ЗИМНИ МАТЕМАТИЧЕСКИ ПРАЗНИЦИ – РУСЕ' 2006

Задача А1. ПРАЗНИК

Разглеждаме числовата права като път. В някои от целочислените точки по пътя може да има къщи (в една точка може да има много къщи), във всяка от които могат да живеят много хора. Така се е образувал град. Хората могат да се придвижват по този път, но трябва да плащат за това. Цената за да пропътува един човек единица разстояние по пътя е 1 лев. Тъй като това е доста скъпо, хората не пътуват често и надалеч. Градската управа се нуждае непрекъснато от пари, но за да не изглежда прекалено нагло, решила да ги събира като пътни такси. Затова периодически организира градски празници, в които участието е задължително. Всеки празник се прави в определена точка с целочислени координати. Жителите на града сами трябва да си осигурят превоза. Градът, за който става дума, е един динамичен град. В него често се разрушават къщи и хората напускат града или пък нови хора се заселват в новопостроени къщи. Затова преди всеки празник управата внимателно трябва да избере мястото, за да може да си осигури предварително намислената сума. Напишете програма **FEST**, която да получава данни за напусналите и заселилите се граждани и, в зависимост от текущото състояние, да може да отговаря на въпроса: "Къде да се организира празника така, че парите които ще се съберат да са максимално близко до дадено число S?". Ще считаме, че в началото градът е празен.

Вход: Програмата трябва да чете данни от стандартния вход. На Всеки ред ще има или данни за заселване/изселване, или въпрос за търсеното място. Ако данните са за заселване или изселване, тогава редът има вида **1 A B**, където **A** е координатата на къщата, откъдето се изселват –**B** човека (ако **B** < 0) или се заселват **B** човека (ако **B** > 0). Ако редът съдържа въпрос, тогава той има вида **2 S**, където **S** е сумата, която управата иска да събере. Първото число във входа винаги ще е 1. Максималният брой редове на входа е 200 000, от които не повече от 20 000 са въпроси. Координатите **A** ще са в интервала [0,10 000 000], числата **B** – в интервала [-1000,1000], а сумата **S** – в интервала [0,10^15].

Изход: Програмата трябва да изведе резултата на стандартния изход, като за всеки въпрос изведе по един ред. Отговорът се състои от две числа **E P**, където **E** (няма да надвишава по абсолютна стойност 10^8) е координатата на мястото на празника, а **P** е абсолютната стойност на разликата между исканите исканата сума и тази, която ще постъпи в касата от таксите. Ако има повече от един възможен отговор Вашата програма трябва да изведе кой да е от тях.

ПРИМЕР:

Вход:	**Изход:**
1 3 7 1 11 5 2 123 2 36	-4 1 3 4

Решение:

Някои константи които ползвам в решението:

manx = 2^27 – максималната абсолютна стойност на координатата която може да бъде отговор на някой въпрос (това не означава, че непременно има такъв тест)

maxx = 2^24 – максималните координати на които може да има къщи (в тестовете това е по-малко)

mx = 2^17 – броя на листата в дървото ми (трябва да е по голям от максималния брой къщи)

За всеки въпрос ние трябва да намерим оптимална точка от числовата права. Тъй като на всяка точка можем да съпоставим цена (сбора от броя на хората във всяка къща умножен по

разстоянието до дадената точка за всички къщи) можем да отговорим на всеки въпрос просто като обходим цялата права и намерим коя точка е оптималната. Но ако разгледаме тази функция то ще видим, че тя е нещо подобно на парабола – т.е. в двата си края функцията приема големи стойности, които намаляват към средата на графиката. Нека първо си отговорим на въпроса "Къде е тази среда на графиката". Нека сега се намираме в произволна точка **X** от числовата права и нека стойността на функцията в тази точка е **C**. Да се опитаме да намерим каква ще е стойността на функцията в точка **X+1**. Всеки човек, който живее в къща с координати **<=X** ще трябва да заплати **1** лев повече, но от друга страна всеки, който живее в къща с координати **>=X+1** ще трябва да заплати **1** лев по-малко. Следователно стойността на функцията в точка **X+1** е равна на **C+(броя на хората живеещи преди или на X) – (броя на хората живеещи след X)**. И това твърдение е вярно за всяка точка. Нека с **Dx** бележим изменението на стойността на функцията между точки **X** и **X+1**. Лесно се вижда, че **Dx > Dy** за **x>y**. Ако за момент си помислим за функцията **Dx** като за производна то лесно че намерим къде е средата (минимума) на функцията. Там където **Dx** е равно на **0**. За съжаление може да няма такава точка (тъй като работим с целочислени стойности), или пък може да има много такива точки. Затова ето къде се намира средата на функцията. Нека къщите са сортирани. Ако съществува къща **K**, такава че **(броя на хората живеещи в къщи 1..K) = (броя на хората живеещи в къщи K+1..N)**, то тогава функцията е минимална в целия затворен интервал **[K,K+1]**, като това са координатите на къщите. Ако пък няма такава къща то тогава съществува къща **K** такава, че **(броя на хората живеещи в къщи 1..K) > (броя на хората живеещи в къщи K+1..N)** и такава къща, че **K** е минимално. Тогава минимума на функцията е в тази точка (координатата на къщата). Нека отсега нататък средата на функцията наричаме **MIDDLE** като това е или произволна точка от този интервал или къща лежаща в този интервал. Трябва да отбележа, че функцията е линейна между всеки две къщи.

Дотук представих две решения:

1) Тривиалното – за всяка точка от числовата права смятаме стойността и сравняваме с исканата стойност.
2) Смятаме последователно за всички точки като по този начин смятането на стойността на функцията за всяка следваща точка е за константно време.

Сега ще представя третото решение, което използва специална структура данни, която позволява всяко заселване/изселване да се извършва със сложност **O(log(mx))** а на всеки въпрос да бъде отговаряно със сложност **O(log(mx)*log(manx))**.

Индексно дърво

В конкретния случай това представлява пълно двоично дърво с определена височина - **h**. Върховете на това двоично дърво са **2^(h+1)-1**. Основното в тази динамична структура е това, че всеки връх пази някаква информация за определен интервал, а децата му пазят същата информация за 2 подинтервала, които не се застъпват и напълно покриват целия бащински интервал. Примерно бащата може да пази информация за интервала (32 99) а децата за интервали (32 39), (33 99). Може и следните интервали (32,66), (66,99). А може и за следните (32 55), (77,99) какъвто е случаят в нашата задача. Макар и на пръв поглед те да не покриват целия интервал те може да покриват само важните точки от него (в нашия случай координатите, в които има къщи). Удобен начин за представяне на тази структура е масив с големина **2^(h+1)**. Това представяне много наподобява представянето на двоична пирамида в масив. Корена на дървото стои в клетка **1**. Децата му стоят в клетки **2** и **3**. Децата на връх **i** стоят в клетки **2i** и **2i+1**. Познаваме, че даден връх е листо ако е по-голям или равен на **2^h=mx**.

При решаване на задачи с такъв вид индексно дърво много често е задължително преди това да прочетем целия вход и да построим дървото вземайки предвид интервалите, за които ще трябва да пазим информация. Ако не можехме да направим това, тогава трябваше да използваме друга структура (балансирано дърво), която е твърде тежка (особено ако трябва да поддържа операциите които се искат в нашата задача) за състезание.

В задачата, във всеки момент има само определен брой къщи и почти никога не присъстват всички. Но нищо не ни пречи да считаме, че винаги имаме всички къщи построени и когато наистина ги няма просто да ги считаме за пусти (т.е. че съдържат 0 жители). По нататък в решението ще използвам означението **K[l]** за координатата на **l**-тата къща.

Сега да построим дървото за нашата задача. Нека всяко листо пази информация за интервала на една единствена къща (т.е. затворен интервал с дължина 0). Тъй като винаги имаме **2^h** листа, а броят на къщите може да не е толкова то можем да сложим фиктивни къщи с по 0 жители, които изобщо не влияят на задачата. Коренът на дървото ще пази информация за интервала **K[0]..K[2^h-1].** Освен координатите на крайните къщи в дървото си пазя и координатата на средната къща или по точно **K[(0+2^h-1)/2]**. Структурата на един възел на дървото е следната:

```
struct node { int l,m,r;  bigtype sum,w; }
```

За яснота в обясненията ще считам, че дървото се пази в масив **h[]**.Текущия връх (или върха, за който става дума в момента) винаги бележа с **where**.

И за корена е изпълнено **h[1].r=K[0], h[1].l=K[2^h-1], h[1].m=K[2^(h-1)-1]**.

Всяко ляво дете има за ляв край левия край на бащата, а за десен средата на бащата. Всяко дясно дете има за десен край десния край на бащата а за ляв край къщата, която се намира непосредствено след средната къща на бащата. Т.е. ако интервала на бащата са координатите на къщите **(l,r)** то на лявото дете интервалът са координатите **(l,(l+r)/2)** а на дясното – **((l+r)/2+1,r)**. За всяко листо имаме **h[where].l=h[where].m=h[where].r**.

Дотук обясних структурните особености на дървото. Сега е ред на информацията, която се пази във всеки връх:

h[where].sum е броя на жителите живеещи в къщите намиращи се в интервала на върха. Този брой е динамичен – той се променя със времето, когато се заселват или изселват хора (във фиктивните къщи **h[where].sum** винаги ще е 0. По време на изпълнение на програмата винаги следните равенства ще са в сила (за да не е противоречиво дървото). **h[where].sum=h[where*2].sum+h[where*2+1].sum** за всяко **where<mx** (т.е за всеки връх - не лист).

h[where].w е цената, която трябва да заплатят жителите в интервала на върха ако трябва да се съберат в точка **h[where].l**. По време изпълнение на програмата следното равенство винаги трябва да е изпълнено за всеки лист: **h[where].w=0** тъй като никой не трябва да пътува, а за всекли не-лист следното:

```
h[where].w = h[where*2].w + (h[where*2+1].l-h[where].l) *
h[where*2+1].sum + h[where*2+1].w
```

Това е така тъй като **h[where*2].w** е цената за жителите в интервала на лявото дете да пропътуват до **h[where].l**, а **h[where*2+1].w** цената за хората от дясното дете да

пропътуват до **h[where*2+1].l**. Но тези които живеят в дясното дете **h[where*2+1].sum** трябва да пропътуват и разстоянието от **h[where*2+1].l** до **h[where].l** и ето откъде идва третото събираемо.

