Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра дослідження операцій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
зі спеціальності 113 «Прикладна математика»
на тему:

"ЗАСТОСУВАННЯ ЗАДАЧ ЦІЛОЧИСЕЛЬНОГО ПРОГРАМУВАННЯ"

Студента 4 курсу
Білоножко Аліна Вадимівна,

Науковий керівник:
Доцент, кандидат фізико-математичних наук
Якимів Роман Ярославович

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол №…….. від… ..... 2021 р.

Завідувач кафедри ДО професор Іксанов О.М.

Київ – 2021

ЗМІСТ

ВСТУП

Після другої світової війни повітряні сили США фінансували дослідження для розв'язку моделей військового планування та розподілу. У 1947 році було розроблено симплекс метод для розв'язку цього типу лінійних проблем. Незабаром, після цього було повідомлено про перше комерційне використання лінійного програмування (ЛП) у великих компаніях, які мали змогу використовувати комп'ютери. За рахунок вдалої розробки та розв'язку лінійних моделей для складних задач такі, непов'язані між собою, сфери як сільське господарство, транспорт, зв'язок, сталь та нафта змогли покращити своє фінансове становище та заощадити мільйони доларів на рік.

У міру того, як обчислювальна потужність ставала більш доступнішою, кількість галузі бізнесу та господарської діяльності, що починали використовувати лінійні моделі, зростали експоненційно.

Якщо усі функції задачі є лінійними, то вона стає задачею ЛП. Термін "програмування" в даному контексті використовується у значенні планування, а не комп'ютерного програмування. Отже, математична оптимізація, може також називатись математичним програмуванням. Також, якщо змінні в ЛП приймають тільки цілі значення, то воно стає цілочисельним ЛП або простим ЦП, також зустрічаються такі назви як: дискретне, комбінаторне або діофантове програмування. Моделлю цілочисельного програмування (МЦП) можна назвати ту, у якій одна або декілька змінних розв'язку повинні приймати цілі або двійкові значення у кінцевому розв'язку.

Цілочисельне програмування можна використовувати для розв'язку складних задач прийняття рішень та планування, що виникають в різних ситуаціях. Застосування ЦП до розв'язку реальної задачі в основному складається з двох етапів: перший етап, це побудова математичної моделі,

другий – розв’язок проблеми, що описана моделлю. Сильний взаємозв’язок між властивостями моделей та методів, що підходять для їх розв’язку, пов’язує між собою ці дві фази, хоча характер проблем, які виникають на цих етапах відрізняється.

На сьогоднішній день широко застосовуються в реальних бізнес та державних сферах. Великі економічні, фінансові, банківські моделі, моделі маркетингових стратегій, планування виробництва, планування робочої сили, моделі комп’ютерного дизайну, мережеві моделі, а також моделі медицини – це лише невелика кількість вдалих прикладів використання моделей лінійного програмування. Вибірка з випадків використання моделей ЛП:

- Задача призначення
- Розширення телекомунікаційних мереж
- Контроль за забрудненням повітря
- Охорона здоров’я
- Вибір банківського портфеля
- Сільське господарство
- Захист від вогню
- Оборонні / аерокосмічні контракти
- Планування землекористування
- Молочне виробництво
- Військового розгортання

Цілочисельне програмування - один з наймолодших, перспективних і швидко розвиваються розділів математичного програмування. Його важливість для задач економіки полягає у тому, що велика кількість ресурсів являється неподільними, а отже в процесі вони приймають участь лише цілими одиницями.

МЦП можна зустріти майже у всіх сферах застосування математичного програмуванння таких як призначення медичного

персоналу, отпимізація групп засобів доставки вантажу, пошуку мінімального холостогу проїзду автомобілів при виконанні заданого плану перевезень, визначення отпимального машинного парку, його розподіл по вказаним роботом за умови мінімізації сумарної вартості(машинного парку та виконуваних робіт), про знаходження мінімальної кількості суднів для здійснення даного графіку перевезень і т. п.. Важливим поштовхом до побудови теорії цілочислового програмування став новий підхід до деяких екстремальних комбінаторних задач. У цих задачах необхідно знайти екстремум лінійної цілочисельної функції, заданої на кінцевому безлічі елементів. Такі завдання прийнято називати завданнями з альтернативними змінними. Завдання цього типу являються актуальними, бо до їх розв’язку зводиться аналіз різних ситуацій, що виникають в великій кількості сфер життя.

РОЗДІЛ 1 ЦІЛОЧИСЕЛЬНЕ ПРОГРАМУВАННЯ

1.1 Історія створення

Серед практичних завдань пошуку екстремуму лінійної функції важливе місце займають завдання з умовою цілочисельності всіх або частинних змінних. Вони називаються задачами цілочисельного або дискретного програмування.

Перишим завданням цілочисельного типу є опопублікована угорським математиком Е. Егерварі в 1932 році задачам з призначення персоналу. У 1951 році Джордж Данціг запровадив цілочисельне програмування. А вже у 1954 року він довів, що задача Комівояжера, була особливим випадком ЦП. Системні дослідження в області цілочисельного програмування розпочато з 1955 р, коли на другому симпозіумі з лінійного програмування була розглянута задача бомбардувальника, нині відома як завдання про ранець. Перший збіжний алгоритм був знайдений у 1958 році Гоморі. У 1960 році було розроблено перший метод гілок і меж для розв'язку задач ЦП Алісою Ленд та Елісон Дойг.

Зараз цілочисельне програмування застосовується у різних проблемних областях:

- Задача розміщення. Наприклад, у якій частині міста потрібно розмістити склад товарів для магазину, щоб мінімізувати траспортні витрати.
- Задача покриття та розбиття множини. Наприклад, планування складу екіпажів для авіакомпаній.
- Мультитоварний розподіл. Створення оптимального маршруту поставок сировини.
- Бюджетування капіталу. Наприклад, розробка річних капіталів для компаній.

1.2 Задача пакування ранцю

Контейнер обладнаний m відсіками місткістю для перевезення n видів продукції. Види продукції характеризуються властивістю неподільності, тобто їх можна брати в кількості 0, 1, 2, ... одиниць. Нехай a_{ij} – витрата і-го відсіку для перевезення одиниці j-ої продукції. Позначимо через c_j корисність одиниці j-ої продукції. Потрібно знайти план $(x_1, x_2, \dots, x_n)$ перевезення, при якому максимізується загальна корисність рейсу. Модель задачі набуде вигляду:

$$\max F = \sum_{j=1}^{n} c_j x_j \qquad (1.2.1)$$

При обмеженні на місткість відсіків, умові невід'ємності та цілочисельності(відповідно):

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad (i = \overline{1,m}) \qquad (1.2.2)$$

$$x_j \ge 0 \qquad (j = \overline{1,n}) \qquad (1.2.3)$$

$$x_j - \text{цілі} \qquad (j = \overline{1,n}) \qquad (1.2.4)$$

Коли для перевезення є один відсік і кожен вид продукції може бути взятий чи ні, модель задачі набуває вигляду:

$$\max F = \sum_{j=1}^{n} c_j x_j \qquad (1.2.5)$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b \qquad (1.2.6)$$

$$x_j \in (0,1) \qquad (j = \overline{1,n}) \qquad (1.2.7)$$

1.3 Задача Комівояжера

Задача полягає у тому, що під час виконання маршруту, кожне з міст повинно бути відвідано лише один, і тільки один раз і у кінці знову потрапити у початкову точку маршруту.

Ми хочемо мінімізувати

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1.3.1}$$

$$\sum_{j=0}^{n} x_{ij} = 1, \qquad for\ i = 1, \dots, n \tag{1.3.2}$$

$$\sum_{i=0}^{n} x_{ij} = 1, \qquad for\ j = 1, \dots, n \tag{1.3.3}$$

$$t_i - t_j + nx_{ij} \leq n - 1, \quad for\ i, j = 2, \dots, n \tag{1.3.4}$$

Де

$$c_{ij} = \text{відстань від міста } i \text{ до міста } j \tag{1.3.5}$$

$$x_{ij} = \begin{cases} 1, & \text{якщо місто } j \text{ відвідали відразу після } i \\ 0, & \text{у іншому випадку} \end{cases} \tag{1.3.6}$$

$$t_i = \text{довільні дійсні числа, які ми повинні знайти, але їх значення не важливі} \tag{1.3.7}$$

Перший набір n умов обмежує кількість разів, коли розв'язок потрапляє у кожне місто, до 1. Другий набір n умов обмежує кількість разів, коли розв'язок виходить з кожного, до 1. Останній набір умов використовується для виключення підмаршрутів. Зокрема, умови гарантують те, що в будь-який маршрут входить місто 1, що у поєднанні з першим та другим набором умов змушує проводити один маршрут.

Для того, щоб зрозуміти, чому останній набір умов примушує будь-який маршрут включати місто 1, пропонуємо розглянути маршрут, який не включає місто 1 і покажемо суперечність, що виникає в такому разі. Припустимо, що маршрут включає у себе таку послідовність міст

$v_1, v_2, v_3, \dots, v_t$ і жодне з них не є містом 1. Давайте додамо рівняння, отримані з останнього набору умов для кожної пари міст, котрі відвідуються один за одним в рамках одного маршруту. Оскільки це маршрут, то кожен t_{v_i}, буде набувати по одному додатному та від'ємному значенні, а отже один з цих разів буде скасування. Отже це залишає нас з $nl \leq (n-1)l$, що суперечить умові.

1.4 Задача призначення

Припустимо, що в нас є n робітників, які можуть виконувати n р різноманітних робіт. C_{ij} – користність, пов'язана з виконанням і-м виконавцем j-й роботи $(i, j = \overline{1,n})$. Нам необхідно назначати робітників так, щоб досягти максимальної корисності, при умові, що робітник повинен бути закріпленений лише за однією роботою і аналогічно для роботи.

Математична модель задачі матиме вигляд:

$$\max F = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \qquad (1.4.1)$$

На кожну роботу призначається тільки один виконавець:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad (1.4.2)$$

Кожен виконавець призначається тільки на одну роботу:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (1.4.3)$$

Умови невід'ємності та цілочисельності:

$$x_j \geq 0 \qquad x_j \in (0,1) \qquad (i, j = \overline{1,n}) \qquad (1.4.4)$$

1.5 Інші умови виражені за допомогою ЦП

Логічні обмеження $(x_1, x_2, binary)$:

$$\text{Або, } x_1 \text{ або } x_2$$
$$\Longrightarrow \quad x_1 + x_2 = 1 \tag{1.5.1}$$

$$\text{Якщо } x_1, \text{ то } x_2$$
$$\Longrightarrow \quad x_1 - x_2 \leq 0 \tag{1.5.2}$$

Для поєднання умов можуть використовуватися і цілі обмеження:

Або

$$\begin{matrix} a_1 x \leq b_1 \\ a_2 x \leq b_2 \end{matrix} \Longrightarrow \begin{cases} a_1 x - My & \leq b_1 \\ a_2 x - M(y-1) & \leq b_2 \\ y \; binary & \end{cases} \tag{1.5.3}$$

М скалярна і повинна бути “великою” (x, a_1, a_2 – вектори). Якщо $y = 1$, то перша рівність всюди істинна і друга у цьому випадку також виконується, якщо $y = 0$, то друга рівність всюди істинна і перша повинна справджуватись. Оскільки на у накладено обмеження двійковості, то перша умова повинна бути виконана.

Подібну підхід можна використовувати для обробки умов m з n.

РОЗДІЛ 2. АЛГОРИТМИ РОЗВ'ЯЗКУ ЗАДАЧ ЦІЛОЧИСЕЛЬНОГО ПРОГРАМУВАННЯ

2.1. Апроксимація лінійного програмування

Розв'язок ЛП – це верхня межа розв'язку ЦП. Якщо неможливий ЛП, то і ЦП також. Якщо розв'язок ЛП є цілим (усі змінні набувають цілочисельні значення), то він також є розв'язком ЦП.

