Въведение в изчисленията и алгоритмите

1. Изчисляващи устройства и алгоритми.

1.1 Изчисляващи устройства, машина на Тюринг.
Примери на прости програми. Решими и нерешими задачи.
Функция S(n). Неизчислимост на S(n).

1.2 Сводимост. Определение. Няколко доказателства на сводимост.
Представяне (кодиране) на данни.

1.3 Още доказателства на сводимост. Тезис на Чърч.
Дефиниция на алгоритъм. Следствия.

1.4 Универсална машина на Тюринг.
Други алгоритмични техники (цикъл, fork и пр.).

1.5 Конструиране на самопораждаща се програма (Quine).
Лема за разгъването { P(Q_P)=Q_P() }.
Нерешимост на stop-задачата.

2. Зависимости между алгоритмични и математически задачи.

2.1 Трудни математически задачи (Goldbach, Ferma, Colatz).
Връзка с проблема за изчисляване на S(n) и stop-задачата.
Формални математически теории. Проверимост на доказателствата чрез алгоритъм.

2.2 Изчислимост на доказателствата. Теорема на Гьодел за непълнота.
Коментар на нашите възможности за изследване на числата.
Принцип на рефлексията.

2.3 Ефективни и полу-ефективни множества. Времева сложност.
Неограничена сложност (непрактичност) на граничните алгоритми (дефиниращи полу-ефективни множества). Съществуване на неограничено сложни математически задачи.

2.4 Диагонален процес - история, приложение, граница между хаос и ред, парадоксална природа на познанието, ограничения на научното изследване ...

3. Сложност на алгоритмите:

3.1 Лесни и трудни задачи, полиномиални и експоненциални алгоритми.

3.2 Определение на класовете **P** и **NP**.

3.3 Полиномиална сводимост. **NP**-пълни задачи. Теорема на Кук.

3.4 Още **NP**-пълни задачи.

3.5 Структурни различия между лесните и трудни задачи.

4. Основни полиномиални алгоритми:

4.1 Алгоритми за функции над естествени числа.

4.2 Сортировка и търсене.

4.3 Алгоритми върху графи.

5. Методология

5.1 Някои принципи на този курс.

5.2 Използвана литература.

1.1a Изчисляващи устройства, машина на Тюринг. Дефиниция и примери на прости програми.

Машина на Тюринг е въображаем компютър, описан от английският математик **Алън Тюринг** през 1936 г.

Работата на Тюринг е първото точно определение на понятието алгоритъм (наричано още механична, формална или ефективна процедура). Използва се при доказването на основни резултати в компютърните науки, най-вече в областите изчислимост и сложност на алгоритмите, както и в математическата логика.

1.1a.1 Определение

Машината на Тюринг се състои от четири компонента:

1. Памет - потенциално безкрайна лента, състояща се от клетки, във всяка от които е записан символ от някаква крайна азбука. Азбуката съдържа специален празен символ (обикновено обозначаван с '0') и един или повече други символи. Във всеки момент от работата на машината лентата е крайна, но при нужда може да и залепяме отляво или отдясно нови клетки, съдържащи празния символ.
2. Глава, която във всеки момент от изчислението се намира над определна клетка от лентата. При всеки такт главата прочита символа от клетката над която се намира, записва нов символ и се премества наляво или надясно по лентата в зависимост от изпълняваната инструкция и прочетения символ.
3. Програма - краен списък от инструкции, който за разлика от съвременните компютри е отделен от паметта. Всяка инструкция е поредица от указания какво да се направи, ако главата е прочела i-тата буква от азбуката. Всяко указание съдържа информация какъв символ да се запише обратно върху лентата, коя инструкция ще се изпълнява на следващата стъпка и накъде (наляво или надясно) да се премести главата.
4. Регистър съдържащ номера на активната в момента инструкция (програмен брояч). Една от инструкциите се приема за начална, т.е. изчислението започва със зареждането на номера й в програмния брояч. Има и крайна инструкция - при достигането й изчислението спира.

Съвкупността (множеството) от всички машини на Тюринг ще обозначаваме с **TM**.

Ще разгледаме няколко прости машини на Тюринг. Програмата можем да изписваме по два начина. Първият е таблица, в която всеки ред съответства на една инструкция, като за всеки символ от азбуката е изписано съответното указание за действие на машината. Вторият е линеен - всеки ред съответства на указание, предшествано от име на инструкция и символ от азбуката.

1.1a.2 Пример **Double:**

Ще разгледаме машина, работеща с двубуквена азбука - буквите са 0 и 1. Ще наречем нашата машина **Double**. Ето списъка от инструкциите й:

Изписване в таблична форма:

```
A       0:(0,s,R)       1:(0,B,R)
B       0:(1,C,L)       1:(1,B,R)
C       0:(1,A,L)       1:(1,C,L)
```

Изписване в линейна форма:

```
A,0: 0,s,R
A,1: 0,B,R
```

```
B,0: 1,C,L
B,1: 1,B,R
C,0: 1,A,L
C,1: 1,C,L
```

Имената на инструкциите са големите латински букви A, B и C. С малка буква s сме означили специалната инструкция за спиране на изчислението.

