Técnicas de Desenho de Algoritmos

Ana Paula Tomás

Desenho e Análise de Algoritmos (CC2001)

Novembro 2019

Técnicas de desenho de algoritmos

- Pesquisa exaustiva ou força-bruta (*exhaustive search* ou *brute-force*)
- Pesquisa exaustiva com retrocesso (e, possivelmente, heurísticas) (*backtracking search*)
- Divisão-e-conquista (*Divide-and-conquer*)
- **Estratégias ávidas**, gananciosas, gulosas (*greedy*)
- **Programação Dinâmica** (*dynamic programmimng*)
- ...

Problemas de trocos com número de moedas ilimitado

Problema: Supondo que se tem um número não limitado de moedas de valores 200, 100, 50, 20, 10, 5, 2, e 1, qual é o **número mínimo** de moedas necessário para formar uma quantia Q?

- Abordagem de programação dinâmica é **ineficiente**.
- Prova-se que a **estratégia greedy** que consiste em começar por **usar a moeda de valor mais alto** $v_k \leq Q$ **o número máximo de vezes que puder** (isto é, $n_k = \lfloor Q/v_k \rfloor$ vezes) e aplicar a mesma estratégia para obter a quantia $Q - n_k v_k$ restante, determina a **solução ótima**, em $O(m)$, sendo m o número de tipos de moedas existentes.

Atenção! Para garantir $O(m)$, é necessário usar $Q - n_k v_k$ em vez de dar uma moeda v_k e aplicar a estratégia a $Q - v_k$. Note que $O(Q)$ é $O(2^{\log_2 Q})$ e, portanto, é exponencial no tamanho da representação de Q (input) em binário (assumido no modelo RAM para análise assintótica).

Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se $\{200, 100, 50, 20, 10, 5, 2, 1\}$:

- Seja $x^\star$ uma solução ótima para a quantia Q. Seja $x_v^\star$ é o número de moedas que usa de valor v.
- Se $x_{100}^\star > 1$, a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto, $x_{100}^\star \leq 1$.
  Analogamente se conclui que: $x_{50}^\star \leq 1$, $x_{10}^\star \leq 1$, e $x_1^\star \leq 1$.
- Se $x_{20}^\star > 2$ então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto, $x_{20}^\star \leq 2$. Analogamente, $x_2^\star \leq 2$.
- Não pode ter simultaneamente $x_2^\star = 2$ e $x_1^\star = 1$, pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto $2x_2^\star + x_1^\star \leq 4$.
  Também não tem simultaneamente $x_{20}^\star = 2$ e $x_{10}^\star = 1$.
- Como $2x_2^\star + x_1^\star \leq 4$, $x_5^\star \leq 1$ e $x_{10}^\star \leq 1$ então $5x_5^\star + 2x_2^\star + x_1^\star \leq 9$ e $10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 19$. Analogamente, se deduz que $20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 49$, $50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 99$. $100x_{100}^\star + 50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 199$.

  Tem-se $\sum_{i=1}^{k} v_i x_{v_i}^\star < v_{k+1}$, para todo k. Portanto, $x^\star$ é a solução greedy. □

NB: A estratégia *greedy* apresentada não seria correta para, por exemplo, $V = \{1, 300, 1000\}$, $Q = 1200$.

Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se $\{200, 100, 50, 20, 10, 5, 2, 1\}$:

- Seja $x^\star$ uma solução ótima para a quantia Q. Seja $x_v^\star$ é o número de moedas que usa de valor v.
- Se $x_{100}^\star > 1$, a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto, $x_{100}^\star \leq 1$.
  Analogamente se conclui que: $x_{50}^\star \leq 1$, $x_{10}^\star \leq 1$, e $x_1^\star \leq 1$.
- Se $x_{20}^\star > 2$ então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto, $x_{20}^\star \leq 2$. Analogamente, $x_2^\star \leq 2$.
- Não pode ter simultaneamente $x_2^\star = 2$ e $x_1^\star = 1$, pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto $2x_2^\star + x_1^\star \leq 4$.
  Também não tem simultaneamente $x_{20}^\star = 2$ e $x_{10}^\star = 1$.
- Como $2x_2^\star + x_1^\star \leq 4$, $x_5^\star \leq 1$ e $x_{10}^\star \leq 1$ então $5x_5^\star + 2x_2^\star + x_1^\star \leq 9$ e $10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 19$. Analogamente, se deduz que $20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 49$, $50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 99$. $100x_{100}^\star + 50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 199$.

  Tem-se $\sum_{i=1}^{k} v_i x_{v_i}^\star < v_{k+1}$, para todo k. Portanto, $x^\star$ é a solução greedy. □

NB: A estratégia *greedy* apresentada não seria correta para, por exemplo, $V = \{1, 300, 1000\}$, $Q = 1200$.

Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se $\{200, 100, 50, 20, 10, 5, 2, 1\}$:

- Seja $x^\star$ uma solução ótima para a quantia Q. Seja $x_v^\star$ é o número de moedas que usa de valor v.
- Se $x_{100}^\star > 1$, a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto, $x_{100}^\star \leq 1$.
  Analogamente se conclui que: $x_{50}^\star \leq 1$, $x_{10}^\star \leq 1$, e $x_1^\star \leq 1$.
- Se $x_{20}^\star > 2$ então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto, $x_{20}^\star \leq 2$. Analogamente, $x_2^\star \leq 2$.
- Não pode ter simultaneamente $x_2^\star = 2$ e $x_1^\star = 1$, pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto $2x_2^\star + x_1^\star \leq 4$.
  Também não tem simultaneamente $x_{20}^\star = 2$ e $x_{10}^\star = 1$.
- Como $2x_2^\star + x_1^\star \leq 4$, $x_5^\star \leq 1$ e $x_{10}^\star \leq 1$ então $5x_5^\star + 2x_2^\star + x_1^\star \leq 9$ e $10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 19$. Analogamente, se deduz que $20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 49$, $50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 99$. $100x_{100}^\star + 50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 199$.

  Tem-se $\sum_{i=1}^{k} v_i x_{v_i}^\star < v_{k+1}$, para todo k. Portanto, $x^\star$ é a solução greedy. □

NB: A estratégia *greedy* apresentada não seria correta para, por exemplo, $V = \{1, 300, 1000\}$, $Q = 1200$.

Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se $\{200, 100, 50, 20, 10, 5, 2, 1\}$:

- Seja $x^\star$ uma solução ótima para a quantia Q. Seja $x_v^\star$ é o número de moedas que usa de valor v.
- Se $x_{100}^\star > 1$, a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto, $x_{100}^\star \leq 1$.
  Analogamente se conclui que: $x_{50}^\star \leq 1$, $x_{10}^\star \leq 1$, e $x_1^\star \leq 1$.
- Se $x_{20}^\star > 2$ então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto, $x_{20}^\star \leq 2$. Analogamente, $x_2^\star \leq 2$.
- Não pode ter simultaneamente $x_2^\star = 2$ e $x_1^\star = 1$, pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto $2x_2^\star + x_1^\star \leq 4$.
  Também não tem simultaneamente $x_{20}^\star = 2$ e $x_{10}^\star = 1$.
- Como $2x_2^\star + x_1^\star \leq 4$, $x_5^\star \leq 1$ e $x_{10}^\star \leq 1$ então $5x_5^\star + 2x_2^\star + x_1^\star \leq 9$ e $10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 19$. Analogamente, se deduz que $20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 49$, $50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 99$. $100x_{100}^\star + 50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 199$.

  Tem-se $\sum_{i=1}^{k} v_i x_{v_i}^\star < v_{k+1}$, para todo k. Portanto, $x^\star$ é a solução greedy. □

NB: A estratégia *greedy* apresentada não seria correta para, por exemplo, $V = \{1, 300, 1000\}$, $Q = 1200$.

Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se $\{200, 100, 50, 20, 10, 5, 2, 1\}$:

- Seja $x^\star$ uma solução ótima para a quantia Q. Seja $x_v^\star$ é o número de moedas que usa de valor v.
- Se $x_{100}^\star > 1$, a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto, $x_{100}^\star \leq 1$.
  Analogamente se conclui que: $x_{50}^\star \leq 1$, $x_{10}^\star \leq 1$, e $x_1^\star \leq 1$.
- Se $x_{20}^\star > 2$ então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto, $x_{20}^\star \leq 2$. Analogamente, $x_2^\star \leq 2$.
- Não pode ter simultaneamente $x_2^\star = 2$ e $x_1^\star = 1$, pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto $2x_2^\star + x_1^\star \leq 4$.
  Também não tem simultaneamente $x_{20}^\star = 2$ e $x_{10}^\star = 1$.
- Como $2x_2^\star + x_1^\star \leq 4$, $x_5^\star \leq 1$ e $x_{10}^\star \leq 1$ então $5x_5^\star + 2x_2^\star + x_1^\star \leq 9$ e $10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 19$. Analogamente, se deduz que $20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 49$, $50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 99$. $100x_{100}^\star + 50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 199$.

  Tem-se $\sum_{i=1}^{k} v_i x_{v_i}^\star < v_{k+1}$, para todo k. Portanto, $x^\star$ é a solução greedy. □

NB: A estratégia *greedy* apresentada não seria correta para, por exemplo, $V = \{1, 300, 1000\}$, $Q = 1200$.

Problemas de trocos com número de moedas ilimitado

Prova de que a estratégia greedy obtém a solução ótima se $\{200, 100, 50, 20, 10, 5, 2, 1\}$:

- Seja $x^\star$ uma solução ótima para a quantia Q. Seja $x_v^\star$ é o número de moedas que usa de valor v.
- Se $x_{100}^\star > 1$, a solução não seria ótima (podia reduzir o número de moedas se substituir duas de 100 por uma de 200). Portanto, $x_{100}^\star \leq 1$.
  Analogamente se conclui que: $x_{50}^\star \leq 1$, $x_{10}^\star \leq 1$, e $x_1^\star \leq 1$.
- Se $x_{20}^\star > 2$ então a solução não seria ótima porque podia trocar três moedas de 20 por uma de 50 e uma de 10. Portanto, $x_{20}^\star \leq 2$. Analogamente, $x_2^\star \leq 2$.
- Não pode ter simultaneamente $x_2^\star = 2$ e $x_1^\star = 1$, pois a solução não seria ótima (podia substituir essas três moedas por uma de 5). Portanto $2x_2^\star + x_1^\star \leq 4$.
  Também não tem simultaneamente $x_{20}^\star = 2$ e $x_{10}^\star = 1$.
- Como $2x_2^\star + x_1^\star \leq 4$, $x_5^\star \leq 1$ e $x_{10}^\star \leq 1$ então $5x_5^\star + 2x_2^\star + x_1^\star \leq 9$ e $10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 19$. Analogamente, se deduz que $20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 49$, $50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 99$. $100x_{100}^\star + 50x_{50}^\star + 20x_{20}^\star + 10x_{10}^\star + 5x_5^\star + 2x_2^\star + x_1^\star \leq 199$.

  Tem-se $\sum_{i=1}^{k} v_i x_{v_i}^\star < v_{k+1}$, para todo k. Portanto, $x^\star$ é a solução greedy. □

NB: A estratégia *greedy* apresentada não seria correta para, por exemplo, $V = \{1, 300, 1000\}$, $Q = 1200$.

