



Модул 10. Алгоритми и структури от данни

Тема 1.. Алчни алгоритми

Задача 1.1. Задача за възлагане на дейности

Даден е списък с дейности, както и тяхното начално и крайно време за изпълнение. Да се намери максималния брой неконфликтни дейности, които човек или машина може да извърши, при условие, че участващият човек или машина може да извършва само една дейност в даден момент. За всеки две дейности се казва, че са неконфликтни, ако началното време на едната дейност е по-голямо или равно на крайното време на другата дейност.

Пример:

Вход			Изход		
Дейност	Начало	Край	Дейност	Начало	Край
a1	1	3	a3	1	2
a2	0	4	a7	3	5
a3	1	2	a6	5	8
a4	4	6			
a5	2	9			
a6	5	8			
a7	3	5			
a8	4	5			

Подсказки:

1. Подредете дейностите по време на завършване във възходящ ред. Ще използваме примерния вход даден по-горе.

Дейност	a1	a2	a3	a4	a5	a6	a7	a8
Начало	1	0	1	4	2	5	3	4
Край	3	4	2	6	9	8	5	5

2. Изберете първата дейност

Сортирани дейности	a3	a1	a2	a7	a8	a4	a6	a5
Начало	1	1	0	3	4	4	5	2
Край	2	3	4	5	5	6	8	9

3. Изберете следващата дейност, чието начално време е по-голямо или равно на времето за завършване на първоначално избраната дейност.



В нашия случай края на дейност а3 съвпада с началото на дейност а5. Намерен е един възможен отговор- а3, а5.

Сортирани дейности	а3	а1	а2	а7	а8	а4	а6	а5
Начало	1	1	0	3	4	4	5	2
Край	2	3	4	5	5	6	8	9

4. Повторете стъпка 3, докато всички дейности са проверени. Намираме следващата дейност, чието начално време е равно или по-голямо от 2. Това е дейност а7. Дейност а7, завършва в 5. Търсим следващата дейност, чиято начално време е по-голямо или равно на 5. Това е дейност а6. Тя завършва в 8. Нямаме следваща дейност, чието начално време е по-голямо или равно на 8.

Сортирани дейности	а3	а1	а2	а7	а8	а4	а6	а5
Начало	1	1	0	3	4	4	5	2
Край	2	3	4	5	5	6	8	9

Отново проверяваме стъпка 3. С по-големи стойности като начало на следващи дейности след а3 са а8 и а4, но техният брой е 2. И затова отговора на задачата с конкретните входни данни е а3, а7 и а6 (максималният брой дейности, които са неконфликтни са 3).

Пример

Вход	Изход
а1 1 3 а2 0 4 а3 1 2 а4 4 6 а5 2 9 а6 5 8 а7 3 5 а8 4 5 END	

Задача 1.2. Задача за магнитната лента

Дадени са n програми с дължини l_1, l_2, \dots, l_n и магнитна лента с последователен достъп. Последователният достъп означава, че за да се прочете запис, намиращ се на дадена позиция върху лентата, трябва да се премине (да се превърти лентата) до указаното място.



K_1	K_2	K_3	K_4	...	K_N
l_1	l_3	l_2	l_{K4}	...	l_{KN}

Програма K_1 е с дължина 1, K_2 - с дължина 3, K_3 - с дължина 2. За да бъде прочетена третата програма (с дължина l_2), трябва да се "превъртят" всички програми преди нея — тези с дължина l_1 и l_3 .

Дадени са вероятностите на изпълнение на всяка програма $p_1, p_2, \dots, p_N, 0 \leq$

$p_i \leq 1, 1 \leq i \leq n, \sum_{i=1}^n p_i = 1$. Например, вероятност 0.5 означава, че програмата се изпълнява толкова често, колкото всички останали програми, взети заедно. При фиксирано подреждане K_1, K_2, \dots, K_N на програмите върху лентата средното време T , необходимо за зареждането на произволна програма, се дефинира по следния начин:

$$T = \frac{\sum_{i=0}^n \left(p_{ki} \sum_{j=1}^i l_{kj} \right)}{n}$$

Задачата е да се намери подреждане на програмите върху лентата, минимизиращо средното време.

Пример:

Вход			Изход
Програма	дължина	вероятност на изпълнение	1 3 4 2
1	1	0.3	
2	3	0.1	
3	2	0.5	
4	4	0.2	

Подсказки:

Използвайте факта, че колкото по-често се използва една програма, толкова по-напред върху лентата трябва да бъде записана тя. В примерния вход по-горе програмите ще са в следния ред: 3, 1, 4, 2.

Другият определящ критерий е дължината на програмата — колкото по-дълга е тя, толкова по-назад трябва да бъде върху лентата (за да "пречи" колкото се може по-малко, когато се наложи да се превърта), т.е. подредбата според този критерий в конкретния пример ще е: 1, 3, 2, 4.



Има нареждане на програми, където не е изпълнено, че $\rho_{ki}/l_{ki} \geq \rho_{ki+1}/l_{ki+1}$. Това означава, че $\rho_{ki} \cdot l_{ki+1} < \rho_{ki+1} \cdot l_{ki}$. Отговорът значително ще се подобри, ако се разменят местата на програма i и $i+1$, т.е. ако прилагаме тази операция ще стигнем до сортиран масив по ρ/l , който е най-оптималния. Записваме последователно програмите върху лентата, като на всяка стъпка избираме програмата K_i , за която отношението ρ_{ki}/l_{ki} е максимално.

Така получаваме следните окончателни данни. За програма 1 - $0.3/1=0.3$. За програма 2 - $0.1/3 = 0.03$, за програма 3 - $0.5/2 = 0.25$ и за програма 4 - $0.2/4=0.05$. Подреждаме резултатите (съответващите им програми) в низходящ ред.

Задача 1.3. Задача за подреждане на работа в срок

Разполагате с n работни места (j_1, j_2, \dots, j_n). За всяко работно място са дадени краен срок на изпълнение (d_1, d_2, \dots, d_n) и каква печалба носи (p_1, p_2, \dots, p_n). Задачата е да се спечели максимална печалба, като в даден момент може да се назначи или обработва само една работа. Печалба има само когато работата е завършена на или преди крайния срок. Предполага се, че всяка работа отнема единица време за завършване.

Пример:

Вход			Изход
5			$j_2 \ j_1 \ j_3$
1	2	60	Максимална печалба 180
2	1	100	
3	3	20	
4	2	40	
5	1	20	

Подсказки:

- Нека имаме n работни места j_1, j_2, \dots, j_n , които са свързани с краен срок d_1, d_2, \dots, d_n и печалба p_1, p_2, \dots, p_n .

индекс	1	2	3	4	5
работно място	j_1	j_2	j_3	j_4	j_5
краен срок	2	1	3	2	1
печалба	60	100	20	40	20



2. Сортирайте работните места според печалбата си в низходящ ред. Ако две или повече работни места имат една и съща печалба, ги сортирайте според вписването им в списъка за работа.

индекс	1	2	3	4	5
работно място	j2	j1	j4	j3	j5
краен срок	1	2	2	3	1
печалба	100	60	40	20	20

3. Изберете първата работа. Тя има краен срок 1. Това означава, че трябва да завършим работа j2 във време 1, ако искаме да вземем нейната печалба. Вземете следващата работа, чиито краен срок е по-голям от 1. На всяка стъпка можете да изпълнявате само една задача. Взимаме j1. Тя има краен срок 2. Това означава, че трябва да завършим работа j1 на или преди време 2, за да спечелим печалбата.

индекс	1	2	3	4	5
работно място	j2	j1	j4	j3	j5
краен срок	1	2	2	3	1
печалба	100	60	40	20	20

4. Следващата по ред работа е j4, но не можете да я изберете, тъй като нейният краен срок е равен на крайния срок на вече избраната с по-висока печалба работа j1. Взимаме работа j3, тя има краен срок 3. Това означава, че трябва да завършим работа j3 на или преди време 3, за да спечелим печалбата.

индекс	1	2	3	4	5
работно място	j2	j1	j4	j3	j5
краен срок	1	2	2	3	1
печалба	100	60	40	20	20

Необходимо е да се проверява, дали задачите за изпълнение до i-тата стъпка (за всяко $i = 1, 2, \dots, n$) не стават повече от i (на всяка стъпка може да се изпълнява по една задача).



Така стигаме до максимална печалба 180 и наемане на три работни места.

Задача 1.4. Задача за множествата

Нека са дадени две множества. Първото е така наречената Вселена, а второто е множество от подмножества. Множеството от подмножества съдържа всички елементи от Вселената (няма други елементи), като някои могат и да се повтарят. Задачата е да се намери най-малкото множество от подмножества, които да съдържат всички елементи от Вселената.

Пример:

Вход	Изход
Вселена: 1, 2, 3, 4, 5 Брой на подмножествата: 4 1 2, 4 5 3	(4) подмножества: { 2, 4 } { 1 } { 5 } { 3 }
Вселена: 1, 2, 3, 4, 5 Брой на подмножествата: 4 1, 2, 3, 4, 5 2, 3, 4, 5 5 3	(1) подмножество: { 1, 2, 3, 4, 5 }
Вселена: 1, 3, 5, 7, 9, 11, 20, 30, 40 Брой на подмножествата: 6 20 1, 5, 20, 30 3, 7, 20, 30, 40 9, 30 11, 20, 30, 40 3, 7, 40	(4) подмножества: { 3, 7, 20, 30, 40 } { 1, 5, 20, 30 } { 9, 30 } { 11, 20, 30, 40 }

Подсказки:

На всяка стъпка взимайте подмножеството, което съдържа най-много елементи от Вселената и което съдържа елементи, които все още не сте взели. На първата стъпка винаги взимайте подмножеството с най-голям брой елементи. Когато вземете подмножеството, премахнете всички елементи в него от Вселената.

