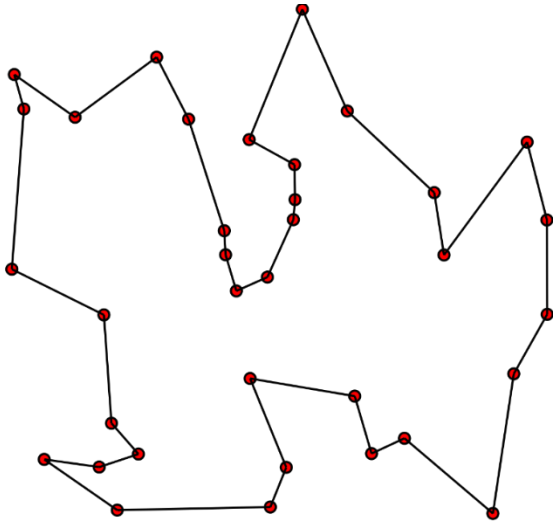
УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ
ФАКУЛТЕТ „ИНФОРМАЦИОННИ НАУКИ“
Курс „Основи на програмирането“

Анализ, решение и имплементация на задача „Бензиностанции“

Автор: Костадин Динков, 1-ви курс, направление ИКН, фак. № 46355з
София, 2020г.



Абстракт

Целта на този документ е да представи практическа имплементация на решение на задачата с „Бензиностанциите“, зададена като освобождаваща от изпит в курса „Основи на Програмирането“ (УНИБИТ, ИКН, първи семестър) и по този начин да запознае читателя с един от най-популярните оптимизационни проблеми в информатиката и компютърните науки. Проблема се анализира и се разглеждат няколко възможни подхода за решаването му, като метод с изчерпателно търсене и приблизителни решения. За самата имплементация на решението се използва алгоритъма на **Белман-Хелд-Карп** за класическия проблем с пътуващият търговец (**Traveling Salesman Problem**). Обясненията са насочени към хора с начален опит в програмирането и алгоритмите, но ще бъдат най-подходящи за тези, които сами са се опитали да се справят със задачата. Решението, поместено в края на документа е имплементирано на C#, използвайки .Net Core технологията.

Съдържание

1. Условие На Задачата

БЕНЗИНОСТАНЦИИ

Фирма има N бензиностанции. Всяка седмица нейна цистерна извършва зареждането им тръгвайки от нейната централна складова база за горива, минавайки през всички бензиностанции и връщайки се обратно в нея. Помогнете на шофьора на цистерната да пресметне дължината на най-краткия път, по който той да извърши зареждането.

Входни данни: На първият ред ще получите броя на тестовете – T. На първият ред от всеки тест ще получите броя на бензиностанциите N (централната база не се включва в N). На следващият ред ще получите най-кратките разстояния $S_{0,j}$ от централната база до всички бензиностанции, подредени по нарастващ ред на номерата на бензиностанциите. На следващите N – 1 реда ще получите най-кратките разстоянията $S_{k,j}$ от поредната бензиностанция K (започвайки от първата във възходящ ред) до всички останали N – K бензиностанции с по-голям номер, подредени по нарастващ ред на номерата на бензиностанциите. Всички пътища са двупосочни. Не е възможно от една бензиностанция да стигнем до друга за по кратък път от този, който е зададен в тестовите данни за съответната двойка бензиностанции.

Ограничения:
$1 \le T \le 10$
$1 \le N \le 17$
$1 \le S_{i,j}$ (разстояние между две бензиностанции – цяло число) ≤ 1500

Изходни данни: За всеки тест изведете на отделен ред дължината на най-краткия път, по който можем да минем през всички бензиностанции, започвайки от централната база и завършвайки пак в нея.

Примерен вход	Примерен изход
1 4 11 13 22 7 15 18 17 11 11 22	58

Коментар на примера: Най-краткият маршрут минава през бензиностанциите 1, 3, 2 и 4 и е с дължина 11 + 18 + 11 + 11 + 7 = 58

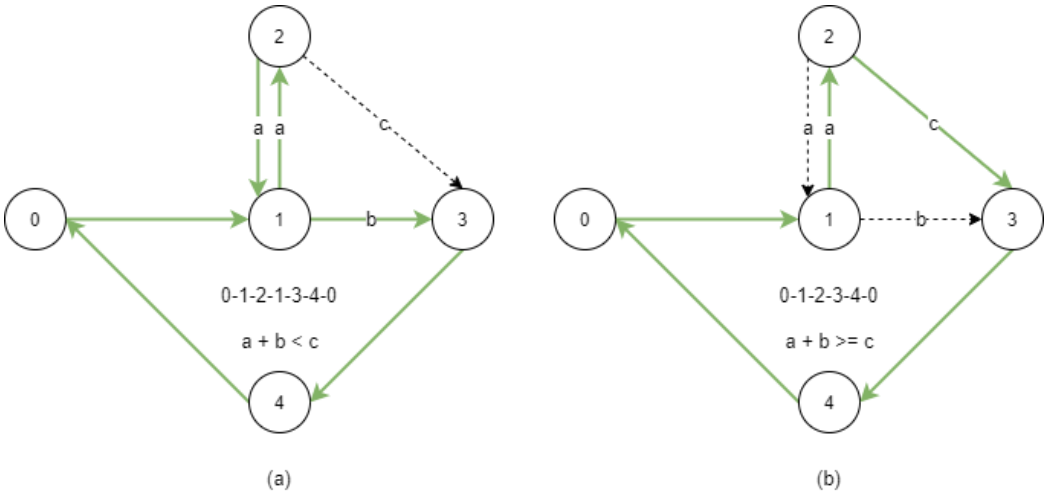
2. Анализ

От условието на задачата е видимо, че тя е много сходна на класическата задача за пътуващия търговец (Traveling Salesman Problem – TSP) [1]. Можем да представим входните данни като пълен, неориентиран граф. Пълен, защото имаме пътища (ребра) от една бензиностанция (връх) до всяка друга бензиностанция и неориентиран защото разстоянието между 2 бензиностанции е еднакво и в двете посоки. Трябва да намерим най-краткия път от даден начален връх , така че този път да премине през всички върхове в графа и да се върне обратно в първия.

