Алгоритми и програмиране - въведение

Проф. д-р Красимир Манев

CSCB029, 20.02.2019

1. Произход понятието алгоритъм

- *Мухаммад ибн-Муса ал Хуаразми (~780-847):*

= Kitāb al-mukhtaṣar fī ḥisāb al-jabr wal-muqābala дава начало на *алгебрата*

= Kitāb al-Jamʿ wat-Tafrīq bi-Ḥisāb al-Hind Съдържа алгоритми за *аритметични операции* с числа представени в *десетична бройна система*

= трактат по астрономия и др.

1. Произход понятието алгоритъм

- В средните векове с понятието ***algorismus*** или ***algorithmus*** са означавали дсетичната бройна система и правилата за извършване на операции в нея.
- В работите на Кр. Лудолф (1525) и Г. В. Лайбниц (1684 т.) добива съвременния си смисъл: *Систематичен изчислителен процес, който с краен брой стъпки решава определена* задача (вж. История на математиката, том 1, изд. Наука и изкуство, София, 1974 т.).

2. Същност на понятието алгоритъм

- Речник на българския език, Издателство на БАН, том I, 1977 г.: *Система от правила, които определят последователност от изчислителни операции, прилагането на които води до решение на дадена задача. Алгоритъм на Евклид.*
- Larousse de la langue Francaise, Lexis, 1979 : *Съвкупност от правила или предписания за получаване, с краен брой операции, на определен резултат.*
- Webster's New Collegiate Dictionary, G.&C. Merriam Company, 1973 : *Процедура за решаване на математическа задача (например, намиране на най-голям общ делител) с краен брой стъпки, която често съдържа повтарящи се операции*.

2. Същност на понятието алгоритъм

- Речник на българския език, Издателство на БАН, том I, 1977 г.: ***Система от правила***, *които определят последователност от изчислителни операции, прилагането на които води до решение на дадена задача. Алгоритъм на Евклид.*
- Larousse de la langue Francaise, Lexis, 1979 : ***Съвкупност от правила*** *или предписания за получаване, с краен брой операции, на определен резултат.*
- Webster's New Collegiate Dictionary, G.&C. Merriam Company, 1973 : *Процедура за решаване на математическа задача (например, намиране на най-голям общ делител) с краен брой стъпки, която често съдържа повтарящи се операции*.

2. Същност на понятието алгоритъм

- Речник на българския език, Издателство на БАН, том I, 1977 г.: *Система от правила, които определят* ***последователност от изчислителни операции****, прилагането на които води до решение на дадена задача. Алгоритъм на Евклид.*
- Larousse de la langue Francaise, Lexis, 1979 : *Съвкупност от правила или* ***предписания*** *за получаване, с краен брой операции, на определен резултат.*
- Webster's New Collegiate Dictionary, G.&C. Merriam Company, 1973 : ***Процедура*** *за решаване на математическа задача (например, намиране на най-голям общ делител) с краен брой стъпки, която често съдържа повтарящи се операции*.

2. Същност на понятието алгоритъм

- Речник на българския език, Издателство на БАН, том I, 1977 г.: *Система от правила, които определят последователност от изчислителни операции, прилагането на които води до* ***решение на дадена задача****. Алгоритъм на Евклид.*
- Larousse de la langue Francaise, Lexis, 1979 : *Съвкупност от правила или предписания за* ***получаване****, с краен брой операции,* ***на определен резултат****.*
- Webster's New Collegiate Dictionary, G.&C. Merriam Company, 1973 : *Процедура за* ***решаване на математическа задача*** *(например, намиране на най-голям общ делител) с краен брой стъпки, която често съдържа повтарящи се операции*.

2. Същност на понятието алгоритъм

- Речник на българския език, Издателство на БАН, том I, 1977 г.: *Система от правила, които определят последователност от изчислителни операции, прилагането на които води до решение на дадена задача. Алгоритъм на Евклид.*
- Larousse de la langue Francaise, Lexis, 1979 : *Съвкупност от правила или предписания за получаване,* ***с краен брой операции****, на определен резултат.*
- Webster's New Collegiate Dictionary, G.&C. Merriam Company, 1973 : *Процедура за решаване на математическа задача (например, намиране на най-голям общ делител)* ***с краен брой стъпки****, която често съдържа повтарящи се операции*.

2. Същност на понятието алгоритъм

- Речник на българския език, Издателство на БАН, том I, 1977 г.: *Система от правила, които определят последователност от изчислителни операции, прилагането на които води до решение на дадена задача.* ***Алгоритъм на Евклид****.*
- Larousse de la langue Francaise, Lexis, 1979 : *Съвкупност от правила или предписания за получаване, с краен брой операции, на определен резултат.*
- Webster's New Collegiate Dictionary, G.&C. Merriam Company, 1973 : *Процедура за решаване на математическа задача (например,* ***намиране на най-голям общ делител****) с краен брой стъпки, която често съдържа повтарящи се операции*.

2. Същност на понятието алгоритъм

- *Задачата за **намиране на най-голям общ делител** на две естествени числа*: Дадени са две естествени числа $0 < b \leq a$. Да се намери НОД на тези две числа, т.е. най-голямото естествено число С, което дели без остатък двете зададени числа,.
- Популярният от хилядолетия ***Алгоритъм на Евклид*** решава задачата за намиране на НОД на две естествени числа, като използва обичайните аритметични операции намиране на частното и остатъка при делене, както и сравняването на числа.