Нека разгледаме третия примерен вход:



Вселена: 1, 3, 5, 7, 9, 11, 20, 30, 40

Брой на подмножествата: 6

20

1, 5, 20, 30

3, 7, 20, 30, 40

9, 30

11, 20, 30, 40

3, 7, 40

1. Сортирайте подмножествата в низходящ ред в зависимост от броя на елементите, които съдържат

3, 7, 20, 30, 40

1, 5, 20, 30

11, 20, 30, 40

3, 7, 40

9, 30

20

2. Вземете подмножеството с най-голям брой елементи (т.е. първото подмножество). Премахнете всички елементи в него от Вселената.

Вселена: 1, 5, 9, 11

Подмножества:

1, 5, 20, 30

11, 20, 30, 40

3, 7, 40

9, 30

20

3. Вземете следващото подмножество с най-голям брой елементи. Премахнете всички елементи в него от Вселената.

Вселена: 9, 11

Подмножества:

11, 20, 30, 40

3, 7, 40

9, 30

20

4. Вземете следващото подмножество с най-голям брой елементи. Премахнете всички елементи в него от Вселената.

Вселена: 9



Подмножества:

3, 7, 40 (тук не съществуват елементи, които все още не сме взели)

9, 30

20

5. Вземете следващото подмножество с най-голям брой елементи.
Премахнете всички елементи в него от Вселената.

Вселена: 9

Подмножества:

9, 30

20

6. Вземете следващото подмножество с най-голям брой елементи.
Премахнете всички елементи в него от Вселената.

Вселена: \emptyset

Подмножества:

20

7. Алгоритъмът спира, тъй като Вселената е празно множество.

Тема 2. Рекурсия, пълно изчерпване и търсене с връщане назад

Задача 2.1. Разходката на коня

Дадена е обобщена шахматна дъска $N \times N$ и шахматен кон, разположен в полето с координати (x_0, y_0) . Търси се такава последователност от ходове на коня в смисъла на шахматните правила за движение на тази фигура, при която всяко поле се посещава точно веднъж.

Пример:

При $n=5$ съществуват 304 различни разходки на коня. Ето две от тях:

Разходка 1	Разходка 89
1 6 15 10 21	1 10 25 16 7
14 9 20 5 16	20 15 8 11 24
19 2 7 22 11	9 2 21 6 17
8 13 24 17 4	14 19 4 23 12
25 18 3 12 23	3 22 13 18 5

Подсказки:

За решаване на задачата е необходимо да се обхоят всички клетки точно по веднъж, т.е. извършване на N^2 на брой ходове.

// Pseudo Code

Try (ход K)



```
while (допустим следващ ход)
    регистрираме хода
    if (броя на ходовете =  $N^2$ ) Try ← хода
    else Try ← Try (ход K+1)
    премахваме регистрацията
Край
```

Регистрацията на поредния ход на коня е свързана освен с маркиране на текущото поле като посетено и с добавянето му към списък от координатите на посетените полета. Според шахматните правила за всяка позиция на коня съществуват точно 8 потенциални кандидата, стига, разбира се да не извеждат коня извън границите на дъската или върху вече посетеното поле. Една от възможностите е да направите два отделни списъка на разликите, съответно за всяка от координатите.. За целта може да въведете масив T, който ще съдържа координатите на относителното отместване по отношение на текущата позиция по x и y съответно. Ще считаме, че началното поле (x_0, y_0) е горният ляв ъгъл на обобщената шахматна дъска. Задачата се решава от функцията Try, която получава като параметри текущите координати на шахматния кон върху дъската, както и поредния номер на хода, който трябва да се извърши.

Задача. 2.2. Пътища в лабиринт

Даден е лабиринт. Задачата е да се намерят всички пътища от началната клетка, която се намира в горния ляв ъгъл (с координата - 0, 0) до изхода, маркиран със символа 'e'. Празните клетки са маркирани с тире "-", а стените със звезда "*". На първия ред се въвеждат размерите на лабиринта. На следващите редове действителния лабиринт. На изхода се извежда пътя, като последователност от символи (Up->Left->Right->Down). Редът на пътищата няма значение.

Пример:

Вход	Изход
3 3 --- -* --e	RRDD DDRR
3 5 -**-e ----- *****	DRRRRU DRRRUR



Подсказки:

Лабиринтът се представя като двумерен масив с N реда и M стълба. Започва се от горния ляв ъгъл, като може да се движите в четирите посоки - нагоре, надолу, наляво и надясно. Проходимите клетки са отбелязани с тире "-", а непроходимите - със звезда "-*".

Да разгледаме първия примерен вход:

-(0, 0)	-	-
-	*	-
-	-	e

RRDD

-(0, 0)	-	-
-	*	-
-	-	e

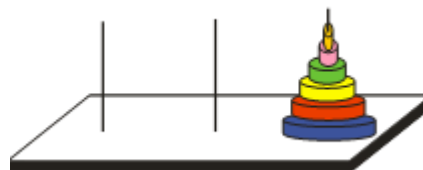
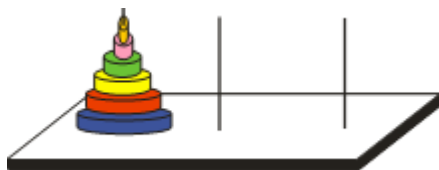
DDRR

Идеята за решаване на тази задача е рекурсия, като се използва backtracking. Нека сме в клетка с координати (X, Y) . Стъпки на рекурсията:

- търси_изход($X, Y+1$)
- търси_изход($X+1, Y$)
- търси_изход($X, Y-1$)
- търси_изход($X-1, Y$)

Задача. 2.3. Ханойски кули

Дадени са три стълба. На първия са поставени n диска с различен диаметър, наредени един върху друг от най-големия към най-малкия диск. Задачата е да се преместят всички дискове на третия стълб, като се запази подредбата им и при разместванията се спазва правилото винаги да се поставя по-малък диск върху по-голям.



Пример:

Вход	Изход
3	1-->3 1-->2 3-->2 1-->3 2-->1 2-->3 1-->3

Подсказки:

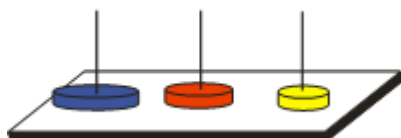
Нека разгледаме задачата при $n=3$.



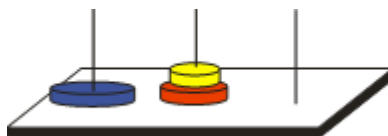
1 стъпка:



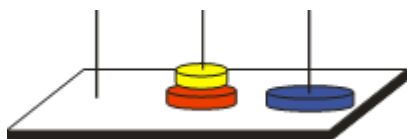
2 стъпка:



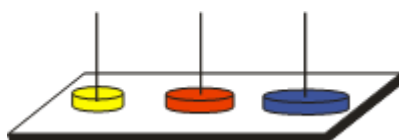
3 стъпка:



4 стъпка:



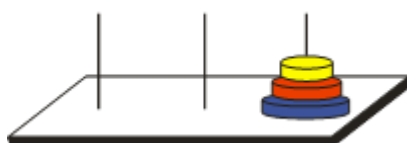
5 стъпка:



6 стъпка:



7 стъпка:



Броят на стъпките за решаване на задачата е $2^n - 1$. При $n=3$ получаваме $2^3 - 1 = 8 - 1 = 7$.

Задачата за n диска се свежда до задача за $n-1$ диска по следния начин:

- премествахме горните $(n-1)$ диска на стълб 2, като използваме стълб 3 за помощен
- премествахме n -тия диск от стълб 1 на стълб 3
- премествахме $(n-1)$ -те диска от стълб 2 на стълб 3, като използваме стълб 1 за помощен

Когато дисковете са 4 на брой, целта ни е да поставим най-големия диск на мястото му на третия стълб, после разиграването на останалите 3 диска ще се сведе до вече решавана задача. Когато дисковете станат 5, отново търсим начин да поставим най-големия на третия стълб, и после следва решаваната задача с 4 диска и т.н. По този начин общата задача се разбива на по-малки задачи от същия тип, което води до рекурсивно решение. За рекурсивната функция са ви необходими 4 параметъра: I - броя на дисковете, II - от кой стълб взимаме, III - кой стълб е помощен и IV - на кой стълб поставяме диска.



Тема 3. Комбинаторни алгоритми

Задача 3.1. Пермутации

Изчислете броя на пермутациите за дадена стойност на N . Известно е, че броят на пермутациите е равен на факториела на N , т.е. $N! = 1.2.3...N$. Рекурсивната дефиниция е $N! = N \cdot (N-1)!$.

Пример:

Функцията има ограничен обхват. Тя ще работи за числа N по-малки или равни на 12. За големи стойности на N се изисква използването на дълги числа (умножение на дълги числа).

Вход	Изход
1	1
2	2
3	6
4	24
5	120
6	720
7	5040 Последното число, което се побира в int
8	40320
9	362880
10	3628800
11	39916800
12	479001600 Последното число, което се побира в long int
13	6227020800
...	...
30	265252859812191058636308480000000
...	...

Подсказки:

```
Fac (k)
Begin
r:=1;
For i:=1 To k Do r:=r*i;
Fac:=r;
End;
```



Задача 3.2. Сума нула

Нека са дадени числата a_1, a_2, \dots, a_n . Да се поставят операции "+" и "-" между числата a_i и a_{i+1} , за $i = 1, 2, \dots, n-1$ така, че резултатът след пресмятане на получения израз да бъде равен на 0.

