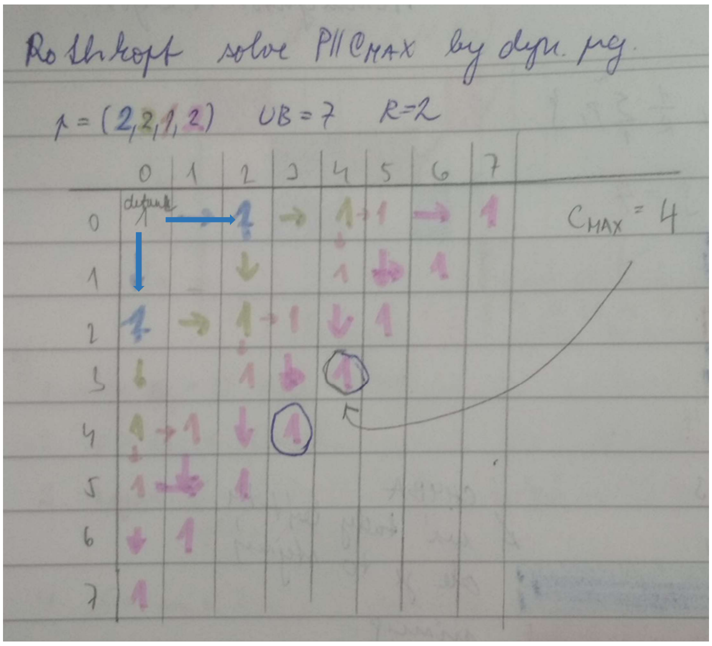
Rothkopt

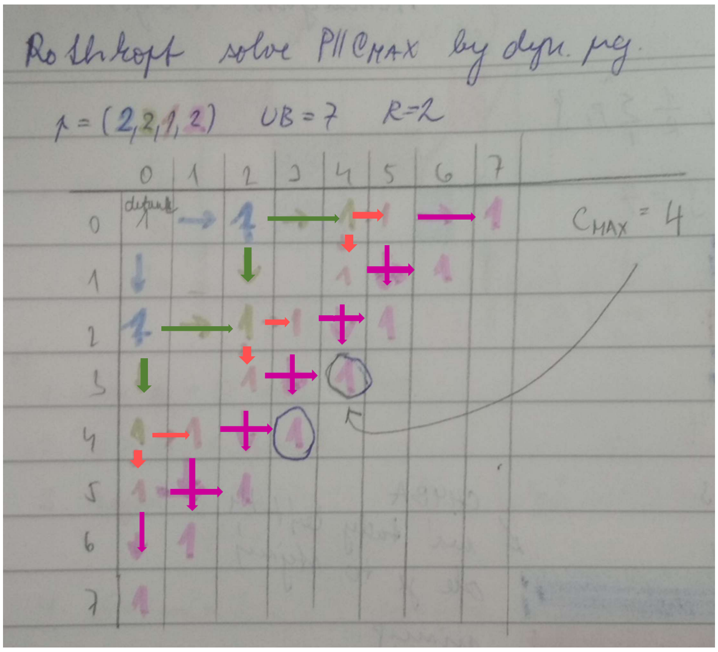
- Uděláme si podle UB 7x7 2D mřížku – 2D, protože máme R = 2 (dva resources)
- Děláme všechny možný kombinace, které by mohly nastat – což zapíšeme do mřížky a pak vykoukáme, kde nám nastane minimum

- Začínám na (0,0) a koukám na p = (2,2,1,2), které mi ukazuje, že mám jít dva kroky do stran

