

Trabalho Prático

MC548

Projeto e Análise de Algoritmos II

Alunos:

Renato Macedo Viegas
Rômulo Braga

R.A.:011729
R.A.:009841

PROBLEMA 1 (model equi_corte)

Variáveis:

Do arquivo de entrada:

n – variável inteira que indica o número de vértices

arestas - variável inteira que indica o número de arestas do grafo

Do programa:

x – vetor (1 x n) de variáveis binárias que indica se um vértice pertence ou não ao equicorte

y – matriz (n x n) de variáveis binárias que indica se uma aresta pertence ou não ao equicorte

Restrições:

1 – Faz com que o número de vértices escolhidos para o corte seja de, aproximadamente, a metade do número de vértices do grafo.

Restringe_num_vertices := $\sum(i \text{ in } 1..n) x(i) = \text{floor}(n/2)$

2 – Faz com que sejam escolhidas arestas onde apenas um de seus vértices esteja dentro do corte. É, na verdade, um XOR entre x(i) e x(j), onde uma aresta só pode pertencer ao corte se apenas um x(i) ou x(j) estiver no corte, se ambos estiverem, ou nenhum estiver, a aresta também não está.

```
forall(i in 1..n, j in 1..n | (i<j) AND (arestas(i,j)=1)) do
    RestringeAresta1(i,j) :=  $y(i,j) \leq x(i) + x(j)$ 
    RestringeAresta2(i,j) :=  $y(i,j) \leq 2 - x(i) - x(j)$ 
    RestringeAresta3(i,j) :=  $y(i,j) \geq x(i) - x(j)$ 
    RestringeAresta4(i,j) :=  $y(i,j) \geq x(j) - x(i)$ 
end-do
```

Função Objetivo (minimizar equicorte do grafo de entrada)

A função objetivo é pegar um equi_corte com o mínimo de arestas possíveis.

Objetivo := $\sum(i \text{ in } 1..n, j \text{ in } 1..n) y(i, j)$

Saída do programa:

- valor da função objetivo
- vértices utilizados
- arestas utilizadas

PROBLEMA 2 (model coleta_lixo)

Variáveis:

Do arquivo de entrada:

n – variável inteira que indica os setores existentes na cidade

m - variável inteira que indica potenciais locais para a instalação de depósitos de lixo

custo_km - variável real que indica o preço por quilômetro e por metro cúbico de lixo transportado, independente do percurso adotado

capacidade – matriz ($1 \times m$) de inteiros que indica capacidade anual de armazenamento de lixo de um dado depósito, em toneladas.

req – matriz ($1 \times n$) de inteiros que indica capacidade anual de produção de lixo de um dado setor, em toneladas.

custo_fixo – matriz ($1 \times m$) de reais que indica o custo para se instalar um depósito de lixo em um dado local

distância – matriz ($n \times m$) de reais que indica a distância entre um local onde pode ser instalado m depósito de lixo e os setores da cidade

Do programa:

nomearq – string que guarda o nome do arquivo que o usuário indica como “input” de dados

y – matriz ($1 \times m$) de variáveis binárias que indica se um depósito está instalado em um local ou não

atende – matriz ($m \times n$) de variáveis binárias que indica se um depósito atende a um setor da cidade ou não

Restrições:

1 – Um setor somente pode ser atendido por um depósito. Essa restrição é executada para cada um dos setores.

$\text{forall } (i \text{ in } 1..n) \text{ sum}(j \text{ in } 1..m) \text{ atende}(j,i) = 1$

2 – A soma dos setores que um depósito atende não pode ser maior que sua capacidade de armazenamento. Essa restrição é executada para cada um dos depósitos.

forall (j in 1..m) sum(i in 1..n) atende(j,i)*req(1,i) <= capacidade(1,j)

3 – Um depósito só pode atender algum setor caso esse depósito esteja instalado, ou seja, seria absurdo não instalar um depósito e mandá-lo atender, por exemplo, dois setores.

forall (j in 1..m, i in 1..n) atende(j,i) <= y(1, j)

Função Objetivo: (minimizar o custo total de operação da solução ótima)

A função objetivo soma duas partes:

Parte I: soma o custo fixo de se instalar todos os depósitos que foram selecionados como solução do programa (aqueles que não foram selecionados, contribuem com o valor “zero”)

Parte II: são dois laços que basicamente dizem: para um dado setor, tome o depósito que o atende. Então, verifique algo que chamaremos aqui de custo do atendimento, que significa quanto é gasto para se transportar a quantidade anual de lixo que um setor produz, desse mesmo setor para o depósito em questão. Para isso, tomamos a distância entre o depósito e o setor, multiplicamos pela quantidade de lixo que o setor produz e, finalmente, multiplicamos pelo custo_km (variável definida logo acima) e por 1000. Isso é feito para todos os setores.

Objetivo := sum(j in 1..m) y(1,j)*custo_fixo(1,j) +
 sum(i in 1..n, j in 1..m)
 1000*atende(j,i)*distancia(i,j)*custo_km*req(1,i)

Saída do programa:

- depósitos devem ser instalados
- que depósito atende cada setor
- valor da função objetivo

PROBLEMA 3: (model restaurante_estrategia_a e b)

Para o problema 3, descreveremos os modelos utilizados para as estratégias “a” e “b” aqui, em um mesmo local, pois a única diferença entre os modelos é a criação de uma restrição que somente é aplicada à estratégia b – tal restrição será destacada para que o docente perceba o ponto sobre o qual estamos falando.