Problema da mochila (*knapsack problem*)

a) *Knapsack binário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \{0,1\} \end{cases}$$

b) *Knapsack inteiro*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \mathbb{Z}_0^+ \end{cases}$$

c) *Knapsack fracionário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad p_i x_i \leq u_i \wedge x_i \in \mathbb{R}_0^+ \end{cases}$$

Exemplo

p	peso (Kg)	42	32	12	20	27
v	valor (u.m.)	90	82	37	61	70
	máximo (Kg)	35	60	30	25	20

(a) Para um limite de carga $L = 80Kg$, dados p_i e v_i para cada cada objeto i, que objetos transporta para maximizar o valor total? ($x_2 = 1$, $x_4 = x_5 = 1$) (b) E, se puder transportar vários idênticos? ($x_3 = 5$, $x_4 = 1$) (c) E, se puder fracioná-los, sendo o valor e o peso proporcionais à fracção que leva, não podendo exceder um limite máximo dado para cada tipo? ($x_3 = 30/12$, $x_4 = 25/20$, $x_5 = 20/27$, $x_2 = 5/32$)

Problema da mochila (*knapsack problem*)

a) *Knapsack binário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \{0,1\} \end{cases}$

b) *Knapsack inteiro*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \mathbb{Z}_0^+ \end{cases}$

c) *Knapsack fracionário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad p_i x_i \leq u_i \wedge x_i \in \mathbb{R}_0^+ \end{cases}$

Exemplo

p	peso (Kg)	42	32	12	20	27
v	valor (u.m.)	90	82	37	61	70
	máximo (Kg)	35	60	30	25	20

(a) Para um limite de carga $L = 80Kg$, dados p_i e v_i para cada cada objeto i, que objetos transporta para maximizar o valor total? ($x_2 = 1$, $x_4 = x_5 = 1$) (b) E, se puder transportar vários idênticos? ($x_3 = 5$, $x_4 = 1$) (c) E, se puder fracioná-los, sendo o valor e o peso proporcionais à fracção que leva, não podendo exceder um limite máximo dado para cada tipo? ($x_3 = 30/12$, $x_4 = 25/20$, $x_5 = 20/27$, $x_2 = 5/32$)

Problema da mochila (*knapsack problem*)

a) *Knapsack binário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \{0,1\} \end{cases}$

b) *Knapsack inteiro*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \mathbb{Z}_0^+ \end{cases}$

c) *Knapsack fracionário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad p_i x_i \leq u_i \wedge x_i \in \mathbb{R}_0^+ \end{cases}$

Exemplo

p	peso (Kg)	42	32	12	20	27
v	valor (u.m.)	90	82	37	61	70
	máximo (Kg)	35	60	30	25	20

(a) Para um limite de carga $L = 80Kg$, dados p_i e v_i para cada cada objeto i, que objetos transporta para maximizar o valor total? ($x_2 = 1$, $x_4 = x_5 = 1$) (b) E, se puder transportar vários idênticos? ($x_3 = 5$, $x_4 = 1$) (c) E, se puder fracioná-los, sendo o valor e o peso proporcionais à fracção que leva, não podendo exceder um limite máximo dado para cada tipo? ($x_3 = 30/12$, $x_4 = 25/20$, $x_5 = 20/27$, $x_2 = 5/32$)

Problema da mochila (*knapsack problem*)

a) *Knapsack binário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \{0,1\} \end{cases}$

b) *Knapsack inteiro*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \mathbb{Z}_0^+ \end{cases}$

c) *Knapsack fracionário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad p_i x_i \leq u_i \wedge x_i \in \mathbb{R}_0^+ \end{cases}$

Exemplo

p	peso (Kg)	42	32	12	20	27
v	valor (u.m.)	90	82	37	61	70
	máximo (Kg)	35	60	30	25	20

(a) Para um limite de carga $L = 80Kg$, dados p_i e v_i para cada cada objeto i, que objetos transporta para maximizar o valor total? ($x_2 = 1$, $x_4 = x_5 = 1$) (b) E, se puder transportar vários idênticos? ($x_3 = 5$, $x_4 = 1$) (c) E, se puder fracioná-los, sendo o valor e o peso proporcionais à fracção que leva, não podendo exceder um limite máximo dado para cada tipo? ($x_3 = 30/12$, $x_4 = 25/20$, $x_5 = 20/27$, $x_2 = 5/32$)

Problema da mochila (*knapsack problem*)

a) *Knapsack binário*

$$\text{maximizar} \sum_{i=1}^{n} v_i x_i$$
$$\text{sujeito a} \begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \{0,1\} \end{cases}$$

b) *Knapsack inteiro*

$$\text{maximizar} \sum_{i=1}^{n} v_i x_i$$
$$\text{sujeito a} \begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \mathbb{Z}_0^+ \end{cases}$$

c) *Knapsack fracionário*

$$\text{maximizar} \sum_{i=1}^{n} v_i x_i$$
$$\text{sujeito a} \begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad p_i x_i \leq u_i \wedge x_i \in \mathbb{R}_0^+ \end{cases}$$

Exemplo

p	peso (Kg)	42	32	12	20	27
v	valor (u.m.)	90	82	37	61	70
	máximo (Kg)	35	60	30	25	20

(a) Para um limite de carga $L = 80Kg$, dados p_i e v_i para cada cada objeto i, que objetos transporta para maximizar o valor total? ($x_2 = 1$, $x_4 = x_5 = 1$) (b) E, se puder transportar vários idênticos? ($x_3 = 5$, $x_4 = 1$) (c) E, se puder fracioná-los, sendo o valor e o peso proporcionais à fracção que leva, não podendo exceder um limite máximo dado para cada tipo? ($x_3 = 30/12$, $x_4 = 25/20$, $x_5 = 20/27$, $x_2 = 5/32$)

Problema da mochila (*knapsack problem*)

a) *Knapsack binário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \{0,1\} \end{cases}$

b) *Knapsack inteiro*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad x_i \in \mathbb{Z}_0^+ \end{cases}$

c) *Knapsack fracionário*

maximizar $\sum_{i=1}^{n} v_i x_i$
sujeito a
$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \quad p_i x_i \leq u_i \wedge x_i \in \mathbb{R}_0^+ \end{cases}$

Exemplo

p	peso (Kg)	42	32	12	20	27
v	valor (u.m.)	90	82	37	61	70
	máximo (Kg)	35	60	30	25	20

(a) Para um limite de carga $L = 80Kg$, dados p_i e v_i para cada cada objeto i, que objetos transporta para maximizar o valor total? ($x_2 = 1$, $x_4 = x_5 = 1$) (b) E, se puder transportar vários idênticos? ($x_3 = 5$, $x_4 = 1$) (c) E, se puder fracioná-los, sendo o valor e o peso proporcionais à fracção que leva, não podendo exceder um limite máximo dado para cada tipo? ($x_3 = 30/12$, $x_4 = 25/20$, $x_5 = 20/27$, $x_2 = 5/32$)

Problema da mochila fracionário (*linear knapsack problem*)

$$\text{maximizar} \quad \sum_{i=1}^{n} v_i x_i \quad \text{sujeito a}$$
$$\begin{cases} \sum_{i=1}^{n} p_i x_i \leq L \\ \forall i \;\; x_i \leq u_i/p_i \wedge x_i \in \mathbb{R}_0^+ \end{cases}$$

Algoritmo **greedy** que calcula uma solução ótima para **knapsack fracionário:**

> *Ordenar os itens por ordem decrescente de* **valor por unidade de recurso despendida** *(i.e., por* v_i/p_i*). Levar a maior quantidade possível do primeiro item, i.e.,* $x_1 = \min(u_1/p_1, L/p_1)$*, e aplicar a mesma estratégia para* $i \geq 2$*, com peso máximo* $L - p_1 x_1$*.*

Ideia da Prova: se em vez de x_j usar $x_j - \frac{\varepsilon}{p_j}$ perde $\frac{v_j}{p_j}\varepsilon$ e pode repor no máximo $\frac{v_{j+1}}{p_{j+1}}\varepsilon$. Logo, perde $(\frac{v_j}{p_j} - \frac{v_{j+1}}{p_{j+1}})\varepsilon$.

Exemplo ($L = 80$, solução ótima: $x_3 = 30/12$, $x_4 = 25/20$, $x_5 = 20/27$, $x_2 = 5/32$)

p	peso (Kg)	42	32	12	20	27
v	valor (u.m.)	90	82	37	61	70
u	máximo (Kg)	35	60	30	25	20
v/p	rendimento (u.m/Kg)	2.14	2.56	3.08	3.05	2.59

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Uma resposta parcial...

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$.
O par $(S, \mathcal{F})$ designa-se por **matróide** sse satisfizer para todo A e B:

- (*Hereditariedade*) se $B \in \mathcal{F}$ e $A \subseteq B$ então $A \in \mathcal{F}$.
- (*Extensão*) Se $A, B \in \mathcal{F}$ e $|A| < |B|$ então $A \cup \{x\} \in \mathcal{F}$, para algum $x \in B$.

Os elementos de $\mathcal{F}$ designam-se por *subconjuntos independentes*.

Propriedade: Os conjuntos independentes **maximais** (para $\subseteq$) têm o mesmo cardinal.

Um matróide pesado é um matróide $(S, \mathcal{F})$ com uma função de peso $w : S \to \mathbb{R}^+$, sendo $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

Teorema

O problema da **determinação de $A \in \mathcal{F}$ com peso $w(A)$ máximo** pode ser resolvido pelo **"algoritmo greedy trivial"**: partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Uma resposta parcial. . .

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$.
O par $(S, \mathcal{F})$ designa-se por **matróide** sse satisfizer para todo A e B:

- (*Hereditariedade)* se $B \in \mathcal{F}$ e $A \subseteq B$ então $A \in \mathcal{F}$.
- (*Extensão)* Se $A, B \in \mathcal{F}$ e $|A| < |B|$ então $A \cup \{x\} \in \mathcal{F}$, para algum $x \in B$.

Os elementos de $\mathcal{F}$ designam-se por *subconjuntos independentes*.

Propriedade: Os conjuntos independentes **maximais** (para $\subseteq$) têm o mesmo cardinal.

Um matróide pesado é um matróide $(S, \mathcal{F})$ com uma função de peso $w : S \to \mathbb{R}^+$, sendo $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

Teorema

O problema da **determinação de $A \in \mathcal{F}$ com peso $w(A)$ máximo** pode ser resolvido pelo **"algoritmo greedy trivial"**: partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Uma resposta parcial...

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$.
O par $(S, \mathcal{F})$ designa-se por **matróide** sse satisfizer para todo A e B:

- (*Hereditariedade*) se $B \in \mathcal{F}$ e $A \subseteq B$ então $A \in \mathcal{F}$.
- (*Extensão*) Se $A, B \in \mathcal{F}$ e $|A| < |B|$ então $A \cup \{x\} \in \mathcal{F}$, para algum $x \in B$.

Os elementos de $\mathcal{F}$ designam-se por *subconjuntos independentes*.

Propriedade: Os conjuntos independentes **maximais** (para $\subseteq$) têm o mesmo cardinal.

