As misteriosas chaves do reino perdido

Anderson R. P. Sprenger

Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681 – 90.619-900 – Porto Alegre – RS – Brasil

anderson.sprenger@edu.pucrs.br

Resumo: Este artigo apresenta uma solução para buscar a quantidade de espaços que um jogador pode explorar partindo de uma determinada posição em um cenário de jogo representado por uma matriz de caracteres em um arquivo de texto. É inclusa uma descrição detalhada do problema, sua análise e uma proposta de solução utilizando um algoritmo de busca em largura sobre um grafo representado pela matriz do cenário. Para tal foram testados os 6 casos fornecidos e analisados os resultados obtidos. Este artigo é referente ao trabalho 2 da disciplina de Algoritmos e Estruturas de Dados 2 da Pontifícia Universidade Católica do Rio Grande do Sul.

1. Introdução

Uma empresa de jogos eletrônicos busca testar o cenário de seu novo jogo para descobrir quantas posições um jogador pode se mover a partir de um ponto de início. Nele contém também várias salas, pontos de início de vários jogadores, chaves e portas trancadas.

Foram disponibilizados casos de teste que representam o cenário de forma semelhante a um mapa, tal como na Figura 1. Ele contém em cada posição do cenário um caractere que é diferente dependendo do que há na posição, podendo ser:

- # - os espaços intransponíveis que compõem as paredes das salas do cenário;
- **.** - os espaços livres por onde o jogador pode se mover pela sala;
- **1-9** - os pontos de início dos jogadores;
- **a-z** - as chaves;
- **A-Z** - e as portas trancadas.

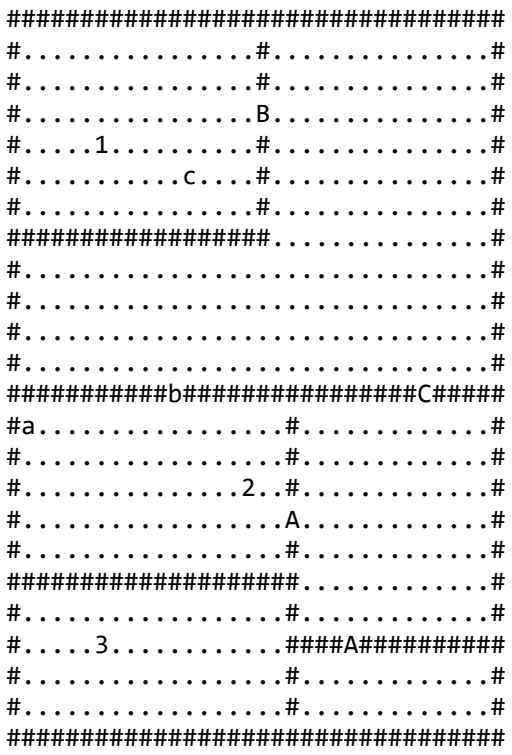

Figura 1: Exemplo de cenário apresentado no enunciado. Nele o jogador 1 pode se mover por 96 posições, o jogador 2 por 541 posições e o jogador 3 por 72 posições.

Portas e chaves podem estar repetidas. As chaves são coletadas quando um jogador se move para uma posição com uma chave e elas podem ser usadas várias vezes para destrancar qualquer porta representada pela mesma letra da chave. As portas trancadas são intransponíveis para o jogador até serem destrancadas, sendo considerada não explorada enquanto não estiver aberta.

2. Análise do Problema

O cenário a ser explorado é representado de modo conveniente para sua conversão a uma matriz de caracteres, com um vetor guardando uma linha do arquivo de texto dividida em um segundo vetor. Isso torna fácil se mover de uma posição para outra a partir das coordenadas x e y correspondentes aos índices da matriz.

Partindo do ponto de início, o jogador deve se mover na vertical ou na horizontal, como mostrado na Figura 2, havendo deste modo 4 nodos diferentes como possíveis destinos partindo de cada posição. Desta forma é possível tratar a matriz do cenário como um grafo, onde cada posição é um vértice que está ligada aos seus 4 possíveis destinos, quando transponíveis, por arestas.

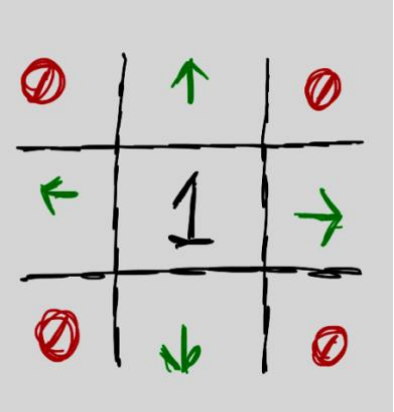

Figura 2: Movimentos possíveis a partir da posição inicial de um jogador, representado pelo número 1. As setas indicam os possíveis destinos para um jogador pode realizar seu próximo movimento.

3. Proposta de Solução

Os algoritmos de busca em largura, em inglês *Breadth-First Search* (BFS), e busca em profundidade, em inglês *Depth-First Search* (DFS), são otimizados para percorrer um grafo a partir de um vértice raiz, neste caso o ponto de início do jogador. Eles possuem complexidade $O(v + a)$, onde v é o número de vértices e a é o número de arestas [Reif, 1985].

