**Теоретичні відомості**

У задачах лінійного і нелінійного програмування досліджувані процеси не залежали від часу. Такі задачі деколи називають **одноетапними** або **однокроковими**, їх розв’язки також не залежать від часу.

У 40-50-х роках ХХ століття виникають задачі, пов’язані з вивченням функціонування складних систем, коли потрібно брати до уваги вплив часового фактора. У 50-х роках американський математик Р.Беллман запропонував для розв’язування оптимізаційних задач новий підхід, який дістав назву **динамічного програмування**. Ідея підходу полягає в тому, що початкову оптимізаційну задачу заміняють низкою однотипних часткових оптимізаційних задач, їх називають **кроками** або **етапами**. Далі знаходять оптимальні розв’язки кожної часткової задачі і вже за ними будують розв’язок початкової. Саму початкову задачу називають також **багатокроковою** або **багатоетапною**.

Динамічне програмування – це сукупність методів, які складають математичний апарат для вивчення багатокрокових задач прийняття оптимальних рішень. Термін «багатокрокова задача» характеризує покроковий (поетапний) підхід до її розв’язання. Цей підхід може відображати характер розгортання процесу і прийняття рішень в часі або може вводитися у задачу штучно, щоб розбити процес прийняття рішень на окремі простіші етапи-кроки.

Сформулюємо задачу динамічного програмування. Нехай керована система, яка перебуває в початковому стані S’, повинна перейти за деякий час у кінцевий стан S\*. Із зміною стану системи пов’язаний деякий числовий критерій F (цільова функція). Потрібно знайти таке планування процесу, за якого критерій F досягав би оптимального значення.

Задачам динамічного програмування можна надати геометричного тлумачення. Наприклад, стан системи, яка залежить від двох параметрів x1 i x2, можна зобразити на площині х1Ох2 точками S(x1; x2), їх також називатимемо **станами**. Тоді областю можливих станів є площина або деяка її частина. Зміна стану системи призводить до переміщення точки S у площині. Перехід зі стану S’ у стан S\* можна здійснювати вздовж декількох різних траєкторій, серед яких потрібно вибрати ту, на якій значення критерію F є оптимальним.

У загальному випадку, коли стан системи залежить від n параметрів xi, i = , областю можливих станів є n-вимірний простір. Тоді задача динамічного програмування полягає в тому, щоб відшукати оптимальну траєкторію між точками S’ та S\* у цьому просторі.

У динамічному програмуванні процес переведення систем з початкового у кінцевий стан здійснюється за скінченну кількість кроків. Причому кожен крок поведінки потрібно планувати так, щоб разом з наступними забезпечити оптимальну поведінку системи в цілому. Тому на першому кроці є найбільший вибір варіантів. Оскільки кожен зроблений крок звужує можливості наступних, то останній крок повинен бути найпростішим для планування. Тому покрокове розв'язання задачі звичайно починають з останнього кроку. Вибір поведінки системи в динамічному програмуванні називається **керуванням**.

Отже, якщо потрібно перевести систему зі стану S0 в станSk за k кроків, то розв’язання задачі починають з останнього кроку.

**1-ий крок розв’язання** (вибір керування для останнього k-го кроку поведінки системи). Результатом цього кроку є вибір оптимального керування (тобто такого, яке забезпечує досягнення оптимального значення критерію F) для останнього кроку поведінки системи.

**2-ий крок розв’язання** (вибір керування для (k-1)-го кроку поведінки системи). Результатом цього кроку є вибір оптимального керування для двох останніх кроків поведінки системи. Причому планувати достатньо лише (k-1)-ий крок поведінки, беручи до уваги те, що для k-го кроку оптимальне керування вже сплановано.

Аналогічно виконують наступні кроки розв’язання і процес побудови оптимальних керувань розгортається від кінцевого до початкового кроків поведінки системи.

**k-ий (останній) крок розв’язання** (вибір керування для 1-го кроку поведінки системи). Цей крок дає можливість отримати розв’язок задачі. Результатом кроку є вибір оптимального керування для всіх k кроків поведінки системи. Причому планувати достатньо лише 1-ий крок поведінки, беручи до уваги, що для всіх наступних (k-1) кроків оптимальне керування вже сплановано.

