Trabalho 2: As Misteriosas Chaves do Reino Perdido

Pedro da Cunha Gaspary (21101429)

Escola Politécnica - PUCRS

14 de junho de 2022

Resumo

Este relatório descreve uma solução para o segundo problema proposto na disciplina de Algoritmos e Estruturas de Dados II. Este problema trata de encontrar o número máximo de casas que cada jogador pode explorar em um cenário de caracteres. A solução encontrada é descrita detalhadamente; e, finalmente, são apresentados os resultados para os casos de teste disponibilizados.

Problema

O problema apresentado propõe que sejam encontradas o número máximo de casas que um jogador pode explorar em um cenário, seguindo regras pré-estabelecidas. O cenário é um arquivo de entrada, como visto na Figura 1, em que cada caractere é um objeto diferente que pode ser explorado:

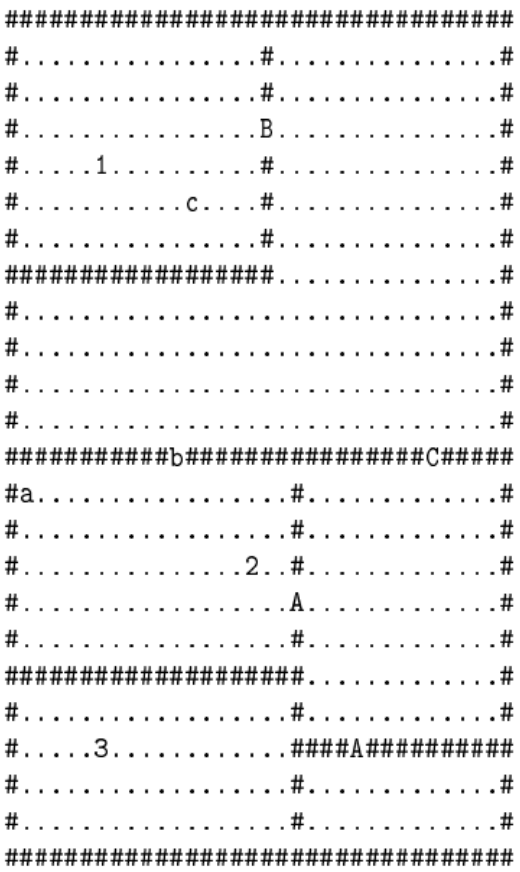

Figura 1. Exemplo de cenário

- **‘#’** são paredes por onde não se pode passar
- **‘.’** representa um espaço livre
- **‘a-z’** são chaves que podem ser coletadas e não bloqueiam o caminho
- **‘A-Z’** são portas fechadas que podem ser abertas se o jogador possuir a chave adequada
- **‘1-9’** indicam a posição por onde cada jogador começa a exploração

Os jogadores não se movem na diagonal e não podem explorar os espaços onde encontrarem paredes. Também, não podem ser exploradas as portas fechadas; neste caso, se a chave correspondente não tiver sido encontrada. A Figura 2 demonstra graficamente o resultado para o cenário de exemplo da Figura 1. Nesse cenário, o jogador 1 pode explorar apenas 96 casas, já que encontra uma chave {c}, a qual não corresponde com a porta {B} de sua sala. Já o jogador 2,

explora 541 casas, visto que pode abrir as portas {A, B, C} com as chaves {a, b, c} encontradas. O jogador 3, restringido a um espaço fechado, explora 72 casas.

