Етапи на пълното разработване на алгоритми и програми

- Постановка на задачата
- Построяване на модел
- Разработване на алгоритъм
- Проверка на правилността на алгоритъма
- Реализация на алгоритъма
- Анализ на алгоритъма и неговата сложност
- Проверка на програмата
- Съставяне на документацията

I. Постановка на задачата

Полезни въпроси за лошо формулирано задачи:

- разбираема ли използваната терминология?
- какво е дадено?
- какво се търси?
- какви данни липсват?
- има ли излишни данни?
- какви допускания (ограничения) са приети?
- как да се намери решението?

Задача за търговския пътник:

Агент по продажба на компютри (търговски пътник) обслужва територия от 20 града. Фирмата му заплаща само 50% от пътните разходи с автомобил. Агентът е пресметнал, колко му струва пътя между всеки два града, и иска да намали пътните си разходи.

Какво е дадено?

- списък на градовете и
- съответстваща му матрица със стойности на разстоянията между всеки два града, т.е. двумерен масив C с размер 20 x 20 и с елементи C_{ij}.

Какво се търси?

Не може да се каже еднозначно.

Уточнение:

- необходим е маршрут, който да започва от определен (базов) град, да завършва в него и да минава по веднъж през всеки от останалите градове.

Очакван резултат:

- списък, в който всеки град фигурира по веднъж, като базовия град стои в началото и в края. Цената на обиколката, т.е. сумата на цените на разстоянията между всеки два града в списъка, трябва да е минимална.

II. Построяване на модел

Процесът не може да бъде автоматизиран.
Всяка задача се разглежда индивидуално.
Полезно е да се отговори на следните въпроси:

1. Кои математически структури най-добре подхождат за решаване на задачата?
 - да опишем математически това, което е дадено и това, което се търси;
- да отчетем удобството на представяне, простота на обработване и ограничеността на знанията си.

2. Съществуват ли решения на аналогични задачи?

След избора на математическата структура задачата трябва да се формулира отново в термините на обектите на избрания модел.

Моделът се смята за задоволителен, ако:

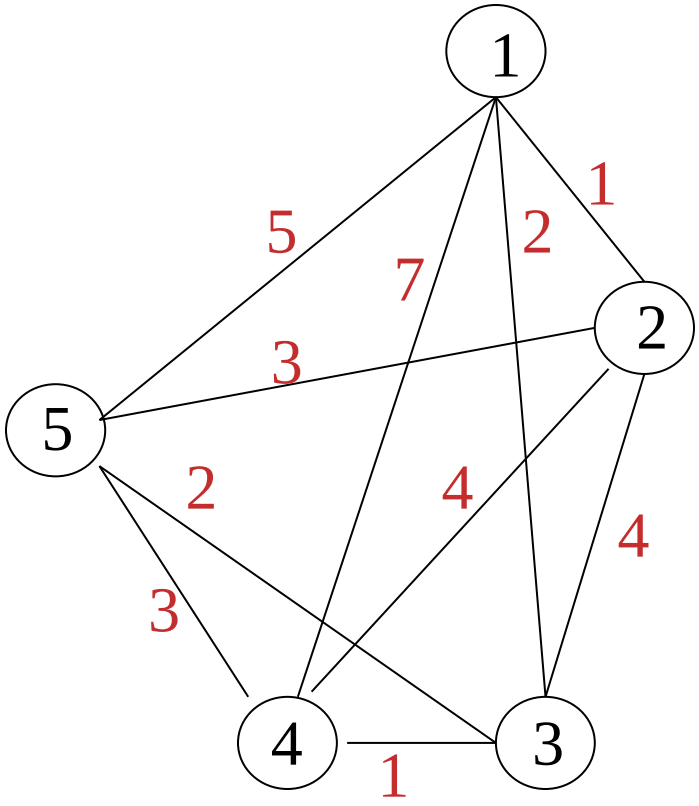
- цялата (или поне най-съществената) информация е добре описана чрез математическите обекти;
- между величините на модела фигурира и самия резултат;
- може (и то удобно!) да се работи с модела;
- намерени са полезни съотношения между обектите на модела, което ще улесни обработката.

Към задачата за търговския пътник:

- Решавали ли сме подобна задача?

Нека да приемем, че градовете са 5 (n=5) и че цената на пътуването от град i в град j съвпада с цената на пътя в обратната посока (пътищата са двупосочни):

	1	2	3	4	5
1	-	1	2	7	5
2	1	-	4	4	3
3	2	4	-	1	2
4	7	4	1	-	3
5	5	3	2	3	-



Схемата (или модел), която получихме, се нарича граф или мрежа.