Como é necessário iterar por todas as posições possíveis, não há diferença na ordem da busca, com os algoritmos BFS e DFS alcançando performance semelhante. Deste modo, foi escolhido arbitrariamente entre as opções o algoritmo DFS, descrito a seguir com a adição de um contador, para realização do trabalho.

```
FUNC dfs(v: Vertice) -> Int:
    verticesMarcados = []
    pilha = [v]

    WHILE pilha não vazia:
        verticeVisitado = pilha.removeUltimo()

        FOR todasArestas a do verticeVisitado:
            verticesMarcados += [a]
            pilha += [a]
        END FOR
    END WHILE

    RETURN verticesMarcados.quantidade()
END FUNC
```

Algoritmo 1: Busca em profundidade com um contador de vértices visitados.

Para adaptar este algoritmo ao problema, é assumido que o grafo tem as 4 arestas referentes aos movimentos possíveis, conforme ilustrado na figura 2. A iteração das arestas é implementada com 4 chamadas de uma função auxiliar, explicada a seguir, que cuidara também de marcar e adicionar os vértices explorados na pilha.

```
    WHILE pilha não vazia:
       vv = pilha.removeUltimo()

       explora(v: Vertice(x: vv.x + 1, y: vv.y))
       explora(v: Vertice(x: vv.x - 1, y: vv.y))
       explora(v: Vertice(x: vv.x, y: vv.y + 1))
       explora(v: Vertice(x: vv.x, y: vv.y – 1))
    END WHILE
```

Algoritmo 2: Trecho da iteração da pilha do DFS modificado para solução do trabalho, mantendo as outras partes do Algoritmo 1 na solução.

Segundo o enunciado [de Oliveira, 22], não é possível visitar vértices correspondentes a paredes ou a portas trancadas antes de encontrar sua chave correspondente. Além disso, uma porta só conta como explorada quando estiver aberta. O algoritmo a seguir cuida destas logicas de jogo, marcando e adicionando o vértice explorado na pilha conforme as regras do problema.

```
FUNC explora(v: Vertice) -> Int:
    IF verticesMarcados.contem(v):
        RETURN
    END IF
    caractere = grafo[v.x][v.y]
    SWITCH caractere:
    case “#”:
        RETURN

    case 1-9 ou .:
        verticesMarcados += [v]
        pilha += [v]

    case a-z:
        verticesMarcados += [v]
        pilha += [v]
        conjuntoChaves += [v]
        FOR porta em portasTrancadas:
            p = grafo[porta.x][porta.y]
            p = l.minusculo()
            IF caractere == p:
                portasTrancadas.remove(porta)
                verticesMarcados += [porta]
                pilha += [porta]
            END IF
        END FOR
```

```
    case A-Z:
        IF conjuntoChaves
                .contem(caractere.minusculo()):
            verticesMarcados += [v]
            pilha += [v]

        ELSE:
            portasTrancadas += [v]
        END IF
    END SWITCH
END FUNC
```

Algoritmo 3: Função auxiliar da iteração das arestas do algoritmo 2, cuidando da logica de exploração do jogo.

Para encontrar a posição inicial dos jogadores, foi feita uma pesquisa sequencial na matriz do cenário, guardando as posições dos caracteres referentes aos números 1 até 9 em um conjunto. Este conjunto foi iterado, com o algoritmo acima sendo executado a partir dos índices da posição inicial de cada jogador na matriz.

4. Resultados da Solução

Ao executar a solução com o caso de teste fornecido no enunciado foram obtidos os valores informados para o exemplo, com os valores dos outros casos dentro do esperado. Apesar dos dados de tempo de execução, dependerem do hardware, sistema operacional, entre outros, o algoritmo executou em menos de 1 segundo em todos os casos exceto o 10, onde executou em menos de 4 segundos.

Caso	0	5	6	7	8	9	10
Tempo (s)	0.0052	0.0323	0.0205	0.0827	0.2392	0.9238	3.5284
Jogador 1	96	113	608	167	2006	309	5817
Jogador 2	541	113	321	33	269	685	9523
Jogador 3	72	1065	189	2020	1603	4611	45
Jogador 4	x	1065	815	2483	873	2433	8611
Jogador 5	x	1065	21	571	105	4579	21
Jogador 6	x	21	45	2762	377	3653	2133
Jogador 7	x	77	9	115	1395	1437	847
Jogador 8	x	9	153	2762	913	825	1089
Jogador 9	x	57	33	525	1125	749	2505

Tabela 1: Quantidade de posições que um jogador pode explorar encontrada para nos casos de teste, com o tempo de execução da solução em cada caso.

5. Conclusão

Os dados obtidos com a solução proposta para o caso de teste presente no enunciado são semelhantes aos esperados. A performance do algoritmo de busca em profundidade é de complexidade $O(v + a)$. Contudo, a função auxiliar só adiciona o vértice na pilha do algoritmo uma vez, com exceção das portas trancadas esperando pela chave, decorrendo em uma complexidade de aproximadamente $O(n)$, onde n é o número de caracteres no cenário. Tendo em vista que a iteração sobre os caracteres eleva o patamar da complexidade do algoritmo para no mínimo $O(n)$ pois são necessárias verificações para todos os n caracteres, considero adequada a solução proposta.

6. Referencias

de Oliveira, J. B. (2022). As misteriosas chaves do reino perdido.

Reif, J. H. (1985). Depth-first search is inherently sequential. Information Processing Letters, 20(5):229–234.