Trabalho 6 – Fila de prioridade usando heap para simulação de atendimento

A estrutura de dados Heap e seus algoritmos são úteis para realizar ordenação de dados, como visto no algoritmo Heapsort, e também para implementar filas de prioridade eficientes. Uma das aplicações de filas de prioridade é a alocação de recursos, como no gerenciamento de memória e tempo de processamento realizado pelo sistema operacional. No entanto é possível aplicar o mesmo conceito a problemas não computacionais.

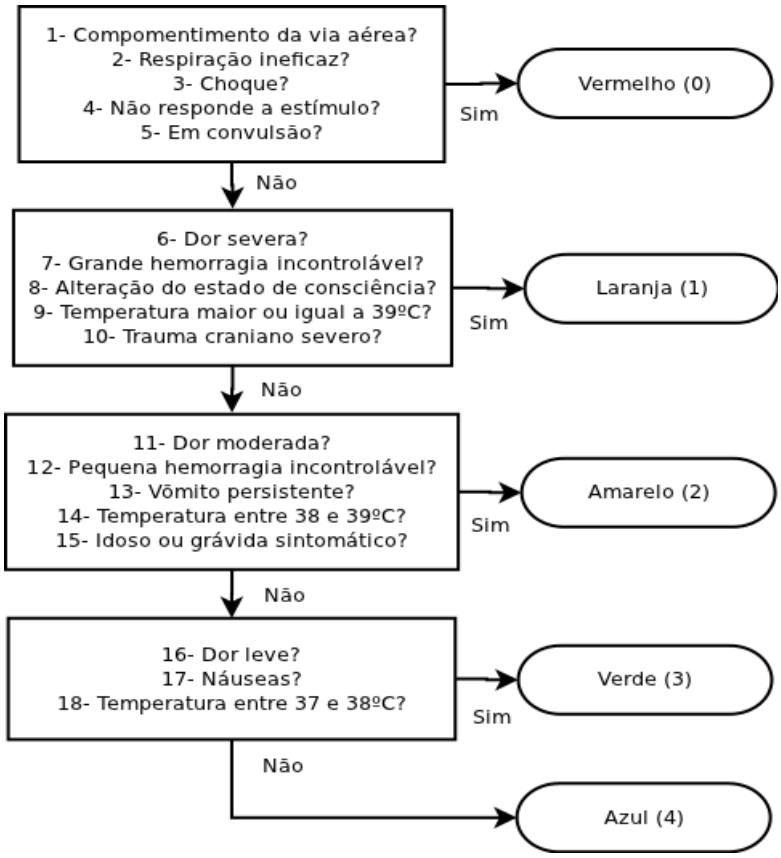
Leia a descrição do trabalho **com atenção e várias vezes**, anotando os pontos principais e as possíveis formas de resolver o problema. Comece a trabalhar o quanto antes para que as dúvidas sejam sanadas a tempo de você conseguir finalizar o trabalho a tempo.

Introdução

Numa unidade médica de urgência e emergência hospitalar, cada paciente a ser atendido recebe uma classificação de risco para definir a prioridade do atendimento. A prioridade é definida pelas cores vermelho (0), laranja (1), amarelo (2), verde (3), e azul (4), cada qual com um determinado tempo de espera tolerável, com os seguintes significados:

- Vermelho: quadro clínico implica em risco de morte, e que o caso deve ser rapidamente encaminhado para a sala de emergência. Atendimento imediato.
- Laranja e amarelo: o paciente não tem risco iminente de morte, mas o atendimento é prioritário, pois o tempo de espera pode aumentar a gravidade do caso. Atendimento com prioridade – laranja: até 10 minutos, amarelo até 20 minutos.
- Verde: não há risco de morte e o paciente deverá ser atendido após os casos vermelhos, laranjas e amarelos. Atendimento com até 60 minutos.
- Azul: quadros crônicos, sem sofrimento agudo. Neste caso, o paciente será encaminhado ao centro de saúde. Atendimento em mais de 60 minutos.

O protocolo para encontrar a prioridade é definido pelo esquema abaixo:



Tarefa

Implementar um programa que controle: i) a fila de pacientes a serem atendidos e ii) os pacientes em atendimento por médicos. Como entrada são oferecidos dados simulados do setor de urgência e emergência de um hospital, que precisa ir alocando as pessoas que vão chegando aos médicos de plantão. Para cada pessoa que chega uma prioridade será definida, e essa pessoa é alocada a um médico diferente. Cada médico demora um tempo para atender cada paciente, e por meio desse tempo o programa irá calcular quantos minutos um novo paciente demorará para ser atendido.

A partir do tempo de espera tolerável de cada cor (descrito na introdução) é possível saber se há uma demanda maior do que a capacidade do hospital. Os tempos médios de consulta a serem considerados pelo sistema para os pacientes são, por cor:

- 0 - Vermelho: 50 minutos
- 1 - Laranja: 20 minutos
- 2 - Amarelo: 15 minutos
- 3 - Verde: 8 minutos
- 4 - Azul: 5 minutos

Iniciamente será definido pelo teclado o número *M* de médicos em plantão, seguido pelos registros de cada médico, no formato:

```
<M>
<Registro1> <Registro2> ... <RegistroM>
```

Considere que o registro é um número inteiro entre 1 e 999999.