Всяка инструкция съдържа 2 указания – какво да прави при прочетена 0 или 1. Например инструкцията B при прочетена 0 трябва да изпълни указанието (1,C,L), чието тълкувание е: запиши върху лентата 1, следващата активна инструкция ще е C, премести главата наляво.

Предназначението на тази проста машина е да удвоява поредица от единици, при следните условия: в началото лентата съдържа поредица от няколко единици, а всички останали клетки са празни (т.е. заети от символа 0) и главата се намира над най-дясната единица и началната активна инструкция е A. Започвайки изчислението, машината в крайна сметка ще спре, като броят на единиците в поредицата ще е два пъти по-голям от началния им брой.

Нека в началото на изчислението лентата съдържа 3 поредни единици. Ето цялото изчисление, стъпка по стъпка (с курсив е отбелязан символа, над който се намира главата):

стъпка	инстр.	лента
1	A	011**1**0000
2	B	0110**0**000
3	C	011**0**1000
4	A	01**1**11000
5	B	010**1**1000
6	B	0101**1**000
7	B	01011**0**00
8	C	0101**1**100
9	C	010**1**1100
10	C	01**0**11100
11	A	0**1**111100
12	B	00**1**111100
13	B	001**1**1100
14	B	0011**1**100
15	B	00111**1**00
16	B	001111**0**0
17	C	00111**1**10
18	C	0011**1**110
19	C	001**1**1110
20	C	00**1**11110
21	C	0**0**111110
22	A	**0**1111110
23	stop	0**1**111110

Изобщо, ако началната поредица съдържа ***k*** единици, машината **Double** ще спре след изпълнението на $2k^2+k+1$ стъпки (без да броим инструкцията за спиране), на лентата ще има ***2k*** единици и главата ще е над най-лявата единица. Тези свойства на **Double** могат да бъдат доказани с използване на математическа индукция.

Можем да модифицираме малко нашата машина, така че тя да върши същата работа, но в началният момент главата да се намира върху някоя от нулите, намиращи се надясно от поредицата единици. Да кръстим тази машина **DoubleR**. Ето кода ѝ:

```
F       0:(0,F,L)       1:(0,B,R)
A       0:(0,s,R)       1:(0,B,R)
B       0:(1,C,L)       1:(1,B,R)
C       0:(1,A,L)       1:(1,C,L)
```

Началната инструкция за тази машина е F. Ще използваме машината **DoubleR** активно в следващата статия. Можем да направим подобни модификации ако искаме в началото главата да се намира върху нула отляво на единиците, върху коя да е единица или в най- сложния случай на неопределена позиция (опитайте се за упражнение да разрешите този последен случай).

1.1a.3 Пример **Clear**

И тази машина работи с двубуквена азбука. Тя има единствена инструкция, която записана в таблична форма изглежда така:

```
A       0:(0,s,R)       1:(0,A,R)
```

Предназначението на тази проста машина е да изтрие поредица от единици, при следните условия: в началото лентата съдържа поредица от няколко единици, а всички останали клетки са празни (т.е. заети от символа 0), главата се намира над най-лявата единица и началната активна инструкция е A. Започвайки изчислението, машината на всеки такт ще нулира по една единица, докато ги изтрие всичките. Лесно се вижда, че тя ще направи точно толкова стъпки, колкото е дължината на поредицата от единици.

1.1a.4 Пример **Const_k**

Тази машина изписва върху празна лента **k** поредни единици. Линейната форма на машината **Const_k** е:

```
1,0:    1,2,R
2,0:    1,3,R
3,0:    1,4,R
....
k,0:    1,stop,R
```

Забележете, че дължината на тази програма зависи от **k**, т.е. числото **k** трябва да е известно предварително (преди написването на програмата).

1.1a.5 Задачи за упражнение:

1. Напишете програма за машина на Тюринг, наречена **Sub**, която да започва работа върху лента, на която са записани две числа **m** и **n** като поредици от съответно **m** единици и **n** единици, разделени с една нула. Главата се намира върху разделящата нула. Целта е в края на работата си машината да замени двойката числа с разликата им, т.е. да изпише число **k=m-n** ако **m>=n** или **k=n-m** ако **m<n.**

2*. Напишете програма за машина на Тюринг, наречена **MCD**, която започва при същите условия като горната и трябва да получи най-големият общ делител на числата **m** и **n** (задачата е аналог на известният от древността алгоритъм на Евклид).

1.1b Нерешими задачи. Функция S(n). Неизчислимост на S(n).

В тази глава ще обсъждаме изключително машини на Тюринг, които ползват азбука от два символа. Множеството от всички такива машини ще означаваме с $\mathbf{TM_2}$.

1.1b.1 Решими и нерешими задачи.

*Определение на **решима** задача:*

Решима ще наричаме всяка задача, за която можем да създадем *машина на Тюринг*, която получавайки условието на задачата като входни данни приключва работата си след като е пресметнала отговора на задачата и го е изписала върху лентата.

В 1.1a написахме няколко програми, решаващи прости задачи – за изписване на предварително известно естествено число K като поредица от единици, за намаляване на число N докато стане равно на 0, задачата за удвояване на естествено число N.