- Какво се търси (в математическия смисъл)?

В термините на теорията на графите може да се каже, че се търси затворен цикъл, който започва от базов град и има минимална дължина.

Например:

ако изберем за базов град 1, един от възможните цикли е

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

цената му е $5 + 2 + 1 + 4 + 1 = 13$ - дали е MIN?

Това е класически пример на задача, която

- лесно се формулира
- лесно се моделира
- но трудно се решава!

III. Разработване на алгоритъма

Алгоритъмът зависи от метода за разработване, който, от своя страна, зависи от избрания модел.

По различни методи могат да се разработят много алгоритми, всеки от които ще има своя собствена ефективност.

Към примера:

Градовете са номерирани с цели числа от 1 до **n**. Да приемем, че базовия град е с номер **n**. Тогава всеки маршрут ще съответства на определено подреждане на числата от 1 до **n-1.**

На всеки маршрут ще съответства определена комбинация-подреждане и на всяко подреждане – определен маршрут. На лице е взаимно еднозначно съответствие.

Очевидно е, че за всяко подреждане може да се проследи по модела съответния маршрут и да се определи цената му.

Алгоритъм:

Образуване на всички пермутации от първите **n**-1 цели положителни числа.
За всяка пермутация се определя съответстващия й маршрут и цената му.
Търсене на най-малката цена.

Входни данни: - брой градове **n**
- ценова матрица **C**(**n**,**n**)

Стъпка:

0 (инициализация) LIST = 0; MIN = ∞

1 (образуване на пермутациите) За **i** от 1 до **n**-1
повтаряй стъпки от 2 до 4;
край;

2 (образуване на нова пермутация) Получаване и запомняне на **i**-та пермутация
на числата 1, 2, …, **n**-1;
P = **i**-та пермутация (как?);

3 (изграждане на нов маршрут) Построяване на маршрута **T**(**P**),
съответстващ на пермутацията **P**, и
пресмятане на цената **COST**(**T**(**P**))

4 (сравнение) Ако **COST**(**T**(**P**))< **MIN** тогава
LIST=**T**(**P**), **MIN**=**COST**(**T**(**P**))

IV. Коректност на алгоритъма

- тестване
- теоретично доказателство

Ако алгоритъмът се състои от **n** последователни стъпки, то трябва да се докаже :

1) коректността на всяка една от стъпките;
2) крайността на алгоритъма

Задача за търговския пътник:

Т.к. се проверява всеки маршрут, то ще бъде проверен и запомнен и минималния (с минималната стойност).

Алгоритъмът ще завърши, т.к. броят на проверяваните маршрути е краен.

Коректността на алгоритъма не означава, че той е ефективен. (Алгоритмите, реализиращи пълна проверка на всички възможности обикновено не са ефективни!)

V. Реализация на алгоритъма

Алгоритъмът е описан като последователност от стъпки. Предстои реализацията му – написване на програма.

Възможните трудности:

1) не всички стъпки са формулирани детайлно (така, че директно могат да се запишат на езика за програмиране – например, получаване на пермутации);

2) трудности при представяне на отделни елементи на модела чрез структурите данни на езика за програмиране. Необходимо е:

- да се определят основните променливи и типът им;
- какви масиви са нужни и с каква размерност;
- необходимо ли е да се използва динамичната памет;
- какви подпрограми (функции) са необходими.

След това се взема решение, какъв език за програмиране ще се използва.

Не само алгоритъмът, но и конкретна реализация на програмата влияе на изискванията към ПАМЕТ и ВРЕМЕ. Основен метод: Top-down програмиране.

VI. Анализ на алгоритъма и неговата сложност

В два аспекта:

- практически, какъв обем памет и колко време ще са необходими за решаване на задачата;
- теоретичен – изчислява се сложността на алгоритъма **O(f(n))** с цел да се определи, дали алгоритъмът е полиномиален или експоненциален.

Задача за търговския пътник:

За **n** града трябва да получим пермутации от (**n-1**) цели положителни числа, т.е (**n-1**)! пермутации. Ако приемем, че всяка пермутация се получава с една операция, то тази част от алгоритъма може да се оцени с **O**((**n-1**)!). Алгоритъмът включва и други действия – търсене на маршрут, изчисляване на цената му, което изисква минимум **n** операции. Затова горната граница на целия алгоритъм ще бъде не по-малка от **O**(**n**!).

Ако търговския пътник трябва да обиколи 20 града (**n**=20, $20! \approx 2 \times 10^{18}$) и ние разполагаме с компютър, изпълняващ една елементарна операция за 10^{-8} сек., необходимото за програмата време е:

$$T = \frac{2 \times 10^{18}}{10^8 \times 3600 \times 24 \times 365} \approx 700 \text{ години}$$

Алгоритъмът е експоненциален - $O(n!)$.