Сега нека обясня как да променяме структурата при всяко заселване/изселване. Целта ни е да намерим листа, който съдържа къщата, в която се заселват хората. Можем да направим рекурсивна функция, която получава като параметър номер на връх, координата на къща и стойност, която указва с колко човека ще се промени населението на дадената къща. Функцията преценява в кое от двете му деца се намира къщата като сравнява координатата й с **h[where].m** и влиза рекурсивно в съответното дете. При това за всеки посетен връх обновява стойността на **h[where].sum**. Когато се връщаме от рекурсията е добре също така да обновим стойността **h[where].w**. Можем да я обновим като използваме горепосочената формула, но можем и директно на пресметнем каква цена биха допринесли новодошлите хора. За всеки посетен връх пресмятаме новите стойности на информацията за интервала му. И тъй като няма връх, който сме обходили а не сме обходили баща му то новополученото дърво не е противоречиво.

Сега вече можем да обновяваме дървото, но да видим как това може да ни помогне да решим задачата. В нея се иска за дадена цена да намерим такава координата **X** така че стойността на функцията е максимално близка до дадено число.

Можем да вземем интервала **–manx..MIDDLE**. И нека вземем този **MIDDLE**, който ще ни направи функцията в този интервал строго намаляваща. Тъй като е строго намаляваща функция можем да приложим метода на двоичното търсене за да намерим такава точка **X**, която е максимално надясно, но стойността на функцията е **>=** от търсената. Тъй като може да нямаме съвпадение то трябва да пресметнем каква е стойността на функцията в точка **X–1** и от тези двете да изберем по доброто. Сега остава въпроса да намерим за дадено **X** стойността на функцията в тази точка без да обхождаме всички къщи. Точно тук е мястото и на индексното дърво.

Само че за целта ще са ни нужни 2. Като строяхме предното дърво ние за всеки интервал смятахме колко пари ще струва придвижването на всички хора от интервала на върха до левия му край. Но тъй като при нас за дадена точка ще трябва да се придвижват хора и отляво надясно лъм дадена точка, то трябва да направим дърво в което всеки връх пази за интервла си колко пари ще трябват на всички хора живеещи в него за да пропътуват до десния му край. За да не пишем нови функции за работа с това дърво можем да направим симетрично дърво. Т.е. за всяка къща **p** в симетричното дърво слагаме къща с координати **maxx–K[p]**. И обновявамв и двете дървета съответно за къщи с различни координати. И стойността **h[where].w** за 2-рото поддърво е все едно каква е цената за всички хора живеещи в интервала му да се съберат в десния му край на обърнатия интервал.

Сега да се върнем на търсенето на стойността на функцията в дадена точка. Разбивам тази процедура на две. В първата част търся цената за всички хора живеещи вляво от дадената точка (за целта ползвам второто индексно дърво). Във втората част търся цената за всички хора живеещи вдясно от дадената точка (за целта ползвам първото индексно дърво). Тъй като функциите за работа с двете дървета са еднакви, различават се само по точа че тези във второто дърво ги извикваме с обърнати координати, то можем да не мислим че имаме 2 дървета ами да се съсредоточим върху функциите за работа със дървото описано горе.

За да пресметна стойността всички хора живеещи вдясно от дадена точка да се съберат в нея използвам следната рекурсивна функция

find (where, left, right)

Това означава, че искам да намеря колко че струва за всички хора от интервала **[left,right]** да се съберат в точка **left**, като знам, че няма къща в този интервал която не се съдържа в интервала на **where**.

Ако **left>h[where].r** или пък **right<h[where].l** , отговорът е 0 тъй като в дадения интервал не живеят хора.

Ако **left<=h[where].l** и **right>=h[where].r** (интервала **[left,right]** покрива целия интервал на върха **where**) отговорът е **h[where].w+h[where].sum*(h[where].l-left)**. **h[where].w** е цената да се съберат всички хора от интервала **[left,right]** (които всъщност живеят в интервала на **h[where]**) в точка **h[where].l**, но те **h[where].sum** ще трябва да пропътуват още **(h[where].l-left)** докато стигнат до исканата точка.

Ако е възможно за функцията да съдържа къща от интервалите на двете му деца (но не и да покрива целите – този случай беше току що разгледан), **(right>=h[where*2+1].l & left<h[where*2+1].l)** отговорът е:

```
find(where*2,left,h[where*2+1].l-1)  +
find(where*2+1,h[where*2+1].l,right)  +
(h[where*2+1].l-left)*h[where*2+1].sum
```

find(where*2,l,h[where*2+1].l-1) е цената за хората живеещи в сечението на интервалите на лявото поддърво **where*2** и на дадения интервал да се придвижат до точка лефт.

find(where*2+1,h[where*2+1].l,right) е цената на хората в интервала на дясното дете да се придвижат то лявата граница **h[where*2+1].l** на интервала на дясното дете.

(h[where*2+1].l-left)*h[where*2+1].sum – това е цената за същите тези хора **h[where*2+1].sum** трябва да се придвижат от **h[where*2+1].**l до **left**.

Ако пък къщите в дадения интервал лежат изцяло в интервала на дясното поддърво (**left>=h[where*2+1].l**) или пък изцяло в интервала на лявото поддърво (**right<h[where*2+1].l**) тогава мога спокойно да извикам функцията за съответния подинтервал - **find(where*2+1,l,r)** и **find(where*2,l,r)** съответно.

Със извикване на функцията **find(1,X,maxx)** за оригиналното дърво и на функцията **find(1,maxx-X,maxx)** за второто дърво можем да намерим каква цена трябва да платят всички хора заедно за да се придвижат до точка **X**. Този алгоритъм ще работи дори и за **X<0** или пък **X>maxx**. Сложността на функцията **find()** е **O(log(mx))**. Доказателството на този факт оставям за читателя.

И заедно с двоичното търсене сложността на отговарянето на въпрос става **O(log(mx)*log(manx))**. Отбелязвам, че двоичното търсене трябва да се пусне и за другата половина от графиката. Разсъжденията там са аналогични.

Задача А2. КОЛИ

Магистрала минава през еднопосочния град **X**. При достигане на града, тя се влива в градската пътна мрежа на едно от кръстовищата (наричано начално) и продължава към следващия град от друго кръстовище (наричано крайно). Градската пътна мрежа се състои от **N** кръстовища и **M** улици. Всяка улица свързва две различни кръстовища и е еднопосочна, т.е. възможно е преминаване на автомобил само в едната посока. От началното кръстовище може да се стигне до всяко друго, като се използва само градската мрежа. От всяко кръстовище може да се стигне до крайното кръстовище, пак използвайки само градската мрежа. Всички пътища водят до крайното кръстовище, т.е. по които и улици да се движим, рано или късно, ще стигнем до крайното кръстовище. Понеже улиците са много тесни е забранено изпреварването, т.е. два автомобила намиращи се в началото на улица не могат да излязат от улицата в обратен ред.

По магистралата се организира автомобилно състезание с **K** автомобила и в един от кръговете състезателите трябва да преминат през града **X**. За да направят състезанието в градската част по интересно, организаторите решили да поставят на участниците условие. Участниците влизат в града един по един (никои двама участника не влизат в града по едно и също време) през началното кръстовище. Този който първи влезе в града е с номер 1, втория – с номер 2 и т.н. Условието е, участниците да излязат от града в обратен ред т.е. първо да излезе състезателят с номер **K**, после състезателя с номер **K–1** и т.н., докато накрая излезе състезателя с номер 1.

Организаторите искали да привлекат колкото може повече състезатели, но се оказало, че ако има много коли, те няма да могат да изпълнят условието. Затова искат да знаят какъв е максималният брой участници, които могат да участват, така че да могат да се изпълни условието. Понеже отговорът може да бъде много голям, трябва да бъде намерен остатъкът му при деление с 987654321. Напишете програма **CARS**, която да решава тази задача.

Вход: На първия ред на стандартния вход ще бъде зададено числото **N**, ($2 \le N \le 500000$). Всеки от следващите **N** реда описва улиците, които излизат от едно кръстовище. Първото число в **(i+1)–**вия ред ще бъде броят U_i на улиците, които излизат от **i**-тото кръстовище. Следват U_i числа, всяко от които е номер на кръстовище такова, че има улица от кръстовището с номер **i** до това кръстовище. Общият брой на улиците няма да надвишава 5 000 000.

Изход: На единствения ред на стандартния изход трябва да се изведе търсеният максимален брой по модул 987654321.