*Определение на **нерешима** задача:*

Нерешима ще наричаме всяка задача, която не е ***решима***, т.е. за която не съществува машина на Тюринг, която да я решава.

1.1b.2 Функция S(n).

*Определение на функцията **S(n):***

Нека е дадено естественото число **n**. Да разгледаме всички машини на Тюринг, чиято програма се състои от точно **n** реда и азбуката на лентата има точно 2 символа. Интересува ни поведението на тези машини, ако ги стартираме при нулево начално състояние на лентата. Някои от тях при работата си ще се зациклят и ще работят безкрайно, други ще спрат. Да изберем от всички спиращи машини тази, която работи най-дълго, т.е. спира след най-много елементарни стъпки. Да означим броят стъпки, които тази машина прави с ***S(n)***.

Определението на ***S(n)*** изглежда напълно законно от математическа гледна точка, тъй като броят на машините от $\mathbf{TM_2}$ с **n** реда програмен код е очевидно краен. От този краен брой машини трябва да подберем тези, които започвайки работа върху чиста лента спират и да изчислим максимума на направените елементарни стъпки.

Всъщност ще се убедим, че тази наглед проста задача не може да бъде решена с никакви изчислителни устройства. В тази статия обаче ще докажем само, че ***S(n)*** не може да бъде пресметната от никаква *машина на Тюринг* с двубуквена азбука.

1.1b.3 Неизчислимост на S(n).

*Теорема **S_no_in_TM₂:***

Изчисляването на функцията ***S(n)*** е нерешима задача, т.е. не съществува машина от $\boldsymbol{TM_2}$, която да получава като входни данни на лентата си естествено число **n** и да приключва работата си нормално, изписвайки на лентата числото ***S(n)***.

***Идея** за доказателство:*

Допускайки че ***S(n)*** може да се изчисли, ще построим машина на Тюринг с $\boldsymbol{N_0}$ състояния (команди), която започвайки работата си при чиста лента прави следното:

1. Изписва върху лентата числото $\boldsymbol{N_0}$.
2. Изчислява $\boldsymbol{S(N_0)}$.
3. Прави поне още $\boldsymbol{S(N_0)}$ стъпки и спира.

Така конструираната машина ще прави строго повече стъпки от $\boldsymbol{S(N_0)}$, а има точно $\boldsymbol{N_0}$ реда, което противоречи на определението на ***S(n)***.

Доказателство:

Да допуснем, че съществува машина от **TM_2**, изчисляваща S(n). Нека кръстим тази машина S_tm_2. Ще предполагаме, че при започване на работата си тази машина очаква върху лентата да има n на брой поредни единици и главата се намира над най-лявата от единиците. Ще предполагаме също, че след приключване на работата си машината S_tm_2 е оставила на лентата точно S(n) поредни единици и главата отново се намира над най-лявата единица. Може тези изисквания към входа и изхода на машината S_tm_2 да ни изглеждат прекалено претенциозни, но в следващата лекция ще направим коментар, който да разсее съмненията ни.

Ще модифицираме S_tm_2 няколко пъти, като всеки път ще добавяме нови парчета код към нея отпред или отзад . Когато от две машини m_1 и m_2 сглобяваме нова, която трябва да стартира m_1, а върху резултата от работата и да приложи m_2, получената нова машина ще означаваме с $m_1|m_2$. Когато извършваме подобна сглобка, се налага промяна на номерата на инструкциите на втората машина m_2, но това е сравнително лесен технически проблем.

Да сглобим S_tm_2 с парче, което ще сложим отзад и ще наречем Clear и което намаля числото, което получава като входни данни с единица и прави това в цикъл дотогава, докато резултатът стане 0, след което машината спира.

Отпред пък да сложим фрагмент DoubleR, идентичен на едноименния пример от 1.1a.2, който удвоява полученото на входа си число. Нека машината $DoubleR|S\_tm_2|Clear$ има n_1 състояния.

Да конструираме сега машина с име $Const\_n_1$, която започвайки от празна лента, изписва на нея линейно число, състоящо се от n_1 единици. За изписването на всяка единица е нужен 1 ред програма, т.е. $Const\_n_1$ ще има n_1 състояния.

Нека вземем числото N_0 да е равно на $2n_1$.

Да означим с S_bad машината $Const\_n_1|DoubleR|S\_tm_2|Clear$. S_bad има точно $2n_1$, т.е. N_0 състояния.

Да проследим поведението на S_bad, като я стартираме при празна лента:

1. Тя ще изпише върху лентата половината си дължината n_1 в линейно представяне (фрагмент $Const\_n_1$).
2. Ще удвои n_1, получавайки на лентата числото N_0 (фрагмент DoubleR).
3. Ще изчисли $S(N_0)$ (фрагмент S_tm_2).
4. Ще направи още точно $S(N_0)$ стъпки (фрагмент Clear).
5. Ще спре.