3. Характеристики на алгоритмите

- Говорейки за алгоритми, предполагаме наличието на ***съвкупност от обекти***, с които можем да изпълняваме ***операции***. Най-често това са съвкупности от математически дефинирани обекти и операции с тях. Например, множеството на целите числа с аритметичните операции и операциите за сравняване на цели числа.
- Котато става въпрос за нематематически обекти – заместваме ги с матетатически, т.е. създаваме на ***математически модел***.

3. Характеристики на алгоритмите

- В общ вид една ***задача, която ще решаваме с алгоритъм*** изглежда така: ***Дадени са*** обекти от избраното множество. С използване на наличните операции ***да се намери*** (построи) обект, имащ зададени характеристики и зависещ по определен начин от зададените.
- Такава е задачата за намиране на НОД на две естествени числа
- Задаваните обекти наричаме ***входни данни*** (или ***вход***), а получаваните обекти – ***резултат*** (или изход). В задачата за намиране на НОД входните данни са кои да са две естествени числа, различни от нула, а резултатът е техният НОД.

3. Характеристики на алгоритмите

- Алгоритъмът е ***процедура***, предназначена да реша-ва задача, с прилагане на допустимите операции в сторого определен ред. По зададени входни данни тя трябва да намира искания резултат.
- Процедурата трябва да е ***формална*** - да не пред-извиква никакви съмнения за това, какви операции ще се прилагат и в какъв ред.
- Процедурата трябва да е ***детерминирана*** - който и да я изпълнява, при едни и същи входни данни, трябва да получи един и същ резултат.
- Процедурата трябва да е ***масова*** – да дава искания резултат за прозволни допустими входове
- Алгоритъмът на Евклид е точно такава процедура.

3. Характеристики на алгоритмите

- Алгоритмичните процедури трябва да намират решението на задачата с ***краен брой стъпки***, т.е. с краен брой прилагания на допустимите операции. Този брой може да бъде намерен (или поне оценен "отгоре") предварително, като функция от големината на входа. Този брой по-нататък ще наречем ***сложност на алгоритъма*** (по време).
- Често, за една и съща задача можем да посочим алгоритми със ***значително различаваща се сложност***. Добре е да можем да избираме от няколко налични алгоритъма този с най-малка сложност.
- В математиката се срещат процедури, за които броят на стъпките, за които завършва процедурата не може да бъде определен предварително - ***числени методи***. Няма да се занимаваме с тях в този курс.

3. Характеристики на алгоритмите

- Не случайно, едно от речниковите описания на понятието алгоритъм подчертава, че в алгоритмичните процедури някои последователности от операции ***се изпълняват многократно***.
- В алгоритмиката и програмирането, такива повтарящи се последователности от операции наричаме ***цикли***.
- Много често създаването на алгоритъм се състои в определянето на подходяща последователност от операции и многократното им повтаряне (над евентуално променяща се подмонжество от данни). Може да се каже, че ***организирането на цикли*** е същностна черта на процеса на създаване на алгоритми.

Пример I

- Задача. Дадени са естествените числа $A=a_m a_{m-1} \dots a_0$ и $B=b_n b_{n-1} \dots b_0$, представени в десетична бройна система. Да се сравнят тези две числа по големина.
- Алгоритъм:

1. Ако $m>n$, тогава $A>B$. Край
2. Ако $m<n$, тогава $A<B$. Край
3. $i=m$. Докато $i>0$ и $a_i=b_i$ направи $i=i-1$

4.1. Ако $i<0$, тогава $A=B$. Край

4.2. Ако $a_i>b_i$, тогава $A>B$. Край

4.3. Ако $a_i<b_i$, тогава $A<B$. Край

Пример 2

- Цитат от Аритметичния трактат на ал-Хуарезми

"Ако искаш да получиш сума на две числа, постави ги в два реда, едно под друго и нека бъде разрядът на единиците под разряда на единиците и разрядът на десетиците под разряда на десетиците... Събери всеки разряд със съответния му, който е над него, т.е. единиците с единиците, десетиците с десетиците и т.н. Ако в някой от разрядите се събере поне десет, тогава постави единица в следващия разряд. Например, ако имаш в единиците десет, то сложи единица в десетиците и там тя ще означава десет. Ако от числото е останало нещо или самото то е било по-малко от десет, остави го в същия разряд. Ако нищо не остане, постави в разряда кръгче (!!!), за да не остане разрядът празен и да не приемеш вторият за първи и да се излъжеш в числото си. Точно така ще направиш с всичките разряди ... "

Въпроси и задачи

- За процедурата от Пример 1 определете:

= кои са участващите обекти и кои са използваните операции?

= кои са входните обекти и резултатът?

= има ли в процедурата цикъл и къде?

= колко стъпки най-много ще направи процедурата?

= защо процедурата е детерминирана и защо е масова?

Въпроси и задачи

- За процедурата от Пример 2:

= кои са участващите обекти и кои са използваните операции?

= кои са входните обекти и резултатът?

= каква е връзката между входните обекти и резултатът?

