MINISTÉRIO DA EDUCAÇÃO
Universidade Federal de Alfenas UNIFAL-MG
Rua Gabriel Monteiro da Silva, 700 – Centro. Alfenas/MG. CEP: 37130-001

Trabalho prático 1- Saída do labirinto.

Docente: Iago Augusto de Carvalho

Discente: Jeann Victor Batista **R.A: 2024.1.08.014**

Discente: Filipe Bessa Carvalho **R.A: 2024.1.08.009**

Discente: Nicolas Rodrigues Teixeira de Oliveira **R.A: 2023.1.08.047**

Introdução

O objetivo do trabalho proposto pelo professor Iago Augusto é a implementação de um algoritmo capaz de resolver um labirinto de tamanho 10x10. O labirinto possui as seguintes propriedades: a letra “E” representa a entrada, o “S” representa a saída e o “X” seria onde não é possível de se locomover dentro do labirinto, já o numero “0” é onde é possível andar dentro do labirinto.

Imagem Ilustrativa:

```
EXXXX
000XX
0X00S
X00XX
X0XXX
```

Estrutura de Dados

Para a resolução do problema, foi utilizada a estrutura de dados conhecida como pilha. Pilhas são estruturas de dados que armazenam os elementos em um formato sequencial, empilhando um item acima do outro, possui as seguintes características: “o último a entrar, será o primeiro a sair” (LIFO), operações de remoção quanto a de inserção, são realizadas apenas no topo.

Imagem Ilustrativa:

No algoritmo de resolução do trabalho foram utilizadas duas pilhas.

A pilha "caminho" armazena o trajeto percorrido e, ao final da execução, contém em ordem inversa todas as posições que descrevem o percurso até a saída. A pilha "bifurcação" funciona como um registro temporário dos possíveis caminhos encontrados ao longo do labirinto. Quando há apenas um caminho viável, ele é removido da pilha de bifurcação e adicionado à pilha "caminho". No caso de duas opções de trajeto, o segundo caminho permanece na pilha, enquanto a posição da bifurcação é registrada.

Dessa forma, a pilha de bifurcações age como uma lista de "checkpoints", onde o topo sempre representa a última bifurcação, facilitando a retomada do percurso quando não há mais posições a explorar. Nesses momentos, o código remove as posições da pilha "caminho" até que o topo coincida com o topo da pilha "bifurcação", permitindo que o trajeto retorne à última bifurcação. Em seguida, a posição seguinte a ser explorada é a próxima na pilha de bifurcações, correspondente ao segundo caminho possível.

O código é projetado para retornar à última bifurcação somente quando não restam mais posições a serem percorridas no caminho atual. Assim, ele explora um trajeto até chegar à saída ou identificar que não é o caminho correto. No primeiro caso, o código encerra a execução responsável pela resolução do labirinto, enquanto no segundo, retorna à última bifurcação para continuar a exploração.

Algoritmos

Funções

A função **“*printlab*”** imprime o conteúdo de um labirinto representado por uma matriz de caracteres de 10x10.A função “***printpilha***” imprime os elementos de uma pilha de coordenadas armazenadas como uma lista encadeada, começando pelo topo. A função “***push”*** adiciona um novo nó à pilha de coordenadas, atualizando o topo da pilha. Nesta função, são passadas como parâmetro as coordenadas X e Y e o novo “no” é alocado na pilha. A função ***“pop”*** remove o topo da pilha, retornando o topo e modificando o topo da pilha.

Leitura

Neste código foram utilizados o ***“argc”*** e ***“argv”*** para que na execução do código possa ser escrito qual dos labirintos se quer rodar. A exemplo disso para executar o labirinto 1 deve se compilar o código e digitar: ***./lab “labirintos/labirinto1.txt”.***

Logo em seguida é feita a leitura do arquivo onde o ponteiro arq. recebe o endereço do labirinto como argumento, é verificado se arq é “***NULL”***, caso seja significa que não foi encontrado arquivo. Se é diferente de ***“NULL”*** o labirinto é guardado em uma matriz 10x10. Na matriz, ao ler sempre é descartado os ***“newlines (\n)”,*** que sempre são encontrados ao fim das linhas.

