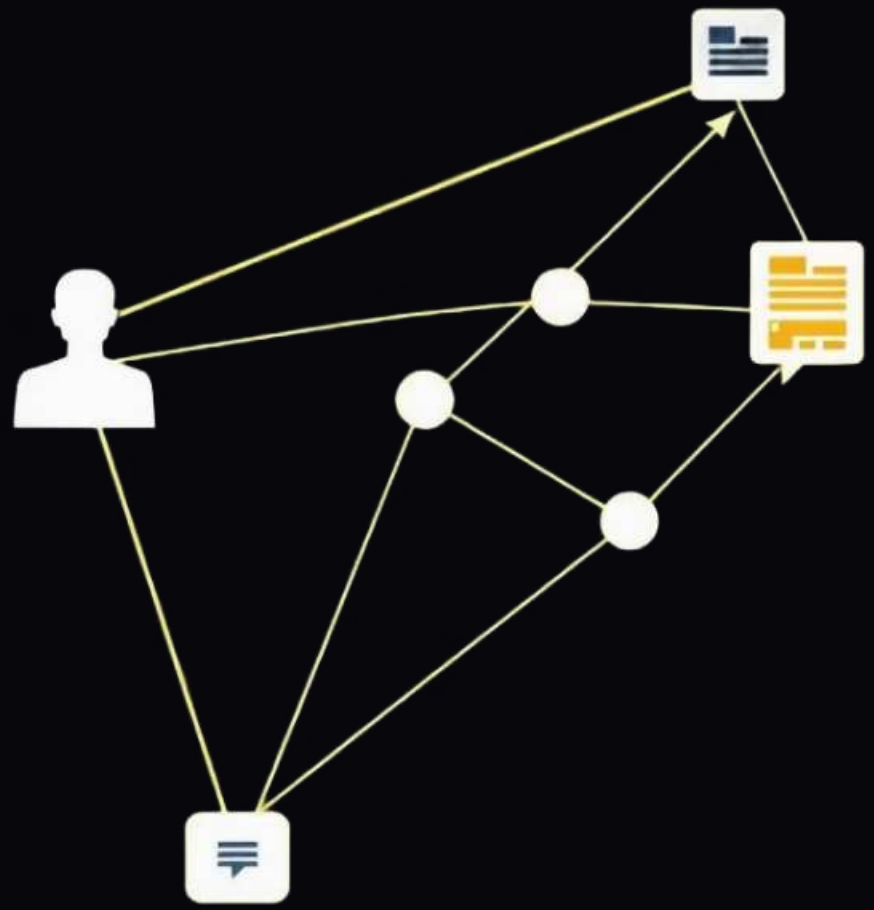


Розв'язання задачі про призначення методом грубої сили

У цій презентації ми розглянемо задачу про призначення. Ми вивчимо, як її розв'язати методом грубої сили. З'ясуємо практичність такого підходу. Також дослідимо обмеження цього методу.

Підготував: студент групи КМ-43 Пилипчук Євгеній

Що таке метод грубої сили взагалі?

Метод грубої сили (або **вичерпний перебір**) — це простий, але неефективний алгоритмічний підхід, який розв`язує задачу, **перебираючи всі можливі варіанти** і вибираючи найкращий.

Основна ідея:

1. Згенерувати всі можливі рішення.
2. **Перевірити кожне з них**.
3. **Вибрати найкраще** за заданим критерієм (наприклад, мінімальна вартість або максимальна вигода).

Недоліки:

- **Дуже повільний** при великих обсягах даних (зазвичай має **експоненційну складність** O(n!)).
- **Неефективний**, якщо є кращі алгоритми