= колко стъпки най-много ще направи процедурата?

5. Въпроси и задачи

- Кои характеристики на алгоритмичните процедури **не притежава** следната процедура: *Посолете нарязани на тънки кръгчета патладжани и оставете доа отделят тъмния сок. Посолете ги, потопете ги в галета или брашно и ги изпържете в силно нарязa мазнина, до златисто оцветяване.*

4. Формални модели

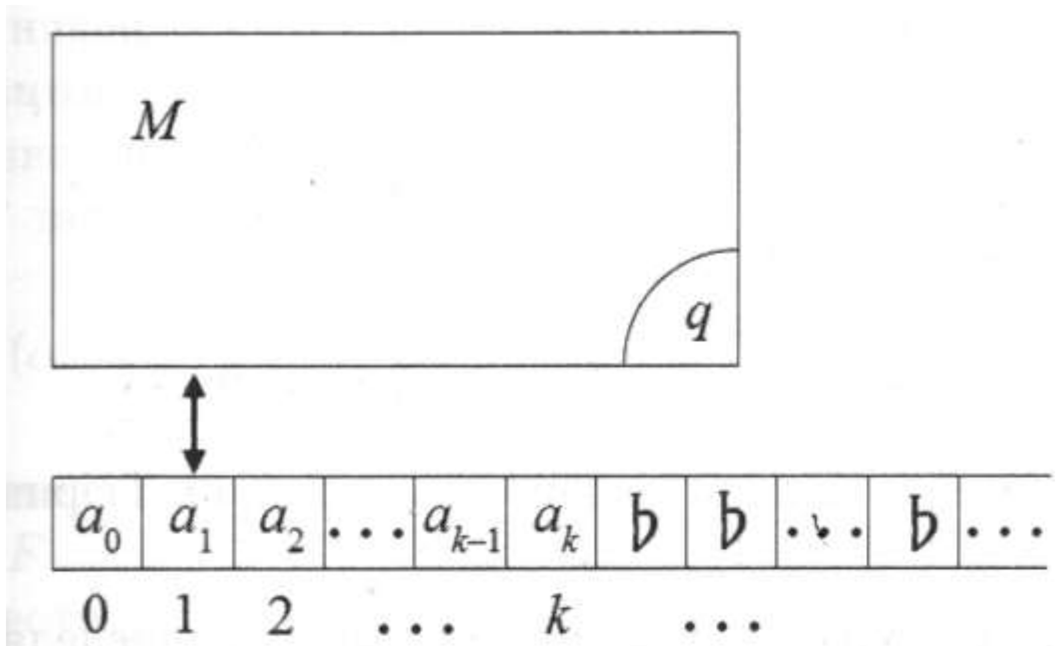
- Както вече видяхме, понятието алгоритъм, както и понятието задача, ***не е формално математическо понятие***, а едно от изискванията за да бъде една процедура алгоритмична е да е формална.
- За да можем да изследваме алгоритмите, е необходимо да ги представяме със средствата на някаква формална, математически дефинирана изчислителна среда – ***формализъм***
- Известни са много различни формализми за представяне на алгоритми – ***машини на Тюринг, машини на Пост, рекурсивни функции, λ-смятане, реален компютър, език за програмиране*** и др.

4. Формални модели

- Много усилия са положили математиците, занимаващи се с формализацията на неформалното понятие алгоритъм за да сравняват мощтта на различните формализми
- ***Тезис на Тюринг-Чърч***: Всички известни до момента формализми (без да броим квантовия компютър) за понятието алгоритъм са ***еквивалентни*** по възможности, т.е. могат да решават едни и същи задачи
- Не всяка задача може да се реши алгоритмично – ***алгоритмично неразрешими задачи***!

4. Формални модели. МТ

- Машината на Тюринг е най-простият изчислителен модел:



- Буквите на лентата са от азбуката X, а състоянията от множеството Q. Състояние-то q_0 е начално, а думата $a_0, a_1, \ldots, a_k$ лентова дума. Заключителни състояния.