- Vyplním tudíž 1 a 1

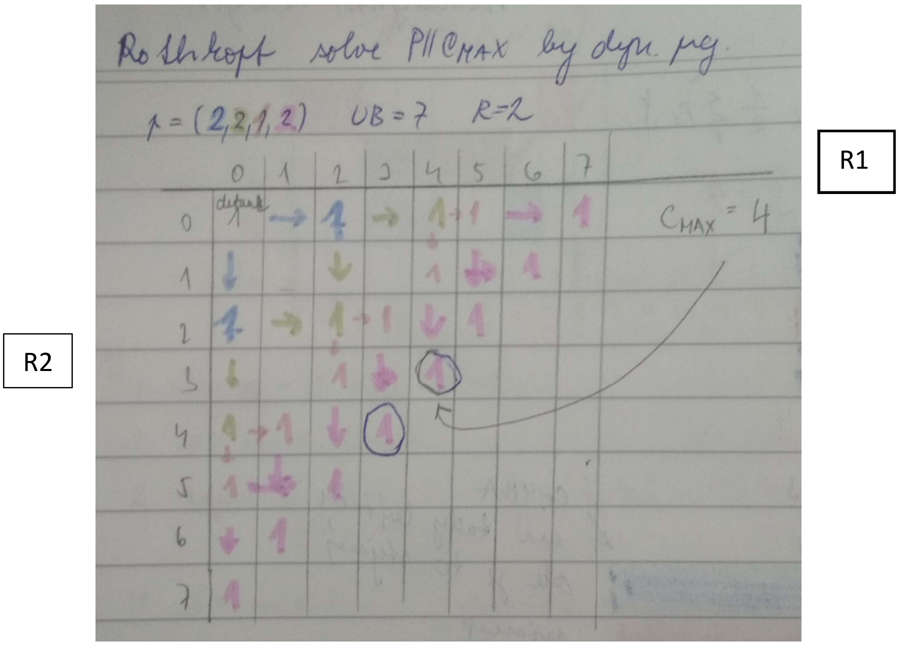


- Pak jdu dál po p a zjistím, že mám jít z modrých jedniček zase dva kroky p = (2,2,1,2), takže z modrých jedniček udělám dva kroky do všech stran (všechny možné kombinace)
- A pak ze zelených jedniček chci jít jeden krok p = (2,2,1,2)
- Pak jdu 2 kroky z růžových jedniček p = (2,2,1,2)



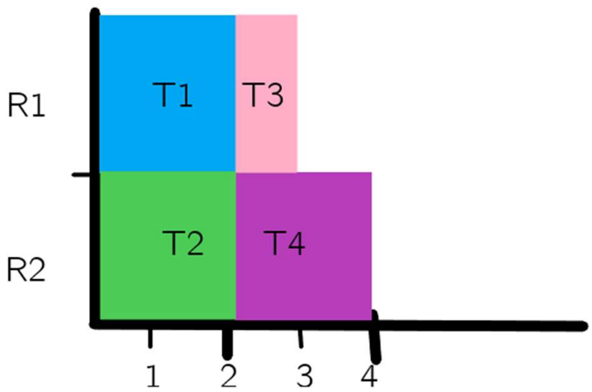
Pak dojdeme k vyplněné diagonále a tam se snažíme najít “minimum z maxima” obou os. Máme následující možnosti (podle diagonály) – [0,7],[1,6],[2,5],[3,4],[4,3],[5,2],[6,1],[7,1]

Takže je nejlepší vybrat buď [3,4], nebo [4,3]. To jsou identická řešení, akorat prohazuju který z těch resources to zpracovává.



Gantt chart

- Máme naše řešení, např. [3,4] a půjdeme po šipkách – z defaultu jdeme 2x doprava (na R1 dáme T1 s časovým úsekem 2), pak 2x dolů (na R2 dáme T2 s trváním 2), pak 1x doprava (na R1 dáme T3 s trváním 1) a pak 2x dolů (na R2 dáme T4 s trváním 2).