Um matróide pesado é um matróide $(S, \mathcal{F})$ com uma função de peso $w : S \to \mathbb{R}^+$, sendo $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

Teorema

O problema da **determinação de $A \in \mathcal{F}$ com peso $w(A)$ máximo** pode ser resolvido pelo **"algoritmo greedy trivial"**: partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Uma resposta parcial. . .

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$.
O par $(S, \mathcal{F})$ designa-se por **matróide** sse satisfizer para todo A e B:

- (*Hereditariedade*) se $B \in \mathcal{F}$ e $A \subseteq B$ então $A \in \mathcal{F}$.
- (*Extensão*) Se $A, B \in \mathcal{F}$ e $|A| < |B|$ então $A \cup \{x\} \in \mathcal{F}$, para algum $x \in B$.

Os elementos de $\mathcal{F}$ designam-se por *subconjuntos independentes*.

Propriedade: Os conjuntos independentes **maximais** (para $\subseteq$) têm o mesmo cardinal.

Um matróide pesado é um matróide $(S, \mathcal{F})$ com uma função de peso $w : S \to \mathbb{R}^+$, sendo $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

Teorema

O problema da **determinação de $A \in \mathcal{F}$ com peso $w(A)$ máximo** pode ser resolvido pelo **"algoritmo greedy trivial"**: partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Teorema (Rado 1957 / Gale 1968)

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$, e $w : S \to \mathbb{R}^+$ uma função de peso. Defina-se $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

- Se $(S, \mathcal{F})$ é um matróide pesado, com função de peso $w : S \to \mathbb{R}^+$, o problema de determinar $A \in \mathcal{F}$ com peso $w(A)$ máximo pode ser resolvido pelo "Algoritmo Greedy Trivial".
- Se o "Algoritmo Greedy Trivial" determinar uma solução ótima para **todas** as funções de peso w, então $(S, \mathcal{F})$ é um matróide.

"Algoritmo Greedy Trivial": partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Existem problemas que podem ser resolvidos por estratégias greedy e não têm estrutura de matróide pesado.
Por exemplo, "interval scheduling".

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Teorema (Rado 1957 / Gale 1968)

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$, e $w : S \to \mathbb{R}^+$ uma função de peso. Defina-se $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

- Se $(S, \mathcal{F})$ é um matróide pesado, com função de peso $w : S \to \mathbb{R}^+$, o problema de determinar $A \in \mathcal{F}$ com peso $w(A)$ máximo pode ser resolvido pelo "Algoritmo Greedy Trivial".
- Se o "Algoritmo Greedy Trivial" determinar uma solução ótima para **todas** as funções de peso w, então $(S, \mathcal{F})$ é um matróide.

"Algoritmo Greedy Trivial": partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Existem problemas que podem ser resolvidos por estratégias greedy e não têm estrutura de matróide pesado.
Por exemplo, "interval scheduling".

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Teorema (Rado 1957 / Gale 1968)

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$, e $w : S \to \mathbb{R}^+$ uma função de peso. Defina-se $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

- Se $(S, \mathcal{F})$ é um matróide pesado, com função de peso $w : S \to \mathbb{R}^+$, o problema de determinar $A \in \mathcal{F}$ com peso $w(A)$ máximo pode ser resolvido pelo "Algoritmo Greedy Trivial".
- Se o "Algoritmo Greedy Trivial" determinar uma solução ótima para **todas** as funções de peso w, então $(S, \mathcal{F})$ é um matróide.

"Algoritmo Greedy Trivial": partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Existem problemas que podem ser resolvidos por estratégias greedy e não têm estrutura de matróide pesado.
Por exemplo, "interval scheduling".

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Teorema (Rado 1957 / Gale 1968)

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$, e $w : S \to \mathbb{R}^+$ uma função de peso. Defina-se $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

- Se $(S, \mathcal{F})$ é um matróide pesado, com função de peso $w : S \to \mathbb{R}^+$, o problema de determinar $A \in \mathcal{F}$ com peso $w(A)$ máximo pode ser resolvido pelo "Algoritmo Greedy Trivial".
- Se o "Algoritmo Greedy Trivial" determinar uma solução ótima para **todas** as funções de peso w, então $(S, \mathcal{F})$ é um matróide.

"Algoritmo Greedy Trivial": partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Existem problemas que podem ser resolvidos por estratégias greedy e não têm estrutura de matróide pesado.
Por exemplo, "interval scheduling".

Como saber se um problema pode ser resolvido por um algoritmo greedy?

Teorema (Rado 1957 / Gale 1968)

Seja S um conjunto **finito** e $\mathcal{F}$ uma família de subconjuntos de S tal que $\mathcal{F} \neq \emptyset$, e $w : S \to \mathbb{R}^+$ uma função de peso. Defina-se $w(A) = \sum_{a \in A} w(a)$, para todo $A \subseteq S$.

- Se $(S, \mathcal{F})$ é um matróide pesado, com função de peso $w : S \to \mathbb{R}^+$, o problema de determinar $A \in \mathcal{F}$ com peso $w(A)$ máximo pode ser resolvido pelo "Algoritmo Greedy Trivial".
- Se o "Algoritmo Greedy Trivial" determinar uma solução ótima para **todas** as funções de peso w, então $(S, \mathcal{F})$ é um matróide.

"Algoritmo Greedy Trivial": partir de $A = \emptyset$ e, tomando os elementos $x \in S$ por ordem decrescente de peso, inserir x em A se $A \cup \{x\} \in \mathcal{F}$.

Existem problemas que podem ser resolvidos por estratégias greedy e não têm estrutura de matróide pesado.
Por exemplo, "interval scheduling".

Exemplos de matróides

Exemplo 1

Seja $G = (V, E)$ um grafo finito não dirigido.
Seja $S = E$ e $\mathcal{F} = \{E' \mid G' = (V, E') \text{ é acíclico, } E' \subseteq E\}$

- Se $B \subseteq E$ tal que $G_B = (V, B)$ é um subgrafo acíclico de G então para todo $A \subseteq B$ também $G_A = (V, A)$ é acíclico.
- Se A e B são subconjuntos de E tais que os subgrafos $G_A = (V, A)$ e $G_B = (V, B)$ são acíclicos e $|A| < |B|$ então existe $e \in B$ tal que $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico porque:
  - Se $|A| < |B|$, o número de componentes conexas de G_A é maior do que o número de componentes conexas de G_B (pois, G_A e G_B são florestas).
  - Pelo princípio de Pigeonhole, existem nós u e v tais que u e v estão na mesma componente conexa em G_B, a aresta $(u, v) \in B$ e u e v estão em componentes conexas distintas em G_A.
  - Assim, para $e = (u, v)$, o grafo $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico. □

Exemplos de matróides

Exemplo 1

Seja $G = (V, E)$ um grafo finito não dirigido.
Seja $S = E$ e $\mathcal{F} = \{E' \mid G' = (V, E') \text{ é acíclico, } E' \subseteq E\}$

- Se $B \subseteq E$ tal que $G_B = (V, B)$ é um subgrafo acíclico de G então para todo $A \subseteq B$ também $G_A = (V, A)$ é acíclico.
- Se A e B são subconjuntos de E tais que os subgrafos $G_A = (V, A)$ e $G_B = (V, B)$ são acíclicos e $|A| < |B|$ então existe $e \in B$ tal que $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico porque:
  - Se $|A| < |B|$, o número de componentes conexas de G_A é maior do que o número de componentes conexas de G_B (pois, G_A e G_B são florestas).
  - Pelo princípio de Pigeonhole, existem nós u e v tais que u e v estão na mesma componente conexa em G_B, a aresta $(u, v) \in B$ e u e v estão em componentes conexas distintas em G_A.
  - Assim, para $e = (u, v)$, o grafo $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico. □

Exemplos de matróides

Exemplo 1

Seja $G = (V, E)$ um grafo finito não dirigido.
Seja $S = E$ e $\mathcal{F} = \{E' \mid G' = (V, E') \text{ é acíclico, } E' \subseteq E\}$

- Se $B \subseteq E$ tal que $G_B = (V, B)$ é um subgrafo acíclico de G então para todo $A \subseteq B$ também $G_A = (V, A)$ é acíclico.
- Se A e B são subconjuntos de E tais que os subgrafos $G_A = (V, A)$ e $G_B = (V, B)$ são acíclicos e $|A| < |B|$ então existe $e \in B$ tal que $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico porque:
  - Se $|A| < |B|$, o número de componentes conexas de G_A é maior do que o número de componentes conexas de G_B (pois, G_A e G_B são florestas).
  - Pelo princípio de Pigeonhole, existem nós u e v tais que u e v estão na mesma componente conexa em G_B, a aresta $(u, v) \in B$ e u e v estão em componentes conexas distintas em G_A.
  - Assim, para $e = (u, v)$, o grafo $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico. □

Exemplos de matróides

Exemplo 1

Seja $G = (V, E)$ um grafo finito não dirigido.
Seja $S = E$ e $\mathcal{F} = \{E' \mid G' = (V, E') \text{ é acíclico, } E' \subseteq E\}$

- Se $B \subseteq E$ tal que $G_B = (V, B)$ é um subgrafo acíclico de G então para todo $A \subseteq B$ também $G_A = (V, A)$ é acíclico.
- Se A e B são subconjuntos de E tais que os subgrafos $G_A = (V, A)$ e $G_B = (V, B)$ são acíclicos e $|A| < |B|$ então existe $e \in B$ tal que $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico porque:
  - Se $|A| < |B|$, o número de componentes conexas de G_A é maior do que o número de componentes conexas de G_B (pois, G_A e G_B são florestas).
  - Pelo princípio de Pigeonhole, existem nós u e v tais que u e v estão na mesma componente conexa em G_B, a aresta $(u, v) \in B$ e u e v estão em componentes conexas distintas em G_A.
  - Assim, para $e = (u, v)$, o grafo $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico. □

Exemplos de matróides

Exemplo 1

Seja $G = (V, E)$ um grafo finito não dirigido.
Seja $S = E$ e $\mathcal{F} = \{E' \mid G' = (V, E') \text{ é acíclico, } E' \subseteq E\}$

- Se $B \subseteq E$ tal que $G_B = (V, B)$ é um subgrafo acíclico de G então para todo $A \subseteq B$ também $G_A = (V, A)$ é acíclico.
- Se A e B são subconjuntos de E tais que os subgrafos $G_A = (V, A)$ e $G_B = (V, B)$ são acíclicos e $|A| < |B|$ então existe $e \in B$ tal que $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico porque:
  - Se $|A| < |B|$, o número de componentes conexas de G_A é maior do que o número de componentes conexas de G_B (pois, G_A e G_B são florestas).
  - Pelo princípio de Pigeonhole, existem nós u e v tais que u e v estão na mesma componente conexa em G_B, a aresta $(u, v) \in B$ e u e v estão em componentes conexas distintas em G_A.
  - Assim, para $e = (u, v)$, o grafo $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico. □

Exemplos de matróides

Exemplo 1

Seja $G = (V, E)$ um grafo finito não dirigido.
Seja $S = E$ e $\mathcal{F} = \{E' \mid G' = (V, E') \text{ é acíclico, } E' \subseteq E\}$

- Se $B \subseteq E$ tal que $G_B = (V, B)$ é um subgrafo acíclico de G então para todo $A \subseteq B$ também $G_A = (V, A)$ é acíclico.
- Se A e B são subconjuntos de E tais que os subgrafos $G_A = (V, A)$ e $G_B = (V, B)$ são acíclicos e $|A| < |B|$ então existe $e \in B$ tal que $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico porque:
  - Se $|A| < |B|$, o número de componentes conexas de G_A é maior do que o número de componentes conexas de G_B (pois, G_A e G_B são florestas).
  - Pelo princípio de Pigeonhole, existem nós u e v tais que u e v estão na mesma componente conexa em G_B, a aresta $(u, v) \in B$ e u e v estão em componentes conexas distintas em G_A.
  - Assim, para $e = (u, v)$, o grafo $G_{A \cup \{e\}} = (V, A \cup \{e\})$ é acíclico. □

Exemplos de matróides

Exemplo 2

$S = \{\text{colunas da matriz de coeficientes de um sistema } AX = b\}$,
$\mathcal{F} = \{\text{subconjuntos de colunas de } A \text{ linearmente independentes}\}$.