От конструкцията и поведението на S_bad следват твърденията:
(1) S_bad започвайки работата си върху празна лента спира.
(2) S_bad има точно N_0 реда.
(3) S_bad прави строго повече от $S(N_0)$ стъпки.
(4) От определението на S(n), **(1)** и **(2)** следва, че S_bad прави най-много $S(N_0)$ стъпки.
(3) и **(4)** си противоречат, което доказва теоремата !

1.1b.4 ***Коментар:***

Може да ни се стори, че невъзможността за пресмятане на S(n) се дължи на простата конструкция на машините на Тюринг. Ако обаче разгледаме по-детайлно горното доказателство, ще забележим че то не поставя никакви изисквания за простота на конструкцията на изчисляващата машина.

Единствените конкретни изисквания на доказателството са за фрагментите Const_n_1, DoubleR и Clear, т.е. от изчисляващото устройство се иска възможност за:

- конструиране на големи числови константи с относително кратка програма, така както фрагментите Const_n_1 и DoubleR конструират числото N_0.
- възможност за бавно работеща и спираща програма с фиксиран брой редове, каквато е Clear.

Да си представим друг модел машини за изчисление. Да наречем класа от тези машини M, а аналогичната лоша функция $S_M(n)$. Подобна на горедоказаната теорема:

Теорема ***S_M_no_in_M:***

Функцията $S_M(n)$ не може да бъде изчислена с машина от класа M.

ще е вярна, стига да можем да програмираме в M аналози на Const_n_1, Double и Clear, т.е. класа M трябва да осигурява някакви минимални възможности за работа с числа. Можем да надарим M с колкото искаме мощни изчислителни способности, без да променим верността на теоремата за неизчислимост на $S_M(n)$.

1.1b.5 ***Задачи*** за упражнение:

Определение на функцията **Σ(n):**

Да означим със **Σ(n)** максималният брой единици, които може да изпише машина на Тюринг с n инструкции и азбука от 2 букви върху празна в начално положение лента и след това да спре.

Задача 1**:**

Докажете че **Σ(n)** не може да бъде изчислена с машина от $\mathbf{TM_2}$.

Определение на машина **TC** (Тривиален Брояч)**:**

Тривиален Брояч ще наричаме машина с един регистър, съхраняващ цяло неотрицателно число. Програмата на машина от този модел съдържа следните команди:

inc увеличава регистъра с 1
dec намалява регистъра с 1
double удвоява числото в регистъра
test n_0 n_1 проверка за 0.
Ако регистъра съдържа 0,
ще се изпълни инструкция с номер n_0,
иначе ще се изпълни инструкция с номер n_1.
stop спира изчислението

Всички инструкции освен **test** предават управлението на следващата инструкция. Инструкцията **dec** спира изчислението, ако регистъра съдържа 0.

Задача 2**:**

Дефинирайте функция $\mathbf{S_{TC}(n)}$, по подобен на **S(n)** начин.

Докажете че $\mathbf{S_{TC}(n)}$ не може да бъде изчислена с машина от **TC**.

Определение на машина **2C** (Два Брояча)**:**

Два Брояча ще наричаме машина с два регистъра A и B, съхраняващи цели неотрицателни числа. Програмата на машина от този модел съдържа следните команди:

A+ увеличава регистъра A с 1
B+ увеличава регистъра B с 1
A- намаля регистъра A с 1
B- намаля регистъра B с 1
A? **n_0 n_1** проверка на A за 0.
Ако A съдържа 0,
то изпълни инструкция n_0,
иначе изпълни инструкция n_1.

B? $\mathbf{n_0}$ $\mathbf{n_1}$ проверка на B за 0.

stop спира изчислението

Всички инструкции освен A? и B? предават управлението на следващата инструкция. Инструкциите A- и B- спират изчислението, ако съответният регистър съдържа 0.

Задача 3:

Дефинирайте функция $\mathbf{S_{2C}(n)}$, по подобен на **S(n)** начин.

Докажете че $\mathbf{S_{2C}(n)}$ не може да бъде изчислена с машина от **2C**.

Упътване: Нека **A**=n, **B**=0 в началото. Напишете програма Double_{2C} за машина модел 2C, която удвоява **A**, т.е. след приключване на работата и **A**=2n, **B**=0. След това създайте конструкция като в теоремата от параграф 1.1b.3.

1.1b.6 Бележки:

(1) Функциите **S(n)** и **Σ(n)** са дефинирани за пръв път около 1960г. от Tibor Rado, professor at the Ohio State University.

(2) Задачата за изчисляването на **S(n)** и **Σ(n)** за малки стойности на **n** е известна като 'задача за работливия бобър'. Най-пълен сайт за тази задача има Heiner Marxen: http://www.drb.insel.de/~heiner/BB/.

Точните стойност на тези функции са известни само за n<5.

За n=5 и n=6 са известни само следните долни граници:

$S(5)>47*10^{6}$

$S(6)>3*10^{1730}.$

1.2 Сводимост. Определение. Няколко доказателства на сводимост.

В тази глава ще сравняваме различни модели изчисляващи машини. Няма да ни интересува скоростта на работа на машините, които сравняваме, а единствено възможността им да извършват различни пресмятания.

