CHEGANDO JUNTO - 1057

Gabriel Henrique Pires dos Santos
Centro de Educação Tecnológica Celso Suckow da Fonseca - CEFET/RJ
Petrópolis - RJ

RESUMO

Este documento tem por objetivo trazer uma explanação sobre os métodos utilizados na resolução do problema proposto para a atividade avaliativa. O desafio passado possui numeração 1057 no site Beecrowd e é nomeado Chegando junto, ao longo do texto será abordado em mais detalhes sobre do que se trata.

PALAVRAS CHAVE. Grafos, Busca, Algoritmos.

Busca em largura em grafos, movimentação em grafos, caminhos

ABSTRACT

This document aims to provide an explanation of the methods used to solve the problem proposed for the evaluative activity. The challenge is numbered 1057 on the Beecrowd website and is titled ”Getting Together.”Throughout the text, further details will be provided about what it entails.

KEYWORDS. Graph. Search. Algorithm.

Breadth-First Search (BFS) in graphs, movement in graphs, paths

1. Introdução

O problema se baseia em uma matriz quadrada **NxN** formada por caracteres, onde o caracter '.' indica uma casa livre, o '#' um obstáculo, o 'X' uma chegada e as letras 'A', 'B' e 'C' representam 3 robôs. Como entrada é enviado primeiramente a quantidade **T** de casos a serem analisados, depois a ordem **N** da matriz, e finalmente a matriz contendo os três robos e três chegadas.

Quando um comando de movimentação é ativado, todos os robôs tentam se mover na direção indicada, porém podem ser impedidos de concluir o movimento caso haja um obstáculo. O objetivo é propor um algoritmo que calcule o menor número de movimentos para que os três robôs estajam cada um sobre uma chegada ao mesmo tempo, o problema não deixa claro se após a um robô chegar ele pode deixar a marcação, porém para resolução correta e mais precisa, foi considerado que isso era possível. Por fim, caso não exista solução deve ser impresso na tela como retorno **Trapped**.

2. Algoritmo Proposto para Solução

O algoritmo escolhido para este problema foi o **BFS** (Breadth-First Search), que consiste na execução de uma busca em largura no grafo, partindo inicialmente de um vértice e percorrendo os nós adjacentes dele [Cormen, 2012]. Majoritariamente, este algoritmo é utilizado para encontrar o menor caminho entre dois nós no grafo. Para isso, ocorre a utilização de mecanismos auxiliares, como a fila, usada para gerenciar os vértices a serem visitados, já que o BFS segue o princípio **FIFO** (First in, First out).

Ao decorrer deste relatório será explicado a aplicação dos conceitos citados, com alterações pontuais para um melhor aproveitamento da teoria e uma resposta preferível em relação ao entendimento do código.

2.1. Estruturas Utilizadas

Foram implementadas duas estruturas, onde a struct **coordenadas** é utilizada para guardar um par **(x, y)** que representa uma posição no mapa, enquanto a estruct **estado** é utilizada para guardar a quantidade de movimentos e a posição atual dos robôs A, B e C.

```
1  // Definicao de uma estrutura para coordenadas
2  typedef struct {
3      int x, y;
4  } coordenadas;
5
6  // Definicao de uma estrutura para estado
7  typedef struct {
8      coordenadas A, B, C;
9      int moves;
10 } estado;
```

2.2. Definições de Constantes e Variáveis

A criação de constantes foi baseada nos limites de memória e no uso repetitivo de variáveis. Para facilitar na hora de criar ou utilizar funções, foram criadas variáveis globais, a seguir será explicado a implementação de algumas variáveis que serão utilizadas no código. O mapa é declarado como uma matriz de caractere **map[MAX_N][MAX_N]**; o vetor de inteiros **vis** possui 6 dimensões para poder representar a combinação da posição dos três robôs; **dx[] e dy[]** representam as possíveis direções de movimento; também foi definido a **Fila[MAX_FILA]** afim de armazenar as posições A, B e C.

```c
#define MAX_N 12    // Constantes declaradas
#define MAX_FILA (MAX_N * MAX_N * MAX_N * 100)

char map[MAX_N][MAX_N]; // Variaveis globais
int vis[MAX_N][MAX_N][MAX_N][MAX_N][MAX_N][MAX_N];
int dx[] = {-1, 1, 0, 0}; //inicializa possiveis direcoes
int dy[] = {0, 0, -1, 1};
int N;
coordenadas A, B, C;
coordenadas a, b, c;
estado Fila[MAX_FILA];
int inicio, fim;
```