$(S, \mathcal{F})$ é matróide porque é conhecido da Álgebra Linear que:

- Qualquer subconjunto de um conjunto de vetores linearmente independentes de $\mathbb{R}^m$ é linearmente independente.
- Se $\mathcal{A}$ e $\mathcal{B}$ são conjuntos de vetores linearmente independentes de $\mathbb{R}^m$ e $|\mathcal{A}| < |\mathcal{B}|$ então existe $\mathcal{B}_j \in \mathcal{B}$ tal que $\mathcal{A} \cup \{\mathcal{B}_j\}$ é linearmente independente.

Exemplos de matróides

Exemplo 2

$S = \{\text{colunas da matriz de coeficientes de um sistema } AX = b\}$,
$\mathcal{F} = \{\text{subconjuntos de colunas de } A \text{ linearmente independentes}\}$.

$(S, \mathcal{F})$ é matróide porque é conhecido da Álgebra Linear que:

- Qualquer subconjunto de um conjunto de vetores linearmente independentes de $\mathbb{R}^m$ é linearmente independente.
- Se $\mathcal{A}$ e $\mathcal{B}$ são conjuntos de vetores linearmente independentes de $\mathbb{R}^m$ e $|\mathcal{A}| < |\mathcal{B}|$ então existe $\mathcal{B}_j \in \mathcal{B}$ tal que $\mathcal{A} \cup \{\mathcal{B}_j\}$ é linearmente independente.

Exemplos de matróides

Exemplo 3

$S = \{\text{tarefas de duração unitária cada uma com um prazo limite}\}$,
$\mathcal{F} = \{A \mid A \subseteq S \text{ e é possível executar todas as tarefas em } A \text{ dentro dos prazos}\}$.

$(S, \mathcal{F})$ é matróide:

- Se $B \in \mathcal{F}$ então $A \in \mathcal{F}$, para todo $A \subseteq B$. Trivial.
- Se $\mathcal{A} \in \mathcal{F}$ e $\mathcal{B} \in \mathcal{F}$ e $|\mathcal{A}| < |\mathcal{B}|$ então existe $b \in \mathcal{B}$ tal que $\mathcal{A} \cup \{b\} \in \mathcal{F}$.

  Prova: Seja $N_t(X)$ número de tarefas em X com prazo até t.

  Tem-se $N_0(A) = N_0(B) = 0$ e $N_n(A) < N_n(B)$, sendo $n = |S|$.

  Seja k o maior instante tal que $N_k(B) \leq N_k(A)$. Para todo $j \geq k+1$ tem-se $N_j(A) < N_j(B)$. Logo, B tem mais tarefas com prazo limite $k+1$ do que tem A. Tome-se um $b \in B$ com prazo limite $k+1$ tal que $b \notin A$. Podemos provar que $A \cup \{b\} \in \mathcal{F}$, pois $N_j(A \cup \{b\}) \leq N_j(A) \leq j$, para $j < k+1$. E, $N_j(A \cup \{b\}) = N_j(A) + 1 \leq N_j(B) \leq j$, para todo $j \geq k+1$.

Exemplos de matróides

Exemplo 3

$S = \{\text{tarefas de duração unitária cada uma com um prazo limite}\}$,
$\mathcal{F} = \{A \mid A \subseteq S \text{ e é possível executar todas as tarefas em } A \text{ dentro dos prazos}\}$.

$(S, \mathcal{F})$ é matróide:

- Se $B \in \mathcal{F}$ então $A \in \mathcal{F}$, para todo $A \subseteq B$. Trivial.
- Se $\mathcal{A} \in \mathcal{F}$ e $\mathcal{B} \in \mathcal{F}$ e $|\mathcal{A}| < |\mathcal{B}|$ então existe $b \in \mathcal{B}$ tal que $\mathcal{A} \cup \{b\} \in \mathcal{F}$.

  Prova: Seja $N_t(X)$ número de tarefas em X com prazo até t.

  Tem-se $N_0(A) = N_0(B) = 0$ e $N_n(A) < N_n(B)$, sendo $n = |S|$.

  Seja k o maior instante tal que $N_k(B) \leq N_k(A)$. Para todo $j \geq k+1$ tem-se $N_j(A) < N_j(B)$. Logo, B tem mais tarefas com prazo limite $k+1$ do que tem A. Tome-se um $b \in B$ com prazo limite $k+1$ tal que $b \notin A$. Podemos provar que $A \cup \{b\} \in \mathcal{F}$, pois $N_j(A \cup \{b\}) \leq N_j(A) \leq j$, para $j < k+1$. E, $N_j(A \cup \{b\}) = N_j(A) + 1 \leq N_j(B) \leq j$, para todo $j \geq k+1$.

Exemplos de Aplicação - Otimização em matróides pesados

- **Exemplo 1:** Determinar uma árvore geradora de peso máximo/mínimo – Algoritmo de Kruskal

  Se G for conexo, o "algoritmo greedy trivial" reduz-se ao algoritmo de Kruskal.

- **Exemplo 2:** Localizar observadores em rotundas para determinar os volumes de tráfego q_{ij}, da entrada i para a saída j, para todos os pares (i, j)

  São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 (i.e., F_1). Assume-se que os veículos não estacionam no anel de circulação. Se colocar observador para q_{ij} tem um custo c_{ij}. Minimizar o custo total.

  O "algoritmo greedy trivial" pode ser usado para resolver o problema (detalhes à frente).

- **Exemplo 3:** Dado um conjunto finito de **tarefas unitárias cada uma com um prazo limite** (*deadline*) d_j e uma **penalização** c_j se ultrapassar esse prazo, determinar a ordem pela qual as tarefas serão realizadas de forma a minimizar o custo (penalização) total.

  O "algoritmo greedy trivial" pode ser usado para resolver o problema (detalhes à frente).

Exemplos de Aplicação - Otimização em matróides pesados

- **Exemplo 1:** Determinar uma árvore geradora de peso máximo/mínimo – Algoritmo de Kruskal

  Se G for conexo, o "algoritmo greedy trivial" reduz-se ao algoritmo de Kruskal.

- **Exemplo 2:** Localizar observadores em rotundas para determinar os volumes de tráfego q_{ij}, da entrada i para a saída j, para todos os pares (i, j)

  São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 (i.e., F_1). Assume-se que os veículos não estacionam no anel de circulação. Se colocar observador para q_{ij} tem um custo c_{ij}. Minimizar o custo total.

  O "algoritmo greedy trivial" pode ser usado para resolver o problema (detalhes à frente).

- **Exemplo 3:** Dado um conjunto finito de **tarefas unitárias cada uma com um prazo limite** (*deadline*) d_j e uma **penalização** c_j se ultrapassar esse prazo, determinar a ordem pela qual as tarefas serão realizadas de forma a minimizar o custo (penalização) total.

  O "algoritmo greedy trivial" pode ser usado para resolver o problema (detalhes à frente).

Exemplos de Aplicação - Otimização em matróides pesados

- **Exemplo 1:** Determinar uma árvore geradora de peso máximo/mínimo – Algoritmo de Kruskal

  Se G for conexo, o "algoritmo greedy trivial" reduz-se ao algoritmo de Kruskal.

- **Exemplo 2:** Localizar observadores em rotundas para determinar os volumes de tráfego q_{ij}, da entrada i para a saída j, para todos os pares (i, j)

  São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 (i.e., F_1). Assume-se que os veículos não estacionam no anel de circulação. Se colocar observador para q_{ij} tem um custo c_{ij}. Minimizar o custo total.

  O "algoritmo greedy trivial" pode ser usado para resolver o problema (detalhes à frente).

- **Exemplo 3:** Dado um conjunto finito de **tarefas unitárias cada uma com um prazo limite** (*deadline*) d_j e uma **penalização** c_j se ultrapassar esse prazo, determinar a ordem pela qual as tarefas serão realizadas de forma a minimizar o custo (penalização) total.

  O "algoritmo greedy trivial" pode ser usado para resolver o problema (detalhes à frente).

Exemplo 3: Unit task scheduling (matróide pesado)

Unit task scheduling

Dado um conjunto $\mathcal{T}$ de tarefas **com duração 1**, cada uma com um **prazo** d_j (*deadline*) e uma **penalização** c_j, se ultrapassar esse prazo, por que ordem as realizar de forma a minimizar a penalização total?

- Um conjunto A de tarefas é **independente** se todas as tarefas em A podem ser executadas até ao seu *deadline*. Prova-se que tal acontece se, para todo $k \geq 0$, o número de tarefas em A com *deadline* até k é menor ou igual a k.
- Resolve-se usando o "Algoritmo greedy trivial": ordenar as tarefas por **ordem decrescente de penalização** (supor que $t_1, t_2, \ldots, t_n$ traduz essa ordem). No início, $S = \emptyset$. Para j de 1 até n, colocar t_j em S desde que $S \cup \{t_j\}$ seja *independente*.
- **As tarefas em S podem ser realizadas sem penalização** (por exemplo, se as realizar por ordem crescente de *deadline*). As tarefas em $\mathcal{T} \setminus S$ são realizadas por qualquer ordem (têm sempre penalização).

Exemplo 3: Unit task scheduling (matróide pesado)

Unit task scheduling

Dado um conjunto $\mathcal{T}$ de tarefas **com duração 1**, cada uma com um **prazo** d_j (*deadline*) e uma **penalização** c_j, se ultrapassar esse prazo, por que ordem as realizar de forma a minimizar a penalização total?

- Um conjunto A de tarefas é **independente** se todas as tarefas em A podem ser executadas até ao seu *deadline*. Prova-se que tal acontece se, para todo $k \geq 0$, o número de tarefas em A com *deadline* até k é menor ou igual a k.
- Resolve-se usando o "Algoritmo greedy trivial": ordenar as tarefas por **ordem decrescente de penalização** (supor que $t_1, t_2, \ldots, t_n$ traduz essa ordem). No início, $S = \emptyset$. Para j de 1 até n, colocar t_j em S desde que $S \cup \{t_j\}$ seja *independente*.
- **As tarefas em S podem ser realizadas sem penalização** (por exemplo, se as realizar por ordem crescente de *deadline*). As tarefas em $\mathcal{T} \setminus S$ são realizadas por qualquer ordem (têm sempre penalização).

Exemplo 3: Unit task scheduling (matróide pesado)

Unit task scheduling

Dado um conjunto $\mathcal{T}$ de tarefas **com duração 1**, cada uma com um **prazo** d_j (*deadline*) e uma **penalização** c_j, se ultrapassar esse prazo, por que ordem as realizar de forma a minimizar a penalização total?