Описана покрокова схема розв’язання задачі ґрунтується на тому, що оптимальне керування для кожного кроку поведінки системи вибирають за принципом оптимальності, сформульованим Беллманом.

**Принцип оптимальності:** керування для m-го кроку поведінки системи в багатокроковій задачі потрібно вибирати так, щоб разом з оптимальним керуванням для (m+1) кроку отримати оптимальне значення цільової функції для всієї решти кроків, починаючи з m-го.

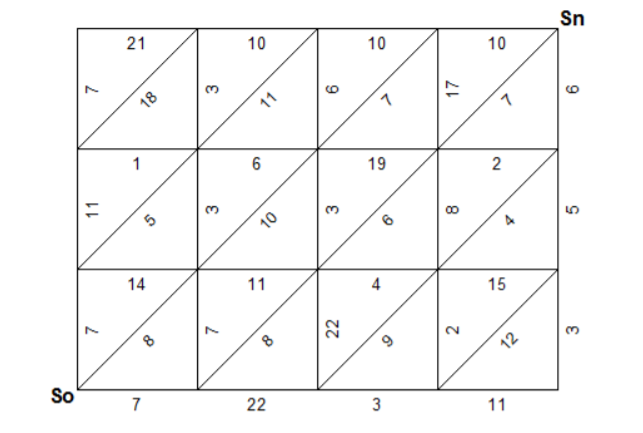
Принцип оптимальності застосовують лише для таких цільових функцій, значення яких у багатокроковому процесі дорівнює сумі їх значень на кожному окремому кроці.

Принцип оптимальності дозволяє зробити такий висновок. Якщо в результаті розв’язання k-крокової задачі вибрано оптимальне керування, яке не переводить систему з початкового стану S0 послідовно у стани S1, …,Sk, то згідно з принципом оптимальності переведення системи з будь-якого проміжного стану Sm, 0<m<k, в кінцевий стан Sk здійснюється також оптимальним керуванням відносно станів Sm і Sk. Якщо розв’язкові такої задачі у просторі можливих станів відповідає оптимальна траєкторія з кінцем у точці Sk, то довільна частина S’Sk цієї траєкторії буде також оптимальною стосовно її початку S’ і кінця Sk. Отже, принцип оптимальності дозволяє отримати розв’язок цілої низки однотипних багатокрокових задач.

**Задача про мінімізацію витрат пального літаком при наборі висоти і швидкості**

Одна з найпростіших задач, до якої можна застосувати метод динамічного програмування, є задача про оптимальні витрати пального при набиранні висоти літаком.

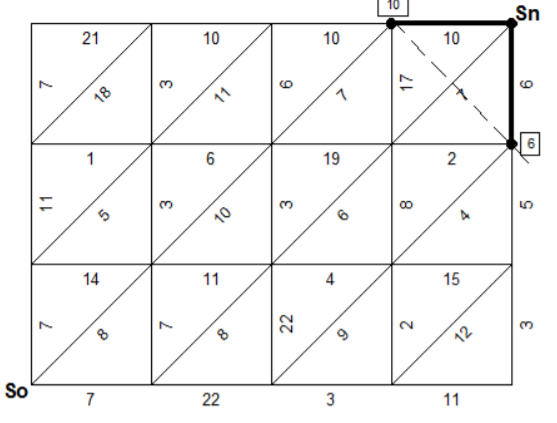
Літак, що знаходиться на висоті h0 та летить зі швидкістю v0 , повинен піднятися на висоту hk та досягнути швидкості yk . Досягнення цих значень здійснюється за k кроків, причому на кожному кроці можливий вибір однієї з двох альтернатив – набирати висоту при постійній швидкості або набирати швидкість при постійній висоті (можливе узагальнення цієї задачі з декількома, не обов’язково двома режимами). Для всіх режимів відомі витрати пального. Необхідно визначити таку траєкторію польоту, щоб досягнути заданої швидкості та висоти польоту за k кроків.