2.3. Implementação Mecanismos de Fila e Checagens

Nesta parte foram declaradas funções de fila, como a inicialização, que zera as variáveis iníco e fim; a verificação se a fila está vazia, ou seja, início e fim sobrepostos; o armazenamento de valor na fila e o exclusão do elemento hÃ¡ mais tempo no vetor, seguindo a lógica FIFO.

```c
void inicializarFila() {
    inicio = fim = 0;
}
int filaVazia() {
    return inicio == fim;
}
void enfileira(estado val) {
    Fila[fim++] = val;
    if (fim >= MAX_FILA) fim = 0;
}
estado desinfileira() {
    estado val = Fila[inicio++];
    if (inicio >= MAX_FILA) inicio = 0;
    return val;
}
```

A seguir são criadas funções de checagem, a **podeir(coordenadas aux)** recebe um ponto e verifica se o mesmo colide com algum obstáculo ou se está fora dos limites do mapa, retorna 1 caso aquela casa esteja liberada, caso contrário, retorna 0; a **chegou()** verifica se o jogo acabou, comparando as posições dos robôs com o caracter que representa a chegada, retorna 1 se verdade, e 0 caso seja falsa.

```c
int podeir(coordenadas aux) {
    return aux.x >= 0 && aux.x < N && aux.y >= 0 && aux.y < N && map[aux
        .x][aux.y] != '#';
}
int chegou() {
    return map[a.x][a.y] == 'X' && map[b.x][b.y] == 'X' && map[c.x][c.y]
         == 'X';
}
```

3. BFS - Função de Busca em Largura

A execução do algoritmo BFS começa pela inicialização de uma fila que armazenará os estados possíveis do problema. O estado inicial, que representa as posições iniciais das entidades **A, B e C**, é enfileirado e marcado como visitado no vetor de 6 dimensões vis para evitar revisitas. Em seguida, o algoritmo processa cada estado da fila, expandindo-o para gerar novos estados ao tentar mover cada entidade em todas as direções possíveis (cima, baixo, esquerda e direita). Antes de enfileirar um novo estado, verifica-se se as posições são válidas (dentro dos limites do labirinto e sem colisões entre entidades) e se o estado ainda não foi visitado. Este processo continua até que o estado objetivo (todas as entidades atingirem suas posições finais) seja encontrado ou até que todos os estados possíveis sejam explorados. Assim, o BFS garante a busca do menor caminho em termos de tempo ou movimentos, respeitando as restrições impostas pelo problema.

```
void BFS(int t) {
    ...
    enfileira(atual);
    vis[A.x][A.y][B.x][B.y][C.x][C.y] = 1;
    while (!filaVazia()) {
        atual = desinfileira();
        if (chegou()) { ...}\\Termina a execucao
        for (int i = 0; i < 4; i++) {...   // Checa todas as direcoes
            if (!podeir(novoA)) novoA = a; // Faz para os tres robos
            if (novoA.x == novoB.x && novoA.y == novoB.y) {
                novoA = a;  // Se A e B colidirem, nao se movem. Faz
                    para todas possibilidades.
                novoB = b;
            }...
            if ((novoA.x == b.x && novoA.y == b.y && novoB.x == a.x &&
                novoB.y == a.y)|| ... // Verifica se um robo trocou de
                posicao com o outro
            if (!vis[novoA.x][novoA.y][novoB.x][novoB.y][novoC.x][novoC.
                y]) {...} // Preenche com 1 quando termina a visita, e
                enfileira a proxima posicao
        }
    }
    printf("Case %d: trapped\n", t); // Sem solucao
}
```

4. Conclusão

A resolução do problema “Chegando Junto“ exigiu a aplicação de conceitos fundamentais de grafos e algoritmos de busca, em particular o BFS, que se mostrou ideal para encontrar o menor caminho em um contexto de movimentação simultânea e restrita de três robôs em uma matriz. O uso de estruturas de dados como filas e a representaç ao de estados multidimensionais permitiram lidar eficientemente com a complexidade do problema, garantindo que todas as combinaç oes possíveis de posiç ões fossem avaliadas de forma sistem’atica e sem repetições.

Além disso, a modelagem clara das regras, como movimentaç oes válidas, colisões e condições de finalização, assegurou que a solução fosse robusta e aderente aos requisitos. Embora o problema tenha limitações impostas pelo tamanho da matriz, a implementação demonstrou ser eficiente dentro dos limites estabelecidos.

Referências

Cormen, T. H. (2012). *Algoritmos - Teoria e Prática*. Elsevier, 3ª edição, Rio de Janeiro.