

## 2º Trabalho Prático

MC658— Análise de Algoritmos III

Prof. Cid C. de Souza

Natanael Ramos (PED)

1º Semestre de 2019

### 1 Introdução

1. *Data/hora de entrega:* 07/05/2019 (até as 23:59.59, de acordo com horário do servidor de *email* do IC).

Haverá uma tolerância de dois dias para a entrega, porém, cada 24 horas de atraso acarretará em uma redução de 20% na nota.

2. *Número de integrantes por grupo:* 2 (dois). Excepcionalmente poderá ser aceito um único grupo com 3 (três) alunos.

### 2 Modelagem usando programação linear inteira (PLI)

Para cada um dos problemas abaixo você deve encontrar um modelo de programação linear inteira (PLI) e escrever um programa usando a linguagem *Julia*, a biblioteca *JuMP* e o *Gurobi* para resolvê-lo. As instâncias de entrada para teste do seu código assim como a descrição do seu formato serão disponibilizadas na página da disciplina. Serão quatro instâncias por problema, sendo que para a menor delas, de nome `<problema>.gnumeric.instance`, existe também uma planilha do `gnumeric` de nome `<problema>.gnumeric` com os mesmos dados de entrada da instância. O nome `<problema>` se refere ao identificador entre os caracteres “[” e “]” nos problemas abaixo. Então, você deve resolver as **quatro** instâncias de extensão **.instance** utilizando sua implementação do modelo em *Julia* e a instância de extensão **.gnumeric** deve ser resolvida utilizando o resolvidor do `gnumeric` com a modelagem em planilha.

A instância de nome `<problema>.3.instance` é a maior dentre todas as quatro para um dado problema. Recomenda-se que seja imposto um limite de memória para o resolvidor *Gurobi* ao resolvê-la, de forma a evitar que o sistema operacional comece a fazer *swap*. Isso pode ser feito da seguinte forma em *Julia*, durante a construção do modelo:

```
Modelo = Model(solver = GurobiSolver(NodefileStart=k))
```

Onde *k* deve ser substituído pelo limite de memória (em gigabytes) que você deseja definir, usualmente depende da quantidade de memória RAM disponível na sua máquina.

Um arquivo de nome `formato-instancias.txt` será disponibilizado na página disciplina, contendo a descrição do formato das instâncias de cada problema. Para os problemas envolvendo grafos, os vértices estão rotulados de 1 a  $n$ , onde  $n$  é o número de vértices.

## Lista de problemas

- [gt54] Seja  $G = (V, A)$  um grafo orientado,  $s, t$  dois vértices distintos de  $G$  e  $C = \{(a_1, b_1), \dots, (a_n, b_n)\}$  pares de vértices de  $G$ . Encontre um caminho mínimo de  $s$  a  $t$  que contenha no máximo um vértice de cada par de  $C$ .
- [mn27] Dado um grafo simples  $G = (V, E)$ , determine o menor valor de  $k$  tal que  $G$  tenha uma  $k$ -coloração nos vértices.
- [ss2] Seja  $T = \{1, \dots, n\}$  um conjunto de tarefas e  $S$  um conjunto de pares  $(i, j)$  para  $1 \leq i, j \leq n$  determinando que a tarefa  $i$  deve ser executada antes da tarefa  $j$ . Para cada tarefa  $i \in T$  seja  $t_i$  o tempo de execução e  $d_i$  o prazo de término da tarefa  $i$ . Determine uma plano de execução das tarefas em um único processador, de forma que a ordem de precedência das tarefas seja respeitada e o número de tarefas que terminem fora do prazo é minimizado.
- [gt10] Dado um grafo simples  $G = (V, E)$ , encontre um emparelhamento maximal de tamanho mínimo de  $G$ . *Nota: um emparelhamento é dito ser maximal se ele não pode ter seu tamanho aumentado pela inclusão de uma nova aresta.*
- [sr2] Seja  $A = \{a_1, \dots, a_n\}$  um conjunto de  $n$  itens que precisam ser estocados em um depósito. Para todo item  $a$  de  $A$  são dados o instante inicial  $r_a$  e o instante final  $d_a (> r_a)$  que definem o intervalo de tempo em que ele deve ficar armazenado no depósito, além do espaço  $s_a$  ocupado por esse item. Suponha que todos esses valores sejam dados por números inteiros positivos. O depósito é dividido em *slots* de tamanho unitário e o espaço  $s_a$  correspondente a um item  $a$  qualquer de  $A$  deve ser ocupado em *slots* dispostos sequencialmente e que ficam reservados para aquele item no intervalo de tempo  $[r_a, d_a)$ . Evidentemente dois itens não podem ocupar um mesmo *slot* ao mesmo tempo. O objetivo deste problema é minimizar a capacidade  $D$  do depósito de modo que ele possa armazenar todos os itens de  $A$ .

## 3 Forma de entrega do trabalho

A entrega deve ser feita por *email* enviado ao docente, **com cópia para o PED**, sendo que:

- o campo `subject` deverá vir preenchido **obrigatoriamente** com os seguintes dizeres:

[MC658-2019s1] TP2 - grupoXX

onde XX é o identificador do grupo (a ser divulgado oportunamente).

- A mensagem deverá conter um anexo composto de um **único** arquivo compactado (com o comando `tar`) e chamado `grupoXX.tgz`.

