Documentação - Trabalho Prático 0

Amanda Virchele Souza

Outubro 2016

1 Introdução

Nesse problema, simularemos um campo de batalha Pokémon, onde treinadores irão caçar pokemons e competir para se tornarem os grandes mestres do ginásio.

O ginásio será simulado por uma matriz bidimensional, que poderá contar com regiões vazias, Pokémons e Pokestops (Onde os treinadores poderão recarregar suas pokebolas). Cada Pokémon contará com um nível que mede sua força, chamado CP.

Cada jogador irá iniciar com 3 pokebolas, e será utilizada 1 Pokébola para a captura de cada Pokémon. Ao capturar um Pokémon, ou passar por um Pokéstop, o mesmo ficará indisponível.

No fim do jogo, o jogador que possuir a maior pontuação se tornará o vencedor. Caso mais de um jogador possua a mesma pontuação mais alta, serão submetidos aos seguintes critérios de desempate: Números de pokemons com CP alto, e menor número de deslocamentos. Se mesmo assim permanecerem empatados, ambos são declarados vencedores.

A resolução desse problema envolve tomada de decisões (Para definir o melhor caminho a ser seguido em cada situação), comparação de tipos abstratos de dados (Para definir quem será o vencedor, e quais as possibilidades em cada casa adjacente ao treinador) e manipulação de arquivos (Para a leitura do ginásio inicial, das coordenadas dos jogadores e para a saída do programa, contendo a movimentação dos jogadores e os respectivos vencedores).

2 Implementação

2.1 Funcionamento do Programa

A solução proposta busca a melhor rota para cada jogador apresentado considerando a sua posição inicial e as células ao seu redor, e armazena, durante o seu trajeto pelo mapa, toda e qualquer informação que seja necessária para uso futuro na declaração de um vencedor entre os participantes, tais como sua pontuação (a soma de CPs dos Pokémons subtraída de todos as células perigosas que foram percorridas), Pokémons que foram capturados e seu caminho percor-

rido, que sempre será diferente do caminho dos outros jogadores (a menos que ambos tenham a mesma posição inicial).

Para o deslocamento do jogador pela matriz que representa o mapa do jogo, o simulador utiliza a submatriz que contorna a posição do jogador para a tomada de decisão sobre qual o melhor passo a ser dado a seguir, utilizando-se de comparações entre células; o programa sempre dará preferência a célula de maior valor para o deslocamento, mas sempre atentando-se ao fator "Pokébola": o deslocamento para um Pokéstop passa a não mais ser preferência em caso de o jogador ter Pokébolas; o CP de um Pokémon não fará diferença em um caso que não há Pokébolas.

O programa deve usar o comando gcc main. Funcoes.c -o main, em ambiente Linux.

2.2 Funções

O problema foi resolvido através das seguintes funções:

- void criaListaTrainer(lTrainer *lista): Cria uma nova lista de jogadores, onde cada célula representa um treinador.
- void insereTrainer(tTrainer x, lTrainer *lista): Insere um novo jogador na lista de jogadores.
- void criaListaPosicoes(lPos *lista): Cria uma nova lista, onde cada célula representa um par de coordenadas do mapa.
- void inserePosicao(tPos x, lPos *lista): Insere um novo par de coordenadas na lista geral de coordenadas.
- void imprimeInicioJogo(lTrainer lista, int numPlayer): Imprime informações iniciais do jogador no início de sua rodada.
- void imprimeNome(lTrainer lista, int numPlayer, FILE *saida): Imprime o nome do jogador vencedor no arquivo de saída.
- void imprimePassos(lPos lista, FILE *saida): Imprime o caminho percorrido pelo jogador correspondente no arquivo de saída.
- void desenhaMapa (int tam, int *map, int xPlayer, int yPlayer): Função auxiliar para ajudar na resolução de problemas, imprimindo na tela uma representação visual do mapa, com suas respectivas casas e conteúdos.
- void infoJogador(lTrainer lista, int numPlayer, int *x, int *y, int *pbs, lPos listaPos, int Pokedex[6], int *sumScore): Quando chamada, essa função recolhe as informações necessárias do jogador correspondente.
- void attJogador(lTrainer lista, int numPlayer, int x, int y, int pbs, lPos listaPos, int Pokedex[6], int sumScore, int contPassos): Função para atualizar os dados do jogador na lista de jogadores.

- void declaraVencedor(lTrainer lista, int numPlayer, FILE *saida): Função que declara o vencedor no fim do programa.

- void explore(int tam, int* map, int x, int y, int *nx, int *ny, int numPBs, int *action, int *perigo, int firstPos, int Pokedex[6], int *sumScore): Função que verifica as regiões adjacentes ao jogador e escolhe a melhor posição para o mesmo se deslocar, considerando os critérios pré-estabelecidos.