Например, ако са дадени естествените числа от 1 до 8, то няколко възможни решения на задачата са:

$$1 + 2 + 3 + 4 - 5 - 6 - 7 + 8 = 0$$

$$1 + 2 + 3 - 4 + 5 - 6 + 7 - 8 = 0$$

$$1 + 2 - 3 + 4 + 5 + 6 - 7 - 8 = 0$$

$$1 + 2 - 3 - 4 - 5 - 6 + 7 + 8 = 0$$

От клавиатурата се въвежда цяло число n - броя на числата и след него n на брой числа.

Пример:

Вход	Изход
8	$+1 +2 +3 +4 -5 -6 -7 +8 = 0$
1 2 3 4 5 6 7 8	$+1 +2 +3 -4 +5 -6 +7 -8 = 0$
	$+1 +2 -3 +4 +5 +6 -7 -8 = 0$
	$+1 +2 -3 -4 -5 -6 +7 +8 = 0$
	$+1 -2 +3 -4 -5 +6 -7 +8 = 0$
	$+1 -2 -3 +4 +5 -6 -7 +8 = 0$
	$+1 -2 -3 -4 -5 +6 +7 -8 = 0$
	$-1 +2 +3 -4 +5 -6 -7 +8 = 0$
	$-1 +2 +3 -4 -5 +6 +7 -8 = 0$
	$-1 +2 -3 +4 +5 -6 +7 -8 = 0$

Подсказки:

Генерирайте всевъзможните вариации с повторение на $n-1$ елемента от втори клас, т.е. всевъзможните наредени $(n-1)$ -орки, съставени от 0 и 1 (което отговаря на положителен и отрицателен знак пред съответното число). За всяка такава $(n-1)$ -орка проверете дали е решение на задачата, като за целта пресметнете стойността на съответния израз.

Задача 3.3. Разбиване на число (като сума от числа)

По дадено естествено число n да се намерят всички възможни не наредени представяния (разбивания) на n като сума от естествени числа (не непременно различни). Така например, числото 5 може да се разбие по следните 7 начина:

$$5 = 5$$

$$5 = 4 + 1$$

$$5 = 3 + 2$$



$5 = 3 + 1 + 1$
 $5 = 2 + 2 + 1$
 $5 = 2 + 1 + 1 + 1$
 $5 = 1 + 1 + 1 + 1 + 1$

От клавиатурата се въвежда едно цяло число n - числото за "разбиване".

Пример:

Вход	Изход
7	6+1 5+2 5+1+1 4+3 4+2+1 4+1+1+1 3+3+1 3+2+2 3+2+1+1 3+1+1+1+1 2+2+2+1 2+2+1+1+1 2+1+1+1+1+1 1+1+1+1+1+1+1

Подсказки:

Може да използвате рекурсивен алгоритъм:

- $\text{разбиване}(0) = \{\}$
- $\text{разбиване}(n) = \{k\} + \text{разбиване}(n-k)$, където $k = n, n-1, \dots, 1$.

Трябва да внимавате и да избегнете генериране на повтарящи се разбивания, като:

$5 = 3 + 2$
 $5 = 2 + 3$

Всяко следващо събираемо е необходимо да бъде по-малко или равно на предходното. Рекурсивната функция, извършваща разбиването, има два аргумента: n (число за разбиване) и променлива, показваща колко пъти досега е било разбивано числото.



Задача 3.4. Разбиване на число (като произведение от числа)

По дадено естествено число n да се намерят всички възможни не наредени представяния (разбивания) на n като произведение от естествени числа (не непременно различни).

От клавиатурата се въвежда едно цяло число n .

Пример:

Вход	Изход
50	25 * 2 10 * 5 5 * 5 * 2

Подсказки:

Алгоритъмът, по който можете да реализирате разлагането, е аналогичен на този, който е описан в задача 3. Вместо $devNum(n-k, cnt+1)$ ще извикваме рекурсивно $devNum(n/k, cnt+1)$, при това не за всяко k , а само за тези, за които $n \% k == 0$. Условието за продължаване на разбиването (цикъла `for`) ще бъде $k > 1$, а не $k \geq 1$, т.е. гъното на рекурсията ще бъде $k == 1$, а не $k == 0$ (последното се обяснява лесно: 0 и 1 са именно идентитетите на операциите събиране и умножение).

Задача 3.5. Разбиване на числа (като сума от дадени числа)

Нека са дадени пощенски марки от 2, 5 и 10 лева. Трябва да се изпратим колет на стойност 20 лева. Всички възможности (общо 6 на брой) за образуване на тази сума са:

$$20 = 10 + 10$$

$$20 = 10 + 5 + 5$$

$$20 = 10 + 2 + 2 + 2 + 2 + 2$$

$$20 = 5 + 5 + 5 + 5$$

$$20 = 5 + 5 + 2 + 2 + 2 + 2 + 2$$

$$20 = 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2$$

От клавиатурата последователно се въвеждат числата n - стойността на колета, m - броя на наличните марки, и t на брой числа - стойностите на марките. Всички числа са цели числа.

Пример:

Вход	Изход
------	-------



20 3	5 + 5 + 5
2 5 10	5 + 5 + 3 + 2
	5 + 3 + 3 + 2 + 2
	5 + 2 + 2 + 2 + 2 + 2
	3 + 3 + 3 + 3 + 3
	3 + 3 + 3 + 2 + 2 + 2
	3 + 2 + 2 + 2 + 2 + 2 + 2

Подсказки:

Алгоритъмът за решаването на тази задача е подобен на този за разбиване на число като сума от естествени числа. Пазете числата, които можете да ползвате при разбиването, в масив `given[gN]`. Цикълът ще се изпълнява за $p = 0, 1, \dots, gN-1$, като при рекурсивното извикване вместо с p , намалете p със съответната стойност на `given[p]`.

Тема 4. Динамично оптимиране

Задача 4.1. Алея

Алея в градската градина на град X има дължина L и трябва да бъде павирана с правоъгълни плочи. Всяка плоча има ширина, колкото алеята, но дължините на плочите са различни. Плочите са N вида и видовете са номерирани от 1 до N . Разполагаме с достатъчно плочи от всеки вид. Определете по колко различни начина може да се павира алеята, като се използват съществуващите видове плочи.

На първия ред на стандартния вход се въвеждат две положителни цели числа, разделени с интервал - L , дължината на алеята и N - броят на видовете плочи. На втория ред се въвеждат N на брой положителни цели числа, разделени с интервал, $d_1 d_2 \dots d_n$, където d_i е равно на дължината на плочата от вид i , $i = 1, \dots, N$.

Програмата да извежда едно положително цяло число, равно на броя начини, по които алеята може да павира.

Пример:

Вход	Изход
5 3 2 1 3	13

Ограничения:

- $0 < L \leq 700$
- $0 < N \leq 250$
- $0 < d_i < 1000$, за всяко $i=1 \dots N$.
- $d_i \neq d_j$, за всяко $i=1 \dots N$.

Резултатът може да има най-много 300 цифри.



Подсказки:

Пребройте различните подреждания на плочите според това каква плоча завършва на L -тия метър, ако позволява дължината ѝ. Ако плочата с пореден номер i , завършва на L -тия метър, то броя на различните начини за това е броят на различните подреждания на плочите до L - дължината на i -тата плоча. И така общия брой е сумата от различните подреждания за всеки вид плоча. Използвайте един масив, в който да пазите дължините на различните плочи – $p[]$, и един масив, в който да пазите броя на подрежданията – $b[]$.

Задача 4.2. Триъгълник от числа

Нека е даден следния триъгълник от числа:

				7					
			3			8			
		8		1			0		
	2		7			4		4	
4		5		2			6		5

Напишете програма, която пресмята най-голямата възможна сума от числа, разположени на някои от пътищата, започващи от най-горната точка на триъгълника и завършващи в точка от основата на триъгълника. Изисква се пътищата да са такива, че при всяка от стъпките движението да се осъществява надолу — в посока по диагонала наляво или по диагонала надясно.

Пример:

Вход	Изход
0 7 0 0 0 0 0 3 8 0 0 0 0 8 1 0 0 0 0 2 7 4 4 0 0 4 5 2 6 5	30

Подсказки:

За да решим задачата в общия случай, нека да запишем данните в двумерен масив $D[i][j]$ с размери $N \times (N + 1)$, $i = 0, 1, \dots, N-1$, $j = 0, 1, \dots, N$. По технически причини, за по-лесно боравене с индексите в програмата, този масив го дефинираме с един стълб в повече, като допълнителният нулев стълб остава зареден с нули. При конкретно зададените числа масивът представя следната таблица от 5 реда и 6 стълба ($N = 5$):



```
0 7 0 0 0 0
0 3 8 0 0 0
0 8 1 0 0 0
0 2 7 4 4 0
0 4 5 2 6 5
```

За всеки елемент $D[i][j]$, включен в дадената триъгълна кон-фигурация от числа, да пресметнем най-голямата стойност $R[i][j]$, която може да се постигне, като се движим според правилата, тръгвайки от върха. Стойностите пресмятаме последователно по редове. Очевидно имаме:

$$R[0][1] = D[0][1];$$

$$R[i][j] = \max\{D[i][j] + R[i-1][j-1], D[i][j] + R[i-1][j]\}$$

за всички $i = 1, \dots, N-1$ и $j = 1, \dots, i+1$. Остава да намерим най-голямата стойност в последния ред на масива R .

Спомагателният масив $R[i][j]$ служи за възстановяване на самия път, по който се достига най-голямата сума от търсения вид.