ПРИМЕР:

Вход	**Изход**	**Вход**	**Изход**
5 2 2 3 1 3 2 4 5 1 5 0	4	9 2 2 4 1 4 2 1 5 1 8 2 6 9 2 7 9 1 9 1 9 0	5

Решение:
Сега ще докажа, че отговорът на задачата е равен на броя на пътищата от началния до крайния връх. Нека си припомним как се пресмята този брой:
Сортираме топологично върховете като началния връх е първи в получената редица. Сега нека въведем масива **A[]. A[l]** ще показва колко различни пътя има от началния връх до връх номер **l**. Нека топологичното сортиране е в масива **T[]**. Инициализираме **A[T[1]]=1**. Алгоритъма почва от **2** и върви до **N** по масива **T[]**.За всеки елемент **T[i]** смята **A[T[i]].** А то е равно на сумата на всички **T[j],** за които съществува ребро **j->i**.
Сега ще покажа, че отговорът на задачата **<=** броя на всички пътища. Нека приемем, че не е. Тогава от принципа на Дирихле ще следва, че има две коли с еднакъв път – а тогава значи или че те не са се изпреварили и не са в обратен ред или че са се изпреварили на някоя улица, което е забранено => не е възможно 2 коли да имат еднакви пътища => отговорът на задачата **<=** броя на всички пътища.
Сега ще покажа начин за придвижване на толкова коли. Нека до даден връх **k1** са се придвижили първите **A[k1]** коли в обратен ред и са продължили малко по някое друго ребро излизащо от **k1**. Нека това ребро е такова че веднъж продължили по него колите не могат да се върнат върхове **k1,k2,.....kl**. Нека това е така за върхове **k1,k2....kl** (само че до **A[k2]** са се придвижили следващите **A[k2]** коли, до **k3** следващите **A[k3]** коли и т.н.). Това е възможно да стане тъй като колите продължили напред от **k1** не пречат на колите пристигащи в **k2...kl**. И нека от тези върхове има ребро към връх **B** и всички коли са избрали да продължат по него. Тогава нека първо влязат в **B** и продължат по някое ребро колите от **kl**, после от **k(l-1)** и т.н. Така се получи че колите излязоха от **B** в обратен ред а броят на колите излезли от **B** е равен на сумата от **A[k1]+..+..+A[kl]**, но пък тази сума е равна на **A[B]**. Следователно за всеки връх **B** могат **A[B]** коли да дойдат от началния и да преминат през **B** в обратен ред.
Отговорът на задачата се намира в **A[T[N]]**. При извършване на всички сметки е необходимо да се пресмята по модул, тъй като числата могат да станат много големи.

```
#include<stdio.h>
#include<vector>
using namespace std;
#define pb push_back
#define sz size()
#define in stdin;//fopen("c:\\tp\\tests\\cars.in","r")
FILE*fn;

#define mx 512
#define base 1000000000
#define lb 9
#define type long long
struct L
{int ss[mx];
L(){for(int q1=0;q1<mx;q1++)ss[q1]=0;}
L(int a){for(int q1=mx-1,c1=a;q1>=0;q1--){ss[q1]=c1%base;c1/=base;}};
};
int dif(const L&a,const L&b)
  {int q1;
  for(q1=0;q1<mx;q1++)if(a.ss[q1]!=b.ss[q1])break;
  if(q1==mx)return 0;
```

```
  return a.ss[q1]-b.ss[q1];}
int operator<(const L&a,const L&b){int c1=dif(a,b);return c1<0;}
int operator>(const L&a,const L&b){int c1=dif(a,b);return c1>0;}
int operator==(const L&a,const L&b){int c1=dif(a,b);return c1==0;}
int operator!=(const L&a,const L&b){int c1=dif(a,b);return c1!=0;}
int operator<=(const L&a,const L&b){int c1=dif(a,b);return c1<=0;}
int operator>=(const L&a,const L&b){int c1=dif(a,b);return c1>=0;}
L operator+(const L&a,const L&b)
  {L c;
  for(int q1=mx-1,c1=0;q1>=0;q1--)
    {c1+=a.ss[q1]+b.ss[q1];
    c.ss[q1]=c1%base;
    c1/=base;}
  return c;}
L operator-(const L&a,const L&b)
  {L c;
  if(b>a)return b-a;
  for(int q1=mx-1,c1=0,c2=0;q1>=0;q1--)
    {c1=a.ss[q1]-b.ss[q1]-c2;
    c2=0;
    if(c1<0){c1+=base;c2=1;}
    c.ss[q1]=c1;}
  return c;}
L operator*(const L&a,int b)
  {L c;
  type c1=0;
  for(int q1=mx-1;q1>=0;q1--)
    {c1+=(type)a.ss[q1]*(type)b;
    c.ss[q1]=(int)(c1%(type)base);
    c1/=(type)base;}
  return c;}
L operator/(const L&a,int b)
  {L c;
  for(int q1=0,c1=0;q1<mx;q1++)
    {c1+=a.ss[q1];
    c.ss[q1]=c1/b;
    c1=(c1%b)*base;}
  return c;}
int operator%(const L&a,int b)
  {type c1=0;
  for(int q1=0;q1<mx;q1++)
    {c1+=a.ss[q1];
    c1=(c1%(type)b)*(type)base;}
  return (int)(c1/(type)base);}
L operator*(const L&a,const L&b)
  {L c;
  int q3;
  type c1=0;
  for(int q1=mx-1;q1>=0;q1--)for(int q2=mx-1;q2>=0;q2--)if(q1+q2-mx+1>=0)
    {c1=(type)a.ss[q1]*(type)b.ss[q2];
    for(q3=q1+q2-mx+1;q3>=0&&c1!=0;q3--)
      {c1+=c.ss[q3];
```

```
      c.ss[q3]=c1%(type)base;
      c1/=base;}}
  return c;}
L operator^(L a,L b)
  {L d=1,c;
  while(a%2==0&&b%2==0)
    {a=a/2;
    b=b/2;
    d=d*2;}
  while(a>0)
    {if(a%2==0)a=a/2;
    else if(b%2==0)b=b/2;
    else
      {c=(a-b)/2;
      if(a<b)b=c;else a=c;}}
  return d*b;}
L read(FILE*fn)
  {int q1,c1,c2,q2,c3;
  char d[mx*lb];
  L c;
  fscanf(fn,"%s",d);
  for(q1=0;d[q1];q1++);
  c1=q1;
  for(q1=c1-1,c2=0,c3=mx-1;q1>=0;q1-=lb,c2=0)
    {for(q2=lb-1;q2>=0;q2--)if(q1-q2>=0)c2=c2*10+d[q1-q2]-48;
    c.ss[c3--]=c2;}
  return c;}
int write(L a)
  {int q1;
  for(q1=0;q1<mx&&a.ss[q1]==0;q1++);
  if(q1==mx)q1--;
  printf("%d",a.ss[q1]);
  for(q1++;q1<mx;q1++)printf("%0*d",lb,a.ss[q1]);
  printf("\n");
  return 0;}

vector < vector < int > > nei;
vector < vector < int > > back;
vector < L > res;

int c,m,q1,q2,c1,c2,c3;

vector < int > top;
vector < bool > v;

int dfs(int a)
  {int q1;
  v[a]=1;
  for(q1=0;q1<nei[a].sz;q1++)if(!v[nei[a][q1]])dfs(nei[a][q1]);
  top.pb(a);
  return 0;}
int main()
```

```
{fn=in;
fscanf(fn,"%d",&c,&m);
nei.resize(c);
back=nei;
v.resize(c);
res.resize(c);
for(q1=0;q1<c;q1++)
  {fscanf(fn,"%d",&c1);
  for(q2=0;q2<c1;q2++)
    {fscanf(fn,"%d",&c2);
    c2--;
    nei[q1].pb(c2);
    back[c2].pb(q1);}
  }
fclose(fn);
for(q1=0;q1<c;q1++)if(back[q1].sz==0)break;
dfs(q1);
res[top[c-1]]=1;
for(q1=c-2;q1>=0;q1--)
  for(q2=0;q2<back[top[q1]].sz;q2++)
    res[top[q1]]=res[top[q1]]+res[back[top[q1]][q2]];
write(res[top[0]]);
return 0;}
```

Задача А3. ИЗРАЗ

Киро и приятелите му играят странна игра. Кодират математически израз с 0 и 1 в строго определен формат. Всеки от играчите има право да промени израза с максимум **K** замени. Печели този от тях, който след като се пресметне променения израз получи, в резултат на пресмятането на израза, максимално число. Изразът е дефиниран с правилата:

<**израз**>:=<цифра><знак><цифра><знак>...<цифра>

<**цифра**>:=0,1,2,3,4,5,6,7,8,9 <**знак**>:=+,-

Записването на цифрите и знаците в израза с 0 и 1 става по следния начин (удебеленият шрифт е за да личат по ясно нулите и единиците) :

0	1	2	3	4	5	6	7	8	9	+	–
111	00**1**	**111**	**111**	**101**	**111**	**111**	**111**	**111**	**111**	000	000
10**1**	00**1**	00**1**	00**1**	**1**0**1**	**1**00	**1**00	00**1**	**1**0**1**	**1**0**1**	0**1**0	000
10**1**	00**1**	**111**	**111**	**111**	**111**	**111**	00**1**	**111**	**111**	**111**	**111**
10**1**	00**1**	**1**00	00**1**	00**1**	00**1**	**1**0**1**	00**1**	**1**0**1**	00**1**	0**1**0	000
111	00**1**	**111**	**111**	00**1**	**111**	**111**	00**1**	**111**	**111**	000	000

Под промяна на израз с **K** замени се разбира инвертирането (замяна на 1 с 0 и обратно) на **K** цифри, така че общият брой на единиците в новия израз да е равен на броя на единиците в оригиналния израз. При промяна на израз, знак може да се трансформира само в знак и цифра може да се трансформира само в цифра, т.е. не може цифра да стане знак и обратното. В израз участва минимум една цифра и максимум 50. Пример за промяна на израз с 4 замени (с удебелен шрифт са промените):

Изразът преди промяната	Изразът след промяната
111000**111**	111000**00**1
100000000**1**	10**1**000001
11111100**1** <=> 5-7 = -2	111111001 <=> 8-1 = 7

00**1**000000**1**	**1**01000001
111000000**1**	111000001

Напишете програма **EXPRES,** която по зададен първоначален израз и число **K** пресмята колко е максималното число, на което може да се оцени променен израз ако се направят максимално **K** замени.

Вход: Данните са записани в 6 реда на стандартния вход. На първия ред стоят две числа: **N** - броят на цифрите участващи в израза и **K** - броят на максимално разрешените замени. На следващите 5 реда е записан изразът.

