Conteúdos necessário para o pdf
Barbara Marques

1. Introdução:(especificar sobre o problema do labirinto)

Neste capítulo, será apresentado o problema do labirinto, definindo e especificando qual é o objetivo do trabalho.

Aqui você pode digitar um texto e adicionar uma imagem de um dos labirintos para facilitar a explicação do problema.

Exemplo de labirinto 10x10:
EXXXXXXXXX
000XXXXXXX
0X0000X000
0X0XX0X0X0
0X0XX000XS
0X0XXXXXXX
0X00000XXX
0XXXXX0XXX
000XX000XX
0XXXX0X0XX

Aqui está um exemplo de um dos labirintos para facilitar a explicação do problema.

2.Estruturas de dados:(Especificar todos as estruturas utilizadas em cada código)
Estruturas principais utilizado no primeiro código:

- **Fila first-in first-out**
- **Grafo**
- **Estrutura de nó para a implementação de uma lista encadeada usada para implementar uma fila(alocação dinâmica)**

Existem mais estruturas no primeiro código para ser analisado pelo gerenciador desse pdf

Estruturas principais utilizado no segundo código:

- **Pilha**
- **Grafo**
- **Estrutura de nó para a implementação de uma lista encadeada para implementar uma pilha(alocação dinâmica)**

3.Algoritmos utilizados e sua complexidade
Complexidade de BFS:

O código fornecido implementa um algoritmo de busca em largura (BFS) para encontrar o caminho em um labirinto representado como um grafo. Vou analisar a complexidade do algoritmo:

1. Verificação de vértices (verificação vértice):Este processo envolve percorrer o labirinto uma vez para encontrar os vértices e construir a lista de vértices. Supondo que o labirinto tenha (n) células, isso leva (O(n)) tempo.

2. Construção da matriz de adjacência (construir matriz de adjacência): Isso envolve percorrer todos os pares de vértices e verificar se eles são adjacentes. Como há (n) vértices, a complexidade é (O(n^2)).**

3. Busca em Largura (BFS Busca largura): A busca em largura percorre todos os vértices e arestas do grafo. Em um grafo com (V) vértices e (E) arestas, a complexidade do BFS é (O(V + E)). No caso deste labirinto, onde (V) é o número de células no labirinto, a complexidade é (O(n)).**

Portanto, a complexidade total do algoritmo é a soma dessas etapas, que é dominada pela complexidade do passo de construção da matriz de adjacência:

[O(n) + O(n^2) + O(n) = O(n^2)]

Então, a complexidade do algoritmo fornecido é (O(n^2)), onde (n) é o número de células no labirinto.

Complexidade DFS:

A complexidade do algoritmo apresentado pode ser analisada em relação às suas principais operações:

1. Leitura do arquivo e criação da matriz de adjacência (verificação vértice e criar lista Vértice):

- **A leitura do arquivo é feita uma vez, percorrendo todas as células do labirinto, que podem ser representadas por (n) células, onde (n) é o número total de células no labirinto.**
- **A criação da lista de vértices e a construção da matriz de adjacência são operações que dependem do número de células no labirinto, portanto, essas operações têm complexidade (O(n)).**

2. Busca em profundidade (profundidade e visita):

- **A busca em profundidade percorre todos os vértices e arestas do grafo.**
- **Se (V) é o número de vértices e (E) é o número de arestas do grafo, a complexidade da busca em profundidade é (O(V + E)).**
- **No caso deste labirinto, onde (V) é o número de células, e como estamos considerando apenas células adjacentes conectadas, a complexidade é (O(n)), onde (n) é o número total de células no labirinto.**

Portanto, a complexidade total do algoritmo é dominada pela complexidade da busca em profundidade, que é (O(n)), onde (n) é o número total de células no labirinto.

Para esse Trabalho utilizamos algoritmos de Busca de Grafos

1. **Explicar o'que é Busca de Grafos**
2. **descrever os dois algoritmos utilizados em cada código sendo eles: Busca em largura(primeiro código) e Busca em profundidade(segundo código)**

Tem material do professor Iago sobre esses dois tipos de algoritmos para ajudar na explicação do pdf https://github.com/iagoac/dce529/blob/main/slides/aula_13.pdf
link acima é a aula sobre Busca de Grafos.