A seguir, cada grupo de pacientes a entrar na fila de atendimento será incluido por meio de um arquivo cujo nome será fornecido como entrada, por exemplo:

```
arq_pacientes_001.txt
```

O arquivo está em formato texto e possui a seguinte configuração:

```
<T>
<N>
<Sobrenome_1> <1> <2> <3> <4> <5> <6> <7> <8> <9> <10> <11> <12> <13> <14> <15>
<16> <17> <18>
...
<Sobrenome_N> <1> <2> <3> <4> <5> <6> <7> <8> <9> <10> <11> <12> <13> <14> <15>
<16> <17> <18>
```

Onde <T> é um número real positivo, representando o tempo passado até o momento da chegada dos pacientes, <N> é o número de pacientes a chegar para atendimento naquele instante, <Sobrenome> é o sobrenome (assuma uma única palavra, sem separação por espaço) do paciente. E os números de <1> a <18> São respostas S (Sim) ou N (Não) a cada uma das 18 perguntas do protocolo de triagem especificado anteriormente.

O arquivo texto poderá ter um ou mais grupos de pacientes conforme exemplificado anteriormente.

O programa deverá inicialmente verificar se há pacientes em atendimento e se houver calcular quantos já foram atendidos de acordo com o tempo <T> passado. Retirar do atendimento os pacientes atendidos, gerando como saída o sobrenome do paciente, o registro do médico e o tempo

de espera, caso haja pacientes que terminaram de ser atendidos, no formato: **S <Num_Registro> <Sobrenome> <Tempo_Espera>/n**

Para cada grupo de pacientes, calcular a prioridade e inserir esse paciente na fila. Após dar entrada em todos os pacientes, colocar em atendimento os pacientes da fila em ordem de prioridade (da menor para a maior). O critério de desempate será a ordem de chegada. A chamada para atendimento só ocorre no tempo <T> informado.

Para cada médico disponível alocar um paciente na fila, alocar os pacientes em ordem de prioridade, usando como ordem o registro profissional (alocar primeiro a médicos com número de registro menor).

- Caso *não* haja médicos disponíveis para atender a todos, o programa deverá percorrer a fila de espera, calcular o tempo de espera estimado os pacientes na espera (naquele instante), em minutos, e gerar esse valor como saída. Esse tempo de espera estimado será o menor tempo entre os pacientes sendo atendidos somado aos tempos de atendimento previstos (de acordo com a prioridade) para os pacientes que estão na frente na fila. A ordem de impressão é a ordem da fila, começando daquele que deverá ser atendido primeiro.
- O formato da saída do tempo em minutos é: **E <Sobrenome> <Tempo>\n**

Ao final, quando não houver mais pacientes a chegar, o programa deverá simular o atendimento até que a fila seja vazia. A partir desse ponto não é necessário exibir o tempo de espera estimado, apenas os tempos de espera reais conforme descrito anteriormente.

Exemplo

Entrada:

```
2
101 102
arq_pacientes_001.txt
```

Conteúdo do arquivo texto:

```
0.0
3
Valle N N N N N N N N N N N N N S S N
Silva N S N N N S N S N N N N N S N N N
Sousa N N N N N N N N N S N N S S N N N
15.0
1
Costa N N N N N N N N S N N N N N N N
22.5
2
Gomes N N N N N N N N N N N N N N N S
Lima N N N N N N N N N N N N N N N N
```

Saída:

```
E Valle 15.0
S 102 Sousa 0.0
E Valle 20.0
```

```
S 102 Costa 0.0
E Gomes 8.0
E Lima 16.0
S 102 Valle 37.5
S 101 Silva 0.0
S 102 Gomes 8.0
S 101 Lima 12.5
```

No exemplo acima, chegam 3 pacientes inicialmente, aqueles com maior prioridade foram atendidos diretamente e 1 ficou na espera, que nesse instante estava estimada em 15 minutos dado que um dos médicos poderia ficar livre nesse tempo após atender o paciente prioridade 2.
Após 15 minutos um novo paciente chegou. Nesse tempo, um dos pacientes terminou de ser atendido tendo esperado 0 minutos. O paciente que já estava esperando teve seu tempo de espera estimado em mais 20 minutos dado que chegou um paciente com maior prioridade.

22,5 minutos depois, dois novos novos pacientes chegam para atendimento e o tempo estimado de espera foi de 8 minutos para um e 16 para o outro.

Ao terminarem as entradas, todos pacientes devem ser atendidos na ordem e retirados, um a um, com seus tempos de espera reais exibidos.

Será fornecido um caso de teste pronto. Você deve montar seus próprios casos de teste adicionais.