Деякі задачі ЛП (задача призначення та транспортна, наприклад) завжди будуть мати цілочисельні розв'язки. Задачі такого типу можна класифікувати як задачі з унімодулярною матрицею A. ($Unimodular \implies \det A = 0, or -1$)

Решта задач ЛП мають дійсні розв'язки, які необхідно округлити до найближчого цілого значення. Проте, це може порушувати задані умови та бути неоптимальним розв'язком. З позитивного боку, якщо цілочисельні розв'язки набувають досить великих значень або якщо точність умов можна поставити під сумнів. Також допускається підхід до розв'язку задач з попередньою обробкою (додати умову до задачі, щоб рішення наближалося до цілих значень)

2.2. Методи пошуку

2.2.1. Метод Гоморі

Алгоритм січних площин Гоморі намагається вирішити цілі задачі програмування, відсікаючи рішення для релаксації лінійного програмування, поки не знайдеться ціле рішення. Він був створений в 1950-х роках американським математиком Ральфом Гоморі. Надрізи створюються з рядів оптимального симплексного розслаблення LP. Якщо не

всі основні змінні є цілими, то існує рядок i, який має дробове значення $B^{-1}b$. З цього моменту $B^{-1}b$ будемо називати просто b. Цей рядок має форму:

$$x_{B_i} + \sum_{J \in NB} a_{ij} x_j = b_i \qquad (2.2.1.1)$$

Де NB - сукупність не основних змінних у симплекс-таблиці. Перезапис у дробових і цілих частинах дає вихід:

$$x_{B_i} + \sum_{J \in NB} [a_{ij}] x_j + \sum_{J \in NB} (a_{ij}) x_j = [b_i] + f(b_i) \qquad (2.2.1.2)$$

Де $f(a) = a - [a]$ Зробивши деякі перетворення в попередній формулі маємо:

$$x_{B_i} + \sum_{J \in NB} [a_{ij}] x_j = [b_i] + f(b_i) - \sum_{J \in NB} (a_{ij}) x_j \qquad (2.2.1.3)$$

Записуємо як нерівність:

$$x_{B_i} + \sum_{J \in NB} [a_{ij}] x_j \leq [b_i] + f(b_i) \qquad (2.2.1.4)$$

Оскільки все ціле, окрім $f(b_i)$то виконується також така нерівність:

$$x_{B_i} + \sum_{J \in NB} [a_{ij}] x_j \leq [b_i] \qquad (2.2.1.5)$$

Оптимальне рішення для релаксації ЛП буде порушувати обмеження (2.2.1.5), якщо b_i не є цілим.

Зауважимо, що з розв'язування рівняння (2.2.1.1) для $x_{!!}$ маємо:

$$x_{B_i} = [b_i] - \sum_{J \in NB} [a_{ij}] x_j \qquad (2.2.1.6)$$

Якщо x_{B_i} в (2.2.1.6) заміщено на (2.2.1.5), то справедлива наступна нерівність:

$$x_{B_i} + \sum_{J \in NB} f(a_{ij}) x_j \geq f(b_i) \qquad (2.2.1.7)$$

Дробові розрізи Гоморі можна посилити декількома різними способами. Перше, про що піде мова це – для будь-якого цілого t, розріз

$$x_{B_i} + \sum_{J \in NB} f(ta_{ij}) x_j \geq f(tb_i) \qquad (2.2.1.8)$$

задовольняється всіма від'ємними цілими рішеннями (1.6), а отже усіма рішеннями. Також, якщо $f(b_i) < \frac{1}{2}$, t додатне, а $\frac{1}{2} \leq t * f(b_i) <$, тоді (2.2.1.8) домінує над (2.2.1.7).

Домінуючі нерівності є дійсними, вони відрізають більш можливими областями, ніж нерівності, над якими вони домінують. Згідно з визначенням:

Якщо $\pi x \leq \pi_0$ і $\mu x \leq \mu_0$ *– дві дійсні нерівності,* $\pi x \leq \pi_0$ *домінуєна* $\mu x \leq \mu_0$*, якщо існує* $u > 0$ *таке що* $\pi > \mathrm{u}\mu$ *і* $\pi_0 > \mathrm{u}\mu_0$ *і* $(\pi, \pi_0) \neq (\mathrm{u}\mu, \mathrm{u}\mu_0)$.

Друга методика - це посилюючий розріз, який можна отримати, переглядаючи змішані цілі надрізи Гоморі. Використаємо теорему:

Якщо $X = \{(x, y) \in \mathbb{R}_+^1 \times \mathbb{Z}^1 : y \leq b + x\}$ *і* $f = b - [b] > 0$, нерівність$[b] + \frac{x}{f-1}$ вірна для X.

Розглянемо лише цілі частини змішаних цілих відрізків. Для цього, як і раніше, повинен бути ряд і, який має дробову праву частину b:

$$x_{B_i} + \sum_{J \in NB} a_{ij} x_j = b_i \qquad (2.2.1.9)$$

Ми можемо розбити не базові змінні на два набори:

$$x_{B_i} + \sum_{f_j \le f_0} a_{ij}x_j + \sum_{f_j > f_0} a_{ij}x_j = b_i \qquad (2.2.1.10)$$

Де $f_0 = f(a_{ij})$ і $f_0 = f(b_i)$ Можемо переписати (2.2.1.10) як:

$$x_{B_i} + \sum_{f_j \le f_0} [a_{ij}]x_j + \sum_{f_j > f_0} [a_{ij}]x_j = b_i - \sum_{f_j \le f_0} f_j x_j + \sum_{f_j > f_0} (1 - f_j)x_j \qquad (2.2.1.11)$$

Запишемо (2.2.1.11) як нерівність:

$$x_{B_i} + \sum_{f_j \le f_0} [a_{ij}]x_j + \sum_{f_j > f_0} [a_{ij}]x_j \le [b_i] - \sum_{f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j \qquad (2.2.1.12)$$

Підставляємо x_{B_i} з (2.2.1.10) у (2.2.1.12) та спрощуємо:

$$x_{B_i} + \sum_{f_j \le f_0} [a_{ij}]x_j + \sum_{f_j > f_0} [a_{ij}]x_j \le [b_i] - \sum_{f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j \qquad (2.2.1.13)$$

$$f_0 \le \sum_{f_j \le f_0} f_j x_j + \sum_{f_j > f_0} \frac{f_0(1 - f_j)}{1 - f_0} x_j \qquad (2.2.1.14)$$

Зауважимо, що $f_j \le f_0$ тоді:

$$f_j \le f_0 = f_0 \frac{1 - f_0}{1 - f_0} \le f_0 \frac{1 - f_j}{1 - f_0} \qquad (2.2.1.15)$$

І коли $f_j > f_0$, тоді:

$$f_0 \frac{1 - f_j}{1 - f_0} \le f_0 \frac{1 - f_0}{1 - f_0} = f_j \le f_0 \qquad (2.2.1.16)$$

Справедливо і (2.2.1.16) сильніше за (2.2.1.7)

Загальний алгоритм Гоморі:

1. Знайти симплекс таблицю
2. Знайти січні площини Гоморі, пов’язані з кожним рядом, який має дробову частину
3. Додайте ці площини до симплекс таблиці
4. Використовуйте алгоритмдвоїстого симплекса, щоб знайти розв’язок нового лінійного програмування.
5. Якщо оптимально, зупиніться.
6. В іншому випадку перейдіть до кроку 2

В основі методу Гоморі закладена ідея, яка полягає в тому, що спочатку вирішується завдання лінійного програмування без урахування умов цілочисельності. Якщо отриманий таким чином розв’язок цілочисельний, то він приймається за оптимальний план завдання. Якщо ж розв’язок нецілочисельний, то система обмежень доповнюється умовою, яке відсікає від безлічі планів задачі нецілочисельного оптимального плану, але при цьому зберігають цілочисельні вершини безлічі планів. Потім розв’язується завдання лінійного програмування з додатковою умовою. Якщо ж і після цього не для всіх змінних виконується умова цілочисельноті, то вводиться нова. Умови-відсікання вибираються таким чином, щоб за кінцеве число кроків прийти до цілочисельного розв’язку (якщо він існує).

2.2.2. Метод гілок та меж

Метод, пов'язаний з розгалуженням, ефективний для вирішення задач програмування змішаних цілих чисел і нелінійних програм. Познайомимось з ідеєю цього метода:

$$Minimize\ f(X) \qquad (2.2.2.1)$$

За умов:

$$g_i(X) \geq 0, \qquad j = 1, 2, \dots, m \tag{2.2.2.2}$$

$$h_k(X) \geq 0, \qquad k = 1, 2, \dots, p \tag{2.2.2.3}$$

$$x_j = integer, \qquad j = 1, 2, \dots, n_0 (n_0 \leq n) \tag{2.2.2.4}$$

Де $X = \{x_1, x_2, \dots, x_n\}^T$. Зауважимо, що у векторі X перші n0 змінних – цілі. Вектор X називається безперервним можливим, якщо (2.2.2.2) і (2.2.2.3) задовольняє обмеженням. Вектор X, який задовольняє всі обмеження, рівняння (2.2.2.2) до (2.2.2.4) називається цілим можливим рішенням.

У методі, пов'язаному з розгалуженням, ціла задача не вирішується безпосередньо. Скоріше, метод спочатку вирішує неперервну задачу, отриману шляхом послаблення цілих обмежень на змінні. Якщо рішення неперервної задачі виявляється цілим числом, це оптимальне рішення задачі. В іншому випадку принаймні одна з цілих змінних, скажімо, x_i, повинна приймати значення, що не є інтегралом. Якщо x_i не є цілим числом, ми завжди можемо знайти ціле число $[x_i]$ таке, що

$$[x_i] < x_i < [x_i] + 1 \tag{2.2.2.5}$$

Потім формуються дві підпрограми, одна з додатковою верхньою межею обмеження

$$x_i \leq [x_i] \tag{2.2.2.6}$$

та інша із нижньою межею обмеження

$$[x_i] \geq [x_i] + 1 \tag{2.2.2.7}$$

Процес пошуку цих підпрограм називається розгалуженням, воно виключає деяку частину безперервного простору. Кожна з цих двох підпрограм розв’язується як повноцінна задача. Видно, що рішення неперервної задачі утворює вузол, і від кожного вузла можуть виникати дві гілки.

Процес розгалуження та розв’язання послідовності безперервних задач, обговорених вище, продовжується до тих пір, поки не знайдеться ціле можливе рішення однієї з двох задач. Коли таке можливе ціле рішення знайдено, відповідне значення цільової функції стає верхньою межею щодо мінімального значення цільової функції. На цьому етапі ми можемо виключити з подальшого розгляду всі безперервні рішення (вузли), значення об'єктивних функцій яких перевищують верхню межу. Кажуть, що вузли, які усуваються, були розроблені, тому що неможливо знайти краще ціле рішення з цих вузлів (простору рішення), ніж у нас зараз. Значення верхньої межі цільової функції оновлюється кожного разу, коли знаходиться краща межа.

Алгоритм методу гілок та меж:

1. Знаходимо рішення задачі лінійного програмування без урахування цілочисельності.
2. Накладаємо допоміжні обмеження на дробову компоненти.
3. Розв’язуємо дві задачі з обмеженнями на компоненту.
4. Накладаємо в разі потреби додаткові обмеження, згідно з можливими випадками отримуємо оптимальний цілочисельний план або встановлюємо нерозв'язність задачі.