Дефиниция 2.1. Път, който тръгва от даден връх, преминава през всички останали върхове поне веднъж и завършва обратно на първия ще наричаме за по-кратко **обиколка.**

2.1. С повторение или без повторение

За разлика от класическия TSP, в който имаме ограничението, че трябва да посетим всеки връх веднъж и само веднъж, нашата задача няма такова условие. Т.е. възможно е да имаме оптимален път, който посещава една бензиностанция повече от веднъж, например 0-1-2-1-3-4. Това означава, че е пътят от 2-3 е по-дълъг отколкото сумата на пътищата от 2-1 и 1-3, така че за цистерната е по-добре да се върне от 2 до 1 и след това да стигне от 1 до 3, отколкото да стигне от 2 директно до 3. Това е демонстрирано на Фигура 1 (a). Но в нашият вариант на задачата имаме и допълнителното ограничение, че „не е възможно от една бензиностанция да стигнем до друга за по-кратък път от този, който е зададен в тестовите данни за дадената двойка“. Това означава, че примера по-горе не отговаря на условията, т.е. директният път от 2 до 3 винаги ще е по-кратък или равен на сумата на пътищата от 2-1 и 1-3 (Фигура 1(b)).



Фигура 1

(a) Възможна обиколка, когато можем да повтаряме посещения на върхове.

(b) Възможна обиколка, когато можем да повтаряме върхове и не съществува по-кратък път от директния между два върха.

От тук следва и е лесно доказуемо, че ако в нашата задача имаме оптимална обиколка с повторения, като например

$$i \to j \to k \to j \to l \to i,$$

където i, j, k, l са върхове от графа, а " $\to$ " е разстоянието между два върха, то ако премахнем повторенията на j, ще получим обиколка

$$i \to j \to \boldsymbol{k \to l} \to i,$$

която е по-кратка или равна на първата, защото

$$(k \to j) + (j \to l) \leq k \to l$$

Всъщност ще можем да премахнем повторенията на всеки връх и да получим обиколка без повторения, която е с по-малка или равна дължина на тази с повторенията. Единственото условие е върхът, който сме избрали за начало да се повтаря в края на обиколката, но можем да премахнем други негови инстанции в средата. Като обобщение можем да кажем, че ако $OPT(R)$ е дължината на оптимална обиколка, в която има повторения на върхове, а $OPT(NR)$ е дължината на такава, в която няма повторения то:

$$OPT(NR) \leq OPT(R)$$

Какво означава този извод за нашия проблем? Това, че в условието не е дадено ограничение за броя на посещенията на един връх няма значение за крайният ни резултат. Обиколката без повторения винаги ще е по-кратка или равна на такава с повторения.

2.2. Извод

От написаното до тук следва, че можем да приравним нашата задача на класически частен случай на TSP, а именно MTSP (metric TSP) където е валидно правилото за неравенства в триъгълника

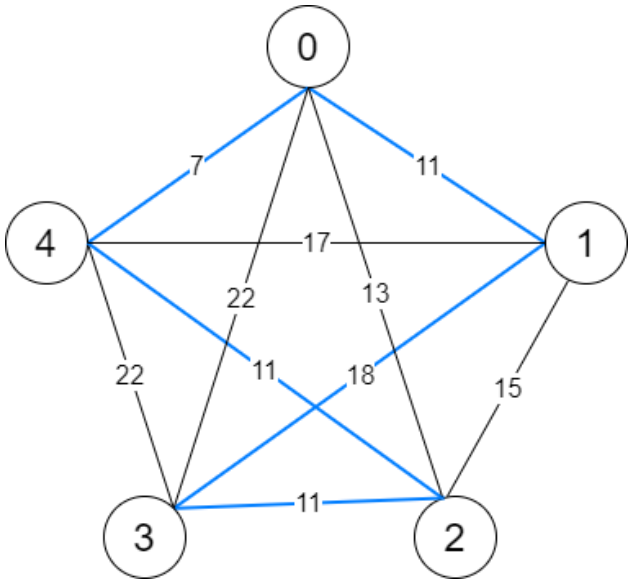
Дефиниция 2.2. Неравенства в триъгълника :
$\forall(i, j, k)\ d(i, j) \leq\ d(i, k) +\ d(k, j)$,
където $d(x, y)$ е функция за разстояние между два върха.

Вече имаме насока, в която можем да направим проучванията си за намиране на решение на проблема. От една страна, TSP е един от най-изследваните и популярни проблеми в компютърните науки и за него съществуват различни решения и алгоритми. От друга страна, към датата на писането на този документ, не е известно бързо решение което да дава точен резултат. Под бързо се има предвид решение, което да дава резултат в полиномно време, т.е. времето за изпълнение е полиномна функция на размера на входните данни. Оказва се, че времето за намиране на точен резултат на нашият проблем не е полиномна, а експоненциална функция на броя на бензиностанциите. За да припомним, ако n е броя на бензиностанциите, то полиномна функция е например $f(n) = 2n$, или $f(n) = n^2 + n + 4$. Експоненциалната функция може да изглежда така $f(n) = 2^n$.

Това което ни остава е да намерим най-подходящ алгоритъм, който да ни дава точен резултат при максимум 17 бензиностанции.

3. Алгоритъм с Изчерпателно Търсене

Най-интуитивният начин за решение е да атакуваме проблема директно, чрез използването на brute-force (груба сила?) алгоритъм, т.е. като изброим (енумерираме) всички възможни обиколки, изчислим техните дължини и накрая изберем като резултат най-малката от тях. Ако приемем, че броят на градовете е N, то всички възможни обиколки от даден връх в графа са $(N-1)!$ (факториел), а **различимите** са $\frac{1}{2}(N-1)!$. Това е така защото една обиколка може да бъде направаена и в обратна посока.