- Um conjunto A de tarefas é **independente** se todas as tarefas em A podem ser executadas até ao seu *deadline*. Prova-se que tal acontece se, para todo $k \geq 0$, o número de tarefas em A com *deadline* até k é menor ou igual a k.
- Resolve-se usando o "Algoritmo greedy trivial": ordenar as tarefas por **ordem decrescente de penalização** (supor que $t_1, t_2, \ldots, t_n$ traduz essa ordem). No início, $S = \emptyset$. Para j de 1 até n, colocar t_j em S desde que $S \cup \{t_j\}$ seja *independente*.
- **As tarefas em S podem ser realizadas sem penalização** (por exemplo, se as realizar por ordem crescente de *deadline*). As tarefas em $\mathcal{T} \setminus S$ são realizadas por qualquer ordem (têm sempre penalização).

Exemplo 3: Unit task scheduling (matróide pesado)

Unit task scheduling

Dado um conjunto $\mathcal{T}$ de tarefas **com duração 1**, cada uma com um **prazo** d_j (*deadline*) e uma **penalização** c_j, se ultrapassar esse prazo, por que ordem as realizar de forma a minimizar a penalização total?

- Um conjunto A de tarefas é **independente** se todas as tarefas em A podem ser executadas até ao seu *deadline*. Prova-se que tal acontece se, para todo $k \geq 0$, o número de tarefas em A com *deadline* até k é menor ou igual a k.
- Resolve-se usando o "Algoritmo greedy trivial": ordenar as tarefas por **ordem decrescente de penalização** (supor que $t_1, t_2, \ldots, t_n$ traduz essa ordem). No início, $S = \emptyset$. Para j de 1 até n, colocar t_j em S desde que $S \cup \{t_j\}$ seja *independente*.
- **As tarefas em S podem ser realizadas sem penalização** (por exemplo, se as realizar por ordem crescente de *deadline*). As tarefas em $\mathcal{T} \setminus S$ são realizadas por qualquer ordem (têm sempre penalização).

Exemplo 3: Unit task scheduling (matróide pesado)

Implementação $O(n^2)$:

1. Ordenar o conjunto de tarefas por ordem decrescente de penalização. Seja $\mathcal{T} = \{t_1, \ldots, t_n\}$ o conjunto ordenado.
2. Seja B um *array* de booleanos tal que $B[k]$ indica se o *slot* k está livre.
3. Para $i \leftarrow 1$ até n, atribuir à tarefa t_i o intervalo de tempo mais tardio que esteja livre e não exceda o deadline d_i, caso exista.

Exemplo 3: Unit task scheduling (matróide pesado)

Uma solução mais rápida (com recolha de informação que pode encurtar pesquisas futuras):

1. Ordenar o conjunto de tarefas por ordem decrescente de penalização. Seja $\mathcal{T} = \{t_1, \ldots, t_n\}$ o conjunto ordenado.
2. Seja S um *array* em que $S[k]$ indica o intervalo de tempo mais tardio que pode estar livre e não excede k. Inicialmente, $S[k] \leftarrow k$ para $k \geq 0$.
3. Para $i \leftarrow 1$ até n, atribuir à tarefa t_i o intervalo mais tardio com $S[k] = k$ para $k \leq d_i$. Para isso:
   1. começar com $k \leftarrow d_i$ e enquanto $S[k] \neq k$ fazer $S[k] \leftarrow S[S[k]]$ e $k \leftarrow S[k]$;
   2. o ciclo termina com $S[k] = k$. Nesse momento, se $k \neq 0$ então t_i fica selecionada e $S[k] \leftarrow S[k-1]$ e $S[d_i] \leftarrow S[k-1]$. Se $k = 0$ então t_i não é selecionada e $S[d_i] \leftarrow 0$.

Invariante: $S[0] = 0$ em todas as iterações; Se $S[k] = k$, o slot k está livre. Se $S[k] \neq k$, os slots desde $S[S[k]] + 1$ até k estão ocupados (logo, se necessitar de agendar uma tarefa com deadline k teria de procurar um slot até $S[S[k]]$).

Exemplo 3: Unit task scheduling (matróide pesado)

Exemplo de execução: Suponhamos que as tarefas estão já ordenadas por ordem decrescente de penalização e que os seus *deadlines* são:

d		3	3	5	7	3	3	4	5	6	7	8	2	5	6	1

S	0	1	2	3	4	5	6	7	8
S	0	1	2	2	4	5	6	7	8
S	0	1	1	1	4	5	6	7	8
S	0	1	1	1	4	4	6	7	8
S	0	1	1	1	4	4	6	6	8
S	0	0	1	0	4	4	6	6	8
S	0	0	1	0	0	4	6	6	8
S	0	0	1	0	0	0	6	6	8
S	0	0	1	0	0	0	0	6	8
S	0	0	1	0	0	0	0	0	8
S	0	0	1	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0

t_1 ficou no slot 3
t_2 ficou no slot 2
t_3 ficou no slot 5
t_4 ficou no slot 7
t_5 ficou no slot 1
t_6 não entra; t_7 ficou no slot 4
t_8 não entra (tinha $S[5] = 4$ e $S[4] = 0$)
t_9 no slot 6
t_{10} não entra (tinha $S[7] = 6$ e $S[6] = 0$)
t_{11} ficou no slot 8
t_{12} não entra (tinha $S[2] = 1$ e $S[1] = 0$)

Melhoramento: Se ficar com $S[MaxTime] = 0$, sendo $MaxTime = max_i(d_i)$, então nenhuma das tarefas que restam pode entrar... o que quer dizer que t_{12} não teria de ser analisada nem as seguintes!

Exemplo (não matróide pesado): Interval scheduling

Interval scheduling

$\mathcal{T}$ é um conjunto de n tarefas. A tarefa t_j teria forçosamente de decorrer no intervalo $[a_j, b_j[$, ou seja, começar no instante a_j e terminar em b_j, para $1 \leq j \leq n$ (notar que $b_j \notin [a_j, b_j[$). Em cada instante, só uma tarefa pode estar a decorrer. Pretende-se **maximizar o número de tarefas realizadas**.

- Não tem estrutura de matróide. Mas, calculamos uma solução ótima em tempo $O(n \log n)$ por um algoritmo *greedy*, usando a estratégia "earliest finish first".

```
Ordenar T por ordem crescente de tempo de finalização;
 /* Supor que t_1, t_2, ..., t_n traduz essa ordem */
S ← ∅;  f ← 0;
Para j ← 1 até n fazer
    Se a[j] ≥ f então
        S ← S ∪ {t_j};  f ← b[j];
```

- **Correção:** Seja $S^\star$ uma solução ótima distinta da solução S calculada pelo algoritmo. Sejam k e j as primeiras duas tarefas que as distinguem. Então, t_j (a escolha greedy) pode substituir t_k em $S^\star$, ou seja, $(S^\star \setminus \{t_k\}) \cup \{t_j\}$ é também uma solução ótima (t_j não pode criar conflitos pois $b[j] \leq b[k]$). Assim, repetindo, acabamos por conseguir transformar qualquer solução ótima $S^\star$ na solução greedy S, pelo que S é ótima.

Exemplo (não matróide pesado): Interval scheduling

Interval scheduling

$\mathcal{T}$ é um conjunto de n tarefas. A tarefa t_j teria forçosamente de decorrer no intervalo $[a_j, b_j[$, ou seja, começar no instante a_j e terminar em b_j, para $1 \leq j \leq n$ (notar que $b_j \notin [a_j, b_j[$). Em cada instante, só uma tarefa pode estar a decorrer. Pretende-se **maximizar o número de tarefas realizadas**.

- Não tem estrutura de matróide. Mas, calculamos uma solução ótima em tempo $O(n \log n)$ por um algoritmo *greedy*, usando a estratégia "earliest finish first".

```
Ordenar T por ordem crescente de tempo de finalização;
 /* Supor que t_1, t_2, ..., t_n traduz essa ordem */
S ← ∅;  f ← 0;
Para j ← 1 até n fazer
    Se a[j] ≥ f então
        S ← S ∪ {t_j};  f ← b[j];
```

- **Correção:** Seja $S^\star$ uma solução ótima distinta da solução S calculada pelo algoritmo. Sejam k e j as primeiras duas tarefas que as distinguem. Então, t_j (a escolha greedy) pode substituir t_k em $S^\star$, ou seja, $(S^\star \setminus \{t_k\}) \cup \{t_j\}$ é também uma solução ótima (t_j não pode criar conflitos pois $b[j] \leq b[k]$). Assim, repetindo, acabamos por conseguir transformar qualquer solução ótima $S^\star$ na solução greedy S, pelo que S é ótima.

Exemplo (não matróide pesado): Interval scheduling

Interval scheduling

$\mathcal{T}$ é um conjunto de n tarefas. A tarefa t_j teria forçosamente de decorrer no intervalo $[a_j, b_j[$, ou seja, começar no instante a_j e terminar em b_j, para $1 \leq j \leq n$ (notar que $b_j \notin [a_j, b_j[$). Em cada instante, só uma tarefa pode estar a decorrer. Pretende-se **maximizar o número de tarefas realizadas**.

- Não tem estrutura de matróide. Mas, calculamos uma solução ótima em tempo $O(n \log n)$ por um algoritmo *greedy*, usando a estratégia "earliest finish first".

```
Ordenar T por ordem crescente de tempo de finalização;
 /* Supor que t_1, t_2, ..., t_n traduz essa ordem */
S ← ∅;  f ← 0;
Para j ← 1 até n fazer
    Se a[j] ≥ f então
        S ← S ∪ {t_j};  f ← b[j];
```

- **Correção:** Seja $S^\star$ uma solução ótima distinta da solução S calculada pelo algoritmo. Sejam k e j as primeiras duas tarefas que as distinguem. Então, t_j (a escolha greedy) pode substituir t_k em $S^\star$, ou seja, $(S^\star \setminus \{t_k\}) \cup \{t_j\}$ é também uma solução ótima (t_j não pode criar conflitos pois $b[j] \leq b[k]$). Assim, repetindo, acabamos por conseguir transformar qualquer solução ótima $S^\star$ na solução greedy S, pelo que S é ótima.

Exemplo (não matróide pesado): Interval scheduling

Interval scheduling

$\mathcal{T}$ é um conjunto de n tarefas. A tarefa t_j teria forçosamente de decorrer no intervalo $[a_j, b_j[$, ou seja, começar no instante a_j e terminar em b_j, para $1 \leq j \leq n$ (notar que $b_j \notin [a_j, b_j[$). Em cada instante, só uma tarefa pode estar a decorrer. Pretende-se **maximizar o número de tarefas realizadas**.