2.2.3 Циклічний алгоритм цілочисельного програмування

Маємо задачу нелінійного програмування з дискретними змінними:

$$Minimize\ f(X) \tag{2.2.3.1}$$

За умов:

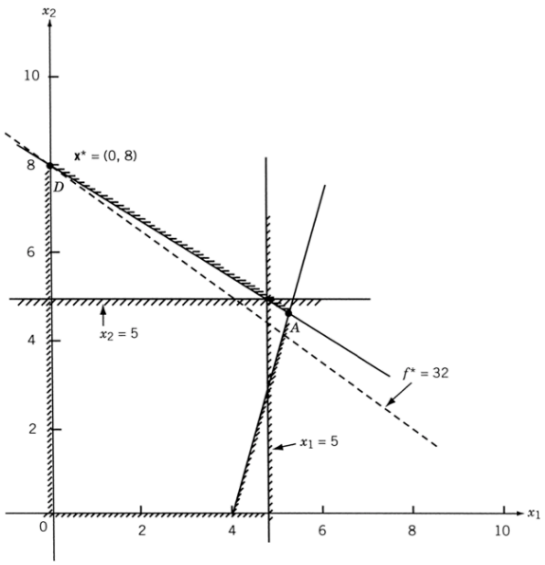
$$g_i(X) \leq 0, \qquad j = 1, 2, \dots, m \tag{2.2.3.2}$$

$$h_k(X) = 0, \qquad k = 1, 2, \dots, p \tag{2.2.3.3}$$

$$x_j \in \{d_{i1}, d_{i2}, \dots, d_{iq}\}, \qquad i = 1, 2, \dots, n_0 \tag{2.2.3.4}$$

$$x_i^{(l)} \leq x_i \leq x_i^{(u)}, \qquad i = n_0 + 1, i = n_0 + 2, \dots, n \tag{2.2.3.5}$$

де змінні n_0 вважаються дискретними, d_{ij} – j-й дискретні значення для змінної i, та $X = \{x_1, x_2, \dots, x_n\}$. Можна знайти рішення шляхом розв'язку ряду задач лінійного програмування змішаних цілих чисел.



Picture 2. 1

Нелінійні вирази в рівняннях (2.2.3.1) до (2.2.3.3) лінеаризовані X^0, використовуючи розширення ряду Тейлора першого порядку, бачимо:

$$Minimize\ f(X) \approx f(X^0) + \nabla f(X^0)\delta X \tag{2.2.3.6}$$

За умов:

$$g_i(X) \approx g_i(X^0) + \nabla g_i(X^0)\delta X \leq 0, \quad j = 1, 2, \dots, m \tag{2.2.3.7}$$

$$h_k(X) \approx h_k(X^0) + \nabla h_k(X^0)\delta X = 0, \quad k = 1, 2, \dots, p \tag{2.2.3.8}$$

$$x_i^0 + \delta x_j \in \{d_{i1}, d_{i2}, \dots, d_{iq}\}, \qquad i = 1, 2, \dots, n_0 \tag{2.2.3.9}$$

$$x_i^{(l)} \leq x_i^0 + \delta x_j \leq x_i^{(u)}, \qquad i = n_0 + 1, i = n_0 + 2, \dots, n \tag{2.2.3.10}$$

$$\delta X = X - X^0 \tag{2.2.3.11}$$

Задачу викладену рівняннях (2.2.3.6) до (2.2.3.11) не можна розв’язати, використовуючи змішані цілі методи лінійного програмування, оскільки деякі змінні проектування дискретні та нецілі. Робимо перетворення:

$$x_i = y_{i1}d_{i1} + y_{i2}d_{i2} + \cdots + y_{iq}d_{iq} = \sum_{j=1}^{q} y_{ij}d_{ij}, \qquad i = 1, 2, \dots, n_0 \tag{2.2.3.12}$$

$$y_{i1} + y_{i2} + \cdots + y_{iq} = \sum_{j=1}^{q} y_{ij} = 1 \tag{2.2.3.13}$$

$$y_{ij} = 0 \text{ або } 1, \qquad i = 1, 2, \dots, n_0, j = 1, 2, \dots, q \tag{2.2.3.14}$$

Використовуючи рівняння (2.2.3.13) до (2.2.3.14) в рівняннях (2.2.3.6) до (2.2.3.11), отримуємо:

$$Minimize\ f(X) \approx f(X^0) + \sum_{i=1}^{n_0} \frac{\partial f}{\partial x_i}\left(\sum_{j=1}^{q} y_{ij} d_{ij} - x_i^0\right) + \sum_{i=n_0+1}^{n_0} \frac{\partial f}{\partial x_i}(x_i - x_i^0) \quad (2.2.3.15)$$

За умов:

$$g_i(X) \approx g_i(X^0) + \sum_{i=1}^{n_0} \frac{\partial g_i}{\partial x_i}\left(\sum_{j=1}^{q} y_{ij} d_{ij} - x_i^0\right) + \sum_{i=n_0+1}^{n_0} \frac{\partial g_i}{\partial x_i}(x_i - x_i^0) \le 0, \quad j = 1, 2, \dots, m \quad (2.2.3.16)$$

$$h_k(X) \approx h_k(X^0) + \sum_{i=1}^{n_0} \frac{\partial h_k}{\partial x_i}\left(\sum_{j=1}^{q} y_{ij} d_{ij} - x_i^0\right) + \sum_{i=n_0+1}^{n_0} \frac{\partial h_k}{\partial x_i}(x_i - x_i^0) \le 0, \quad k = 1, 2, \dots, p \quad (2.2.3.17)$$

$$\sum_{j=1}^{q} y_{ij} = 1, \qquad i = 1, 2, \dots, n_0 \quad (2.2.3.18)$$

$$y_{ij} = 0 \text{ або } 1, \quad i = n_0 + 1, i = n_0 + 2, \dots, n \quad (2.2.3.19)$$

Задача, викладена в рівняннях (2.2.3.18) до (2.2.3.19) тепер можна вирішити як проблему лінійного програмування зі змішаним цілим числом.

РОЗДІЛ 3. ПРАКТИЧНА ЧАСТИНА. ЗАСТОСУВАННЯ ЗАДАЧ ЦІЛОЧИСЕЛЬНОГО ПРОГРАМУВАННЯ.

У цьому розділі, на прикладах задач з цілочисельними умовами:

- ми реалізуємо методи пошуку розв'язків задач, використовуючи мову програмування Swift
- розробимо мобільні додатки під операційну систему IOS, для демонстрації роботи алгоритмів: перегляду вхідних та вихідних даних, зміни умови задачі та отримання кінцевого результату.

3.1 Застосування задачі пакування ранцю

Розглянемо задачу. Турист планує відпочинок на вихідних у горах, отже йому потрібно взяти з собою речі, які можуть знадобитись у подорожі (їжа, одяг і т.п.). У нього є рюкзак, який розрахований на 4 кг.

Турист створює список речей, які він хотів би узяти з собою, але їх загальна вага у списку, значно перевищує вміст рюкзаку. Отже для кращого сортування хлопець додає "пріорітет" у шкалі від 1 до 200, для кожного елементу зі списку, що символізує значимість цього предмету для нього, у подорожі.

Таблиця містить предмети, що хотів би взяти турист, їх вагу у грамах, заплановану кількість від кожного предмету та важливість.

Список предметів:

№	**Предмет**	**Вага (г.)**	**Пріорітет**	**К-сть**
1	Мапа	90	150	1
2	Компас	130	35	1
3	Вода	1530	200	2
4	Бутерброд	500	60	2

5	Цукор	150	60	2
6	Банка	680	45	3
7	Банан	270	60	3
8	Яблуко	390	40	3
9	Сир	230	30	1
10	Сік	520	10	3
11	Крем від опіків	110	70	1
12	Фотоапарат	320	30	1
13	Футболка	240	15	2
14	Штани	480	10	2
15	Парасолька	730	40	1
16	Водонепроникні штани	420	70	1
17	Блокнот	220	80	1
18	Окуляри	70	20	1
19	Рушник	180	12	1
20	Шкарпетки	40	50	1
21	Книга	300	10	2
22	Дощовик	430	75	1
	Загальний вміст рюкзаку	≤ 4000	-	-

Таблиця 3. 1

Турист може обрати будь-яку комбінацію предметів зі списку, і доступну кількість кожного предмету.

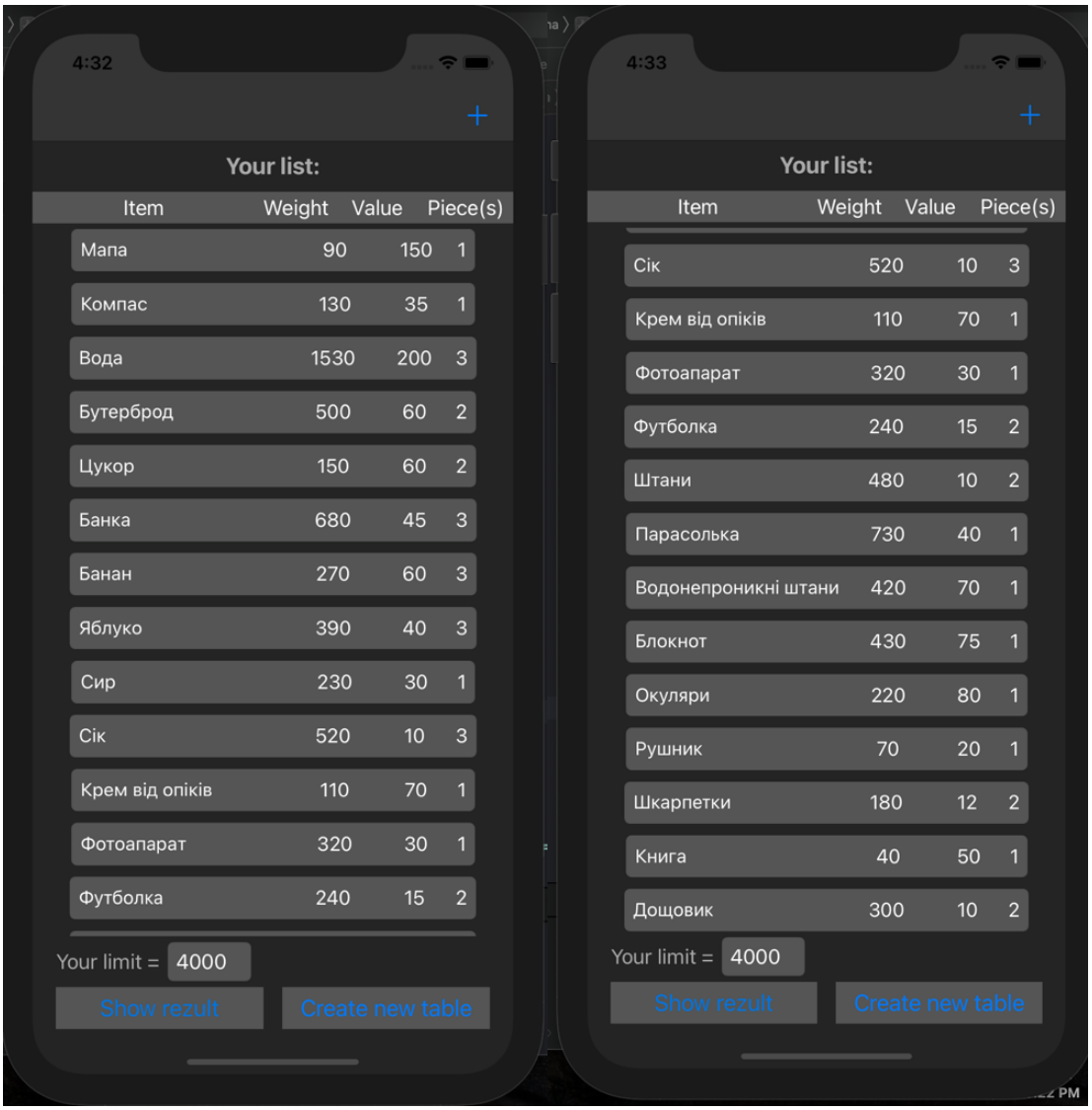
Він не має змоги розрізати чи ділити предмети, тому може взяти лише цілі одиниці.