Обиколка	Обиколка в обратна посока	Дължина
0-1-2-3-4-0	0-4-3-2-1-0	11+15+11+22+7=66
0-1-3-2-4-0	0-4-2-3-1-0	11+18+11+11+7=58
0-1-3-4-2-0	0-2-4-3-1-0	11+18+22+11+13=75
0-1-2-4-3-0	0-3-4-2-1-0	11+15+11+22+13=72
0-1-4-3-2-0	0-2-3-4-1-0	11+17+22+11+13=74
0-1-4-2-3-0	0-3-2-4-1-0	11+17+11+11+22=72
0-2-1-3-4-0	0-4-3-1-2-0	13+15+18+22+7=75
0-2-1-4-3-0	0-3-4-1-2-0	13+15+17+22+22=89
0-2-3-1-4-0	0-4-1-3-2-0	13+11+18+17+7=66
0-2-4-1-3-0	0-3-1-4-2-0	13+11+17+18+22=81
0-3-1-2-4-0	0-4-2-1-3-0	22+18+15+11+7=73
0-3-2-1-4-0	0-4-1-2-3-0	22+11+15+17+7=72

Фигура 2.
Диаграма на граф с 5 върха и оптимална обиколка 0-1-3-2-4-0 с дължина 58. В таблицата са дадени всички възможни обиколки с начален връх 0, както и техните дължини.

На Фигура 2Фигура 2. е дадена диаграмата на примерния входен тест от заданието на проблема. В таблицата на Фигура 2 са изброени всички възможни обиколки с начален връх 0, преминаващи през всички останали върхове и връщащи се обратно до началния връх. Виждаме, че броят на възможните обиколки е 24, което е точно :

$$(N-1)! = (5-1)! = 4! = 4 \times 3 \times 2 \times 1 = 24$$

Виждаме също, че една обиколка може да бъде направена и в обратна посока и това няма да промени нейната дължина. Например обиколката 0-1-2-3-4-0 с размер 66 е с еднаква дължина и в обратна посока 0-4-3-2-1-0. Това намалява размера на възможните решения на 12, т.е. 2 пъти:

$$\frac{1}{2}(N-1)! = \frac{(5-1)!}{2} = \frac{4!}{2} = \frac{24}{2} = 12$$

Факториела е експоненциална функция, която нараства изключително бързо. В Таблица 1 е показано как броят на възможните обиколки расте с увеличаване на броя на бензиностанциите.

Брой на бензиностанциите N	Брой на възможните обиколки $(N-1)!$	Време за изчисление при 15000000 операции/сек
2	1	
3	2	
4	6	0.0004 мс
5	24	0.0016 мс
10	362880	24.192 мс
15	8.717829×10^{10}	1.6 часа
18	3.556874×10^{14}	274 дни
20	1.216451×10^{17}	260 години
21	2.4329×10^{18}	5 200 години
22	5.10909×10^{19}	109 202 години
23	1.124×10^{21}	2 402 450 години

Таблица 1.
Нарастване на възможните обиколки според броя на бензиностанциите и времето нужно за изчисление на оптимална обиколка

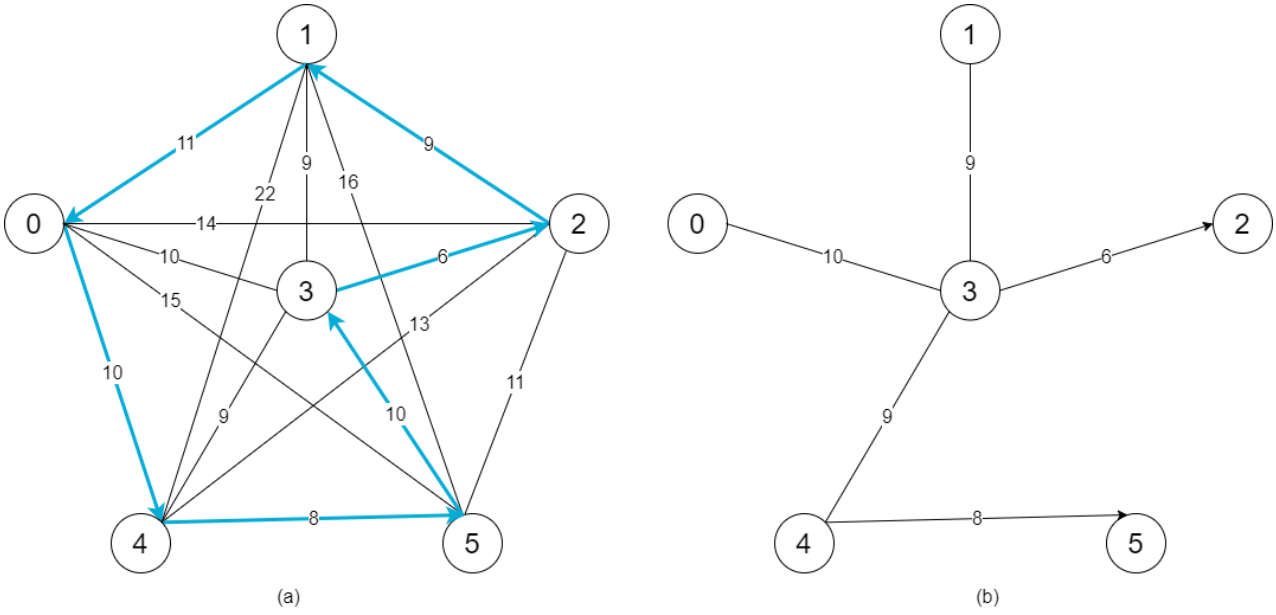
От Таблица 1 виждаме, че ако получим като входни данни 18 бензиностанции (17 + началната), което е максимумът по условие, ще имаме 355687428096000 възможни обиколки! По време на писането на този документ, средно статистически персонален компютър успява да направи изчисление за 15 000 000 обиколки за 1 сек. Това означава, че **(18-1)! ще отнеме около 274 дни**. Увеличаването със само още 1 бензиностанции т.е (**19-1)! ще отнеме 13 години**, а (20-1)! – 5200г.

3.1 Извод

Въпреки че в условието на задачата не е дадено крайно време за намиране на резултат, можем смело да предположим, че търпението на проверяващият е с горна граница от около 7 секунди. Това означава, че не можем да използваме brute-force алгоритъм и се налага да намерим по-бърз вариант за решаването на проблема.

4. Алгоритъм за Приблизително Решение

Нека разгледаме диаграмата на граф (Фигура 3a), който отговаря на условията на нашата задача. Виждаме, че има път от всеки връх, до всеки друг връх, разстоянията между върховете са положителни и отговарят на правилото за неравенства в триъгълника (Дефиниция 2.2). Нека OPT е дължината на оптималната обиколка с начало връх 0 и път 0-4-5-3-2-1-0. OPT = 54. По дефиниция знаем, че ако построим минимално покриващо дърво върху началния граф (Фигура 3b), сумата на ребрата на това дърво ще е минимална т.е. гарантирано е че сумата на разстоянията между върховете в това дърво ще е минималната. Нека означим тази сума като MST.



