Проектиране и Анализ на Компютърни Алгоритми – част 1
Теми на курсовите проекти

Версия: **1.0**
Последна промяна: **11 януари 2003**

Изисквания

Всяка задача трябва да бъде реализирана на един от програмните езици C, C++, Java, C#, Pascal или Perl. Трябва да може да се компилира на стандартен за съответния език компилатори (най-добре избягвайте да използвате възможности, специфични само за вашия компилатор).

Изискванията към студентите относно курсовия проект

- Да предоставят право на преподавателският екип и университета да използва кода на проекта за образователни и научни цели, включително да го публикува и модифицира.
- Предадат на хартиен носител разпечатка на кода на програмата и нейната документация (без примерните тестове).
- Да предадат архивен файл (на някой от популярните компресиращи програми) чието съдържание е описано по-долу с име факултетния номер на студента със формат:

  `fn<факултетен номер>_<име на проблема>.<разширение на архива>`

  Пример: `fn42955_maxpath.zip`.
  Архивният файл не трябва да превишава 500 KB.

Архивът трябва да не съдържа директории и да съдържа единствено

- Сорс-кода на вашата програма – трябва да бъде именуван със съответното име на проблема плюс стандартното разширение за съответният език (например `maxpath.cpp`). Поради спецификата на изложените проблеми се позволява да се използва само един файл с код. Изключение правят интерактивните програми (с визуализация) за които ще е позволено да имат повече от един файл с код и поради факта че ще се тестват ръчно.
- Проектен файл, или скрипт който да компилира кода в зависимост от използваните средства за разработка.
- Изпълним за Win32 платформата .exe файл (или .class файл за Java програмите) с името на съответния проблем (например `maxpath.exe`). За програмите на Perl при невъзможност да се представи `.exe` файл (например защото това изисква комерсиален софтуер) да се предостави скриптов файл (например `maxpath.bat`) който извиква Perl интерпретатора с необходимите за изпълнението на програмата настройки.
- Описание на реализирания алгоритъм, оценка на сложностите в най-лошия и в средния случай, както и специфични за проблема/реализацията бележки, ако са необходими. Документацията трябва да бъде около 1 – 2 страници и да бъде в един от форматите HTML, DOC (MS Word, Open Office) или RTF с името на проблема (например `maxpath.rtf`).
- 3 примерни теста, които показват, че вашата програма работи правилно. Тестовете трябва да се състоят от номерирани двойки входен + изходен файл и да спазват изискванията за формата на входа и изхода. Имената на примерните тестове трябва да са като имената на входния и изходния файл за съответната задача, но с добавен номер след тях (напр. за задача "perm" това трябва да са файловете perm1.inp, perm1.out, perm2.inp, perm2.out, perm3.inp и perm3.out). Вашите примерни файлове могат да бъдат използвани за тестване на решенията на ваши колеги., така че не трябва да са тривиално лесни (все пак внимавайте за

Изискванията за самата програма

- Да решава точно и единствено зададената задача без да извършва други действия (например да променя системният часовник, да чете/пише от/по екрана или изтрива файловете от текущото устройство).
- Да спазва стриктно формата на входните и изходните данни.

- Да бъде написана четливо. Сорс-кодът трябва да е лесен за разбиране и проследяване как работи. Това съответно означава да се именуват променливите и подпрограмите с подходящи имена, да се слагат коментари на местата с тежка логика, кодът да се подравнява (форматира) по подходящ начин, сложните подпрограми да се разделят на няколко по-прости, да се пише по най-много от 1 израз на ред, да се минимизира обхватът на променливите, да се използва слаба връзка между подпрограмите (loose coupling) и силна вътрешна здравина (strong cohesion) т.н. (вж. курса по качествен програмен код – http://codecourse.sourceforge.net).
- Да работи под Win32 платформа с инсталирани .NET Framework 1.1, JDK 1.4 и ActiveState ActivePerl 5.6 с настройки по подразбиране.
- Предложеният изпълним файл да може да се получи от предложения сорс-код при компилацията му с използване само на стандартните за компилатора библиотеки (да не се използват нестандартни библиотеки).
- Да генерира винаги еднакъв изход при еднакви входни данни (ако се използва генератор на псевдослучайни числа, трябва да се внимава да се генерират винаги еднакви редици от числа).
- Изходът, който се генерира да е резултат от изчислителен процес, реализиран в програмата. Не е позволено програмата просто да отпечатва някакви предварително изчислени данни (например ако задачата има малко възможности за входни данни).
- Да чете точно и единствено от файла с входните данни за съответния проблем (например `maxpath.inp`) и да пише единствено във файла с изходните данни (например `maxpath.out`) в текущата директория. Четене и писане от други файлове и директории не се позволява (включително и използването на временни файлове).
- Да използва не повече от 64MB оперативна памет.

Компилатори и среди, с които ще се тества

- Microsoft Visual C++ 7.1 (VS.NET 2003).
- Microsoft Visual C++ 6.0.
- Borland Delphi 6.0.
- Microsoft .NET Framework 1.1.
- Sun JDK 1.4.
- ActiveState ActivePerl 5.6.

Ако реализирате проекта си с различен от изброените компилатори, след като го завършите, го прехвърлете на някой от изброените. Те могат да се намерят в компютърните зали на ФМИ. Посоченият брой точки за задачата е максимален и ще бъде присъден само ако решението е наистина правилно и достатъчно ефективно. За неоптимални решения ще се дава частичен брой точки.

Други изисквания

Не се разрешава директно използване на чужд сорс-код от книги, статии, Интернет или други източници! Позволено е използването на алгоритми от всякакви източници, стига студентът да ги разбира в детайли и да е написал сорс-кода на програмата си собственоръчно от първия до последния ред. При установяване, че даден студент не разбира достатъчно добре собствената си програма или използвания в нея алгоритъм, работата му ще бъде анулирана!

Преподавателският екип ще оценява единствено програми които спазват стриктно правилата описани по-горе. Особено важно е да се спазват стриктно имената и форматите на входните и изходните файлове (поради автоматизирания процес на тестване, който ще бъде приложен).

Преподавателският екип ще санкционира студенти които не спазват указаните или подразбиращите се изисквания – при злонамерени действия породени от вашите програми виновните ще се наказват по административен път.

Защита на проектите

Проектите ще бъдат проверявани в присъствие на студента, който трябва да ги защитава пред преподавателския екип. Студентът трябва да убеди проверяващите, че е автор на програмата, която е предал, чрез отговаряне на въпроси по сорс-кода и използваните алгоритми. Ако се установи, че студентът е преписвал от някъде, работата му ще бъде анулирана!

Предварителна проверка

Ден преди защитата програмите на студентите ще бъдат тествани автоматизирано с подходящ набор от тестови примери. Ще се проверява както коректността на решенията (дали извеждат верни отговори при определени тестове), така и производителността (дали скоростта на работа е приемлива).

Дати за защита

Датите за защитата на проектите са както следва:

- 25 януари (неделя) - 10-16, зала 326
- 22 февруари (неделя) - 10-16, зала 101

Всеки студент трябва да се яви на една от двете дати в удобно за него време между 10 и 16 часа. За да се избегнат опашките пред изпитната зала, първите, които дойдат пред залата, трябва да направят списък, по който ще става влизането.

Студентите трябва да носят на защитата разпечатка на проектите си на хартия (сорс-код и документация, изготвена съгласно изискванията).

Срокове и начин за предаване

Крайният срок за предаване на проектите е 2 дни преди изпитната дата, на която се явява студента, т.е. съответно за първата и втората дати сроковете са:

- до 24 часа на 23 януари (петък)
- до 24 часа на 20 февруари (петък)

Предварителното предаване е необходимо, за да може журито да прегледа кода и да провери задачите чрез автоматизирано тестване. Моля спазвайте стриктно формата на входа и изхода. За предаването на проектите е създадена специална Web-базирана форма на сайта на курса.

Относно задачите

Задачите са разделени на 3 групи. Всеки студент има право да избира между 3 лесни, 2 средно трудни и 8 трудни задачи, които са определени според факултетният му номер по специална формула. Разпределението е публикувано на сайта на курса. Всяка задача е оценена по трудност с число – максималният брой точки който може да се спечели от нея. Този брой ще се присъжда само при много добро, вярно и ефективно решение, а в останалите случаи ще се присъжда част от него. Студент, който е решил правилно задачи за над 35 точки може да получи най-много до 35.

Лесни задачи

Задача 1. Suffix – Строене на Suffix Array – 15т.

След като мита че СтанчоСорт (Виж зад. Fastsort) е ултра-мега-бърз алгоритъм за сортиране бил разбит (оказало се че тестове са много специфични и СтанчоСорт е просто един от известните алгоритми за сортиране чрез трансформация комбиниран с бъглива реализация на сортирането чрез вмъкване /която се извиквала по 3 пъти за да работи правилно/), Станчо отново се оттеглил и решил да покаже на света че ще измисли нещо велико. Той вече имал база на която да стъпи – знаел сортиране чрез вмъкване и още един алгоритъм който не бил сигурен как се казва. За това решил да разработи нещо подобно и отново потънал из старата библиотека. Чел цял месец – прочел 4 книги за алгоритми и нищо не разбрал (освен сортирането чрез вмъкване което срещнал в едната). Изведнъж на средата на 5-тата той успял да разбере за какво става дума в едната глава. Тя била книга за алгоритми върху низове – нещо което му звучало доста познато. Това което горе долу разбрал било Suffix Array.

Нека е даден символен низ с N символа. Тъй като в книгата реализациите са били на чисто C ще приемаме позициите в низа са числата от 0 до $N-1$ а на позиция N стои специален символ с код 0 който не се среща никъде преди нея. Тогава suffix или наставка на низа от позиция i разбираме последователността от позиция i до позиция N включително (този низ пак има специален символ с код 0 накрая и за това дължината му е $N-i$). Тогава suffix array представлява масив съдържащ всички $N+1$ suffix-а на низа така че те да са представени с позициите от които започват но да бъдат сортирани лексикографски по самите символи в тях. Например за низът “alabala” наставките ще се сортират по

следният начин (“”, “a”, “abala”, “ala”, “alabala”, “bala”, “la”, “labala”) и тогава suffix array-ът на низът ще бъде (7, 6, 2, 4, 0, 3, 5, 1).

В книгата били предложени някои доста добри алгоритми за строене на suffix array. Станчо обаче не ги разбрал.

След известно време обаче той решил да да опита да комбинира алгоритмичните си знания с бутилка минерална вода. За съжаление обаче нищо не се получило. Тогава му хрумнала гениалната идея да се опита да комбинира самите знания. Все пак си помогнал с бутилка минерална вода. Изсипал знанията вътре, затворил, раздрусал и я ударил на екс. Изведнъж изпаднал в момент на гениалност и измислил алгоритъм за строене на suffix array.

При строене на suffix array скоростта много пада от факта че сравнението на два низа може да изисква време пропорционално на дължината на по-късия тъй като може да имат много дълги съвпадащи начални части (префикси). Сравнението обаче може да се ускори с помощта на следният факт: ако символите на позиция i и позиция j са равни то отношението на suffix-ите от позиции i и j е същото като отношението на suffix-ите от позиции $i+1$ и $j+1$. Тогава ако започнем от нарастващ масив (наредени числата от 0 до N) и приложим обърнато сортиране чрез вмъкване (вмъкваме към края на масива отляво на дясно вместо към началото) като във всеки момент помним за сортираната вече част от масива всеки suffix на коя позиция се намира то ще построим целият suffix array.