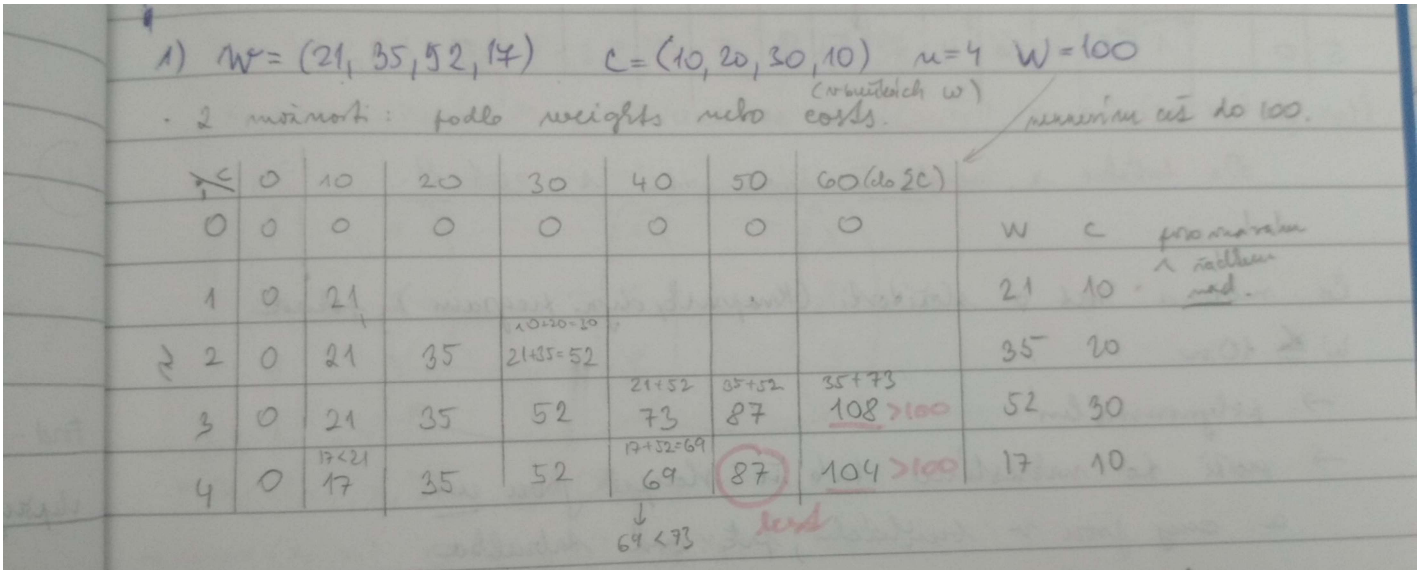


Knapsack

- Cost je na sloupcích, item i je na řádcích
- Postupně přisuzujeme item weights a snažíme se o nejlepší možnou kombinaci
- V řádku 1 – dali jsme do baťohu item o váze 21, který stojí 10. Je to nejlepší možná kombinace, kterou teď můžeme udělat, protože máme jen 1 item.
- V řádku 2 – na sloupci 10 máme variantu, kdy jsme tam dali item o váze 21, na sloupci 20 máme variantu, kdy jsme tam dali item o váze 35. Na sloupci 30 máme variantu, kdy jsme tam dali oba itemy, dohromady s vahou 35 + 21 = 52.
- V řádku 3 – se nejdříve podíváme, jestli item 3 (který stojí 30 a váží 52) není lepší variantou, než to, co tam máme (kombinace itemu 1 a 2). Není, protože váží a stojí stejně.

  Pak na tomto řádku děláme další kombinace, který obsahují item 3 a kombinace z předchozích řádků (nejlepší kombinace, který jsme doteď mohli mít), které nám dají dohromady cost 40 (item 1 + item 3), cost 50 (item 2 + item 3), cost 60 (součet všech).

- V řádku 4 – podíváme se, jestli item 4 (váží 17, stojí 10) není lepší než item, který mám na sloupci s costem 10. Ano je, váží méně. Pak opakuju to, co se dělo v řádku 3.