Завдання

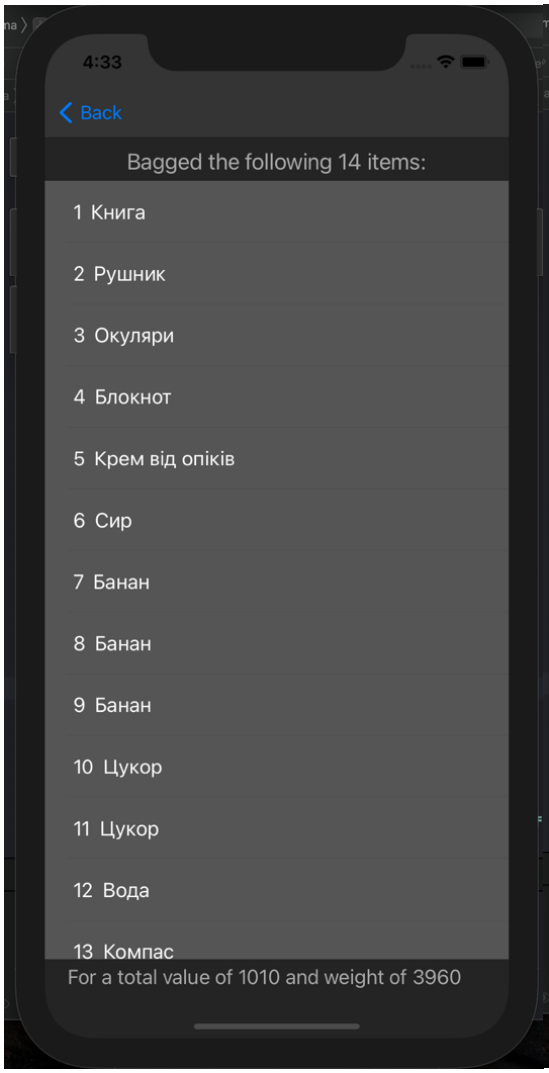
Показати, які предмети турист може покласти у рюкзак, щоб їх загальна вага не перевищувала 4 кг, і значення було максимальним.

Реалізація алгоритму представлена у Додатку А

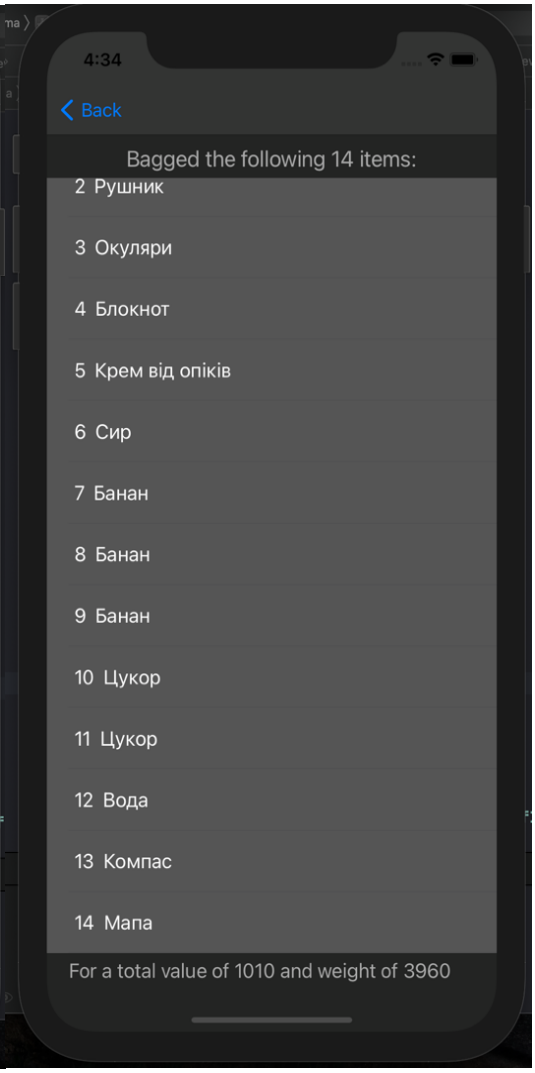
На базі розглянутої задачі, було розроблено мобільний додаток як було зазначено на початку розділу. У програмі продемонстрована задана задача та є можливість як додати предмет до таблиці так і повністю змінити вхідні данні.



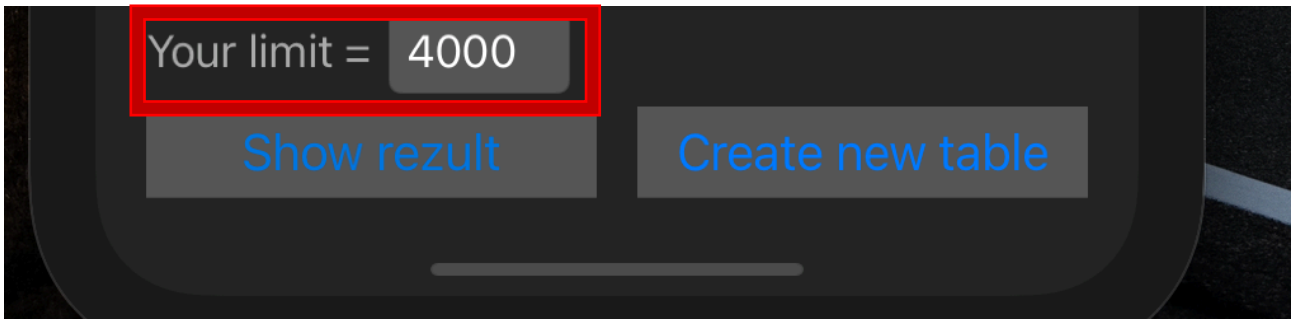
Picture 3. 1 *Picture 3. 2*



Picture 3. 3

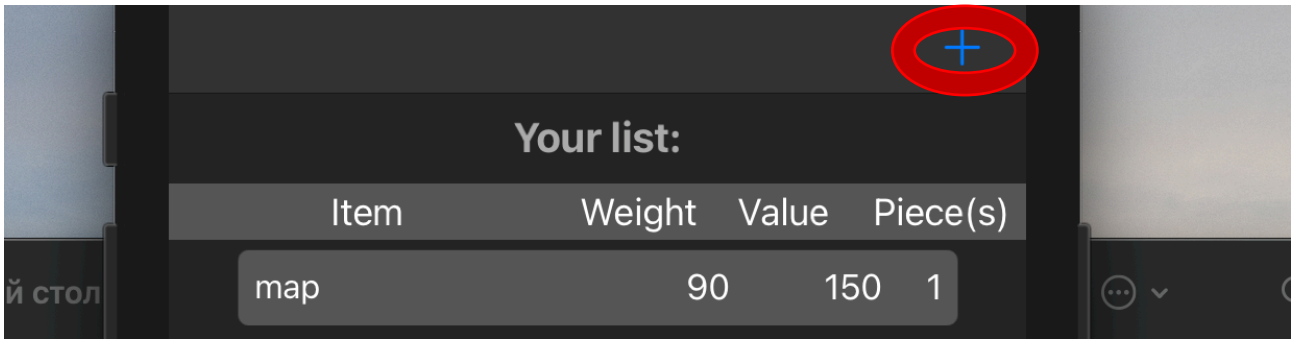


Picture 3. 4



Picture 3. 5

На Picture 3. 5 зображено поле у якому ви можете змінити максимальну вагу рюкзака.



Picture 3. 6

На Picture 3. 6 показано кнопку, натиснувши яку ви матимете змогу додати новий елемент у таблицю, як у початковому варіанті так і у кастомному.

Picture 3. 7 *Picture 3. 8*

Якщо ви натиснете на копку зображену на Picture 3. 7, то перейдете у кастомний режим та зможете додати власні вхідні данні.

Зображена кнопка на Picture 3. 8, перенаправляє вас на інший екран, де власне і відображаються отримані результати.

Код програми надано у Додатку Б.

Дана задача широко використовується у логістиці. Як приклад типового логістичного завдання наведемо елементарну задачу. Припустимо, що компанія з транспортних перевезень отримала замовлення: необхідно доставити зі складу в магазин товар. Компанія має лише один вільний автомобіль на запланований день, причому вміст цього транспорту має максимальне значення – 1000 кг. Замовник – це власник меблевою компанії і предмети інтер'єру, що повинні перевезти, мають бути неподільними. Отже наше завдання з отриманого переліку сформувати оптимальний вміст найбільшого автомобіля з замовленням, для того, щоб розрахувати затрачений час, вартість перевезення і т. п.

Задамо таблицю значень замовлення, за прикладом розглянутої вище задачі, у колонці пріорітет ми зазначимо необхідність магазину у даному товарі, вагу зазначимо у кг :

Стіл	15	167	8
Шафа	50	100	5
ЛіжкоА	20	135	15
Стілець	5	170	15
ЛіжкоБ	35	160	23
СтілА	22	76	9
Дзеркало	7	140	10
Полиця	3	50	25
СтілецьА	11	156	5
ПартаА	30	120	10
Дошка	17	180	4

Таблиця 3. 2

Задамо значення з Таблиця 3. 2 , як вхідні данні до мобільного додатку та вкажемо ліміт.

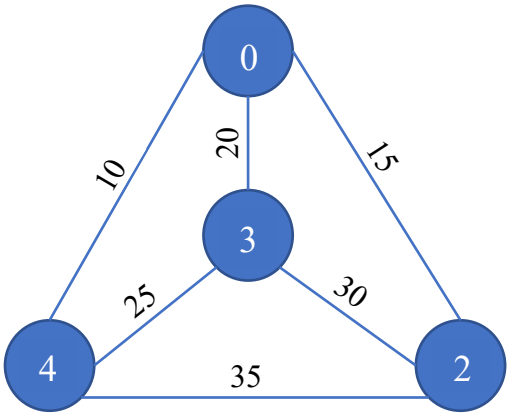
Picture 3. 9

Отже на Picture 3. 9 зображено список отриманих результів, усього у автомобіль поміститься 89 предметів з загальною вагою 998 кг, що задовольняє умові задачі. Отримані результати надані у Додатку Г.

Отже розроблений нами додаток можна використовувати у реальному житті, для розв’язку задачі пакування ранцю з цілими значеннями.

3.2 Задача Комівояжера

Для початку давайте розглянемо елементарну задачу. Спочатку реалізуємо алгоритм меж і гілок для графу Picture 3.10, з використанням мови програмування Swift:



Picture 3. 10

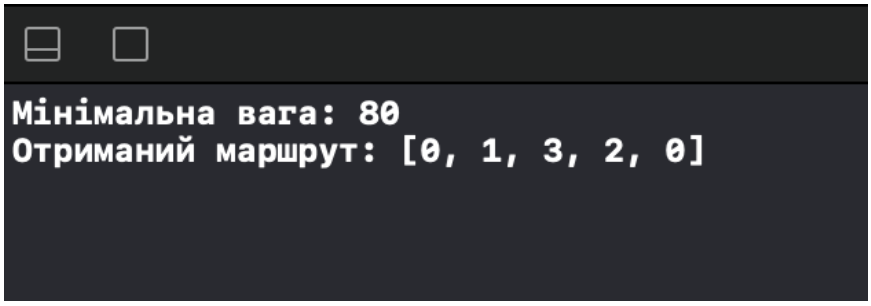
Розв'язком задачі Комівояжера для даної умови є шлях [0, 1, 3, 2, 0], з ціною 80.

Реалізацію алгоритму з поясненням надано у Додатку А.

```
// Driver
var solver = TSPSolver()
var matrix = [
            [0, 10, 15, 20],
            [10, 0, 35, 25],
            [15, 35, 0, 30],
            [20, 25, 30, 0],
          ]
solver.TSP(matrix: matrix)
print("Мінімальна вага: \(solver.finalRes)")
print("Отриманий маршрут: \(solver.finalPath)")
```

Picture 3. 11

Далі задаємо умови задачі, що розглядається, і перевіряємо коректність роботи запрограмованого алгортиму.