Изход: На първия ред на стандартния вход програмата трябва да запише максималната възможна стойност на променения израз, на останалите 5 реда – променениият израз. Ако има няколко решения програмата да изведе кое да е от тях.

ПРИМЕР

Вход:	**Изход:**
3 4	11
1110001110001 11	111000**00**1000111
1000000010000 01	100000001**0**0001
1111110011110 01	111111001111001
0010000010000 01	001000001**0**0001
1110000010000 01	111000001000001

Решение:

Задачата се решава динамично.

1. Прочитат се входните данни.
2. Изчисляват се разликата между всеки две от числата от -9..9 като се слага допълнително и -0.
3. Създава се тримерен масив по max[1..N,0..K,-K..K], които попълваме динамично: първо се движим по цифрите, после по промените до сега и накрая по баланса на 1-ците т.е. дали досега сме ползвали повече 1-ци или са ни останали в излишни. На всяка стъпка минаваме през всички числа от -9 до 9 (като се специални случаи са -0 и първата цифра в израза, която няма знак) и трансформираме текущото число от израза до него с разлика d и разлика в броя на 1-ците r тогава max[currentNumber,currentDifference,currentBalance] = max(max[currentNumber,currentDifference,currentBalance],max[currentNumber-1,currentDifference - d,currentBalance - r])
4. Накрая извеждаме максимума по i от max[N,i,0]

```
{$H+}
uses SysUtils;

Var
   FIn,FOut : Text;

Const
   BitMap : array[0..11] of string =
      ('111101101101111', // 0
       '001001001001001', // 1
       '111001111100111', // 2
       '111001111001111', // 3
       '101101111001001', // 4
       '111100111001111', // 5
```

```
        '111100111101111', // 6
        '111001001001001', // 7
        '111101111101111', // 8
        '111101111001111', // 9
        '000010111010000', // +
        '000000111000000'  // -
        );

    PLUS_INDEX : integer = 10;
    MINUS_INDEX : integer = 11;

    MAXN = 100;
    MAXK = 50;

Var
    numbers : array [1..MAXN] of integer;
    signs : array [1..MAXN] of integer;
    inputStrings : array [1..MAXN * 2] of string;
    N,K : integer;
    max : array [1..MAXN,0..MAXK,-MAXK..MAXK] of integer;
    solution : array [1..MAXN,0..MAXK,-MAXK..MAXK] of string;
    diff : array[0..15,0..15] of integer;
    ones : array[0..15] of integer;

Function GetDifference(index1,index2 : integer) : integer;
Var
    i,sum : integer;
    s1,s2:string;
Begin
    s1 := BitMap[index1];
    s2 := BitMap[index2];
    sum := 0;
    for i := 1 to length(s1) do
        if s1[i] <> s2[i] then
            inc(sum);
    GetDifference := sum;
End;

Function GetSignBitMapIndex(sign : integer) : integer;
Begin
    if sign = 1 then GetSignBitMapIndex := PLUS_INDEX
    else GetSignBitMapIndex := MINUS_INDEX;
End;

Function GetOnesCount(index:integer) : integer;
Var
    i,sum : integer;
    s: string;
Begin
    s := BitMap[index];
    sum := 0;
    for i := 1 to length(s) do
```

```
      if s[i] = '1' then inc(sum);

   GetOnesCount := sum;
End;

Procedure Init;
Var
   i,j : integer;
Begin
   for i := 0 to 11 do
      for j := 0 to 11 do
         diff[i,j] := GetDifference(i,j);

   for i := 0 to 11 do
      ones[i] := GetOnesCount(i);
End;

Procedure Solve(var result : integer; var solutionString : string);
Var
   numIndex,diffIndex,balIndex : integer;
   sign,number : integer;
   currentDiff,currentBalance : integer;
   oldDiff,oldBalance : integer;
   currentPath : string;
   currentValue : integer;

Begin
   for numIndex := 1 to N do begin
      for diffIndex := 0 to K do begin
         for balIndex := -K to K do begin
            max[numIndex,diffIndex,balIndex] := -10000;
            sign := 1;
            while sign >= -1 do begin

               if (sign = -1) and (numIndex = 1) then break;

               for number := 0 to 9 do begin
                  currentDiff :=
                     diff[GetSignBitMapIndex(sign),GetSignBitMapIndex(signs[numIndex])] +
                     diff[number, numbers[numINdex]];

                  if currentDiff > K then continue;

                  oldDiff := diffIndex - currentDiff;

                  if (oldDiff > K) or
                    (oldDiff < 0) then
                     continue;

                  currentBalance :=
                     ones[GetSignBitMapIndex(sign)] -
                     ones[GetSignBitMapIndex(signs[numIndex])] +
```

```
                ones[number] -
                ones[numbers[numIndex]];
            oldBalance := balIndex - currentBalance;

            if (oldBalance < -K) or
              (oldBalance > K) then
                continue;

            currentValue := sign * number;

            if numIndex > 1 then begin
                currentPath := Chr( Ord('0') + number);
                if sign = -1 then
                    currentPath := solution[numIndex-1,oldDiff,oldBalance] + '-' + currentPath
                else
                    currentPath := solution[numIndex-1,oldDiff,oldBalance] + '+' + currentPath;

                currentValue := currentValue + max[numIndex - 1,oldDiff,oldBalance];
            end
            else begin
                if (currentDiff <> diffIndex) or
                  (currentBalance <> balIndex) then continue;

                currentPath := Chr( Ord('0') + number);
            end;

            if currentValue > max[numIndex,diffIndex,balIndex] then
            begin
                max[numIndex,diffIndex,balIndex] := currentValue;
                solution[numIndex,diffIndex,balIndex] := currentPath;
            end;
          end;
          sign := sign - 2;
        end;
      end;
    end;
  end;

  result := -1000;
  for diffIndex := 0 to K do begin
    if result < max[N,diffIndex,0] then begin
      result := max[N,diffIndex,0];
      solutionString := solution[N,diffIndex,0];
    end;
  end;
End;

Procedure ReadInput;
Var
  lineIndex,i,j,stringIndex,sign : integer;
  line : string;
```

```
Begin
   assign(FIn,'Expression.inp');
   reset(FIn);
   readln(FIn,N,K);

   for lineIndex := 1 to 5 do begin
      Readln(FIn,line);
      stringIndex := 0;
      for i := 1 to (N*2 - 1) * 3 do begin
         if i mod 3 = 1 then inc(stringIndex);
         inputStrings[stringIndex] := inputStrings[stringIndex] + line[i];
      end;
   end;

   stringIndex := 1;
   for i := 1 to N do begin
      sign := 1;

      if i > 1  then begin
         if inputStrings[stringIndex] = BitMap[MINUS_INDEX] then // - sign
            sign := -1;
         inc(stringIndex);
      end;

      for j := 0 to 9 do
         if BitMap[j] = inputStrings[stringIndex] then begin
            numbers[i] := j;
            signs[i] := sign;
            break;
         end;

      inc(stringIndex);
   end;

   close(FIn);
End;

Function GetCharBitMapIndex(c:char) : integer;
begin
   if c in ['0'..'9'] then
      GetCharBitMapIndex := ord(C) - ord('0')
   else if c = '+' then
      GetCharBitMapIndex := PLUS_INDEX
   else
      GetCharBitMapIndex := MINUS_INDEX;
end;

Procedure WriteSolution(result : integer; solution : string);
Var
   line,i,j : integer;
   s : string;
begin
```

```
   assign(FOut,'expression.out');
   rewrite(FOut);
   writeln(FOut,result);
   for line := 0 to 4 do begin
      for i := 1 to length(solution) do begin
         s := BitMap[ GetCharBitMapIndex(solution[i])];
         for j := 1 to 3 do
            write(FOut,s[ line * 3 + j]);
      end;
      writeln(FOut);
   end;
   close(FOut);
End;

var
   r : integer;
   s : string;

Begin
   ReadInput;
   init;
   solve(r,s);
   WriteSolution(r,s);
End.
```

ТЕМА ЗА ГРУПА В

Задача В1. БЛИЗКИ ДУМИ

Крайната азбука **A** се състои от главните латински букви, арабските цифри и подчертаващо тире. За всеки знак a от **A** имаме зададено цяло число $V(a)$ – видимост на знака, $0 \le V(a) \le 100$. Видимостта $V(a)$ на знака a е числова оценка за това, колко е вероятно човек да пропусне този знак при четене. Например, буквата **'W'** е много по-видим знак от подчертаващото тире (**'_'**). За всеки два знака **a** и **b** от **A** е зададено цяло число $D(a,b)$ – различност на двата знака, $0 \le D(a,b) \le 100$, $D(a,b) = D(b,a)$ и $D(a,a)=0$. Различността $D(a,b)$ е толкова по-малка, колкото е по-вероятно знакът a да бъде сбъркан със знака **b** при четене. Например, различността между буквата '**O**' и цифрата '**0**' е малко число. Да разгледаме низа $\alpha=a_1a_2...a_m$ над **A**. Дефинираме следните "претеглени" операции върху низа α:

insert – вмъкване на знак на произволна позиция в низа α. Тази операция има тегло $V(a)$, където a е вмъкнатият знак.

update – замяна на един знак от низа α с друг. Ако знакът a е заменен със знака b, теглото на операцията е $D(a,b)$.

delete – изтриване на знак. Тежестта на операцията е $V(a)$, където a е изтритият знак.

Разстояние между низовете $\alpha=a_1a_2...a_m$ и $\beta=b_1b_2...b_m$ наричаме минималната сума от тегла на редица от операции, чрез които от α може да се получи β. Напишете програма **LDIST**, която при зададени $V(a)$ и $D(a,b)$ да намира разстоянието между два низа α и β.

