MC548: Projeto e Análise de Algoritmos II

Prof. Cid C. de Souza – 3ª Prova – (20/06/2011)

Nome:

RA: **Turma:**

Observação: o peso das questões 1, 2 e 3 será decidido pelo docente da seguinte forma: dentre estas três, as duas questões que você responder melhor terão peso 3 e a restante terá peso 2. A questão 4 terá peso 2.

Questão	frac	Peso	Nota
1			
2			
3			
4		2.0	
Total		10.0	

Instruções:

1. A duração da prova é de 110 minutos.
2. Nenhum aluno poderá sair da sala antes de terem sido transcorridos 60 minutos de prova.
3. O aluno que sair de sala antes do término da prova deve entregá-la em definitivo.
4. Coloque o seu nome, RA e turma em **no alto desta página e em todas as folhas de resposta**.
5. Não é permitido usar qualquer material de consulta.
6. **Questões mal justificadas serão consideradas erradas !**
7. Use as folhas de papel almaço entregues pelo docente para responder às questões da prova.
8. A prova pode ser feita a lápis porém, nesse caso, fica a critério do docente aceitar eventuais pedidos de revisão de nota.
9. O uso de calculadoras ou quaisquer outros equipamentos eletrônicos, **inclusive celulares**, está proibido durante a prova.
10. Não desgrampeie o caderno de questões.
11. É **obrigatória** a devolução deste caderno contendo os enunciados junto com as suas folhas de resposta.

1. Seja $G = (V, E)$ um grafo **completo** e não-orientado com custos nas arestas. Define-se como sendo um ***2-emparelhamento mínimo*** de G um subgrafo gerador deste grafo que tenha custo mínimo e tal que todos os vértices possuem grau 2 (dois) (NOTA: o *custo* de um subgrafo qualquer G' de G é dado pela soma dos custos das arestas pertencentes a G').
Sabendo que existem algoritmos polinomiais que encontram um 2-emparelhamento mínimo em um grafo, projete um algoritmo $\frac{4}{3}$-aproximado para o *problema do caixeiro viajante (TSP)* definido para grafos completos cujos custos de todas arestas sejam dados pelos valores 1 (um) ou 2 (dois).
Para ser considerada correta a sua resposta deve apresentar um pseudo-código do algoritmo (em alto nível), uma breve análise da complexidade do mesmo (para isso, considere que o 2-emparelhamento mínimo tem complexidade $O(|V|^k)$ para alguma constante k) e, claro, uma prova (bem argumentada) da aproximação.

---

2. Seja $G = (V, E)$ um grafo não-orientado (qualquer). Foi visto em aula que:

- um *emparelhamento* em G é um subconjunto M de arestas de E tal que, para todo vértice u em V, no máximo uma aresta de M incide em u.
- um *emparelhamento maximal* em G é um emparelhamento M tal que, para todo $f \in E \setminus M$, o conjunto $M \cup \{f\}$ **não** é um emparelhamento de G.

