13. Метод підйому. Евристики

За *методом підйому* будують алгоритм, який приймає деяке початкове припущення або обчислює початковий розв'язок задачі, а потім якомога швидше поліпшує його, намагаючись отримати оптимальний розв'язок задачі. Алгоритми підйому доволі грубі, бо намагання покращити розв'язок у будь-який спосіб робить ці алгоритми дещо недалекоглядними. Такі алгоритми використовують для швидкого обчислення наближеного розв'язку задачі.

Евристикою називають алгоритм з такими властивостями:

- він здебільшого знаходить хороші, проте не завжди оптимальні розв'язки;
- його можна швидше і легше реалізувати, ніж будь-який точний алгоритм.

Безліч евристичних алгоритмів базується або на методі часткових цілей, або на методі підйому. Один загальний підхід до побудови евристичних алгоритмів полягає у переліченні всіх вимог до точного розв'язку і поділі їх на дві групи:

1) ті, які легко задовольнити;
2) усі інші.

Або:

1) ті, які треба задовольнити обов'язково;
2) ті, які можна виконати не повною мірою.

Тоді будують алгоритм, який виконує вимоги з першої групи і намагається виконати вимоги другої групи.

13.1. Задача комівояжера

Продемонструємо процес побудови евристики методом підйому на прикладі *задачі комівояжера*. Це класична задача, умову якої сформульовано так.

> ***Задача 50.*** *Припустимо, що комівояжер мусить відвідати клієнтів, які живуть у n різних містах. Кожне місто він відвідує один раз і повертається додому (такий маршрут називається туром). Яким буде найкоротший тур (найдешевший, найшвидший)?*

Цю задачу легше сформулювати, ніж розв'язати. Вона належить до недетерміновано поліноміальних за складністю (*NP*-складних): для таких проблем невідомі алгоритми, які здатні відшукати оптимальний розв'язок за час, що зростає як поліноміальна функція вхідних даних задачі. Справді, кількість можливих турів у задачі комівояжера зростає як $n!$. Наприклад, для ста міст існує понад 10^{155} шляхів (в історії Всесвіту минуло лише 10^{18} секунд). Перебрати усі тури і знайти найкращий неможливо. У цьому випадку можна формулювати задачу тільки про відшукання деякого наближеного розв'язку задачі комівояжера, наприклад, евристичного розв'язку.

Оптимальний розв'язок задачі комівояжера має дві основні властивості:

1) він складається з ребер, які утворюють простий цикл (тур);
2) вартість будь-якого іншого туру не буде меншою за його вартість.

Евристичний алгоритм задовольняє першу вимогу і не обов'язково виконує другу.

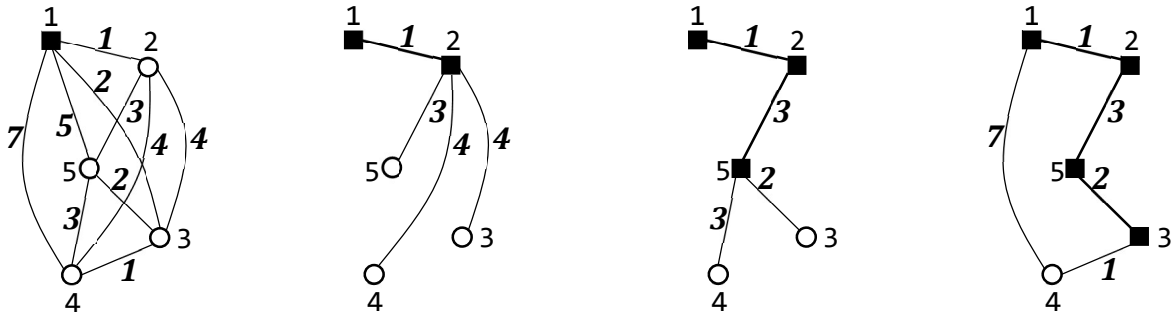
Щоб розпочати проектування алгоритму, необхідно передусім визначити, як задано інформацію щодо відстаней між містами, можливостей переїзду з міста до міста, вартості і тривалості. Пронумеруємо всі міста, які має відвідати комівояжер. Якщо припустити, що він користується власним автомобілем (що доволі ймовірно), то відстань між містами, вартість

переїзду і час будуть практично пропорційними. Їхню числову величину можна задавати, наприклад, за допомогою матриці вартостей *C*. Елемент c_{ij} задає вартість переїзду з *i*-го міста до *j*-го. Очевидно, що $c_{ij} = c_{ji}$. Якщо між окремими містами немає сполучення, то значення відповідних елементів матриці вартостей можна задати максимально великим. Такими ж великими задамо і діагональні елементи, щоб усунути можливість «переїзду» комівояжера з будь-якого міста у те ж саме місто.