Фигура 3
(a) Пълен граф, с разстояния отговарящи на правилото за неравенства в триъгълник.
(b) Минимално покриващо дърво на същия граф.

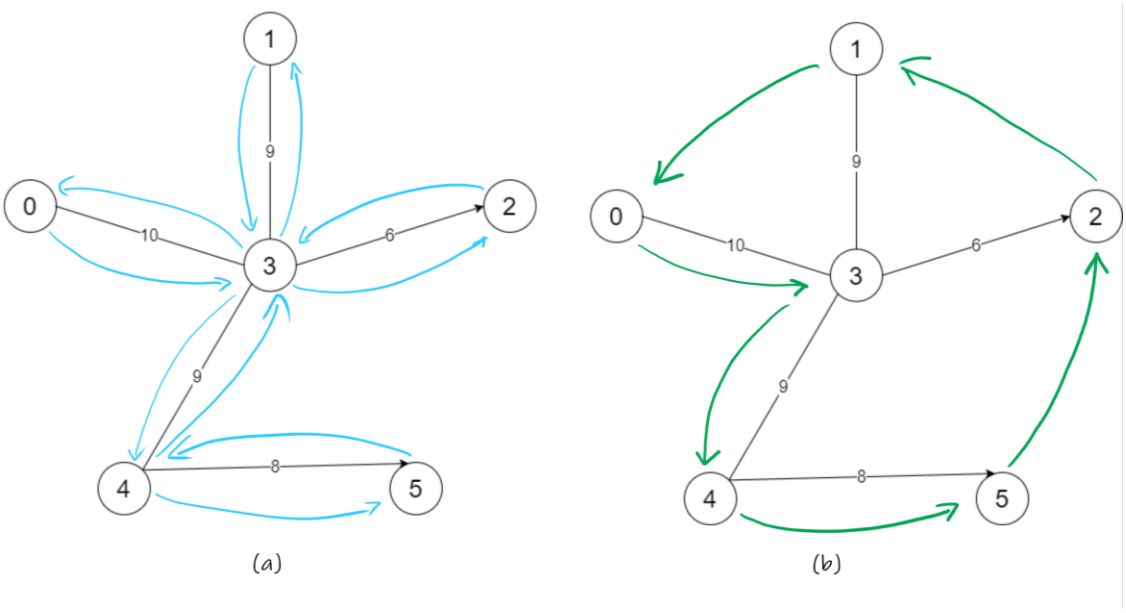
Твърдение 4.1: $MST \leq OPT$

Доказателство чрез противоречие**:** Нека приемем случай на граф, при който $OPT < MST$, което означава, че дължината на оптималната обиколка е по-малка от сумата на ребрата на минималното покриващо дърво. Ако премахнем едно ребро от обиколката на OPT ще получим ацикличен граф, който включва всички върхове точно веднъж, следователно той е покриващо дърво със сума на ребрата $T < OPT$. Получава се, че $T < OPT < MST$, следователно MST не е минимално – противоречие. [2]

■

Ако направим търсене в дълбочина (DFS) върху MST, преминавайки през всяко ребро точно по 2 пъти (Фигура 4а) ще получим обиколка PT с дължина 2 пъти по-голяма от тази на MST:

$$PT = 2 \times MST \Rightarrow PT \leq 2 \times OPT$$



Фигура 4
Със сини стрелки е демонстрирано обхождането на минималното покриващо дърво чрез търсене в дълбочина (DFS)

Записваме всеки връх когато го посещаваме, чрез DFS алгоритъма и за PT получаваме:

$$0 - 3 - 4 - 5 - 4 - 3 - 2 - 3 - 1 - 3 - 0$$

Използвайки това, че началният ни граф отговаря на неравенствата в триъгълника (Дефиниция 2.2), можем да премахнем всяко последващо посещение на връх в пътят на PT и ще получим нов тур T^*:

$$0 - 3 - 4 - 2 - 5 - 1 - 0$$

Следователно:

$$T^* \leq PT \leq 2 \times OPT,$$

$$T^* \leq 2 \times OPT$$

Като заключение можем да твърдим, че решението от описаните стъпки по-горе ще ни гарантира краен резултат, който в най-лошият случай ще е 2 пъти по-голям от оптималния. Самият алгоритъм можем да обобщим така:

1. Намираме минимално покриващо дърво (Прим, Крускал)
2. Обхождаме това дърво чрез търсене в дълбочина и записваме всеки връх по реда на посещението му в поредица.
3. Обхождайки поредицата от ляво надясно, премахваме всеки връх, който вече е бил записан поне веднъж.
4. В края на поредицата добавяме началния връх.

В Таблица 2 може да се види приблизително сравнение на времевата сложност (time complexity) между алгоритъма чрез изчерпателно търсене(Глава 3) и алгоритъма за приблизителен резултат разгледан тук. Виждаме, че в сравнение с brute-force алгоритъма, MST+DFS работи светкавично, въпреки че е с квадратно време $O(N^2)$.

Брой на бензиностанциите N	Сложност по време при brute force $O(N-1)!$	Сложност по време при MST+DFS $O(N^2)$
2	1	4
3	2	9
4	6	*16*
5	24	*25*
10	362880	*100*
15	8.717829×10^{10}	*225*
18	3.556874×10^{14}	289
20	1.216451×10^{17}	400

Таблица 2

Приблизително сравнение на времевата сложност (time complexity) на двата алгоритъма разгледани до тук.

4.1 Извод

За съжаление, както се вижда и от заглавието на текущата глава, бързият алгоритъм е неточен. Дава ни приблизителна стойност за най-краткия път. Единствената гаранция е, че резултатът от ще бъде максимум 2 пъти по-голям от дължината на оптималния път. От експериментални данни знаем, че оптималната обиколка за примерния граф е 54 и нейният път минава през 0-4-5-4-3-2-1-0. Алгоритъмът, който описахме изчислява обиколка с дължина 58 и път 0-3-4-2-5-1-0. В практиката съществува и по-точен алгоритъм – този на Christofides [3], който гарантира, че в най-лошият случай, резултатът ще е 1.5 пъти по-голям от оптималния, което е значително подобрение. Нито един от приблизителните алгоритми не е решение на нашата задача. Трябва да получим точен резултат, който лесно да може да бъде тестван за коректност.