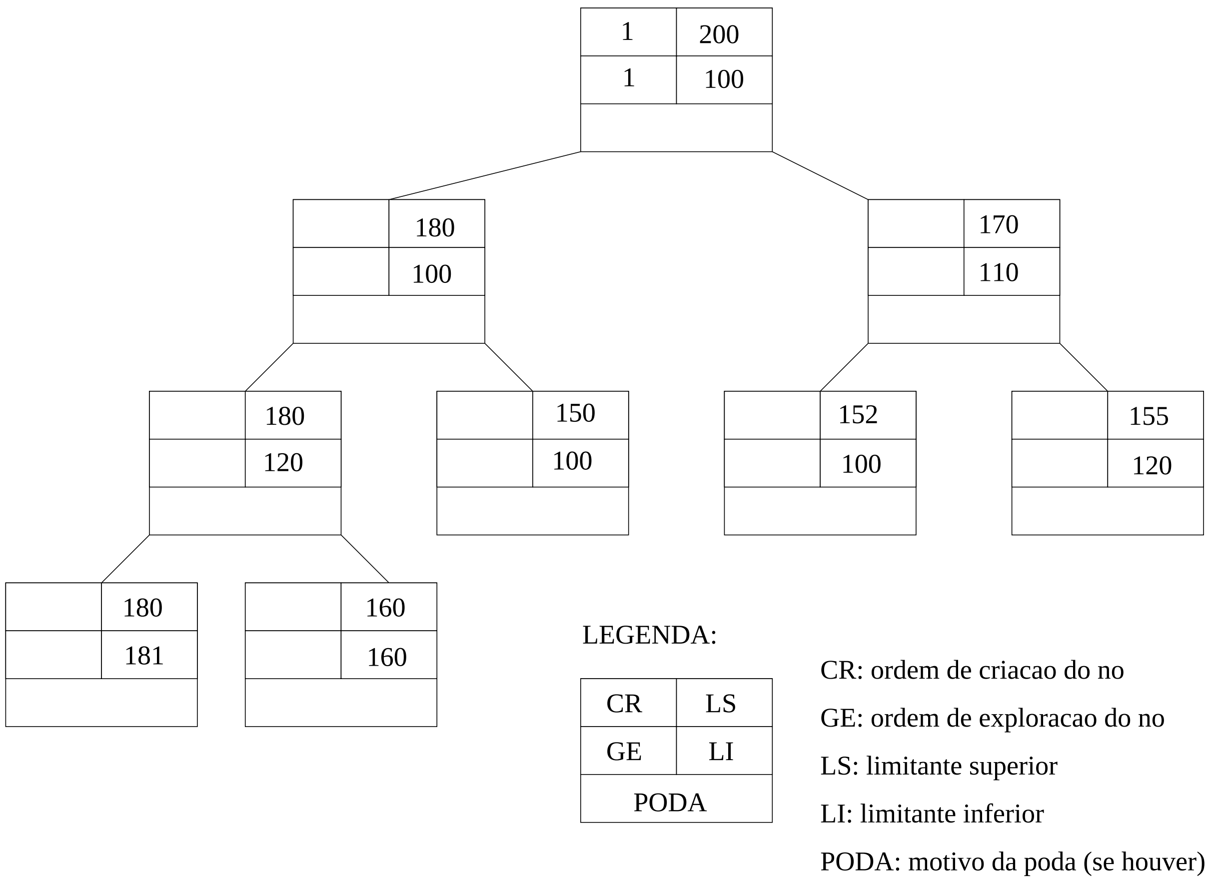
O problema de encontrar um emparelhamento maximal de tamanho mínimo (i.e., menor cardinalidade), ao qual denotaremos por MIN-MAXIMAL-MATCHING, é $\mathcal{NP}$-difícil.
Faça uma formulação **compacta** (i.e., de tamanho polinomial em $n = |V|$ e $m = |E|$) do problema MIN-MAXIMAL-MATCHING usando Programação Linear Inteira (PLI).
Para ser considerada correta a sua resposta deve apresentar: (a) a definição das variáveis usadas; (b) a função objetivo; (c) as restrições do modelo junto com uma explicação sucinta do seu significado; e (d) um resumo da quantidade total de variáveis e de restrições do modelo em função de n e de m.

---

3. Na figura a seguir é mostrada a porção da árvore de enumeração explorada por um algoritmo *branch-and-bound* ao resolver uma instância de um problema de maximização. O algoritmo faz uso de uma função classificadora que, **no momento da criação de um nó da árvore**, calcula tanto o limitante dual (superior) quanto o limitante primal (inferior). Quando a função classificadora detecta que o subproblema representado por um nó é **inviável**, os valores retornados por ela são tais que o limitante inferior é uma unidade maior que o limitante superior. Sabe-se ainda que o algoritmo utiliza a estratégia de "*best bound*" (melhor limitante) para explorar a árvore de enumeração.