1.2.1 **Сводимост**

Неформално *определение* на **сводимост:**

Да означим с M_1 и M_2 два различни модела изчисляващи устройства.
Очевидно ако за всяка фиксирана машина (програма) от модел M_1 можем да конструираме машина (програма) от модел M_2, вършеща *идентични изчисления*, модел M_2 ще има по-големи изчисляващи възможности. Ще казваме, че модел M_1 е сводим към модел M_2 и ще пишем $M_1 => M_2$.

Как да разбираме фразата *идентични изчисления*?
Изясняването на този въпрос ще отложим за малко, като най-напред ще разгледаме няколко примера на сводимост:

1.2.1.1

*Теорема **TC_=>_2C:***
За всяка програма P_{TC} за машината *Тривиален Брояч* съществува програма P_{2C} за машината *Два Брояча*, която върши същата работа.

Доказателство:

В упъването към задача 3 от предната статия има препоръка за написването на програма за удвояване на регистъра A на машината *Два Брояча*. Ето примерна реализация на такава програма:

Програма $Double_{2C}$:

```
Double  A?      End     Mul2
Mul2    A-
        B+
        B+
        A?      Swap    Mul2
Swap    A+
        B-
        B?      End     Swap
End     Stop
```

Машината *Два Брояча* може да изпълнява всички команди с регистъра си A, които може и машината *Тривиален Брояч* с нейният единствен регистър, с единствено изключение - удвояването на A. Но програмата $Double_{2C}$ извършва точно такова удвояване, ползвайки временно регистъра B. Идеята на доказателството на теоремата е, като вземем програмата P_{TC} от условието на теоремата, да я променим до програма P_{2C} за машината *Два Брояча*, като редовете от P_{TC} съдържащи команда double, ще заменим със целия текст на програмата $Double_{2C}$ без последни й ред 'End Stop'. Останалите команди заменяме с аналогичните им (inc заменяме с A+, dec заменяме с A- и пр.).

Ще се сблъскаме и с технически проблем - новополучената програма ще е по дълга от оригиналната и съответните редове сочени от командите от тип **test** ще променят

номерата си при заместването на ред с няколко реда (какъвто е случая със замяната на double с $Double_{2C}$). Този проблем е разрешим и го оставяме на читателя като задача за упражнение.

В крайна сметка ще получим еквивалентната на P_{TC} програма P_{2C}. Да си представим сега че сме записали число **n** в регистъра на *Тривиален Брояч* и в регистър A на *Два Брояча*, а в регистър B сме записали 0. Да пуснем двете машини да работят паралелно, стъпка по стъпка, като всеки път когато се наложи изпълнение на double, да забавим *Тривиален Брояч* докато съответният фрагмент от P_{2C} приключи работата си. Двете машини ще работят синхронно и ако стигнат до команда stop, съответните регистри ще съдържат един и същ резултат. Ако пък една от машините не стига никога до команда stop, такова ще е и поведението на другата, т.е. ако гледаме на P_{TC} и P_{2C} като на машини за изчисляване на функция на цялото число **n**, излиза, че те изчисляват една и съща функция.

Първо пояснение на *идентични изчисления*:

Две машини правят *идентични изчисления*, когато при еднакви начални данни и двете изчисляват един и същ краен резултат или и двете се зациклят и никога не завършват изчисленията си.

Горното пояснение изглежда задоволително за машините от теоремата ***TC_=>_2C***, с което можем да завършим доказателството.

1.2.1.2

Теорема ***$TM_=>_TM_2$***:

За всяка *машина на Тюринг* съществува *машина на Тюринг с двубуквена азбука*, която върши същата работа.

Идея за доказателство:

Ще използваме техниката от предната теорема - всяка инструкция на машината от **TM** ще заместим с група инструкции за машина от $\mathbf{TM_2}$. Проблемът при това доказателство е представянето на данните върху лентата: изходната машина от **TM** използва богата азбука, която може да се състои от поне 3 букви, докато резултатната машина може да използва само 2. Решението е да представим всяка буква от по-богатата азбука като кратка поредица в двубуквената азбука. Такава поредица ще наричаме *байт* по аналогия с реалните компютри, където буквите и символите на естествения език се представят като 8-битови поредици от битове в електронната памет.

Доказателство:

Нека P_{TM} е машина на Тюринг, чиято азбука се състои от ***k*** букви. Ще построим машина с двубуквена азбука P_{TM2}, която да прави аналогични изчисления. Първо трябва да изберем подходящо предтавяне на буквите на P_{TM} върху двоичната лента на P_{TM2}. Нека ***l*** е най-малкото число, за което $\boldsymbol{2^{l-1}<k\leq 2^l}$. Очевидно *l* бита (така ще наричаме буквите от двубуквената азбука на P_{TM2}) ще са достатъчни за представяне на буквите на P_{TM}. Трябва да си направим табличка за съответствието между азбуката от *k* букви и съответните байтове (нашият байт ще има дължина *l*).