Задача 4.3. Управление на врата

В един ресторант се срещат N бандити. Бандитът с номер i , $i = 1, 2, \dots, N$ идва в момента от време T_i и носи P_i долара. Входната врата на ресторанта има K състояния, различаващи се по степента на отвореност. Състоянието на вратата може да се променя с една единица за всяка една единица време, т.е. степента на отваряне на вратата или се увеличава с единици, или се намалява с единица, или остава в същото състояние. В началния момент от време вратата е затворена (състояние 0). Бандитът с номер i може да влезе в ресторанта само ако врата-та с отворена специално за него, т. е. когато степента на отвореност на вратата съвпада със степента S_i на пълнота на бандита. При едновременно идване на няколко бандити с еднаква пълнота, съвпадаща със степента на отвореност на вратата, влизат всичките бандити с тази пълнота. Ако в момента на идване на един бандит състоянието на вратата не съвпада със степента на пълнота на бандита, той си отива и повече не се връща. Ресторантът работи в течение на време T . Напишете прог-рама, която да покаже по какъв начин вратата да се отваря или затваря във всяка стъпка от времето, така че в ресторанта да се съберат бандити с максимално количество долари.

Входни данни за задачата са:

- целите числа N , K и T (примерни ограничения: $1 \leq N \leq 100$, $1 \leq K \leq 100$, $0 \leq T \leq 3000$);
- моментите от време на пристигане на бандитите T_1, T_2, \dots, T_N , зададени като цели числа от интервала $[1, T]$.



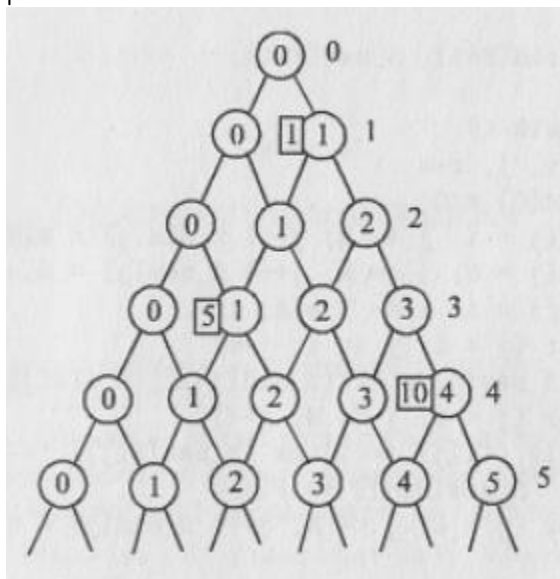
- количеството долари, които носи всеки бандит: P_1, P_2, \dots, P_N , зададени като цели числа (например между 0 и 300).
- степенята на пълнота на всеки от бандитите: S_1, S_2, \dots, S_N , които са цели числа такива, че $1 < S_i < K$ за всяко $i = 1, 2, \dots, N$.

Пример

Вход	Изход
$N=3$ $K=5$ $T=5$ $T_1=3$ $T_2=4$ $T_3=1$ $P_1=5$ $P_2=10$ $P_3=1$ $S_1=1$ $S_2=4$ $S_3=1$	11

Подсказки:

Състоянието на отвореност на вратата изобразяваме като триъгълна решетка. На изображението по-долу са представени входните данни за задачата. Всеки връх определя степенята q на отвореност в момента време t . Моментите от време са отбелязани като числа отгъсно на решетката. При някои от върховете в правоъгълна рамка е дадено количеството долари които има този бандит (с пълнота q), който идва в момента време t . За всеки връх това количество долари ще наричаме тегло на върха. За върховете, които нямат съпоставено количество долари, приемаме, че теглото им е нула. За да решим задачата, трябва да намерим път по решетката, започващ от най-горния връх и минаващ през върхове така, че сумата от теглата на върховете по пътя да е максимална. За дадения при-мер, тази максимална стойност лесно може да бъде намерена и тя е равна на 11.



За програмната реализация ще отбележим, че не е необходимо да се пазят оценките за всеки връх. За да ги намерим в момента t , е необходимо да ползваме само стойностите им в предишния момент $t-1$. Това се осъществява в програмата чрез двата едномерни масива S_{old} и S_{new} . В основния цикъл променливата i пробягва моментите от време. В тялото



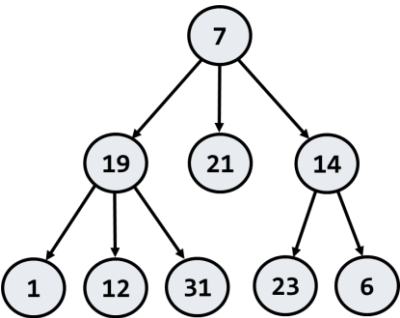
на този цикъл първоначално елементите на масива S_new се зареждат с по-голямата от двете „горни“ стойности. След това програмата преглежда времената на пристигане на бандитите. Ако намери бандит, който пристига в текущия момент ($T[j]=i$) и установи, че съществува степен на отвореност на вратата, съответстваща на пълнотата на бандита ($S_new[S[j]] \neq MINVALUE$), програмата коригира стойността на S_new .

Тема 5. Дървовидни структури от данни

Задача 5.1. Въведение

Дадено е дърво от N възела, представено като набор от $N-1$ двойки възли (възел-родител, възел дете). Реализирайте операциите, описани по-долу:

Пример:

Вход	Коментари	Дърво	Дефиниции
9 7 19 7 21 7 14 19 1 19 12 19 31 14 23 14 6 27 43	$N = 9$ Възли: 7->19, 7->21, 7->14, 19->1, 19->12, 19->31, 14->23, 14->6 $P = 27$ $S = 43$		Корен: 7 Листа: 1, 6, 12, 21, 23, 31 Междинни възли: 14, 19 Най-дълбокия ляв възел: 1 Най-дълъг път: 7 -> 19 -> 1 (дължина = 3) Пътища от суми 27: 7 -> 19 -> 1 7 -> 14 -> 6 Поддървета от суми 43: 14 + 23 + 6

Подсказки:

Създайте двоично дърво за търсене и създайте допълнителен метод за добавяне на възел-дете към конкретен възел-родител.

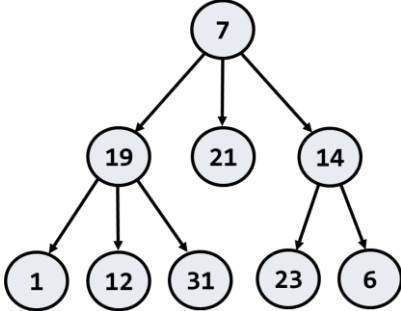
Задача 5.2. Намиране на корен на дърво

Напишете програма, която прочита дърво от N възела, представено като набор от $N-1$ двойки възли (възел-родител, възел дете) и намира корена му:

Пример:

Вход	Изход	Дърво
------	-------	-------



9	Корен: 7	
---	----------	--

Подсказки:

Използвайте рекурсивна дефиниция `Tree<T>`. Пазете стойност, родител и деца за всеки възел от дървото:

```
public class Tree<T>
{
    public T Value { get; set; }
    public Tree<T> Parent { get; set; }
    public List<Tree<T>> Children { get; private set; }

    public Tree(T value, params Tree<T>[] children) {...}
}
```

Променете конструктора на `Tree<T>` така че да може да се присвои родител за всеки възел-geme:

```
public Tree(T value, params Tree<T>[] children)
{
    this.Value = value;
    this.Children = new List<Tree<T>>();
    foreach (var child in children)
    {
        this.Children.Add(child);
        child.Parent = this;
    }
}
```

Използвайте речник, за да пазите колекция от възлите и техните стойности. Това ще ви позволи да намирате много по-бързо възлите на дървото по време на неговото конструиране:



```
public class Program
{
    static Dictionary<int, Tree<int>> nodeByValue = new Dictionary<int, Tree<int>>();

    static void Main()
    {
        // Problem solution
    }
}
```

Напишете метод за намиране на възел от дървото по неговата стойност или ако не съществува да създава нов възел:

```
static Tree<int> GetTreeNodeByValue(int value)
{
    if (!nodeByValue.ContainsKey(value))
    {
        nodeByValue[value] = new Tree<int>(value);
    }

    return nodeByValue[value];
}
```

Създайте метод за добавяне на връзка в дървото.

```
public void AddEdge(int parent, int child)
{
    Tree<int> parentNode = GetTreeNodeByValue(parent);
    Tree<int> childNode = GetTreeNodeByValue(child);

    parentNode.Children.Add(childNode);
    childNode.Parent = parentNode;
}
```

Създайте дървото. Дадени са ви връзките между възлите в дървото (родител + дете). Използвайте речника за да намирате децата и родителите по техните стойности:



```
static void ReadTree()
{
    int nodeCount = int.Parse(Console.ReadLine());
    for (int i = 1; i < nodeCount; i++)
    {
        string[] edge = Console.ReadLine().Split(' ');
        AddEdge(int.Parse(edge[0]), int.Parse(edge[1]));
    }
}
```

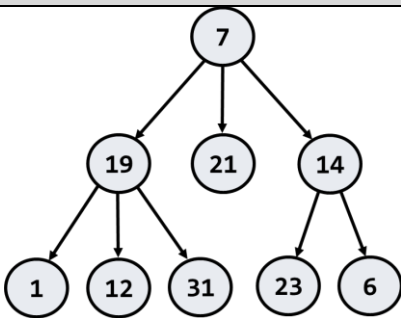
Накрая можете да намерите корена - той няма родител.

```
static Tree<int> GetRootNode()
{
    return nodeByValue.Values
        .FirstOrDefault(x => x.Parent == null);
}
```

Задача 5.3. Отпечатайте дърво

Напишете програма, която прочита дърво от конзолата и го отпечата във следния формат - (всеки елемент на нов ред, като за всяко следващо ниво елементите се отместват с по 2 интервала отместването за предишното):

Пример:

Вход	Изход	Дърво
9	7	
7 19	19	
7 21	1	
7 14	12	
19 1	31	
19 12	21	
19 31	14	
14 23	23	
14 6	6	

Подсказки:

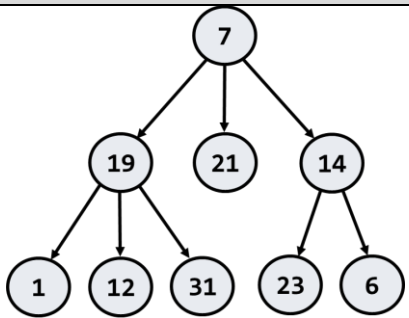


Намерете корена и рекурсивно отпечатайте дървото.

