BCC202 – Estruturas de Dados I (2013-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Reinaldo Fortes** (`www.decom.ufop.br/reinaldo`)
Estagiária docente: **Josiane Rezende**
Monitores: **Bruno H. M. dos Santos**

Trabalho Prático 01 (TP01) – Tipos Abstratos de Dados - TADs

- **Data de entrega: 08/11/2013 até 23:55. O que vale é o horário do Moodle, e não do *seu*, ou do *meu*, relógio!!!**
- **Decréscimo por atraso até:** Sábado: 12:00 = **30%**, 23:55 = **40%**; Até Domingo: 12:00 = **60%** 23:55 = **70%**.
- O padrão de entrada e saída deve ser respeitado **exatamente** como determinado no enunciado.
- **Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos**.
- ***Bom trabalho!***

1 Objetivos

Este trabalho prático tem como objetivo principal fundamentar os conceitos de Tipos Abstratos de Dados (TADs) e aplicar noções básicas de análise de complexidade de algoritmos. Como objetivos secundários pretende-se prover uma revisão de conceitos básicos de programação e a alocação dinâmica de memória.

1.1 Descrição do Problema

Com o crescimento do DECOM, os professores tiveram considerável aumento de encargos didáticos e o crescimento de suas tarefas. Em uma assembléia do departamento foi estabelecida a criação de uma aplicação que auxilie no gerenciamento do tempo dos professores. Considere que você foi o aluno selecionado para executar esta importante tarefa, e, para isso, deverá respeitar a especificação a seguir.

1.1.1 Tipos Abstratos de Dados

O sistema deverá comportar dois TADs principais, definidos a seguir.

Tarefa: A agenda de um professor é composta de tarefas simples, definidas basicamente pelo seu tempo de **chegada**, tempo de **duração** e sua **penalidade por tempo de espera**, todos valores inteiros *não negativos*. Uma tarefa tem como operações:

- `cria`: cria uma tarefa, recebendo como entrada seus respectivos dados.
- `inicia`: determina o momento em que a tarefa começou a ser executada.
- `termina`: determina o momento em que a tarefa foi encerrada.
- `espera`: retorna o tempo de espera da tarefa, aplicando a penalidade por tempo de espera: $E_i * P_i$, onde E_i é o tempo de espera da tarefa i e P_i é a penalidade por espera da tarefa i.

Agenda: Conjunto de Tarefas de um professor, contendo as seguintes operações:

- `cria`: cria uma agenda de comprissos, aloca a memória necessária para armazenar todas as tarefas.
- `insere`: insere uma nova tarefa na agenda.
- `processa`: processa a agenda, realizando o escalonamento das tarefas (seção 1.1.2) e gerando a saída para a agenda em questão (seção 1.2).
- `esperaTotal`: retorna o *tempo de espera total*: $\sum_{i=1}^{n} E_i * P_i$, onde n é o número de tarefas, E_i é o tempo de espera da tarefa i e P_i é a penalidade por espera da tarefa i.

1.1.2 Sistema de Escalonamento de Compromissos

O sistema deverá escalonar as tarefas respeitando as seguintes restrições:

- Uma tarefa não pode iniciar antes de sua chegada.
- Uma tarefa iniciada não pode ser interrompida.
- Uma nova tarefa pode ser iniciada no tempo imediatamente seguinte ao tempo de término da tarefa anterior, caso todas as demais restrições sejam satisfeitas.

1.2 Exemplo de Entrada e Saída

A **entrada** começa definindo o número de agendas que serão simuladas na execução da aplicação. Nas linhas seguintes são definidos dados de cada agenda, separados por uma linha em branco. Cada agenda é definida da seguinte maneira: na primeira linha é definido o número de tarefas da agenda, em cada linha seguinte são definidas as tarefas, através de três valores separados por espaço: **chegada**, **duração** e **penalidade por espera**.

A **saída** deve apresentar a linha de tempo das tarefas, seguida pelo *tempo de espera total* (não ultrapassando `99999` unidades de tempo), conforme exemplo a seguir.

Entrada

```
1

4
0 5 2
0 2 3
12 4 3
3 3 2
```

Saida

```
tempo | T001 | T002 | T003 | T004
    0 | #### | ---- |      |
    1 | #### | ---- |      |
    2 | #### | ---- |      |
    3 | #### | ---- |      | ----
    4 | #### | ---- |      | ----
    5 |      | ---- |      | ####
    6 |      | ---- |      | ####
    7 |      | ---- |      | ####
    8 |      | #### |      |
    9 |      | #### |      |
   10 |      |      |      |
   11 |      |      |      |
   12 |      |      | #### |
   13 |      |      | #### |
   14 |      |      | #### |
   15 |      |      | #### |

Espera total.:  28
```

Para a impressão de números com zeros à esquerda, utilize a máscara `"%03d"` no `printf` (o limite do número de tarefas será `999`). Para a impressão de números com espaços à esquerda, utilize a máscara `"%5d"`.

2 Imposições e comentários gerais

- É fundamental que os conceitos aprendidos sobre TAD sejam respeitados.
- Seu programa não pode ter "*memory leaks*", ou seja, toda memória alocada pelo seu código deve ser corretamente liberada pelo seu código.
- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.
- Trabalhos copiados (e FONTE) terão nota zero, além de os alunos envolvidos no plágio perderem toda a nota atribuída a participação e pontos extras, entre outros (...). **Isto vale para qualquer entregável** (seção 3).
- Caso seja necessário, alunos poderão ser convocados para entrevista.

3 Entregáveis

Deverão ser entregues para avaliação do trabalho:

- **Código fonte**: código fonte do programa em `C`.
- **Arquivos de entrada e saída**: arquivos de entrada e saída utilizados para testar seu programa. Não será considerada entrada igual ao exemplo dado neste enunciado, procure elaborar um conjunto de testes abrangente, envolvendo várias agendas e muitas tarefas.
- **Documentação**: a documentação deve ser entregue em um único arquivo `PDF`, e conter:
  1. **Implementação**: descrição da implementação do programa. Não faça "print screens" de telas e não inclua o código fonte. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  2. **Análise de complexidade**: apresentar a análise de complexidade do seu algoritmo de escalonamento.
  3. **Impressões gerais**: descreva o seu processo de implementação deste trabalho. Aponte coisas que gostou bem como aquelas que o desagradou. Avalie o que o motivou, conhecimentos que adquiriu, entre outros.
  4. **Conclusão**: conclusões e comentários gerais sobre o trabalho.

3.1 Como fazer a entrega

Verifique se seu programa compila e executa na linha de comando (utilizando o compilador `GCC`) antes de efetuar a entrega. Quando o resultado for correto, entregue via Moodle um arquivo `.ZIP` e outro `.PDF` com seu nome e sobrenome. Exemplo:

- **PrimeiroNome-UltimoNome.zip**: este arquivo `.ZIP` deve conter apenas os arquivos `.c` e `.h`, utilizados na implementação, e os arquivos `.TXT` de entrada e saída, utilizados para testar seu programa.
- **PrimeiroNome-UltimoNome.pdf**: este arquivo `.PDF` deve conter a documentação do trabalho.

4 Bônus: pontos extras

Receberão **`1.0` pontos extras**, em cada quesito, os alunos que obtiverem:

1. O menor *tempo de espera total*.
2. *Tempo de espera total* inferior ao tempo da solução proposta pelo monitor da disciplina.

Ou seja, estão em disputa a bagatela de **`2.0` pontos extras**.

OBS.: A pontuação total no critério de avaliação **Trabalhos Práticos** não poderá exceder os **`30` pontos** distribuídos neste critério.