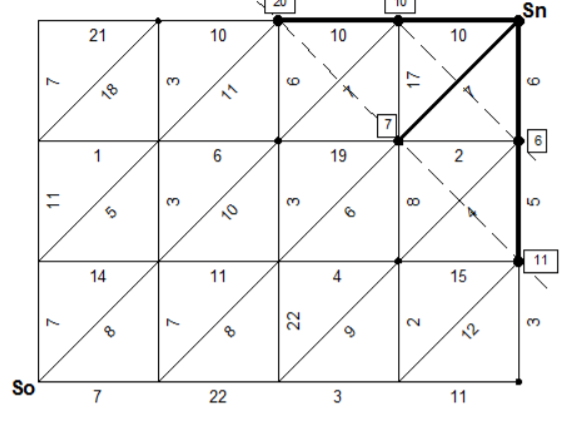
Стан si = (hi, vi) характеризується двома складовими – висотою та швидкістю польоту. Відобразимо всі можливі стани та витрати пального при однокроковому переході зі стану в стан: розіб’ємо відрізки (h0 ,hk ) та (v0,vk ) на h1 та h2 інтервалів відповідно. В цьому випадку досягнення кінцевого стану здійснюється за k=h1+h2 кроків.

Рухаючись від останнього стану до початкового, будуємо пучок оптимальних траєкторій, рухаючись якими з кожного стану досягаємо останнього з мінімальними витратами пального.

