UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
Rua Dom Manoel de Medeiros, s/n – Dois Irmãos 52171-900 Recife-PE
Fone: 81 3302 1000 www.dc.ufrpe.br

DISCIPLINA: Algoritmos e Estruturas de Dados **CÓDIGO**: 06214
DEPARTAMENTO: Computação **ÁREA**: Informática
CURSO: Licenciatura em Computação
PROFESSOR RESPONSÁVEL: Luciano Demétrio Santos Pacífico
DATA MÁXIMA DE ENTREGA: 24-10-2020

Regras do Projeto 02 – 2ª Verificação de Aprendizagem

1. **Não é permitido o uso de Estruturas de Dados prontas de Linguagens de Programação**. O aluno deve implementar suas próprias Estruturas de Dados. Na Linguagem de Programação **C**, deve-se usar **structs**. Nas demais Linguagens de Programação permitidas (vide **Regras da Disciplina**), deve-se usar **classes**.

2. **Não é permitido o uso de Algoritmos e comandos otimizados prontos de Linguagens de Programação**. Todos os algoritmos solicitados devem ser implementados pelos alunos como **procedimentos** (funções, métodos, etc.).

3. As questões que solicitam **escrita de código** devem ser resolvidas **apenas através dos recursos oferecidos pela pseudolingaugem definida para a disciplina, e dos recursos equivalentes em Linguagens de Programação reais**, sendo eles: variáveis, constantes e tipos primitivos, expressões, estruturas condicionais, estruturas de repetição, sub-rotinas, estruturas de dados homogêneas (**Arrays**) e estruturas de dados heterogêneas (registros – classes e sctructs).

4. Para o Projeto 02, os **Arrays** devem ser implementados através de **Alocação Estática**. Não será permitido o uso de **Estruturas de Dados dinâmicas** implementadas em Linguagens de Programação, como os **Vectors** e **Lists** da Linguagem de Programação **Java**, por exemplo. Deve-se usar **arrays**.

5. Como a Linguagem de Programação **Python** não suporta **arrays estáticos**, o aluno que optar por usar esta Linguagem deverá usar **Lists**, **única e exclusivamente para simular o comportamento de arrays estáticos**, de acordo com os seguintes critérios:

    a. Deve-se criar uma **List** com posições vazias através do método **append** dessa estrutura. O uso do append será permitido **apenas na alocação de memória para a variável que representará o array estático**;

    b. O limite máximo de M objetos deve ser controlado **através de código**;

    c. A Estrutura deve ser manipulada como se fosse um **array estático**, com os **procedimentos escritos pelo aluno**, **não sendo permitido** o uso de métodos, funções ou otimizações oferecidos pela Linguagem Python.

6. **Regra de Ouro**: **Todos os alunos envolvidos em cópias terão suas notas ANULADAS nas referidas questões.**

7. O arquivo de resposta com o código do projeto deve ser nomeado na forma “P02.c”, “P02.java”, etc.

8. A resposta do Projeto 02 deve ser submetida **unicamente através da tarefa criada no Google Classroom para este propósito**.

Projeto 02 – 2ª Verificação de Aprendizagem

Uma das atividades mais frequentes em jogos do tipo RTS (*Real-Time Strategy*) é a coleta de recursos. Nesses jogos, um time de mineradores (ou coletores) é usado para coletar os recursos disponíveis em áreas

específicas do jogo (reservas ou minas) e levá-los à base. Neste projeto, cada mina será capaz de conter uma quantidade máxima de mineradores por vez, devendo os demais mineradores (se existirem) aguardar a liberação de uma mina para poderem executar a coleta. Implemente, em uma **Linguagem de Programação**, as **Estruturas de Dados** e **Procedimentos** necessários para representar esse jogo, de acordo com o que é especificado abaixo.

Entrada

Cada conjunto de dados de entrada (que poderá ser fornecido via **arquivo de texto**, ou via **menu interativo com o usuário**) conterá duas linhas, com as seguintes informações:

Linha 1: *t n d r*
onde *t* é o tamanho da mina, *n* é o número de mineradores, *d* é a distância à base e *r* é o número de turnos que o jogo terá.

Linha 2: a linha 2 conterá 2*n* valores, correspondendo cada dupla de valores a um dos *n* mineradores, sendo o primeiro valor sua **chave** e o segundo sua **capacidade de coleta máxima**.

O jogo começa com uma etapa de inicialização (turno 0), onde cada minerador é atribuído à sua posição inicial. A **mina suportará no máximo *t* mineradores por vez**, sendo as **posições** dos mesmos na mina resolvidas através da técnica de *hashing* baseada na **chave de cada minerador**.