Подобни алгоритми (с пълна, изчерпателна проверка на всички възможни варианти) са неефективни и могат да се използват само при малки ***n***.

Изход – използване на други известни подходи и методи.

Евристични алгоритми

Свойства:

- дават добри, но не винаги оптимални решения;
- могат да бъдат реализирани бързо и лесно

Нямат универсален подход за създаването им (“Еврика!”).

Препоръки:

Всички изисквания към решението се разделят на 2 групи:

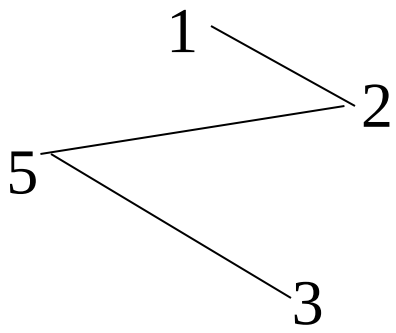
1. Тези, които лесно се постигат
2. Тези, които не се постигат лесно.

или

1. Тези, които са задължителни
2. Тези, за които може да се направи компромис.

Целта е да се изгради алгоритъм, гарантиращ изпълнение на изискванията от 1 група и не гарантиращ тези от 2-та (което не означава, че няма да се правят опити да се изпълнят и те).

Задача за търговския пътник:

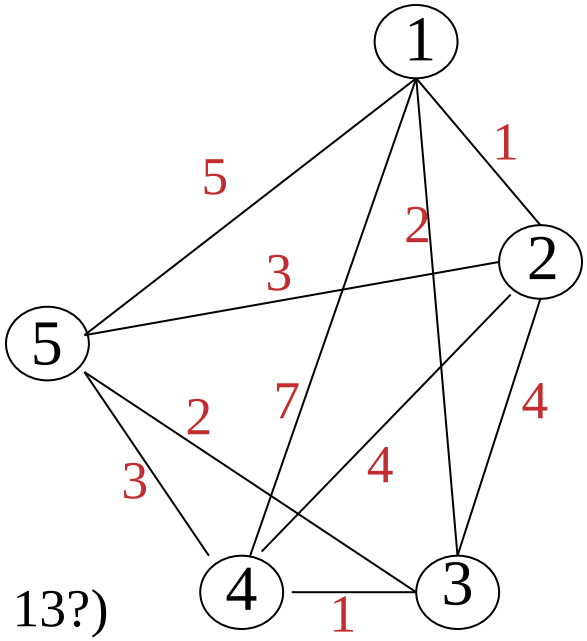


2 – 5 (3)

5 – 3 (2)

3 – 4 (1)

4 – 1 (7) = 14 (MIN = 13?)



Стъпка:

0 (инициализация) LIST = 0; COST = 0, V=1 (нач. възел)

1 (посещение на всички градове) За **i** от 1 до **n**-1
повтаряй стъпка 2;

2 (избор на следващо ребро) (V,W) е ребро с MIN цена, свързващо V с "непосетен" възел W:
LIST=LIST+W;
COST=COST+стойност(V,W);
отбелязване на W като посетен;
V=W;

3 (завършване на маршрута) LIST=LIST+1; (добавяне на нач.град)
OST=COST+стойност(V,1);

4 Край

Сложност на алгоритъма е $O(n^2)$

VII. Проверка на програмата

Проверката е тестване (отстраняване на синтактични и логически грешки, както и изпълнение на контролен пример/примери).

Как се избират данните за тестване?

Зависи от:

- сложността на програмата;
- наличното време;
- персонала и т. н.

- Пълна проверка практически не е възможна!
- Тестването може да служи за доказателство на наличието на грешки, но никога не доказва тяхното отсъствие.
- Надеждността на програмата се осигурява от програмирането, а не от тестването.

(Лингер)

VIII. Съставяне на документацията

1. Външна документация (ръководство за потребителя)
   Включва това, което не се съдържа в самата програма:
   Например: - формално описание на алгоритъма/програмата
   - инструкцията за потребителя
   - контролни входно/изходни данни
   - използваната литература и т. н.
2. Вътрешна (програмна)документация или коментари.

Няколко съвета:

- коментариите трябва да бъдат правилни и информативни;
- пишете коментарите едновременно с програмата;
- използвайте за променливи и функции мнемонични имена;
- не се престаравайте;
- избягвайте нестандартни средства (или ги отбелязвайте с подходящ коментар).