5. Bellman-Held-Karp

Съществуват много публикации, изследвания и литература по проблема за „пътуващия търговец“ (TSP). Както беше споменато в Глава2, TSP се класифицира като NP-hard проблем. Опростено казано, не можем да изчислим точен резултат в полиномно време. От една страна, ако искаме точно решение, ще трябва да жертваме време за изпълнение. От друга, ако искаме бързо решение, ще трябва да жертваме точност на резултата. От условието на задачата е видимо, че търсим точен резултат. Разбрахме също, че чрез метода на изчерпателно търсене ще са ни нужни около 274 дни за да изчислим точен резултат при брой на бензиностанциите от 17. Можем ли да се справим по-добре? [4]

През 1962 година, Ричард Белман от една страна и Майкъл Хелд и Ричард Карп от друга, независимо едни от други предлагат алгоритъм за решение на TSP, който използва техника наречена динамично програмиране и постига сложност по време $O(N^2 2^N)$. В

Брой на бензиностанциите N	Време за изпълнение при brute force $O(N-1)!$	Време за изпълнение при Bellman-Held-Karp $(N-1)(N-2)2^{N-3}+(N-1)$
5	0.0016 мс	0.001 мс
10	24.192 мс	0.239 мс
15	1.6 часа	21.3 мс
18	274 дни	262 мс
20	260 години	1.34 сек
21	5200 години	3 сек
22	109202 години	6.6 сек
23	2402450 години	15 сек

Таблица 3 е направено сравнение на броя операции при алгоритъм с изчерпателно търсене и алгоритъма на Bellman-Held-Karp. Въпреки, че алгоритъма на Хелд-Карп също е с експоненциална сложност по време, от таблицата е видимо, че работи много, много по-бързо. Като използваме данните, че съвременен компютър може да извършва изчисления за 15 000 000 обиколки в секунда (грубо) получаваме, че за 18 бензиностанции ще са ни нужни 262 милисекунди, ако използваме алгоритъма на Хелд-Карп.

Брой на бензиностанциите N	Време за изпълнение при brute force $O(N-1)!$	Време за изпълнение при Bellman-Held-Karp $(N-1)(N-2)2^{N-3}+(N-1)$
5	0.0016 мс	0.001 мс
10	24.192 мс	0.239 мс
15	1.6 часа	21.3 мс
18	274 дни	262 мс
20	260 години	1.34 сек
21	5200 години	3 сек
22	109202 години	6.6 сек
23	2402450 години	15 сек

Таблица 3
Приблизително сравнение на времето за изчисление на резултат при норма от 15млн. обиколки в секунда между алгоритъм с изчерпателно търсене и алгоритъма на Bellman-Held-Karp

Интересното е, че за 60 години откакто е предложен този по-бърз алгоритъм, не е постигнато значително подобрение във времето за намиране на точно решение. И въпреки че няма доказателство, че не съществува алгоритъм, който решава проблема в полиномно време, много учени и изследователи смятат, че такъв не съществува. От това следва, че алгоритъмът на Хелд-Карп е може би най-подходящият кандидат за решение на нашата задача. Нека разгледаме самия алгоритъм по-подробно.

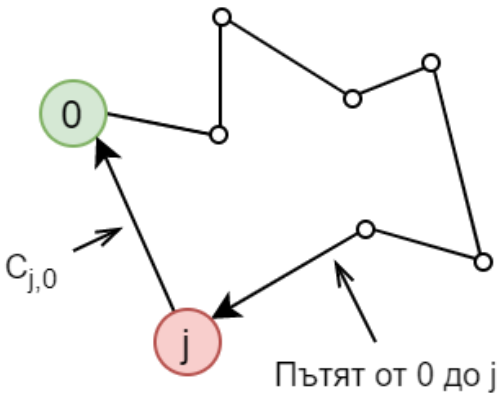
5.1 Динамично програмиране

В основата на алгоритъма на Белман-Хелд-Карп лежи парадигмата на динамичното програмиране, която накратко можем да формулираме така:

1. Идентифицираме сравнително малка колекция от подзадачи на главния проблем
2. Показваме как бързо и коректно да решим „големи“ подзадачи, като преизползваме решенията на „малките“
3. Показваме как бързо и коректно да направим заключение за финалния резултат на базата на решенията на всички под-проблеми.

Ключът към прилагането на динамичното програмиране е идентифицирането на правилната колекция от подзадачи. Най-добрият начин да открием такива под-проблеми е да анализираме как оптималното решение може да е съставено от оптимални решения на по-малки подзадачи . [4]

Да предположим, че разполагаме с оптимална обиколка T, която преминава през върховете от $V = \{0,1,2, \dots, n\}$, където $n \geq 3$. Как би изглеждала тази обиколка? По-колко начина би могла да бъде съставена от оптимални решения на по-малки под-проблеми. Нека нашата обиколка започва и завършва на 0. Ако разгледаме последното ребро, което се връща обратно в 0 от някакъв връх j, то тогава нашата обиколка ще бъде съставена от минималния път, без цикли от 0 до j последван от последното ребро от j обратно до 0. След като j може да бъде всеки един връх от 1 до n, то в този случай съществуват точно n-1 на брой пътя с j за последен връх. Например ако означим с $M(j, S)$ минимален път, който започва от 0, преминава през всички върхове на S и завършва на j, където $S = V - \{j\}$, то в нашия конкретен пример можем да запишем $(M(\{0,1,2,3\}, 4))$,



Фигура 5
Пътят от 0 до j минава през всички върхове на T и е с минимална дължина.

което ще означава, че тръгваме от 0, минаваме по-някакъв път през 1,2 и 3 и завършваме в 4 и това е минималния път от 0 до 4.