От вас се иска да разберете и реализирате гореописаното. Suffix array е реална структура от данни която се ползва за низови проблеми и за нея съществуват доста добри алгоритми за строене и използване, но ние сега не се интересуваме от тях.

Входни данни:

Входните данни са дадени в текстов файл с име `suffix.inp`. На единственият му ред се съдържа символен низ от букви и знаци (без специални символи като например интервал). Дължината на низа няма да надхвърля 30000 символа.

Резултат:

Резултатът трябва да се запише в текстов файл с име `suffix.out`. Той трябва да съдържа един ред съдържащ числата от Suffix Array-а на дадения низ разделени с интервал.

Примерен входен файл:

```
alabala
```

Примерен резултатен файл:

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

Задача 2. Cocktail – Коктейл на катедрата – 10т.

Катедрата на Станчо често организира коктейли заради научните й успехи. Интересното за неговата катедра е, че всички от нея са големи ценители на минералната вода. Станчо също не прави изключение. За всеки коктейл той има план за това каква минерална вода и в какъв ред трябва да пие. На масата са наредени разнообразни марки минерална вода (може да се повтарят). Тъй като членовете на неговата катедра имат проблем с обръщането, особено когато употребяват минерална вода, Станчо може да се движи само от ляво надясно. За да не прави лошо впечатление, той не иска да налива от една бутилка повече от веднъж. Освен от минералната вода Станчо се интересува и от комбинаторика. За това той иска да разбере по колко начина може да изпълни плана си за текущия коктейл. Понеже обикновено в разгара на такива коктейли Станчо губи голяма част от математическите си способности, той се нуждае от програма която да реши задачата.

Входни данни:

Входните данни са дадени в текстов файл с име `cocktail.inp`. На първия ред от файла има някаква последователност от букви от ‘a’ do ‘z’ представляващи съкращения на видовете минерална вода наредени по масата от ляво надясно. (т.е. всеки вид минерална вода се представя уникално с малка латинска буква). Вторият ред също се състои от малки латински букви представляващи програмата на Станчо за тази вечер. Дължината на никой от низовете не надвишава 1000 символа.

Резултат:

Резултатът трябва да се запише в текстов файл с име `cocktail.out`. На единственият ред от изходният файл трябва да се запише броят на начините по които Станчо може да реализира програмата си ако се движи само от ляво на дясно и налива от всяка бутилка най много по-веднъж. (Ако не може да реализира програмата си, този брой трябва да е 0).

Примерен входен файл:

```
aabbab
ab
```

Примерен резултатен файл:

```
7
```

Задача 3. Perm – Пермутации – 10т.

Дадено е мултимножество M. На различните му пермутации са съпоставени естествените числа 0, 1, 2... в съответствие с лексикографската им наредба. Да се проектира и реализира ефективен алгоритъм, който по дадено число P намира P-тата по лексикографски ред пермутация на мултимножеството M. Да се оцени сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Входните данни се задават в текстов файл с име `perm.inp`. На първия ред на входния файл стои едно число K – брой пермутации, които програмата трябва да възстанови по поредния им номер ($1 \leq K \leq 1000$). Следват K двойки редове. На първия от всяка двойка има две цели числа N и P, разделени с интервал ($1 \leq N \leq 12$). На втория ред има N числа разделени с интервал описващи зададеното мултимножество.

Резултат:

Резултатът трябва да се запише в текстов файл с име `perm.out`. Файлът трябва да се състои от точно K реда. На всеки ред трябва да има по една пермутация съответстваща на указания й номер във входния файл, записана като последователност от числа, разделени с интервал.

Примерен входен файл:

```
4
4 0
1 2 3 4
3 1
1 2 3
6 718
1 2 3 5 6 4
5 2
1 2 1 2 1
```

Примерен резултатен файл:

```
1 2 3 4
1 3 2
6 5 4 3 1 2
1 1 2 2 1
```

Задача 4. Game21 – Игра 21 – 15т.

Игрална дъска се състои от 21 полета и има следната форма:

```
    X X X
  X X X X X
  X X X X X
  X X X X X
    X X X
```

Всяко поле (означено на фигурата с "X") може или да съдържа стойност или 0 или 1. Върху дъската можем да прилагаме 3 вида операции:

- инвертиране на 9 полета разположени във формата на квадрат с размери 3 x 3;

- инвертиране на 5 полета разположени във формата на кръстче;
- инвертиране на всичките 21 полета.

Инвертиране наричаме промяна на стойността на дадено поле, при която 0 става 1, а 1 става 0. Да се проектира и реализира алгоритъм, който по зададена целева дъска (описана като последователност 21 числа, ред по ред отляво надясно) намира минималния брой операции от 3-те посочени вида, чрез които от дъска, запълнена цялата с 0 се достига до зададената целева дъска. Например дъската (3) може да се получи от началната дъска (0) с 3 на брой преобразувания съгласно описаните по-горе правила:

Да се оцени сложността на предложения алгоритъм в средния и в най-лошия случай.

Входни данни:

Входните данни се задават в текстов файл с име `game21.inp`. На първия ред на входния файл стои едно число K – брой дъски, които програмата трябва да оцени за най-малко колко хода могат да се получат ($1 \leq K \leq 500$). На всеки от следващите K реда има по 21 числа, разделени с интервал, всяко от които е или 0 или 1.

Резултат:

Резултатът трябва да се запише в текстов файл с име `game21.out`. Файлът трябва да се състои от точно K реда. На всеки ред трябва да има по едно цяло число – минимален брой ходове за получаване на съответната дъска или -1 ако съответната дъска не може да се получи от началната.

Примерен входен файл:

```
2
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 1 1
```

Примерен резултатен файл:

```
-1
3
```

Задача 5. Family – Родословно дърво – 10т.

Родствени връзки на планетата ФМИ са доста объркани. Те се събират в групи, така че фмитяните могат да имат както един, така и десет родителя и никои не е учуден от стотина деца. Фмитяните са свикнали с техния начин на живот, и го намират за много естествен, но във планетарния парламент това объркано родословно дърво води до объркване. Там се събират най-достойните фмитяни и затова за да не се засегне някои по време на дискусии е решено първо да се дава думата на най-възрастните фмитяни, след това на по-младите и най-накрая на най-младите и бездетни фмитяни. Разбира се спазването на това решение не е тривиална задача. Не винаги фмитяните познават всичките си родители. Ако по грешка, някой говори преди родителите си или прародителите си, тогава става голям скандал. Задачата е да се напише програма която, да определи реда на изказване в фмитянския парламент, и да прекратят скандалите в веднъж за винаги.

Входни данни:

Входните данни са дадени в текстов файл с име `family.inp`. Първия ред от него съдържа само числото N ($1 <= N <= 2\ 000$) – броя на членовете на фмитянския парламент. Според вековните традиции членовете на парламента са номерирани с естествените числя от 1 до N. Следват точно N реда, като i-тия ред съдържа списък с децата на i-тия член на парламента. Списъка с децата с поредица от номерата на децата в произволен ред разделени с интервали. Списъкът може да е празен. Списъкът, дори и празен, завършва с 0.

Резултат:

Резултатът трябва да се запише в текстов файл с име `family.out`. Той съдържа една единствена линия, списък с членовете на фмитянския парламент по реда на изказване, разделени с интервали. Ако задачата има няколко решения, отпечатайте някое от тях. Поне едно решение винаги съществува.

Примерен вход:

```
4
0
4 1 0
1 0
3 0
```

Примерен резултат:

```
2 4 3 1
```

Задача 6. Teams – Два отбора – 15т.

Дадена е група от N човека. Всеки член на групата има един или повече приятели в нея. Да се напише програма която разделя групата на два отбора, така че всеки член на всеки отбор трябва да има поне един приятел в противниковия отбор.

Входни данни:

Входните данни са дадени в текстов файл с име `teams.inp`. Първия ред от него съдържа само числото N ($N \leq 100$) – броя на членовете в групата. Членовете на групата са номерирани с числата от 1 до N. Следват N реда, като i-тия ред съдържа списък с приятелите на i-тия член на групата. Списъкът завършва с 0. Приятелството е винаги взаимно.

Резултат:

Резултатът трябва да се запише в текстов файл с име `teams.out`. На първия ред от него се съдържа броят на хората в първия отбор или нула ако не е възможно да се раздели групата на два отбора. Ако задачата има решения на втория ред от резултата трябва да се запише списък с хората от първия отбор, разделени с един интервал. Ако задачата има няколко решения, трябва да се отпечата някое от тях.

Примерен вход:

```
5
2 3 0
3 1 0
1 2 4 5 0
3 0
3 0
```

Примерен резултат:

```
3
1 4 5
```

Задача 7. Power – Повдигане на степен – 15т.

Дадени са целите числа N, M и Y. Да се напише програма която намира всички цели числа X в интервала [0, M-1] такива че $X^N \bmod M = Y$.

Входни данни:

Входните данни са дадени в текстов файл с име `power.inp`. Той съдържа само една линия с числата N, M и Y разделени с интервал ($0 < N < 1\,000\,001$, $1 < M < 100\,001$).

Резултат:

Резултатът трябва да се запише в текстов файл с име `power.out`. Отпечатайте в него всички числа за X на една линия, разделени с един интервал, Числата трябва да бъдат отпечатани в нарастващ ред. Ако не съществува X с търсеното свойство отпечатайте -1.

Примерен вход:

```
2 5 1
```

Примерен резултат:

```
1 4
```

Задача 8. Flags – Знамена – 15т.

За международния ден на знамето един продавач решил да си украси магазина със знаменца на бели, залени и червени ивици. Той иска да са изпълнени следните условия:

1. Ивици с един и същи цвят не могат да заемат съседни позиции.
2. Зелената ивица винаги трябва да бъде подставяна между бяла и червена или червена и бяла ивица.

Напишете програма която намира по колко начина може да бъде изпълнено неговото желание.

Входни данни:

Входните данни са дадени в текстов файл с име `flags.inp`. Той съдържа само числото N – броя на ивиците, 1 <= n <= 45.

Резултат:

Резултатът трябва да се запише в текстов файл с име `flags.out`. Отпечатайте в него броя на различните знамена които могат да се получат ако се изпълнени горните две условия.

Примерен вход:

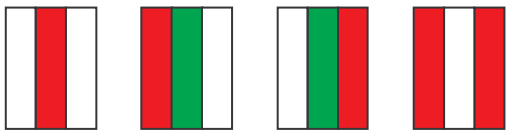
```
3
```

Примерен резултат:

```
4
```

Пример:

Всички допустими конфигурации с 3 ивици:



Задача 9. Election – Избори – 5т.

Наближават следващите избори. Навсякъде са разлепени предизборни плакати и всяка вечер по телевизията ни обещават колко ще се подобри живота ни. На един програмист обаче има много работа преди тези избори. Неговата задача е да напише програма която ще пресметне резултата от изборите. Помогнете му.

Входни данни:

Входните данни са дадени в текстов файл с име `election.inp`. Първия ред от него съдържа числата N (N <= 100 000) – броя на кандидатите и M (M <= 100 000) броя на избирателите. Следващите M реда съдържат по едно число – номера на кандидата за които е гласувал i-тия избирател.