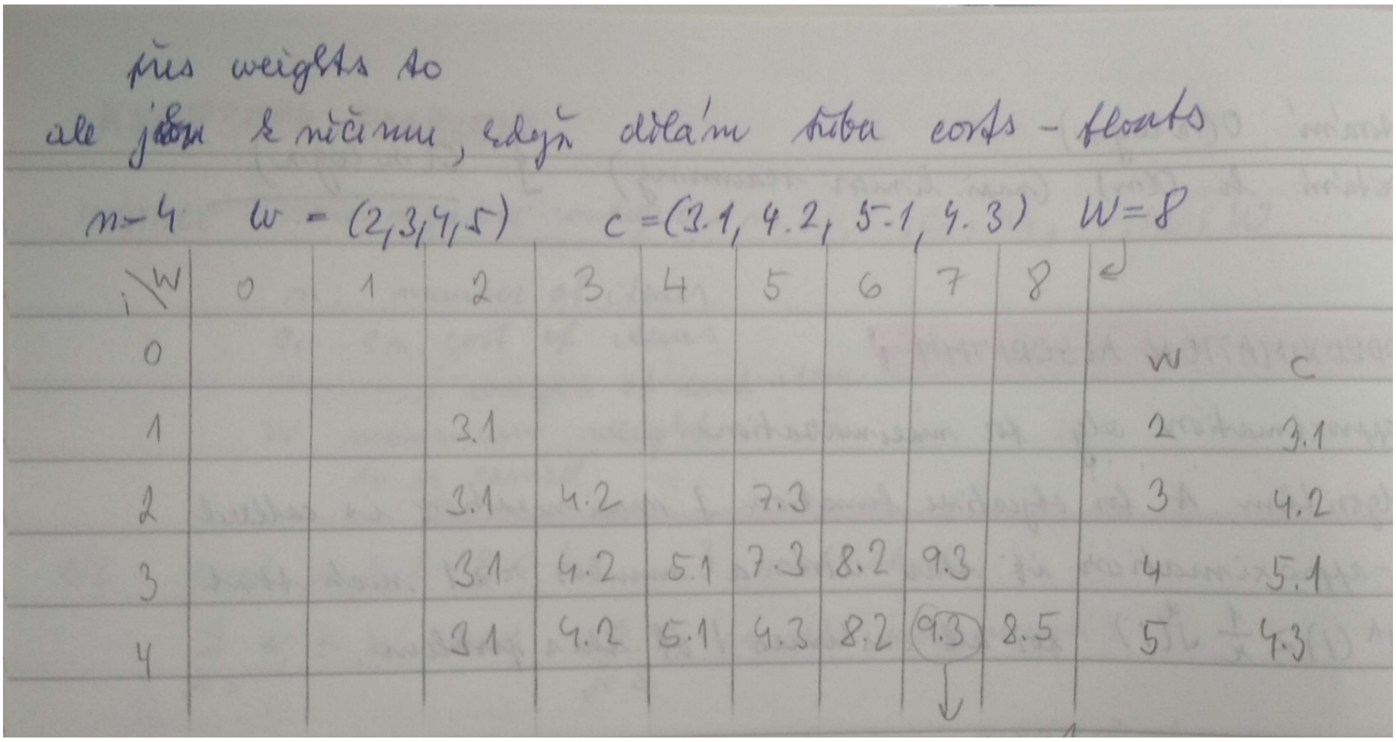
1) w = (21, 35, 52, 17)   c = (10, 20, 30, 10)   n = 4   W = 100

i \ c	0	10	20	30	40	50	60
0	0	0	0	0	0	0	0
1	0	21					
2	0	21	35	10+20=30 21+35=52			
3	0	21	35	52	21+52 73	35+52 87	35+73 108 >100
4	0	17<21 17	35	52	17+52=69 69	87	104 >100

w	c
21	10
35	20
52	30
17	10

69 < 73

Je možnost to taky dělat přes weights. Místo cost připíšu weights do sloupců. To se hodí, když mám costs jako floats.



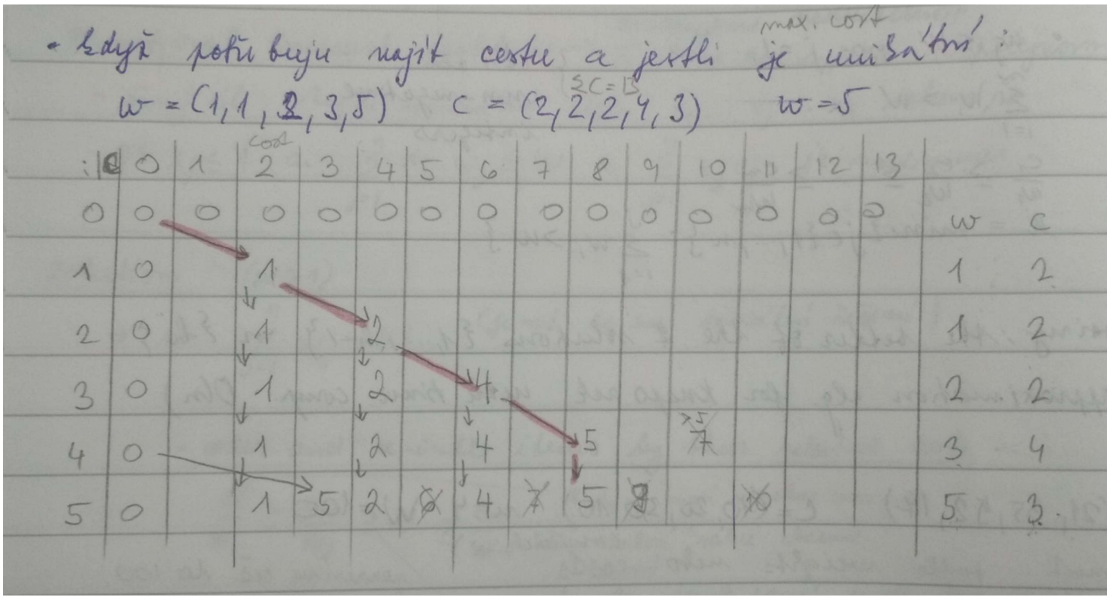
přes weights to
ale jsou k ničemu, když dělám třeba costs – floats