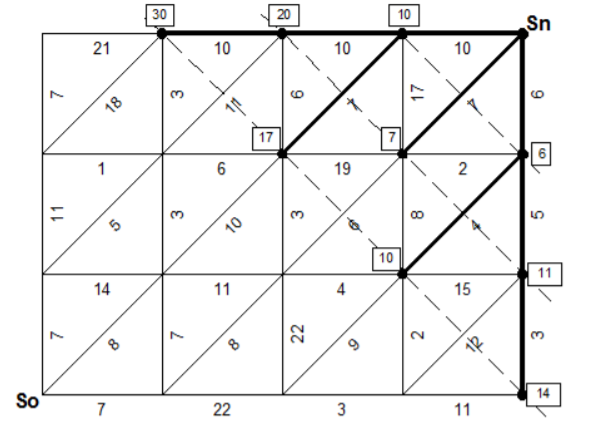
**Крок 1.**



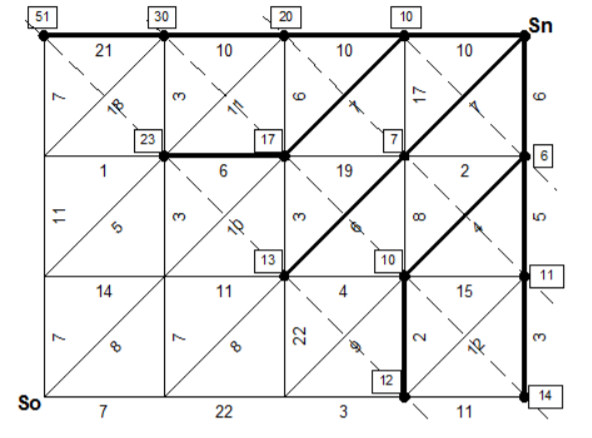
**Крок 2.**



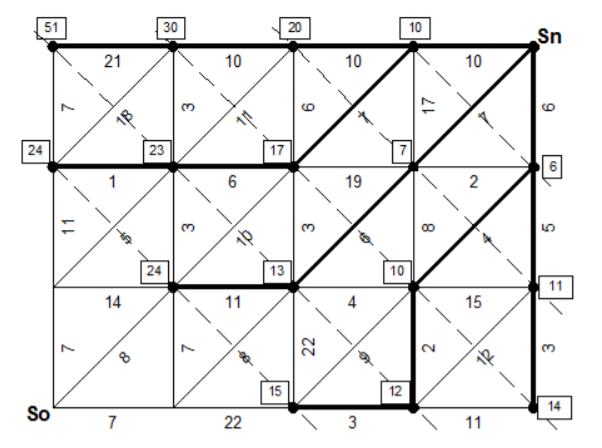
**Крок 3.**



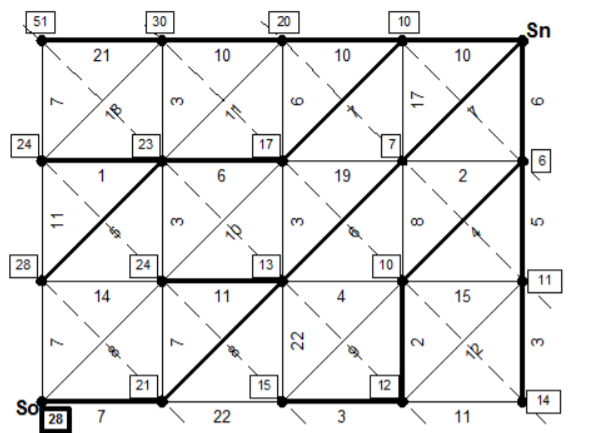
**Крок 4.**



**Крок 5.**



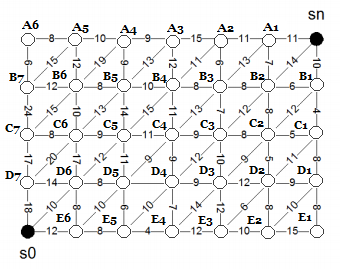
**Крок 6.**



Мінімальні витрати пального при русі з початкового стану становитимуть 28 одиниць.

**Завдання**

Визначити шлях з S0 до Sn з найменшими витратами пального.



**Розв’язання**

|  |  |  |
| --- | --- | --- |
| **№ кроку** | **Пояснення** | **Рисунок** |
| Крок 1 | *Для «діагоналі»* ***А1В1****:*  Точка А1 => min{11} = 11 =>  Точка В1 => min{10} = 10 => |  |
| Крок 2 | *Для «діагоналі»* ***А2B2C1****:*  Точка A2 => min{11+11} = 22 =>  Точка B2 => min{7+11; 14; 6+10} = 14 =>  Точка C1 => min{4+10} = 14 => |  |
| Крок 3 | *Для «діагоналі»* ***A3B3C2D1****:*  Точка A3 => min{15+22} = 37 =>  Точка B3 => min{9+22; 13+11; 8+14} = 22 =>  Точка C2 => min{8+14; 12+10; 5+14} = 19 =>  Точка D1 => min{8+14} = 22 => |  |
| Крок 4 | *Для «діагоналі»* ***A4B4C3D2E1****:*  Точка A4 => min{9+37} = 46 =>  Точка B4 => min{12+37; 11+22; 8+22} = 30 =>  Точка C3 => min{7+22; 12+14; 8+19} = 26 =>  Точка D2 => min{5+19; 11+14; 9+22} = 24 =>  Точка E1 => min{8+22} = 30 => |  |
| Крок 5 | *Для «діагоналі»* ***A5B5C4D3E2****:*  Точка A5 => min{10+46} = 56 =>  Точка B5 => min{9+46; 13+37; 10+30} = 40 =>  Точка C4 => min{11+30; 13+22; 9+26} = 35 => ,  Точка D3 => min{10+26; 9+19; 12+24} = 28 =>  Точка E2 => min{8+24; 10+22; 15+30} = 32 => , |  |
| Крок 6 | *Для «діагоналі»* ***A6B6C5D4E3:***  Точка A6 => min{8+56} = 64 =>  Точка B6 => min{12+56; 19+46; 8+40} = 48 =>  Точка C5 => min{14+40; 15+30; 11+35} = 45 =>  Точка D4 => min{9+35; 12+26; 9+28} = 37 =>  Точка E3 => min{12+28; 6+24; 10+32} = 30 => |  |
| Крок 7 | *Для «діагоналі»* ***B7C6D5E4:***  Точка B7 => min{9+64; 15+56; 12+48} = 60 =>  Точка C6 => min{10+48; 13+40; 9+45} = 53 =>  Точка D5 => min{11+45; 9+35; 0+37} = 37 =>  Точка E4 => min{7+37; 14+28; 12+30} = 42 => , |  |
| Крок 8 | *Для «діагоналі»* ***C7D6E5:***  Точка C7 => min{24+60; 15+48; 8+53} = 61 =>  Точка D6 => min{17+53; 12+45; 8+37} = 45 =>  Точка E5 => min{6+37; 10+37; 4+42} = 43 => |  |
| Крок 9 | *Для «діагоналі»* ***D7E6****:*  Точка D7 => min{17+61; 20+53; 14+45} = 59 =>  Точка E6 => min{8+45; 10+37; 8+43} = 47 => |  |
| Крок 10 | *Для точки* ***S0****:*  Точка S0 => min{18+59; 10+45; 12+47} = 55 => |  |
| Крок 11 | Отже, шлях з мінімальними витратами палива з точки S0 до точки Sn: **S0D6D5D4D3C2C1B1Sn**  Щоб подолати цей шлях, літаку потрібно витратити **55 одиниць** палива. |  |