Ако например азбуката на P_{TM} се състои от 6-те букви ',' : * / 0 1, ще е необходимо 3-битово представяне. Ето една възможна табличка за кодиране на тези букви:

символ от лентата на P_{TM}	символ от лентата на P_{TM2}
,	000

:	001
*	010
/	011
0	100
1	101

Да си представим, че лентата на P_{TM2} е разделена на байтове с въображаеми вертикални линии. Ако в примера по-горе машината P_{TM} има записана на лентата си поредицата '/00,0:10,1,1/', аналогичният запис върху лентата на P_{TM2} ще изглежда така:

...|011|100|100|000|100|001|101|100|000|101|000|101|011|...

Удобно е да изберем представянето така, че празният символ на P_{TM} (',' в горният пример) да се кодира с поредица от *l* нули.

За всяка инструкция на P_{TM} ще създадем група инструкции за P_{TM2}, които да имитират точно работата ѝ върху двоичната лента на P_{TM2}. Ще приемем че в началото на тази имитация главата на P_{TM2} се намира над най-левия бит от байта, който съответства на някой символ от лентата на P_{TM}.

Имитацията на инструкцията на P_{TM} ще разбием на три фази:

През първата фаза главата на P_{TM2} ще обходи байта отляво надясно, за да разпознае кой символ от азбуката на P_{TM} представя този байт. За тази фаза ще са достатъчни около $2.k$ нови състояния (може да използваме само k състояния, ако съчетаем разпознаването на последния бит с началото на следващата фаза).

През втората фаза главата на P_{TM2} ще запише движейки се отдясно наляво представянето на символа, който съобразно указанието на имитираната инструкция трябва да замени разпознатия по време на първата фаза символ. За тази фаза ще са необходими $k.l$ нови състояния (l състояния за замяна на всяка от възможните k стойности на обработваният байт).

В началото на третата фаза главата на P_{TM2} ще се намира отново в началото на байта. Сега според всяко от k-те възможни указания главата ще трябва да се премести l позиции наляво или надясно, без да променя нищо по лентата, за да стигне до най-левия бит на някой от съседните байтове, т.е. ще са необходими още $k.l$ нови инструкции за P_{TM2}.

Горната схема определя машината P_{TM2}. Броят на нейните инструкции ще най-много $k.(2l+2)$ пъти по-голям от броя на инструкциите на P_{TM}. Детайлното описание на тези инструкции оставяме на любознателния читател (щото е доста досадно).

Да си представим сега някакви начални данни (поредица от символи), изписани върху лентата на P_{TM}. Използвайки табличката за кодиране, можем да изпишем еквивалентните им байтове върху лентата на P_{TM2}. Да пуснем двете машини да работят паралелно, стъпка по стъпка, като забавим P_{TM} дотолкова, че изпълнението на всяка нейна инструкция да съвпада с началото на съответната имитираща поредица на P_{TM2}. Двете машини ще работят синхронно и ако стигнат до команда stop, лентите им ще съдържат еквивалентни (в смисъла на кодирането описано в табличката) резултати. Ако пък една от машините не стига никога до команда stop, такова ще е и поведението на другата, т.е. както и в предната теорема можем да смятаме, че двете машини вършат една и съща работа.

Второ пояснение на *идентични изчисления*:

Две машини правят *идентични изчисления*, когато при *идентични* начални данни и двете изчисляват *идентични* крайни резултати или и двете се зациклят и никога не завършват изчисленията си. Под *идентични* данни/резултати разбираме всяка разумна схема за еднозначно преобразуване на данните на едната машина до данните на другата и обратно. Такава схема за преобразуване ще наричаме по-нататък '*разумно кодиране*'.

Горното пояснение изглежда задоволително за машините от теоремата ***ТМ_=>_ТМ$_2$***, с което можем да завършим доказателството.

Пояснение на *разумно кодиране*:

По горе определихме термините *идентични* данни и *разумно кодиране* с доста неясна граматична конструкция. Можем да дадем по-строго определение, а именно: можем да разглеждаме данните на коя да е машина като поредица от символи в нейния език Г (множеството от всички произволни поредици от символи от азбуката на машината). Нека смислените поредици, дефиниращи конкретно условие на конкретна задача да разглеждаме като фрази в някакъв формален език L, определящ синтактично правилните записи на условията (L ще е подмножество на Г). Ще казваме, че езикът L е разрешим, ако има машина на Тюринг, която за всяка фраза от Г определя дали е или не от L.

Например при машината Double Г се състои от всички поредици от нули и единици, а L - от всички поредици от единици, като разбира се отчетем нулите заграждащи поредицата от единици (тези 'празни' символи, маркиращи краищата на лентата, можем да заменим с един специален символ - '.', означаващ потенциално безкраен брой 'празни' символи). В тази трактовка езикът L за Double се състои от фразите '..', '.1.', '.11.', '.111.' и т.н.

Определение на *разумно кодиране*:

Нека сега имаме две машини, M_1 и M_2 и съответните езици $Г_1$, $Г_2$, L_1 и L_2. Ще казваме, че L_1 и L_2 са идентични, или че L_1 се свежда чрез *разумно кодиране* до L_2 ако съществуват две машини на Тюринг M_{1_2}(кодираща) и M_{2_1}(декодираща) със следните свойства:

1. Азбуката им се състои от буквите на азбуките на M_1 и M_2.