- Não tem estrutura de matróide. Mas, calculamos uma solução ótima em tempo $O(n \log n)$ por um algoritmo *greedy*, usando a estratégia "earliest finish first".

```
Ordenar T por ordem crescente de tempo de finalização;
 /* Supor que t_1, t_2, ..., t_n traduz essa ordem */
S ← ∅;  f ← 0;
Para j ← 1 até n fazer
    Se a[j] ≥ f então
        S ← S ∪ {t_j};  f ← b[j];
```

- **Correção:** Seja $S^\star$ uma solução ótima distinta da solução S calculada pelo algoritmo. Sejam k e j as primeiras duas tarefas que as distinguem. Então, t_j (a escolha greedy) pode substituir t_k em $S^\star$, ou seja, $(S^\star \setminus \{t_k\}) \cup \{t_j\}$ é também uma solução ótima (t_j não pode criar conflitos pois $b[j] \leq b[k]$). Assim, repetindo, acabamos por conseguir transformar qualquer solução ótima $S^\star$ na solução greedy S, pelo que S é ótima.

Localizar observadores em rotundas para deduzir volumes de tráfego direcionais q_{ij}' s (matróide pesado)

Problema: Localizar observadores em rotundas para obter os volumes de tráfego direcionais, q_{ij}, da entrada i para a saída j, para todos os pares (i, j). São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 F_1. Assume-se que os veículos não estacionam no anel de circulação.

$$\sum_{j \in \mathcal{D}} q_{ij} = O_i, \text{ para } i \in \mathcal{O}$$

$$\sum_{i \in \mathcal{O}} q_{ij} = D_j, \text{ para } j \in \mathcal{D}$$

$$\sum_{i \in \mathcal{O} \setminus \{1\}} \sum_{j \in \mathcal{D},\, 1 \prec j \preceq i} q_{ij} = F_1$$

Se colocar observador para q_{ij} tem um custo c_{ij}. **Pretendemos minimizar o custo total.** Quais dos q_{ij}'s não serão obtidos por observação, sendo calculados por resolução do sistema?

A **base ótima** pode ser calculada pelo "algoritmo greedy trivial".

Localizar observadores em rotundas para deduzir volumes de tráfego direcionais q_{ij}' s (matróide pesado)

Problema: Localizar observadores em rotundas para obter os volumes de tráfego direcionais, q_{ij}, da entrada i para a saída j, para todos os pares (i, j). São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 F_1.
Assume-se que os veículos não estacionam no anel de circulação.

$$\sum_{j \in \mathcal{D}} q_{ij} = O_i, \text{ para } i \in \mathcal{O}$$

$$\sum_{i \in \mathcal{O}} q_{ij} = D_j, \text{ para } j \in \mathcal{D}$$

$$\sum_{i \in \mathcal{O} \setminus \{1\}} \sum_{j \in \mathcal{D},\, 1 \prec j \preceq i} q_{ij} = F_1$$

Se colocar observador para q_{ij} tem um custo c_{ij}. **Pretendemos minimizar o custo total.** Quais dos q_{ij}'s não serão obtidos por observação, sendo calculados por resolução do sistema?
A **base ótima** pode ser calculada pelo "algoritmo greedy trivial".

Localizar observadores em rotundas para deduzir volumes de tráfego direcionais q_{ij}' s (matróide pesado)

Problema: Localizar observadores em rotundas para obter os volumes de tráfego direcionais, q_{ij}, da entrada i para a saída j, para todos os pares (i, j). São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 F_1. Assume-se que os veículos não estacionam no anel de circulação.

$$\sum_{j \in \mathcal{D}} q_{ij} = O_i, \text{ para } i \in \mathcal{O}$$

$$\sum_{i \in \mathcal{O}} q_{ij} = D_j, \text{ para } j \in \mathcal{D}$$

$$\sum_{i \in \mathcal{O} \setminus \{1\}} \sum_{j \in \mathcal{D},\, 1 \prec j \preceq i} q_{ij} = F_1$$

Se colocar observador para q_{ij} tem um custo c_{ij}. **Pretendemos minimizar o custo total.** Quais dos q_{ij}'s não serão obtidos por observação, sendo calculados por resolução do sistema?

A **base ótima** pode ser calculada pelo "algoritmo greedy trivial".

Localizar observadores em rotundas para deduzir volumes de tráfego direcionais q_{ij}' s (matróide pesado)

Problema: Localizar observadores em rotundas para obter os volumes de tráfego direcionais, q_{ij}, da entrada i para a saída j, para todos os pares (i, j). São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 F_1. Assume-se que os veículos não estacionam no anel de circulação.

$$\begin{aligned} \sum_{j \in \mathcal{D}} q_{ij} &= O_i, \text{ para } i \in \mathcal{O} \\ \sum_{i \in \mathcal{O}} q_{ij} &= D_j, \text{ para } j \in \mathcal{D} \\ \sum_{i \in \mathcal{O} \setminus \{1\}} \sum_{j \in \mathcal{D},\, 1 \prec j \preceq i} q_{ij} &= F_1 \end{aligned}$$

Se colocar observador para q_{ij} tem um custo c_{ij}. **Pretendemos minimizar o custo total.** Quais dos q_{ij}'s não serão obtidos por observação, sendo calculados por resolução do sistema?

A **base ótima** pode ser calculada pelo "algoritmo greedy trivial".

Localizar observadores em rotundas para deduzir volumes de tráfego direcionais q_{ij}' s (matróide pesado)

Problema: Localizar observadores em rotundas para obter os volumes de tráfego direcionais, q_{ij}, da entrada i para a saída j, para todos os pares (i, j). São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 F_1. Assume-se que os veículos não estacionam no anel de circulação.

$$\begin{aligned} \sum_{j \in \mathcal{D}} q_{ij} &= O_i, \text{ para } i \in \mathcal{O} \\ \sum_{i \in \mathcal{O}} q_{ij} &= D_j, \text{ para } j \in \mathcal{D} \\ \sum_{i \in \mathcal{O} \setminus \{1\}} \sum_{j \in \mathcal{D},\, 1 \prec j \preceq i} q_{ij} &= F_1 \end{aligned}$$

Se colocar observador para q_{ij} tem um custo c_{ij}. **Pretendemos minimizar o custo total.** Quais dos q_{ij}'s não serão obtidos por observação, sendo calculados por resolução do sistema?

A **base ótima** pode ser calculada pelo "algoritmo greedy trivial".

Localizar observadores em rotundas para deduzir volumes de tráfego direcionais q_{ij}' s (matróide pesado)

Problema: Localizar observadores em rotundas para obter os volumes de tráfego direcionais, q_{ij}, da entrada i para a saída j, para todos os pares (i, j). São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 F_1. Assume-se que os veículos não estacionam no anel de circulação.

$$\begin{aligned} \sum_{j \in \mathcal{D}} q_{ij} &= O_i, \text{ para } i \in \mathcal{O} \\ \sum_{i \in \mathcal{O}} q_{ij} &= D_j, \text{ para } j \in \mathcal{D} \\ \sum_{i \in \mathcal{O} \setminus \{1\}} \sum_{j \in \mathcal{D},\, 1 \prec j \preceq i} q_{ij} &= F_1 \end{aligned}$$

Se colocar observador para q_{ij} tem um custo c_{ij}. **Pretendemos minimizar o custo total.** Quais dos q_{ij}'s não serão obtidos por observação, sendo calculados por resolução do sistema?

A **base ótima** pode ser calculada pelo "algoritmo greedy trivial".

Localizar observadores em rotundas para deduzir volumes de tráfego direcionais q_{ij}' s (matróide pesado)

Problema: Localizar observadores em rotundas para obter os volumes de tráfego direcionais, q_{ij}, da entrada i para a saída j, para todos os pares (i, j). São dados os volumes totais O_i e D_j e ainda o que passa frontalmente ao ramo 1 F_1. Assume-se que os veículos não estacionam no anel de circulação.

$$\begin{aligned} \sum_{j \in \mathcal{D}} q_{ij} &= O_i, \text{ para } i \in \mathcal{O} \\ \sum_{i \in \mathcal{O}} q_{ij} &= D_j, \text{ para } j \in \mathcal{D} \\ \sum_{i \in \mathcal{O} \setminus \{1\}} \sum_{j \in \mathcal{D},\, 1 \prec j \preceq i} q_{ij} &= F_1 \end{aligned}$$

Se colocar observador para q_{ij} tem um custo c_{ij}. **Pretendemos minimizar o custo total.** Quais dos q_{ij}'s não serão obtidos por observação, sendo calculados por resolução do sistema?

A **base ótima** pode ser calculada pelo "algoritmo greedy trivial".

Sobre resolução de sistemas de equações...

$$\begin{cases} x_1 - 4x_2 + 5x_3 - 10x_4 = 2 \\ -x_1 + 7x_2 + 2x_3 - 4x_4 = 1 \\ 3x_2 + 7x_3 - 14x_4 = 3 \\ x_1, x_2, x_3, x_4 \in \mathbb{R} \end{cases} \qquad A = \begin{bmatrix} 1 & -4 & 5 & -10 \\ -1 & 7 & 2 & -4 \\ 0 & 3 & 7 & -14 \end{bmatrix}$$

- A matriz A tem característica 2 porque a terceira equação é redundante. $car(A)$ é igual também à dimensão do espaço pelas colunas de A.
- Dos $\binom{n}{2} = \binom{4}{2} = 6$ subconjuntos $\{A_i, A_j\}$, com $i \neq j$, cinco são **bases** do espaço gerado pelas colunas de A. Cada base define uma *"forma resolvida"*.

$$\begin{cases} x_1 = 6 - 43/3x_3 + 86/3x_4 \\ x_2 = 1 - 7/3x_3 + 14/3x_4 \end{cases} \qquad \begin{cases} x_1 = -1/7 + 43/7x_2 \\ x_3 = 3/7 - 3/7x_2 + 2x_4 \end{cases}$$

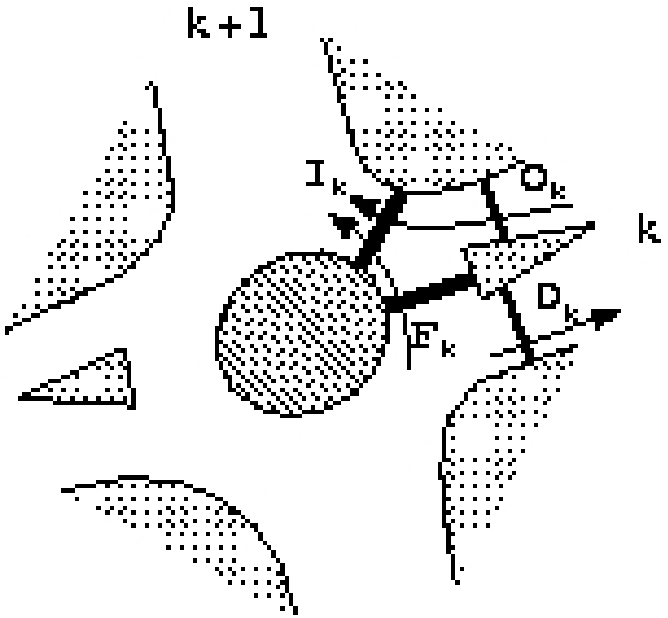
$$\begin{cases} x_1 = -1/7 + 43/7x_2 \\ x_4 = -3/14 + 3/14x_2 + 1/2x_3 \end{cases} \qquad \begin{cases} x_2 = 1/43 + 7/43x_1 \\ x_3 = 18/43 - 3/43x_1 + 2x_4 \end{cases}$$

$$\begin{cases} x_2 = 1/43 + 7/43x_1 \\ x_4 = -9/43 + 3/86x_1 + 1/2x_3 \end{cases} \qquad \begin{cases} x_3 = \text{....}A_3 \text{ e } A_4 \text{ linearmente} \\ x_4 = \text{.....dependentes} \end{cases}$$

- Sistema **indeterminado**. Se se atribuir valores às $n - car(A)$ **variáveis livres**, admite uma única solução para as restantes $car(A)$ variáveis.

