UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA
NÚCLEO DE COMPUTAÇÃO - NUCOMP

ALGORITMOS E ESTRUTURAS DE DADOS I – 2018/1
PROF. FLÁVIO JOSÉ MENDES COELHO

PROJETO PRÁTICO 1 - versão 4

1 Objetivos

Este **Projeto Prático 1 – PP1**, tem o objetivo de exercitar e avaliar suas habilidades em:

- Codificar os tipos abstratos de dados LISTAS, PILHAS, FILAS e TABELA HASH na linguagem de programação exigida neste enunciado, e solucionar problemas empregando estas estruturas de dados.
- Desenvolver código de qualidade com boas práticas de programação: boa indentação, boa nomeação, refinamentos sucessivos, POO (exceto herança), programação genérica, etc.
- Explicar com segurança e lógica suas decisões ao escrever o código do projeto, respondendo de forma coerente às perguntas do professor.
- Aprender a lidar com desenvolvimento de software em equipe.

2 Descrição do problema

Guerra em Tau

Por décadas, avançados centros de astronomia e agências espaciais internacionais encetam frustradas tentativas na busca por vida extraterrestre inteligente. O que governos e agências terrestres jamais poderiam supor é que neste momento, há cerca de 65 anos-luz, oculto sob a luz de Aldebaran na constelação do Touro, flutua, parcialmente invisível, o palco de um luta épica entre três grandes raças: os Azuri, os Ianteco, e os Umashi. Cada uma das raças visa a supremacia sobre os recursos naturais do pequeno planeta Tau. Os Azuri são a raça nativa de Tau. Os Ianteco formam uma raça alienígena que invadiu Tau em busca de um novo lar por haverem esgotado todos os recursos de seu planeta natal. Os Umashi são uma raça mutante criada pelos Ianteco por meio do cruzamento genético de azurianos capturados e de renegados do próprio povo Ianteco, com o fim de aumentar sua força bélica contra os Azuri. Porém, os Umashi se revoltaram e passaram a lutar por conta própria contra as outras duas raças.

Os Azuri são um povo pacífico, e lutam somente com o fim de expulsar as raças invasoras de Tau. Por esta razão, criaram "O Santuário", uma grande nave interestelar capaz de armazenar milhões de almas de seus inimigos, para fazê-las renascer em corpos semi-sintéticos. O povo Azuri visava derrotar os Ianteco e os Umashi, depositando suas almas no Santuário para serem enviados de volta ao seu planeta de origem, levando todos os recursos necessários para a reconstrução natural daquele orbe. Porém, a terrível guerra corrompeu o Santuário que passou a ser empregado pelas três raças como um meio de consultar a sabedoria das almas de seus soldados mortos em batalha.

A Real Academia de Ciências Computacionais dos Azuri convocou você e sua equipe para desenvolver um simulador de batalha visando estudar melhores táticas de guerra, e assim, por um fim à terrível hacatombe que devasta a pequena jóia da constelação do Touro: o planeta Tau. A partir de hoje, a sobrevicência das três raças e o fim desta guerra sangrenta depende de você!

3 Elementos do simulador

Os seguintes elementos fazem parte do simulador a ser construído:

1. **Unidade tática**. É um soldado ou uma tropa.
2. **Soldado**. É um personagem que pertence a uma das três raças. Possui um nome, uma identificação da sua raça, e um número inteiro representando sua força de combate. Um soldado também mantém sob seu poder um grupo de zero ou mais soldados inimigos capturados. Este grupo é organizado sob a forma de uma sequência de soldados. Um soldado capturado sempre entrará no fim desta sequência. A **força de um soldado** é a soma das forças de combate de seus capturados. Um soldado sem capturados inicia com uma força aleatória fornecida na entrada de dados.
3. **Tropa**. É um empilhamento de soldados de uma mesma raça. O soldado no topo de um tropa é o líder da tropa e possui o maior nível de força dentro da tropa. Logo abaixo do soldado líder, encontra-se o soldado com o segundo maior nível de força, seguido do soldado com o terceiro maior nível de força, e assim por diante, até o soldado com menor força na base da tropa. Uma tropa sempre se manterá em ordem, do soldado mais forte no topo até o mais fraco na base. A **força de uma tropa** é a soma das forças de seus soldados. Na entrada de dados, uma tropa será formada quando dois ou mais soldados fornecidos na entrada estiverem posicionados sobre um mesmo território.
4. **Mapa de batalha**. É um "tabuleiro" com dimensões $M \times N$ ($M = 10, N = 10$), contendo 100 quadrados chamados de **Territórios**, cada território representando uma posição (i, j) no mapa de batalha, onde $0 \leq i \leq M - 1$ e $0 \leq j \leq N - 1$. Um território pode estar: (a) desocupado; (b) ocupado por um único soldado; (c) ocupado por uma única tropa. Uma unidade tática movimenta-se pelo mapa de batalha horizontal ou verticalmente (nunca em diagonais), movendo-se três territórios a cada passo, como mostra a Figura 3, abaixo. Nesta Figura, os terrítorios marcados com "x" formam o **alcance de movimentos** da unidade tática `G4`. Se uma unidade tática A ocupa um território e outra unidade B está sobre o alcance de A, diz-se que B é alcançável por A (por simetria, a unidade A também é alcaçavel por B).

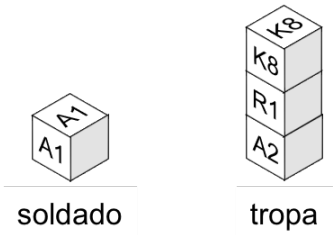

Figura 1: Unidades táticas

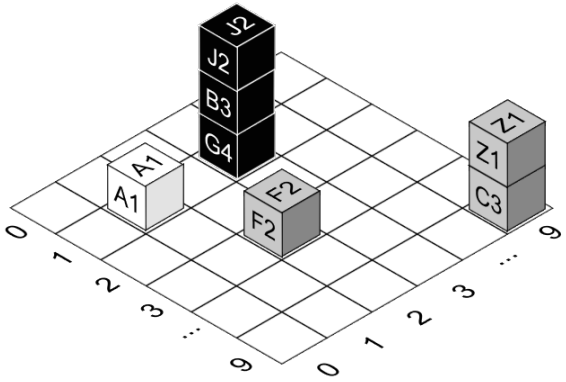

Figura 2: Mapa de batalha com unidades táticas das diferentes raças

5. **Santuário**. Contém uma tabela hash para armazenar as almas dos soldados mortos em batalha (à frente, você verá como ocorre a morte de um soldado). Para consultar a sabedoria das almas dos combatentes no santuário durante um ataque, um soldado fornece como parâmetro de consulta uma string concatenando seu nome e sua raça (`nome+raça`) convertidos em minúsculo (caixa baixa). Para converter a string `nome+raça` para inteiro, utilize o método explicado na aula sobre Tabela Hash, utilizando como base o valor 31. O santuário retornará a força das almas dos combatentes cujos `nome+raça` colidam com o `nome+raça` do soldado consultante. Porém, este soldado poderá utilizar somente 70% da força obtida na consulta ao santuário, e deverá descartar esta força após uma batalha.

4 Entradas

As entradas para o simulador seguirão o seguinte padrão:

1. Será dado um mapa de batalha inicial com uma parte dos territórios ocupada por unidades táticas de duas ou mais raças. Cada território poderá ser ocupado por uma unidade tática de uma única raça. Cada unidade tática ocupando um território será fornecida no seguinte formato: `nome raça` i j `força`, onde `nome` é o nome do soldado, `raça` é o nome de sua raça, i e j são as suas coordenadas no território ocupado, e `força` é o valor inicial de sua força. A entrada da unidade tática, possivelmente, será seguida da sequência de seus capturados, de acordo com o seguinte padrão: `nome raça força`. Com exceção da