Задача 5.4. Възли - листа

Напишете програма, която намира всички възли-листа и ги отпечата на стандартния изход.

Пример:

Вход	Изход	Дърво
9 7 19 7 21 7 14 19 1 19 12 19 31 14 23 14 6	Листа: 1 6 12 21 23 31	

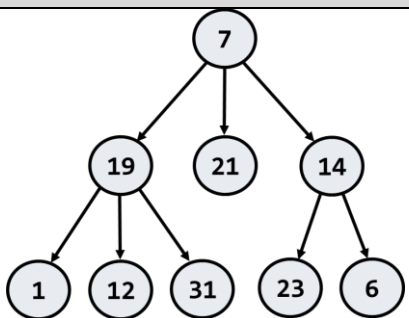
Подсказки:

Намерете всички възли, които нямат деца.

Задача 5.5. Междинни възли

Напишете програма, която прочита дървото и намира всички междинни възли (в нарастващ ред).

Пример:

Вход	Изход	Дърво
9 7 19 7 21 7 14 19 1 19 12 19 31	Междинни възли: 14 19	



14 23		
14 6		

Подсказки:

```
static void PrintMiddleNodes()
{
    var nodes = nodeByValue.Values
        .Where(x => x.Parent != null && x.Children.Count != 0)
        .Select(x => x.Value)
        .OrderBy(x => x)
        .ToList();

    Console.WriteLine("Middle nodes: " + string.Join(" ", nodes));
}
```

Задача 5.6 Най-дълбок възел

Напишете програма, която прочита дърво от конзолата и отпечатва най-дълбокия възел (ако са повече от един с еднаква дълбочина - първия от ляво надясно)

Пример:

Вход	Изход	Дърво
9 7 19 7 21 7 14 19 1 19 12 19 31 14 23	Най-дълбок възел: 1	<pre> graph TD 7((7)) --> 19((19)) 7 --> 21((21)) 7 --> 14((14)) 19 --> 1((1)) 19 --> 12((12)) 19 --> 31((31)) 14 --> 23((23)) 14 --> 6((6)) </pre>



14 6		
------	--	--

Подсказки:

Насоки за решаване на задачата с връзки към материали и екрани с код от Visual Studio...

Задача 5.7. Най-дълъг път

Напишете програма, която прочита дърво от конзолата и отпечатва най-дългия път в него (ако са повече от един с еднаква дължина - първия намерен от ляво надясно)

Пример:

Вход	Изход	Дърво
9 7 19 7 21 7 14 19 1 19 12 19 31 14 23 14 6	Най-дълъг път: 7 19 1	<pre>graph TD; 7((7)) --> 19((19)); 7 --> 21((21)); 7 --> 14((14)); 19 --> 1((1)); 19 --> 12((12)); 19 --> 31((31)); 14 --> 23((23)); 14 --> 6((6));</pre>

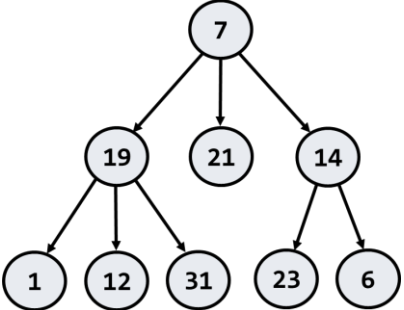
Задача 5.8. Всички пътища с дадена сума

Намерете всички пътища с дадена сума от възлите им (от ляво надясно). Първата стойност от входа са броя възли, втората - сумата, а останалите - връзките между възлите.

Пример:

Вход	Изход	Дърво
------	-------	-------



9	Пътища със сума 27:	
27	7 19 1	
7 19	7 14 6	
7 21		
7 14		
19 1		
19 12		
19 31		
14 23		
14 6		

Задача 5.9. Реализирайте двоично дърво за търсене

Реализирайте двоично дърво за търсене със следната функционалност:

- Добавяне на елемент
- Търсене на елемент
- Премахване на елемент
- Проверка дали даден елемент съществува в дървото

Създайте програма, която прочита от конзолата цяло число N, след което N на брой елементи и ги добавя в двоично дърво за търсене.

Изтрийте от дървото следните елементи:

- Най-големия
- Най-малкия
- най-близкия (със закръгляне нагоре) до средното аритметично на всички елементи.

Отпечатайте дървото на конзолата - всеки елемент на нов ред с отместване по 2 интервала повече за всяко по-дълбоко ниво.

Тема 6. Хеш таблици

Задача 6.1. Прочетете повече за хеш таблиците.

Преди да започнете се запознайте с концепцията за хеш таблица: <https://bg.wikipedia.org/wiki/Хеш-таблица>. Забележете, че съществуват много стратегии за управление на колизиите като свързване на елементи или отворено адресиране. Тук ще използваме една от най-простите стратегии - свързване на елементите в колизия чрез свързани списъци.



Типичните операции в хеш таблица са:

- Добавяне на елемент
- Премахване на елемент
- Проверка дали даден ключ е в таблицата
- Извличане на елемент
- Промяна на елемент
- Обхождане на всички елементи
- Обхождане на всички ключове
- Обхождане на всички стойности
- Извличане на броя на елементите

Задача 6.2. Дефиниране на класове за хеш таблица

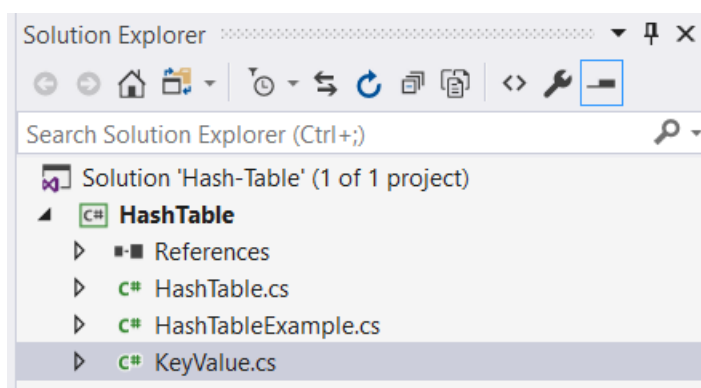
Създайте класове, реализиращи хеш таблица със следната функционалност:

- Добавяне на елемент
- Премахване на елемент
- Проверка дали даден ключ е в таблицата
- Извличане на елемент
- Промяна на елемент

Реализация:

Създайте нов проект (конзолно приложение) във Visual Studio и добавете следните класове:

- `HashTable<TKey, TValue>`
- `KeyValue<TKey, TValue>`



Класът `KeyValue<TKey, TValue>` ще съдържа елемент от хеш таблицата, като ключа на елемента ще бъде от тип `TKey`, а стойността от тип `TValue`.



Два елемента от тип `KeyValuePair<TKey, TValue>` ще бъдат считани за еднакви ако ключовете и стойностите им са еднакви.

Хешът на обект от тип `KeyValuePair<TKey, TValue>` ще бъде получаван от комбинацията между хешовете на ключа и на стойността.

```
using System;

public class KeyValuePair<TKey, TValue>
{
    public TKey Key { get; set; }
    public TValue Value { get; set; }

    public KeyValuePair(TKey key, TValue value)
    {
        this.Key = key;
        this.Value = value;
    }

    public override bool Equals(object other)
    {
        KeyValuePair<TKey, TValue> element = (KeyValuePair<TKey, TValue>)other;
        bool equals = Object.Equals(this.Key, element.Key) && Object.Equals(this.Value, element.Value);
        return equals;
    }

    public override int GetHashCode()
    {
        return this.CombineHashCodes(this.Key.GetHashCode(), this.Value.GetHashCode());
    }

    private int CombineHashCodes(int h1, int h2)
    {
        return ((h1 << 5) + h1) ^ h2;
    }

    public override string ToString()
    {
        return $" [{this.Key} -> {this.Value}]";
    }
}
```

Класът `HashTable<TKey, TValue>` ще съдържа масив от свързани списъци. Този масив ще пази елементите на хеш таблицата. При липса на колизии запълнените клетки на този масив ще съдържат свързан списък с точно един елемент. При идентифициране на колизия съответния свързан списък ще съдържа всички елементи, които са в колизия.



```
using System;
using System.Collections;
using System.Collections.Generic;

public class HashTable<TKey, TValue> : IEnumerable<KeyValuePair<TKey, TValue>>
{
    public int Count { get; private set; }

    public int Capacity[...]

    public HashTable(...)

    public HashTable(int capacity) [...]

    public void Add(TKey key, TValue value) [...]

    public bool AddOrReplace(TKey key, TValue value) [...]

    public TValue Get(TKey key) [...]

    public TValue this[TKey key] [...]

    public bool TryGetValue(TKey key, out TValue value) [...]

    public KeyValuePair<TKey, TValue> Find(TKey key) [...]

    public bool ContainsKey(TKey key) [...]

    public bool Remove(TKey key) [...]

    public void Clear() [...]

    public IEnumerable<TKey> Keys [...]

    public IEnumerable<TValue> Values [...]

    public IEnumerator<KeyValuePair<TKey, TValue>> GetEnumerator() [...]

    IEnumerator IEnumerable.GetEnumerator() [...]
}
```

Задача 6.3. Реализация на конструктора на хеш таблицата

Реализирайте конструктора на хеш таблицата. Неговото предназначение е да задели памет за клетките, които ще съдържат елементите на хеш таблицата. Ще ни бъдат нужни два конструктора:

- Конструктор без параметри
- Конструктор, приемащ 1 целочислен параметър - капацитета на хеш таблицата



Реализация:

```
public const int InitialCapacity = 16;

24 references | 0/23 passing
public HashTable(int capacity = InitialCapacity)
{
    this.slots = new LinkedList<KeyValuePair<TKey, TValue>>[capacity];
    this.Count = 0;
}
```

Тук константата InitialCapacity дефинира началния размер на хеш таблицата - 16 елемента. За съхранение на елементите се използва масив от свързани списъци, за да може да бъде приложена стратегията за разрешаване на конфликти - свързани елементи.