Виждаме, че за нашата задача можем да имаме точно 4 такива пътя и единият от тях ще е оптималният. Нека с $D(k,j)$ записваме директното разстояние между 2 върха k и j. Тогава формулата за нашата оптимална обиколка OPT ще изглежда така:

$$OPT = min \begin{Bmatrix} (M(\{0,1,2,3\},4)) + D(4,0) \\ (M(\{0,2,3,4\},1)) + D(1,0) \\ (M(\{0,1,2,4\},3)) + D(3,0) \\ (M(\{0,1,3,4\},2)) + D(2,0) \end{Bmatrix}$$

5.2 Подзадачи

Нека атакуваме задачата като започнем да намираме решения на подзадачи. Можем да мислим за подзадачите като подмножества на V. Например при подмножествата $S \subset V$ където $|S| = 2$, възможните варианти са 4, а именно S = {0,1}, S = {0,2}, S = {0,3} и S = {0,4}. При тях имаме комбинация само на 2 върха, единият от които задължително е стартовия връх 0. Ако използваме двоичното записване на цели числа, можем много лесно да визуализираме комбинациите в подмножествата. Нека приемем, че 11111 представлява запис на пълното множество от 5 върха. Индексите на това число (т.е. местата на единиците) се четат от дясно на ляво и представляват номерата на върховете (бензиностанциите) от 0 до 4. При пълното множество имаме единици на всеки индекс което означава, че имаме всички възможни комбинации на пътища, които преминават през всички върхове. Ако имаме записано двоичното 00011, то тогава имаме подмножество на пълното и имаме единици само на индекси 0 и 1. Това означава, че съществува път само между връх 0 и връх 1. В Таблица 4 са дадени няколко примера за по-добро разбиране на техниката която ще използваме за генериране и записване на подмножества.

Размер на под-множеството	Двоичен вид на под-множеството	Десетична стойност	Обяснение
|S| = 2	00011	3	Понеже размера на под-проблема е |S|=2 имаме единици само на 2 места в двоичния запис. Когато единиците са на индекси 0 и 1, имаме път между върхове 0 и 1.
	01001	9	Тук имаме път между 0 и 3
	10001	17	А тук между 0 и 4
|S| = 3	00111	7	Размера е 3 и затова имаме 3 единици. Пътят тръгва от 0 и минава през 1 и 2. Последователността може да е всяка от възможните: 0-1-2 или 0-2-1
|S| = 4	10111	23	Тук пътят минава от 0 през 1,2 и 4. Във всеки двоичен запис задължително имаме единица на нулевия индекс. Това е така, защото връх 0 е начало на всички възможни пътища и трябва да присъства в записа.

Таблица 4

Двоичен начин на записване на възможните подмножества от върхове при пълно множество от 5 елемента

5.3 Мемоизация

Важна техника при динамичното програмиране е запаметяването на решения на подзадачи в подходяща структура от данни, като по този начин те могат да бъдат преизползвани за изчисляването на подзадачи с по-голям размер. Тук трябва да отбележим, че подобряването на скоростта чрез алгоритъма на Хелд-Карп е за сметка на изчислителна памет. За нашето решение като структура от данни ще използваме двумерен масив M със следните характеристики (Таблица 5):

1. Редовете на масива са номерата на върховете, на които завършва пътя за дадена подзадача. Редовете са $N - 1$ на брой.
2. Колоните са 2^N на брой. Това е така, защото възможните подмножества на едно множество V са $2^{|V|}$. В нашата задача имаме общо 5 града, следователно $2^5 = 32$ колони.
3. Колоните са десетичните стойности на двоичните записи на подмножествата от върхове. Това означава, че ако имаме размер на подзадачата равен на 3, и подмножество 00111, то резултатът за него ще бъде отразен в колона 7 ($00111_2 = 7_{10}$)
4. Можем да оптимизираме матрицата, като пропуснем всички четни индекси на колоните. Щом задължително трябва да тръгваме от връх 0, то в двоичната форма на подмножеството винаги ще имаме единица на нулевия индекс. От това следва, че десетичните стойности ще са само нечетни.
5. Също така можем да премахнем индекси 0 и 1 на колоните защото винаги ще имаме път през 2 или повече върха.

	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
	00011	00101	00111	01001	01011	01101	01111	10001	10011	10101	10111	11001	11011	11101	11111
1															
2															
3															
4															

Таблица 5
Масив M с N-1 реда и 2^N стълба, в който ще запаметяваме резултатите за подзадачите на проблема

5.4 Алгоритъм

Ще приемем, че входните данни ще бъдат представени като матрица $D = N \times N$ (Таблица 6), където N е общият брой на бензиностанциите. Наредените двойки D[j, k] ще ни дават директните разстояния между две бензиностанции j и k. Алгоритъмът, който ще използваме изглежда така :

	0	1	2	3	4
0	0	11	13	22	7
1	11	0	15	18	17
2	13	15	0	11	11
3	22	18	11	0	22
4	7	17	11	22	0

Таблица 6
Матрица D за разстоянията между всеки един връх от графа

```
// псевдокод за Белман-Хелд-Карп
1. нека s = 2
2. За всяко j от 1 до N-1:
3.   За всяко S с мощност |S| = s и 0 ∈ S:
4.        Ако j∈ S и 0 ∈ S:
5.             M[j,S] = D[j,0]

6. За всяко s = 3 до N:
7.   За всяко S ∈ V с мощност |S| = s и 0 ∈ S:
8.        За всяко j ∈ S-{0}:
9.             нека min = +∞
10.            За всяко k ∈ S, k≠ 0,k ≠ j
11.                 нека path = M[k, S-{j}] + D(j, k)
12                  Ako path < min:
13.                      min = path
14.            M[j, S] = min

15.За всяко j от 1 до N:
16.     Намираме OPT = min(M[j, V-{j}])+ D[j,0]
        и го връщаме като финален резултат
```

5.5 Обяснение на алгоритъма

Първо инициализираме размера s на собствените подмножеството от върхове $S \subset V$, с които ще работим, да е равен на 2. Това означава, че понеже 0 задължително трябва да присъства в подмножеството, ще имаме следните възможности 00011, 00101, 01001 и 1001.Всяко от тези подмножества описва точно ребрата от връх 0 до всеки друг връх. За всяко от тези подмножества имаме вложен цикъл, който за всеки връх различен от 0 проверява дали този връх е в подмножеството. Например ако имаме текущо подмножество 00101 и j = 2, виждаме, че на индекс 2 от подмножеството имаме единица, което означава, че j се включва в текущото S. Ако това условие е изпълнено, то тогава прочитаме разстоянието между j и 0 от D[j,0] и го записваме в масива M на индекси ред j и колона S. За индекса на колоната взимаме десетичната стойност на S. Ако j не е в S, продължаваме с цикъла докато изброим всички j от 1 до 4. След като приключи тази част от алгоритъм (ред 1-5) нашият масив M ще изглежда както е показано в Таблица 7.

