhos em determinados contextos.

03 - Algoritmos

Busca por Profundidade

```c
void buscaPorProfundidade(No *nos[LINHAS][COLUNAS], No *entrada, No *saida, Pilha *pilha)
{

    if (topoDaPilha(pilha) == saida)
    {
        return;
    }
    else
    {
        No *atual = topoDaPilha(pilha);
        Aresta *listaAdjacencia = atual->listaAdjacencia;
        while (listaAdjacencia != NULL)
        {
            if (listaAdjacencia->destino->visitado == 0)
            {
                listaAdjacencia->destino->visitado = 1;
                empilhar(pilha, listaAdjacencia->destino);
                buscaPorProfundidade(nos, entrada, saida, pilha);
            }
            listaAdjacencia = listaAdjacencia->proximaAresta;
        }
        if (topoDaPilha(pilha) != saida)
        {
            desempilhar(pilha);
        }
    }
}
```

03 - Algoritmos

Busca por Largura

```c
No *buscaPorLargura(No *nos[LINHAS][COLUNAS], No entrada, No saida, Fila *fila)
{
    while (!filaVazia(fila))
    {
        No *atual = desenfileirar(fila); // Obtemos o próximo nó da fila

        // Verifica se o nó atual é a saída
        if (atual->linha == saida.linha && atual->coluna == saida.coluna)
        {
            return atual; // Achou o nó de saída
        }

        // Marca o nó atual como visitado
        atual->visitado = true;

        // Itera sobre os nós adjacentes
        Aresta *listaAdjacencia = atual->listaAdjacencia;
        while (listaAdjacencia != NULL)
        {
            No *adjacente = listaAdjacencia->destino;
            // Verifica se o nó adjacente não foi visitado e não é uma parede
            if (!adjacente->visitado)
            {
                // Marca o nó adjacente como visitado
                adjacente->visitado = true;
                // Enfileira o nó adjacente e seu nó pai
                enfileirar(fila, adjacente, atual);
            }
            listaAdjacencia = listaAdjacencia->proximaAresta;
        }
    }
    return NULL; // Não encontrou o caminho até a saída
}
```

04 - Complexidade

Busca por Profundidade

- Tempo: O tempo de execução é O(V + E), onde V é o número de vértices (nós) e E é o número de arestas no grafo. Isso ocorre porque, em cada iteração do loop while, todas as arestas de um nó são examinadas.

- Espaço: A complexidade de espaço é O(V), onde V é o número de vértices no grafo. Isso se deve ao uso da pilha de execução, que armazena os nós visitados durante a busca. No pior caso, a pilha pode conter todos os vértices do grafo.

04 - Complexidade

Busca por Largura

- Tempo: O tempo de execução é O(V + E), onde V é o número de vértices (nós) e E é o número de arestas no grafo. No pior caso, onde todos os nós do grafo são visitados, cada nó e cada aresta são examinados uma vez.

- Espaço: A complexidade de espaço é O(V), onde V é o número de vértices no grafo. Isso ocorre devido ao uso da fila de execução, que armazena os nós visitados durante a busca. No pior caso, a fila pode conter todos os vértices do grafo, resultando em uma complexidade de espaço proporcional ao número de vértices.

05 - Conclusão

- O trabalho demonstrou com sucesso a implementação e aplicação dos algoritmos de busca em grafos para resolver o desafio de encontrar caminhos em labirintos. Os algoritmos de busca por profundidade e por largura foram eficazes, cada um com suas estratégias distintas de exploração do espaço de busca, proporcionando resultados satisfatórios na resolução dos labirintos fornecidos.

- Após analisar a complexidade de ambos os algoritmos de busca implementados foi possível notar que apresentam a mesma complexidade tanto para tempo quanto para espaço.

06 - Referências

- **Carvalho, Iago.** Slides de Aula sobre Busca em Profundidade e Busca em Largura. Disponível em:
- https://github.com/iagoac/dce529/blob/main/slides/aula_13.pdf. Acesso em: 27 mar 2024.
- **Fischer, Ricardo.** "Causalidade e Teoria dos Grafos: Uma Breve Reflexão." LinkedIn, Disponível em: https://www.linkedin.com/pulse/causalidade-e-teoria-dos-grafos-uma-brev%C3%ADssima-ricardo-fischer/?originalSubdomain=pt. Acesso em: 31 mar. 2024.
- **Tab News.** "Uma Breve Introdução à Teoria dos Grafos." Disponível em: https://www.tabnews.com.br/UlisssesRosa/uma-breve-introducao-a-teoria-dos-grafos. Acesso em: 31 mar 2024.

OBRIGADO!

LUCAS DOGO
MARIA EDUARDA PIRES
MATHEUS BARBOSA
TIAGO COSTA
RYAN RODRIGUES