$n=4$ $w=(2,3,4,5)$ $c=(3.1, 4.2, 5.1, 4.3)$ $W=8$

i \ W	0	1	2	3	4	5	6	7	8	w	c
0											
1			3.1							2	3.1
2			3.1	4.2		7.3				3	4.2
3			3.1	4.2	5.1	7.3	8.2	9.3		4	5.1
4			3.1	4.2	5.1	4.3	8.2	9.3	8.5	5	4.3

Pokud máme úlohu, kde máme najít cestu, tak nejdřív zase najdeme tu nejlepší variantu. Pak se podíváme, jestli jsme tam ten item dali nebo nedali (indikovaný šipkou).

Svislá šipka (růžová) značí, že jsme převzali lepší variant a item jsme nevložili. Šikmá značí, že jsme tam ten item vložili. Výsledná kombinace je vložení itemů 1, 2, 3, 4. Na item 5 jsme šli svisle, takže jsme ho tam nedali.

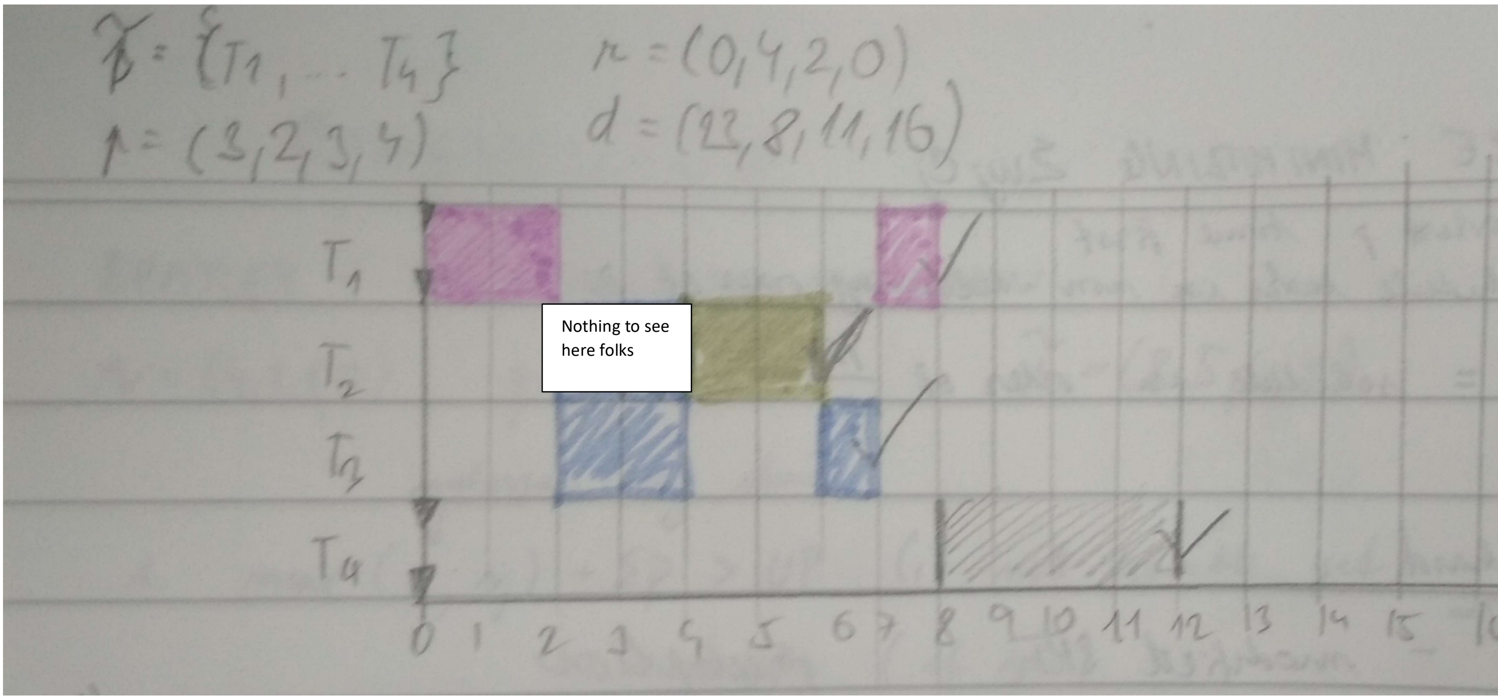


• když potřebuju najít cestu a jestli je umístění: max. cost

$w=(1,1,2,3,5)$ $c=(2,2,2,4,3)$ $\Sigma C=13$ $W=5$

i \ c	0	1	2	3	4	5	6	7	8	9	10	11	12	13	w	c
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
1	0		1												1	2
2	0		1		2										1	2
3	0		1		2		4								2	2
4	0		1		2		4		5		>5 ~~7~~				3	4
5	0		1	5	2	~~6~~	4	~~7~~	5	9		~~10~~			5	3

Hornův algoritmus



1. Najdeme nejmenší r_j a nastavíme, že $t_j = r_j$.
   Takže nastavíme, že $t_1 = 0$.