ЗИМНИ МАТЕМАТИЧЕСКИ ПРАЗНИЦИ – РУСЕ' 2006

Вход: На първия ред на стандартния вход е зададен броят **P** на буквите, за които **V(a)** ≠100. На всеки от следващите **P** реда е зададена по една двойка: **a V(a)**, разделени с интервал. Следва описанието на **D(a,b)**. На отделен ред е зададен броят **Q** на двойките знаци, чиято различност не е 100. На всеки от следващите **Q (Q≤100)** реда има по една тройка **a b D(a,b)**, разделени с по един интервал. Следват два реда с низовете α и β. Последният ред на стандартния вход е празен, т.е. след β има знак за нов ред. Дължините на α и β са между 1 и 1023 знака.

Изход: Разстоянието между α и β да се изведе на стандартния изход.

ПРИМЕР:

Вход:	**Изход:**
1 _ 50 3 O 0 50 1 I 50 A 4 10 BIOGRAPHY B10_GR4PHIE	360

Решение:

Това е модификация на задачата за намиране на минимално разстояние при редактиране на низове (minimum edit distance). Решението и тук става с динамично оптимиране.

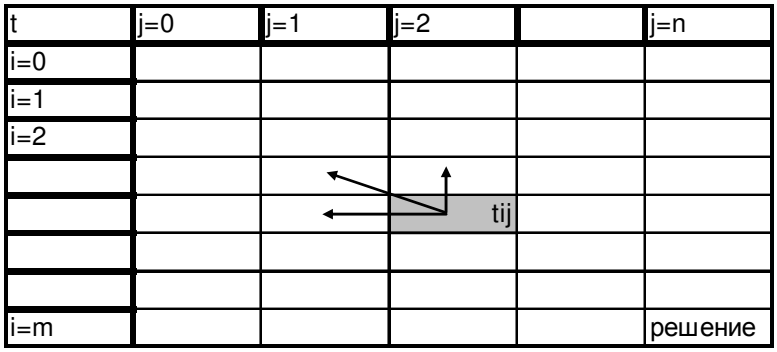
Оригиналната задача (без тежести на операциите insert, update, delete) се решава с помощта на двумерна $(m+1)\times(n+1)$ табличка t, в която t_{ij} е разстоянието между префиксите $a_0a_1\ldots a_{i-1}$ и $b_0b_1\ldots b_{j-1}$:

$$\begin{cases} t_{00} = 0 \\ t_{i0} = t_{i-1,0} + 1 = i \\ t_{0j} = t_{0,j-1} + 1 = j \\ t_{ij} = \min\{t_{i-1,j} + 1, \quad t_{i,j-1} + 1, \quad t_{i-1,j-1} + \varepsilon(a_{i-1}, b_{j-1})\} \end{cases}$$

където

$$\delta(x, y) = \begin{cases} 0, & x \neq y \\ 1, & x = y \end{cases}$$

Решението на цялата задача е t_{mn}



При варианта с тежести рекурентната връзка изглежда така:

$$\begin{cases} t_{00} = 0 \\ t_{i0} = t_{i-1,0} + V(a_{i-1}) \\ t_{0j} = t_{0,j-1} + V(b_{j-1}) \\ t_{ij} = \min\{t_{i-1,j} + V(a_{i-1}),\quad t_{i,j-1} + V(b_{j-1}),\quad t_{i-1,j-1} + D(a_{i-1}, b_{j-1})\} \end{cases}$$

```
#include <stdio.h>
#include <string.h>

#define MAX_N 1024

char v[256];       // Values are between 0 and 100
char d[256][256];  // Values are between 0 and 100
char alpha[MAX_N]; // No need for "unsigned char" - acceptable chars have ascii codes less than
128
char beta[MAX_N];
int t[MAX_N][MAX_N];

int min3(int x, int y, int z) {
   return (x < y)
      ? ((x < z) ? x : z)
      : ((y < z) ? y : z);
}

void main() {

   // Input: visibility
   memset(v, 100, sizeof(v));
   int p;
   scanf("%d", &p);
   for (int i = 0; i < p; i++) {
      unsigned char s[1];
      int x;
      scanf("%s%d", s, &x);
      int ch = s[0];
      v[ch] = x;
   }

   // Input: difference
   memset(d, 100, sizeof(d));
   for (int i = 0; i < (sizeof(d) / sizeof(d[0])); i++) {
      d[i][i] = 0;
   }
   int q;
   scanf("%d", &q);
   for (int i = 0; i < q; i++) {
      unsigned char s1[1];
      unsigned char s2[1];
      int x;
      scanf("%s%s%d", s1, s2, &x);
      int ch1 = s1[0];
```

```
    int ch2 = s2[0];
    d[ch1][ch2] = d[ch2][ch1] = x;
  }

  // Input: alpha and beta
  scanf("%s%s", alpha, beta);
  int nAlpha = strlen(alpha);
  int nBeta = strlen(beta);

  // Calculate distance
  t[0][0] = 0;
  for (int i = 1; i <= nAlpha; i++) {
    t[i][0] = t[i - 1][0] + v[alpha[i - 1]];
  }
  for (int j = 1; j <= nBeta; j++) {
    t[0][j] = t[0][j - 1] + v[beta[j - 1]];
  }
  for (int i = 1; i <= nAlpha; i++) {
    for (int j = 1; j <= nBeta; j++) {
      int a = alpha[i - 1];
      int b = beta[j - 1];
      t[i][j] = min3(
          t[i - 1][j - 1] + d[a][b],  // update
          t[i][j - 1] + v[b],         // insert
          t[i - 1][j] + v[a]          // delete
      );
    }
  }

  // Output
  printf("%d\n", t[nAlpha][nBeta]);
}
```

Задача B2. Y1984

Годината е 0x1984 (или 6532 по старото десетично летоброене). На света са останали само три, постоянно воюващи помежду си държави, и, като служител на тайните служби на Евразия, ти имаш задачата да шпионираш тайните агенти на останалите две. Знае се, че на територията на страната има **N** чужди тайни агента, с кодови имена Агент 0 ... Агент **N–1**, но не се знае кой за коя от вражеските държави работи. За щастие разполагаш със записи на телефонни разговори между някои от тях. От всеки записан разговор става ясно дали говорещите са приятели (от една и съща държава) или врагове. От вас се иска да напишете програма **Y1984**, с помощта на която да можете да отговаряте на въпроса: „Приятели или врагове са агентите **X** и **Y**". Възможните отговори са „приятели", „врагове" или „недостатъчно информация".

Вход: На първите два реда от стандартния вход са дадени числата **N** и **M** – броят на тайните агенти и броят на записаните телефонни разговори, съответно. На всеки от следващите **M** реда има по една тройка числа **A**, **B**, **R**, разделени с по един интервал, където **A** и **B** са номера на агенти, а **R** е връзката между тях. **R**=0 за „приятели", и **R**=1 за „врагове". Следват въпросите. На отделен ред е даден броят им **Q**. На всеки от следващите **Q** реда има

по един „въпрос", който се състои от номерата **X** и **Y** на агентите, за които е зададен въпросът. Входните данни са коректни и непротиворечиви.

Изход: За всеки въпрос от входа на отделен ред на стандартния изход програмата трябва да се изведе отговора като число: 0, ако **X** и **Y** са приятели, 1 – ако са врагове, 2 – ако информацията не е достатъчна за да се даде един от предните два отговора.

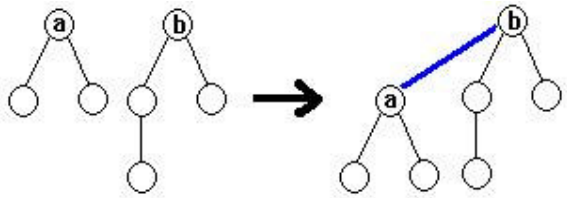
ПРИМЕР:

Вход:	**Изход:**
4 3 0 1 1 1 2 0 2 3 1 1 0 3	0

Решение:

Това решение ползва една структура от данни представяща разбиване на множество, която позволява лесно да се извършва операцията обединение на две групи от разбиването – наричат я "disjoint sets" или "union find".

Всяка група се представя като дърво от елементи (пазим си само бащата на всеки възел), а коренът му се явява „официалния представител", или „големия брат", или „вожда" на всички под него. Когато обединяваме групи, просто закачаме вожда на едната за вожда на другата – така че да станат баща и син.



Можем да намерим вожда на произволен елемент като тръгнем нагоре по дървото до корена. Два елемента са в една и съща група от разбиването точно когато имат един и същ вожд.

Смисълът който влагаме е следния: разбиваме множеството от агенти на групи, в рамките на всяка от които сме сигурни че агентите са приятели. Това представяне е удачно, защото приятелството (~) е релация на еквивалентност:

- Аз съм приятел със себе си (`a~a`)
- За двама приятели чувствата са взаимни (`a~b => b~a`)
- Приятелите на моите приятели са и мои приятели (`a~b & b~c => a~c`)

Трябва ни и начин да представяме знанието си че две групи приятели са врагове помежду си. Враждата (`*`) при този „двуполюсен модел" (има само две чужди държави) не е нищо друго освен антоним на приятелството:

- Ако не си ми приятел, значи си ми враг, и обратно (`a~b <=> !(a*b)`)

От изброените по-горе принципи, могат да се изведат интересни следствия:

- За двама врагове чувствата са взаимни (`a*b => b*a`)
- Враговете на моите приятелите са и мои врагове (`a~b & b*c => a*c`)
- Враговете на моите врагове са ми приятели (`a*b & b*c => a~c`)

Доказват се лесно с допускане на противното.

И така, оказва се достатъчно да си пазим най-много по един враг за група приятели. Няма смисъл да пазим по два врага, защото ще е ясно че те са приятели помежду си и веднага можем да ги обединим.

За конкретната реализация ползваме два масива:

`friend[x]` – Бащата на `x` в дървото (гората) на приятелството.