Diogo moreira e Gustavo marcelino - Conteúdos para os slides

1.Primeiro Slide

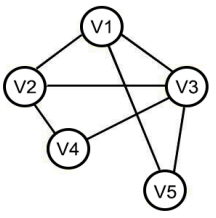
Ter os nomes de todos os participantes do trabalho com o nome completo e o tema(Busca de Grafos)
Abner Gomes Guimarães
Diogo da Silva Moreira
Gustavo Marcelino Izidoro
Marcelo Bernardino da Silva Júnior
Barbara Marques
Felipe Araújo Correia

Nome do Docente: Iago Augusto de Carvalho

2.Segundo Slide

Explicação sobre matriz de adjacência utilizando imagem para não ler muito texto durante a apresentação
Exemplo de imagem:

Matriz de Adjacência



	V1	V2	V3	V4	V5
V1	0	1	1	0	1
V2	1	0	1	1	0
V3	1	1	0	1	1
V4	0	1	1	0	0
V5	1	0	1	0	0

Imagem: Paulo Martins

O vértice V2 está conectado em v1,v3 e v4

Isso é representado na linha do vértice 2 com o número 1 em cada vértice correspondente

3.Terceiro Slide
Como representamos a matriz adjacente em código

Primeiro criamos uma matriz de vértices para representar os vértices existentes lendo o labirinto e transformando E, S e 0 como vértices em número 1(caminhos a serem percorridos)

```c
void verificacao_vertice(Grafo *grafo, FILE *arq) {
  char caractere;
  int linha = 0, coluna = 0;
  grafo->num_vertices = 0;
  Coordenada coordenada_inicio;
  Coordenada coordenada_fim;

  while ((caractere = fgetc(arq)) != EOF) {
    if (caractere == 'E' || caractere == 'S' || caractere == '0') {
      grafo->matriz_ver[linha][coluna] = 1;
      coluna++;
      if (caractere == 'E') {
        grafo->coordenada_inicio.linha = linha;
        grafo->coordenada_inicio.coluna = coluna;
      }
      if (caractere == 'S') {
        grafo->coordenada_fim.linha = linha;
        grafo->coordenada_fim.coluna = coluna;
      }
      grafo->num_vertices++;
    } else if (caractere == 'X') {
      grafo->matriz_ver[linha][coluna] = 0;
      coluna++;

    } else {
      linha++;
      coluna = 0;
    }
  }
}
```

E também pegamos as coordenadas do E e S para ser utilizado no método BFS e DFS

4.Quarto slide(pode ser colocado junto com o terceiro slide caso o gerenciador preferir)

Com as informações dos vértices existentes, precisamos criar uma matriz adjacente que vai ser utilizada para representar as conexões entre os vértices para representar o labirinto. Primeiro iniciamos a matriz de adjacência com nenhuma conexão(ou seja 0)

```c
void inicializar_matriz_adjacencia(Grafo *grafo, int num_vertices) {
  grafo->num_vertices = num_vertices;

  for (int i = 0; i < MAX_vet; i++) {
    for (int j = 0; j < MAX_vet; j++) {
      grafo->matriz_adj[i][j] = 0;
    }
  }
}
```

5.Quinto Slide

Construindo a matriz adjacência com as conexões utilizando a matriz de vértices

```c
void construir_matriz_adjacencia(int num_vertices, Coordenada *listaVertice) {

  int matriz_adj[num_vertices][num_vertices];
  for (int i = 0; i < num_vertices; i++) {
    for (int j = 0; j < num_vertices; j++) {

      if (&listaVertice[i] == &listaVertice[j]) {
        matriz_adj[i][j] = 0;
      } else {
        // verifica se conectado em cima
        if (listaVertice[j].linha - 1 == listaVertice[i].linha &&
            listaVertice[j].coluna == listaVertice[i].coluna) {
          matriz_adj[i][j] = 1;
        }
        // verifica se conectado em baixo
        else if (listaVertice[j].linha + 1 == listaVertice[i].linha &&
                 listaVertice[j].coluna == listaVertice[i].coluna) {
          matriz_adj[i][j] = 1;
        }
        // verifica se conectado na direita
        else if (listaVertice[j].coluna + 1 == listaVertice[i].coluna &&
                 listaVertice[j].linha == listaVertice[i].linha) {
          matriz_adj[i][j] = 1;

        }
        // verifica se conectado na esquerda
        else if (listaVertice[j].coluna - 1 == listaVertice[i].coluna &&
                 listaVertice[j].linha == listaVertice[i].linha) {
          matriz_adj[i][j] = 1;
        } else {
          matriz_adj[i][j] = 0;
        }
      }
```

