```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Markdown, display
import ipywidgets
```

```
def same(*args):
    for arg in args:
        assert args[0] == arg
    return args[0]
```

> **Note:** In cod, am folosit tabele indexate de la 0, insa in formule am folosit indexare de la 1. Deci A^1 se regaseste in `A[:,0]`, iar $\alpha_{0,0}$ se regaseste in `alpha[(-1,-1)]`

Rezolvam urmatoarea problema de optimizare, prin algoritmul simplex primal:

$$(P)\begin{cases} f(x) = x_1 + 2x_2 + 2x_3 - 3x_4 \to \min \\ x_1 - 2x_2 - 4x_3 = 6 \\ x_2 + 3x_3 + x_4 = 8 \\ x_1, x_2, x_3, x_4 \geq 0 \end{cases}$$

Mai intai, extragem c, b, si A.

```
c = np.array([1, 2, 2, -3])
c
```

```
array([ 1,  2,  2, -3])
```

```
b = np.array([[6], [8]])
b
```

```
array([[6],
       [8]])
```

```
A = np.array([
    [1, -2, -4, 0],
    [0, 1, 3, 1],
])
A
```

```
array([[ 1, -2, -4,  0],
       [ 0,  1,  3,  1]])
```

A^1 este prima baza, iar A^4 este cea de-a doua.

```
Js = [0, 3]
```

```
def generate_table(Is, Js, alpha):
    Is = list(Is)
    Js = list(Js)
```

```
    table = []
    for i in ["head"] + Is + [-1]:
        table.append("|")
        for j in ["head"] + Js + [-1]:
            match (i, j):
                case ("head", "head"):
                    table.append("1")
                case ("head", -1):
                    table.append("X")
                case (-1, "head"):
                    table.append("b")
                case ("head", val):
                    table.append(f" $A^{val+1}$ ")
                case (val, "head"):
                    table.append(f" $A^{val+1}$ ")
                case _:
                    table.append(f" $\\alpha_{{{i+1},{j+1}}}={alpha[(i,j)]}$ ")
            table.append("|")

        if i == "head":
            table.append("\n")
            table.append("|")
            for i in ["head"] + Is + [-1]:
                table.append("---|")

        table.append("\n")
    table = "".join(table)
    table = Markdown(table)
    return table

def simplex_primal(A, b, c, Js):
    same(2, len(A.shape), len(b.shape))
    same(1, len(c.shape))

    n = same(A.shape[1], c.shape[0])
    m = same(A.shape[0], b.shape[0])

    Js = set(Js)
    Is = set(range(n)) - Js

    # compute alpha
    alpha = dict()

    for i in Is:
        for j, val in zip(Js, A[:, i]):
            alpha[(i, j)] = val
```

```
for j, val in zip(Js, b[:, 0]):
    alpha[(-1, j)] = val

for i in Is:
    alpha[(i, -1)] = sum(
        alpha[(i, j)]*c[j]
        for j in Js
    ) - c[i]

alpha[(-1, -1)] = sum(alpha[(-1, j)]*c[j] for j in Js)

while True:
    display(generate_table(Is, Js, alpha))

    Isp = []
    for i in Is:
        if alpha[(i, -1)] > 0:
            Isp.append(i)

    # found optimal solution
    if len(Isp) == 0:
        return alpha[(-1, -1)]

    for i in Isp:
        fail = all(alpha[(i, j)] <= 0 for k in Js)
        if fail:
            raise RuntimeError("Function does not have lower bound, hence the problem ha

    h = Isp[0]

    k = min(
        (j for j in Js if alpha[(h, j)] > 0),
        key=lambda j: alpha[(-1, j)] / alpha[(h, j)],
    )

    p = alpha[(h, k)]

    new_alpha = dict()
    for i in Is.union({-1}):
        for j in Js.union({-1}):
            match (i == h, j == k):
                case (True, True):
                    new_alpha[(k, h)] = 1 / p
                case (True, False):
                    new_alpha[(k, j)] = -(alpha[(i, j)] / p)
```

```
                case (False, True):
                    new_alpha[(i, h)] = alpha[(i, j)] / p
                case _:
                    new_alpha[(i, j)] = alpha[(i, j)] - alpha[(h, j)] * alpha[(i, k)] /
        alpha = new_alpha

        Is.remove(h)
        Is.add(k)

        Js.remove(k)
        Js.add(h)


simplex_primal(A, b, c, Js)
```

```
<IPython.core.display.Markdown object>
```

```
-18
```

$\alpha_{0,1} = 6 \geq 0$ si $\alpha_{0,4} = 8 \geq 0 \Rightarrow$ baza primal admisibila $\alpha_{2,0} = -1 \leq 0$ si $\alpha_{3,0} = -15 \leq 0 \Rightarrow$ baza primal admisibila $\Rightarrow B$ este baza optima pentru problema $(P) \Rightarrow$ valoarea minima a lui f este $\alpha_{0,0} = -18$ obtinuta prin $x = (x_1, x_2, x_3, x_4) = (6, 0, 0, 8)$

verificare:

```
def f(x):
    return np.sum(x * c)

f(np.array([6, 0, 0, 8]))
```

```
-18
```