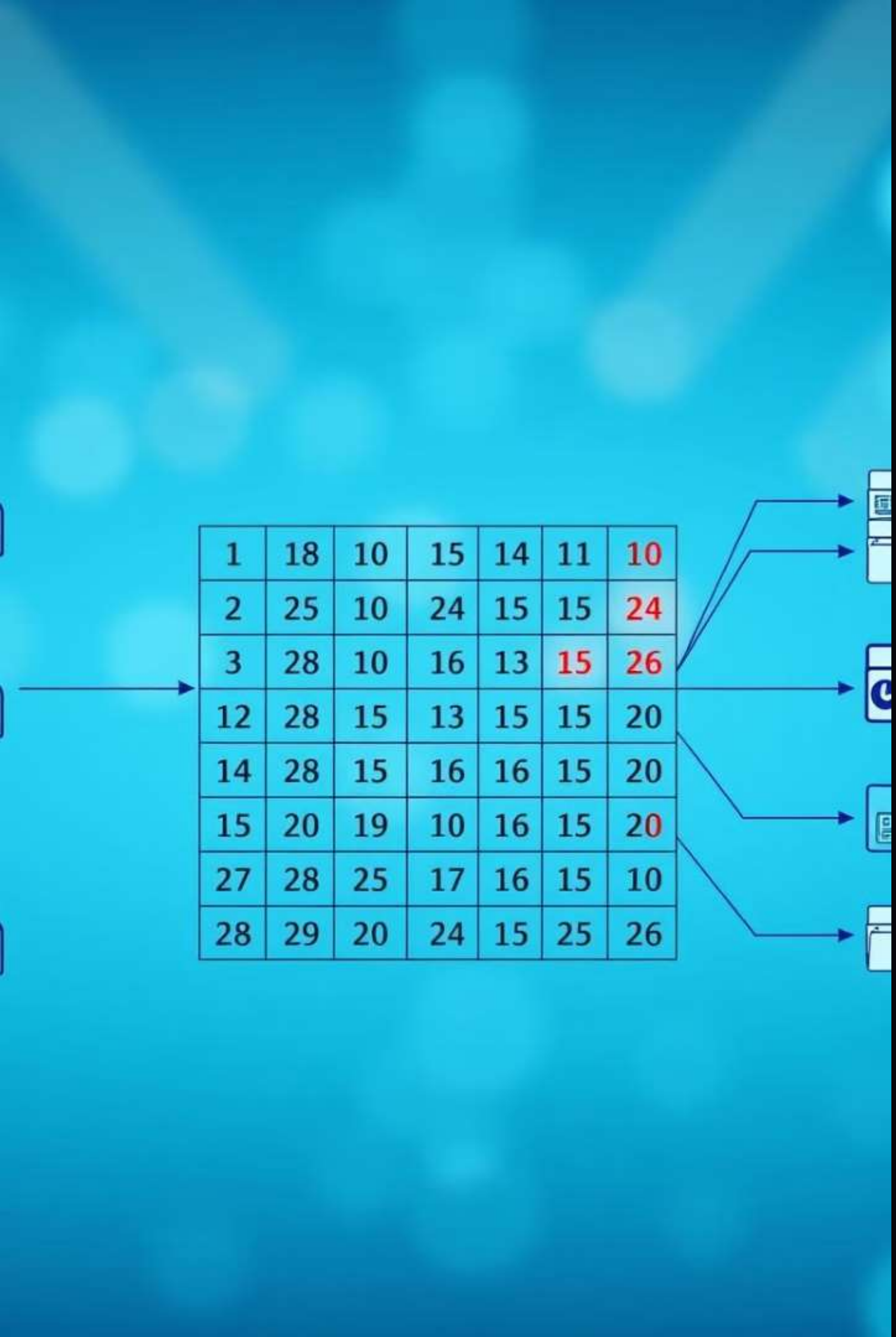
Вступ до задачі про призначення

Що це?

Задача про призначення - це оптимізаційна задача. Вона полягає у розподілі ресурсів. Мета - мінімізувати загальну вартість.

Важливість

Ця задача важлива для логістики. Також вона корисна в управлінні ресурсами. Вона є ключовою в багатьох сферах.



Постановка задачі

1. Є **n** працівників та **n** завдань. Кожен працівник може виконати рівно одне завдання, і кожне завдання має бути призначене одному працівнику. Вартість виконання i-им працівником j-го завдання задається **матрицею вартості** C[i, j],де: C[i, j] – вартість виконання завдання **j** працівником **i**. Потрібно знайти найдешевший варіант розподілу завдань.