Variáveis:

Do arquivo de entrada:

n – variável inteira que indica possíveis locais de instalação de restaurantes

lucro – matriz ($n \times 2$) de reais que indica o lucro esperado quando instalado um restaurante de uma das cadeias em um local (coluna 1, cadeia 1 / coluna 2, cadeia 2)

Menos5 – matriz ($n \times n$) de inteiros que indica se a distância entre dois locais é menor que 5 Km

Do programa:

nomearq – string que guarda o nome do arquivo que o usuário indica como “input” de dados

x – matriz ($1 \times n$) de variáveis binárias que indica se a cadeia 1 está instalada em um dos n locais

y – matriz ($1 \times n$) de variáveis binárias que indica se a cadeia 2 está instalada em um dos n locais

Restrições:

1 – Um local só pode ter um restaurante, no máximo, ou seja, ou tem um restaurante da cadeia 1, ou tem um restaurante da cadeia 2, ou não tem nenhum.

forall (i in $1..n$) $x(1,i) + y(1,i) \leq 1$

2 – Ao todo, só podem ser instalados “n” restaurantes, somando-se os instalados das cadeias 1 e 2.

sum(i in $1..n$) ($x(1,i) + y(1,i)$) $\leq n$

3 – Para cada local onde pode ser instalado um restaurante, são verificados todos seus vizinhos. Tal restrição impede que dois restaurantes da mesma cadeia sejam instalados em uma distância inferior a 5 Km, assim como ressaltado no enunciado do problema

```

forall (i in 1..n, j in 1..n | (i<>j)) do
    1 - (x(1,i)*Menos5(i, j)) >= x(1, j)
    1 - (y(1,i)*Menos5(i, j)) >= y(1, j)
end-do

```

4 - ***OBS: Esta restrição somente é aplicável à estratégia “b” do Problema 3

A restrição 4, encontrada apenas na estratégia “b”, faz com que **se** um restaurante de uma dada cadeia é instalado em um local, obrigatoriamente, um restaurante de outra cadeia deve ser instalado a um local que não diste mais de 5 Km do primeiro.

```

forall (i in 1..n) do
    sum(k in 1..n | (i<>k) ) Menos5(i,k)*y(1,k) >= x(1,i)
    sum(k in 1..n | (i<>k) ) Menos5(i,k)*x(1,k) >= y(1,i)
end-do

```

Função Objetivo (maximizar os lucros)

Deve-se instalar restaurantes de diferentes cadeias, nos locais disponíveis, de forma a obter o máximo lucro possível. Para tal, é levado em consideração o lucro presumido, que é passado como um dos itens do “input” do programa. Assim, somamos os lucros da instalação dos restaurantes nos locais disponíveis (se um restaurante não está instalado em algum local, ele colabora com uma parcela “zero” da soma).

Objetivo := $\sum(i \text{ in } 1..n) x(1,i) \cdot \text{lucro}(i,1) + \sum(i \text{ in } 1..n) y(1,i) \cdot \text{lucro}(i,2)$

Saída do programa:

- locais onde instalou restaurantes da cadeia 1
- locais onde instalou restaurantes da cadeia 2
- valor da função objetivo

PROBLEMA 4 (model distr_combustivel)

Variáveis:

Do arquivo de entrada:

n – variável inteira que indica o número de tanques disponíveis no caminhão

m – variável inteira que indica o número de combustíveis diferentes que existem

capacidade – matriz ($1 \times n$) de inteiros que indica a capacidade de cada tanque

demanda – matriz ($1 \times m$) de inteiros que indica a quantidade de cada combustível que foi solicitada.

Do programa:

nomearq – string que guarda o nome do arquivo que o usuário indica como “input” de dados

x – matriz ($m \times n$) de inteiros que indica a quantidade de combustível i que é levado em um dado compartimento j

y – matriz ($m \times n$) de variáveis binárias que indica se levo combustível i no compartimento j

Restrições:

1 – Somando toda a quantidade de combustível i que é levada no caminhão, esse valor deve ser menor (quando não é possível levar tudo) ou igual à quantidade demandada desse combustível. Isso é feito para cada combustível.

forall (i in $1..m$) sum(j in $1..n$) $x(i, j) \leq$ demanda($1,i$)

2 – A quantidade de combustível que é levada em cada tanque do caminhão não pode ultrapassar a capacidade do tanque em questão. Isso é checado para cada tanque.

forall (j in $1..n$) sum(i in $1..m$) $x(i, j) \leq$ capacidade($1,j$)

3 – Um tanque só pode levar um tipo de combustível

forall (j in $1..n$) sum(i in $1..m$) $y(i, j) \leq 1$

4 – Um tanque só pode levar uma determinada quantidade de combustível, se a variável “ y ” na posição do tanque e combustível em questão for igual a “1”, pois seria absurdo dizer que um tanque leva uma determinada quantidade > 0 de um

combustível qualquer e, a variável y indicar que o tanque em questão não leva o combustível citado. Isso cria a dependência abaixo esboçada entre as variáveis binárias $x(i,j)$ e $y(i,j)$.

forall (i in 1..m, j in 1..n) $x(i, j) \leq y(i, j) * demanda(1,i)$

Função Objetivo (minimizar a quantidade de combustível que deixa de ser transportada)

Basicamente, a função objetivo soma a quantidade de combustível que consigo levar em cada tanque e subtrai esse valor do total de combustível solicitado pelos clientes.

Objetivo := $\sum(i \text{ in } 1..m) demanda(1,i) - \sum(i \text{ in } 1..m, j \text{ in } 1..n) x(i,j)$

Saída do programa:

- O tipo e a quantidade de combustível que cada tanque leva
- Valor da função objetivo