Резултат:

Резултатът трябва да се запише в текстов файл с име `election.out`. Отпечатайте в него N реда всеки от които съдържа процента на избирателите които са гласували за i-тия кандидат, с точност до 2 знака.

Примерен вход:

```
2 3
1
1
2
```

Примерен резултат:

```
66.67%
33.33%
```

Задача 10. Democracy – Демокрацията е в опасност – 10т.

В една малка страна всички решения се вземали с обикновено мнозинство от гласовете на жителите (за щастие те не били много) на общото събрание. Една от местните партии решила да въведе драстични промени в избирателната система, като основния и довод бил че населението на острова се е увеличило в последно време и вече не е рентабилно да се организират общи събрания.
Промените били следните: Всички жители на острова били разделени в K групи (не непременно равни). Гласуването за всеки въпрос става в всяка една от групите. Групата гласува ЗА ако повече от половината членове на групата гласуват ЗА, в противен случай групата гласува ПРОТИВ. Въпроса се приема ако броя на групите гласували ЗА е по-голям от броя на групите гласували ПРОТИВ.
Първоначално жителите на острова приветствали това решени, но след това разбрали някои негови негативи. Станало ясно че партията която е въвела системата могли да повлияят на формирането на групите и след това да приемат всяко решение без да има мнозинство от гласували ЗА. Нека имаме 3 групи съответно от по 7, 7 и 9 човека, тогава за една партия е достатъчно да има само по 4 поддръжника във всяка една от първите две групи за да влияе на всяко едно решение. По този начин само 8 гласа са достатъчни за приемането на всяко решение, вместо 12 при общо гласуване.
Задачата е да се напише програма която установява минималния брой на поддръжници на една партия, достатъчни да контролират всяко гласуване, при някое разпределение на поддръжниците между групите.

Входни данни:

Входните данни са дадени в текстов файл с име `democracy.inp`. Първия ред се съдържа числото K ($K \leq 100\,000$) – броя на групите. На втория ред са K естествени числа разделени с интервал. Това са броя на гласоподаватели във всяка група. С цел да се опрости термина „мнозинство” ще приемем че броя на групите и броя на гласоподавателите във всяка група е нечетно число. Населението на острова не надвишава 1 000 000 000 души.

Резултат:

Резултатът трябва да се запише в текстов файл с име `democracy.out`. Отпечатайте в него единствено естествено число – минималния брой на поддръжници достатъчен да се приеме всяко решение.

Примерен вход:

```
3
7 9 7
```

Примерен резултат:

```
8
```

Задача 11. Amoebas – Амеби – 10т.

Нашите учени наскоро откриха нов вид амеби. Тези създания (амебите, не учените) живеят на колонии в желеобразна хранителна среда. През повечето време тези странни амеби се движат, ако когато две от тях се сблъскат, те умират и на тяхно място се появява нова амеба. След дълги наблюдения нашите учени установили че когато се сблъскат две амеби с тегла m_1 и m_2, то теглото на новополучената амеба е $2\sqrt{m_1 m_2}$. Учените също така скат да знаят минималното тегло до което може да се свие една такава колония. Напишете програма която отговаря на този въпрос. Можем да предположим че 3 и повече амеби никога не се блъскат по едно и също време.

Входни данни:

Входните данни са дадени в текстов файл с име `amoebas.inp`. Първия ред от него съдържа числото N ($1 <= N <= 100$) – броя на амебите в колонията. Следващите N реда съдържат по едно естествено число в интервала от 1 до 10 000 – теглото на съответната амеба.

Резултат:

Резултатът трябва да се запише в текстов файл с име `amoebas.out`. Отпечатайте в него едно единствено число – минималното тегло което може да се получи от колонията, с точност до 2 знака след запетаята.

Примерен вход:

```
2
72
50
```

Примерен резултат:

```
120.00
```

Задача 12. Exam – На изпит – 15т.

Студентите се явяват на изпит. За всеки студент се знае времето, когато е готов за изпитване T_1 (т.е. за колко време си развива въпросите и ги предава), колко минути са необходими за завършване на изпита T2 (т.е. за колко време преподавателят му проверява развитите въпроси и го изпитва по тях) и време когато трябва да е свободен T_3 (T_1 и T_3 се измерват в минути от началото на изпита). По време на изпита има опашка пред преподавателя. Опашката е като на всеки изпит – който свърши по-рано с въпросите си, бива изпитан по-рано, защото се нарежда на по-предна позиция на опашката. Ако преподавателят е зает с някой студент, другите чакат. Възможно е някои студенти да не са свободни в T_3. Възможно е също в някои моменти преподавателят да си почива, ако няма никой на опашката. Преподавателят е добър човек и може да премести началото на изпита по-рано, така че всички студенти да бъдат свободни в съответното за тях T_3 (изчислено спрямо началото на изпита преди да бъде преместено). Вашата задача е да напишете програма която да изчисли колко минути по-рано трябва да се премести изпита. Преместването трябва да е с минимален брой минути.

Входни данни:

Входните данни са дадени в текстов файл с име `exam.inp`. Първия ред от него съдържа числото N ($1 \leq N \leq 100$) – броя на студентите. Следващите N реда съдържат T_1, T_2, T_3 ($0 \leq T_1 \leq T_3 \leq 600$, $1 \leq T_2 \leq 240$). Всички T_1 са различни.

Резултат:

Резултатът трябва да се запише в текстов файл с име `exam.out`. Отпечатайте в него минималният брой минути, с които трябва да се премести началото на изпита или 0 ако студентите имат достатъчно време и преместване не е необходимо.

Примерен вход:

```
4
110 10 120
70 40 150
95 15 400
60 10 160
```

Примерен резултат:

```
15
```

Задача 13. Digitmania – Цифромания – 15т.

Нека означим с S(N) сумата на цифрите на естественото число N. Задачата е да се определи колко често се среща следното равенство: S(A + B) = S(A) + S(B)

Входни данни:

Входните данни са дадени в текстов файл с име `digitmania.inp`. Единствения ред от него съдържа числото K (1 <= K <= 5000).

Резултат:

Резултатът трябва да се запише в текстов файл с име `digitmania.out`. Отпечатайте в него броя на двойките (A, B) където A и B са K-цифрени числа. При броенето имайте на предвид:

1. Числата A и B нямат водещи нули.
2. Двойките (A, B) са наредени, т.е. (12, 13) и (13, 12) се броят като две различни двойки.
3. Изходът би могъл да е огромно число.

Примерен вход:

```
2
```

Примерен резултат:

```
1980
```

Задача 14. Bingame – Двоична игра – 10т.

Двама крале играят една интересна игра. Пред тях на куп има N плочки. Кралете играят един след друг последователно. Всеки от тях, когато е на ход, може да взема плочки от купчината. Двамата имат право да вземат произволен брой плочки, но при условие, че числото е неотрицателна целочислена степен на 2 например 1, 2, 4, 8, 16, 32... Кралят, който вземе последната плочка, печели играта. Ако приемем, че всеки играч играе оптимално, вашата е задача е да напишете програма, която определя кой е победител в играта. Оптимална игра означава, че всеки играч винаги играе най-изгодния за него ход, т.е. този ход, който води към победа за него (когато победа е възможна възможно).

Входни данни:

Входните данни са дадени в текстов файл с име `bingame.inp`. Единствения ред от него съдържа числото N ($N <= 10^{250}$) – броя на плочките.

Резултат:

Резултатът трябва да се запише в текстов файл с име `bingame.out`. Отпечатайте в него 1 ако печели първия играч или 2 ако печели втория. Ако първия е спечелил отпечатайте на нов ред минималния брой плочки които той трябва да вземе на първия си ход.

Примерен вход:

```
4
```

Примерен резултат:

```
1
1
```

Задача 15. Multigame – Мултиигра – 15т.

Вероятно вече знаете играта, в която всеки от двамата играчи взема от 1 до 3 пула от купчина. Губи този които вземе последния пул. Нека обобщим играта. Да предположим че играчите не могат да вземат не 1, 2 или 3 пула, а k_1, k_2, .. k_m пула. Задачата е да се определи кой печели при оптимална игра на двамата играчи, при условие, че играч 1 прави първия ход. За загубил играта се счита този играч, който е направил последния ход, т.е. печели играчът, който е на ред и няма валиден ход. За тази игра е добре известно, че може да се определи оптималният следващ ход без да се знае нищо за предишните ходове.

Входни данни:

Входните данни са дадени в текстов файл с име `multigame.inp`. Първия ред от него съдържа числата N и M ($N <= 1\ 000\ 000$, $M <= 50$) – броят на пуловете и броят на числата k_1, k_2, .. k_m. На втория ред са числата k_1, k_2, .. k_m разделени с интервал.

Резултат:

Резултатът трябва да се запише в текстов файл с име `multigame.out`. Отпечатайте в него 1 ако печели първия играч или 2 ако печели втория.

Примерен вход:

```
17 3
2 1 3
```

Примерен резултат:

```
2
```

Задачи със средна трудност

Задача 21. Maxpath – Максимален път в неориентиран мултиграф – 20т.

Противно на задачата за намиране на минимален път в (мулти)граф задачата за намиране на максимален прост път в (мулти)граф няма толкова ефективно решение. Нашата задача е да намерим максималният прост път в неориентиран мултиграф възможно най-ефективно. Тъй като може да има повече от 1 максимален път се иска да се намери лексикографски най-малкият (по номерата на върховете му). Да се оцени сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Входните данни са дадени в текстов файл с име `maxpath.inp`. На първия ред стоят две цели числа N и M, разделени с интервали, задаващи съответно броя върхове и броя ребра в графа ($1 \leq N \leq 100$). Следват M реда описващи ребрата. Едно ребро се описва от 3 числа, разделени с интервал – начален връх, краен връх и тегло на реброто. Върховете са номерирани с числата от 1 до N. Възможно е да има по няколко ребра между двойка върхове.

Резултат:

Резултатът трябва да се запише в текстов файл с име `maxpath.out`. На първия ред трябва да стоят две числа C и L. C указва броят на върховете участващи в пътя а L теглото на пътя. На всеки от останалите C реда трябва да има по едно число указващо номер на връх от пътя.

Примерен входен файл:

```
8 14
1 2 4
3 2 8
3 1 20
2 3 8
4 5 2
4 6 2
4 8 3
4 7 8
8 7 2
8 6 12
6 7 6
6 5 3
5 7 15
7 5 6
```

Примерен резултатен файл:

```
5 38
4
7
5
6
8
```

Задача 22. Tour – Обиколка на забележителности – 20т.

Станчо често развежда туристи из забележителностите на Софийски университет. Тъй като Софийският университет е уникален, а и туристите, които искат да го разглеждат, обикновено страдат от психични заболявания, то може да приемем, че всичко принадлежащо на Софийският Университет, е забележителност. Разходката представлява тръгване от някоя произволна забележителност (където се пристига с автобус), обиколка из забележителностите (преминава се само през някои забележителности в някакъв ред) и връщане на началното място (при автобуса). За да не бъдат отегчени туристите трябва да не се минава през едно и също място повече от веднъж и между две съседни забележителности да не се преминава повече от веднъж (например един път на отиване и един път на връщане). Тъй като Станчо цял ден не е пил минерална вода, а вече е започнала сбирката на неговата катедра, на която има крайно количество минерална вода, той иска да приключи максимално бързо с обиколката за да остане минерална вода и за него. Помогнете на Станчо да намери най-бързия маршрут за обиколка.