4.1. Формални модели. МТ

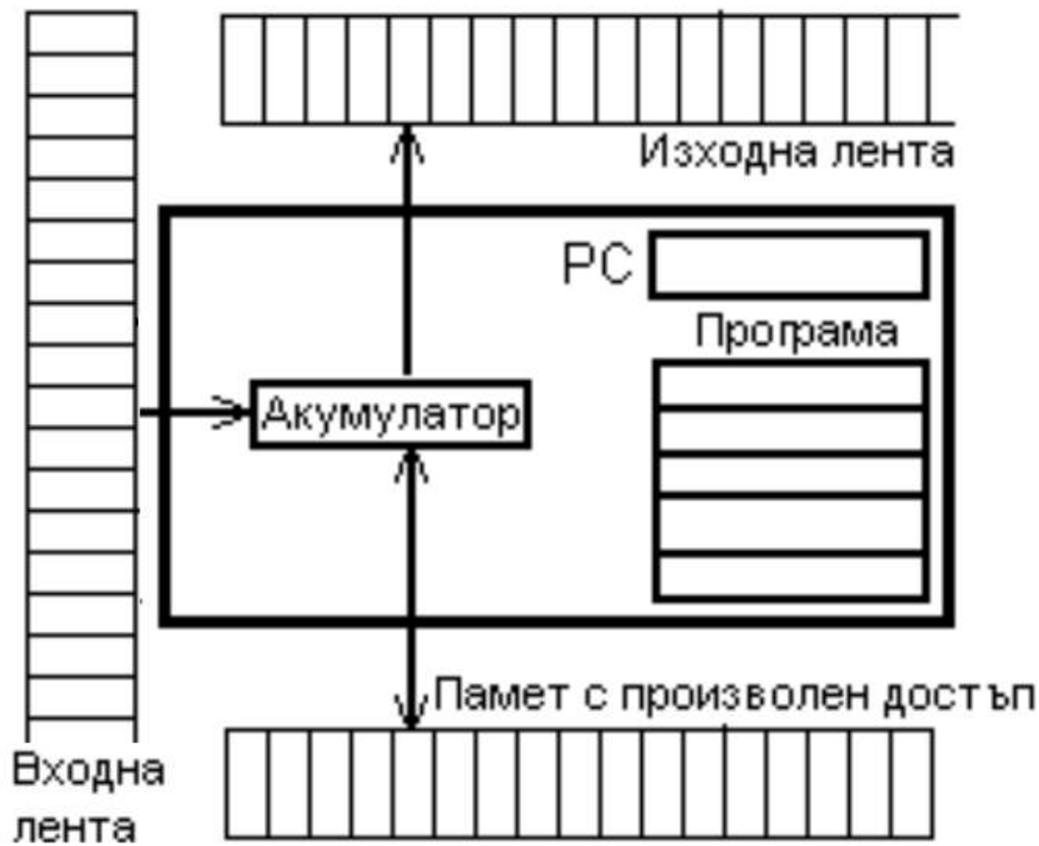
- Ако в началото на лентата имаме думата α, МТ $\mathscr{M}$ стартирайки в състояние q_0 спре в заключително състояние при работа над α и остави на лентата думата β_α , казваме че $\mathscr{M}$ е ***пресметнала*** β_α
- Функцията $f_{\mathscr{M}}(\alpha)= \beta_{\alpha}$, за всяко α, за което $\mathscr{M}$ спира и неопределена, ако $\mathscr{M}$ спре аварийно или не завърши, наричаме ***изчислима по Тюринг***, а процесът на пресмятане – ***изчисляване на функция***

4.1. Формални модели. МТ

- Нека $\mathcal{M}$ има две заключителни състояния - $q_{\text{да}}$ и $q_{\text{не}}$. Ако в началото на лентата имаме думата α, МТ $\mathcal{M}$ стартирайки в състояние q_0 спре в $q_{\text{да}}$ при работа над α, казваме че $\mathcal{M}$ е ***разпознала*** α, а иначе – че не е разпознала α
- Езикът $L_{\mathcal{M}} = \{\alpha | \mathcal{M}$ разпознава $\alpha\}$, наричаме език ***разпознаван от Машина на Тюринг***, а процесът на пресмятане – ***разпознаване на език***

4.2. Формални модели. МПД

- Машината с произволен достъп до паметта (МПД) е упростен модел на съвременните компютри:



4.2. Формални модели. МПД

- Всяка команда има свой ***код*** и може да има или да няма ***аргумент***.

  адрес) код [аргумент]

- Команди с пряка адресация

```
1000) ADD 245
```

- Команди с непосредствен аргумент

```
1000) ADD# 245
```

- Команди с непряка адресация

```
1000) ADD@ 245
```

4.2 Формални модели. МПД

Команда	Действие
LOAD# I	<AC> := I; <PC> := <PC> + 1
LOAD A	<AC> := <A>; <PC> := <PC> + 1
LOAD@ A	<AC> := <<A>>; <PC> := <PC> + 1
STORE A	<A> := <AC>; <PC> := <PC> + 1
STORE@ A	<<A>> := <AC>; <PC> := <PC> + 1
ADD# I	<AC> := <AC> + I; <PC> := <PC> + 1
ADD A	<AC> := <AC> + <A>; <PC> := <PC> + 1
ADD@ A	<AC> := <AC> + <<A>>; <PC> := <PC> + 1
SUB# I	<AC> := <AC> - I; <PC> := <PC> + 1
SUB A	<AC> := <AC> - <A>; <PC> := <PC> + 1
SUB@ A	<AC> := <AC> - <<A>>; <PC> := <PC> + 1
MUL# I	<AC> := <AC> * I; <PC> := <PC> + 1
MUL A	<AC> := <AC> * <A>; <PC> := <PC> + 1
MUL@ A	<AC> := <AC> * <<A>>; <PC> := <PC> + 1
DIV# I	<AC> := <AC> / I; <PC> := <PC> + 1
DIV A	<AC> := <AC> / <A>; <PC> := <PC> + 1
DIV@ A	<AC> := <AC> / <<A>>; <PC> := <PC> + 1
MOD# I	<AC> := <AC> % I; <PC> := <PC> + 1