	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
	00011	00101	00111	01001	01011	01101	01111	10001	10011	10101	10111	11001	11011	11101	11111
1	11														
2		13													
3				22											
4								7							

Таблица 7

Така ще изглежда масивът M след изпълнението на алгоритъма в редове 1-5

За всяка клетка от масива M запаметяваме най-краткия път, който започва от 0, минава през всички други върхове с изключение на върха j (с индекс реда на клетката), и завършва на j. Т.е при клетка M[1,3] имаме най-краткия път от 0, който преминава през всички върхове на подмножеството S-{1} (00011 – 00010 = 00001) и завършва на 1. В този базов случай S-{1} е подмножеството което включва единствено стартовия връх 0, така че най-краткото разстояние е това от 0 до 1, което можем да вземе директно от масив D.

От ред 6 до ред 10 започва изчисляването на решения за подмножества с размер от 3 до 5. Нека приемем, че s = 3. На ред 7 започваме цикъл, който итерира върху всички възможни подмножества с размер 3, в които задължително имаме връх 0. Това са всички колони от M оцветени в зелено : 00111, 01011, 01101, 10011, 10101 и 11001. На ред 8 започваме да изброяваме всички върхове j, които са в текущото подмножество и са различни от връх 0. Можем да мислим за j като за върхът, на който завършва пътят за текущото подмножество. При 00111 и j = 1 виждаме, че пътят започва от 0 (винаги) и завършва на j = 1. Което означава, че в този случай имаме единствена възможност 0-2-1. На ред 9 създаваме временна променлива в която ще записваме минималната стойност на текущия път, който изчисляваме. При инициализацията на min даваме начална стойност от положителна безкрайност.
На ред 10 започваме изброяването на всички върхове k, които са в текущото подмножество и не са връх 0 или връх j. k е върхът, на който завършва предишният изчислен минимален път върху подмножество което не включва j.

На ред 11 сме в „най-дълбоката“ част на четирите вложени цикъла от ред 6 до 14. В променливата path изчисляваме текущия път за подмножеството на базата на предишни данни от масива M и данни от масива D. Нека приемем текущи стойности за както следва:
s = 3,

S = 00111, решаваме за подмножество съставено от върховете {0, 1, 2}
j = 1, означава, че текущия път през върховете {0,1,2} трябва да завърши на вр. 1
min = +inf
При k = 1 или k = 0 пропускаме изпълнението на цикъла
защото имаме условието $k \neq j$ и $k \neq 0$.

При k = 2 :

В следващият ред (11) се крие най-важната част от алгоритъма:

path = M[k, S-{j}] + D [k, j]

Този ред преизползва решението запомнено в M за най-краткият път, който започва от 0, завършва на k и минава през всички върхове в подмножеството S-{j} и накрая добавя към този път разстоянието между j и k.

щом k = 2 и S-{j} = 00111 – индекс 1 = 00101, то

path = M[2, 00101] + D[2,1] = 13 + 15 = 28

На следващият ред правим проверка дали path < min и ако това е така, то новото най-кратко разстояние за текущия път ще бъде равно на path.

След като намерим минималния път за всички стойности на k, запаметяваме резултата в M[j, S] =M[1, 00111] = 28

Не можем да имаме k = 3 и k = 4 защото индексите 3 и 4 не са в 00111

В Таблица 8 можем да видим всички изчисления, които ще направи нашият алгоритъм в случая когато решаваме подзадача, включваща само 3 върха.

s	S	j	S - { j }	k	път	path =M [k, S - { j } + D [k, j]	Min	M [j, S]
3	00111 (7)	1	00101 (5)	2	0-2-1	13 + 15 = 28	28	28
		2	00011 (3)	1	0-1-2	11 + 15 = 26	26	26
	01011 (11)	1	01001 (9)	3	0-3-1	22 + 18 = 40	40	40
		3	00011 (3)	1	0-1-3	11 + 18 = 29	29	29
	01101 (13)	2	01001 (9)	3	0-3-2	22 + 11 = 33	33	33
		3	00101 (5)	2	0-2-3	13 + 11 = 24	24	24
	10011 (19)	1	10001 (17)	4	0-4-1	7 + 17 = 24	24	24
		4	00011 (3)	1	0-1-4	11 + 17 = 28	28	28
	10101 (21)	2	10001 (17)	4	0-4-2	7 + 11 = 18	18	18
		4	00101 (5)	2	0-2-4	13 + 11 = 24	24	24
	11001 (25)	3	10001 (17)	4	0-4-2	7 + 22 = 29	29	29
		4	01001 (9)	3	0-3-4	22 + 22 = 44	44	44

Таблица 8

Показани са всички изчисления, които ще направи нашият алгоритъм при размер на подмножествата s = 3

В Таблица 9 са отразени резултатите от изчисленията на алгоритъма до този момент. При изпълнението забелязваме, как програмата преизползва решения направени в предната стъпка, когато размера на подмножествата беше 2.

	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
	00011	00101	00111	01001	01011	01101	01111	10001	10011	10101	10111	11001	11011	11101	11111
1	11		28		40				24						
2		13	26			33				18					
3				22	29	24						29			
4								7	28	24		44			

Таблица 9

Така ще изглежда масивът M след решаването на подмножествата с размер s = 3

В таблиците 10, 11, 12, 13 можем да проследим изчисленията на алгоритъма за всички променливи и за размери на подмножествата 4 и максималния 5.