(3a) Seguindo a legenda indicada, preencha os valores faltantes na figura abaixo. Você deve supor que a poda (ou amadurecimento) de um nó é feita no processo de sua *criação* e imediatamente após a execução da função classificadora desde que, é claro, os valores retornados por ela assim o permitam. O outro momento em que um nó pode ser podado é quando ele é retirado da lista de nós ativos, ou seja, no momento da sua *exploração*. Isso é feito seguindo as regras usuais de um algoritmo *branch-and-bound*. Considere, finalmente, que o valor do **melhor limitante primal** mantido pelo algoritmo é atualizado sempre que, ao ser chamada durante a criação de um nó, a função classificadora encontra um limite inferior maior que o atual.

(3b) Usando a numeração dos nós induzida pela ordem em que eles foram **criados (!)**, indique nos campos apropriados na figura quais nós foram *podados* (ou *amadurecidos*) pelo algoritmo e por qual razão. (NOTA: para todos os efeitos, considere que os nós *explorados* pelo algoritmo não são classificados como tendo sido *podados* por ele.)



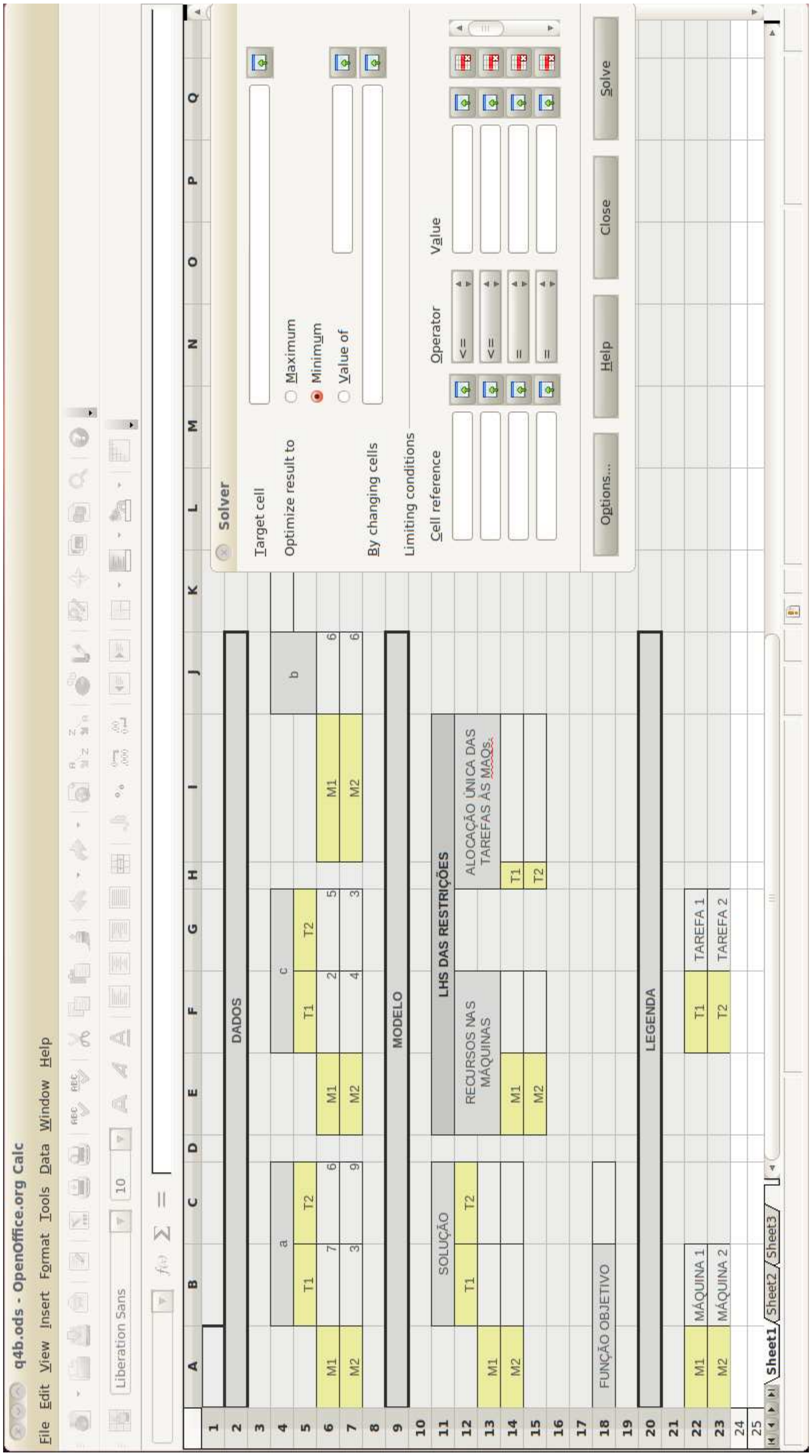
4. Responda os itens a seguir.