Задача 6.4. Реализация на добавяне на елемент

Реализирайте метода Add(key, value), който добавя елемент в хеш таблицата. Този метод трябва да вземе предвид следните ситуации:

- Добавяне на елемент, който няма колизия със вече съществуващ
- Установяване на колизия при добавяне на елемент
- Установяване на дублиращ се ключ
- Увеличаване на размера на хеш таблицата при нужда (удвояване на размера ѝ, когато сме близо до запълване)

Реализация:

```
public void Add(TKey key, TValue value)
{
    GrowIfNeeded();
    int slotNumber = this.FindSlotNumber(key);
    if (this.slots[slotNumber] == null)
    {
        this.slots[slotNumber] = new LinkedList<KeyValuePair<TKey, TValue>>();
    }
    foreach (var element in this.slots[slotNumber])
    {
        if (element.Key.Equals(key))
        {
            throw new ArgumentException("Key already exists: " + key);
        }
    }
    var newElement = new KeyValuePair<TKey, TValue>(key, value);
    this.slots[slotNumber].AddLast(newElement);
    this.Count++;
}
```

Стартираме с проверка дали хеш таблицата е пълна. При идентифицирано запълване на хеш таблицата трябва да удвоим размера ѝ. Това се прави от метода GrowIfNeeded(). Нека разгледаме поведението на този метод по-късно и за момента го оставим празен:

```
private void GrowIfNeeded()
{
    // TODO: implement this later!
}
```




Следващата стъпка е да намерим клетката в хеш таблицата, която ще държи нашия елемент. Индекса на клетката се изчислява от хеша на ключа. Обикновено за тази цел се използва `GetHashCode()`, наследен от класа `System.Object`, който предоставя изчисление на хеш за въградени и дефинирани от потребител типове в работната рамка .NET. Този метод връща 32 битово число. Тъй като числото ще бъде използвано за индекс в масив, то трябва да е в интервала $[0, \text{размера на масива} - 1]$. Затова делим по модул абсолютната стойност на хеша на размера на таблицата. По този начин винаги получаваме индекс в рамките на размера на хеш таблицата.

```
private int FindSlotNumber(TKey key)
{
    var slotNumber = Math.Abs(key.GetHashCode()) % this.slots.Length;
    return slotNumber;
}
```

След като вече имаме номера на клетката там се съдържа или инстанция на свързан списък или null. И в двата случая в тази клетка трябва да имаме свързан списък, съдържащ всички елементи на хеш таблицата с хеш равен на този на елемента, който добавяме.

Проверяваме свързания списък за елемент с ключ равен на ключа на елемента, който добавяме. Ако намерим такъв - операцията спира и излизаме от метода с изключение. Ако ключа не се дублира, елемента се добавя към свързания списък и увеличаваме броя на елементите в хеш таблицата с 1.

Задача 6.5. Реализация на `memogume GrowIfNeeded()` и `Grow()`

Предназначението на метода `GrowIfNeeded()` е да разпознава дали прагът на запълване на хеш таблицата е достигнат и в този случай да удвои размера ѝ.

Реализация:

В нашия случай капацитета на хеш таблицата ни ще бъде удвоен ако тя е запълнена на 75% или повече и ние се опитаме да добавим нов елемент. В този случай ще бъде извикан методът `Grow()`, който извършва оразмеряването на таблицата.

```
public const float LoadFactor = 0.75f;

1 reference
private void GrowIfNeeded()
{
    if ((float)(this.Count + 1) / this.Capacity > LoadFactor)
    {
        // Hash table loaded too much --> resize
        this.Grow();
    }
}
```



Метода `Grow()` райте интерфейса `IEnumerable<T>`, така че да е възможно итерирането през елементите

Задача 6.6. Реализация на обхождане на всички елементи

Имплементирайте интерфейса `IEnumerable<T>`, така че да е възможно итерирането през елементите на хеш таблицата посредством конструкцията `foreach`.

Реализация:

За да можем да използваме конструкцията `foreach` с потребителски дефинирани колекции в C#, те трябва да имплементират интерфейса `IEnumerable<T>`. Тъй като хеш таблицата съдържа елементи от тип `KeyValue<TKey, TValue>`, трябва да се имплементира интерфейса `IEnumerable<KeyValue<TKey, TValue>`. Този интерфейс задължава реализацията на следните два метода:

```
IEnumerator IEnumerable.GetEnumerator()  
{  
    return this.GetEnumerator();  
}
```

и

```
public IEnumerator<KeyValue<TKey, TValue>> GetEnumerator()  
{  
    foreach (var elements in this.slots)  
    {  
        if (elements != null)  
        {  
            foreach (var element in elements)  
            {  
                yield return element;  
            }  
        }  
    }  
}
```

Първия метод извиква втория, а втория върши реалната работа по предоставяне на елементите. В него се обхождат всички елементи в хеш таблицата и за всеки от свързаните списъци, съхранени там, се обхождат всички елементи. Този метод използва конструкцията `yield return` (генерираща функция), която връща елементите "при поискване". За да научите повече за генериращите функции прочетете [https://en.wikipedia.org/wiki/Generator_\(computer_programming\)](https://en.wikipedia.org/wiki/Generator_(computer_programming)).

Задача 6.7. Реализация на метода `Find(key)`

Реализирайте метод, който търси елемент в хеш таблицата по ключ и връща като резултат стойността му, а в случай, че ключа не е намерен - връща `null`. Алгоритъмът за търсене на елемент в хеш таблицата трябва да бъде с константна сложност.



Подсказка:

Вариант на реализацията с линейна сложност би била обхождане на всички елементи в свързаните списъци от хеш таблицата.

За да може да сведем сложността до константна трябва да намерим позицията на свързания списък, отговарящ за този ключ, да обходим елементите му и да намерим този с ключ равен на ключа, с който търсим.

Задача 6.8. Реализация на методите `Get(key)`, `TryGetValue(key, out value)` и `ContainsKey(key)`

Реализирайте следните методи, като алгоритмите за търсене на елемент в хеш таблицата трябва да бъдат с константна сложност:

- `Get(key)` - намира елемент по ключ и връща като резултат стойността му. Ако ключа не бъде намерен, метода хвърля изключение.
- `TryGetValue(key, out value)` - намира елемент по ключ:
 - ако елементът е намерен метода връща като резултат `True`, а стойността на намерения елемент се записва в изходния параметър `value`
 - ако елементът не бъде намерен, метода връща `False`
- `Contains(key)` - търси конкретен ключ в хеш таблицата и връща булев резултат - ако ключа е намерен метода връща `True` иначе `False`.

Подсказка:

Виж подсказката на Задача 7.

Задача 6.9. Реализация на метода `AddOrReplace(key, value)`

Реализирайте метод, който търси елемент в хеш таблицата по ключ и:

- ако ключа съществува в таблицата променя стойността му със стойността на параметъра `value`
- ако ключа не съществува в таблицата - създава нов елемент и го добавя

Подсказка:

Решението е комбинация на Заг. 7 и Заг. 4.

Задача 6.10. Реализация на `Indexer this[key]`

Реализирайте `Indexer` в класа `HashTable`, като `get` секцията му търси елемент в хеш таблицата по зададен ключ, а `set` секцията му променя стойността на елемент, съхранен на даден ключ. Ако ключа не бъде намерен в таблицата се хвърля изключение.

Подсказка:

Синтаксиса на `Indexer` е:



```
public TValue this[TKey key]
{
    get
    {
        // TODO: return the value by key
    }
    set
    {
        // TODO: add or replace the value by key
    }
}
```

Решението е комбинация на Заг. 7 и Заг. 9.

Задача 6.11. Реализация на Remove(key)

Реализирайте метод за премахване на елемент от хеш таблицата, който приема като параметър ключа на елемента, който трябва да бъде премахнат и връща boolean стойност - True ако елемента е премахнат успешно и False ако не е намерен.

Подсказка

За да се премахне елемент по ключ, трябва да бъде намерен свързания списък, в който се намира елемента, след което да се премахне елемента от свързания списък.

Задача 6.12. Реализация на Clear()

Реализирайте метод, който премахва всички елементи на хеш таблицата.

Подсказка:

Можете да заделите нова памет за хеш таблицата и да върнете брояча на елементите на нула по същия начин, по който това е направено в конструктора.