s	S	j	S - { j }	k	Път	path =M [k, S - { j } + D [k, j]	min	M [j, S]
4	01111 (15)	1	01101 (13)	2	0-3-2-1	33 + 15 = 48	42	42
				3	0-2-3-1	24 + 18 = 42		
		2	01011 (11)	1	0-3-1-2	40 + 15 = 55	40	40
				3	0-1-3-2	29 + 11 = 40		
		3	00111 (7)	1	0-2-1-3	28 + 18 = 46	37	37
				2	0-1-2-3	26 + 11 = 37		
	10111 (23)	1	10101 (21)	2	0-4-2-1	18 + 15 = 33	33	33
				4	0-2-4-1	24 + 17 = 41		
		2	10011 (19)	1	0-4-1-2	24 + 15 = 39	39	39
				4	0-1-4-2	28 + 11 = 39		
		4	00111 (7)	1	0-2-1-4	28 + 17 = 46	37	37
				2	0-1-2-4	26 + 11 = 37		
	11011 (27)	1	11001 (25)	3	0-4-3-1	29 + 18 = 47	47	47
				4	0-3-4-1	44 + 17 = 61		
		3	10011 (19)	1	0-4-1-3	24 + 18 = 42	42	42
				4	0-1-4-3	28 + 22 = 50		
		4	01011 (11)	1	0-3-1-4	40 + 17 = 57	51	51
				3	0-1-3-4	29 + 22 = 51		
	11101 (29)	2	11001 (25)	3	0-4-3-2	29 +11 = 40	40	40
				4	0-3-4-2	44 + 11 = 55		
		3	10101 (21)	2	0-4-2-3	18 + 11 = 29	29	29
				4	0-2-4-3	24 + 22 = 46		
		4	01101 (13)	2	0-3-2-4	33 + 11 = 44	44	44
				3	0-2-3-4	24 + 22 = 46		

Таблица 10

Показани са всички изчисления, които ще направи нашият алгоритъм при размер на подмножествата s = 4

	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
	00011	00101	00111	01001	01011	01101	01111	10001	10011	10101	10111	11001	11011	11101	11111
1	11		28		40		42		24		33		47		
2		13	26			33	40			18	39			40	
3				22	29	24	37					29	42	29	
4								7	28	24	37	44	51	44	

Таблица 11

Така ще изглежда масивът M след решаването на подмножествата с размер s = 4

s	S	j	S - { j }	k	Път	path = M [k, S - { j } + D [k, j]	min	M [j, S]
5	11111(31)	1	11101 (29)	2	0-{3,4}-2-1	40 + 15 = 55	47	47
				3	0-{2,4}-3-1	29 + 18 = 47		
				4	0-{2,3}-4-1	44 + 17 = 61		
		2	11011 (27)	1	0-{3,4}-1-2	47 + 15 = 62	53	53
				3	0-{1,4}-3-2	42 + 11 = 53		
				4	0-{1,3}-4-2	51 + 11 = 63		
		3	10111 (23)	1	0-{2,4}-1-3	33 + 18 = 51	50	50
				2	0-{1,4}-2-3	39 + 11 = 50		
				4	0-{1,2}-4-3	37 + 22 = 59		
		4	01111 (15)	1	0-{2,3}-1-4	42 + 17 = 59	51	51
				2	0-{1,3}-2-4	40 + 11 = 51		
				3	0-{1,2}-3-4	37 + 22 = 59		

Таблица 12

Така ще изглежда масивът M след решаването на подмножествата с размер s = 4

	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
	00011	00101	00111	01001	01011	01101	01111	10001	10011	10101	10111	11001	11011	11101	11111
1	11		28		40		42		24		33		47		43
2		13	26			33	40			18	39			40	62
3				22	29	24	37					29	42	29	41
4								7	28	24	37	44	51	26	41

Таблица 13

Така ще изглежда масивът M след решаването на подмножествата с размер s = 4

За последната колона от M получаваме минималните пътища тръгващи от 0, преминаващи през всички върхове и завършващи съответно на връх 1, 2, 3, 4. В последната част от алгоритъма в редове 15 – 16, за всеки от тези пътища добавяме последната отсечка, която затваря обиколката към 0 и избираме обиколката с най-малка дължина като краен резултат:

```
M[j, V]+ D[j,0]
```

j = 1 => M[1, 11111] + D[1,0] = 47 + 11 = 58
j = 2 => M[2, 11111] + D[2,0] = 53 + 13 = 66
j = 3 => M[3, 11111] + D[3,0] = 50 + 22 = 77
j = 4 => M[4, 11111] + D[4,0] = 51 + 7 = 58

Най-кратката от четирите обиколки е с дължина 58

Псевдокодът показан в 5.4 е достатъчно подробен, така че лесно да можем да имплементираме решението на избран от нас език за програмиране. Като допълнителен проблем при имплементацията остава генерирането на подмножества с дадена големина, но това е лесно решим комбинаторен проблем. Четенето на входните данни е тривиален проблем и не е нужно да отделяме време за обяснението му. Може да бъде разгледан директно в програмния код. В Приложение 1 е поместена информация за имплементацията на проблема написана на C#. Тя е съобразена със спецификацията на заданието.

6. Заключение

Проблемът за пътуващия търговец – TSP е един от най-познатите NP-hard оптимизационни проблеми, а приложенията му се срещат в различни сфери като транспорт и логистика, роботизирано производство и много други. От направеното проучване на проблема виждаме, че можем да получим точни оптимални решения за сметка на изчислително време, използвайки алгоритми с динамично оптимиране, като този на Белман-Хелд-Карп. Но дори и тогава обемът от данни, върху който можем да правим изчисленията е твърде ограничен. Когато са нужни бързи резултати или обема на проблема е по-голям, се използват приблизителни алгоритми, които в практиката дават много добри резултати. Използван като образователна задача, TSP дава възможност за научаването на много нова и задълбочена информация от сферата на компютърните науки, информатиката и дискретната математика.

7. Референции

[1] R. Bellman, "Dynamic Programming Treatment of the Travelling Salesman Problem," *J. Assoc. Comput. Mach,* pp. 61-63, 1962.

[2] C. L. prof., "Lecture 3, Comp260: Advanced Algorithms," Tufts University, 2002.

[3] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Graduate School of Industrial Administration, Pittsburgh, 1976.

[4] T. Roughgarden, Algorithms Illuminated, New York, NY: Soundlikeyourself Publishing, LLC, 2020, pp. 105-112.

[5] M. T. Goodrich and R. Tamassia, "18.1.2 The Christofides Approximation Algorithm", Algorithm Design and Applications, Wiley, 2015.

8. Таблици и фигури

9. Приложение 1

Кодът за имплементацията на проблема може да бъде разгледан в GitHub на адрес: https://github.com/KostaDinkov/unibit-op-tsp