2. Najdeme další nejmenší release time r_i, který bude větší než ten release time, co jsme doteď našli. A nastavíme, že $t_{j+1} = r_i$.

   Takže r_i. = 2 a $t_2 = 2$.

3. Najdu task/tasky, které mají ten první release time r_j a přidám je do T‘.

   T‘ = {T1, T4}

   Podíváme se, kdo z těch tasků má minimální dealine d:

   T1 má deadline 13, takže vybereme ten.

4. Podívám se, jestli T1 stihne dodělat task, nebo ho dožene release time dalšího tasku a bude se muset přerušit:

$$\delta = min\{p_k, t_{j+1} - t_i\}$$

   Zde jsme zjistili, že $\delta = min\{3, 2\} = 2$.

   Takže se T1 nestačí dodělat. Uděláme, co můžeme a ten zbytek doděláme “někdy jindy”.

   Teď musíme ještě snížit processing time T1 o to, co jsme už udělali a všem taskům z T’ navýšit release time (v našem případě jen T4). Jelikož jsme měli čas jen $\delta = 2$, tak processing time T1 bude 3 – 2 = 1 a nové p bude p = (1, 2, 3, 4) a nové r = (2, 4, 2, 2).

   Opakujeme dokud všichni neskončí.

   1. $r_j = 2$ a $t_1 = 2$.
   2. $r_i = 4$ a $t_2 = 4$.

3. T‘ = {T4,T3, T1} → T3 má dřívější deadline - 11
4. *$\delta = min\{3, 2\} = 2$ → zase nestíhá*
   P = (1, 2, 1, 4), r = (4, 4, 4, 4)