**Програмне розв’язання**

**Код програми**

import matplotlib.pyplot as plt

import networkx as nx

import numpy as np

G = nx.DiGraph()

layout = { }

inf = 1000000

delta\_v = np.array([

[8, 10, 9, 15, 11, 11, 0],

[12, 8, 10, 8, 8, 6, inf],

[8, 9, 11, 9, 8, 5, inf],

[14, 8, 0, 9, 12, 9, inf],

[12, 8, 4, 12, 10, 15, inf]

])

delta\_h = np.array([

[inf, inf, inf, inf, inf, inf, 0],

[6, 12, 9, 12, 6, 7, 10],

[24, 10, 14, 11, 7, 8, 4],

[17, 17, 11, 9, 10, 5, 8],

[18, 8, 6, 7, 12, 8, 8]

])

delta\_h\_v = np.array([

[inf, inf, inf, inf, inf, inf, 0],

[15, 19, 13, 11, 13, 14, inf],

[15, 13, 15, 13, 12, 12, inf],

[20, 12, 9, 12, 9, 11, inf],

[10, 10, 10, 14, 6, 10, inf]

])

mitka = np.array([

[0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0]

])

i = 5

while i != -6:

if i >= 0:

m = 0

n = i

else:

m = -i

n = 0

zminna1 = delta\_v.diagonal(i)

zminna2 = delta\_h.diagonal(i)

zminna3 = delta\_h\_v.diagonal(i)

for x in range(0, len(zminna1), 1):

if m == 0:

mitka[m][n] = zminna1[x] + mitka[m][n+1]

G.add\_edge(mitka[m][n], mitka[m][n+1])

elif n == 6:

mitka[m][n] = zminna2[x] + mitka[m-1][n]

G.add\_edge(mitka[m][n], mitka[m-1][n])

else:

list\_zminna = np.array([zminna1[x] + mitka[m][n+1], zminna2[x] + mitka[m-1][n], zminna3[x] + mitka[m-1][n+1]])

mitka[m][n] = min(list\_zminna)

l = list\_zminna.tolist()

ind = l.index(min(l))

if ind == 0:

G.add\_edge(mitka[m][n], mitka[m][n+1])

elif ind == 1:

G.add\_edge(mitka[m][n], mitka[m-1][n])

elif ind == 2:

G.add\_edge(mitka[m][n], mitka[m-1][n+1])

layout[mitka[m][n]] = [n, m]

m+=1

n+=1

i-=1

layout[0] = [6, 0]

print("Minimal resources in path from S0 to Sn = " + str(mitka[4][0]))

plt.figure(figsize=(12, 12))

nx.draw(G, layout, edges=G.edges(), edge\_color='green', node\_color='firebrick')

nx.draw\_networkx\_labels(G, layout, font\_color='black')

plt.show()

**Робота програми**

*Пояснення до візуалізації:*

*Позначки на точках означають мінімальну кількість палива для кожної з них, щоб пройти шлях до Sn.*

*Стрілками між точками позначено цей оптимальний шлях між кожною точкою та Sn, щоб витратити стільки одиниць палива, скільки позначено на мітках цих точок.*