Входни данни:

Софийският университет е представен чрез неориентиран претеглен граф, като забележителностите са означени с върхове, а възможностите за преминаване между две съседни забележителности са означени с ребра. Входните данни са дадени в текстов файл с име `tour.inp`. На първия ред стоят две цели числа N и M, разделени с интервали, задаващи съответно броя на забележителности и броя възможности за преминаване между двойки забележителности в университета ($1 \leq N \leq 800$). Следват N реда, описващи времето, което е необходимо за разглеждане на всяка от забележителностите – цяло неотрицателно число брой в минути. Следват M реда описващи възможностите за придвижване от една забележителност към друга (ребрата в графа). Една такава възможност се описва от 3 числа, разделени с интервал – начална забележителност, крайна забележителност и време необходимо за преминаване (цяло неотрицателно число, указващо времето в минути). Възможно е преминаване и в двете посоки, като времето за преминаване е еднакво, но не се допуска в един маршрут да се преминава повече от веднъж по едно ребра на графа. Забележителностите са номерирани с числата от 1 до N. Възможно е да има няколко възможности за преминаване между някои двойки забележителности. В такъв случай е възможно маршрутът да се състои дори само от 2 забележителности, но това се допуска само ако на отиване и на връщане се минава по различни ребра на графа.

Резултат:

Резултатът трябва да се запише в текстов файл с име `tour.out`. На първия ред трябва да стоят две числа C и L. C указва броят на забележителностите участващи в оптималния маршрут, а L необходимото време. На всеки от останалите C реда трябва да има по едно число указващо номер на забележителност от маршрута. Тъй като се тръгва и се стига до една и съща забележителност то тя се извежда само веднъж във файла. Общото време за обиколката се получава като се сумират времената за всяка забележителност (началната се брои само веднъж) и пътищата по които е минато. Ако има повече от 1 най-бързи обиколки, трябва да се изведе лексикографски най-малката. Ако няма валидни според условието обиколки трябва да се изведе 0 0.

Примерен входен файл:

```
8 14
8
20
30
8
2
3
5
6
1 2 4
3 2 8
3 1 20
2 3 8
4 5 2
4 6 2
4 8 3
4 7 8
8 7 2
8 6 12
6 7 6
6 5 3
5 7 15
7 5 6
```

Примерен резултатен файл:

```
3 20
4
5
6
```

Задача 23. Fastsort – Бързо сортиране – 25т.

След като катедрата на Станчо започнала да се занимава с алгоритми и след успешното анализиране и дори опит за реализация на алгоритъма за сортиране чрез вмъкване Станчо забелязал че съществуват

някои по-добри алгоритми за сортиране като в някои по-специални случаи имало и още по-добри. Тогава той изчезнал от света за 2 месеца и когато се върнал заявил, че е реализирал ултра мега бърз алгоритъм за сортиране и предизвикал цялата катедра, че и останалите (без тази по алгоритми разбира се) да се опитат да реализират по-бърз. Естествено никой не се осмелил да се пробва и всичко било страхотно докато един ден това не се дочуло от катедрата по алгоритми. За да не бъдат прекалено брутални към старият Станчо те решили да дадат на някой студент да напише по-бързо сортиране от СтанчоСорт. За нещастие обаче като човек с голямо влияние и авторитет тестването ще се извършва единствено върху тестове съставени от Станчо които на всичкото отгоре са тайни. За това се предполага че за много от тестовете са доста специфични – например цялата редица представлява цели числа (въпреки че във форматът са обявени числа с плаваща точка и двойна точност (double типът за популярните програмни езици). Дори се чува че някои от тестовете били само от едноцифрени други пък били почти сортирани или представлявали пермутация на числата от 0 до N-1.... Изобщо тази работа намирисвала. За щастие обаче е известно че Станчо пише единствено на QBasic за DOS и при това кодът му е много неоптимизиран (все пак е работил по него едва 2 месеца). За това програмата която трябва да се напише има малко време и да анализира подадената последователност за да избере най-подходящият алгоритъм за сортиране. Последното условие което Станчо поставил било да не се използват алгоритмите за сортиране от стандартните библиотеки като например STL тъй като те били прекалено бързи и били писани от киборзи дошли от бъдещето за да направят зъл изкуствен интелект който да превземе света. Така че без използване на сортиранията предлагани от стандартните библиотеки.

Входни данни:

Входните данни са дадени в текстов файл с име `fastsort.inp`. За да може да се прецени по-прецизно бързодействието той съдържа серия от последователности. Всяка последователност е зададена на 2 реда. На първият се има едно число N – броят на числата в дадената последователност. На следващият ред има N числа с десетична точка задаващи самата последователност. Файлът завършва с 0 за брой на елементите в последователност. Броят на последователностите не надвишава 50, а за всяка последователност броят елементи за сортиране N не надвишава 10 000 000.

Резултат:

Резултатът трябва да се запише в текстов файл с име `fastsort.out`. Той трябва да съдържа по един ред за всяка последователност – съответната и сортирана последователност. За да няма проблеми с точността на числото се иска числата да се изведат до последната ненулева цифра след десетичната точка (стандартното извеждане на числа с плаваща запетая).

Примерен входен файл:

```
6
1 3 2 4 5 3
9
1.0 2 1 2.00 1 1.00000 2 1 1
6
1.12 1.13 1.14 1.15 1.16 1
7
6 5 4 3 2 1 0
0
```

Примерен резултатен файл:

```
1 2 3 3 4 5
1 1 1 1 1 1 2 2 2
1 1.12 1.13 1.14 1.15 1.16
0 1 2 3 4 5 6
```

Задача 24. Insertsort – Сортиране чрез вмъкване – 25т.

За да бъде модерна катедрата на Станчо трябва често да актуализира направленията в които работи. Въпреки големите успехи в анализа на минералната вода темата вече доста се изчерпала и все по-рядко достигали до откритие. За това те решили да се насочат към друга област – компютърните алгоритми. Тъй като катедрата не е много навътре в алгоритмите те решили да започнат от нещо по-просто. По точно алгоритмите за сортиране. И тъй като и при тях има далеч нетривиални те избрали един

сравнително прост алгоритъм – алгоритъмът за сортиране чрез вмъкване. Не след дълго те забелязали че при проектирането и анализа на компютърни алгоритми често се споменава непозната за тях магическа дума. Това било точно думата “сложност”. Но такива сложно звучащи думи не могат да уплашат старо куче като нашият Станчо. Веднага се разровил из старата библиотека и прегледал голямо количество писания относно въпросният термин. Дори успял да разбере някои от тях. Оказало се, че сложността не е шега работа но все пак намерил връзка между нея и броят размени които извършва вълшебният алгоритъм. За това събрал катедрата и им заявил че трябва да се напише програма която по дадена последователност пресмята този брой. Понеже обаче този брой зависи от точната реализация Станчо уточнил задачата. За всеки елемент в сортираната последователност броят размени е точно броят на по-големите от него числа които стоят преди него. Тогава Станчо се замислил че в някои науки това се наричат инверсии но го заболяла главата и се отказал да си спомня за какво точно става дума. Все пак той бил хитър човек и се сетил че този брой може да е доста голям (дори му се въртяло нещо като квадрат в главата но решил че главата му е кръгла и трудно ще се завърти такова ръбато нещо там). За това от вас се иска да напишете програма която пресмята този брой значително по-бързо.

Входни данни:

Входните данни са дадени в текстов файл с име `insertsort.inp`. На първият ред от него има едно число N – броят на числата в дадената последователност ($1 \leq N \leq 500\ 000$). На следващият ред има точно N числа разделени с интервал – числата от последователността за сортиране.

Резултат:

Резултатът трябва да се запише в текстов файл с име `insertsort.out`. Той трябва да съдържа единствено число – броят размени които би извършило сортирането чрез вмъкване по дефинираният горе начин.

Примерен входен файл:

```
6
1 3 2 4 5 3
```

Примерен резултатен файл:

```
3
```

Задача 25. Party – Парти на катедрата – 20т.

Станчо е много контактна личност и се разбира със всички от неговата катедра. Тъй като е голям почитател на минералната вода, а също и голям аналитик, той забелязал, че на купони на катедрата минерална вода се пие винаги по двойки и при това двама души пият заедно само ако се разбират. За това пресметнал, че ако на купона дойдат група взаимно неразбиращи се хора и той самия, то той ще изпие сам половината количество минерална вода, а неговите колеги ще си разделят по равно останалата половина. Помогнете на Станчо да намери такава група хора.

Входни данни:

Входните данни са дадени в текстов файл с име `party.inp`. На първият ред от него има 3 числа N, M и K ($1 \leq N \leq 500$, $1 \leq K \leq 50$). N е броят на членовете на катедрата (без Станчо), M е броят на двойките които се разбират, а K е броят на хората, които Станчо трябва да покани (без него). На следващите M реда стои по една двойка числа между 1 и N – двама члена на катедрата които се разбират.

Резултат:

Резултатът трябва да се запише в текстов файл с име `party.out`. В случай че има група от K члена на катедрата, за които никой не се разбира с никой друг, трябва да се изведе лексикографски най-малката такава група (спрямо номерата на хората) с по един човек на ред. В противен случай трябва да се изведе само един ред с числото -1.

Примерен входен файл:

```
6 8 4
1 2
1 3
2 3
```

```
5 3
4 3
3 6
2 5
2 6
```

Примерен резултатен файл:

```
1
4
5
6
```

Задача 26. Meeting – Среща между катедрите – 20т.

За обмяна на опит и подобряване на образователния процес често се организират срещи между катедрите. Този път вражеската катедра е поканила катедрата на Станчо на среща с подобна цел. Тъй като Станчо е човек с много големи връзки и опит в организирането на подобни сбирки отново са поверили на него организацията. За нещастие Станчо намерил единствено място в което всички маси са с по две места. Неговата аналитична мисъл обаче веднага проработила и преценил че той може да има изгода от това. Тъй като вражеската катедра кани, то тя ще заплати всичката изпита минерална вода. При това положение ако се изпие много минерална вода то вражеската катедра ще влезе в преразход, и съответно катедрата на Станчо няма да е единствената надхвърлила бюджета си. Станчо също забелязал че ако двама души имат общи интереси то те започват задълбочени разговори, съответно пиенето на минерална вода остава на заден план. Докато ако нямат такива изпиват много повече минерална вода. Помогнете на Станчо да подбере максимален брой двойки на членове (по 1 от 2-те катедри) така че никой да не участва повече от веднъж и всяка двойка да нямат общи интереси.

Входни данни:

Входните данни са дадени в текстов файл с име `meeting.inp`. На първият ред от него стоят две числа N и M ($1 \leq N \leq 800$, $1 \leq M \leq 800$). Те представляват съответно броят на членовете на катедрата на станчо и броят на членовете на вражеската катедра. Следват N реда описващи интересите на членовете на катедрата на Станчо и M реда описващи интересите на членовете на вражеската катедра. Всеки от следващите редове представлява низ от малки латински букви – интересите на съответният член. Т.е. всеки интерес се записва с една уникална буква.