4(a) Considere o problema do *Generalized Assignment* descrito a seguir. São dadas m máquinas e n tarefas. Para todo $i = \{1 \ldots m\}$ e $j = \{1 \ldots n\}$, a execução da tarefa j na máquina i consome uma quantidade (positiva) de recursos dada por a_{ij} a um custo c_{ij}. Além disso, para todo $i = \{1 \ldots m\}$, a máquina i dispõe de uma quantidade **máxima** de recursos dada por b_i. O problema que se quer resolver é alocar cada uma das n tarefas a uma das m máquinas de modo que o custo total, dado pela soma dos custos de execução de cada tarefa, seja minimizado, respeitadas as restrições de recurso nas máquinas.

A seguir vê-se uma porção de um código feito no *MathProg* para modelar o *Generalized Assignment*. Usando os comandos dessa linguagem, complete o programa de modo que ele formule corretamente o problema descrito acima.

```
=====================================================
# modelo GLPK do generalized assignment (questão 4a)

/* indices dos conjuntos */
set MAQUINAS;
set TAREFAS;
/* ===== DADOS DE ENTRADA ===== */
/* — consumo de recurso por tarefa nas maquinas */
param a{i in MAQUINAS, j in TAREFAS};
/* — custo da execucao da tarefa em cada maquina */
param c{i in MAQUINAS, j in TAREFAS};
/* — quantidade maxima de recurso por maquina */
param b{i in MAQUINAS};

/* ===== VARIAVEIS DO MODELO ===== */
var x{i in MAQUINAS, j in TAREFAS}, integer, >= 0;

/* ===== FUNCAO OBJETIVO ===== */
[                                                  ] ;

/* ===== RESTRICOES ===== */
/* — limitando recursos nas maquinas — */
[                                                  ] ;

/* — impondo a execucao das tarefas — */
[                                                  ] ;
...
=====================================================
```

4(b) Considere novamente o problema do item anterior. Na figura da página 5 vê-se uma planilha do *Open Office* na qual deseja-se resolver uma instância do problema envolvendo duas máquinas ($m = 2$) e duas tarefas ($n = 2$). Usando as funções SUMPRODUCT e SUM do *software*, escreva os conteúdos das células correspondentes aos lados esquerdos (LHS) das restrições e à função a ser otimizada. Em seguida, na janela do SOLVER (à direita na figura), preencha os campos: (a) *"Target cell"*; (b) *"By changing cells"*, e (c) aqueles que definem os lado esquerdo e direito relativos às restrições do modelo (suponha que, usando o botão *"Options"*, já tenha sido definido que as variáveis são inteiras e não-negativas).



q4b.ods - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Liberation Sans
10
DADOS
a
T1
T2
M1
7
6
M2
3
9
c
T1
T2
M1
2
5
M2
4
3
M1
M2
b
6
6
MODELO
SOLUÇÃO
T1
T2
M1
M2
LHS DAS RESTRIÇÕES
RECURSOS NAS MÁQUINAS
M1
M2
ALOCAÇÃO ÚNICA DAS TAREFAS ÀS MÃOs
T1
T2
FUNÇÃO OBJETIVO
LEGENDA
M1
MÁQUINA 1
T1
TAREFA 1
M2
MÁQUINA 2
T2
TAREFA 2
Sheet1
Sheet2
Sheet3
Solver
Target cell
Optimize result to
Maximum
Minimum
Value of
By changing cells
Limiting conditions
Cell reference
Operator
Value
<=
<=
=
=
Options...
Help
Close
Solve