2. Ако на лентата и е изписана коя да е дума **x** от L_1 и стартираме M_{1_2}, тя ще завърши изчислението си и ще получи дума **y** от L_2. Ако сега върху думата **y** стартираме машината M_{2_1}, тя ще завърши работата си и ще получи думата **x**.

3. M_{2_1}, подобно на M_{1_2} също е обратима, т.е. ако за всяка дума **y** от L_2 стартираме машината M_{2_1}, тя ще завърши работата си и ще получи дума **x** от L_1.

Примери за *разумни кодирания*:

1. Когато работим с числа ползваме различни бройни системи за представянето им. Нашата програма Double ползва странното и непрактично унарно (линейно) кодиране - всяко число се представя с толкова поредни символа (единици, чертички или каквото си изберем) колкото е самото число. От такова представяне тръгват всички древни бройни системи, с малки подобрения - пример е системата на римските цифри, която все още се ползва. По модерните системи са позиционни - двоична при реалните компютри, десетична за ръчни изчисления. През 60-те години на XX век в Съветският съюз е произвеждан серийно компютърът *Сетун*, който е представял числата в балансиран троичен код (троична бройна система с цифри -1, 0 и 1). Всички тези представяния на естествените (или цели с добавяне на знак при нужда) числа са разумно сводими едно към друго. Любознателният читател би могъл за упражнение да си напише кодираща/декодираща програма за коя да е двойка бройни системи. Изобщо всяко представяне на числата, което може с *разумно кодиране* да се сведе до унарното (линейно) представяне ще наричаме *разумно представяне*

на числата.

2. В първата статия изписвахме програмите за машините на Тюринг по два възможни начина - табличен (всеки ред описва една инструкция) и линеен (всеки ред описва едно указание). Тези представяния също са *идентични*.

Коментар за S_tm_2 от предната статия:

Нека допуснем, че машината S_tm_2 очаква входните си данни (т.е. числото n) в някакво *разумно представяне*, но не точно поредица от n единици. Да допуснем, че и изхода на машината е в кое да е *разумно представяне*.

Нека входното число е кодирано в разумно представяне K_0. Разумните представяния са сводими едно към друго, тъй че можем да добавим пред програмата прекодиращ фрагмент, който да очаква на входа число в просто представяне (поредица от n единици), след което да го превръща в представяне K_0. Така ще получим машина $S_1\_tm_2$, която ще е малко по-голяма от изходната и ще приема входните си данни в просто представяне.

Да направим подобна добавка и в края на машината. Ако изхода на нашата изчисляваща машина е в представяне K_1, ще добавим фрагмент код, който да превърне резултата S(n) от представяне K_1 в просто (линейно) представяне. Така ще получим машина $S_2\_tm_2$.

Можем също да променим тази машина така, че при началото и края на работата й главата да се намира над най-лявата единица. Така ще получим машина, удобна за нуждите на доказателството.

1.2.1.3

Теорема ***TM_2_=>_TM:***

За всяка *машина на Тюринг с двубуквена азбука* съществува *машина на Тюринг*, която върши същата работа.

Доказателство:

Нека P_{TM2} е *машина на Тюринг с двубуквена азбука.* Но P_{TM2} е същевременно и *машина на Тюринг,* т. е. тя самата е търсената съответна машина с което доказателството е приключено.

1.2.2 Някои свойства на сводимостта:

Нека M_1, M_2 и M_3 са различни модели изчисляващи устройства, M_1 е сводим към M_2, а M_2 е сводим към M_3. Тогава:

(1) M_1 е сводим към M_3 (транзитивност на сводимостта).
(2) Ако една задача е решима в модела M_1, тя е решима и в модела M_2.
(3) Ако една задача не е решима в модела M_2, тя не е решима и в модела M_1.

Доказателствата на тези свойства следват пряко от определението на сводимост.

1.2.3 Определение за *еквиваленост* (в смисъл на изчислимост):

За класовете машини TM_2 и TM доказахме с последните две теореми, че всеки е *сводим* към другия. Такива класове ще наричаме *еквивалентни в смисъл на изчислимост*. Подобна еквивалентност ще обозначаваме така: TM_2 <==> TM.

Очевидно еквивалентността на класове изчисляващи устройства е транзитивна, подобно на сводимостта.

1.2.4 Задачи за упражнение:

Определение на *обръщащи машини на Тюринг*:

Да означим с RTM (Reversal Turing Machines) всички машини на Тюринг с двубуквена азбука (тоест машини от класа TM_2), които след прочитане на 0 от лентата записват 1, а след прочитане на 1 записват 0 (главата винаги обръща символа над който се намира след прочитането му).

Задача 1*: Докажете еквивалентността TM_2 <==> RTM.

Задача 2: Докажете еквивалентността TM <==> RTM.

Определение на *разрушаващи машини на Тюринг*:

Да означим с DTM (Destructive Turing Machines) всички машини на Тюринг с двубуквена азбука (т.е. от класа TM_2), които могат да заменят 0 с 1 върху лентата, но не могат да заменят 1 с 0 (т.е. за тези машини можем да си представяме, че лентата е от хартия, главата може да продупчи лентата, но не може да залепи вече пробита дупка. 0 съответства на здрава клетка, 1 – на продупчена).