Резултат:

Резултатът трябва да се запише в текстов файл с име `meeting.out`. На първият ред от него трябва да има едно число K – максималният брой двойки които могат да се изберат. На всеки от следващите K реда трябва да има по две числа – първото между 1 и N указващо номерът на член от катедрата на Станчо и второто между 1 и M указващо номерът на член от вражеската катедра. Тези членове трябва да нямат общи интереси. При наличие на повече от едно решение се иска което и да е от тях.

Примерен входен файл:

```
5 4
abz
ab
za
abc
mn
amp
pma
stz
mnz
```

Примерен резултатен файл:

```
2
2 4
5 3
```

Задача 27. Science – Научна дейност – 25т.

След последната среща между катедрите вражеската катедра обвинила Станчо, че неговата катедра няма абсолютно никаква целенасоченост и работи хаотично. Тогава Станчо решил да покаже на другите катедри че това далеч не е така и неговата катедра работи целенасочено в определена област. За съжаление не могъл лесно да определи каква точно е тази област и за това разровил архивите на катедрата и намерил всички научни открития. За всяко научно откритие той определил 3 стойности: D_i, T_i и V_i. D_i показва дискретността на съответното откритие – положително число означава че то е от дискретните науки, а отрицателно, че е от непрекъснатите. T_i показва теоретичността на съответното откритие – положително число означава че е теоретично а отрицателно че е приложно насочено. V_i показва колко е велико съответното откритие. Очевидно (поне за Станчо) за група открития S общата специфичност се определя по формулата:

$$\sqrt{\left(\sum_{i\in S} D_i V_i\right)^2 + \left(\sum_{i\in S} T_i V_i\right)^2}$$

Помогнете на Станчо да открие групата с най-голяма специфичност.

Входни данни:

Входните данни са дадени в текстов файл с име `science.inp`. На първият ред от него стои едно число N указващо броя открития на катедрата ($1 \le N \le 200\,000$). На следващите N реда има по 3 числа - D_i, T_i и V_i за съответното откритие.

Резултат:

Резултатът трябва да се запише в текстов файл с име `science.out`. На първия ред от него трябва да има едно число K – броят на открития в групата с максимална специфика. На всеки от следващите K реда стои по едно число – номерът на съответното откритие. Ако има няколко групи с максимална специфика да се изведе тази от тях, която е най-малка лексикографки. Номерата на откритията, описващи групата, трябва да са подредени по големина.

Примерен входен файл:

```
5
1 1 5
-1 1 1
-3 -3 4
-1 0 2
5 -3 4
```

Примерен резултатен файл:

```
2
1
5
```

Задача 28. Castle – Замък с тунели – 20т.

Станчо много обичал да се разхожда из замъка на Софийски Университет. За съжаление обаче в момента катедрите били във война и се разпределяли територии. След последното псевдо-примирие всяка катедра започнала да контролира превзетите тунели. За щастие залите били обявени за неутрални територии. Всъщност замъкът на Софийски Университет представлява две множества. Множество от зали. И множество от тунели. Всеки тунел свързвал две зали. Тъй като катедрите ограничили движението на много от тунелите може да се счита че всички тунели са еднопосочни. Тунелите които трябвало да останат двупосочни били разделени на две с маркировка по средата и всички ги приемали за 2 различни тунела. След известно време катедрата на Станчо стигнали до извода че всеки ден по техните тунели минавали хиляди хора и те съответно могат да препечелят от това. Въвели такса за преминаване по тунел която се измервала в милилитри минерална вода – основната валутна единица на Софийски Университет. За нещастие и другите катедри се досетили същото и скоро на всички тунели в замъкът били въведени такси за преминаване. Катедрите приложили най-различни подходи за изчисляване на таксите. Голяма пречка се оказал факта че таксите не можели да използват комплексни числа. Още по-голяма че трябвало да бъдат и цели. Някои използвали крайни изчислителни машини и изгубили много точност. Други пък искали да представят таксите в ред на Фурие... Общо взето всички

отдавна били забравили какво е естествено число и поради това много от таксите се оказали отрицателни. Това означавало че ако минеш по такъв тунел охраната ти дава известно количество минерална вода вместо да взема. Тогава Станчо се зачудил – дали не може да намери маршрут достижим от неговият кабинет (който е с номер 1 тъй като Станчо е най-важната личност в университета) който да е цикличен и обикаляйки по него Станчо да печели минерална вода. Тъй като това би бил земен рай за Станчо той не се интересува дали от този маршрут може да се върне в кабинета си – рано или късно ще го пренесат. За това вашата задача е да намерите такъв маршрут. Понеже Станчо лесно се отегчава той би искал маршрутът да е опростен – да не се минава през една и съща зала повече от веднъж за една обиколка на маршрута.

Входни данни:

Входните данни са дадени в текстов файл с име `castle.inp`. На първият ред има две числа N и M ($1 \leq N \leq 500$). N е броят на зали в замъка а M е броят тунели. Следват M реда всеки от които описва един тунел с 3 числа – начална зала, крайна зала и такса за преминаване. Залите са с номера от 1 до N като зала с номер 1 е кабинетът на Станчо.

Резултат:

Резултатът трябва да се запише в текстов файл с име `castle.out`. На първият ред трябва да стои K – броят зали на намерения цикличен маршрут. На всеки от следващите K реда трябва да има по едно число – съответният номер на зала. Маршрутът трябва да има отрицателна сума на таксите, да няма дадена зала повече от веднъж и да е достижим от зала номер 1. При повече от един възможни отговора изведете един кой да е от тях.

Примерен входен файл:

```
6 10
4 3 5
4 5 3
5 3 -2
2 6 4
3 2 -1
3 4 -2
1 2 3
1 3 -1
2 1 5
2 3 -1
```

Примерен резултатен файл:

```
3
3
4
5
```

Задача 29. Snow – Бой със сняг – 25т.

От катедрата на Станчо много обичат да си играят на полянката пред факултета. Когато завалял снегът всички били много щастливи и веднага им хрумнало да направят Станчо на снежен човек. Свикало се събрание на катедрата при което N – 1 гласували за и 1 гласувал против. Станчо бил здраво загазил. Но изведнъж му хрумнала идея. Предложил на всички масов бой с топки и 3-литрова бутилка минерална вода за победителят която лично той щял да осигури (разбира се на Станчо не му се давало такова голямо количество минерална вода за това той предварително бил напълнил няколко бутилки със сняг който в момента се топлели на парното) . Разбира се тогава възникнал проблем – как ще бъдат дефинирани правилата. Набързо се спретнал един квази-полином който да бъде оценяваща функция. Обаче поради частичната дискретна ориентация на някои от катедрата се оказало че някои хора мятат топки много по-точно когато стрелят успоредно на координатните оси или на ъглополовящите им (по 8те посоки). Съвсем случайно също се оказало че полянката пред факултета представлява матрица с N реда и N стълба. Станчо трябвало да разположи членовете на катедрата така че да няма двама които стоят на един ред, стълб или диагонал (защото ще започнат да се целят взаимно за да трупат точки тъй като катедрата на Станчо е много навътре в теорията на игрите). Ако Станчо не се справи със задачата и

не успее да измори всички с жестоката снежна битка ще се премине към точка 2 от дневният ред което не е никак хубаво.

Входни данни:

Входните данни са дадени в текстов файл с име `snow.inp`. На единственият ред от него стои числото N ($1 \leq N \leq 500$).

Резултат:

Резултатът трябва да се запише в текстов файл с име `snow.out`. Тъй като на всеки ред има по един човек изходът съдържа N реда с по едно число – колоната на която се намира човекът от съответният ред. Колоните са номерирани от 1 до N. За дадените входни данни винаги ще има решение.

Примерен входен файл:

```
4
```

Примерен резултатен файл:

```
2
4
1
3
```

Задача 30. AllCycles – Всички цикли в граф – 25т.

Да се проектира и реализира ефективен алгоритъм за намиране на всички цикли в неориентиран граф. Всеки цикъл трябва да се отпечата точно веднъж, като редът на отпечатване няма значение. Алгоритъмът трябва да работи със сложност, зависеща от броя цикли в графа, т.е. при ацикличен граф да работи много бързо, а при пълен граф да работи експоненциално (това е съвсем естествено). Да се оцени сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Графът е описан в текстов файл с име `allcycles.inp`. На първия ред стоят две цели числа N и M, разделени с интервали, задаващи съответно броя върхове и броя ребра в графа. На всеки от следващите M реда има описание на едно ребро от графа – 2 числа, разделени с интервал, задаващи съответно начален връх и краен връх на реброто.

Резултат:

Резултатът трябва да се запише в текстов файл с име `allcycles.out`. На всеки ред от файла трябва да стои по един цикъл. Всеки цикъл се задава с последователност от върховете, които го съставят, разделени с интервал, като първият и последният връх трябва да съвпадат. Редът, в който се отпечатват циклите е без значение.

Примерен входен файл:

```
11 13
1 2
1 4
2 3
2 4
3 4
3 5
5 6
6 7
6 8
7 8
9 10
9 11
10 11
```

Примерен резултатен файл:

```
3 4 1 2 3
3 2 4 3
2 4 1 2
```

```
6 7 8 6
9 10 11 9
```

Задача 31. Cover – Покрития – 25т.

Дадено е число n и квадратна таблица с размери 2n+1 на 2n+1, която се състои от еднакви квадратни клетки. Всяка клетка съдържа стойност 0 или 1. Покриващо множество наричаме такова множество от 2n+1 клетки от таблицата, че в това множество има по точно една клетка от всеки ред и всеки стълб от таблицата.

Върху таблицата е допустима следната операция: Избираме произволно покриващо множество и за всяка клетка от него инвертираме стойностите в таблицата, т.е. всички клетки от това множество, които са били със стойност 0, стават със стойност 1, а всички клетки които са били със стойност 1, стават със стойност 0.

Да се проектира и реализира ефективен алгоритъм, който определя дали след не повече от (2n+1)! на брой прилагания на горната операция може да се получи таблица, съдържаща не повече от 2n на брой клетки със стойност 1. Да се оцени сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Входните данни се задават в текстов файл с име `cover.inp`. Първият ред съдържа едно цяло число n ($1 \leq n \leq 100$). Следващите 2n+1 реда съдържат описание на редовете от таблицата. Всеки ред от таблицата е описан от 2n+1 числа, всяко от които е или 0 или 1, разделени помежду си с един интервал.

Резултат:

Резултатът трябва да се запише в текстов файл с име `cover.out`. Първият ред на този файл трябва да съдържа съответно “YES” или “NO” в зависимост от това дали съществува решение или не. Останалите редове описват решението, ако има такова. Ако съществува решение, то трябва да бъде отпечатано като последователност от K пермутации на числата от 1 до 2n+1, предхождани от числото K само на ред. Всяка пермутация съответства на една операция инвертиране на покриващо множество, като тя задава за всеки от стълбовете от 1 до 2n+1 номерът на реда на клетката от съответния стълб, която трябва да се инвертира. Всяка от пермутациите трябва да се отпечата на отделен ред, а числата, които ги съставят, трябва да са разделени по между си с интервал.

Примерен входен файл:

```
1
1 1 1
1 1 0
1 0 1
```

Примерен резултатен файл:

```
YES
2
3 2 1
2 1 3
```

Задача 32. Chess – Силно опростен шах (мат с цар и топ) – 20т.