Resolução

```
//ENCONTRA A ENTRADA
for(int i = 0; i < 10; i++)
    for(int j = 0; j < 10; j++)
        if(lab[i][j] == 69){
            push(i, j, &cam);
        }
```

```
int saida = 1;//para o código quando a saída é encontrada
int conta;//conta o número de caminhos seguintes possíveis
int sonda;//guarda a posição que será analizada
no* tmp;
```

Essa primeira parte do código tem o intuito de encontrar na matriz, onde é o local de entrada.

```
while(saida == 1){//roda enquanto a saída não tiver sido encontrada
    conta = 0;//reinicia o valor toda vez que o loop começa
    int xa = cam -> x;//xa = x atual
    int ya = cam -> y;//ya = y atual
```

Esse while tem a condição de continuar enquanto a saída não for encontrada dentro da matriz.

```
if(xa > 0){//garante que há casa a ser explorada em cima
    sonda = lab[xa-1][ya];//CIMA

    switch(sonda){
        case 48://48 = ZERO
        push(xa-1, ya, &bif);
        conta++;
        break;
        case 83://83 = S
        push(xa-1, ya, &cam);
        saida = 0;
        conta++;
        break;
    }
}
```

O "if" verifica se a variável "XA" é maior que zero, pois, quando "XA" é igual a zero, não é possível explorar a posição acima, uma vez que isso representa uma barreira (como uma "parede"). Nesse caso, a variável "sonda" recebe o valor de "XA - 1" e "YA" (o que corresponde a verificar a posição imediatamente superior), sendo que essa variável "sonda" será utilizada posteriormente em uma estrutura "switch".

No "case" 48, ocorre a inserção na pilha de bifurcação das coordenadas "XA - 1" e "YA".

No "case"83, a instrução serve para verificar se a posição "XA - 1" e "YA" corresponde à posição onde o ponto "S" está localizado. Caso essa condição seja verdadeira, o laço "while" será interrompido.

```
if(ya > 0){//garante que há casa a ser explorada na esquerda
    sonda = lab[xa][ya-1];//ESQUERDA

    switch(sonda){
        case 48://48 = ZERO
        push(xa, ya-1, &bif);
        conta++;
        break;
        case 83://83 = S
        push(xa, ya-1, &cam);
        saida = 0;
        conta++;
        break;
    }
}
```

O "if" verifica se a variável "YA" é maior que zero, pois, quando "YA" é igual a zero, não é possível explorar a posição da esquerda, uma vez que isso representa uma barreira (como uma "parede"). Nesse caso, a variável "sonda" recebe o valor de "XA " e "YA - 1 " (o que corresponde a verificar a posição da esquerda), sendo que essa variável "sonda" será utilizada posteriormente em uma estrutura "switch".

No "case" 48, ocorre a inserção na pilha de bifurcação das coordenadas "XA " e "YA - 1".

No "case"83, a instrução serve para verificar se a posição "XA" e "YA - 1" corresponde à posição onde o ponto "S" está localizado. Caso essa condição seja verdadeira, o laço "while" será interrompido.

```
if(xa < 9){//garante que há casa a ser explorada embaixo
    sonda = lab[xa+1][ya];//BAIXO

    switch(sonda){
        case 48://48 = ZERO
        push(xa+1, ya, &bif);
        conta++;
        break;
        case 83://83 = S
        push(xa+1, ya, &cam);
        saida = 0;
        conta++;
        break;
    }
}
```

O "if" verifica se a variável "XA" é menor que nove, pois, quando "XA" é igual a nove, não é possível explorar a posição de baixo, uma vez que isso representa uma barreira (como uma "parede"). Nesse caso, a variável "sonda" recebe o valor de "XA + 1 " e "YA " (o que corresponde a verificar a posição de baixo), sendo que essa variável "sonda" será utilizada posteriormente em uma estrutura "switch".

No "case" 48, ocorre a inserção na pilha de bifurcação das coordenadas "XA + 1 " e "YA".