4.2 Формални модели. МПД

MOD A	<AC> := <AC> % <A>; <PC> := <PC> + 1
MOD@ A	<AC> := <AC> % <<A>>; <PC> := <PC> + 1
JMP B	<PC> := B
JMPZ B	Ако <AC> = 0, то <PC> := B, иначе <PC> := <PC> + 1
JMPP B	Ако <AC> > 0, то <PC> := B, иначе <PC> := <PC> + 1
JMPN B	Ако <AC> < 0, то <PC> := B, иначе <PC> := <PC> + 1
INPUT	Поредната клетка на входната лента се прочита в AC, главата се мести на следваща клетка
INPUT	<AC> се записва в поредна клетка на изходната лента, а главата се мести на следваща клетка
STOP	Прекратява изпълнението на програмата

Пример

- Като пример нека напишем програма за МПД, която въвежда от входната лента кое-фициентите a, b и c на квадратно уравнение, пресмята дискриминантата D на уравнението по формулата b^2-$4ac$ и извежда получения резултат на изходната лента.
- Първата работа при решаването на задача с МПД е да определим къде ще поставим даните

  0) за коефициента a

  1) за коефициента b

  2) за коефициента c

  3) за дискриминантата D

  4) за междинен резултат

Пример

- Ето програмата която решава задачата

```
 0) INPUT      // въвеждаме a в AC
 1) STORE 0    // съхраняваме a в паметта
 2) INPUT      // въвеждаме b в AC
 3) STORE 1    // съхраняваме b в паметта
 4) INPUT      // въвеждаме c в AC
 5) STORE 2    // съхраняваме c в паметта
 6) LOAD# 4    // поставяме 4 в AC
 7) MUL 0      // умножаваме AC по a
 8) MUL 2      // умножаваме AC по c
 9) STORE 4    // запазваме 4ac в паметта
10) LOAD 1     // поставяме b в AC
11) MUL 1      // умножаваме AC по b
12) SUB 4      // изваждаме 4ac от AC
13) STORE 3    // запазваме резултата
14) OUTPUT     // извеждаме резултата
15) STOP       // прекратяваме изпълнението
```

Задача 1

- Какво ще направи програмата, която е показана вдясно?

```
 0) LOAD# 1
 1) STORE 0
 2) INPUT
 3) JMPZ 9
 4) STORE@ 0
 5) LOAD 0
 6) ADD# 1
 7) STORE 0
 8) JMP 2
 9) LOAD 0
10) SUB# 1
11) STORE 0
12) OUTPUT
13) STOP
```

Задача 2

- Напишете програма за МПД , която въвежда от лентата дължините на двете страни на правоъгълник, пресмята и извежда на изходната лента периметъра и лицето на правоъгълника.

4.3.

- Език за програмиране

Не се нуждае от дефиниране, а не е и лесно!

5. Сложност на алгоритми

- Вече споменахме, че някои задачи ***са алгоритмично неразрешими***.
- Когато една математическа задача е алгоритмически неразрешима – ***интересът към нея спада***, но с използването на компютър решаването на ***отделни случаи*** на алгоритмично неразрешими задачи, стана ***възможно***, но става бавно
- За съжаление, дори с компютър, решаването много алгоритмично разрешими задачи остава ***проблематично бавно***

5. Сложност на алгоритми . Нуждата от оценка на сложността

- Разгледайте подзаглавието на слайда. То е съставено от 27 букви

  аааааdeжжклннннооооссттттуц

- Задача: да се възстанови заглавието от зададения списък с букви
- Задачата е алгоритмически разрешима – трябва да се разгледат всички пермутации 27!=10888869450418352160768000000

  Ако компютър преглежда 1000000 пермутации/сек -> ще реши задачата за 345283785211135 години

5. Сложност на алгоритми . Нуждата от оценка на сложността

- Друг пример: космически апарат пътува към Марс и изпраща данни. За да пристигнат данните без загуби от електромагнитния „шум“ на Космоса - система за защита.
- Бордовият компютър разполага с голяма, но ограничена ОП. Когато апаратът се отдалечава, системата за шумозащита се окозва слаба и трябва да се смени
- Проблем: новата система изисква памет, каквато на борда няма

5. Сложност на алгоритми

- При работата си всяка програма използва два ресурса – ***време*** и ***памет***
- Ресурсът памет често, но не винаги, може да бъде разширен, докато ресурсът време – не може
- Сложността на алгоритъма по време и памет е механизмът за оценка доколко програмата имплементираща алгоритъма е ПРИЕМЛИВА ПО ОТНОШЕНИЕ НА НАЛИЧНИТЕ РЕСУРСИ

5. Сложност на алгоритъм

- Формализъм за представяне на алгоритми – ***МПД***
- Дадена е ***задача*** π, за която се търси алгоритмично решение
- ***Вход*** - редица от цели, поставени на входната лента; броят им n – ***размер на входа***
- Известен ни е алгоритъм А, т.е. програма за МПД, която решава π

5. Сложност на алгоритъм (по време)

Сложност на алгоритъма A по време в ***най-лошия случай***:

$T_A(n)=\max_{\forall\alpha}${брой изпълнени команди от A върху вход α с размер *n*}

Сложност на алгоритъма A по време в ***средния случай***:

$t_A(n)=\sum_{\forall\alpha}${брой изпълнени команди от A върху вход α с размер *n*}/ {брой входове с размер *n*}

5. Сложностна алгоритъм по памет

Сложност на алгоритъма А по памет в ***най-лошия случай***:

$S_A(n)=\max_{\forall\alpha}$\{брой използвани клетки памет от А върху вход α с размер *n*\}

Сложност на алгоритъма А по памет в ***средния случай***:

$s_A(n)=\sum_{\alpha}$\{брой използвани клетки памет от А върху вход α с размер *n*\}/\{брой входове с размер *n*\}

Пример

0) LOAD# 1	1 път
1) STORE 0	1 път
2) INPUT	N+1 пъти
3) JMPZ 9	N+1 пъти
4) STORE@ 0	N пъти
5) LOAD 0	N пъти
6) ADD# 1	N пъти
7) STORE 0	N пъти
8) JMP 2	N пъти
9) LOAD 0	1 път
10) SUB# 1	1 път
11) STORE 0	1 път
12) OUTPUT	1 път
13) STOP	1 път

$T(N) = 7N+9$

$t(N) = T(N)$, защото всички случаи са еднакво лоши

Пример

```
 0) LOAD# 1
 1) STORE 0
 2) INPUT
 3) JMPZ 9
 4) STORE@ 0
 5) LOAD 0
 6) ADD# 1
 7) STORE 0
 8) JMP 2
 9) LOAD 0
10) SUB# 1
11) STORE 0
12) OUTPUT
13) STOP
```

$S(N) = s(N) = N+1$

защото всички случаи
са еднакво лоши

6. Алгоритмично трудни задачи

- Когато имаме няколко алгоритъма за една задача, добре е да изберем този с най-малка сложност по време. Оценяването на ***сложността на алгоритъм*** по време е сравнително лесно вече.
- Когото решим да дефинираме ***сложност на задача*** срещаме проблем – можем ли да определим алгоритъма с минимална сложност за да го обявим за сложност на задачата по време?

6. Алгоритмично трудни задачи

- Ще разглеждаме само задачи във формата разпознаване на език, като задачата Хамилтонов цикъл.
- Задачите, които не са в такъв вид упростяваме до разпознаване на език. Например „Намерете най-голямото N, токава че $\pi(x)$=true“ става „Съществува ли $N > B$ такова, че $\pi(x)$=true за някаква константа B?“

6. Алгоритмично трудни задачи

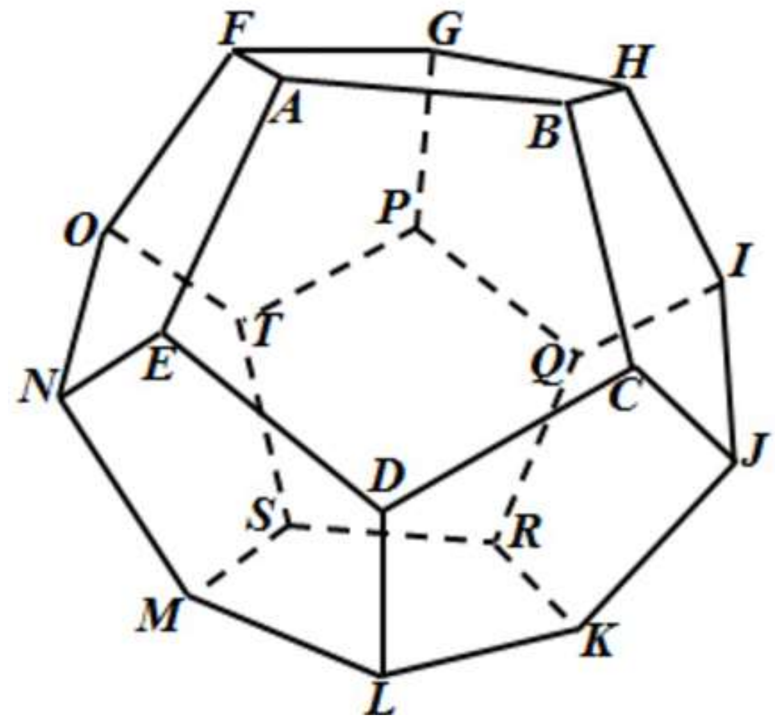
Сър Уйлям Хамилтон (1805 – 1865)

Формулира задача много подобна на задачата на Ойлер Euler – *Хамилтонов цикъл*, смятана за втората задача в Теорията на графите.