Да се проектира и реализира алгоритъм, който играе опростен шах и винаги побеждава след минимален брой ходове. При опростен шах се играе на стандартна шахматна дъска и правилата на играта са същите като при шаха, но на дъската има ограничен брой фигури. В нашия случай има един бял цар, един бял топ и един черен цар. Началната позиция е валидна съгласно правилата на играта шах. Компютърът играе с белите фигури и започва първи. Целта е компютърът да матира потребителския цар за минимален брой ходове.

Входни данни:

Входните данни се задават в текстов файл с име `chess.inp`, който съдържа три реда. На първия ред е записана позицията на белия цар, на втория ред е записана позицията на белия топ, а на третия ред е записана позицията на черния цар. Всяка позиция се записва по стандартното за шаха означение – чрез главна латинска буква (от A до H), последвана от цифра (от 1 до 8) без разделител между тях.

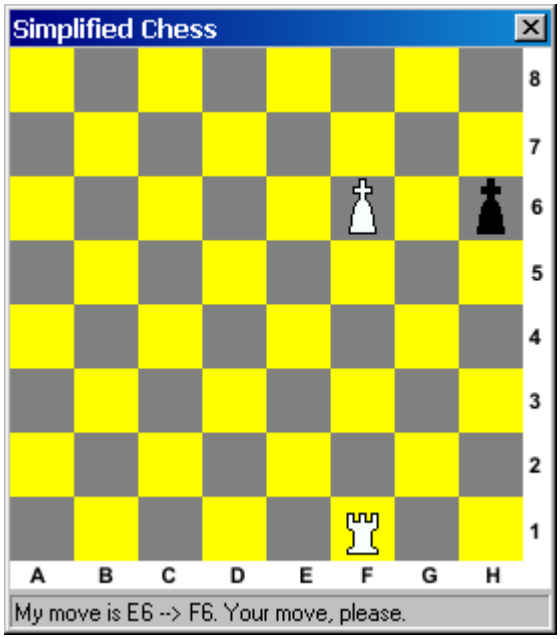
Резултат:

Резултатът е интерактивна игра, при която компютърът започва пръв и след всеки свой ход дава възможност на потребителя да играе своя ход, а след това играе оптимално. В крайна сметка компютърът трябва да матира потребителя за минимален брой ходове. Необходимо е играта да се визуализира по подходящ начин.

Примерен входен файл:

```
E6
F1
H6
```

Примерен резултат:



Задача 33. MoreChess – Леко опростен шах (цар и топ срещу цар и кон) – 25т.

Да се проектира и реализира алгоритъм, който играе опростен шах и винаги побеждава след минимален брой ходове. При опростен шах се играе на стандартна шахматна дъска и правилата на играта са същите като при шаха, но на дъската има ограничен брой фигури. В нашия случай има един бял цар, един бял топ, един черен цар и един черен кон. Началната позиция е валидна съгласно правилата на играта шах. Компютърът играе с белите фигури и започва първи. Целта е компютърът да играе оптимално и съответно да матира противника е възможно или да постигне поне реми ако победа не е възможна.

Входни данни:

Входните данни се задават в текстов файл с име `morechess.inp`, който съдържа четири реда. На първия ред е записана позицията на белия цар, на втория ред е записана позицията на белия топ, на третия ред е записана позицията на черния цар, а на четвъртия ред е записана позицията на черния кон. Всяка позиция се записва по стандартното за шаха означение – чрез главна латинска буква (от A до H), последвана от цифра (от 1 до 8) без разделител между тях.

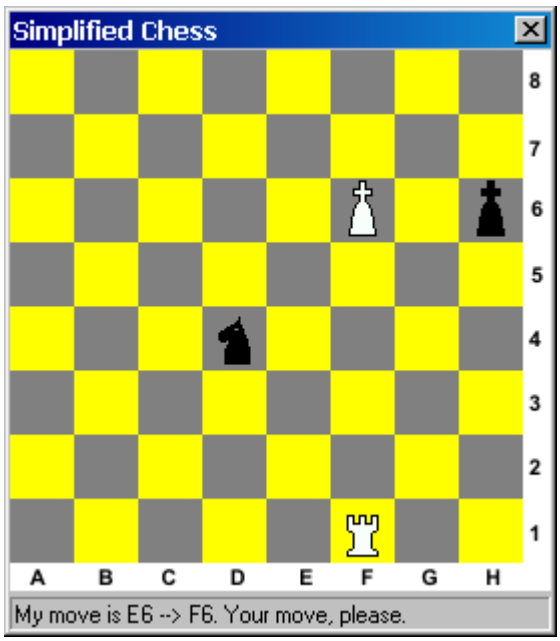
Резултат:

Резултатът е интерактивна игра, при която компютърът започва пръв и след всеки свой ход дава възможност на потребителя да играе своя ход, а след това играе оптимално. В крайна сметка компютърът трябва да победи, ако е възможно или да постигне поне реми, ако победа не е възможна при оптимална игра на черните.

Примерен входен файл:

```
E6
F1
H6
D4
```

Примерен резултат:



Задача 34. Min4 – Най-близки 4 върха в граф – 20т.

Картата на едно населено място е моделирана чрез свързан неориентиран граф с тегла по ребрата и по върховете. Път между два върха в графа наричаме последователност от върхове, започваща от началния връх и завършваща в крайния връх. Дължина на път наричаме сумата от тежестите на всички ребра, участващи в него, към която са добавени тежестите на всички междинни върхове, през които се преминава (началния и крайния не се броят). Разстояние между два върха наричаме дължината на най-късия път между тях. Да се проектира и реализира ефективен алгоритъм, който намира 4-те най-близки един до друг върха на графа, т.е. тези 4 върха, за които сумата от 6-те разстояния по между им е минимална. Да се определи сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Входните данни се задават в текстов файл с име `min4.inp`. На първия ред на входния файл стоят две цели положителни числа N и M, разделени с интервал, които задават съответно броя върхове и броя ребра в графа ($1 \le N \le 500$, $1 \le M \le N*(N-1)/2$). На следващия ред стоят N цели положителни числа, задаващи съответно теглата по върховете. Следващите M реда описват ребрата на графа. Всеки от тези редове съдържа по три числа, разделени с интервали, които задават съответно начален връх за реброто, краен връх за реброто и тегло (дължина). Всички тегла по върховете и по ребрата са цели положителни числа ненадвишаващи 30000.

Резултат:

Резултатът трябва да се запише в текстов файл с име `min4.out`. На първия ред на този файл трябва да стоят 4 числа, разделени с интервал – 4-те най-близки върха в графа. Ако има няколко решения, да се отпечата само едно от тях.

Примерен входен файл:

```
12 16
7 9 5 1 1 2 4 2 5 3 8 8
1 2 17
1 4 4
1 9 3
1 12 9
2 6 20
2 4 1
3 6 4
3 7 9
4 6 1
4 7 11
```

```
4 8 3
5 9 8
7 8 5
8 12 2
9 11 6
10 12 5
```

Примерен резултатен файл:

```
18
2 4 6 8
```

Задача 35. Sequence – Редица със знаци – 20т.

Дадени са две цели положителни числа N и C. Разглеждаме равенството:

$$C = 1 \Delta 2 \Delta \ldots \Delta N$$

Да се проектира и реализира ефективен алгоритъм, който проверява дали даденото равенство може да бъде удовлетворено, ако всеки от знаците Δ бъде заменен или с "+", или с "-", или с "*". Пресмятането на получения числов израз се извършва отляво надясно, като операциите са с еднакъв приоритет.

Входни данни:

Входните данни се задават в текстов файл с име `sequence.inp`. На първия ред на входния файл стои едно число K – брой равенства, които програмата трябва да изследва ($1 \leq K \leq 100$). На всеки от следващите K реда има по две цели числа N и C, разделени с интервал ($1 \leq N \leq 50$, $1 \leq C \leq 30\,000$).

Резултат:

Резултатът трябва да се запише в текстов файл с име `sequence.out`. Файлът трябва да се състои от точно K реда. На всеки от тях трябва да има информация за поредното равенство, описано във входния файл. Ако има решение, то трябва да бъде описано във вида:

$$C = 1 \Delta 2 \Delta \ldots \Delta N$$

където знаците Δ за заменени със съответните аритметични операции, водещи до удовлетворяване на равенството. Ако няма решение на съответния ред трябва да стои фразата “No solution”.

Примерен входен файл:

```
3
19 5
177 4
912 11
```

Примерен резултатен файл:

```
19=1+2+3*4-5
No solution
912=1*2*3*4*5-6-7-8*9+10+11
```

Задача 36. Maxsum – Максимална сума – 20т.

Даден е двумерен масив NxN от цели числа. Да се напише програма която намира подправоъгълника на двумерния масив с най-голяма сума на елементите. Подправоъгълник наричаме всеки подмасив с размери 1x1 или по-голям, намиращ се в дадения масив.

Входни данни:

Входните данни са дадени в текстов файл с име `maxsum.inp`. Първия ред от него съдържа числото N (N <= 800) размерността на масива. Следват N^2 естествени числа в интервала [-127, 127] разделени с бели полета (white-space) Тези N^2 числа представляват масива (всички числа от първия ред от ляво на дясно, всички числа от втория ред от ляво на дясно, ...)

Резултат:

Резултатът трябва да се запише в текстов файл с име `maxsum.out`. Отпечатайте в него сумата на максималния подправоъгълник.

Примерен вход:

```
2
-3 1
1 4
```

Примерен резултат:

```
5
```

Трудни задачи

Задача 41. Rooks – Задачата за топовете – 30т.

При сбирките си катедрата на Станчо обича да се занимава с различни проблеми. Известен за тях проблем бил задачата за разполагане на царици които не се бият взаимно върху квадратна дъска. За съжаление след поглъщане на голямо количество минерална вода тази задача вече не била интересна за тях и решили да измислят и решат по-интересна задача – задачата за топовете. Тя била да се разположат N топа по квадратна таблица така че никои два да не се бият (един топ бие по неговият ред и стълб). За съжаление само след час Станчо открил че задачата е тривиална тъй като топовете могат да се разположат по главния диагонал. За това решил да усложни задачата. На всяка позиция сложил чаша минерална вода или краставичка. Във всяка чаша има цяло положително число милилитри вода. Така той променил задачата на: "Да се разположат N топа на мястото на N чаши така че никои два да не се бият и изпитото количество минерална вода да е максимално.". По този начин събрал вниманието на цялата катедра върху задачата. Както личи от условието върху краставичка не може да се слага топ, но това не бил проблем за катедрата тъй като те имали цял чувал с краставици.

Входни данни:

Входните данни са дадени в текстов файл с име `rooks.inp`. На първият ред от него стои едно число N указващо размера на таблицата ($1 \leq N \leq 500$). На следващите N реда има по N числа описващи всяка позиция с по едно цяло неотрицателно число. Ако числото е 0 то на съответното квадратче има краставичка. В противен случай на него има чаша със съответното количество минерална вода.

Резултат:

Резултатът трябва да се запише в текстов файл с име `rooks.out`. Ако задачата няма решение съдържа единствен ред с числото -1. В противен случай N реда с по две числа – номерът на ред и стълб където е поставен топ. Позициите трябва да се изведат сортирани по ред. Ако има няколко решения да се изведе кое да е от тях.

Примерен входен файл:

```
5
0 5 1 0 0
5 0 0 2 0
1 1 0 0 2
1 1 3 3 2
0 0 1 2 2
```

Примерен резултатен файл:

```
1 2
2 1
3 5
4 3
5 4
```

Задача 42. Quasirooks – Задачата за квази-топовете – 30т.