Exemplo 2: Contagem de tráfego em rotundas

Obter os volumes direcionais q_{ij}, para todos os (i, j), com custo total mínimo.



$$\sum_{j \in \mathcal{D}} q_{ij} = O_i, \text{ para } i \in \mathcal{O}$$

$$\sum_{i \in \mathcal{O}} q_{ij} = D_j, \text{ para } j \in \mathcal{D}$$

$$\sum_{i \in \mathcal{O} \setminus \{k\}} \sum_{j \in \mathcal{D},\, k \prec j \preceq i} q_{ij} = F_k, \text{ para } 1 \leq k \leq N$$

$$I_k = F_k + O_k, \text{ para } k \in \mathcal{O}$$

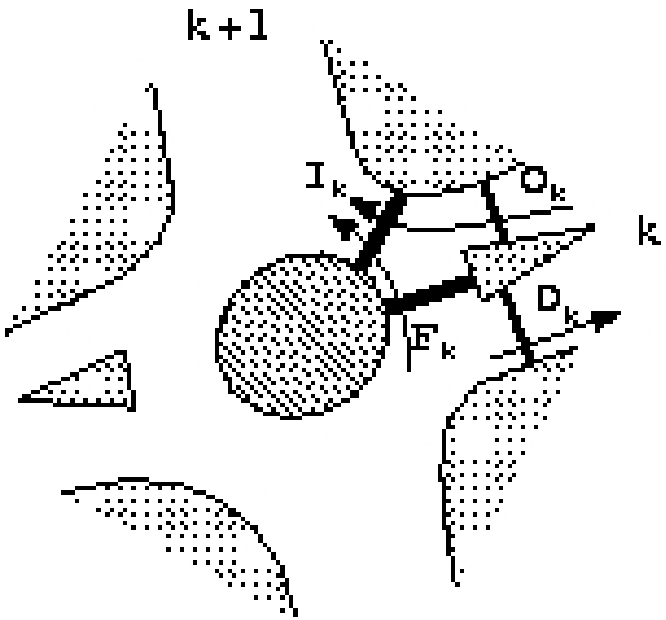
$$I_k = F_k, \text{ para } k \notin \mathcal{O}$$

O_i: total na entrada i; D_j: total na saída j;
F_k: total na secção frontal a k; I_k: total na secção intermédia k.

O número de variáveis q_{ij} é $|\mathcal{O}| \times |\mathcal{D}|$ mas a **caraterística da matriz** do sistema é $|\mathcal{O}| + |\mathcal{D}|$ ou $|\mathcal{O}| + |\mathcal{D}| - 1$. **Quais dos q_{ij} não serão obtidos por observação, sendo calculados por resolução do sistema?**

Exemplo 2: Contagem de tráfego em rotundas

Obter os volumes direcionais q_{ij}, para todos os (i, j), com custo total mínimo.



- Qualquer uma das equações para os O_i's e D_j's é redundante face às restantes. Também, apenas um dos F_k's poderá ser não redundante face aos O_i's e D_j's.
- A caraterística da matriz do sistema é $|\mathcal{O}| + |\mathcal{D}|$ ou $|\mathcal{O}| + |\mathcal{D}| - 1$.
- É $|\mathcal{O}| + |\mathcal{D}| - 1$ sse a equação que define F_k é $F_k = 0$, para algum k. Isto acontece se a rotunda for do tipo $S^\star(D + SE)E^\star$, onde S designa saída, E entrada e D sentido duplo.

A.P Tomás, M. Andrade and A. Pires da Costa (2001) Obtaining Origin-Destination Data at Optimal Cost at Urban Roundabouts. In CSOR - EPIA'01.
A.P. Tomás (2002). Solving Optimal Location of Traffic Counting Points at Urban Intersections in CLP(FD). In Proc. MICAI'2002. LNAI 2313, 242-251. `http://www.dcc.fc.up.pt/~apt/onlinepapers/micai02.pdf`

Exemplo 2: Contagem de tráfego em rotundas

Sendo r a caraterística da matriz do sistema, queremos escolher os r volumes direcionais **independentes** que **não serão** obtidos por contagem mas deduzidos dos restantes q_{ij}'s e dos volumes totais em secção (O_i's, D_j's e F_k's), de forma a minimizar o **custo da recolha**.

Exemplos: $custo(q_{ij})$ = número de ramos de i para j

		volumes a observar
SEDE	r=3+2=5 vars $= 3 \times 2 = 6$	q(4,1)
DDDE	r=4+3=7 nvars $=4 \times 3 = 12$	q(1,2) q(2,3) q(3,1) q(4,1) q(4,2)
DDSE	r= 3+3 = 6 nvars $=3 \times 3 = 9$	q(1,2) q(2,3) q(4,1)
SSSD	r= 1+4-1 = 4 nvars $=1 \times 4 = 4$	

SSSD é do tipo $S^\star(D + SE)E^\star$

Exemplo 2: Contagem de tráfego em rotundas

Exemplo: A rotunda SDSDEE, com $c_{o_i d_j}$ = número de ramos de o_i para d_j.

- A caraterística da matriz do sistema é $|\mathcal{O}| + |\mathcal{D}| = 8$ e o número de variáveis é 16.
- Considerando os volumes de tráfego O_i's, D_j's e F_1, a matriz de coeficientes é:

q_{21}	q_{22}	q_{23}	q_{24}	q_{41}	q_{42}	q_{43}	q_{44}	q_{51}	q_{52}	q_{53}	q_{54}	q_{61}	q_{62}	q_{63}	q_{64}
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1

- **Propriedade da matriz do sistema:** a coluna $\mathbf{p}'_{o_i d_j}$ associada à variável $q_{o_i d_j}$ é

$$\mathbf{p}'_{o_i d_j} = \mathbf{e}_i + \mathbf{e}_{m+j} + \theta_{o_i d_j} \mathbf{e}_{m+n+1}.$$

  Os vetores $\mathbf{e}_t$ definem a base canónica de $\mathbb{R}^{n+m+1}$, $\theta_{o_i d_j}$ indica se $q_{o_i d_j}$ passa ou não frontalmente ao ramo 1.

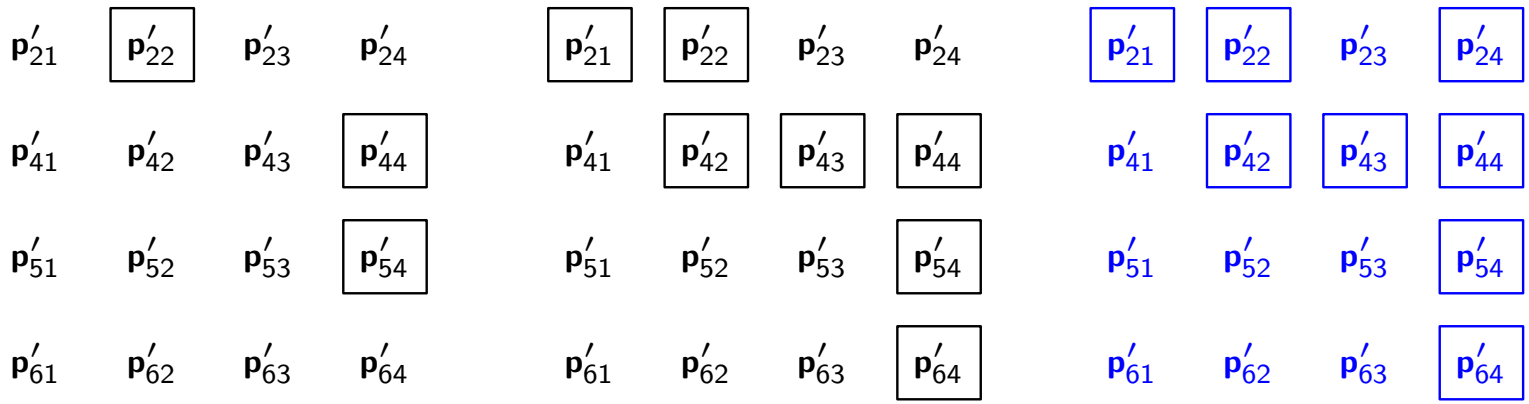
- **Ordem decrescente de custos:** $c_{22} = c_{44} = 6$, $c_{54} = c_{21} = c_{43} = 5$, $c_{64} = c_{53} = c_{42} = 4$, $c_{41} = c_{52} = c_{63} = 3$, $c_{24} = c_{51} = c_{62} = 2$, $c_{23} = c_{61} = 1$. A base ótima pode ser obtida pelo **"algoritmo greedy trivial"**.

  Como testar independência?

Exemplo 2: Contagem de tráfego em rotundas

Se custo total for dado pela soma dos custos da contagem de cada um dos q_{ij}'s, o problema corresponde à determinação da solução de peso máximo num matróide pesado. A solução ótima pode ser determinada pelo "algoritmo greedy trivial".

A rotunda SDSDEE com c_{ij} = número de ramos de i para j



- **Ordem decrescente de custos:**
  $c_{22} = c_{44} = 6, c_{54} = c_{21} = c_{43} = 5, c_{64} = c_{53} = c_{42} = 4, c_{41} = c_{52} = c_{63} = 3,$
  $c_{24} = c_{51} = c_{62} = 2, c_{23} = c_{61} = 1.$
- Dependências lineares:

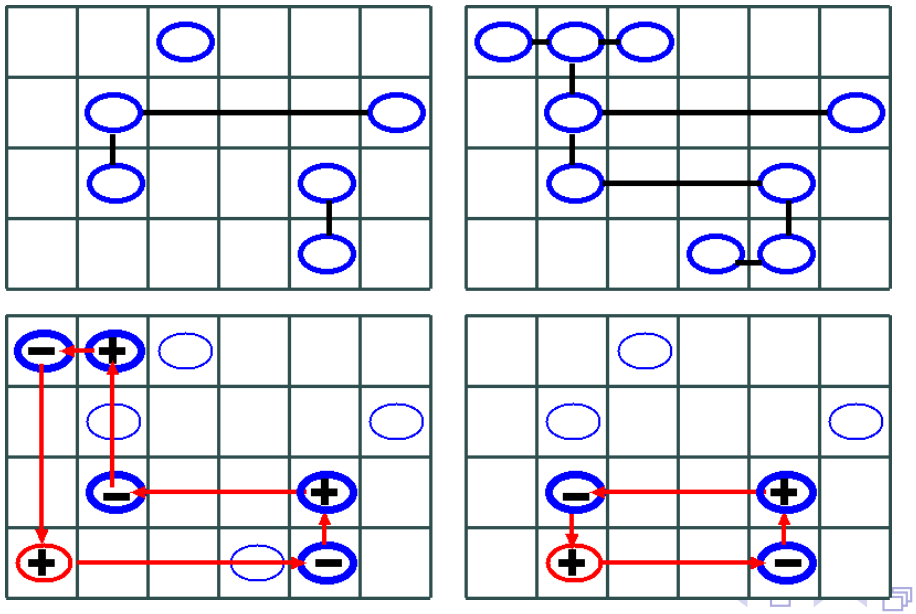
$$\mathbf{p}'_{53} = \mathbf{p}'_{54} - \mathbf{p}'_{44} + \mathbf{p}'_{43}$$
$$\mathbf{p}'_{41} = \mathbf{p}'_{42} - \mathbf{p}'_{22} + \mathbf{p}'_{21}$$
$$\ldots$$

Exemplo 2: Contagem de tráfego em rotundas

- $\mathbf{p}'_{o_i d_j} = \mathbf{e}_i + \mathbf{e}_{m+j} + \theta_{o_i d_j} \mathbf{e}_{m+n+1}$: coluna de q_{ij} no sistema. **No subsistema relativo a entradas e saídas**, $\mathbf{p}_{o_i d_j} = \mathbf{e}_i + \mathbf{e}_{m+j}$ e tem-se a seguinte propriedade.