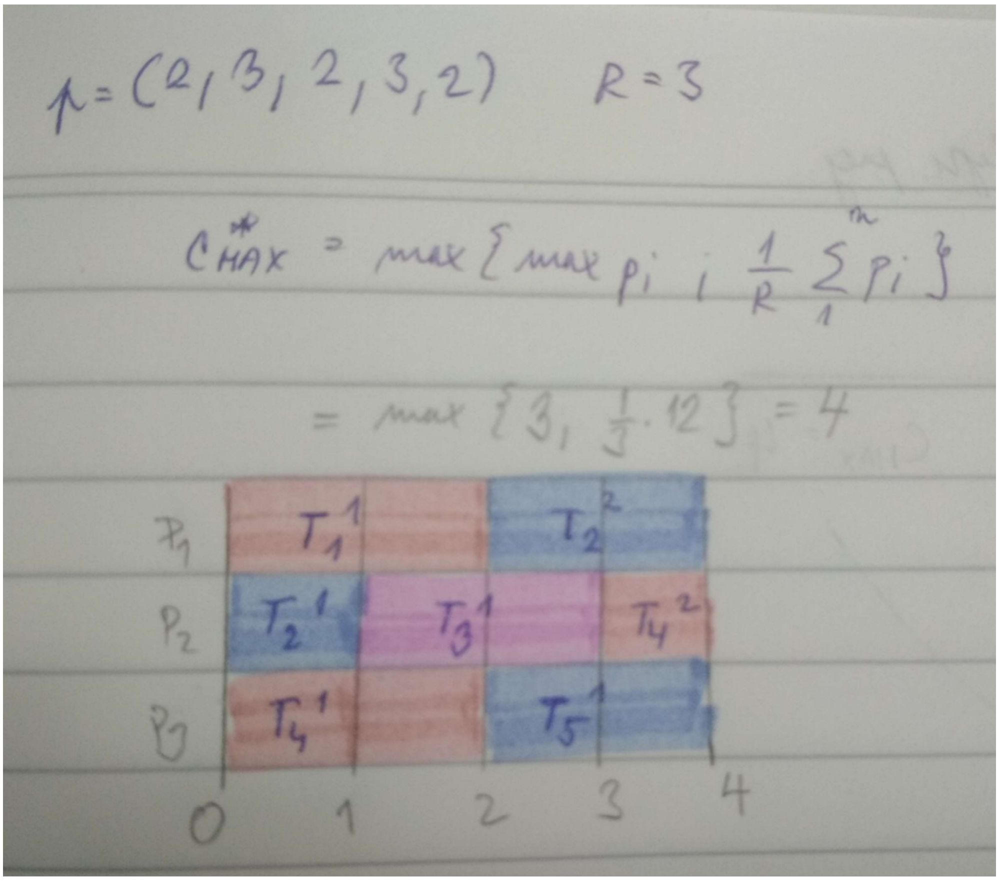
Další iterace

1. $r_j = 4$ a $t_1 = 4$.
2. $t_2 = \infty$ (at this point už je nám tohle jedno, protože všichni mají stejný release time)
3. T‘ = {T4, T3, T1, T2} → T2 začne nestíhat nejdřív
4. Protože t_2 je nekonečno, tak víme, že uděláme T2 už celou. Zvednu release times o $\delta = 2$
   p = (1, X, 1, 4), r = (6, X, 6, 6).

Dál už ani nemusíme vlastně iterovat podle algoritmu, v podstatě vezmeme ty tasky, který budou mít nejzazší deadline a uděláme je celé.