Тур комівояжера розпочинається з його рідного міста. Черговий переїзд вибиратимемо з умови мінімальності: наступним містом туру буде те, потрапити в яке з попереднього найдешевше. Щоб не зачисляти до туру місто удруге, запам'ятовуватимемо множину міст, які комівояжер уже відвідав. Щоб не відволікатися на технічні деталі щодо введення матриці вартостей, вважатимемо, що кількість міст фіксовано, і матрицю можна задати безпосередньо в тексті програми. Врахуємо також те, що ми звикли лічити від 1, а елементи масиву індексують від 0. Програма, що реалізує такий алгоритм, матиме вигляд:

```
#include <iostream>
#include <set>
#include <Windows.h>
using word = unsigned short;

int main()
{
	SetConsoleOutputCP(1251);	// налаштування кирилиці
	const int N = 5;  // розглянемо задачу з п'ятьма містами
	// матриця вартостей
	word C[N][N] = { { 65535, 1, 2, 7, 5 },
	                 { 1, 65535, 4, 4, 3 },
	                 { 2, 4, 65535, 1, 2 },
	                 { 7, 4, 1, 65535, 3 },
	                 { 5, 3, 2, 3, 65535 } };
	std::cout << "Введіть номер початкового міста: ";
	int town;
	std::cin >> town;

	int tour[N + 1] = { town }; // тур розпочинається в рідному місті
	int current = town - 1;             // поточне місто
	std::set<int> visited = { current }; // множина відвіданих міст
	word cost = 0;                      // вартість туру

	for (int k = 1; k < N; ++k)
	{
		int index_of_min;                    // шукаємо продовження туру
		word min_value = C[current][current];  // з найменшою вартістю
		for (int i = 0; i < N; ++i)          // серед невідвіданих міст
			if (visited.count(i) == 0 && C[current][i] < min_value)
			{
				min_value = C[current][i];
				index_of_min = i;
			}
		tour[k] = index_of_min + 1;          // переїхали в нове місто
		cost += C[current][index_of_min];

		visited.insert(index_of_min);
		current = index_of_min;
	}
	// тур завершується в рідному місті
```

```
        tour[N] = town;
        cost += C[current][town - 1];

        std::cout << "Прокладено маршрут:\n";
        for (int i = 0; i < N; ++i)
              std::cout << tour[i] << " --> " << tour[i + 1] << '\n';

        std::cout << "Вартість туру: " << cost << '\n';
        return 0;
}
```



а *б* *в* *г*

Рис. 12. Евристичний розв'язок задачі комівояжера з п'ятьма містами. Для кожного етапу розв'язування задачі подано вартість побудованого туру (значення *cost*)

Граф, який відповідає заданій у програмі матриці вартостей, зображено на рис. 12, *а*.

Простежимо, як відбуватиметься побудова туру, якщо комівояжер вирушає з міста, зазначеного за номером 1. У початковий момент відвіданим є тільки перше місто (на схемі позначено квадратиком), вартість туру поки що дорівнює нулю. Серед можливих продовжень туру з вартостями 1, 3, 7, 5 алгоритм вибере найдешевше, і комівояжер вирушить у місто за номером 2 (рис. 12, *б*). Тепер відвідано два міста. Продовження алгоритм шукатиме серед решти. На останньому кроці вибір у алгоритму невеликий (рис. 12, *г*): невідвіданим залишилось тільки четверте місто. Тому саме його наш алгоритм додасть до туру і завершить тур поверненням у перше місто, додавши до туру найдорожчу ланку.

Результати виконання програми:

```
Введіть номер початкового міста: 1
Прокладено маршрут:
1 --> 2
2 --> 5
5 --> 3
3 --> 4
4 --> 1
Вартість туру: 14
```

Бачимо, що остаточна вартість евристики дорівнює 14. Це не найгірший результат. А оптимальним для задачі є тур {1, 2, 5, 4, 3, 1} з вартістю 10.

Недоліком цього алгоритму є те, що наприкінці може залишитись місто, вартість переїзду до якого значна. Такий розв'язок буде неоптимальним.

13.2. Запитання та завдання для самоперевірки

1. Якими властивостями володіє евристичний алгоритм?
2. Для побудови евристики всі вимоги до розв'язку ділять на дві групи. Пригадайте, на які саме.
3. Які вимоги до розв'язку задачі комівояжера є обов'язковими, а якими можна знехтувати?
4. З якою метою в програмі побудови туру комівояжера використано контейнер *set<int>*? Чи можна використати замість нього інший стандартний контейнер?
5. Метод підйому на кожному кроці обирає ребро найменшої вартості. Чому ж він не знаходить оптимальний розв'язок задачі комівояжера? У чому полягає головний недолік цього методу?
6. Завантажте програми за наведеним посиланням, запустіть їх на виконання.
7. Запропонуйте та випробуйте власні зміни та доповнення до програм, наприклад, відшукайте тур комівояжера для більшої кількості міст.
8. Перевірте, чи впливає порядок стовпців (рядків) у матриці вартостей на розв'язок задачі комівояжера, отриманий методом підйому.
9. Напишіть програму відшукання туру комівояжера методом підйому для випадку, коли граф задано таблицею інцидентності.