Задача 6.13. Реализация на свойствата за достъп до всички ключове и всички стойности - Keys и Values

Реализирайте свойства Keys и Values в класа HashTable. Двете свойства трябва да имат единствено get секции, в които да се връща колекция от съответно всички ключове или всички стойности на елементите в хеш таблицата.

Подсказка:

Има различни начини да построите колекцията от ключове и колекцията от стойности.

Един начин да върнете всички ключове е да направите колекция List<TKey> и в нея последователно да добавяте ключовете на елементите от хеш таблицата докато ги обхождате. Аналогично за стойностите.

Друг начин е да използвате Linq extension методи по следния начин:



```
public IEnumerable<TKey> Keys
{
    get { return this.Select(element => element.Key); }
}

3 references | 0/1 passing
public IEnumerable<TValue> Values
{
    // TODO: similar to Keys --> just select the Key from all hast table elements
}
```

Задача 6.14 Преброяване на символи

Напишете програма, която прочита текст от клавиатурата и преброява употребата на всеки символ в него. Уникалните символи се отпечатват в азбучен ред, а срещу всеки от тях се отпечатва цяло число - колко пъти той се среща в текста.

Input	Output
Coding rocks	: 1 time/s C: 1 time/s c: 1 time/s d: 1 time/s g: 1 time/s i: 1 time/s k: 1 time/s n: 1 time/s o: 2 time/s r: 1 time/s s: 1 time/s

Input	Output
Did you know Math.Round rounds to the nearest even integer?	: 9 time/s .: 1 time/s ?: 1 time/s D: 1 time/s M: 1 time/s R: 1 time/s a: 2 time/s d: 3 time/s e: 7 time/s g: 1 time/s h: 2 time/s i: 2 time/s k: 1 time/s n: 6 time/s o: 5 time/s r: 3 time/s s: 2 time/s t: 5 time/s u: 3 time/s v: 1 time/s w: 1 time/s y: 1 time/s

Задача 6.15. Телефонен указател

Напишете програма, която прочита от клавиатурата имена и телефони на контакти от телефонен указател.

Формата на името и телефона са както следва: {име}-{телефон}



Попълването на телефонния указател приключва при въвеждане на командата search

След въвеждане на командата search, програмата ви трябва да може да търси контакт по име. При въвеждане на име, програмата извежда на екрана името на контакта и телефонния му номер, а ако не намери контакта - връща "Contact {name} does not exist!".

Програмата приключва изпълнение при въвеждане на команда end.

Input	Output
Peter-0888080808 search Mariika Peter end	Contact Mariika does not exist. Peter -> 0888080808
Peter-+359888001122 Stamat(Gosho)-666 Gero-5559393 Simo-02/987665544 search Simo simo Stamat Stamat(Gosho) end	Simo -> 02/987665544 Contact simo does not exist. Contact Stamat does not exist. Stamat(Gosho) -> 666

Тема 7. Алгоритми в графи

Задача 7.1. Разстояние между върховете

Даден е ориентиран граф и съответните свързващи двойки върхове. Да се намери най-краткото разстояние между двойките върхове или -1, ако няма път, който ги свързва.

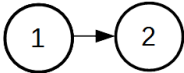
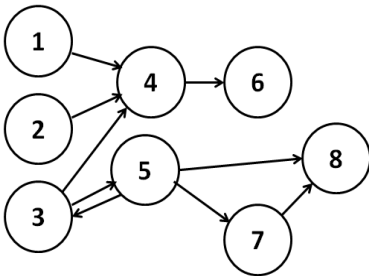
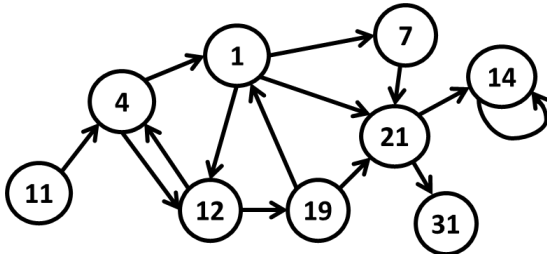
На първия ред се въвежда N - броя на върховете на графа.

На втория ред се въвежда едно цяло число P - броя на двойките върхове, между които ще се търси минималното разстояние. На следващите N реда върховете на графа, а на следващите P реда - двойките върхове.

Пример:

Вход	Визуализация	Изход
------	--------------	-------



2 2 1:2 2: 1-2 2-1		{1, 2} -> 1 {2, 1} -> -1
8 4 1:4 2:4 3:4 5 4:6 5:3 7 8 6: 7:8 8: 1-6 1-5 5-6 5-8		{1, 6} -> 2 {1, 5} -> -1 {5, 6} -> 3 {5, 8} -> 1
9 8 11:4 4:12 1 1:12 21 7 7:21 12:4 19 19:1 21		{11, 7} -> 3 {11, 21} -> 3 {21, 4} -> -1 {19, 14} -> 2 {1, 4} -> 2 {1, 11} -> -1 {31, 21} -> -1 {11, 14} -> 4



21:14 31		
14:14		
31:		
11-7		
11-21		
21-4		
19-14		
1-4		
1-11		
31-21		
11-14		

Подсказки:

За всяка двойка използвайте BFS, за да намерите всички пътища от началния до крайния връх.

Задача 7.2. Области в матрица

Дадена е матрица с размери $N \times M$, чиито стойности са малки латински букви. Две клетки са съседни, ако те имат обща стена. Напишете програма, която намира свързаните области на съседните клетки, съдържащи една и съща буква. Програмата да извежда общия брой области и броя на зоните за всяка латинска буква, подредени лексикографски.

На първия ред се посочва броят на редовете.

Пример:

Вход	Визуализация	Исход																																																
6 aaccsaac baaaaacc baabaccc bbdaaccc ccdcccc ccdcccc	<table><tr><td>a</td><td>a</td><td>c</td><td>c</td><td>c</td><td>a</td><td>a</td><td>c</td></tr><tr><td>b</td><td>a</td><td>a</td><td>a</td><td>a</td><td>c</td><td>c</td><td>c</td></tr><tr><td>b</td><td>a</td><td>a</td><td>b</td><td>a</td><td>c</td><td>c</td><td>c</td></tr><tr><td>b</td><td>b</td><td>d</td><td>a</td><td>a</td><td>c</td><td>c</td><td>c</td></tr><tr><td>c</td><td>c</td><td>d</td><td>c</td><td>c</td><td>c</td><td>c</td><td>c</td></tr><tr><td>c</td><td>c</td><td>d</td><td>c</td><td>c</td><td>c</td><td>c</td><td>c</td></tr></table>	a	a	c	c	c	a	a	c	b	a	a	a	a	c	c	c	b	a	a	b	a	c	c	c	b	b	d	a	a	c	c	c	c	c	d	c	c	c	c	c	c	c	d	c	c	c	c	c	Области: 8 Буква 'a' -> 2 Буква 'b' -> 2 Буква 'c' -> 3 Буква 'd' -> 1
a	a	c	c	c	a	a	c																																											
b	a	a	a	a	c	c	c																																											
b	a	a	b	a	c	c	c																																											
b	b	d	a	a	c	c	c																																											
c	c	d	c	c	c	c	c																																											
c	c	d	c	c	c	c	c																																											



3 aaa aaa aaa	<table><tr><td>a</td><td>a</td><td>a</td></tr><tr><td>a</td><td>a</td><td>a</td></tr><tr><td>a</td><td>a</td><td>a</td></tr></table>	a	a	a	a	a	a	a	a	a	Области: 1 Буква 'a' -> 1																																				
a	a	a																																													
a	a	a																																													
a	a	a																																													
5 asssaadas adsdasdad sdsdadsas sdasdsdsa ssssasddd	<table><tr><td>a</td><td>s</td><td>s</td><td>s</td><td>a</td><td>a</td><td>d</td><td>a</td><td>s</td></tr><tr><td>a</td><td>d</td><td>s</td><td>d</td><td>a</td><td>s</td><td>d</td><td>a</td><td>d</td></tr><tr><td>s</td><td>d</td><td>s</td><td>d</td><td>a</td><td>d</td><td>s</td><td>a</td><td>s</td></tr><tr><td>s</td><td>d</td><td>a</td><td>s</td><td>d</td><td>s</td><td>d</td><td>s</td><td>a</td></tr><tr><td>s</td><td>s</td><td>s</td><td>s</td><td>a</td><td>s</td><td>d</td><td>d</td><td>d</td></tr></table>	a	s	s	s	a	a	d	a	s	a	d	s	d	a	s	d	a	d	s	d	s	d	a	d	s	a	s	s	d	a	s	d	s	d	s	a	s	s	s	s	a	s	d	d	d	Области: 21 Буква 'a' -> 6 Буква 'd' -> 7 Буква 's' -> 8
a	s	s	s	a	a	d	a	s																																							
a	d	s	d	a	s	d	a	d																																							
s	d	s	d	a	d	s	a	s																																							
s	d	a	s	d	s	d	s	a																																							
s	s	s	s	a	s	d	d	d																																							

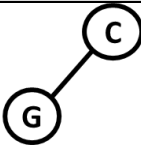
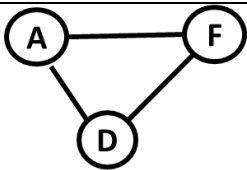
Подсказки:

В началото маркирайте всички клетки като непосетени. Стартирайте рекурсивен DFS (или BFS) от всяка непосетена клетка и маркирайте всички достигнати клетки като посетени. Всяко DFS (или BFS) преминаване ще намери една от свързаните области.