`enemy[x]` – Ще го ползваме само ако `x` е вожд – `enemy[x]` сочи към вожда на вражеската група приятели. Ако не знаем враг за групата на `x`, тогава `enemy[x]` ще бъде `-1`. Винаги ще поддържаме масива такъв че ако `enemy[x] == y`, то и `enemy[y] == x`.

Ще прилагаме последователно информацията от дадените телефонни разговори, т.е. ще прилагаме операциите: „направи `x` и `y` приятели", и „направи `x` и `y` врагове". Да означим с `getLeader(x)` функцията която ни връща вожда на `x`, а за всяка стъпка нека:

```
lx = getLeader(x);
ly = getLeader(y);
ex = enemy[lx];
ey = enemy[ly];
```

Разглеждаме следните случаи:

- „направи X и Y приятели"
  - ако `lx == ly`, те вече са си приятели, не правим нищо.
  - ако `enemy[lx] == ly`, нещо не е наред – стигнали сме до противоречие. Това не трябва да се случва според условието на задачата.
  - иначе, сприятеляваме `lx` и `ly`, определяйки едното от тях за вожд и закачайки другото за вожда (нека за определеност закачим `ly` за `lx`), и след това:
    - ако `ex != -1` и `ey != -1`, сприятеляваме `ex` и `ey`, а този от тях, който е нов вожд, го правим взаимен враг с `lx`.
    - ако точно едно от `ex` и `ey` е различно от `-1`, правим го взаимен враг с `lx`.
    - ако `ex == -1` и `ey == -1`, всичко е ok, не правим нищо.
- „направи X и Y врагове"
  - ако `enemy[lx] == ly`, те вече са си врагове, не правим нищо.
  - ако `lx == ly`, имаме противоречие.
  - иначе, правим `lx` и `ly` взаимни врагове, като преди това: ако `ex != -1`, сприятеляваме го с `ly`; и ако `ey != -1`, сприятеляваме го с `lx`.

И накрая, отговаряме на въпросите за отношенията произволни x и y:

- ако `getLeader(x) == getLeader(y)`, те са приятели.
- ако `enemy[getLeader(x)] == getLeader(y)`, те са врагове.
- иначе – не е ясно.

```
#include <stdio.h>
#include <assert.h>
#include <memory.h>

#define MAX_N 1024
```

```
int n;
int fri[MAX_N];      // "friend" is a reserved word in C++
int enemy[MAX_N];


int getLeader(int x) {
   int leader = x;
   while (fri[leader] != -1) {
      leader = fri[leader];
   }
   if (x != leader) {
      fri[x] = leader; // "Path compression" optimization
   }
   return leader;
}

void main() {
   scanf("%d", &n);
   memset(fri, -1, sizeof(fri));
   memset(enemy, -1, sizeof(enemy));
   int m;
   scanf("%d", &m);
   for (int i = 0; i < m; i++) {
      int a, b, r;
      scanf("%d%d%d", &a, &b, &r);
      a = getLeader(a);
      b = getLeader(b);
      int ea = enemy[a];
      int eb = enemy[b];
      if (r == 0) {
         // Must make them friends, unless they already are
         assert(ea != b);
         if (a != b) {
            // The groups of a and b merge, and b becomes the leader
            fri[a] = b;
            if (ea != -1) {
               if (eb == -1) {
                  // Since b is the new leader, he adopts the enemies of a
                  enemy[b] = ea;
                  enemy[ea] = b;
               } else {
                  // If a and b both have enemies, those enemies become friends
                  fri[ea] = eb;
               }
            }
         }
      } else {
         // Must make them enemies, unless they already are
         assert(a != b);
         if (ea != b) {
            if (ea != -1) {
```

```
            // The enemies of a become friends of b
            fri[ea] = b;
          }
          if (eb != -1) {
            // The enemies of b become friends of a
            fri[eb] = a;
          }
          enemy[b] = a;
          enemy[a] = b;
        }
      }
    }
    int q;
    scanf("%d", &q);
    for (int i = 0; i < q; i++) {
      int x, y;
      scanf("%d%d", &x, &y);
      x = getLeader(x);
      y = getLeader(y);
      if (x == y) {
        printf("0\n");
      } else if (enemy[x] == y) {
        printf("1\n");
      } else {
        printf("2\n");
      }
    }
}
```

Задача B3. ПРАЗНИК

Разглеждаме числовата права като път. В някои от целочислените точки по пътя може да има къщи (в една точка може да има много къщи), във всяка от които могат да живеят много хора. Така се е образувал град. Хората могат да се придвижват по този път, но трябва да плащат за това. Цената за да пропътува един човек единица разстояние по пътя е 1 лев. Тъй като това е доста скъпо, хората не пътуват често и надалеч. Градската управа се нуждае непрекъснато от пари, но за да не изглежда прекалено нагло, решила да ги събира като пътни такси. Затова периодически организира градски празници, в които участието е задължително. Всеки празник се прави в определена точка с целочислени координати. Жителите на града сами трябва да си осигурят превоза. Преди всеки празник управата внимателно трябва да избере мястото, за да може да си осигури предварително намислената сума. Напишете програма **FEST**, която да може да отговаря на въпроса: "Къде да се организира празника така, че парите които ще се съберат да са максимално близко до дадено число S?". Ще считаме, че в началото градът е празен.

Вход: Програмата трябва да чете данни от стандартния вход. На първия ред са зададени две числа **M N**, където **M** е броят на къщите, а **N** – броят на въпросите. Всеки от следващите **M** ред ще има вида **A B**, където **A** е координатата на къщата, а **B** – броят на хората, които живеят в нея. Следващите **N** реда са въпроси, като всеки въпрос съдържа по едно число **S -** сумата, която управата иска да събере. Максималният брой редове на входа е 200 000. Координатите **A** ще са в интервала [0,10 000 000], числата **B** – в интервала [0,1000], а сумата **S** – в интервала [0,10^15].

Изход: Програмата трябва да изведе резултатите на стандартния изход, като за всеки въпрос изведе по един ред. Отговорът се състои от две числа **E P**, където **E** (няма да надвишава по абсолютна стойност 10^8) е координатата на мястото на празника, а **P** е абсолютната стойност на разликата между исканите исканата сума и тази, която ще постъпи в касата от таксите. Ако има повече от един възможен отговор Вашата програма трябва да изведе кой да е от тях.

ПРИМЕР

Вход:	**Изход:**
2 2 3 7 11 5 123 36	-4 1 3 4

Решение:

Някои константи които ползвам в решението:

manx = 2^27 – максималната абсолютна стойност на координатата която може да бъде отговор на някой въпрос (това не означава, че непременно има такъв тест)

maxx = 2^24 – максималните координати на които може да има къщи (в тестовете това е по-малко)

mx = 2^17 – броя на листата в дървото ми (трябва да е по голям от максималния брой къщи)

За всеки въпрос ние трябва да намерим оптимална точка от числовата права. Тъй като на всяка точка можем да съпоставим цена (сбора от броя на хората във всяка къща умножен по разстоянието до дадената точка за всички къщи) можем да отговорим на всеки въпрос просто като обходим цялата права и намерим коя точка е оптималната. Но ако разгледаме тази функция то ще видим, че тя е нещо подобно на парабола – т.е. в двата си края функцията приема големи стойности, които намаляват към средата на графиката. Нека първо си отговорим на въпроса "Къде е тази среда на графиката". Нека сега се намираме в произволна точка **X** от числовата права и нека стойността на функцията в тази точка е **C**. Да се опитаме да намерим каква ще е стойността на функцията в точка **X+1**. Всеки човек, който живее в къща с координати **<=X** ще трябва да заплати **1** лев повече, но от друга страна всеки, който живее в къща с координати **>=X+1** ще трябва да заплати **1** лев по-малко. Следователно стойността на функцията в точка **X+1** е равна на **C+(броя на хората живеещи преди или на X) – (броя на хората живеещи след X)**. И това твърдение е вярно за всяка точка. Нека с **Dx** бележим изменението на стойността на функцията между точки **X** и **X+1**. Лесно се вижда, че **Dx > Dy** за **x>y**. Ако за момент си помислим за функцията **Dx** като за производна то лесно че намерим къде е средата (минимума) на функцията. Там където **Dx** е равно на **0**. За съжаление може да няма такава точка (тъй като работим с целочислени стойности), или пък може да има много такива точки. Затова ето къде се намира средата на функцията. Нека къщите са сортирани. Ако съществува къща **K**, такава че **(броя на хората живеещи в къщи 1..K) = (броя на хората живеещи в къщи K+1..N)**, то тогава функцията е минимална в целия затворен интервал **[K,K+1]**, като това са координатите на къщите. Ако пък няма такава къща то тогава съществува къща **K** такава, че **(броя на хората живеещи в къщи 1..K) > (броя на хората живеещи в къщи K+1..N)** и такава къща, че **K** е минимално. Тогава минимума на функцията е в тази точка (координатата на къщата). Нека отсега нататък средата на функцията наричаме

MIDDLE като това е или произволна точка от този интервал или къща лежаща в този интервал. Трябва да отбележа, че функцията е линейна между всеки две къщи.

Дотук представих две решения:

3) Тривиалното – за всяка точка от числовата права смятаме стойността и сравняваме с исканата стойност.
4) Смятаме последователно за всички точки като по този начин смятането на стойността на функцията за всяка следваща точка е за константно време.

Сега ще представя третото решение, което използва метода на двоичното търсене за да отговори на всеки въпрос, което прави сложността **O((M+N)logN)**.

Интерполация

Сега ще покажа алгоритъм за намиране на точка от графиката в даден линеен интервал (такъв между две последователни къщи) при условие, че знаем стойностите на функцията в двата края на интервала. Нека са дадени следните стойности:

C,ans1,ans2,sum1,sum2,a,b

a и **b** са съответно координатите на левата и дясната граница на интервала.

C е стойността, за която търсим координатата където се получава максимално близка стойност.

sum1 е броя на жителите вляво от **a**.

sum2 е броя на жителите вдясно от **b**.

ans1 е стойността на функцията в точка **a**.

ans2 е стойността на функцията в точка **b**.