A ***hash function*** utilizada para determinar a posição de um minerador é dada abaixo:

$$h(k) = k \bmod t$$

onde *k* é a chave do minerador. **Enquanto houver posições disponíveis na mina**, as colisões deverão ser resolvidas pela técnica ***Linear Probing* (*Open Address Hashing*)**. Caso a **mina já esteja cheia** (ou seja, caso suas *t* posições já estejam ocupadas), colisões devem ser resolvidas por ***Closed Address Hashing***.

Após a etapa de inicialização, o jogo será executado durante ***r* turnos**. Em cada turno, os mineradores que estão atualmente na mina (ou seja, em uma das *t* posições da mesma), se aptos, **coletam 1 minério**.

Se algum minerador **atingir sua capacidade máxima de armazenamento em um turno**, no final deste turno, o mesmo irá **retornar à base** para depositar os recursos coletados.

Quando o **minerador sai de uma posição na mina**, o primeiro minerador na fila de espera para àquela posição (se houver) deve substituí-lo.

Quando um minerador é alocado em uma posição da mina, o mesmo **não coletará no turno imediatamente após sua alocação (período de adaptação)**, **exceto após a inicialização** (após a inicialização os mineradores podem executar a coleta), passando a coletar normalmente nos turnos seguintes (1 minério por turno).

Após **abandonar uma posição na mina**, um minerador levará *d* turnos para ir à base e mais *d* turnos para retornar à mina. **Apenas após decorridos *d* turnos de sua saída de uma posição na mina é que os recursos coletados por um minerador poderão ser adicionados à reserva da base**. Ao retornar à mina, o minerador deve procurar novamente uma posição na mesma de acordo com sua chave, como na etapa de inicialização.

É possível que dada a saída de um minerador, a posição na mina anteriormente ocupada pelo mesmo fique **vazia**, tendo em vista que pode não haver mineradores na fila de espera. Mineradores que retornem à mina devem priorizar posições ainda não ocupadas, como descrito na **etapa de inicialização**.

Saída

A saída (fornecida em um **arquivo de texto**, ou **impressa na tela**) deve imprimir no início de cada conjunto de dados a frase “novo jogo comecou.”.

A primeira etapa do jogo (inicialização, ou turno 0) começa com a impressão da frase “etapa de inicializacao.”.

Na **etapa de inicialização**, todos os mineradores serão alocados a posições da mina, de acordo com o valor de suas chaves.

Quando um minerador é inserido em uma posição que estava vazia, a mensagem “minerador #chave inserido na mina #posicao_mina.” deve ser impressa.

Caso já exista algum minerador na posição em questão, a mensagem “minerador #chave_inserido colidiu com minerador #chave_ocupante na mina #posicao_mina.” deve ser impressa, e uma nova posição deve ser calculada para o minerador de acordo com o *Linear Probing*. Se a mina já estiver totalmente ocupada, o minerador é associado à fila de espera de acordo com sua chave e a *hash function*, sendo a mensagem “minerador #chave inserido na fila de espera da mina #posicao_mina.” impressa.

Após as inserções, deve ser impressa uma linha com a situação atual da mina. Para cada posição da mina, deve ser impresso a chave do minerador que a ocupa, seguida pelo número de minérios coletados até o momento pelo mesmo entre parênteses e um espaço em branco (exemplo: se o minerador 8 ocupa a posição 0 da mina e o mesmo já coletou 4 minérios, deve-se imprimir na avaliação da posição 0 da mina a saída “8(4) ”). Essa linha termina com um espaço em branco.

Cada turno começa com a impressão da mensagem “turno #turno comecou.”. Em seguida, os mineradores que estão aptos executam a coleta, sendo novamente impressa à situação atual da mina (como descrito no parágrafo anterior).

Caso algum minerador tenha atingido sua capacidade máxima, o mesmo deve retornar à base, sendo a mensagem “minerador #chave a caminho da base.” Impressa.

Por fim, a situação dos mineradores que se retiraram da base deve ser avaliada. Caso o minerador já tenha percorrido *d* turnos fora das minas, o mesmo depositará o que coletou à reserva da base, sendo a mensagem “minerador #chave depositou #capacidade minerio(s) na base.” impressa. O minerador levará os próximos *d* turnos para retornar à mina. Ao retornar à mina, a mensagem “minerador #chave retornou as minas.” deve ser impressa, assim como as mensagens relacionadas à inserção do mesmo em uma das posições na mina.

Caso haja mais de um minerador retornando à base ou à mina, a prioridade será resolvida de acordo com a ordem na qual os mesmos saíram/retornaram à mina (ou seja, o primeiro a sair/retornar à mina agirá primeiro).

Na última linha de cada turno deve ser impressa a seguinte mensagem “base: #total_coletado.”, informando a quantidade de recursos atual na base.

Na última linha de cada conjunto de dados deve ser impressa a mensagem “fim de jogo. #total_coletados minerios coletados.”.

Verifique os arquivos de entrada e saída exemplos (P02_in.txt e P02_out.txt, respectivamente).