Ao ser descompactado `grupoXX.tgz` deve gerar um diretório chamado `grupoXX/` contendo: (i) um subdiretório `codigo/` e (ii) o arquivo `relatorio.pdf` (ver especificação mais adiante). Dentro do subdiretório `codigo`, para cada exercício resolvido, deve existir um arquivo em linguagem *Julia* chamado `NOME.jl` onde `NOME` é o identificador do exercício, dado entre “[” e “]” na primeira linha do enunciado.

Cada arquivo com extensão “.jl” no diretório `codigo/` ao ser executado com a linha de comando,

```
julia <NOME>.jl <arg1> <arg2>
```

dever gerar um arquivo de saída chamado `NOME.out` em que a primeira linha deve conter apenas o valor da função objetivo e mais nada. Os argumentos passados na linha de comando referem-se ao nome do arquivo de dados (`arg1`) e ao tempo limite de execução do modelo em segundos (`arg2`), sendo este último um valor inteiro.

Além disso, todo programa em *Julia* que modela um problema deve terminar com as seguintes linhas de código, as quais pressupõem que: (i) você tenha declarado o seu modelo através do comando `MeuModelo = Model(solver=GurobiSolver(TimeLimit=TL))`<sup>1</sup> e que (ii) este tenha sido resolvido através do comando `status=solve(MeuModelo)`. Note que estes comandos imprimem estatísticas sobre a execução do modelo na *saída padrão do sistema*.

```
# -----
# Relatório
println("=====")
if status == :Optimal
    println("Solução ótima encontrada.")
elseif status == :Unbounded
    println("Problema é ilimitado.")
elseif status == :Infeasible
    println("Problema é inviável.")
elseif status == :UserLimit
    println("Parado por limite de tempo ou iterações.")
elseif status == :Error
    println("Erro do resolvedor.")
else
    println("Não resolvido.")
end

println("Número de nós explorados: ", getnodecount(BAND::Model))
D = getobjbound(MeuModelo::Model)
```

<sup>1</sup>Note que a variável TL deverá receber o segundo argumento (`arg2`) passado na linha de comando.

```

P = getobjectivevalue(MeuModelo::Model)
@printf("Melhor limitante dual: %.2f\n", D)
@printf("Melhor limitante primal: %.2f\n", P)
Gap = (abs( D - P )/P)*100
@printf("Gap de otimalidade: %.2f\n", Gap)
@printf("Tempo de execução: %.2f\n", getsolvetime(MeuModelo::Model))

# -----

```

Finalmente, o subdiretório `codigo` deve conter também as planilhas do `gnumeric` referentes a cada um dos problemas do enunciado. A planilha de um dado problema deve ter duas abas: `Entrada` e `Solucao`. Na aba `Entrada` estão inseridos os dados da instância, conforme os arquivos disponibilizados na página da disciplina. Na aba `Solucao`, a célula A1 deve estar reservada para conter o valor ótimo da instância (função objetivo). O formato do restante da aba é livre, mas devem ser claramente indicadas as células que são variáveis, e para cada restrição ou grupo de restrições, deve ser indicado: o lado esquerdo, o lado direito e o operador, onde operador deve ser:  $\leq$ ,  $\geq$ , ou  $=$ , `Integer` ou `Binary` (para especificar domínios de variáveis quando for o caso). Na página do trabalho foi disponibilizado um exemplo de planilha que modela o problema da mochila binária.

**Formato do relatório.** O relatório deverá ser entregue em formato `pdf` (nenhum outro formato será aceito) no arquivo chamado `relatorio.pdf`, conforme já destacado anteriormente. O texto deverá ser composto das seguintes seções:

1. Seção 1: Identificação dos integrantes do grupo com nome e RA.
2. Seção 2: Para cada um dos cinco exercícios deverão ser apresentadas as seguintes informações: *(i)* o identificador do exercício que foi resolvido no formato [XXX] conforme o enunciado recebido pelo grupo; *(ii)* a definição das variáveis usadas no modelo; *(iii)* a descrição de cada uma das (famílias de) restrições do modelo, dando a sua fórmula acompanhada de uma explicação **sucinta** do seu significado; e *(iv)* a fórmula da função objetivo.

Ao final dessa seção deverá ser dada uma tabela de resultados no seguinte formato. Cada linha estará associada a um exercício, identificado por [XXX] na primeira coluna, e as demais colunas corresponderão ao identificador da instância que foi resolvida (`gnumeric` ou um inteiro de 1 a 3). Em cada célula deverá ser apresentado o valor da função objetivo obtido para o par (exercício,instância) correspondente. Se o valor ótimo não foi alcançado no tempo limite estabelecido, escreva na célula os valores dos limitantes primal ( $P$ ) e dual ( $D$ ) no formato  $P-D$ .

O texto deverá informar **necessariamente** o ambiente computacional onde foram realizados os experimentos, incluindo detalhes de *hardware* (CPU, memória RAM, clock, etc) e *software* (sistema operacional, linguagem de programação, compilador, etc).

**OSERVAÇÃO IMPORTANTE:**

- não colocar no relatório o enunciado dos problemas tratados. O relatório deve limitar-se a descrever o que foi feito pelo grupo e, quando solicitado, a analisar os resultados obtidos.

## 4 Critérios de Correção

A distribuição de pontos do trabalho será feita do seguinte modo:

- Corretude das formulações: até 2 pontos.
- Implementação em *Julia*: até 4 pontos, dependendo da clareza do código e dos resultados;
- Implementação no *gnumeric*: até 2 pontos, dependendo da clareza da planilha e da corretude dos resultados;
- Relatório: até 2 pontos, dependendo da qualidade do documento;