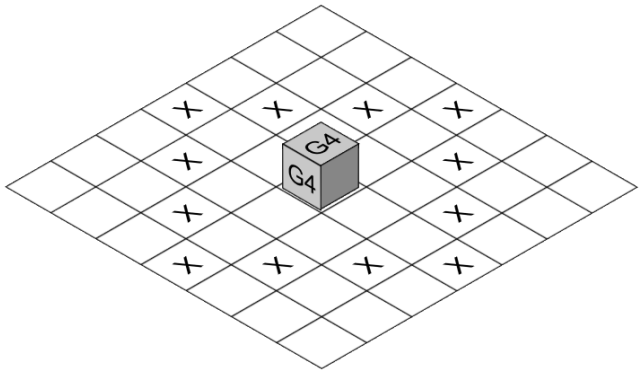

Figura 3: Movimentos de uma unidade tática

primeira unidade tática, cada unidade tática fornecida durante a entrada do tabuleiro será precedida pelo comando de entrada "uta" (que significa "unidade tática").

2. Em seguida, será fornecido o santuário com a lista das almas dos soldados, no seguinte formato: `nome raça força`, onde `nome` é o nome do soldado, `raça` é o nome de sua raça e `força` é o valor de sua força (suponha que este foi o valor de sua força, antes de morrer em batalha). Antes de começar a entrada do santuário você lerá o comando de entrada "sto" (que significa "santuário").

3. Finalizando, serão fornecidos um ou mais comandos de batalha, um comando por linha. Os comandos são: `mova`, `forca_mapa`, `forca`, `terr`, `max_ataque` e `fim`. Antes de começar a entrada dos comandos de batalha você lerá o comando de entrada "cmd" (que significa "comandos").

Veja o seguinte exemplo completo de entrada:

```
Z1 azuri 2 3 2 A2 umashi 11 B4 umashi 3
uta
A5 azuri 0 8 3 D1 ianteco 4
uta
G1 umashi 0 11 1 D2 azuri 1 H7 ianteco 2 F5 ianteco 8
uta
T9 ianteco 3 3 4 H9 umashi 5 K6 azuri 12
uta
S3 azuri 0 8 4 B7 ianteco 2 F7 ianteco 9 A3 umashi 12
uta
T8 ianteco 3 3 2 H8 azuri 2 I1 umashi 19 C13 azuri 5
uta
X9 umashi 0 1 4
sto
B9 azuri 13
A12 umashi 8
B7 ianteco 4
N2 ianteco 5
```

```
T6 umashi 6
Q3 azuri 21
A10 umashi 13
F12 azuri 12
S4 azuri 8
R1 ianteco 3
W7 azuri 4
W2 umashi 2
cmd
forca 3 3
forca 0 8
mova 0 8 3 3
forca 3 3
forca 0 8
max_ataque 2 3
terr 2 3
fim
```

5 Processamento de comandos e saídas

Cada comando é processado como descrito abaixo:

`mova` i_1 j_1 i_2 j_2: ordena o movimento de uma unidade tática localizada no território de posição inicial (i_1, j_1) para o território de posição final (i_2, j_2), caso este seja um movimento válido. Um movimento é válido obedece às regras de movimentação de uma unidade tática explicadas na Figura 3, e não ultrapassa os limites do mapa de batalha. Em um movimento podem acontecer quatro situações:

(a) **Formação de tropa (ou fortalecimento de tropa)**: se as posições inicial e final estão ocupadas por unidades táticas de mesma raça e o movimento for valido, as unidades se unem formando uma tropa, na posição final, que manterá sua propriedade estrutural (ordenada do topo para baixo com o soldado mais forte e líder no topo e o mais fracao no fundo).