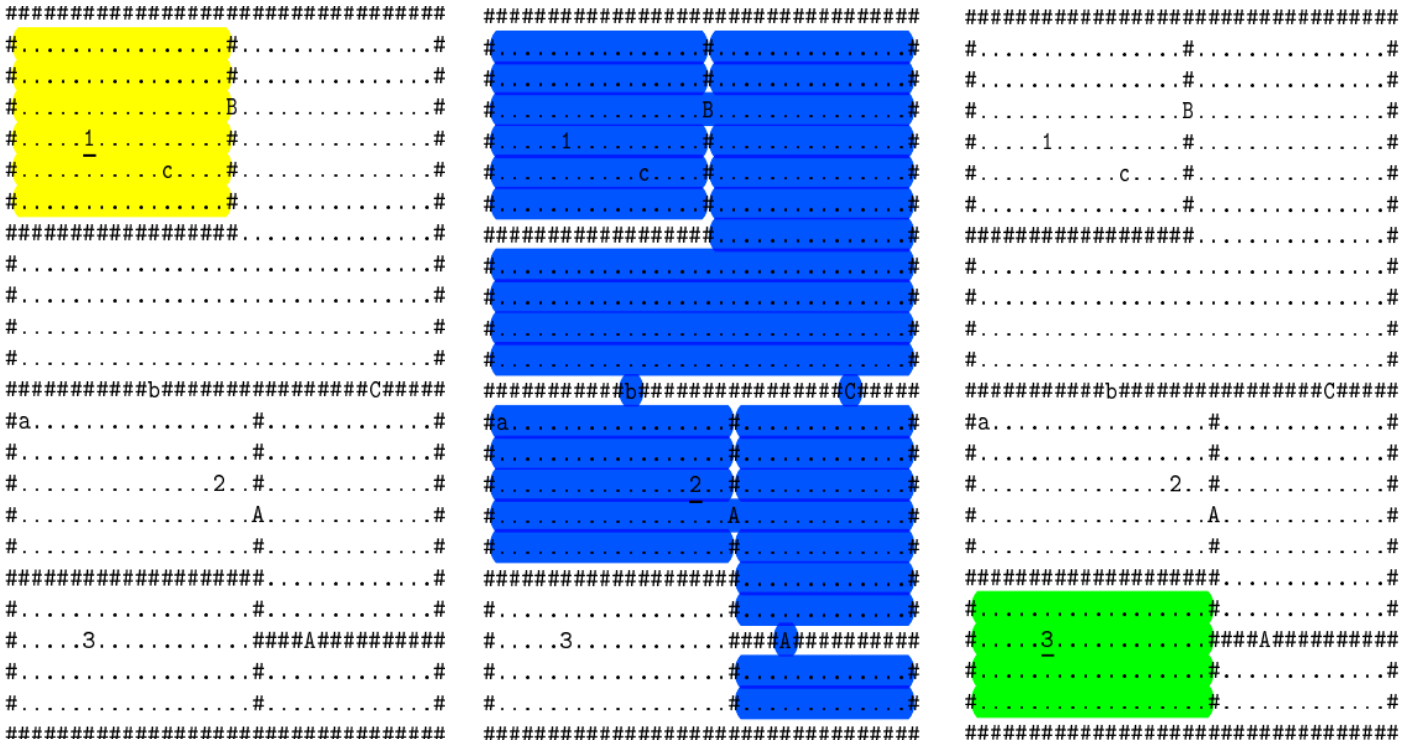

Figura 2. Exemplo gráfico de resultado para um cenário

Modelagem da Solução

Para a solução, o cenário é tratado como um grafo, em que cada casa é um nodo. Dessa maneira, o cenário foi mapeado em uma matriz de nodos. O código foi desenvolvido em java 18.0.1, assim, foi criada uma classe pública que descreve a estrutura de um objeto nodo. Um nodo é ligado aos seus vizinhos na matriz, a partir de sua coordenada *(i, j)*, onde *i* é o índice da linha na matriz e *j* é o índice do nodo naquela linha.

