# IMPLEMENTAÇÃO DE DIFERENTES ALGORITMOS DE BUSCA EM GRAFOS SEM PESOS

AEDs III

CAIO FERNANDO DIAS

FELIPE DE GODOI CORRÊA

MATHEUS REIS DE LIMA



#### PROBLEMA DO LABIRINTO

- Labirinto 10x10:
  - o "E" simboliza a entrada.
  - "S" simboliza a saída.
  - "X" simboliza as paredes.
  - "0" simboliza o caminho disponível.

EXXXX	0,4
LAAAA	0,3
000XX	1,3
OXOOS	2,3
XOOXX	2,2
	3,2
XXXXX	4,2

Figura 1: Labirinto 5x5

Figura 2: Saída do labirinto



#### ESTRUTURAS DE DADOS

- Fila:
  - FIFO (First-In-First-Out).
  - Gerenciamento de tarefas em SO.
  - Dinâmica.
  - Ponteiros:
    - Dois ponteiros: início e fim.

```
void FFVazia(Fila *f) {
    f->prim = (BlockB*) malloc (sizeof(BlockB));
    f->ult = f->prim;
    f->prim->prox = NULL;
void Enfileira(Fila *f, ItemB d) {
   f->ult->prox = (BlockB*) malloc (sizeof(BlockB));
    f->ult = f->ult->prox;
   f->ult->data = d;
    f->ult->prox = NULL;
void Desenfileira(Fila *f, ItemB *d) {
   if(f->prim == f->ult | f == NULL | f->prim->prox == NULL) {
        cout << "Erro: Fila vazia!" << endl;</pre>
        return;
   BlockB *aux = f->prim->prox;
    f->prim->prox = aux->prox;
    if (f->prim->prox == NULL) {
       f->ult = f->prim;
    *d = aux->data;
    free(aux);
bool VerificaFilaVazia(Fila *f) {
    return (f->prim == f->ult || f == NULL || f->prim->prox == NULL);
}
```

Figura 3: Implementação da Fila no algoritmo BFS



#### ESTRUTURAS DE DADOS

- Pilha:
  - LIFO (Last-In-First-Out).
  - Botão "voltar" do navegador.
  - Dinâmica.
  - Ponteiros:
    - Um ponteiro: topo da pilha.

```
void FPVazia(Pilha *p){
    p->base = new Block;
    p->top = p->base;
    p->base->prox = nullptr;
void Push(Pilha *p, Posicao d){
    Block *aux = new Block;
    aux->data = d;
    aux->prox = p->top;
    p->top = aux;
void Pop(Pilha *p, Posicao *d){
    Block *aux;
    if(p->base == p->top || p == nullptr){
        cout << "PILHA VAZIA!\n";
        return;
    aux = p->top;
    p->top = aux->prox;
    *d = aux->data;
    delete aux;
bool PilhaVazia(Pilha *p){
    return (p->top == p->base);
```

Figura 4: Implementação da Pilha no algoritmo DFS



### BFS (breadth-first search)

- Começa no nó raiz e visita todos vizinhos antes de ir pro próximo nível.
- Fila para manter a visitação.
- Complexidade O(V+E)