(b) **Ataque**: se as raças das unidades táticas nas posições inicial e final diferem, ocorre um ataque da unidade na posição inicial sobre a unidade inimiga na posição final. Se a unidade atacante possui força maior que a unidade atacada pode acontecer uma de duas situações:

(1) Se o líder atacado possui capturados, ele entrega seu capturado mais antigo e em troca não é capturado. O capturado entregue não volta ao combate; ele morre e é enviado para o santuário onde passa a existir como alma. Sua posição no santuário é definida em função de seu nome e raça (*string* `nome+raça`). Ao entregar um capturado, o líder não perde a força que já tinha (a força do capturado entregue não é subtraído da força do líder);

(2) Se o líder atacado não possui nenhum capturado para entregar, ele é capturado pelo líder atacante e entra no fim da sequência de capturados do líder. Como consequência, o líder atacante acrescenta à sua força a força do soldado capturado.

Neste caso, o simulador imprime "Ataque bem sucedido". Se o ataque resultar na aniquilação da unidade inimiga, a unidade atacante passa a ocupar o território desta unidade aniquilada e imprime a mensagem "unidade `nome+raça` avança para (i_2, j_2)".

Por outro lado, se a unidade atacante tiver força menor ou igual a 2/3 da força inimiga, o líder atacante consulta o santuário para obter a força da sabedoria de antigos soldados mortos em batalha. A consulta acrescenta, momentaneamente, à força de ataque da unidade atacante 70% da força de todos as almas dos soldados do santuário cuja *string* `nome+raça` colidam com a *string* `nome+raça` do líder atacante. Após a consulta ao santuário, se a nova força da unidade atacante superar a força da unidade atacada, ocorre um ataque como descrito no início do item (b). Caso contrário, ocorre um contra-ataque, e os papéis se invertem: a unidade que seria atacada passa a ser atacante, e a unidade inicialmente atacante passa a ser a unidade atacada. Ocorre um ataque como descrito no início do item (b). Neste caso, o simulador imprime "Contra-ataque!".

Ainda, se em um ataque houver empate de forças (mesmo após uma consulta ao santuário), então o ataque é cancelado e ambas as unidades permanecem em seus lugares.

(c) **Deslocamento**: há uma unidade tática na posição inicial, mas não há outra unidade na posição final. Nesse caso, a unidade apenas passa a ocupar a posição final, deixando sua posição anterior desocupada, caso o movimento seja válido. Nenhuma saída é gerada.

(d) **Nada**: isto ocorre se: (a) não houver unidades em nenhuma das posições fornecidas; (b) o movimento não for válido, ou seja, a unidade não puder executar tal movimento.

`forca_raca raça`: comando que solicita impressão da soma das forças da raça `raça` espalhadas pelos mapa de batalha.

`forca` i j: comando que solicita a impressão da força da unidade tática localizada no territótio de posicao (i, j). Caso não exista um unidade neste território, a simulador deverá imprimir o valor -1.

`terr` i j: comando que solicita a impressão do que está no território de posição (i, j). Se houver um soldado na posição, imprimir `[nome raça força]`, onde `nome`, `raça` e `força` são seu nome, raça e força, respectivamente. Se houver uma tropa, imprimir `[[nome raça força]]`, onde o `nome`, `raça` e `força` são o nome do soldado líder da tropa, sua raça e a força da tropa, respectivamente. Caso não exista um unidade neste território, a simulador deverá imprimir o valor "`[ ]`".

`max_ataque` i j: comando para imprimir a força da unidade tática localizada no territótio de posicao (i, j) após *simular* o melhor ataque contra seus inimigos alcançáveis, ao seu redor. Caso não exista uma unidade neste território, ou não existam inimigos alcançáveis

pela unidade, o simulador deverá imprimir o valor -1. Este comando, não efetiva um ataque, mas descobre qual movimento (ou movimentos) *resultaria* no melhor ataque, mostrando a força resultante deste ataque.

`fim`: finaliza a simulação.