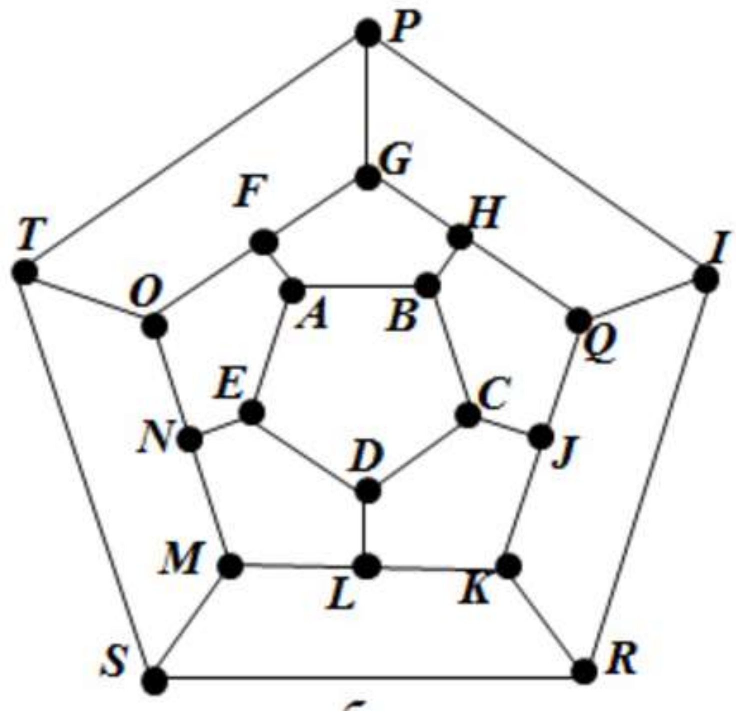
6. Алгоритмично трудни задачи

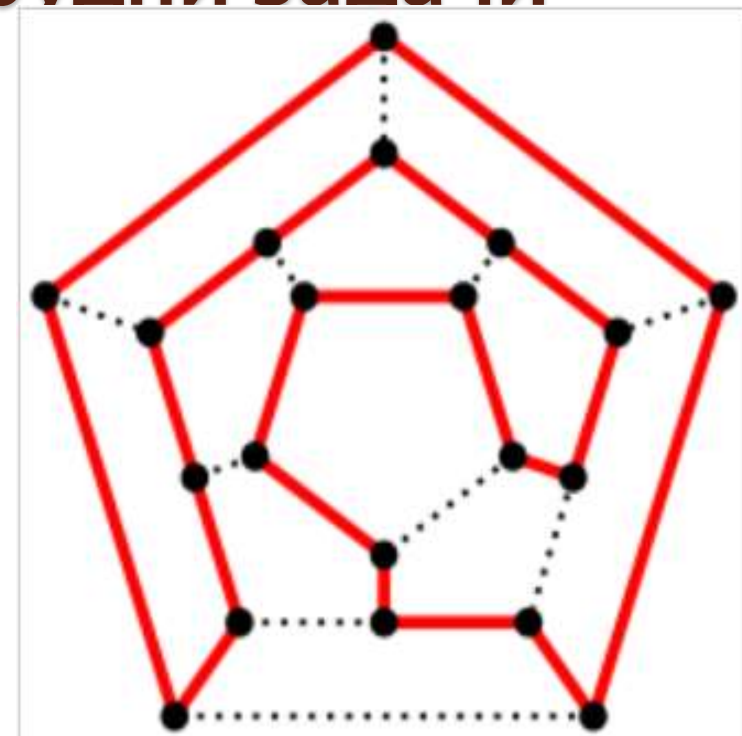
Додекаедър: изпъкнало тяло с 20 върха, 30 ребра и 12 страни с форма на правилни петоъгълници.

Задача: Възможно ли е да тръгнем от връх на додекаедъра, да вървим само по ребра и да се върнем в същия връх, след като минем **по всички върхове точно един път – Хамилтонов цикъл**.



6. Алгоритмично трудни задачи





За додекаедъра, обхождането е възможно. Има много такива обхождания. В общия случай обаче има ли ХЦ в граф, можем да проверим засега само с пълно изчерпване – $O(n.n!)$.

6. Алгоритмично трудни задачи

- Задачи за които съществува поне един алгоритъм с полиномиална или суб-полиномиална сложност ($O(n^d)$ или $O(n^d \log^d n)$) наричаме полиномиално разрешими
- Множеството от полиномиално разрешимите задачи означаваме с $\mathscr{P}$.
- Много такива задачи се изучават в курсовете по алгоритми

6. Алгоритмично трудни задачи

- Задача за която не ни е известен полиномиален алгоритъм, но по зададен **проект за решение** можем полиномиално да проверим дали това е така, наричаме ***полиномиално проверима***
- Множеството от полиномиално проверимите задачи означаваме с $\mathcal{NP}$.
- Задачата Хамилтонов цикъл е полиминално проверима

