Relatório do trabalho 1 de CI1238

Nome: João Gabriel Baraldi GRR20205509
Nome: Nicolas Andre Rizzardi GRR20206152
Nome: Patrick de Oliveira Lemes GRR20211777

O problema

Distribuição de carga em avião
Um avião de carga tem k compartimentos, numerados de 1 a k, para armazenar sua carga. Cada um destes compartimentos tem limites de peso e espaço ocupado, dados pelos valores wi (em toneladas) e vi (em m^3)), respectivamente, para i = 1,2,..., k.

Além disso, o peso em cada compartimento deve ter a mesma proporção em relação a sua capacidade que os demais, para manter a distribuição de peso do avião equilibrada.
Temos n carregamentos para embarcar no avião, numerados de 1 a n, cada um com peso, volume e ganho por tonelada transportada, dados pelos valores pj (em toneladas), tj (em m^3) e gj (em reais/tonelada), para j = 1, 2, . . . , n.
É possível carregar parte de cada um dos carregamentos. Ou seja, qualquer proporção não negativa dos carregamentos pode ser aceita.
O objetivo é determinar quanto de cada carregamento deve ser aceito e como distribuir pelos compartimentos de forma a maximizar o ganho total.

Modelagem

Queremos maximizar o lucro dos carregamentos distribuídos nos compartimentos do avião, o ganho é proporcional ao peso. Portanto:
$max\sum_{i=1}^k$ $\sum_{j=1}^n g_j \cdot p{_j}{_i}$
$S.A.$

- **Restrição peso do compartimento:** A soma dos pesos dos carregamentos não deve exceder a capacidade maxima de peso do compartimento:
  $\sum_{j=1}^n p{_j}{_i} \leq w{_i}, : \forall : i=1,2,\dots,k$
- **Restrição volume relativo ao peso:** A soma dos volumes do itens carregados não deve exceder o volume maximo do compartimento:
  $\sum_{j=1}^n t{_j}\cdot p{_j}{_i} \leq v{_i}, : \forall : i=1,2,\dots,k$
- **Restrição peso do carregamento:** A proporção de peso carregado (todos os carregamentos do compartimento) pela capacidade de peso do compartimento deve ser igual para todos os compartimentos:
  $a = \sum_{j=1}^n \frac{p{_j}{_i}}{w{_i}}, : \forall : i=1,2,\dots,k$
- **Restrição peso do compartimento:** A soma do peso do carregamento j em todos os compartimentos não pode exceder o peso total dele mesmo
  $\sum_{i=1}^k p{_j}{_i} : \leq : p{_j}, : \forall :j=1,2,\dots,n$
- **Restrição peso maior ou igual a zero:** Como é um problema de enumeração, o peso dos carregamentos nos compartimentos não pode ser negativo.

$p{_j}{_i} : \geq 0, : \forall i=1,2,...,k : e : \forall j=1,2,...,n$
ou
$p{_j}{*i}|in|mathbb{R}*{{\geq0}}, : \forall i=1,2,...,k : e : \forall j=1,2,...,n$

Compartimentos

$i=indices:k$	**$w{_i} = Peso:t$**	**$v{_i} = Volume:m{^3}$**
1	10	6800
2	16	8700
3	8	5300
$\vdots:i$	$\vdots$	$\vdots$
k		

Carregamentos

$j = indices (n)$	**($p{_j}) = Peso:t$**	**$(t{_j}) = Volume : m{^3}$**	**$g{_j}= Ganho:por:tonelada$ $\frac{Reais}{t}$**
1	18	480	310
2	15	650	80
3	23	580	350
4	12	390	285
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n			

Implementação

Com a modelagem pronta, a implementação é relativamente simples, só precisamos passar as variáveis, restrições e o objetivo para o programa formatado no padrão do lp_solve.

Entrada de dados:

Preenche a tabela de compartimento com sua capacidade de peso e de volume. E também a tabela de carregamentos com seus atributos respectivos.

```
for(int i = 0; i < n_compartimentos; ++i){
    scanf("%d %d", &w[i], &v[i]);
  }

  for(int i = 0; i < n_carregamentos; ++i){
    scanf("%d %d %d", &p[i], &t[i], &g[i]);
  }
```

Função objetivo:

Gera função objetivo com base na função objetivo do modelo, que é uma maximização, percorrendo os compartimentos e os carregamentos, onde $x{_i}{_j}$ seria $p{_i}{_j}$, desenrolando os somatórios da função objetivo.
Isso é feito para ser entendido pelo lp_solve.

```
fprintf(f, "max : ");

  int i, j;
  for(i = 0; i < n_compartimentos - 1; ++i){
    for(j = 0 ; j < n_carregamentos; ++j){
      fprintf(f, "%dx%d%d + ", g[j], i, j);
    }
  }

  for(j = 0 ; j < n_carregamentos - 1; ++j){
    fprintf(f, "%dx%d%d + ", g[j], i, j);
  }

  fprintf(f, "%dx%d%d;\n\n", g[j], i, j);
```

Restrição peso do compartimento:

Gera a restrição de capacidade de peso maximo do compartimento no formato aceito pelo lp_solve

```
for(i = 0; i < n_compartimentos; ++i){
    for(j = 0; j < n_carregamentos - 1; ++j){
      fprintf(f, "x%d%d + ", i, j);
    }
    fprintf(f, "x%d%d <= %d;\n", i, j, w[i]);
  }

  fprintf(f, "\n");
```

Restrição volume relativo ao peso

Gera a restrição de capacidade de volume maximo do compartimento no formato aceito pelo lp_solve.

```
for(i = 0; i < n_compartimentos; ++i){
    for(j = 0; j < n_carregamentos - 1; ++j){
      fprintf(f, "%dx%d%d + ", t[j], i, j);
    }
    fprintf(f, "%dx%d%d <= %d;\n", t[j], i, j, v[i]);
  }

  fprintf(f, "\n");
```

Restrição peso do carregamento

Gera a restrição para que a quantidade maxima da carga disponivel não seja excedida.

```
for(j = 0; j < n_carregamentos; ++j){
    for(i = 0; i < n_compartimentos - 1; ++i){
      fprintf(f, "x%d%d + ", i, j);
    }
    fprintf(f, "x%d%d <= %d;\n", i, j, p[j]);
  }

  fprintf(f, "\n");
```

Restrição peso do compartimento:

Gera a restrição que garante que não seja carregado pelos compartimentos mais do que está disponivel em cada carregamento.

```
 for(i = 0; i < n_compartimentos; ++i){
    fprintf(f, "%da = ", w[i]);
    for(j = 0; j < n_carregamentos - 1; ++j){
      fprintf(f, "x%d%d + ", i, j);
    }
    fprintf(f, "x%d%d;\n", i, j);
  }

  fprintf(f, "\n");
```

Restrição peso maior ou igual a zero

Gera a restrição que garante que nenhum peso será negativo.

```
for(i = 0; i < n_compartimentos; ++i){
    for(j = 0; j < n_carregamentos; ++j){
        fprintf(f, "x%d%d >= 0;\n", i, j);
    }
  }

  fprintf(f, "\n");
```

No final foi optado por colocar cada variável em um vetor próprio para não precisar chamar uma struct completa para utilizar apenas uma propriedade da mesma por vez.