Você deve imprimir as respostas a todas os comandos de batalha uma linha por vez, conforme explicado acima. Se um comando `mova` resultar na eliminação de uma raça [1], imprima o nome da raça e a mensagem " foi eliminada". Se um comando `mova` resultar na eliminação de todas raças exceto uma, imprima o nome desta raça seguido da mensagem "... tem supremacia sobre Tau!".

6 Requisitos do projeto

1. **Equipes**. Este projeto deve ser desenvolvido por uma equipe de **dois** estudantes. Não serão aceitas equipes com número menor ou maior de participantes, a não ser que isso seja permitido pelo professor, e discutido/decidido antes do início do projeto. Qualquer equipe com um número diferente de dois participantes (sem a justificativa explicada acima), terá seu projeto valendo no máximo 80% da pontuação total obtida.

2. **Ferramentas e técnicas**. O projeto deve ser codificado em C++. Será permitido programação procedural, mas todos as estruturas de dados (TADS) principais deverão ser programadas orientadas a objeto, generalizadas (templates) e encapsuladas.

3. **Padrões de codificação**. Siga o *CamelCase* do Java como padrão de nomeação de identificadores. A indentação e posicionamento de chaves deve seguir o padrão K&R. variante de Stroustrup (`en.wikipedia.org/wiki/Indent_style#K.26R_style`). Veja as aulas iniciais de AED 1 sobre boas práticas de nomeação, etc.

4. **Compilador**. Indica-se o uso do compilador *GCC - the GNU Compiler Collection* (http://gcc.gnu.org), ou sua variante *Mingw* para para Microsoft Windows, ou ainda o compilador *clang* (https://clang.llvm.org). **Sempre teste seu projeto em vários compiladores online!** Às vezes, o projeto pode funcionar bem na sua máquina, mas pode apresentar erros em outros compiladores.

5. **Submissão**. Todo o projeto deverá ser submetido ao juiz online (a ser informado pelo professor) em um único arquivo fonte com extensão `cpp`.

6. **Bibliotecas e funções**. Sua equipe não deve utilizar nenhuma estrutura de dados pronta (listas, pilhas, filas, vectors, etc) da *Standard Template Library* - STL, ou de qualquer outra biblioteca C++. É suficiente o uso de `iostream` e `cstdlib` (para uso de qualquer outro arquivo de cabeçalho, fale com o professor). O uso de estruturas de dados prontas ou funções de bibliotecas conforme explicado acima, implicará na atribuição da nota mínima ao projeto.

[1] A eliminação de uma raça significa, na verdade, que todos seus soldados sobrevivem, agora, como almas no sanutário.

7 Pontuação

O **PP1** vale no mínimo 0.0 e no máximo 10.0 (ou no máximo 8,0, no caso explicado na seção 6, item 1). As notas são distribuídas em duas partes:

(1) A **avaliação funcional** avalia o quanto foi implementado do que foi pedido, e se as saídas corretas são obtidas. Esta parte vale de 0.0 à 5.0 pontos. O código do projeto será submetido a um juiz online com N casos de testes (N será informado pelo professor). Cada caso de teste vale $5,0/N$.

(2) **Inspeção de código**. Esta parte vale de 0.0 à 5.0 pontos, e avalia os critérios especificados à seguir:

   (a) Segurança do aluno nas respostas às perguntas do professor sobre o código desenvolvido. Uso adequado de linguagem técnica nas explicações: nomes corretos de estruturas de dados, seus componentes e algoritmos, de elementos da linguagem de programação, e das técnicas de programação utilizadas.

   (b) Indentação de código correta, utilização de padrão de nomeação, nomes adequados para identificadores, etc.

   (c) Implementação correta/coerente dos tipos abstratos de dados (seguindo o que foi apresentado nas aulas).

   (d) Implementação orientada a objetos (com encapsulamento) das estruturas de dados.

   (e) Código genérico (templates) onde for aplicável e útil.

   (f) Codificação racional e eficiente do código, e criatividade.