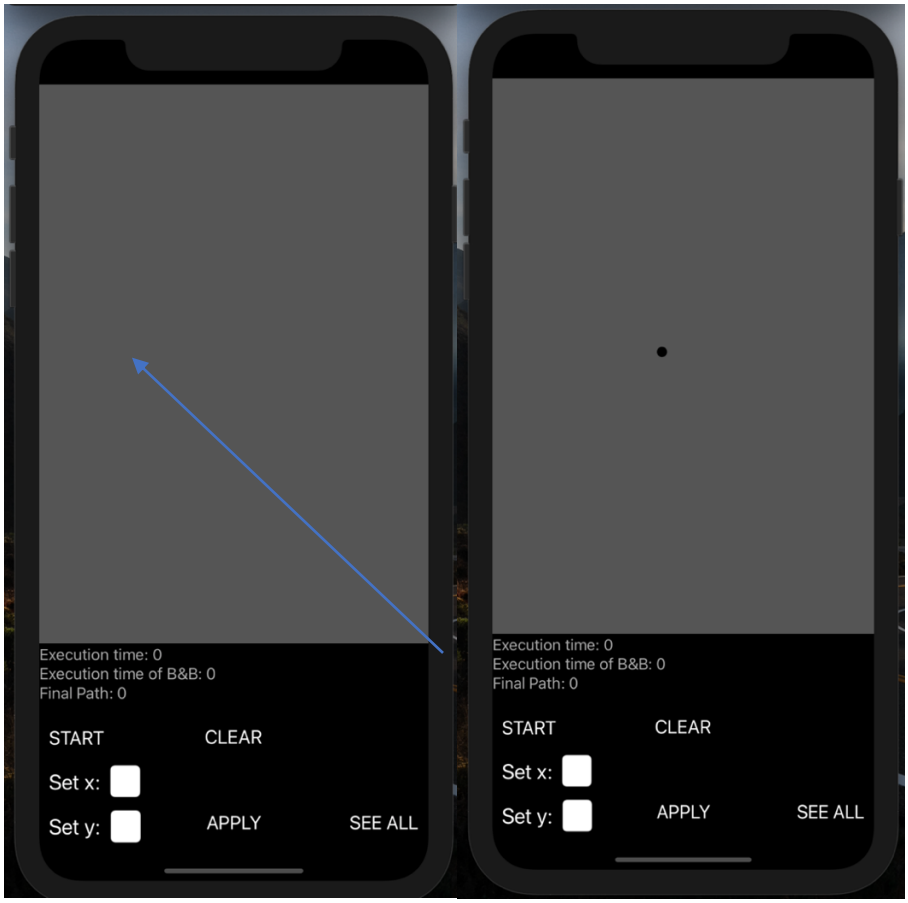
Picture 3. 12

Після того, як ми переконались у коректності роботи алгоритму. Переходимо до наступного пункту поставлених завдань, для даного розділу. Інтерфейс програми зображено Picture 3.13

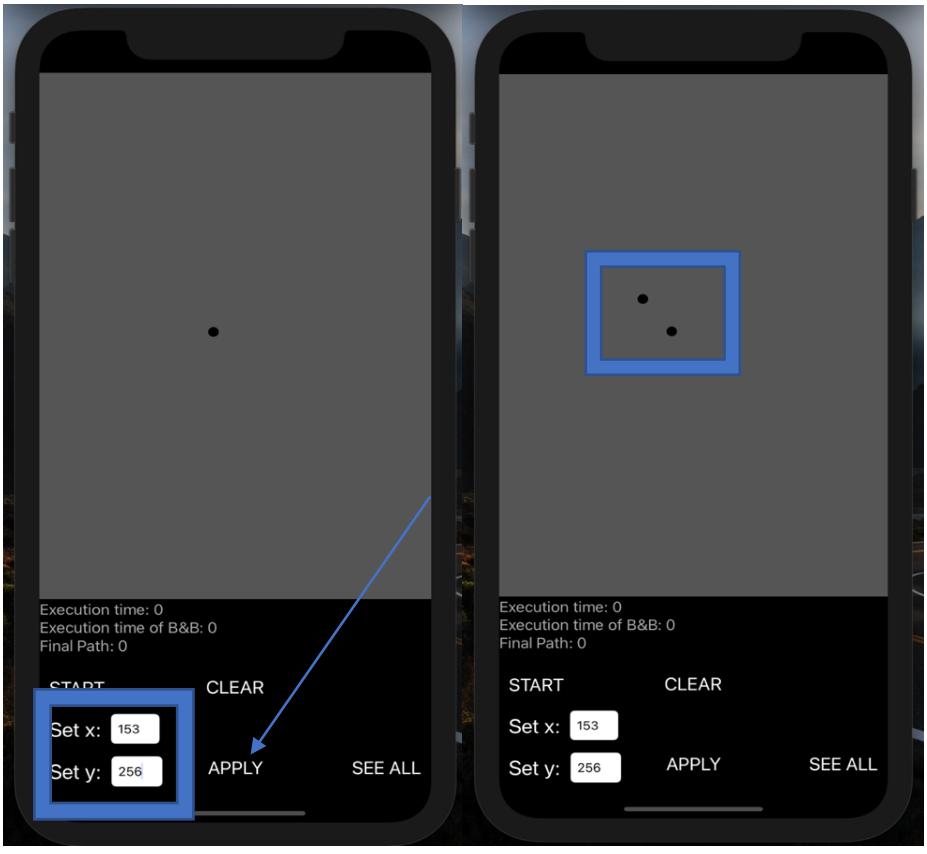
Користувач має змогу додати вузол у два варіанти:

- натиснувши на сіре поле на екрані, як зображено на Picture 3.13 (вузли додаються до масиву у порядку додавання на екран)
- задавши координати, вершини власноруч як зображено на Picture 3.14

Перевірити задані координати вузлів та коректність внесення вузлів у масив, можна натиснувши “SeeAll”

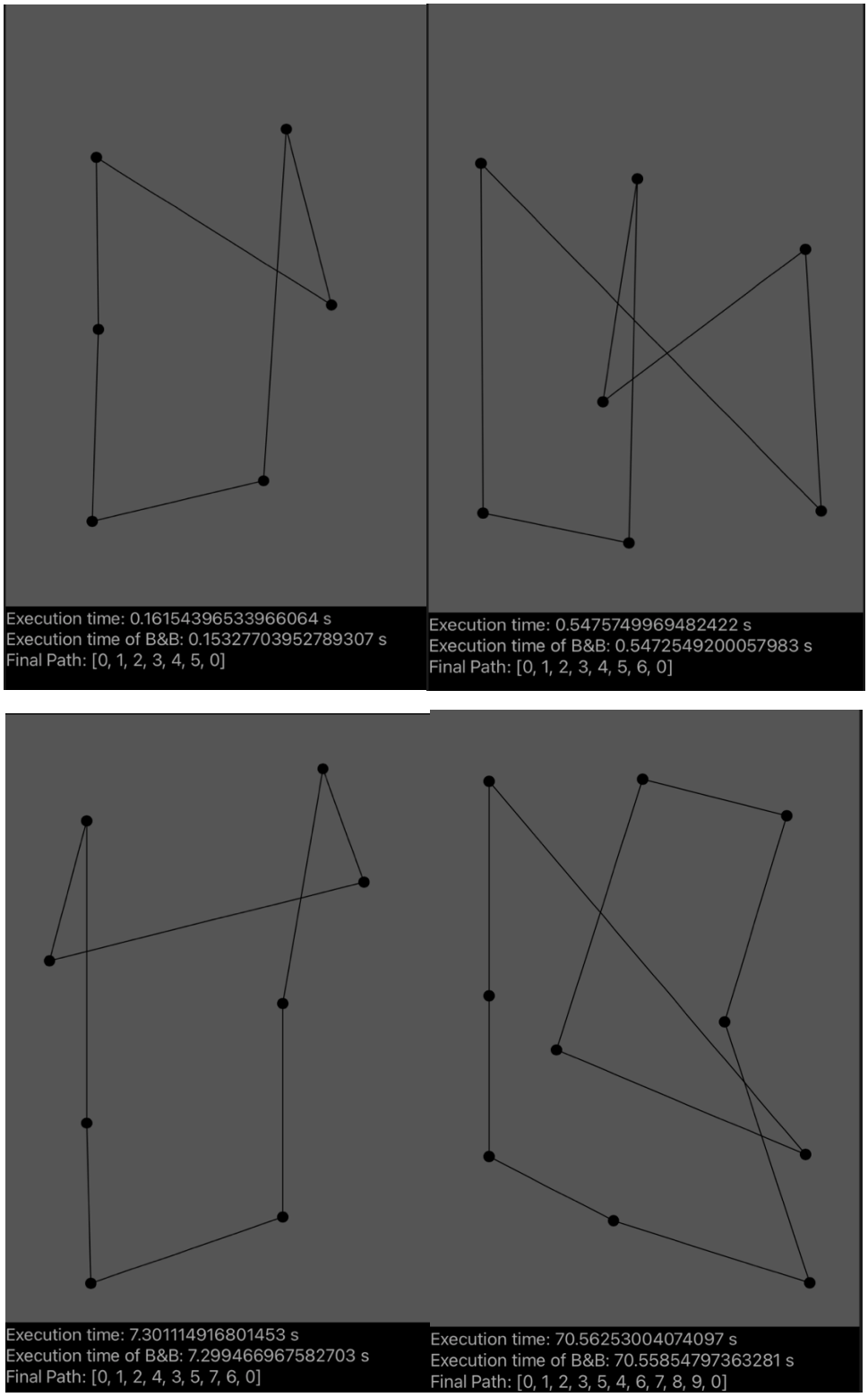


Picture 3. 13



Picture 3. 13

Приклади відображення отриманих результатів зображено на Picture 3.14. Із зазначенням часу виконання та кінцевого результату. Код програми надано у Додатку В.



Picture 3. 14

ВИСНОВОК

Під час виконання роботи був проведений аналіз наукової та навчальної літератури з теми дослідження, були розглянуті задачі та алгоритми розв’язку цілочисельного програмування.

Також, на прикладах задач з цілочисельного програмування:

1. Ми реалізували метод пошуку розв’язків задач, використовуючи мову програмування Swift
2. На базі розглянутих задач, розробили дві програми, використовуючи які ми можемо:
    - продемонструвати застосування розв’язку задач
    - задати власну умову
    - отримати кінцевий розв’язок
    - отримати час знаходження оптимального розв’язку

Підсумовуючи, можемо зробити наступні висновки:

1. У результаті проведенних досліджень були проаналізовані алгоритми розв’язку задач цілочисельного програмування: метод Гоморі, метод гілок та меж, циклічний алгоритм цілочисельного програмування.
2. Проблеми цілочисельного програмування - актуальні та широко використовувані в наш час. Їх розвиток не стоїть на місці, а навпаки удосконалюється та помітно ускладнюється.
3. Завдання даних типів є вагомим, так як до їх вирішення зводиться аналіз різноманітних ситуацій, що виникають в економіці, техніці, військовій справі та інших галузях. Також розв’язок таких задачзавдання вважається цікави і з математичної точки зору.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Кузнецов А. В., Сакович В. А., Холод Н. И. Высшая математика. Математическое программирование.. Минск: Вышэйшая школа, 2001. 351с.
2. В.Г.Карманов. Математическое программирование: Учебное пособие – 5-е издание, стереотип-М:ФИЗМАТ, 2001г.-264с.
3. Е.Г.Белоусов. Введение в выпуклый анализ и целочисленное программирование. М.:Издательство МГУ, 1977г.
4. Miller, C. E., A. W. Tucker, R. A. Zemlin. 1960. Integer programming formulation of traveling sales- man problems. Journal of the Association for Computing Machinery 7, 326–329
5. Miliotis,P.1978.Using cutting planes to solve the symmetric traveling salesman problem. Mathematical Programming 15, 177–188
6. Ананий В. Левитин Глава 3. Метод грубой силы: Задача о рюкзаке // Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Aigorithms. — М.: «Вильямс», 2006. — С. 160-163. — ISBN 0-201- 74395-7
7. A. Schrijver. Theory of linear and integer programming. Wiley, Chichester,1986.
8. G.L. Nemhauser, L.A. Wolsey. Integer and Combinatorial Optimization. John Wiley, & Sons, 1988.
9. Williams, H.P. (2009). Logic and integer programming. International Series in Operations Research & Management Science. 130.
10. Araoz J, Evans L, Gomory RE, Johnson EL (2003) Cyclic group and knapsack facets. Math Program Ser B 96:377–408
11. Schrijver A (1986) Theory of linear and integer programming, Wiley interscience series indiscrete mathematics and optimization. Wiley, New York