No "case"83, a instrução serve para verificar se a posição "XA + 1" e "YA" corresponde à posição onde o ponto "S" está localizado. Caso essa condição seja verdadeira, o laço "while" será interrompido.

```
if(ya < 9){//garante que há casa a ser explorada na direita
    sonda = lab[xa][ya+1];//DIREITA

    switch(sonda){
        case 48://48 = ZERO
        push(xa, ya+1, &bif);
        conta++;
        break;
        case 83://83 = S
        push(xa, ya+1, &cam);
        saida = 0;
        conta++;
        break;
    }
}
```

O "if" verifica se a variável "YA" é menor que nove, pois, quando "YA" é igual a nove, não é possível explorar a posição da direita, uma vez que isso representa uma barreira (como uma "parede"). Nesse caso, a variável "sonda" recebe o valor de "XA" e "YA + 1" (o que corresponde a verificar a posição da direita), sendo que essa variável "sonda" será utilizada posteriormente em uma estrutura "switch".

No "case" 48, ocorre a inserção na pilha de bifurcação das coordenadas "XA" e "YA + 1".

No "case"83, a instrução serve para verificar se a posição "XA" e "YA + 1" corresponde à posição onde o ponto "S" está localizado. Caso essa condição seja verdadeira, o laço "while" será interrompido.

```c
if (saida == 1)
{
    switch (conta)
    {

    case 1:               // há apenas um caminho a ser percorrido
        lab[xa][ya] = 67; // a posição atual recebe C
        tmp = pop(&bif);  // pega o primeiro da pilha de bifurcações e guarda na pilha de caminho (próximo a ser acessado)
        push(tmp->x, tmp->y, &cam);
        break;
```

Caso a saída não tenha sido encontrada o programa entra em um switch que leva em consideração a variável conta, com os possíveis casos:

Caso 1, isso significa que existe apenas um caminho para ser percorrido sendo assim o único nó na pilha de bifurcações é colocado na pilha do caminho a posição tem o “0” trocado por “C”.

```c
case 2:               // há dois caminhos a serem percorridos
    lab[xa][ya] = 67; // a posição atual recebe C
    tmp = pop(&bif);  // pega o primeiro da pilha de bifurcações e guarda na pilha de caminho (próximo a ser acessado)
    push(tmp->x, tmp->y, &cam);
    push(xa, ya, &bif); // guarda a posição onde ocorreu a bifurcação
    break;
```

Caso 2, então temos mais de um caminho para percorrer, primeiramente marcamos o local atual como percorrido trocando o “0” por “C”, colocamos o primeira da pilha de bifurcação na pilha de caminho para ser testado e é guardado onde ocorreu a bifurcação.

```c
case 0: // não há mais caminhos
    while (cam->x != bif->x || cam->y != bif->y)
    { // volta até a bifurcação anterior
        tmp = pop(&cam);
        free(tmp); // remove o valor do topo de cam
    }
    pop(&bif);        // remove o endereço da bifurcação
    tmp = pop(&bif); // passa o endereço do próximo caminho para cam
    push(tmp->x, tmp->y, &cam);
    break;
```

Caso 0, então não a mais caminho a ser percorrido, o programa volta até a bifurcação anterior removendo o valor do topo da pilha de caminho.

```c
default: // caso de erro
    printf("erro\n");
    saida = 0;
    break;
```

E caso a variável conta tenha como resultado um valor diferente o programa exibe uma mensagem de erro.

```
while (bif != NULL)
{ // esvazia bif
    tmp = pop(&bif);
    free(tmp);
}

while (cam != NULL)
{ // inverte a ordem de cam em resultado
    tmp = pop(&cam);
    push(tmp->x, tmp->y, &resultado);
    free(tmp);
}

printpilha(resultado);
printlab(lab);
```

Ao final do programa a pilha de bifurcação é esvaziada e a pilha de caminho é colocada invertida dentro da pilha resultados para que possa ser impressa na ordem correta.

Complexidade

No pior caso todas as células serão visitadas apenas uma única vez, sendo assim, a complexidade do algoritmo é O(n), sendo n o número máximo de células do labirinto.

Makefile

O makefile utilizado primeiramente cria um arquivo objeto chamado" lab.o" a partir do arquivo "main.c", então criando o executável lab.exe a partir do arquivo objeto. Para compilar o código basta utilizar o comando "make all".