Задача 3*: Докажете еквивалентността TM_2 <==> DTM.

Упътване:

Да си представим, че работната част от лентата е оградена с някаква маркираща поредица (специален байт). Ако трябва да сменим 0 с 1 върху тази част – правим го по нормалния начин. Ако трябва да извършим обратната замяна, първо нека препишем цялата работна област отдясно на текущата, без да перфорираме бита над който е била главата. По подобен начин ще постъпим и когато главата трябва да напусне работната област. За да извършим самият презапис на работната област ще е необходимо представянето на всеки бит да се извършва чрез специално раздуто кодиране !

1.2.5 Бележки:

(1) Машините от клас DTM са описани около 1960г. от Hao Wang – американски логик и философ от китайски прозход.

(2) Машините от клас RTM са описани от мен около 1990г. http://skelet.ludost.net/bb/RTM.htm

1.3 Още доказателства на сводимост. Тезис на Чърч. Дефиниция на алгоритъм. Следствия.

1.3.1 Сводимост между различни изчисляващи устройства.

.....

Тук ще има още някое и друго доказателство за сводимост, така че да се получи голяма група устройстава за изчисляване, еквивалентни на TM.

.....

1.3.2 Тезис на Чърч и дефиниция на алгоритъм.

През 30-те години на XX век били предложени няколко математически определения на понятието ефективен метод (алгоритъм, формална процедура, изчисляващо устройство, компютър). Било също доказано, че всички такива определения са еквивалентни, т.е. класа задачи, които можем да решим с някоя от предложените схеми (примерно с машини на Тюринг) точно съвпада с класа задачи, решими с всяка друга разумна схема за правене на изчисления (примерно най-мощните съвременни компютри, с допълнителна възможност да им добавяме памет по време на изчислението). Тази еквивалентност, приемана като природен закон за всички възможни схеми за изчисляване се нарича ***тезис на Чърч*** по името на американският логик Алонсо Чърч, който я забелязва най-рано.

Тезисът на Чърч ни позволява да дефинираме понятието алгоритъм така:

1.3.3 *Определение* на *алгоритъм*:

Алгоритъм е всяко правило за изчисляване, което може да бъде представено чрез конкретна ***машина на Тюринг***.

1.3.4 Следствия от приемането на тезиса на Чърч:

Машината **Double**, която описахме в 1.1a.2 представя алгоритъм за удвояване на число, независимо колко голямо е то. За решаването на една задача може да има различни алгоритми. Числото в нашия примерен алгоритъм беше представено в линейно кодиране и удвояването му е доста бавно. Можем да направим друга, по-ефективна машина на Тюринг за удвояване, която да ползва азбука от 3 букви (празен символ ' ', '0' и '1'), числото на лентата да е записано в двоичен код, при което удвояването ще се сведе до изписването на '0' в първата празна клетка отдясно на числото.

Въпреки факта, че дефинираме понятието алгоритъм чрез конкретна изчисляваща машина, то има универсална природа, дължаща се на еквивалентността на изчислителните схеми. В този смисъл тезисът на Чърч дава плътност на понятието алгоритъм и оправдава връзката между точната му математическа дефиниция и нашата интуитивна представа за ефективен метод.

Пак от тезисът на Чърч следва, че не е необходимо когато създаваме нов алгоритъм да се мъчим да го опишем като машина на Тюринг. Достатъчно е да опишем алгоритъма на кой да е от стандартните езици за програмиране. Тезисът гарантира, че получената програма може да се трансформира в еквивалентна машина на Тюринг.

Коментар за функциите S(n) и Σ(n):

От тезиса на Чърч и свойство (3) на сводимостта следва, че тези функции не могат да бъдат изчислени с никакви изчисляващи устройства.

Прието е задачите, които не могат да бъдат решени с изчисляващи устройства да се наричат *алгоритмично нерешими задачи*.

Коментар за езиците за програмиране:

Можем да разглеждаме всеки език за програмиране като модел изчисляващи устройства (всяка програма от този език разглеждаме като конкретна машина от модела).

Възникват интересни въпроси, след като сме приели тезисът на Чърч: защо има толкова много езици за програмиране, след като всички те са еквивалентни в смисъл на изчислимост ? Не е ли възможно да се проектира единствен език за програмиране, на който да бъдат преведени всички съществуващи програми и оттук нататък да ползваме само него ?

Възможният отговор е, че езиците за програмиране могат да се разглеждат като обекти, съставени от две компоненти (притежаващи две природи) - математическа и човешка. Математическа компонента е обща за всички езици - чрез тях можем да изразим всеки алгоритъм (разбира се, с някаква разлика в ефективността). Човешката природа на езиците за програмиране отразява различни други свойства, нямащи пряко отношение към изчислимостта - историчност, изразни средства, разпространение, специализация и пр.

Точно нематематическата компонента е източник на разнообразие, което е непреодолимо, но този въпрос излиза далеч извън рамките на тази статия и се нуждае от допълнителна, по-широка дискусия.