12. Schrijver A (2002) Combinatorial optimization. Springer, Berlin, Vol A Chapters 1–38, pp 1–648; Vol B Chapters 39–69, pp 649–1218; Vol C Chapters 70–83, pp 1219–1882
13. M. Padberg, G. Rinaldi A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems
SIAM Rev., 33 (1991), pp. 60-100
14. TSPLIB—a traveling salesman problem library ORSA J. Comput., 3 (1991), pp. 376-384
15. P.J. Kolesar A branch and bound algorithm for the knapsack problem
Manage. Sci., 13 (1967), pp. 723-735
16. J. Clausen Branch and bound algorithms-principles and examples. Technical Report Department of Computer Science, University of Copenhagen (1999)

ДОДАТОК А

1. Реалізація алгоритму для задачі пакування ранцю

```
import Foundation

public struct KnpItem: Hashable {
  public var name: String
  public var wght: Int
  public var prior: Int

  public init(name: String, wght: Int, prior: Int) {
    self.name = name
    self.wght = wght
    self.prior = prior
  }
}

typealias GroupedOfItem = (name: String, wght: Int, prior: Int,
n: Int)

class Service {

    public func knapsack(items: [KnpItem], limit: Int) ->
[KnpItem] {

      var table = Array(repeating: Array(repeating: 0, count:
limit + 1), count: items.count + 1)

      for j in 1...items.count {
        let item = items[j-1]

        for w in 1...limit {
          if item.wght > w {
            table[j][w] = table[j-1][w]
```

```
          } else {
            table[j][w]   =   max(table[j-1][w],   table[j-1][w-
item.wght] + item.prior)
          }
        }
      }

      var rez = [KnpItem]()
      var w = limit

      for j in stride(from: items.count, to: 0, by: -1) where
table[j][w] != table[j-1][w] {
        let item = items[j-1]

        rez.append(item)

        w -= item.wght
      }

      return rez
    }
}



var groupedOfItems: [GroupedOfItem] = [
      ("Мапа", 90, 150, 1),
      ("Компас", 130, 35, 1),
      ("Вода", 1530, 200, 3),
      ("Бутерброд", 500, 60, 2),
      ("Цукор", 150, 60, 2),
      ("Банка", 680, 45, 3),
      ("Банан", 270, 60, 3),
      ("Яблуко", 390, 40, 3),
      ("Сир", 230, 30, 1),
```

```
        ("Сік", 520, 10, 3),
        ("Крем від опіків", 110, 70, 1),
        ("Фотоапарат", 320, 30, 1),
        ("Футболка", 240, 15, 2),
        ("Штани", 480, 10, 2),
        ("Парасолька", 730, 40, 1),
        ("Водонепроникні штани", 420, 70, 1),
        ("Блокнот", 430, 75, 1),
        ("Окуляри", 220, 80, 1),
        ("Рушник", 70, 20, 1),
        ("Шкарпетки", 180, 12, 2),
        ("Книга", 40, 50, 1),
        ("Дощовик", 300, 10, 2)
    ]

// Код, що використовується у ViewController.swift
   let items = modelView.groupedItems.flatMap({item in
            (0..<item.n).map({_  in  KnpItem(name:  item.name,
wght: item.wght, prior: item.prior) })
          })
         guard let limitText = limitTextField.text else {return}
         let limit = Int(limitText) ?? 0
         let  bagged  =  service.knapsack(items:  items,  limit:
limit)
         let  (totalVal,  totalWeight)  =  bagged.reduce((0,  0),
{cur, item in (cur.0 + item.prior, cur.1 + item.wght) })
```

2. Реалізація алгоритму для задачі Комівояжера

```
import Foundation
class TSPSolver {
   let N = 4
   // finalPath – зберігає кінцевий розв'язок
   lazy var finalPath: [Int] = Array(repeating: 0,
count: self.N + 1)

    // visited – відстежує вузли, що були відвідані на заданому
маршруті
    lazy var visited: [Bool] = Array(repeating: false, count:
self.N)

    // finalRes – зберігає мінімальну кінцеву вагу, найкоротшого
кінцевого
    // маршруту
    var finalRes = LONG_MAX
    // copyToFinal – функція, що зберігає тимчасовий розв'язок
у кінцевий
    func copyToFinal(currentPath: [Int]) {
        for i in 0...N-1 {
            finalPath[i] = currentPath[i]
        }
        finalPath[N] = currentPath[0]
    }

    // firstMin – знаходить мінімальну вартість ребра,
    // кінець якого знаходиться у вершині i
    func firstMin(matrix: [[Int]], i: Int) -> Int {
        var min = LONG_MAX
        for k in 0...N-1 {
            if matrix[i][k] < min && i != k {
                min = matrix[i][k]
            }
```

```
        }
        return min
    }

    // secondMin - знаходить другу мінімальну вартість
    // ребра, кінець якого знаходиться у вершині i
    func secondMin(matrix: [[Int]], i: Int) -> Int {
        var first = LONG_MAX
        var second = LONG_MAX

        for j in 0...N-1 {
            if i == j{
                continue
            }

            if matrix[i][j] <= first {
                second = first
                first = matrix[i][j]
            } else if matrix[i][j] <= second && matrix[i][j] !=
first {
                second = matrix[i][j]
            }
        }
        return second
    }

    // tspRec – функція, що примає такі вхідні данні
    // currentBound - нижня межа кореневого вузла
    // currentWeight - зберігає вагу маршруту на даний момент
    // level - зберігає рівень переміщення по дереву пошуку на
даний момент
    // currentPath - зберігає розв'язок, що згодом буде
скопійовано до finalRes
    func tspRec(matrix: [[Int]], currentBound: Int,
currentWeight: Int, level: Int, currentPath: [Int]) {
        var currWeight = currentWeight
```

```
        var currBound = currentBound
        var currPath = currentPath

        // Базовий випадок, що відбудеться лише у тому
        // випадку, коли ми дістанемось N-го рівня.
        // Це означає те, що ми покрили усі вузли за один раз.
        if level == N {

            // Перевіряємо наявність ребра на шляху від
            // останньої вершини до першої
            if matrix[currPath[level - 1]][currPath[0]] != 0 {
                // currRes - зберігає загальну вагу отриманого
розв'язку
                let currRes = currWeight + matrix[currPath[level
- 1]][currPath[0]]

                // Якщо даний результат краще попереднього,
                // оновлюємо finalRes та currRes
                if currRes < finalRes {
                    copyToFinal(currentPath: currPath)
                    finalRes = currRes
                }
            }
            return
        }

        // Для усіх інших рівнів ітерації
        // будуємо дерево просторового пошуку
        for  i in 0...N-1 {

            //  Розглянемо наступну вершину, якщо вона
відрізняється від даної
            //  (до діагонального елементу матриці, ми ще не
потрапили)
            if matrix[currPath[level - 1]][i] != 0 && visited[i]
== false {
```

```
                let temp = currBound
                currWeight += matrix[currPath[level - 1]][i]

                // Розрахунки для currBound для другого рівня відрізняється від інших
                if level == 1 {
                    currBound -= (firstMin(matrix: matrix, i: currPath[level - 1]) + firstMin(matrix: matrix, i: i) / 2)
                } else {
                    currBound -= (secondMin(matrix: matrix, i: currPath[level - 1]) + firstMin(matrix: matrix, i: i) / 2)
                }

                // currBound + currWeight - фактична нижня межа вузла, у який ми
                // потрапили
                // Якщо currBound + currWeight < finalRes, продовжуємо
                // досліджувати вузол
              if (currBound + currWeight) < finalRes {
                    currPath[level] = i
                    visited[i] = true

                    // викликаємо фунцію tspRec, яку ми задали раніше, для переходу
                   // на наступний рівень
                    tspRec(matrix: matrix, currentBound: currBound, currentWeight: currWeight, level: level + 1, currentPath: currPath)
                } else {

                   // у іншому випадку нам необхідно, обрізати вузол
                  // та онулити зміни currWeight та currBound
                currWeight -= matrix[currPath[level - 1]][i]
                currBound = temp
```

```
                }

                // І також повертаємо, масив visited у дефолтне
положення
                for i in 0...N-1 {
                    visited[i] = false
                }

                for j in 0...level-1 {
                    visited[currPath[j]] = true
                }
            }
        }
    }

    // TSP – встановлює finalPath
    func TSP(matrix: [[Int]]) {

        var currentPath: [Int] = Array(repeating: 0, count:
self.N + 1)

        // Розраховуємо початкову нижню границю для кореневого
вузла,
        // для усіх ребер використовуємо формулу (firstMin +
secondMin) / 2
        // Ініціалізуємо currentBound та visited
        var currentBound = 0

        for i in 0...N {
            currentPath[i] = -1
            if i < N {
                visited[i] = false
            }
        }

        // Обчислення початкової границі
```

```
        for i in 0...N-1 {
            currentBound += (firstMin(matrix: matrix, i: i) +
secondMin(matrix: matrix, i: i))
        }

        // У цьому фрагменті коду, ми округлюємо нижню межу до
цілого числа
        if currentBound ==  1 {
            currentBound = currentBound/2 + 1
        } else {
            currentBound = currentBound/2
        }

        // Так як ми починаємо з вузла 1, то перша вершина у
currentPath буде
       // дорівнювати 0
        visited[0] = true
        currentPath[0] = 0

        // Виклик tspRec для currentWeight, що дорівнює 0 та має
перший рівень
        tspRec(matrix:  matrix,  currentBound:  currentBound,
currentWeight: 0, level: 1, currentPath: currentPath)
    }
}
```

ДОДАТОК Б

ModelKnapsack.swift

```swift
//
//  ModelKnapsack.swift

import Foundation

public struct KnapsackItem: Hashable {
  public var name: String
  public var weight: Int
  public var value: Int

  public init(name: String, weight: Int, value: Int) {
    self.name = name
    self.weight = weight
    self.value = value
  }
}

typealias GroupedItem = (name: String, weight: Int, val: Int, n: Int)
```

Service.swift

```swift
//
//  Service.swift


import Foundation

class Service {

    public func knapsack(items: [KnapsackItem], limit: Int) -> [KnapsackItem] {

      var table = Array(repeating: Array(repeating: 0, count: limit + 1), count: items.count + 1)

      for j in 1...items.count {
        let item = items[j-1]

        for w in 1...limit {
          if item.weight > w {
            table[j][w] = table[j-1][w]

          } else {
            table[j][w] = max(table[j-1][w], table[j-1][w-item.weight] + item.value)
          }
        }
      }

      var result = [KnapsackItem]()
      var w = limit

      for j in stride(from: items.count, to: 0, by: -1) where table[j][w] != table[j-1][w] {
        let item = items[j-1]

        result.append(item)

        w -= item.weight
      }

      return result
    }
}
```

RezultViewController.swift

```swift
//
//  RezultViewController.swift

import UIKit

class RezultViewController: UIViewController {

    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var rezultTableView: UITableView!
    @IBOutlet weak var rezultTextLabel: UILabel!

    var baggedCount = 0
    var totalVal = 0
    var totalWeight = 0

    var bagged = [KnapsackItem]()

    override func viewDidLoad() {
        super.viewDidLoad()
        setValuesForUI()
        rezultTableView.dataSource = self
        rezultTableView.delegate = self
    }

    private func setValuesForUI() {
        titleLabel.text = "Bagged the following \(baggedCount) items:"
        rezultTextLabel.text = "For a total value of \(totalVal) and weight of \(totalWeight)"

    }


}
```

```swift
extension RezultViewController: UITableViewDelegate, UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return baggedCount
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = rezultTableView.dequeueReusableCell(withIdentifier: "RzultsTableViewCell", for: indexPath) as!
            RzultsTableViewCell
        let item = bagged[indexPath.row]
        cell.nameLabel.text = item.name
        let countItem = indexPath.row + 1
        cell.countLabell.text = "\(countItem)"
        return cell
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 55
    }


}
```