След невероятния успех на задачата за топовете (виж задачата за топовете) Станчо решил че има още какво да се постигне около тази задача и започнал да мисли за сходни проблеми. Точно в този момент на главата му паднала краставица и той изпуснал топът които държал. Топът се счупил на две и застанал на 2 различни позиции по дъската. Станчо се ядосал и натрошил всички топове с които катедрата разполагала. Те обаче били направени от диаманти и платина и стрували доста пари. След

като се осъзнал разбрал че бързо трябва да измисли нещо за да замаже положението. Така Станчо измислил задачата за квази-топовете.
Имаме квадратна таблица N на N. Върху някои от позициите има краставички (минералната вода била отдавна изпита). Един квази-топ представлява фигура от 2 части – глава и тяло. Главата на квазитопа може да се разположи на ред R колона C така че R е различно от C и тогава тялото на квази-топа трябва да бъде на ред C и колона R. И двете позиции трябва да бъдат свободни (да няма краставичка на тях). Понеже според Станчо топа е доста силна фигура то квази-топа е още по-силна и бие 2та реда и 2те колони които заема. Така задачата е да се разположат максимален брой квазитопове така че никои два от тях да не се бият.

Входни данни:

Входните данни са дадени в текстов файл с име `quasirooks.inp`. На първият ред от него стои едно число N указващо размера на таблицата ($1 \leq N \leq 500$). На следващите N реда има по N числа описващи всяка позиция с по едно цяло неотрицателно число. Ако числото е 0 то на съответното квадратче е свободно. В противен случай на него има краставичка.

Резултат:

Резултатът трябва да се запише в текстов файл с име `quasirooks.out`. Ако задачата няма решение съдържа единствен ред с числото -1. В противен случай на първият ред стои числото K – броят на квази-топовете поставени на дъската. Следват K реда с по две числа – номерът на ред и стълб където е поставена главата на квази-топа. Позициите трябва да се изведат сортирани по ред. Ако има няколко решения да се изведе кое да е от тях.

Примерен входен файл:

```
6
0 0 1 1 0 0
0 1 1 1 1 1
1 1 0 1 1 0
0 1 1 0 1 0
1 0 0 1 1 0
1 1 0 0 0 0
```

Примерен резултатен файл:

```
2
1 2
5 6
```

Задача 43. GraphCount – Броене на планарни графи – 30т.

Както вече споменахме Станчо много обичал комбинаториката. Един ден той имал странен сън. 3 кръвожадни крави го гонели защото бил оставил полянката пред факултета без трева.... Почти се бил измъкнал, когато го пресрещнал Големият Лош Граф. Графът го сграбчил с големите си върхове и го оплел с дългите си ребра. Казал му, че ще го пусне и ще го спаси от кравите, само ако напише програма която да брои уникалните (неизоморфни един на друг) планарни графи с даден брой върхове и ребра. Пуснал го да се събуди, но му казал, че иска програмата до крайния срок за предаване на проекти по ПрАнКА-1. За това Станчо поискал помощ от нас.

Входни данни:

Входните данни са зададени в текстов файл с име `graphcount.inp`. На единствения ред от него стоят 2 числа N и M указващи съответно броя върхове и броя ребра в графите ($1 \leq N \leq 10$, $1 \leq M \leq 50$).

Резултат:

Резултатът трябва да се запише в текстов файл с име `graphcount.out`. На единствения му ред трябва да се съдържа едно число – броят на планарните графи с даденият брой върхове и ребра (като класовете изоморфни графи се броят само по веднъж). Трябва да се броят всички неориентирани планарни графи (без мултиграфите). Не се изисква графите които се броят да са свързани.

Примерен входен файл:

```
4 5
```

Примерен резултатен файл:

```
1
```

Задача 44. Reverse – Играта “Reverse” – 30т.

Станчо един ден решил да се изфука в катедрата с успехите си по играта Reverse или както я наричал той “Реверси”. Не пропуснал да спомене и че е трикратен световен шампион, двукратен галактически шампион и дори има най-висок рейтинг в Yahoo Games. Тогава цялата катедра много се ентусиазирала защото никой не подозирал, че Станчо може да е добър в каквото и да е. Авторитетът на Станчо се вдигнал до небето и всички искали да се пробват да играят с най-добрия. Това, разбира се, било супер, като изключим малка подробност която Станчо забравил. Всичко това било станало в предният му живот. За съжаление междуживотната памет на Станчо била много слаба и той бил изгубил всичките си умения да играе най-великата за него игра на всички времена. За това той иска от вас да му помогнете да замаже положението, като напишете програма, която да му помага с играта.

Понеже Станчо вече бил забравил и правилата на играта, той се разровил из Интернет и намерил 2 сайта по темата: http://takegame.com/online/reversi/ и http://www.gamesite2000.com/reversirules.htm.

Входни данни:

Входните данни са дадени в текстов файл с име `reverse.inp`. На първият ред от него стои едно число, указващо с кои пулове играе Станчо, т.е. кои пулове са на ход. 1 означава че белите са на ход, а 2 означава, че черните са на ход. Следват 8 реда с 8 числа, разделени с интервал. Всяко от тях има стойност 0, 1 или 2 и съответства на една позиция от дъската, като 0 означава, че позицията е свободна, 1 означава бял пул, а 2 означава черен пул.

Резултат:

Резултатът трябва да се запише в текстов файл с име `reverse.out`. Неговият формат е абсолютно същия като входния и описва позицията след хода на Станчо. Ходът, който е играл Станчо трябва да е валиден. Програмата ви ще бъде извиквана последователно за да изиграе цяла партия. За това и ограничението за време ще бъде за текущият ход. Ще бъдат организирани и турнири между отделните програми.

Примерен входен файл:

```
1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 2 0 0 0
0 0 0 2 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Примерен резултатен файл:

```
2
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0
0 0 0 2 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Задача 45. Mincut – Минимален разрез на поток – 30т.

Стандартно алгоритмите за намиране на максимален поток между два върха в граф могат да намерят стойността на максималния поток, както и големината на потока преминаващ през всяко ребро от графа. Според една добре известна теорема максималният поток е равен на минималният разрез (съвкупността от ребра, които формират най-тясното място, през което протича потока). Да се проектира и реализира

ефективен алгоритъм за намиране на минималния разрез в неориентиран граф. Да се оцени сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Графът е описан в текстов файл с име `mincut.inp`. На първия ред стоят две цели числа N и M, разделени с интервали, задаващи съответно броя върхове и броя ребра в графа ($1 \le N \le 500$). На всеки от следващите M реда има описание на едно ребро от графа. Едно ребро се описва от 3 числа, разделени с интервал – начален връх, краен връх и капацитет на реброто. На последния ред във входния файл стоят две числа S и T, разделени с интервал, които задават номерата на източника и консуматора, между които се търси максимален поток и минимален разрез.

Резултат:

Резултатът трябва да се запише в текстов файл с име `mincut.out`. На първия ред трябва да стоят две числа F и C, разделени с интервал, съответстващи на големината на максималния поток и броя ребра в минималния разрез. На всеки от останалите C реда трябва да има по две числа, разделени с интервал, задаващи по едно от ребрата на намерения минимален разрез. Ако има няколко минимални разреза, трябва да се отпечата само един от тях.

Примерен входен файл:

```
10 14
1 2 2
1 3 3
1 4 5
2 5 3
3 5 4
3 8 2
4 8 1
5 6 1
5 7 1
6 10 3
7 10 4
8 7 3
8 9 2
9 10 1
1 10
```

Примерен резултатен файл:

```
5 4
3 8
4 8
5 6
5 7
```

Задача 46. Planar – Визуализация на планарен граф в равнината – 35т.

Да се проектира и реализира алгоритъм, който проверява за даден неориентиран граф дали е планарен и ако е планарен, го визуализира графично. При визуализацията е необходимо всеки връх да се изобразява като окръжност, в която е записан номерът му, а всяко ребро като права линия свързваща два върха, като не се допуска никои две ребра да се пресичат. Да се реализира възможност за смяна на мащаба (приближаване и отдалечаване) и скролиране на видимата част от полученото графично изображение. Да се оцени сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Графът е описан в текстов файл с име `planar.inp`. На първия ред стоят две цели числа N и M, разделени с интервали, задаващи съответно броя върхове и броя ребра в графа ($1 \le N \le 500$). На всеки от следващите M реда има описание на едно ребро от графа – 2 числа, разделени с интервал, задаващи съответно начален връх и краен връх на реброто.

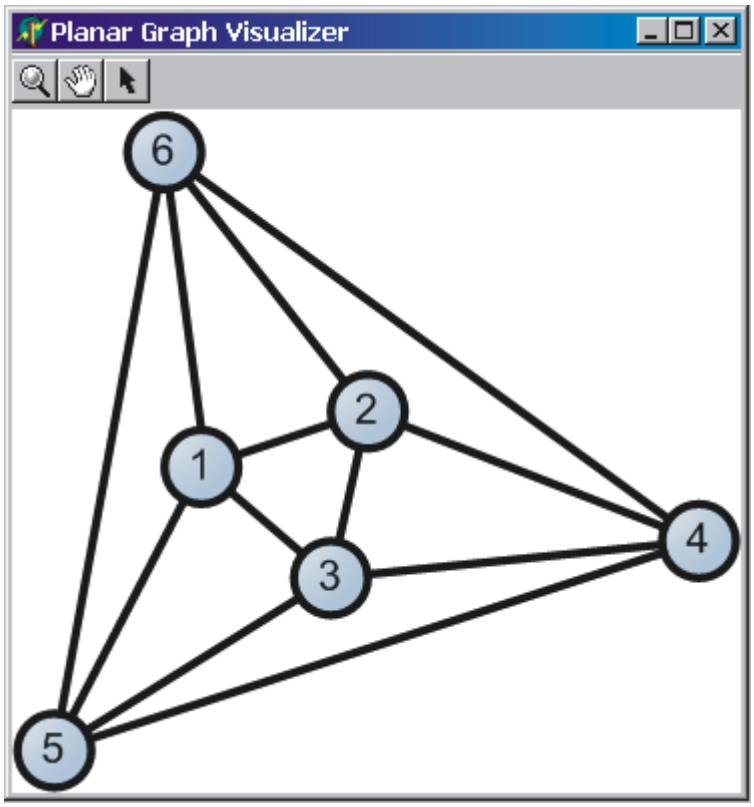
Резултат:

Резултатът е приложение с графичен потребителски интерфейс, което визуализира зададения граф или съобщава, че той не е планарен и дава възможност за мащабиране и скролиране, както е описано в условието.