Próximos slides:

Os próximos slides vai ser utilizado para explicar os códigos de busca, como por exemplo BFS(Busca em largura) e DFS(Busca de profundidade)
e apresentar as complexidades de cada algoritmo

BFS:

```c
bool visitaL(Grafo *grafo, int s, bool *explorados,int *distancia, Fila *fila) {
    // Marca o vértice inicial como explorado e o adiciona à fila
    explorados[s] = true;
    distancia[s] = 0;
    inserir_V_fila(fila, s);

    while (fila->inicio != NULL) {
        // Remove o primeiro vértice da fila
        int u = remover_V_fila(fila);

        // Percorre todos os vértices adjacentes a u
        for (int v = 0; v < grafo->num_vertices; v++) {
            // Verifica se v é adjacente a u e se ainda não foi exploradov
            if (grafo->matriz_adj[u][v] && !explorados[v]) {
                // Marca v como explorado e o adiciona à fila
                explorados[v] = true;
                distancia[v] = distancia[u] + 1;
                inserir_V_fila(fila, v);
            }
        }
    }

     return true;
}
```

```c
int BFS_Busca_largura(Grafo *grafo) {
    bool explorados[grafo->num_vertices];
    int distancia[grafo->num_vertices];
    Fila fila;
    iniciar_fila(&fila);

    // Inicializa todos os vértices como não explorados
    for (int u = 0; u < grafo->num_vertices; u++) {
        explorados[u] = false;
        distancia[u] = -1;
    }

    // Começa a busca em largura a partir do vértice de origem
    int origem = grafo->coordenada_inicio.linha;
    visitaL(grafo, origem, explorados,distancia, &fila);

    for (int i = 0; i < grafo->num_vertices; i++) {
        if(distancia[i] >= distancia[i-1] || i == 0)
            printf("%d,%d\n", grafo->listaVertice[i].coluna, MAX_vet - 1 - grafo-
>listaVertice[i].linha);
    }
}
```

DFS:

```
bool visitaP(Grafo *grafo, int u, int cor[], No *caminho) {
    cor[u] = CINZA;


    No *vertice = malloc(sizeof(No));
    vertice->vertice_linha = grafo->listaVertice[u].linha;
    vertice->vertice_coluna = grafo->listaVertice[u].coluna;
    inserir_pilha(&caminho, vertice);

    if(grafo->listaVertice[u].linha == grafo->coordenada_fim.linha && grafo-
>listaVertice[u].coluna == grafo->coordenada_fim.coluna){
      imprimir(caminho);
      return true;
    }

    for (int v = 0; v < grafo->num_vertices; v++) {
        if (grafo->matriz_adj[u][v] == 1 && cor[v] == BRANCO) {
            visitaP(grafo, v, cor, caminho);
        }
    }

    cor[u] = PRETO;
```

```
void profundidade(Grafo *grafo){
  int num_vertices = grafo->num_vertices;
  int cor[num_vertices];
  No *caminho = NULL;
  int u;
  for(u = 0; u < num_vertices; u++){
    cor[u] = BRANCO;
  }

  u = grafo->coordenada_inicio.linha;
      visitaP(grafo, u, cor, caminho);
}
```

Data provável de finalização dos códigos 31/03/2024 até 00:00

Avisos importantes: esse pdf é para ser utilizado apenas como guia para ajudar os gerenciadores da apresentação e criador do pdf original.

- **Pesquisar cada método de busca para estar informado sobre o projeto e utilizar nas apresentações.**
- **Utilizar os slides pré pronto mas mudar os códigos para que fique bom para cada imagem especifica(não apenas mudar a imagem do código LATEX)**
- **Não colocar muito texto**
- **Utilizar mais imagem para representar cada tópico**
- **Confirmar as estruturas de dados com cada programador do algoritmo**

Método BFS e DFS: Abner Gomes Guimarães, Felipe Correia e Marcelo Bernardino

Alguma reclamação ou sugestão de mudança? Vamos discutir isso com todos do grupo para chegarmos a uma solução que atenda a todos!

Agradeço e espero que esse guia ajude na criação dos slides e pdf.