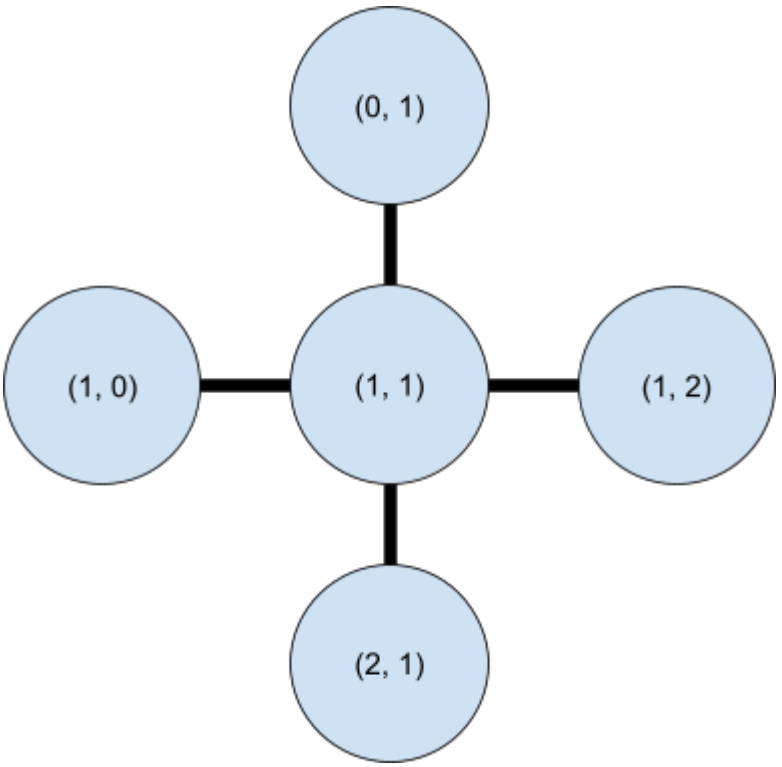

Figura 3. Exemplo de grupo de nodos vizinhos

Graças ao conceito de índices inerente à matriz, não é necessário que o nodo guarde o endereço de seus vizinhos. Por conseguinte, o objeto nodo não necessita de arestas definidas na sua estrutura. A Figura 3 exemplifica o grupo de vizinhos do nodo (1, 1) e suas coordenadas na matriz. Além da coordenada na matriz, cada instância de nodo armazena o caractere correspondente a sua posição no cenário e possui um booleano para indicar se o jogador já o explorou ou não.

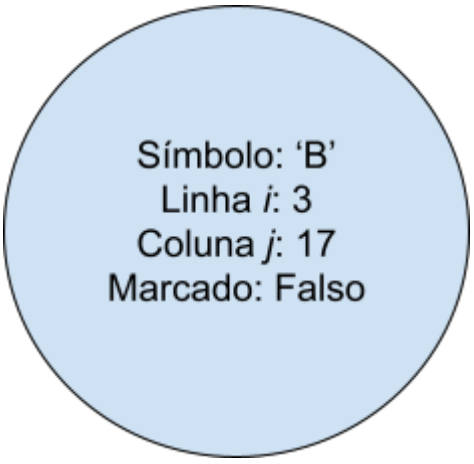

Figura 4. Exemplificação da estrutura inicial de um nodo

A Figura 4 demonstra como seria a estrutura completa do nodo com o caractere de porta {B}, recém criado na matriz do cenário exemplo da Figura 1. No caso de um jogador encontrar a chave {b}, como acontece com o jogador 2, a variável ‘marcado’ seria alterada para o valor verdadeiro como aparece na figura 5.

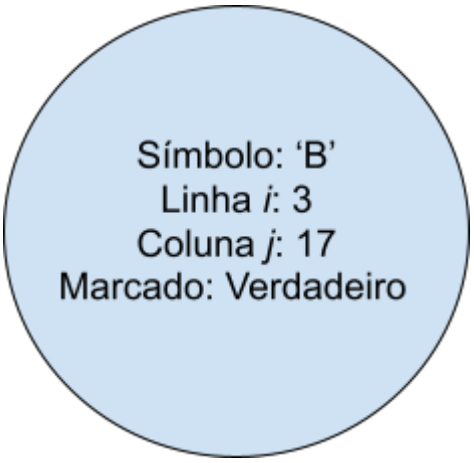

Figura 5. Exemplificação da estrutura de um nodo, quando visitado

Solução

Para solucionar o problema, o programa começa lendo o arquivo de entrada em uma lista de Strings. Posteriormente, essa lista é mapeada em uma matriz de nodos. Posto que cada nodo de posição (*i*, *j*) receba o caractere que está na linha com índice *i* do texto, e índice *j* da linha.

Durante o processo de mapeamento da matriz, quando um nodo é criado com um caractere (1-9), ele é guardado em um conjunto de nodos. Subsequentemente, esse conjunto será aproveitado para fornecer os nodos por onde cada jogador inicia a contagem de casas a serem exploradas. Salienta-se que todas as instâncias de nodo são iniciadas com o seu booleano falso.

Visando a evitar conflito entre a exploração de jogadores diferentes, o programa testa o problema para cada jogador individualmente. Isto é, posteriormente à execução para um jogador, desmarca-se todos os nodos da matriz, tornando os seus booleanos falsos. Garantindo assim, que um jogador não deixe de explorar nodos visitados por outro.

O algoritmo usado para descobrir quantas casas um jogador pode explorar é baseado na ideia de caminhamento em largura para grafos. Essa ideia consiste em, a partir de um nodo inicial, explorar seus vizinhos, e depois os vizinhos de seus vizinhos, até não haver mais nodos para explorar.