Importante: (1) Para a 3a. chamada não haverá defesa. O projeto valerá no máximo 5,0 pontos e poderá sofrer penalizações de 0,5 ponto para cada critério acima (de b à f) não atendido. (2) Se o projeto for implementado em outra linguagem de programação, ganhará integralmente a nota mínima.

8 Datas

- Emissão deste enunciado: 23/04/2018.
- Abertura do juiz online: 17/05/2018 (3a. e última chamada).
- Fechamento do juiz online: 21/05/2018 às 22h00min (hora local).

9 Contribuições

Agradeço aos alunos Francisco Elio Parente Arcos Filho (GAPA), Rodrigo da Costa Moraes (GAPA), Vitor Matheus de Souza Carvalho (GAPA, monitor) e Levi da Silva Lima (GAPA, monitor) que criaram os elementos e regras do jogo (ou simulador como chamei) aplicado neste

Projeto Prático. Eu contribui criando a ideia do santuário, e adaptei o jogo ao universo do planeta Tau e sua guerra entre as raças (e impus algumas limitações às ideias malévolas desse quarteto malígno! Caso contrário, o jogo/simulador ficaria bem mais difícil!).

CÓDIGO DE ÉTICA

Este projeto é uma avaliação acadêmica e deve ser concebido, projetado, codificado e testado pela equipe, com base nas referências fornecidas neste enunciado ou nas aulas de Algoritmos e Estruturas de Dados, ou por outras referências indicadas pelo professor, ou com base em orientações do professor para com a equipe, por solicitação desta. Portanto, não copie código pronto da Internet para aplicá-lo diretamente a este projeto, não copie código de outras equipes, não forneça seu código para outras equipes, nem permita que terceiros produzam este projeto em seu lugar. Isto fere o código de ética desta disciplina e implica na atribuição da nota mínima ao trabalho.

Referências

[1] COELHO, Flávio. Slides das aulas de *Algoritmos e Esruturas de Dados I*. Disponível em https://sites.google.com/a/uea.edu.br/fcoelho/. Universidade do Estado do Amazonas, Escola Superior de Tecnologia, Núcleo de Computação - NUCOMP. Semestre letivo 2018/1.

[2] C++. In: *cppreference.com*, 2016. Disponível em <http://en.cppreference.com/w/>. Acesso em: 17 abr. 2016.

[3] CORMEN, T. H., Leiserson, C. E., Rivest, R. L., Stein C. *Introduction to Algorithms*, 3rd edition, MIT Press, 2010

[4] KNUTH, Donal E. *Fundamental Algorithms*, 3rd.ed., (vol. 1 de The Art of Computer Programming), Addison-Wesley, 1997.

[5] KNUTH, Donal E. *Seminumerical Algorithms*, 3rd.ed., (vol. 2 de The Art of Computer Programming), Addison-Wesley, 1997.

[6] KNUTH, Donal E. *Sorting and Searching*, 2nd.ed., (vol. 3 de The Art of Computer Programming), Addison-Wesley, 1998.

[7] STROUSTRUP, Bjarne. *The C++ Programming Language*. 4th. Edition, Addison-Wesley, 2013.

[8] STROUSTRUP, Bjarne. *A Tour of C++*. Addison-Wesley, 2014.

[9] SZWARCFITER, Jayme Luiz et. alii. *Estruturas de Dados e seus Algoritmos*. Rio de Janeiro. 2a. Ed. LTC, 1994.

[10] WIRTH, Niklaus. *Algoritmos e Estruturas de Dados*. Rio de Janeiro. 1a. Ed. Prentice - Hall do Brasil Ltda., 1989.

[11] ZIVIANI, Nívio. *Projeto de Algoritmos com Implementação em Java e C++*. 2a. Edição. Cengage Learning, 2010.

[12] ZIVIANI, Nívio. *Projeto de Algoritmos com Implementação em Pascal e C*. 3a. Ed. São Paulo: Cengage Learning, 2012.