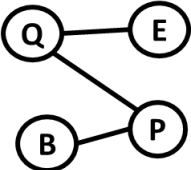
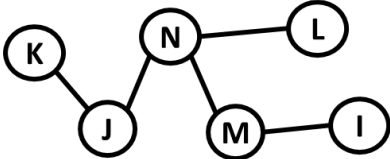
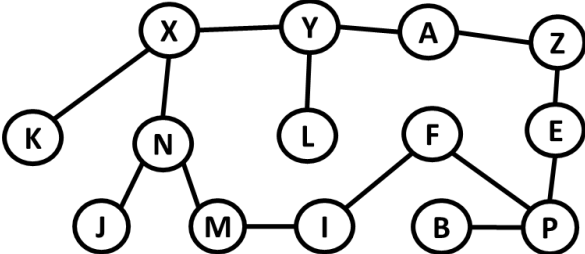
Задача 7.3. Цикъл в граф

Напишете програма, която проверява дали даден неориентиран граф съдържа цикли или е ацикличен.

Пример:

Input	Picture	Output
C-G		Ациклична: Yes
A-F F-D D-A		Ациклична: No



E-Q Q-P P-B		Ациклична: Yes
K-J J-N N-L N-M M-I		Ациклична: Yes
K-X X-Y X-N N-J M-N A-Z B-P I-F A-Y Y-L M-I F-P Z-E P-E		Ациклична: No

Подсказки:

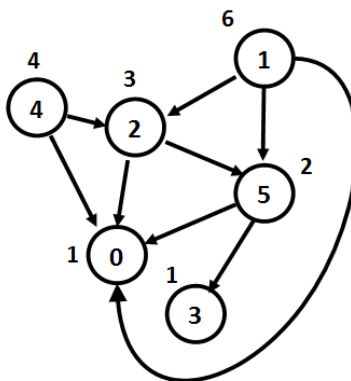
Модифицирайте алгоритъма за топологично сортиране.

Задача 7.4. Заплата

Между служителите в фирма X има йерархия. Служителите могат да имат един или няколко преки началника. Хората, които не управляват никой, се наричат редовни служители и заплатата им е 1. Хората, които управляват поне един служител, се наричат мениджъри. Всеки мениджър взема заплатата, равна на сумата от заплатите на своите пряко управлявани служители.



Мениджърите не могат управляват пряко или косвено себе си . Някои служители може да нямат мениджър (като големия шеф). Вижте примерна йерархия във фирма, заедно с изчислените заплати, следвайки гореописаното правило:



И така: служителите 0 и 3 са редовни служители и взимат заплатата 1. Всички останали са мениджъри и вземат сумата от заплатите на своите пряко управлявани служители. Например, мениджър 1 взема заплатата $3 + 2 + 1 = 6$ (сума от заплатите на служителите 2, 5 и 0). Служителите 4 и 1 нямат мениджър.

В задачата се знаят броя на служителите - N на брой, те са с индекси от 0 до $N - 1$. За всеки служител има информация във вид на низ от символите „Y“ или „N“ в зависимост от това дали настоящият служител е пряк мениджър на служителя i или не.

Съвет: намерете връх, който няма родител и стартирайте DFS, за да изчислите заплатите по дървото рекурсивно.

Вход:

- Входните данни се въвеждат в конзолата
- На първия ред се въвежда стойността на N - цяло число
- На следващите N реда се въвеждат низове от двата символа 'Y' или 'N' с дължина N .

Изход:

- Изходните данни се извеждат в конзолата на един ред.
- На изхода се извежда едно число - сумата от заплатите на всички служители

Ограничения:

- N - цяло число в интервала $[1 \dots 50]$.
- За всеки i -ти ред, i -тия символ е 'N'.
- Ако служител A е мениджър на служител B , B не може да бъде мениджър на A .
- Време за работа на програмата (time limit): 0.1 секунди. Използвана памет:(memory limit) 16 MB.



Пример:

Вход	Изход	Пояснения
1 N	1	Само 1 служител със заплата 1.
4 NNYN NNYN NNNN NYYN	5	Имаме 4 служителя. 0, 1, и 3 са мениджъри на 2. 3 е мениджър на 1. Тогава: заплата(2) = 1 заплата(0) = заплата(2) = 1 заплата(1) = заплата(2) = 1 заплата(3) = заплата(2) + заплата(1) = 2
6 NNNNN N YNYNN Y YNNNN Y NNNNN N YNYNN N YNNYN N	17	

Задача 7.5. Прекъсване на цикли

Даден е неориентиран мултиграф. Задачата да отстраните минимален брой ръбове, така, че да направите графа ацикличен. (т.е. Трябва да прекъснете всички цикли). На изхода е необходимо да отпечатате броя на отстранените ръбове и кои точно са те. Ако няколко ръба могат да бъдат премахнати, за да се прекъсне определен цикъл, премахнете най-малкия от тях по азбучен ред (най-малкият начален връх и най-малкият краен връх по лексикографска наредба).



Пример:

Вход	Визуализация	Изход	Визуализация на графа след отстраняването на ребрата
1 -> 2 5 4 2 -> 1 3 3 -> 2 5 4 -> 1 5 -> 1 3 6 -> 7 8 7 -> 6 8 8 -> 6 7		Брой на премахнатите ребра: 2 1 - 2 6 - 7	
K -> X J J -> K N N -> J X L M X -> K N Y M -> N I Y -> X L L -> N I Y I -> M L A -> Z Z Z Z -> A A A F -> E B P E -> F P P -> B F E B -> F P		Брой на премахнатите ребра: 7 A - Z A - Z B - F E - F I - L J - K L - N	



Подсказки

- Избройте ребрата $\{s, e\}$ по лексикографска наредба. За всяко ребро $\{s, e\}$ проверете дали то затваря някакъв цикъл. Ако отговорът е да - премахнете го.
- За да проверите дали дадено ребро $\{s, e\}$ затваря някакъв цикъл, временно премахнете реброто $\{s, e\}$ и след това опитайте да намерите път от s до e , използвайки DFS или BFS.



Съдържание

Модул 10. Алгоритми и структури от данни.....	1
Тема 1.. Алчни алгоритми	1
Задача 1.1. Задача за възлагане на дейности	1
Задача 1.2. Задача за магнитната лента	2
Задача 1.3. Задача за подреждане на работа в срок.....	4
Задача 1.4. Задача за множества.....	6
Тема 2. Рекурсия, пълно изчерпване и търсене с връщане назад.....	8
Задача 2.1. Разходката на коня.....	8
Задача. 2.2. Пътища в лабиринт	9
Задача. 2.3. Ханойски кули.....	10
Тема 3. Комбинаторни алгоритми	13
Задача 3.1. Пермутации	13
Задача 3.2. Сума нула.....	14
Задача 3.3. Разбиване на число (като сума от числа).....	14
Задача 3.4. Разбиване на число (като произведение от числа).....	16
Задача 3.5. Разбиване на числа (като сума от дадени числа)	16
Тема 4. Динамично оптимиране	17
Задача 4.1. Алея.....	17
Задача 4.2. Триъгълник от числа	18
Задача 4.3. Управление на врата.....	19
Състоянието на отвореност на вратата изобразяваме като триъгълна решетка. На изображението по-долу са представени входните данни за задачата. Всеки връх определя степеня q на отвореност в момента време t . Моментите от време са отбелязани като числа отгясно на решетката. При някои от върховете в правоъгълна рамка е дадено количеството долари които има този бандит (с пълнота q), който идва в момента време t . За всеки връх това количество долари ще наричаме тегло на върха. За върховете, които нямат съпоставено количество долари, приемаме, че теглото им е нула. За да решим задачата, трябва да намерим път по решетката, започващ от най-горния връх и минаващ през върхове така, че сумата от теглата на върховете по пътя да е максимална. За дадения при-мер, тази максимална стойност лесно може да бъде намерена и тя е равна на 11.	20
Тема 5. Дървовидни структури от данни	21



Задача 5.1. Въведение	21
Задача 5.2. Намиране на корен на дърво	21
Задача 5.3. Отпечатайте дърво.....	24
Задача 5.4. Възли - листа.....	25
Задача 5.5. Междинни възли	25
Задача 5.6 Най-дълбок възел.....	26
Задача 5.7. Най-дълъг път	27
Задача 5.8. Всички пътища с дадена сума	27
Задача 5.9. Реализирайте двоишно дърво за търсене	28
Тема 6. Хеш таблици	28
Задача 6.1. Прочетете повече за хеш таблиците.....	28
Задача 6.2. Дефиниране на класове за хеш таблица	29
Задача 6.3. Реализация на конструктора на хеш таблицата	31
Задача 6.4. Реализация на добавяне на елемент	32
Задача 6.5. Реализация на методите GrowIfNeeded() и Grow()	33
Задача 6.6. Реализация на обхождане на всички елементи	34
Задача 6.7. Реализация на метода Find(key)	34
Задача 6.8. Реализация на методите Get(key), TryGetValue(key, out value) и ContainsKey(key)	35
Задача 6.9. Реализация на метода AddOrReplace(key, value).....	35
Задача 6.10. Реализация на Indexer this[key]	35
Задача 6.11. Реализация на Remove(key).....	36
Задача 6.12. Реализация на Clear().....	36
Задача 6.13. Реализация на свойствата за достъп до всички ключове и всички стойности - Keys и Values	36
Задача 6.14 Преброяване на символи.....	37
Задача 6.15. Телефонен указател.....	37
Тема 7. Алгоритми в графи.....	38
Задача 7.1. Разстояние между върховете.....	38
Задача 7.2. Области в матрица	40
Задача 7.3. Цикъл в граф.....	41
Задача 7.4. Заплата.....	42
Задача 7.5. Прекъсване на цикли.....	44