Para tanto, foi criada uma lista de nodos a serem explorados, inicializada apenas com o nodo de onde o jogador inicia o jogo. Além disso, se fez necessário criar uma variável para contar o número de casas exploradas. Enquanto esta lista não estiver vazia, o seu primeiro elemento é retirado. Esse nodo é, então, marcado, alterando seu booleano para verdadeiro, e o contador de casas exploradas é incrementado em um.

Então, verifica-se se algum dos vizinhos do nodo retirado não está marcado, com o booleano falso, o que indicaria que ainda não foi explorado. Ademais, deve-se confirmar que o caractere desse vizinho não é uma parede. No caso de ambas verificações serem confirmadas, esse nodo vizinho é adicionado à lista de nodos a serem visitados.

Ademais, se o nodo vizinho sendo analisado possuir um caractere de porta (A-Z), ele é adicionado a uma lista de portas que o jogador encontrou. Similarmente, um conjunto é criado com os caracteres de chave (a-z) que foram encontrados. Sendo confirmado que o nodo vizinho em análise não é uma parede, não foi explorado ainda e não é uma porta; ele é marcado e é adicionado à lista de nodos a serem visitados.

Finalmente, é analisado se o jogador encontrou uma chave que abre alguma das portas que encontrou. Se esse for o caso, o nodo correspondente àquela porta é adicionado à lista de nodos a serem visitados e ele é marcado. Com isso, a porta agora aberta, poderá ser transposta, permitindo que diversos novos nodos sejam explorados.

Caminhamento em Largura (grafo, jogador):

INICIALIZA: o contador de casas exploradas em 0
INICIALIZA: a lista de nodos com o nodo do jogador
CRIA: lista de portas encontradas pelo jogador
CRIA: conjunto de chaves encontradas pelo jogador

ENQUANTO: a lista de nodos não estiver vazia

REMOVE: o primeiro nodo V da lista
MARCA: o booleano de V como verdadeiro
INCREMENTA: o contador em 1

PARA: cada nodo vizinho de V
SE: o vizinho não estiver marcado & não for parede (#)
SE: o vizinho for porta (A-Z)
ADICIONA: o vizinho à lista de portas
SE: o vizinho não for porta (A-Z)
MARCA: o booleano do vizinho como verdadeiro
ADICIONA: o vizinho à lista de nodos a serem explorados
SE: o vizinho for chave (a-z)
ADICIONA: o caractere do vizinho à lista de chaves

PARA: cada nodo porta na lista de portas
SE: o conjunto de chaves possui a chave correspondente à porta & a porta não está marcada
MARCA: o booleano da porta como verdadeiro
ADICIONA: a porta à lista de nodos a serem explorados

RETORNA: o contador de casas exploradas

Resultados

A solução foi desenvolvida em um Acer Aspire 3 rodando o sistema operacional EndeavourOS Linux x86_64. O computador conta com 4 núcleos Intel Core i5 de 10° geração, de velocidade base de 1,19 GHz e velocidade máxima de 3,6 GHz. Além disso, a máquina possui 20GB de RAM.

Os resultados encontrados estão dispostos na Tabela 1. As células representam quantas casas foram visitadas por cada um dos 9 jogadores de cada caso. Vale ressaltar que não foram disponibilizados os casos 01, 02, 03 e 04; assim, os casos estão dispostos em ordem crescente, a partir do 05. Outrossim, o tempo médio para cinco execuções de cada caso, está na coluna mais à direita.

Tabela 1. Resultados Experimentais

Jogador \Caso	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Tempo médio de Execução: (ms)**
05	113	113	1065	1065	1065	21	77	9	57	131,8
06	608	321	189	815	21	45	9	153	33	157,2
07	167	33	2020	2483	571	2762	115	2762	525	310
08	2006	269	1603	873	105	377	1395	913	1125	453,6
09	309	685	4611	2433	4579	3653	1437	825	749	1113
10	5817	9523	45	8611	21	2133	847	1089	2505	3178,2

Conclusões

Finalmente, conclui-se que a solução se mostrou de simples implementação, ao mesmo tempo que se mostrou eficiente para um problema complexo. Dessarte, este trabalho serviu como um importante

ensinamento na modelagem e uso de grafos para solução de problemas.