RzultsTableViewCell.swift

```swift
//
//  RzultsTableViewCell.swift


import UIKit

class RzultsTableViewCell: UITableViewCell {

    @IBOutlet weak var countLabell: UILabel!
    @IBOutlet weak var nameLabel: UILabel!



    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state
    }

}
```

TableVCCell.swift

```swift
//
//  TableVCCell.swift


import UIKit

class TableVCCell: UICollectionViewCell {

    @IBOutlet weak var nameLabel: UILabel!
    @IBOutlet weak var weightLabel: UILabel!
    @IBOutlet weak var valueLabel: UILabel!
    @IBOutlet weak var piecesLabel: UILabel!

    func setCellWithValuesOf(_ item: GroupedItem) {
        updateUI(name: item.name, weight: item.weight, val: item.val, n: item.n)
    }

    // Update the UI Views
    private func updateUI(name: String, weight: Int, val: Int, n: Int) {
        nameLabel.text = name
        weightLabel.text = String(weight)
        valueLabel.text = String(val)
        piecesLabel.text = String(n)

    }

}
```

KnapsackModelView.swift

```swift
//
//  KnapsackModelView.swift

import Foundation

class KnapsackModelView {

    private var service = Service()

     var groupedItems: [GroupedItem] = [
      ("map", 90, 150, 1),
      ("compass", 130, 35, 1),
      ("water", 1530, 200, 3),
      ("sandwich", 500, 60, 2),
      ("glucose", 150, 60, 2),
      ("tin", 680, 45, 3),
      ("banana", 270, 60, 3),
      ("apple", 390, 40, 3),
      ("cheese", 230, 30, 1),
      ("beer", 520, 10, 3),
      ("suntan cream", 110, 70, 1),
      ("camera", 320, 30, 1),
      ("t-shirt", 240, 15, 2),
      ("trousers", 480, 10, 2),
      ("umbrella", 730, 40, 1),
      ("waterproof trousers", 420, 70, 1),
      ("waterproof overclothes", 430, 75, 1),
      ("note-case", 220, 80, 1),
      ("sunglasses", 70, 20, 1),
      ("towel", 180, 12, 2),
      ("socks", 40, 50, 1),
      ("book", 300, 10, 2)
    ]

    func numberOfRowsInSection (section: Int) -> Int {
        return groupedItems.count
    }

    func cellForRowAt(indexPath: IndexPath) -> GroupedItem {
        return groupedItems[indexPath.row]
    }

}
```

TableViewController.swift

```swift
//
//  TableViewController.swift


import UIKit

class TableViewController: UIViewController {

    private var modelView = KnapsackModelView()
    private var service = Service()

    var lists: [String] = ["The first item"]

    @IBOutlet weak var limitTextField: UITextField!
    @IBOutlet weak var tableCollectionView: UICollectionView!



    override func viewDidLoad() {
        super.viewDidLoad()
        tableCollectionView.dataSource = self
        tableCollectionView.delegate = self


    }
```

```swift

    @IBAction func showRezultsTapped(_ sender: Any) {

        let items = modelView.groupedItems.flatMap({item in
            (0..<item.n).map({_ in KnapsackItem(name: item.name, weight: item.weight, value: item.val) })
          })
        guard let limitText = limitTextField.text else {return}
        let limit = Int(limitText) ?? 0
        let bagged = service.knapsack(items: items, limit: limit)
        let (totalVal, totalWeight) = bagged.reduce((0, 0), {cur, item in (cur.0 + item.value, cur.1 + item.weight) })

        let mainStorybord = UIStoryboard(name: "Main", bundle: Bundle.main)
        guard let destinationViewController = mainStorybord.instantiateViewController(withIdentifier: "RezultViewController")
            as? RezultViewController else {
            print("Could not find the view controller")
            return
        }
        print("REZULT")
        for item in bagged {
          print("\t\(item.name)")
        }

        destinationViewController.baggedCount = bagged.count
        destinationViewController.totalVal = totalVal
        destinationViewController.totalWeight = totalWeight
        destinationViewController.bagged = bagged
        navigationController?.pushViewController(destinationViewController, animated: true)

    }
```

```
    @IBAction func plusButtonTapped(_ sender: Any) {
        let alert = UIAlertController(title: "Add Content", message: "", preferredStyle: .alert)
               alert.addTextField { (UITextField) in
                UITextField.placeholder = "Enter name of item"
               }
               alert.addTextField { (UITextField) in
                 UITextField.placeholder = "Enter weight of item"
                }
               alert.addTextField { (UITextField) in
                 UITextField.placeholder = "Enter value of item"
                }
               alert.addTextField { (UITextField) in
                  UITextField.placeholder = "Enter count of item"
                 }


               alert.addAction(UIAlertAction(title: "Add", style: .default, handler: { (UIAlertAction) in
                   let name = alert.textFields![0] as UITextField
                   let nameString = name.text!

                   let weight = alert.textFields![1] as UITextField
                   guard let weightInt = Int(weight.text!) else {return}

                   let value = alert.textFields![2] as UITextField
                   guard let valueInt = Int(value.text!)  else {return}


                   let count = alert.textFields![3] as UITextField
                   guard let countInt = Int(count.text!) else {return}


                   let newItem: GroupedItem = (name: nameString, weight: weightInt, val: valueInt, n: countInt)
                   self.modelView.groupedItems.append(newItem)

                   self.tableCollectionView.reloadData()
               }))
               alert.addAction(UIAlertAction(title: "Cancel", style: .destructive, handler: nil))
               self.present(alert, animated: true, completion: nil)
    }

}
```

```
extension TableViewController: UICollectionViewDelegate, UICollectionViewDataSource, UICollectionViewDelegateFlowLayout {
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return modelView.numberOfRowsInSection(section: section)
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        let cell = tableCollectionView.dequeueReusableCell(withReuseIdentifier: "TableVCCell", for: indexPath) as!
            TableVCCell
        let item = modelView.cellForRowAt(indexPath: indexPath)
        cell.setCellWithValuesOf(item)
        cell.desingMyCell()
        return cell
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        sizeForItemAt indexPath: IndexPath) -> CGSize {
        return CGSize(width: 414, height: 55)
    }

}

extension UIView {

    func desingMyCell() {

        self.layer.cornerRadius = 5
        self.layer.borderWidth = 0.5
        self.layer.borderColor = UIColor.darkGray.cgColor
        self.layer.masksToBounds = true
    }
}
```

ДОДАТОК В

```swift
//  ViewController.swift
//  TSP_B&B

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var nodesView: UIView!

    @IBOutlet weak var timePerformLabel: UILabel!

    @IBOutlet weak var timeAlgoLabel: UILabel!

    @IBOutlet weak var finalPathLabel: UILabel!


    @IBOutlet weak var setXTextField: UITextField!

    @IBOutlet weak var setYTextField: UITextField!


    var vertexes: [Vertex] = []
    var route: [Int] = []

    var locations: [CGPoint] = []
    var arrayOfDistance: [[Double]] = [[]]

    let mainStorybord = UIStoryboard(name: "Main", bundle: Bundle.main)

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    override func viewWillAppear(_ animated: Bool) {
        self.navigationController?.navigationBar.isHidden = true

    }

```

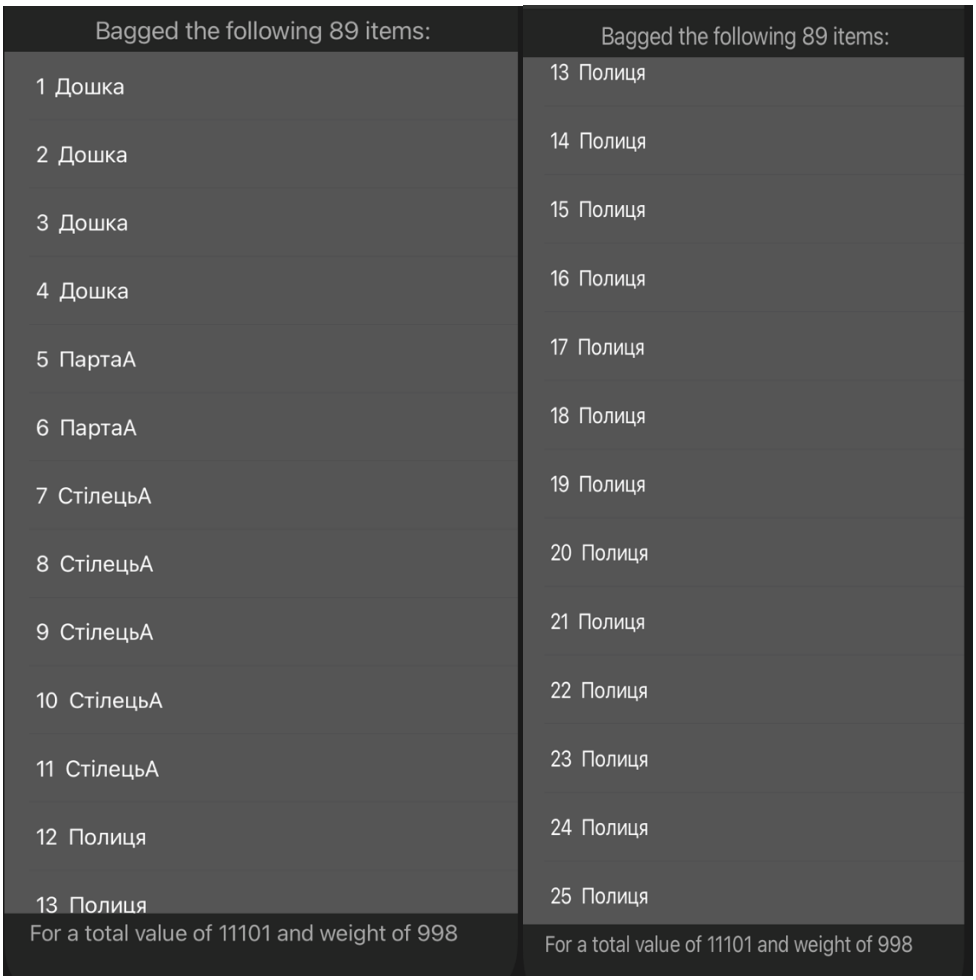
```swift
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        guard let touch = touches.first else {return}

        if nodesView.point(inside: touch.location(in: nodesView), with: event){
            var location = touch.location(in: nodesView)
            let intForX = modf(location.x)
            let intForY = modf(location.y)
            location.x = intForX.0
            location.y = intForY.0
            locations.append(location)
            let vertex = Vertex(location: location)
            vertexes.append(vertex)
            drawNodes()
        }
    }

    private func drawNodes() {
        self.nodesView.layer.sublayers?.removeAll()

        self.locations.forEach { (location) in
            let circle = UIBezierPath.init(arcCenter: location, radius: 5, startAngle: 0, endAngle: CGFloat.pi * 2,
                clockwise: true)
            let circleLayer = CAShapeLayer()
            circleLayer.path = circle.cgPath
            circleLayer.fillColor = UIColor.black.cgColor
            circleLayer.strokeColor = UIColor.black.cgColor
            self.nodesView.layer.addSublayer(circleLayer)
        }
    }

    private func createMatrix() {
        let count = locations.count
        var array = Array(repeating: Array(repeating: Double(0), count: count), count: count)
        for i in 0...count-1 {
            for j in 0...count-1 {
                let from = locations[i]
                let to = locations[j]
                let dist = culDistanc(location1: from, location2: to)
                array[i][j] = dist
            }
        }
        arrayOfDistance = array
    }
```