- void walk(int tam, int* map, int *x, int *y, int nx, int ny, int action, int *numPBs, int *atualPlay, lPos *listaPos, int *pokeCapturados, int *contPassos): De acordo com a ação retornada pela função explore, a função walk muda a posição do jogador, além de tornar o espaço anterior instransponível (ou não, dependendo do caso)

3 Análise de Complexidade

3.1 Custo de Espaço

No que diz respeito ao custo de espaço ocupado pelo programa em função do arquivo de entrada e da dimensão N da matriz, temos somente um fator considerável no cálculo de espaço: Os dados armazenados da leitura do arquivo. Esse fator se deve a nenhuma função necessitar de espaço dinâmico adicional para a realização dos cálculos, uma vez que todas as operações realizadas pelas funções são executadas diretamente sobre os dados principais armazenados.

A alocação dos dados consome um espaço de memória determinado. Como se trata de uma matriz quadrada, são lidos e armazenados N x N valores inteiros, representando os possíveis elementos a serem encontrados. Sendo assim, sua complexidade é dada por $O(n^2)$.

3.2 Custo de Tempo

O custo de tempo de cada uma das funções é descrita indiviualmente a seguir:

- void criaListaTrainer(lTrainer *lista): Realiza somente uma operação constante de criação de uma lista. Sua complexidade é dada então por $O(1)$.

- void insereTrainer(tTrainer x, lTrainer *lista): Também executa um número constante de operações, somente inserindo um novo elemento na lista. Sua complexidade é $O(1)$,

- void criaListaPosicoes(lPos *lista): Cria uma lista de posições, através de um numero constante de operações. Complexidade $O(1)$.

- void inserePosicao(tPos x, lPos *lista): Também executa um número constante de operações, somente inserindo um novo par de coordenadas na lista. Sua complexidade é $O(1)$,

- void imprimeInicioJogo(lTrainer lista, int numPlayer): Imprime os dados dos jogadores inicialmente, seguinto como critério o número de jogadores. Sua complexidade é então dada por $O(N)$, onde N representa o número de jogadores.

- void imprimeNome(lTrainer lista, int numPlayer, FILE *saida): Função constante para impressão do nome do jogador. Complexidade dada por $O(1)$.

- void imprimePassos(lPos lista, FILE *saida): Função que imprime o percurso de um jogador. Como o número máximo de passos dado por um jogador é 3N-1, a complexidade da função no pior caso é dado por $O(N)$, onde N representa a dimensão da matriz.

- void desenhaMapa (int tam, int *map, int xPlayer, int yPlayer): Função auxiliar para visualização do mapa. Sua complexidade é dada por $O(N^2)$, porém não é executado no programa principal, dessa forma é desconsiderada.

- void infoJogador(lTrainer lista, int numPlayer, int *x, int *y, int *pbs, lPos listaPos, int Pokedex[6], int *sumScore): Imprime as informações dos jogadores, seguindo uma ordem pré definida. Sua complexidade é dada por $O(N)$, onde N representa o número de jogadores.

- void attJogador(lTrainer lista, int numPlayer, int x, int y, int pbs, lPos listaPos, int Pokedex[6], int sumScore, int contPassos): Essa função percorre a lista de jogadores buscando o valor específico. No pior caso, ela percorrerá todos os jogadores, e sua complexidade será dada por $O(N)$, onde N representa o número de jogadores.

- void declaraVencedor(lTrainer lista, int numPlayer, FILE *saida): Busca na lista de jogadores aquele com maior pontuação, e em caso de empate segue os critérios pré-estabelecidos. Sua complexidade é dada por $O(N)$, onde N representa o número de jogadores.

- void explore(int tam, int* map, int x, int y, int *nx, int *ny, int numPBs, int *action, int *perigo, int firstPos, int Pokedex[6], int *sumScore): Essa função observa os valores adjacentes à casa em que o jogador está e avalia a melhor posição futura. Como o número de comparações é constante (8), a função sempre realiza o mesmo número de comparações independente das dimensões. Sua complexidade é dada por $O(1)$.

- void walk(int tam, int* map, int *x, int *y, int nx, int ny, int action, int *numPBs, int *atualPlay, lPos *listaPos, int *pokeCapturados, int *contPassos): Altera a posição atual do jogador, indo para a casa escolhida como melhor posição. Como essa alteração é feita através de uma operação constante, sua complexidade é $O(1)$.

4 Conclusão

Nesse trabalho, conseguimos entender o uso e implementar tipos abstratos de dados de maneira eficiente, para resolver o problema proposto. Também conseguimos desenvolver um algoritmo interessante de tomada de decisão, que deve avaliar a melhor opção dentre várias propostas. Também foi interessante trabalhar novamente com a manipulação de arquivos, uma vez que é um importante recurso computacional e permite inúmeras possibilidades.

As maiores dificuldades foram na implementação da função que define os vencedores, uma vez que são necessários vários critérios de desempate, e mesmo assim pode haver mais de um vencedor. Durante os testes, também foram corrigidas várias pequenas falhas, como uma situação em que o jogador iniciava em um Pokéstop e o programa considerava como um fim de execução.