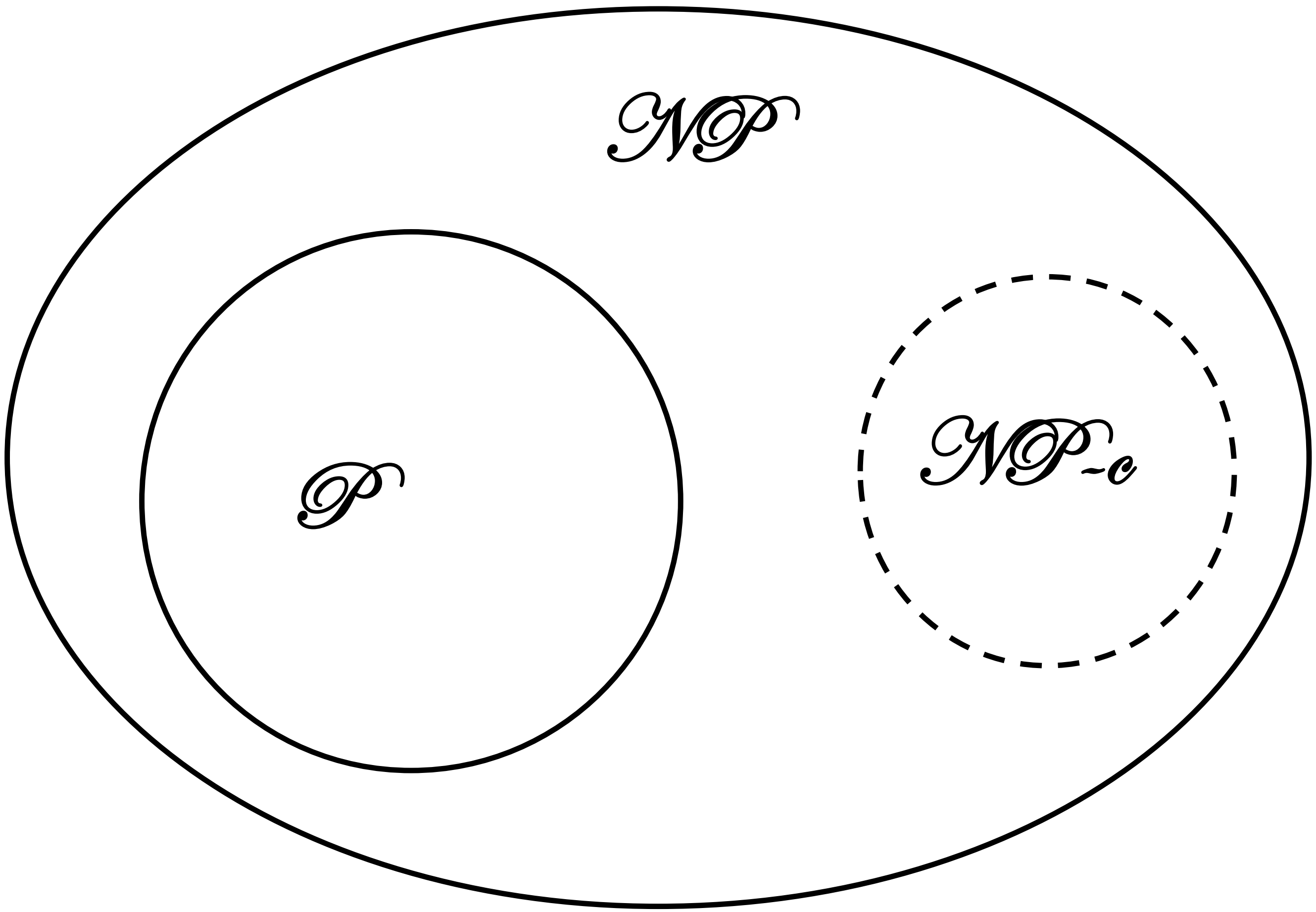
6. Алгоритмично трудни задачи

- Stephen A. Cook (1939 -) е канадски математик-информатик, който въвежда понятието полиномиална сводимост дало тласъкв развитието на теорията

6. Алгоритмично трудни задачи

- Задачата за разпознаване на език L се ***свежда полиномиално*** към задачата за за разпознаване на език L' ако съществува полиномиален алгоритъм който трансформира $\forall\alpha\in$ L в $\alpha'\in$ L', а $\forall\beta\notin$ L в $\beta'\notin$ L'
- Задача π която е от $\mathscr{NP}$ и всяка друга която е от $\mathscr{NP}$ се свежда полиномиално към π наричаме $\mathscr{NP}$–пълна, и означаваме множеството от тези задачи с $\mathscr{NP}$–c
- Теорема: $\mathscr{P}\subseteq\mathscr{NP}$
- Огромен проблем: $\mathscr{P}$ = $\mathscr{NP}$?

6. Алгоритмично трудни задачи



6. Алгоритмично трудни задачи

- Кардинален въпрос на теорията е **съществуват ли въобще $\mathcal{NP}$–пълни задачи**. Ако не, то $\mathcal{P} = \mathcal{NP}$.
- $x_1^{\sigma 1} \vee x_2^{\sigma 2} \vee \ldots \vee x_n^{\sigma n}$ – ***елементарна дизюнкция*** - $x_1^{1} = x_1, x_1^{0} = \neg x_1$
- $f(x_1, x_2, \ldots, x_n) = D_1 \wedge D_2 \wedge \ldots \wedge D_k$ – ***конюнктивна нормална форма на БФ***
- Задача (Удовлетворимост на КНФ) Дадена е КНФ. Съществува ли вектор, за който формата има стойност **0**?

6. Алгоритмично трудни задачи

- Теорема (St. Cook, 1970) Задачата Удовлетворимост на КНФ е $\mathcal{NP}$–пълна.
- Доказателство. 1. Задачата очевидно е от $\mathcal{NP}$, защото по зададен вектор-предположение за литейно време се проверява дали нулира КНФ
- 2. Доказателството, че всяка $\mathcal{NP}$–задача се свежда полиномиално към Удовлетворимост на КНФ е гениално

6. Алгоритмично трудни задачи

St. Cook създава уникален алгоритъм който за полиномиално време трансформира работата на работеща за полиномиално време МТ в КНФ. Така всяка $\mathscr{NP}$–задача се свежда полиномиално към Удовлетворимост на КНФ

Следствие. Всички $\mathscr{NP}$–с задачи са еквивалентни – или за всички има $\mathscr{P}$ алгоритъм или за нито една няма!!!