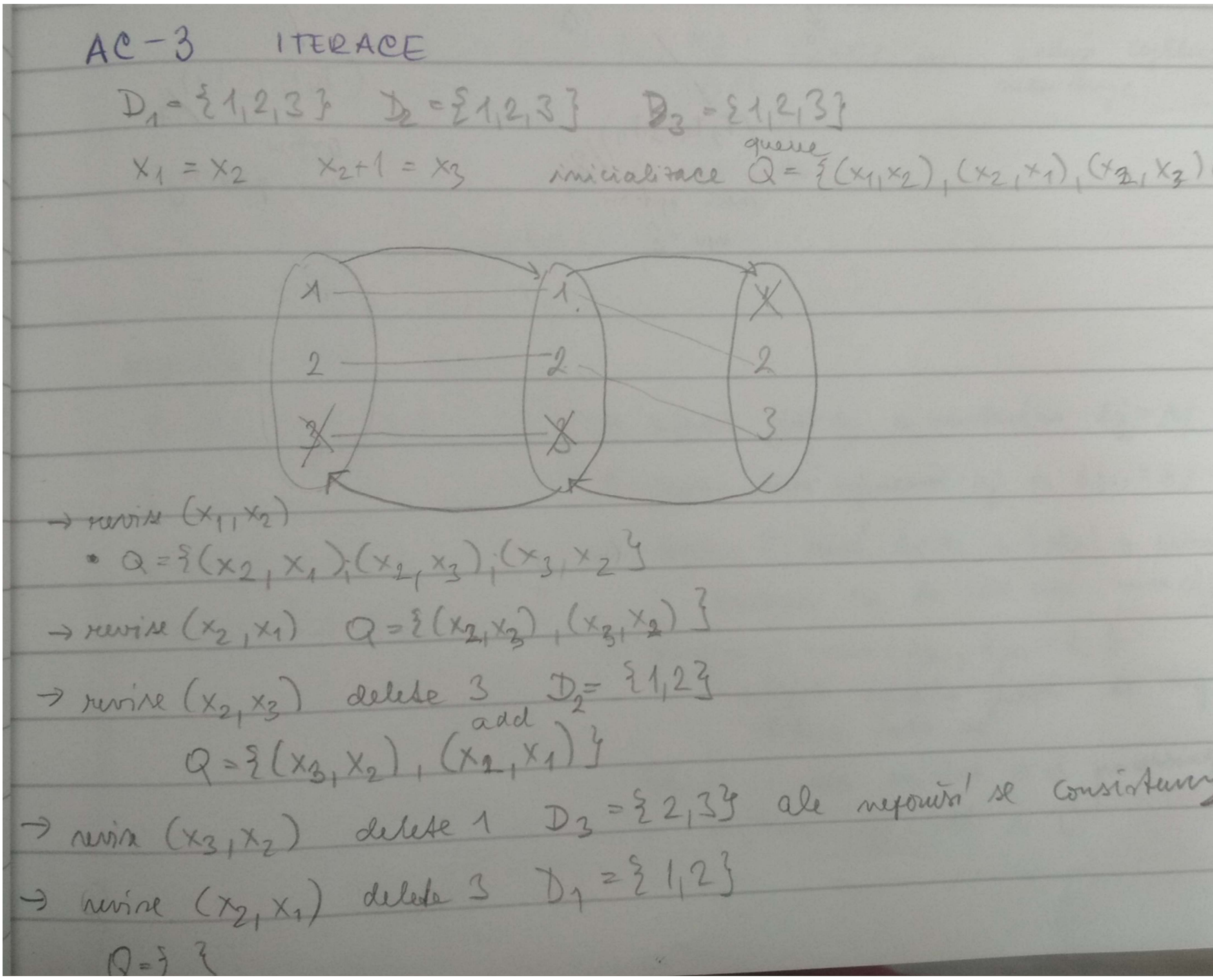
McNaughtonův algoritmus

- Podle vzorečku spočítáme Cmax
- Myšlenka je taková, že děláme do Cmax, dokud můžeme. Pokud můžeme task udělat celý, tak ho uděláme a řekneme, že jsme udělali 1. část. Pokud task nestihneme udělat do té doby, tak nejdříve uděláme druhou část, posuneme se na zdrojích a uděláme první část.



T1 má processing 2, to se vejde do P1, takže ho udělám celý. Pak jdu dělat T2, který má processing 3 a ten se mi celý na P1 nevejde. Udělám 2. část, pohnu se na zdroji do P2 a dodělám první část, která mi zbývá. T3 stihnu udělat celou. T4 si zase rozkouskuju, ….

AC-3 CSP



Přidáme do Q = {(x1, x2), (x2, x1), (x2, x3), (x3, x2)}.
Máme vztahy: x1 = x2, x2 + 1 = x3
Máme domainy D1 = {1,2,3}, D2 = {1,2,3}, D3 = {1,2,3}

Revise(x1, x2)
Existuje v D1 nějaké číslo, u kterého by neplatilo x1 = x2 ? Ne, jdeme dál. Popneme z Q to, co jsme zpracovali a máme Q = {(x2, x1), (x2, x3), (x3, x2)}.

Revise (x2, x1)
Existuje v D2 nějaké číslo, u kterého by neplatilo x1 = x2? Ne, jdeme dál.
Q = {(x2, x3), (x3, x2)}.

Revise (x2, x3):
Máme v D2 číslo, u kterého by neplatilo x2 + 1 = x3 ? Ano – číslo 3.
Smažeme z D2 číslo 3 a popneme z Q. Tím, že jsme smazali z domainu jsme ale porušili konzistenci, takže musíme přidat zpátky do Q revision (x2, x1).
D2 = {1,2}, Q = {(x3, x2), (x2, x1)}.

Revise (x3, x2):
Máme v D3 číslo, u kterého by neplatilo x2 + 1 = x3 ? Ano – číslo 1, protože v D2 není žádné číslo, které bychom mohli sečíst s 1 a dalo by to dohromady 1. Smažeme 1, což neporuší konzistenci, protože by to ovlivnilo pouze ten opačný směr (x2, x3).
D3 = {2, 3}, Q = {(x2, x1)}.

Revise (x2, x1):
D2 = {1,2}, takže to znamená, že pro x1 = x2 by nefungovalo číslo 3. Smažeme 3 z D1 → D1 = {1, 2}. Toto opět neporuší konzistenci, protože by to ovlivnilo pouze opačný směr.
Už nemáme nic ve frontě, ukončujeme algoritmus.