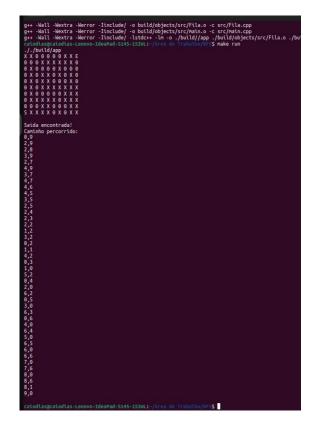


Figura 5: Saída BFS do labirinto 10x10



### BFS (breadth-first search)

```
// Função para realizar a busca em largura no labirinto
void BFS(Labirinto *lab) {
    Fila fila:
   FFVazia(&fila);
    int inicio_i, inicio_j;
    EncontrarEntrada(lab, &inicio_i, &inicio_j);
    ItemB inicio;
    inicio.val = inicio_i * MAXTAM + inicio_j;
   Enfileira(&fila, inicio);
   int direcoes[4][2] = \{\{-1, 0\}, \{0, -1\}, \{0, 1\}, \{1, 0\}\}; // \text{ cima, esquerda, direita, baixo}
   // Vetor para armazenar o caminho percorrido
   vector<pair<int, int>> caminho;
    while (!VerificaFilaVazia(&fila)) {
        ItemB atual;
        Desenfileira(&fila, &atual);
        inicio_i = atual.val / MAXTAM;
        inicio_j = atual.val % MAXTAM;
        for (int d = 0; d < 4; d++) {
           int prox_i = inicio_i + direcoes[d][0];
           int prox_j = inicio_j + direcoes[d][1];
           if (prox_i >= 0 && prox_i < lab->tam && prox_j >= 0 && prox_j < lab->tam &&
                lab->matriz[prox i][prox j] != 'X') {
                if (lab->matriz[prox_i][prox_j] == 'S') {
                    // Encontrou a saída, armazene o caminho e retorne
                    caminho.push back({prox i, prox j});
                    cout << "Saida encontrada!" << endl;
                    // Imprime o caminho encontrado
                    cout << "Caminho percorrido:" << endl;
                    for (const auto& p : caminho) {
                       cout << p.first << "," << p.second << endl;
                    return;
                lab->matriz[prox_i][prox_j] = 'X'; // Marcar como visitado
                ItemB prox;
                prox.val = prox_i * MAXTAM + prox_j;
                Enfileira(&fila, prox);
                caminho.push_back({prox_i, prox_j}); // Armazena o caminho
    cout << "Saída não encontrada!" << endl;
```

Figura 6: Função BFS



### DFS (depth-first search)

- Começa no nó raiz e vai o mais fundo possível, antes de retroceder.
- Pilha para manter a visitação.
- Complexidade O(V+E)

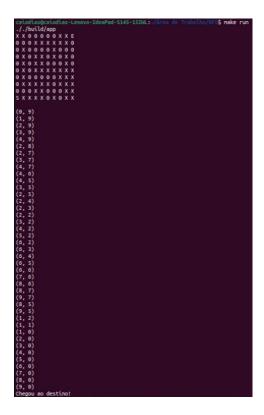


Figura 7: Saída DFS do labirinto 10x10



#### DFS (breadth-first search)

```
bool DFS(Labirinto *lab, int x, int y, bool visited[][MAXTAM], Pilha *p){
    static int row[] = \{-1, 0, 0, 1\};
    static int col[] = \{0, -1, 1, 0\};
    Push(p, {x, y});
   visited[x][y] = true;
   while (!PilhaVazia(p)) {
       Posicao curr;
       Pop(p, &curr);
       cout << "(" << curr.x << ", " << curr.y << ")" << endl; // Processa o vértice visitado
       if (lab->matriz[curr.x][curr.y] == 'S') {
           return true; // Chegou ao destino
       for (int i = 0; i < 4; ++i) {
           int newRow = curr.x + row[i];
           int newCol = curr.y + col[i];
            if (newRow >= 0 && newRow < lab->tam && newCol >= 0 && newCol < lab->tam &&
                lab->matriz[newRow][newCol] != 'X' && !visited[newRow][newCol]) {
                Push(p, {newRow, newCol});
               visited[newRow][newCol] = true;
    return false; // Não encontrou o destino
```

Figura 8: Função DFS



## REFERÊNCIAS

https://www.ime.usp.br/~pf/algoritmos/aulas/fila.html

https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html

https://www.ime.usp.br/~pf/algoritmos\_para\_grafos/aulas/bfs.html

https://www.ime.usp.br/~pf/algoritmos\_para\_grafos/aulas/dfs.html



## PERGUNTAS?

**OBRIGADO!** 

CAIO FERNANDO DIAS FELIPE DE GODOI CORRÊA MATHEUS REIS DE LIMA