Примерен входен файл:

```
6 12
1 2
1 3
1 5
1 6
2 3
2 4
2 6
3 4
3 5
4 5
4 6
5 6
```

Примерен резултат:



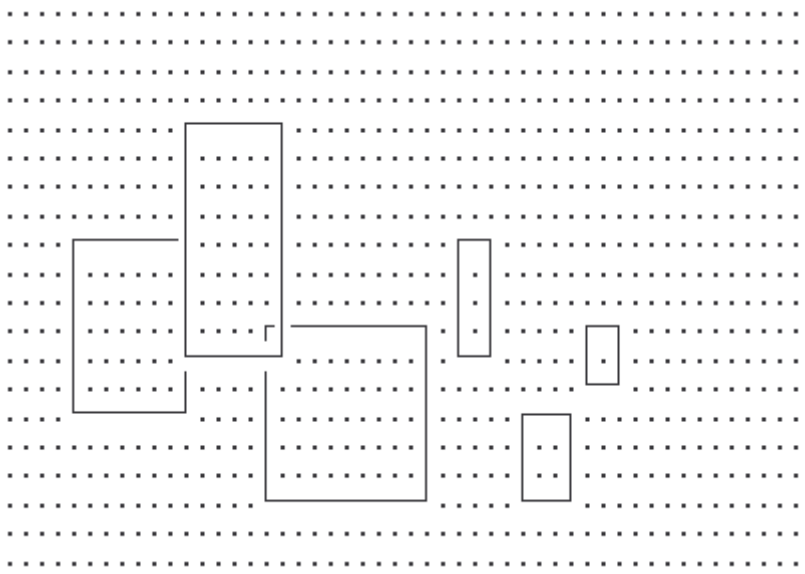
Задача 47. Frames – Рамки – 30т.

Едно време, преди много години, когато още нямаше Unicode, когато още нямаше графичен режим и всичкият софтуер работеше в текстов режим, беше разпространена т. нар. псевдографика. Тя даваше възможност чрез комбинация от текстови символи да се рисуват някои графични изображения. Така например някои програми, които предоставяха на потребителите прозоречен потребителски интерфейс, използваха някои специални символи за изчертаване на рамките на прозорците в текстов режим. Това бяха символите:

Символ	ASCII код	Описание
┌	218	Горен ляв ъгъл на прозорец
┐	191	Горен десен ъгъл на прозорец
└	192	Долен ляв ъгъл на прозорец
┘	217	Долен десен ъгъл на прозорец

│	179	Лява или дясна греда на прозорец
─	196	Горна или долна греда на прозорец

Да си представим, че имаме някаква стара програмна система, която чертае правоъгълни рамки на прозорци в текстов режим посредством символите посочени в таблицата по-горе. Екранът, който е с размери W на H, първоначално е запълнен целия със символа "." (точката е с ASCII код 46). Програмната система е изчертала някакъв неизвестен за нас брой рамки с неизвестни размери на неизвестни позиции от екрана. Някои рамки частично или напълно са закрили някои други, които са били изчертани преди тях. В резултат целият екран се е запълнил със символите "." и символите, използвани за чертане на рамки от таблицата по-горе. Например след изчертаването на няколко рамки върху екран с размери 50 на 20 може да се получи следното изображение:



Да се проектира и реализира ефективен алгоритъм, който по зададено текстово изображение да намира най-късата последователност от команди за изчертаване на рамки, чрез която от празен екран може да се получи зададеното изображение. Командите се описват в XML стил и форматът им е следния:

```
<frame top="top_value" left="left_value" bottom="bottom_value" right="right_value" />
```

където top_value, left_value, bottom_value и right_value задават местоположението и размерите на съответната рамка, която трябва да се изчертае. Валидни са само команди, за които $1 \le$ top_value $<$ bottom_value $\le H$ и $1 \le$ left_value $<$ right_value $\le W$. В използваната координатна система се приема, че най-горният ред от екрана има номер 1, а последният ред има номер H и че най-левият стълб от екрана има номер 1, а най-десният има номер W. Да се оцени сложността на алгоритъма в средния и в най-лошия случай.

Входни данни:

Входните данни се задават в текстов файл с име `frames.inp`. Първият ред съдържа две цели числа W и H ($1 \le W, H \le 1000$). Следващите H реда от файла съдържат описание на редовете от екрана. Всеки ред от екрана е описан от точно W символа, всеки от които е един от допустимите за екрана символи ("." или символите използвани за чертане на рамки, описани в таблицата по-горе).

Резултат:

Резултатът трябва да се запише в текстов файл с име `frames.out` и трябва да представлява XML документ със следната структура:

```
<?xml version="1.0"?>
<commands>
     (команди)
</commands>
```

където `(команди)` е последователност от команди за изчертаване на рамка, описани по една на ред в XML вид съгласно начина, указан по-горе. Ако зададеното текстово изображение не може по никакъв

начин да се начертае чрез използване на валидни команди за изчертаване на рамки, секцията `(команди)` трябва да се състои от един единствен XML таг:

```
    <error message="Invalid text image!" />
```

Примерен входен файл:

```
9 4
.┌──────┐.
.│....┌┐│
.└────┘└┘
...
```

Примерен резултатен файл:

```
<?xml version="1.0"?>
<commands>
    <frame top="1" left="2" bottom="3" right="8" />
    <frame top="2" left="6" bottom="4" right="8" />
    <frame top="3" left="4" bottom="4" right="6" />
</commands>
```

Задача 48. CutGlass – Стъклорезница – 35т.

Една стъклорезница работи с големи листи стъкло с правоъгълни размери M на N. В един работен ден стъклорезницата изпълнява K поръчки за отрязване на стъкла с правоъгълни размери, по-малки от тези на листите. Да се разработи помощна програма, която да оптимизира работата на стъклорезницата по такъв начин, че всички поръчки да бъдат изпълнени с използване на минимален брой листи. Не е необходимо всеки лист да се използва напълно. Възможно е части от използваните листи да бъдат изхвърлени като отпадъчни изрезки. Приемаме, че дебелината на всички разрези е 0, т.е. че при срязване на едно стъкло на няколко парчета сумата от лицата на получените парчета е точно равна на лицето на стъклото, от което са получени.

Входни данни:

Входните данни се задават в текстов файл с име `cutglass.inp`. На първия ред на входния файл стоят целите числа M и N, разделени с интервал, които задават размера на листите, които стъклорезницата използва като материал за изработка на поръчаните стъкла ($1000 \leq M, N \leq 5000$). На следващия ред стои числото K, което задава броя стъкла, които са поръчани за изрязване ($1 \leq K \leq 200$). На следващите K реда стоят по две цели положителни числа – размерите на всяко от поръчаните стъкла за изрязване.

Резултат:

Резултатът трябва да се запише в текстов файл с име `cutglass.out`. На първия ред на този файл трябва да стои едно число – броя на използваните листи. На всеки следващ ред трябва да стоят по 5 цели положителни числа L, X, Y, W, H, разделени с интервал, които означават, че от лист с номер L трябва да се отреже парче с дължина W и ширина H, като горният му ляв ъгъл се започва от координати (X, Y). Координатната система на всеки лист започва от горния му ляв ъгъл, където е позиция (0, 0) и завършва в долния му десен ъгъл, където е позиция (M, N). Намереното оптимално разрязване трябва задължително да се визуализира по някакъв начин, така че лесно да се вижда кои парчета стъкло от кои листи се изрязват и кои части на листите остават като изрезки.

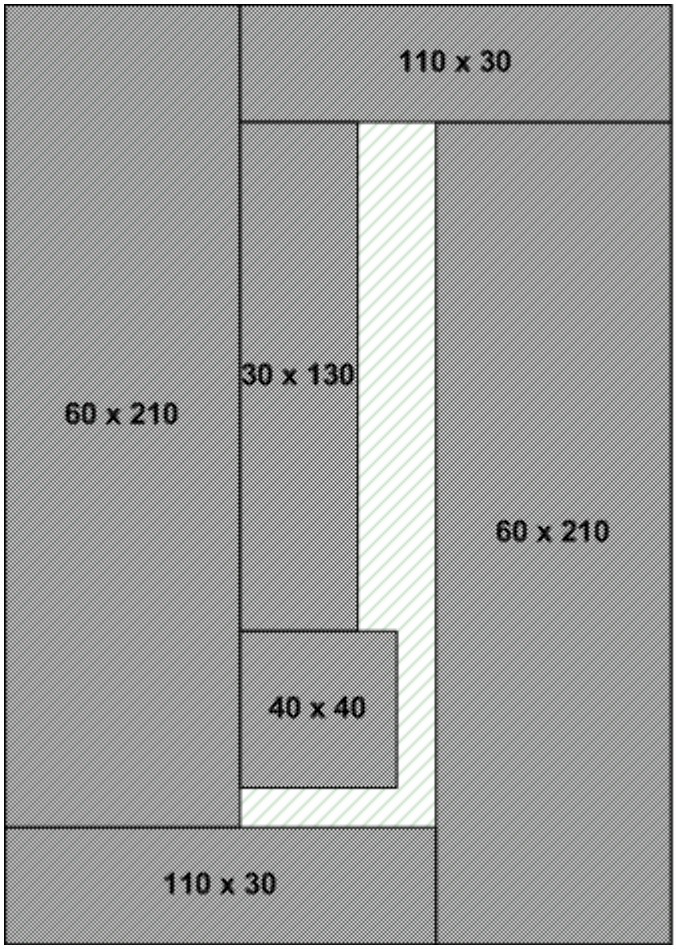
Примерен входен файл:

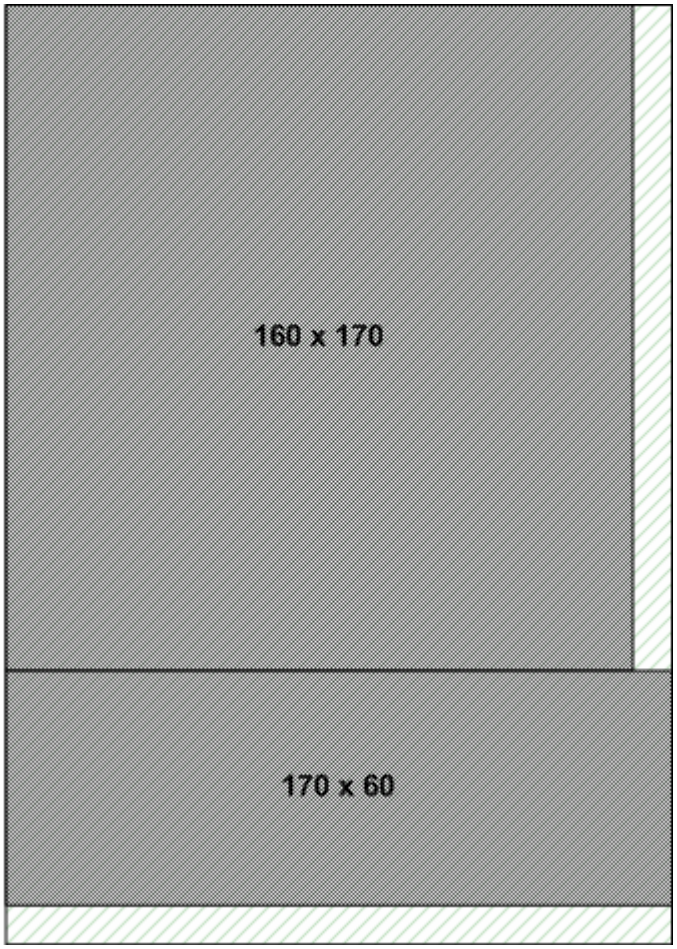
```
170 240
8
170 160
210 60
60 170
40 40
130 30
210 60
110 30
30 110
```

Примерен резултатен файл:

```
2
1 0 0 60 210
1 60 0 170 30
1 60 30 90 160
1 110 30 170 240
1 60 160 100 200
1 0 210 110 240
2 0 0 160 170
2 0 160 170 230
```

Примерна визуализация на резултата:





Автори на задачите:

Светлин Наков
Милослав Средков
Красимир Добрев