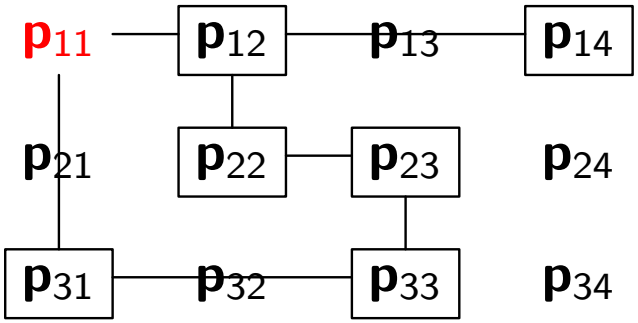
Propriedade: um conjunto B é independente sse o grafo que se obtém se se ligar por um ramo cada elemento de B aos mais próximos na mesma linha e coluna da tabela for acíclico. Se B for uma base, tem $m + n - 1$ elementos. Cada coluna da matriz do sistema escreve-se de forma única como combinação linear dos elementos da base B. Os coeficientes da combinação são 1, -1 ou 0.

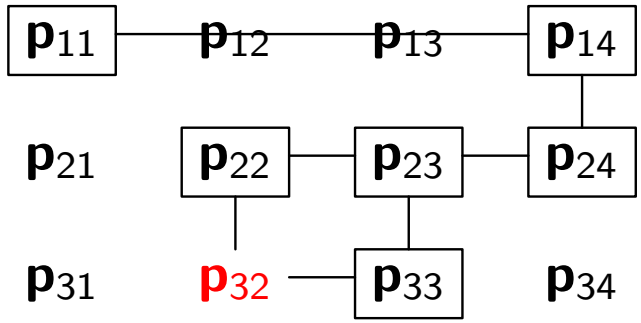
Exemplo anterior: $\mathbf{p}'_{53} = \mathbf{p}'_{54} - \mathbf{p}'_{44} + \mathbf{p}'_{43}$ porque $\mathbf{p}_{53} = \mathbf{p}_{54} - \mathbf{p}_{44} + \mathbf{p}_{43}$ e $\theta_{53} = \theta_{54} - \theta_{44} + \theta_{43}$.



Ciclo de Dependências

Exemplo: Sejam $m = 3$ e $n = 4$. O ciclo de dependência para $\mathbf{p}_{ij}$ define a coluna $\mathbf{p}_{ij}$ como combinação linear das colunas da base. Por definição de base, para cada base fixa, tal combinação é única. $\therefore$ **o ciclo é único**.





$$\mathbf{p}_{11} = \mathbf{p}_{31} - \mathbf{p}_{33} + \mathbf{p}_{23} - \mathbf{p}_{22} + \mathbf{p}_{12} \qquad \mathbf{p}_{32} = \mathbf{p}_{22} - \mathbf{p}_{23} + \mathbf{p}_{33}$$

Estão representadas duas soluções básicas. Tais combinações resultam de:

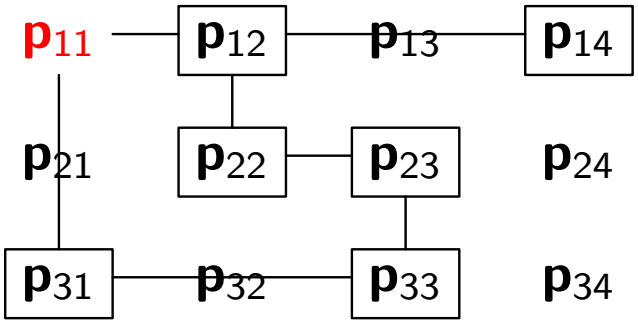
$$\mathbf{p}_{11} = \mathbf{e}_1 + \mathbf{e}_{3+1} = (\mathbf{e}_3 + \mathbf{e}_{3+1}) - (\mathbf{e}_3 + \mathbf{e}_{3+3}) + (\mathbf{e}_2 + \mathbf{e}_{3+3}) - (\mathbf{e}_2 + \mathbf{e}_{3+2}) + (\mathbf{e}_1 + \mathbf{e}_{3+2})$$

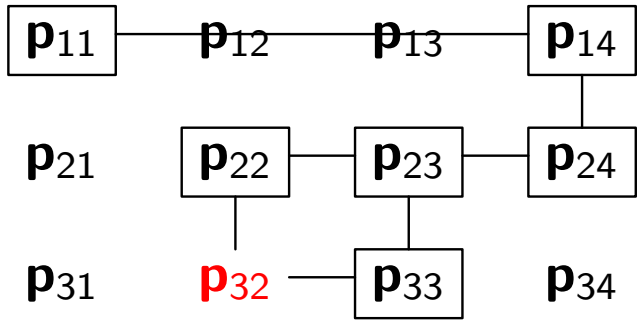
$$\mathbf{p}_{32} = \mathbf{e}_3 + \mathbf{e}_{3+2} = \underbrace{(\mathbf{e}_2 + \mathbf{e}_{3+2})}_{\mathbf{p}_{22}} - \underbrace{(\mathbf{e}_2 + \mathbf{e}_{3+3})}_{\mathbf{p}_{23}} + \underbrace{(\mathbf{e}_3 + \mathbf{e}_{3+3})}_{\mathbf{p}_{33}} = \mathbf{p}_{22} - \mathbf{p}_{23} + \mathbf{p}_{33}$$

Observe o cancelamento de vetores canónicos em termos consecutivos.

Ciclo de Dependências

Exemplo: Sejam $m = 3$ e $n = 4$. O ciclo de dependência para $\mathbf{p}_{ij}$ define a coluna $\mathbf{p}_{ij}$ como combinação linear das colunas da base. Por definição de base, para cada base fixa, tal combinação é única. $\therefore$ **o ciclo é único**.





$$\mathbf{p}_{11} = \mathbf{p}_{31} - \mathbf{p}_{33} + \mathbf{p}_{23} - \mathbf{p}_{22} + \mathbf{p}_{12} \qquad \mathbf{p}_{32} = \mathbf{p}_{22} - \mathbf{p}_{23} + \mathbf{p}_{33}$$

Estão representadas duas soluções básicas. Tais combinações resultam de:

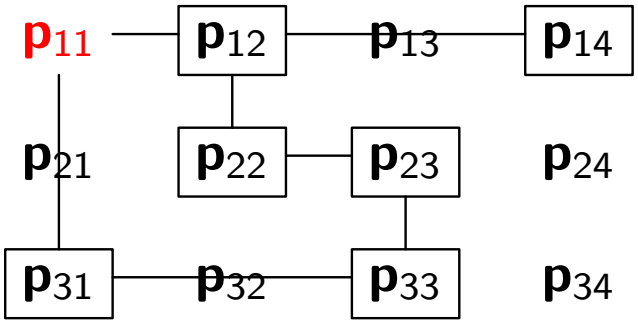
$$\mathbf{p}_{11} = \mathbf{e}_1 + \mathbf{e}_{3+1} = (\mathbf{e}_3 + \mathbf{e}_{3+1}) - (\mathbf{e}_3 + \mathbf{e}_{3+3}) + (\mathbf{e}_2 + \mathbf{e}_{3+3}) - (\mathbf{e}_2 + \mathbf{e}_{3+2}) + (\mathbf{e}_1 + \mathbf{e}_{3+2})$$

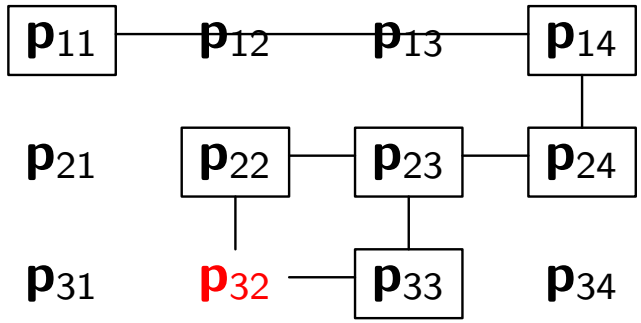
$$\mathbf{p}_{32} = \mathbf{e}_3 + \mathbf{e}_{3+2} = \underbrace{(\mathbf{e}_2 + \mathbf{e}_{3+2})}_{\mathbf{p}_{22}} - \underbrace{(\mathbf{e}_2 + \mathbf{e}_{3+3})}_{\mathbf{p}_{23}} + \underbrace{(\mathbf{e}_3 + \mathbf{e}_{3+3})}_{\mathbf{p}_{33}} = \mathbf{p}_{22} - \mathbf{p}_{23} + \mathbf{p}_{33}$$

Observe o cancelamento de vetores canónicos em termos consecutivos.

Ciclo de Dependências

Exemplo: Sejam $m = 3$ e $n = 4$. O ciclo de dependência para $\mathbf{p}_{ij}$ define a coluna $\mathbf{p}_{ij}$ como combinação linear das colunas da base. Por definição de base, para cada base fixa, tal combinação é única. $\therefore$ **o ciclo é único**.





$$\mathbf{p}_{11} = \mathbf{p}_{31} - \mathbf{p}_{33} + \mathbf{p}_{23} - \mathbf{p}_{22} + \mathbf{p}_{12} \qquad \mathbf{p}_{32} = \mathbf{p}_{22} - \mathbf{p}_{23} + \mathbf{p}_{33}$$

Estão representadas duas soluções básicas. Tais combinações resultam de:

$$\mathbf{p}_{11} = \mathbf{e}_1 + \mathbf{e}_{3+1} = (\mathbf{e}_3 + \mathbf{e}_{3+1}) - (\mathbf{e}_3 + \mathbf{e}_{3+3}) + (\mathbf{e}_2 + \mathbf{e}_{3+3}) - (\mathbf{e}_2 + \mathbf{e}_{3+2}) + (\mathbf{e}_1 + \mathbf{e}_{3+2})$$

$$\mathbf{p}_{32} = \mathbf{e}_3 + \mathbf{e}_{3+2} = \underbrace{(\mathbf{e}_2 + \mathbf{e}_{3+2})}_{\mathbf{p}_{22}} - \underbrace{(\mathbf{e}_2 + \mathbf{e}_{3+3})}_{\mathbf{p}_{23}} + \underbrace{(\mathbf{e}_3 + \mathbf{e}_{3+3})}_{\mathbf{p}_{33}} = \mathbf{p}_{22} - \mathbf{p}_{23} + \mathbf{p}_{33}$$

Observe o cancelamento de vetores canónicos em termos consecutivos.