	Завдання 1	Завдання 2	Завдання 3
Працівник 1	9	2	7
Працівник 2	6	4	3
Працівник 3	5	8	1

Формалізація задачі

Рішення задачі можна представити як **перестановку чисел** від **1 до n**, де кожен елемент перестановки вказує на завдання, яке отримує певний працівник.
Наприклад, перестановка (2,3,1) означає, що:

- Працівник 1 отримує завдання 2
- Працівник 2 отримує завдання 1
- Працівник 3 отримує завдання 3

	Завдання 1	Завдання 2	Завдання 3
Працівник 1	9	2	4
Працівник 2	6	4	3
Працівник 3	5	8	1

Мета розв'язку

Необхідно знайти таку перестановку, яка **мінімізує загальну вартість** виконання завдань:

$$\min \sum_{i=1}^{n} C[i, \mathrm{perm}[i]]$$

- C[i,j] – це вартість виконання j-го завдання i-м працівником.
- perm[i] – це завдання, яке призначене i-му працівнику (воно змінюється при перестановках).
- Фактично ми беремо по **одному елементу** з **кожного рядка матриці**, але так, щоб вони всі були з **різних стовпців** (тобто кожне завдання отримує рівно один працівник).
- **min** – ми шукаємо таку перестановку perm, яка дає мінімальне значення цієї суми.

Приклад

	Завдання 1	Завдання 2	Завдання 3
Працівник 1	9	2	4
Працівник 2	6	4	3
Працівник 3	5	8	1

Можливі перестановки (відповідають варіантам розподілу завдань між працівниками):

$$(1,2,3) \to C[1,1] + C[2,2] + C[3,3] = 9 + 4 + 1 = 14$$
$$(1,3,2) \to C[1,1] + C[2,3] + C[3,2] = 9 + 3 + 8 = 20$$
$$(2,1,3) \to C[1,2] + C[2,1] + C[3,3] = 2 + 6 + 1 = 9 \quad \checkmark$$

Було знайдено мінімальну суму

Представлення допустимого розв’язку

1. Генерація всіх можливих розподілів (перестановок)
2. Обчислення вартості кожного розподілу
3. Збереження найкращого розподілу

1) Генерація всіх можливих перестановок

- Використовуємо **рекурсивну функцію** `permute()`, щоб створити **всі можливі перестановки**.
- Для кожної перестановки:

Міняємо місцями два елементи (swap).

- Рекурсивно викликаємо `permute()` для подальших елементів.
- Повертаємо перестановку назад (backtracking).

```
void permute(int perm[MAX_N], int l, int r) {
    if (l == r) { // Якщо досягнуто кінця перестановки
        int cost = calculate_cost(perm); // Обчислюємо вартість перестановки
        static int first = 1; // Флаг для першої перестановки
        static int best_cost; // Найкраща вартість
        if (first || cost < best_cost) { // Якщо це перша або краща вартість
            first = 0;
            best_cost = cost; // Оновлюємо найкращу вартість
            for (int i = 0; i < n; i++) { // Зберігаємо найкращу перестановку
                best_perm[i] = perm[i];
            }
        }
    }
    else {
        for (int i = l; i <= r; i++) { // Генеруємо всі перестановки рекурсивно
            swap(&perm[l], &perm[i]);
            permute(perm, l + 1, r);
            swap(&perm[l], &perm[i]); // Повертаємо назад (backtrack)
        }
    }
}
```

2)Обчислення вартості кожного розподілу

- Для кожної знайденої перестановки обчислюємо **загальну вартість**:
  - Перебираємо кожного працівника `i`.
  - Додаємо вартість виконання завдання `perm[i]` для працівника `i` з матриці

```
int calculate_cost(int perm[MAX_N]) {
    int cost = 0;
    for (int i = 0; i < n; i++) { // Прохід по всіх працівниках
        cost += matrix[i][perm[i]]; // Додаємо вартість відповідного завдання
    }
    return cost; // Повертаємо загальну вартість
}
```

3)Збереження найкращого розподілу

Якщо поточний розподіл має **меншу вартість,** ніж найкращий знайдений раніше:

- Запам'ятовуємо цю перестановку у `best_perm[]`.
- Оновлюємо **найменшу вартість** `best_cost`.

```
void permute(int perm[MAX_N], int l, int r) {
    if (l == r) { // Якщо досягнуто кінця перестановки
        int cost = calculate_cost(perm); // Обчислюємо вартість перестановки
        static int first = 1; // Флаг для першої перестановки
        static int best_cost; // Найкраща вартість
        if (first || cost < best_cost) { // Якщо це перша або краща вартість
            first = 0;
            best_cost = cost; // Оновлюємо найкращу вартість
            for (int i = 0; i < n; i++) { // Зберігаємо найкращу перестановку
                best_perm[i] = perm[i];
            }
        }
    }
    else {
        for (int i = l; i <= r; i++) { // Генеруємо всі перестановки рекурсивно
            swap(&perm[l], &perm[i]);
            permute(perm, l + 1, r);
            swap(&perm[l], &perm[i]); // Повертаємо назад (backtrack)
        }
    }
}
```

Аналіз складності

1. **Як працює алгоритм**?

Метод **грубої сили** перебирає **всі можливі розподіли завдань між працівниками**, обчислює їхню вартість і вибирає найкращий.
Оскільки кожен працівник може отримати **одне з n завдань**, а жодне завдання не може повторюватися, ми фактично **перебираємо всі перестановки** множини {1, 2, ..., n}.

2. **Часова складність**

Кількість можливих перестановок:

$$n! = n \times (n - 1) \times (n - 2) \times \ldots \times 1$$

Таким чином, складність алгоритму:

$$O(n!)$$

Альтернативні методи

Угорський алгоритм

Швидший поліноміальний метод. Складність $O(n^3)$.

Суть алгоритму

Основні кроки:

1. **Знаходимо мінімальні елементи** в кожному рядку та віднімаємо їх від всіх елементів цього рядка.
2. **Знаходимо мінімальні елементи** в кожному стовпці та віднімаємо їх від всіх елементів цього стовпця.
3. **Закреслюємо всі нулі мінімальною кількістю ліній**.
   - Якщо потрібно **n** ліній → оптимальне рішення знайдено.
   - Інакше коригуємо матрицю і повторюємо кроки.
4. **Формуємо оптимальне призначення**, використовуючи знайдені нулі.

Використання угорського методу для даної задачі

	Завдання 1	Завдання 2	Завдання 3
Працівник 1	9	2	4
Працівник 2	6	4	3
Працівник 3	5	8	1

Крок 1: Віднімання мінімальних значень у рядках

Для кожного рядка шукаємо **мінімальне значення** та віднімаємо його від кожного елемента рядка:

$$C' = \begin{bmatrix} 7 & 0 & 5 \\ 3 & 1 & 0 \\ 4 & 7 & 0 \end{bmatrix}$$

Крок 2: Віднімання мінімальних значень у стовпцях

Тепер у кожному **стовпці** шукаємо **мінімальне значення** та віднімаємо його:

- Мінімум у 1-му стовпці: 3
- Мінімум у 2-му стовпці: 0
- Мінімум у 3-му стовпці: 0

$$C'' = \begin{bmatrix} 4 & 0 & 5 \\ 0 & 1 & 0 \\ 1 & 7 & 0 \end{bmatrix}$$

Використання угорського методу для даної задачі

Крок 3: Покриття всіх нулів мінімальною кількістю ліній

- У матриці є **три нулі**, які можна закреслити **трьома лініями**.
- Це означає, що **оптимальне призначення знайдено**.

$$C'' = \begin{bmatrix} 4 & 0 & 5 \\ 0 & 1 & 0 \\ 1 & 7 & 0 \end{bmatrix}$$

Крок 4: Вибір оптимального розподілу

Ми можемо призначити нулі так, щоб кожен працівник мав **рівно одне завдання**:

- Працівник **1** → Завдання **2**
- Працівник **2** → Завдання **3**
- Працівник **3** → Завдання **1**

$$C'' = \begin{bmatrix} 4 & 0 & 5 \\ 0 & 1 & 0 \\ 1 & 7 & 0 \end{bmatrix}$$

Висновки

Малі n

Метод грубої сили працює для малих n. Коли кількість варіантів невелика.

Великі n

Потрібні ефективніші алгоритми. Угорський алгоритм або інші.

Практичність

Важливо вибирати правильний метод. Залежно від розміру задачі.