Informações importantes (LEIA COM ATENÇÃO)

- Sobre a avaliação

    1. Um dos objetivos da disciplina de ICC2 é o **aprendizado individual** dos conceitos de programação. A principal evidência desse aprendizado está nos trabalhos, que são individuais neste curso. Você deverá desenvolver seu trabalho sem copiar trechos de código de outros alunos, nem codificar em conjunto. Portanto, compartilhem idéias, soluções, modos de resolver o problema, mas não o código.
        - O plágio vai contra o código de ética da USP.
        - Quando autores e copiadores combinam, estão ludibriando o sistema de avaliação.
        - O trabalho em grupo e a cooperação entre colegas é em geral benéfico e útil ao aprendizado. Para ajudar um colega você pode lhe explicar estratégias e idéiar. Por exemplo, pode explicar que é preciso usar dois loops para processar os dados, ou que para poupar memória basta usar uma certa estrutura de dados, etc. O que você **não** deve fazer é mostrar o seu código. Mostrar/compartilhar o código pode prejudicar o aprendizado do seu colega:
            - depois de o seu colega ter visto o seu código, será muito mais difícil para ele imaginar uma solução original e própria;
            - o seu colega não entenderá realmente o problema: a compreensão passa pela prática da codificação e não pela imitação/cópia.
        - Um colega que tenha visto a sua solução pode eventualmente divulgá-la a outros colegas, deixando você numa situação muito complicada, por tabela.
        - O texto acima foi baseado e adaptado da página http://www.ime.usp.br/~mac2166/plagio/, da qual recomendo a leitura completa.

    2. Todos os códigos fontes serão comparados por um (ou mais) sistema(s) de detecção de plágio, e **os trabalhos com alta similaridade detectada terão suas notas zeradas**, tanto aqueles relativos ao código de origem quanto do código copiado.

3. A avaliação incluirá a porcentagem de acertos verificada pelo SQTPM e também a análise do seu código, incluindo endentação, comentários, bom uso da memória e práticas de programação. Portanto faça seu código com cuidado, da melhor forma possível.

- Sobre o sistema de submissão (SQTPM):

1. Seu código deverá ser submetido num arquivo fonte .c. Esse arquivo deverá **obrigatoriamente** conter no início um comentário com seu nome, número USP, turma e data da entrega do trabalho.

2. **A data/hora de entrega do trabalho é aquela estipulada no sistema SQTPM. Trabalhos entregues por email NÃO serão aceitos, mesmo que dentro da data/hora estipulada. Faça seu trabalho com antecedência para evitar entregar em cima da hora e ter problemas de submissão, pois o sistema tende a ficar lento com as múltiplas submissões feitas geralmente próximas ao fechamento do sistema.**
    - A submissão é de responsabilidade do aluno, e os problemas comuns à entrega próxima ao fechamento do sistema também. Portanto: problemas de acesso à rede e ao servidor SQTPM **não** serão aceitos como desculpa para entrega por email ou fora do prazo.

3. A compilação do código é feita no sistema pelo comando:
```
gcc -o prog proc.c -lm -Wall
```

4. A saída do seu programa deve ser exatamente igual à saída esperada, incluindo: espaços em branco, quebras de linha e precisão decimal.

5. Há um limite de 10 segundos para a execução dos casos de teste e um limite de 16.5MB de memória total para ser utilizada. Você deverá gerenciar bem o tempo de execução e o espaço ocupado para que seu programa fique dentro desses limites, para evitar uso excessivo, pois o sistema irá invalidar o caso em que o limite foi excedido.

6. Ao enviar, aguarde um pouco mais de 20 segundos, **sem recarregar a página nem pressionar ESC**, para obter a resposta. Caso demore mais do que 1 minuto para dar uma resposta, feche e abra novamente a página do servidor. Verifique se seu programa tem algum problema de entrada de dados (ou se tem algum loop infinito ou parada para pressionamento de tecla). Caso não tenha, aguarde alguns minutos e tente novamente.

7. O erro de "Violação de Memória" significa acesso indevido a arranjo ou arquivo. Use compilação com **-g** e o programa valgrind para tentar detectar a linha de código com erro.
    - Exemplo 1 de violação de memória:
```
char *array = (char *)malloc(sizeof(char) * N);
scanf("%s", &filename); // acesso indevido, violacao de memoria
free(array);
```

    - Exemplo 2 de violação de memória:
```
int **mat = (int **)malloc(sizeof(int *) * 3);
mat[0] = (int *)malloc(sizeof(int)*10);
for (i = 1; i < 3; i++) {
      // apenas a posicao 0 de mat foi alocada as outras nao
      // portanto esta liberando regiao nao alocada
      // gerando violacao de memoria
      free(mat[i]);
}
```

    - Exemplo 3 de violação de memória:
```
int A[N];
int B[N];
int j = 0, i = 5;
while (j < N){
```

```
        A[i] = B[j]; // como j e i sao indices diferentes
        j++;         // quando j = (N-1) i = (N-1)+5 e
        i++;         // havera escrita indevida em A
}
```