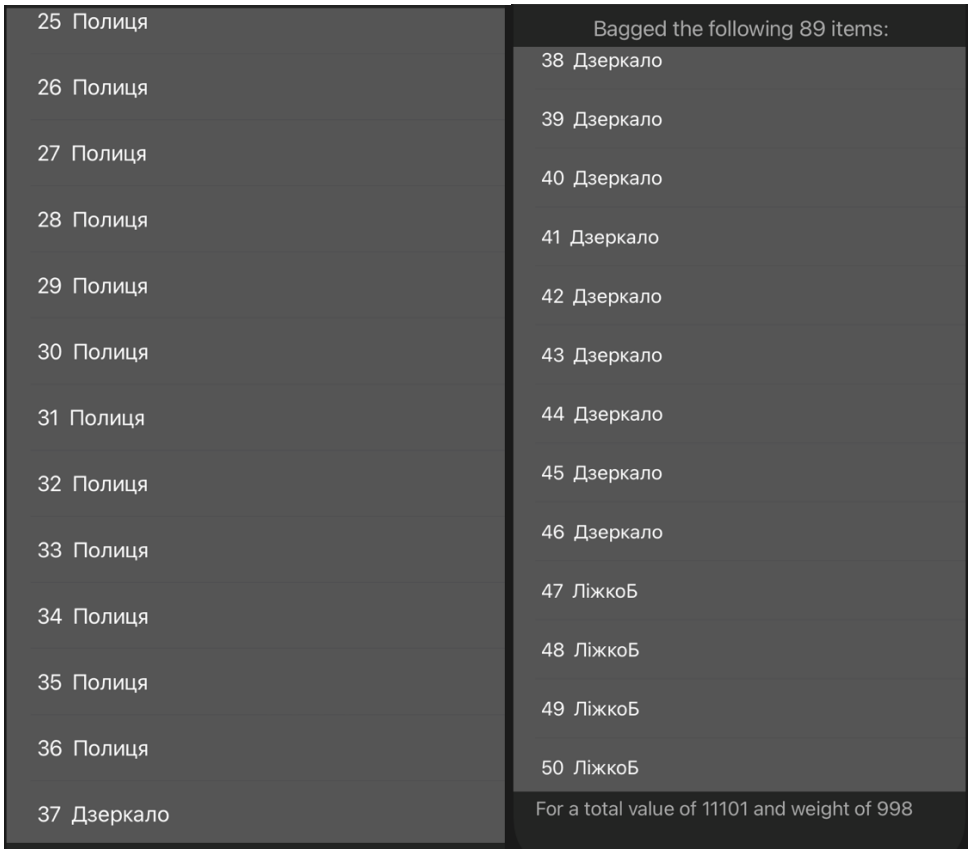
```swift
    private func culDistanc(location1: CGPoint, location2: CGPoint) -> Double {
        let dist = sqrt(pow(location1.x - location2.x, 2) + pow(location1.y - location2.y, 2))
        let intDist = Double(dist)
        return intDist
    }


    @IBAction func clearTapped(_ sender: Any) {
        locations.removeAll()
        vertexes.removeAll()
        arrayOfDistance.removeAll()
        route.removeAll()
        nodesView.layer.sublayers?.removeAll()
        timePerformLabel.text = "Execution time: 0"
        timeAlgoLabel.text = "Execution time of B&B: 0"
        finalPathLabel.text = "Final Path: 0"
        setXTextField.text = ""
        setYTextField.text = ""
    }

    @IBAction func startTapped(_ sender: Any) {

        let start = CFAbsoluteTimeGetCurrent()
        findRoute()
        let diff = CFAbsoluteTimeGetCurrent() - start
        finalPathLabel.text = "Final Path: \(route)"
        timePerformLabel.text = "Execution time: \(diff) s"
        print("\(route)")
        drawRoute(rout: route)


    }

    func newView() {
        let start = CFAbsoluteTimeGetCurrent()
        findRoute()
        let diff = CFAbsoluteTimeGetCurrent() - start
        finalPathLabel.text = "Final Path: \(route)"
        timePerformLabel.text = "Execution time: \(diff) s"
        print("\(route)")
        drawRoute(rout: route)
    }
```

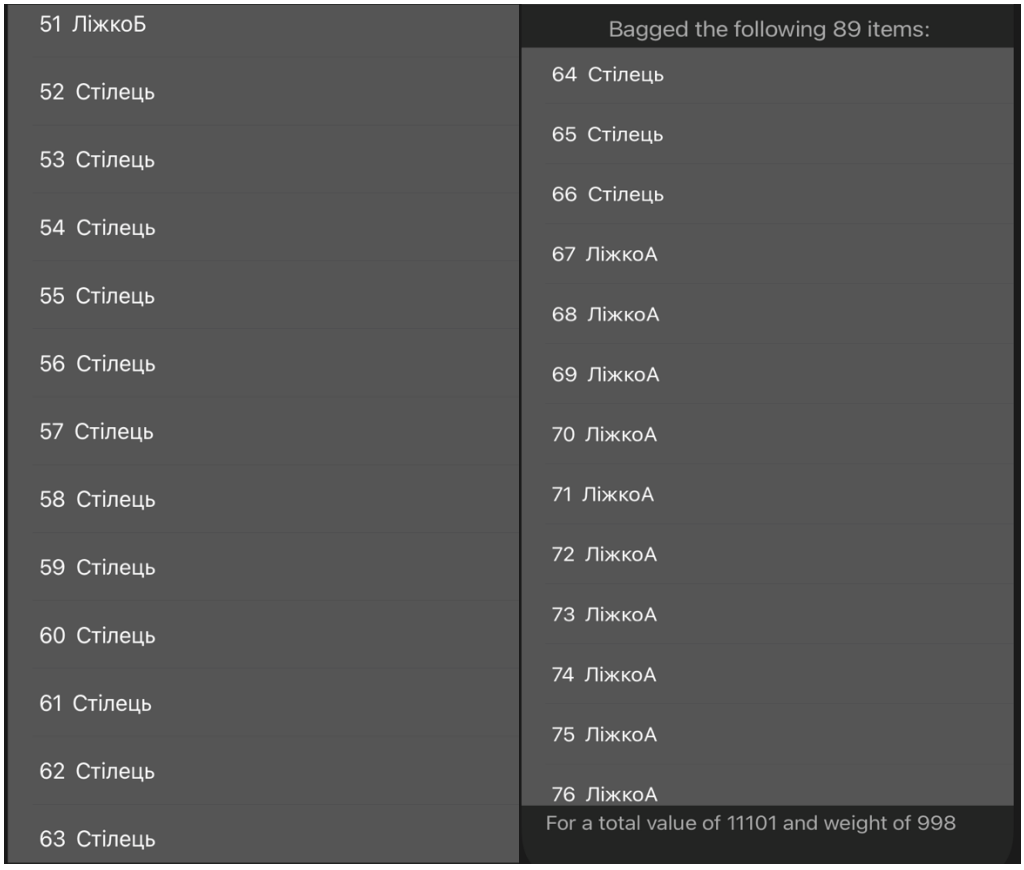
```swift
    @IBAction func seeAllTapped(_ sender: Any) {

        guard let destinationViewController = mainStorybord.instantiateViewController(withIdentifier:
            "ShowAllNodesViewController") as? ShowAllNodesViewController else {
                print("Could not find the view controller")
                return
        }
        destinationViewController.locations = locations
        navigationController?.pushViewController(destinationViewController, animated: true)
    }

    private func findRoute() {
        createMatrix()
        let tspAlgo = GFG()
        tspAlgo.N = vertexes.count
        let start = CFAbsoluteTimeGetCurrent()
        tspAlgo.TSP(adj: arrayOfDistance) { [self] in
            route = tspAlgo.finalPath
            print("Minimum cost : \(tspAlgo.finalRes)")
            print("Path Taken :")
            print("\(route)")
            for i in 0...tspAlgo.N {
                print("\(route[i])")
            }
        }
        let diff = CFAbsoluteTimeGetCurrent() - start
        timeAlgoLabel.text = "Execution time of B&B: \(diff) s"
    }

    private func drawRoute(rout: [Int]) {
        guard let firstVertexPoint = rout.first else {return}
        var otherVertex = rout
        print("\(rout)")
        otherVertex.remove(at: 0)


        let firstNode = locations[firstVertexPoint]

        drawNodes()
```

```
169         DispatchQueue.main.async { [self] in
170             let path = UIBezierPath()
171             UIColor.black.setStroke()
172             path.lineWidth = 1
173 
174             path.move(to: firstNode)
175             otherVertex.forEach { (vertexNumber) in
176                 let nextNode = locations[vertexNumber]
177                 print("NODE: \(nextNode)")
178                 path.addLine(to: nextNode)
179 
180             }
181             path.addLine(to: firstNode)
182             path.stroke()
183             let pathLayer = CAShapeLayer()
184             pathLayer.path = path.cgPath
185             pathLayer.fillColor = UIColor.clear.cgColor
186             pathLayer.strokeColor = UIColor.black.cgColor
187             self.nodesView.layer.addSublayer(pathLayer)
188         }
189 
190 
191     }
192 
193 
194     @IBAction func applyTapped(_ sender: Any) {
195 
196         guard let xConstrString = setXTextField.text else {
197             return
198         }
199         guard let yConstrString = setYTextField.text else {
200             return
201         }
202         guard let xConst = Int(xConstrString) else {
203             return
204 
205         }
206         guard let yConst = Int(yConstrString) else {
207             return
208         }
209         let location: CGPoint = CGPoint(x: xConst, y: yConst)
210         let node = Vertex(location: location)
211         locations.append(location)
212         vertexes.append(node)
213         drawNodes()
214 
215 
216     }
```

ДОДАТОК Г


Bagged the following 89 items:
1 Дошка
2 Дошка
3 Дошка
4 Дошка
5 ПартаА
6 ПартаА
7 СтілецьА
8 СтілецьА
9 СтілецьА
10 СтілецьА
11 СтілецьА
12 Полиця
13 Полиця
For a total value of 11101 and weight of 998
Bagged the following 89 items:
13 Полиця
14 Полиця
15 Полиця
16 Полиця
17 Полиця
18 Полиця
19 Полиця
20 Полиця
21 Полиця
22 Полиця
23 Полиця
24 Полиця
25 Полиця
For a total value of 11101 and weight of 998


25 Полиця
26 Полиця
27 Полиця
28 Полиця
29 Полиця
30 Полиця
31 Полиця
32 Полиця
33 Полиця
34 Полиця
35 Полиця
36 Полиця
37 Дзеркало
Bagged the following 89 items:
38 Дзеркало
39 Дзеркало
40 Дзеркало
41 Дзеркало
42 Дзеркало
43 Дзеркало
44 Дзеркало
45 Дзеркало
46 Дзеркало
47 ЛіжкоБ
48 ЛіжкоБ
49 ЛіжкоБ
50 ЛіжкоБ
For a total value of 11101 and weight of 998


51 ЛіжкоБ
52 Стілець
53 Стілець
54 Стілець
55 Стілець
56 Стілець
57 Стілець
58 Стілець
59 Стілець
60 Стілець
61 Стілець
62 Стілець
63 Стілець
Bagged the following 89 items:
64 Стілець
65 Стілець
66 Стілець
67 ЛіжкоА
68 ЛіжкоА
69 ЛіжкоА
70 ЛіжкоА
71 ЛіжкоА
72 ЛіжкоА
73 ЛіжкоА
74 ЛіжкоА
75 ЛіжкоА
76 ЛіжкоА
For a total value of 11101 and weight of 998


Bagged the following 89 items:
77 ЛіжкоА
78 ЛіжкоА
79 ЛіжкоА
80 ЛіжкоА
81 ЛіжкоА
82 Стіл
83 Стіл
84 Стіл
85 Стіл
86 Стіл
87 Стіл
88 Стіл
89 Стіл
For a total value of 11101 and weight of 998