Нека **x** е търсената координата.

Следното равенство важи за **C**.

C=ans1+(x-a)*(sum1-sum2) - това е стойността на функцията в точка **x**.

От тук стигаме до:

x=(C-ans1+a*sum1-a*sum2)/(sum1-sum2).

Тъй като това е целочислено деление реалният отговор за **X** е между **X** и **X+1** и затова трябва да проверим кое от двете е по близко до търсената стойност.

Двоичното търсене

За да правим двоично търсене първо намираме средата на функцията – **MIDDLE**. Нека в случая **MIDDLE** значи една от къщите, където се постига минимума на функцията. Първо правим двоично търсене в интервала **-maxx..MIDDLE**. Намираме такава къща K в този интервал, за която стойността на функцията в къща **K>=(търсената стойност)** а в къща **K+1<(търсената стойност).** Тогава вече с интерполация може да се намери къде точно между двете къщи се намира оптимума. Двоичното търсене трябва да се пусне и в другия интервал **MIDDLE..maxx**.

ТЕМА ЗА ГРУПА С

Задача С1. МОРЗОВ КОД

В далечната и не особено известна страна, наричана от хората Здравословна Монархия за Програмисти (накратко, ЗМП) е пълно със зимни курорти, позволяващи на вече

пенсионираните се програмисти да похарчат натрупания през годините упорита работа капитал, като се насладят на ски-спортове, сауна и масаж. В ЗМП всичко се управлява от компютри, включително и системата за направа на изкуствен сняг. Поради програмна грешка, обаче, системата е създала 255 пъти повече сняг от необходимото и в момента входовете на всички сгради са затрупани, а телефонната връзка и достъпа до Интернет са прекъснати. Един от програмистите едва успял да се свърже с Интернет по сателит, за да изпрати сигнал за помощ, тъй като батерията на лаптопа му била почти изтощена. Машините за почистване на сняг щели да пристигнат едва след няколко дни, а комуникацията между програмистите, намиращи се в различни сгради била сериозно застрашена.

Тъй като всички те имали GSM-телефони с вградени фенерчета, един от тях се сетил, че макар и да нямат обхват, могат да използват телефоните си, за да си разменят съобщения. Просто използвали светлинни сигнали и позабравения морзов код. Морзовият код е система за преобразуване на текст в сигнал и обратно, чрез използване на два прости символа: тире (-) и точка (.). Кодът носи името на създателя си Самюел Морз. Чрез дълги или кратки сигнали (звукови или светлинни) може да се предаде кратко съобщение. Например, "SOS" чрез морзов код се представя като "`... ––– ...`".

A	.-	H	....	O	---	V	...-	3	...--	0	-----
B	-...	I	..	P	.--.	W	.--	4	....-	Точка(.)	.-.-.-
C	-.-.	J	.---	Q	--.-	X	-..-	5	.....	Запетая(,)	--..--
D	-..	K	-.-	R	.-.	Y	-.--	6	-....	Тире (-)	-..-.
E	.	L	.-..	S	...	Z	--..	7	--...	Въпросителна(?)	..--..
F	..-.	M	--	T	-	1	.----	8	---..	Интервал в текста	.-.-.
G	--.	N	-.	U	..-	2	..---	9	----.	Край на съобщението	...-.-

Таблица на морзовите кодове

За съжаление, за да станеш гражданин на ЗМП е нужно да подпишеш клетвена декларация, че никога повече няма да пишеш програми. Ето защо на вас се пада задачата да напишете програма **MORSE**, преобразуваща текст в морзов код.

Вход: От стандартния вход се прочита текста, съдържащ малки и големи букви от латинската азбука, цифрите от 0 до 9, интервали, както и знаците точка, запетая, тире и въпросителна. Текстът е не по-дълъг от 255 знака.

Изход: За всеки знак от текста, на стандартния изход трябва да се извежда морзовия му код – последователността от тирета (−) и точки (.), както е дадена в таблицата по-горе. Еднаквите малка и голяма буква се кодират с една и съща морзова последователност. Кодовете на всеки два съседни знака трябва да са разделени с точно един интервал (ASCII код 40). Всеки две думи от текста трябва да се разделят с точно една морзова последователност за интервал, а в края да се изведе морзовата последователност за край на съобщението (вж. тези два кода в Таблицата).

ПРИМЕР:

Вход:	**Изход:**
`SOS - coordinates 20-06.`	`... ––– ... .–.–. –..–. .–.–. –.–. ––– ––– .–. –.. .. –. .– – . ... .–.–. ..––– ––––– –..– –––––  –.... .–.–.– ...–.–`

Решение:

1. Прочита се в string входът.
2. Всички двойни интервали се преобразуват в единични и това се повтаря, докато във входния string няма повече интервали.
3. Съгласно таблицата се прочита всеки символ и на екрана се извежда съответната последователност
4. Накрая се извежда ...-.-

```
const  NumCodes=41;   { Number of symbols to be coded }
       Symb:array [1..Numcodes] of char=
       ('A','B','C','D','E','F','G','H','I','J','K','L','M','N',
       'O','P','Q','R','S','T','U','V','W','X','Y','Z',
       '1','2','3','4','5','6','7','8','9','0','.',',','-','?',' ');
       Codes:array [1..Numcodes] of string[10]=
       ('.-','-...','-.-.','-..','.','..-.','--.','....','..','.---',
       '-.-','.-..','--','-.','---','.--.','--.-','.-.','...','-','..-',
       '...-','.--','-..-','-.--','--..','.----','..---','...--','....-',
       '.....','-....','--...','---..','----.','-----','.-.-.-','--..--',

       '-..-.','..--..','.-.-.');


var  s:string;
     change:boolean;
     i:byte;

procedure Convert(c:char);
{ Converts symbol into morse code }
var i:byte;
begin
  i:=0;
  repeat
    inc(i)
  until (c=Symb[i]) or (i>NumCodes);
  if i<=NumCodes then { If found... }
    write(Codes[i]+' ');
end;

begin
  readln(s);
  while pos('  ',s)>0 do          { Removes extra empty spaces }
    delete(s,pos('  ',s),1);
  for i:=1 to length(s) do        { Converts input string into morse codes }
    Convert(UpCase(s[i]));
  writeln ('...-.-');             { Message end string }
end.
```

Задача С2. РЕБУС

Киро е ученик и за домашно по математика всеки ден му дават да реши по един ребус. Понеже Киро е доста мързелив той се чуди дали не може да се напише програма, която решава ребусите вместо него. Може ли да му помогнете?

Ребус се създава по следните правила:

```
<ребус> := <израз>=<израз>
<израз> := <число><операция><число>...<операция><число>
<число> := <буква><буква>...<буква>
<буква> := 'A','B','C'...'Z'
<операция> := '+','-'
```

ЗИМНИ МАТЕМАТИЧЕСКИ ПРАЗНИЦИ – РУСЕ' 2006

Различните букви в ребуса са съпоставени на различни цифри и тъй като цифрите са десет в един ребус може да има най-много десет различни букви. Всяко число се състои от поне 1 и не повече от 100 букви. Не може число да започва с буква, която е съпоставена на цифрата 0. Всеки израз съдържа поне едно и не повече от 20 числа. Решение на ребуса наричаме такова съпоставяне на различни цифри на различните букви в ребуса, че да се получи вярно математическо равенство. Напишете програма **REBUS**, която по зададен ребус намира едно решение.

Вход: На единствения ред на стандартния вход ще бъде зададен коректен ребус. Входните данни са така подбрани, че винаги да има поне едно решение.

Изход: На стандартния изход програмата трябва да изведе, за всяка буква участваща в ребуса, по един ред от вида <**буква**>=<**цифра**>. Ако ребусът има много решения, програмата трябва да изведе кое да е от тях.

ПРИМЕР:

Вход:	**Изход:**
TWO+TWO=FIVE-ONE	E=0 F=2 I=3 N=4 O=5 T=9 V=7 W=1

Решение:
Задачата се решава чрез backtrack, като се използват няколко оптимизации:

1. Прочитат се входните данни.
2. Ребусът се преобразува като всички числа от дясно на знака за равенство се прехвърлят от ляво с обратен знак, т.е. ребусът се свежда до <израз>=0
3. Асоциирането на буква с цифра се извършва по време на самото пресмятане на израза: първо се започва от цифрите на единиците, минава се последователно през числата от израза и ако дадена буква е асоциирана с цифра се преминава на следващото число, ако не е се пробват всички възможни цифри за нея и се продължава нататък. След като се пресметне цифрата на единиците на целия израз и остатъка, ако цифрата е различна от 0 то това означава че ребуса е некоректен и се прави връщане назад, ако е 0 се повтаря същата процедура за цифрите на десетиците, стотиците и т.е. като се вземе предвид и остатъка от сумата на предните цифри.
4. Резултатът се извежда на изхода.

```
{$H+}
Uses SysUtils;
Type
   TNumber = record
      sign : integer;
      digits : string;
   End;
Var
   FIn,FOut : Text;
   numbers : array [1..64] of TNumber;
   N : integer;
```

```
  canBeZero : array ['A'..'Z'] of boolean;
  values : array ['A'..'Z'] of integer;
  used : array [0..9] of boolean;
  maxLength : integer;
Procedure ReadInput;
Var
  rebus : string;
  sign : integer;
  currentSign : integer;
  i : integer;
Begin
  assign(FIn,'rebus.inp'); reset(FIn);
  readln(FIn,rebus);
  close(FIn);
  rebus := rebus + '+';
  sign := 1;
  currentSign := 1;
  N := 1;
  numbers[1].sign := 1;
  numbers[1].digits := '';
  for i := 1 to length(rebus) do
  begin
    if rebus[i] in ['A'..'Z'] then begin
      numbers[N].digits := rebus[i] + numbers[N].digits;
    end
    else begin
      if rebus[i] = '=' then begin
        sign := -1;
        currentSign := -1;
      end
      else if rebus[i] in ['-','+'] then begin
        if rebus[i] = '-' then
          currentSign := -1 * sign
        else
          currentSign := sign;
      end;
      if rebus[i-1] <> '=' then begin
        inc(N);
        numbers[N].sign := currentSign;
        numbers[N].digits := '';
      end;
    end;
  end;
  dec(N);
End;
Procedure Init;
Var
  i : integer;
  c : char;
Begin
  fillchar(used,sizeof(used),0);
  fillchar(canBeZero,sizeof(canBeZero),true);
```

```
   for i := 1 to N do
   begin
      canBeZero[ numbers[i].digits[ length(numbers[i].digits) ] ] := false;
      if length(numbers[i].digits) > maxLength then
         maxLength := length(numbers[i].digits);
   end;

   for c := 'A' to 'Z' do
   begin
      values[c] := -1;
   end;
End;

Function BackTrack(letterIndex,numberIndex,sum : integer) : boolean;
Var
   num : TNumber;
   c : char;
   dig : integer;
   result : boolean;

Begin
   if letterIndex > maxLength then begin
      BackTrack := sum = 0;
      exit;
   end;

   if numberIndex > N then begin
      if sum mod 10 = 0 then
         BackTrack := BackTrack(letterIndex+1,1,sum div 10)
      else
         BackTrack := false;
      exit;
   end;

   num := numbers[numberIndex];

   if length(num.digits) < letterIndex then
   begin
      BackTrack := BackTrack(letterIndex,numberIndex + 1, sum);
      exit;
   end;

   c := num.digits[letterIndex];
   if values[c] <> -1 then begin
      BackTrack := BackTrack(letterIndex,numberIndex + 1, sum + values[c] * num.sign);
      exit;
   end
   else begin
      for dig := 0 to 9 do begin
         if not used[dig] then begin
            if (dig = 0) and (not canBeZero[c]) then
```

```
            continue;

          used[dig] := true;
          values[c] := dig;
          result := BackTrack(letterIndex,numberIndex + 1,sum + dig * num.sign);
          if result then begin
            BackTrack := true;
            exit;
          end;
          used[dig] := false;
          values[c] := -1;
        end;
      end;
    end;
    BackTrack := false;
End;

Procedure WriteOutput;
Var
    c : char;
Begin

    assign(FOut,'rebus.out');
    rewrite(FOut);

    for c := 'A' to 'Z' do
      if values[c] <> -1 then
        writeln(FOut,c,'=',values[c]);
    close(FOut);
End;

Begin
    ReadInput;
    Init;

    if BackTrack(1,1,0) then
      WriteOutput
    else
      writeln('NOPE');
End.
```

Задача С3. ТЕЛЕФОН

Старите мобилни телефони, които Сладкодумчо Вртисметков и приятелите му си купиха на специалната промоция от първите Зимни Математически Празници през 1980 г., се повредиха. Телефоните са много стар модел и набирането става само с бутоните от 0 до 9, някои от които вече не функционират. Точно сега, обаче, Сладкодумчо непременно трябва да се обади на приятелката си Бърборанка Неспирова, за да я покани на среща. Уви, изобщо не се знае дали ще може да й позвъни директно, тъй като номерът й може да съдържа цифри от неработещи бутони. Сладкодумчо може да звънне на някои от приятелите си и да ги помоли те да се обадят на Бърборанка и да й предадат за тяхната среща (или пък да ги помоли те да звъннат на някой друг от приятелите и да го помолят същото и т.н.). Освен това

ЗИМНИ МАТЕМАТИЧЕСКИ ПРАЗНИЦИ – РУСЕ' 2006

Сладкодумчо много е закъсал с финансите и би искал съобщението му да стигне по най-евтиния начин до Бърборанка.

Приятелите на Сладкодумчо са колкото услужливи, толкова и стиснати – Сладкодумчо ще трябва да им плати до стотинка всички телефонни разходи за направената услуга. Най-неприятното е, че приятелите на Сладкодумчо Вертисметков също имат неговия проблем – и техните телефони са купени от същата промоция и затова и на техните апарати някои от бутоните не работят.

И тъй, на вас се пада тежката задача да напишете програма **PHONE** с която да откриете най-евтиния начин Сладкодумчо да покани приятелката си на среща. Счита се, че всеки предава съобщението за точно една минута.

Вход: На първия ред на стандартния вход е зададено цяло число **N** (**N**≤100) – броят на приятелите на Сладкодумчо. Вторият ред съдържа описание на телефона на Сладкодумчо, а всеки от следващите **N** – на телефоните на приятелите му. Всяко описание съдържа числата: **K**, **M**, **P**, p_1, p_2, … , p_L, където: **K** е цяло число с не повече от 10 цифри – телефонният номер на съответния човек; **M** е реално число, не по-голямо от 5 и с не повече от две цифри след десетичната точка – цената, която съответният човек плаща за минута разговор; **L** е цяло от 1 до 10 – броят на все още работещите бутони; числата $p_1, p_2, \ldots, p_L$ са цифри от 0 до 9, подредени във възходящ ред и показващи работещите бутони на телефона на съответния човек. Последният ред на стандартния вход съдържа телефонния номер на Бърборанка. Тестовете са така зададени, че винаги има възможност съобщението да достигне до Бърборанка.

Изход: Програмата трябва да изведе на стандартния изход намерената минимална сума, с два знака след десетичната точка.

ПРИМЕР:

Вход:	**Изход:**
3 0888123456 1.20 5 0 2 3 8 9 0889023023 0.5 10 0 1 2 3 4 5 6 7 8 9 0898283089 0.2 3 4 0 8 166 0 10 0 1 2 3 4 5 6 7 8 9 0898400900	1.70

Решение:

Всъщност, задачата се свежда до намиране на стандартен път в граф, като трябва да се изпълнят следните стъпки:

1. Прочитат се входните данни.
2. Всеки възел има идентификация, съответстваща на телефонния номер на дадения човек.
3. Създава се булева функция f(number, list), където за списък от цифри list се определя дали може с тях да бъде получено числото number.
4. С използване на функцията f се откриват връзките между различните възли в графа.
5. Теглото на връзките между възлите е цената за минута разговор.
6. Прилага се алгоритъм за най-кратък път в граф с тегла.
7. Дължината на най-краткия път се извежда на изхода.

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R+,S+,T-,V+,X+,Y+}
{$M 65384,0,655360}
const MaxN=102;                { Maximum number of users }
    InfiniteRate=maxint;       { A rate value bigger than the maximum }
```

```
type TPhoneUser=record
                  Num:string;       { Phone number }
                  Butt:set of char; { Set of buttons that work }
                  Rate:real;        { Price for outgoing call }
                end;

var a:array [1..MaxN] of TPhoneUser;       { Info about all users }
    b:array [1..MaxN,1..MaxN] of integer;  { Currently found rates }
    n:byte;                                { Number of users }

procedure Read_Data;
{ Reads the input data }
var i,j,k,l:byte;
    c:string;

  procedure ReadString(var s:string);
  { Reads the phone number as string }
    var c:char;
    begin
      s:='';
      read(c);
      while c in ['0'..'9'] do
      begin
        s:=s+c;
        read(c);
      end;
    end;

begin
n:=0;
readln(n);
fillchar(a,sizeof(a),0);
inc(n); { Include Sladkodumcho }
for i:=1 to n do
  begin
    ReadString(a[i].Num);          { Reads the phone number }
    read(a[i].Rate);               { Reads the rate }
    a[i].Butt:=[];                 { Reads the buttons into a set }
    read(k);
    for j:=1 to k do
      begin
        read(l);
        str(l,c);
        include(a[i].Butt,c[1]);
      end;
    readln;
  end;
inc(n);                        { Include Barboranka }
ReadString(a[n].Num);          { Reads Barboranka's number }
end;

procedure Find_Path;
```

```
var i,j,k:byte;

 function CanCall(x,y:byte):boolean;
 { Checks if user X is able to call user Y }
  var i:byte;
      bool:boolean;
  begin
    bool:=true;
    for i:=1 to length(a[y].Num) do
      if not ((a[y].Num[i]) in (a[x].Butt)) then bool:=false;
    CanCall:=bool;
  end;

begin
{ Set rates between all the users }
for i:=1 to n-1 do
 for j:=1 to n do
   if CanCall(i,j)
     then b[i,j]:=round(a[i].Rate*100)
     else b[i,j]:=InfiniteRate;
{ Checks for a better rate from user I to user J through user K }
for i:=n-1 downto 1 do
 for j:=2 to n do
   if (i<>j) then
     for k:=2 to n-1 do
       if (i<>k) and (k<>j)
          and (b[i,k]<InfiniteRate)
          and (b[k,j]<InfiniteRate) then
            if (b[i,k]+b[k,j]<b[i,j]) then
              b[i,j]:=b[i,k]+b[k,j];
end;

procedure Show_Result;
begin
writeln((b[1,n]/100):0:2);
end;

begin
Read_Data;
Find_Path;
Show_Result;
end.
```

ТЕМА ЗА ГРУПА D

Задача D1. ЙОДА

Йода е 66-сантиметрово същество, познато ни от сагата „Междузвездни войни". Освен, че е старши джедай, на иврит „йода" означава „който знае".

ЗИМНИ МАТЕМАТИЧЕСКИ ПРАЗНИЦИ – РУСЕ' 2006

Йода говори език, наречен „галактически бейсик" и се отличава с това, че често разменя словореда в изреченията си. Например, вместо „Аз съм Йода, старши джедай", той би казал „Джедай старши, Йода съм аз."

Тъй като за много цивилизации неговият език остава неразбран, Йода отправя предизвикателство към юбилейното състезание „Зимни математически празници". Напишете програма, която извежда думите от дадено изречение в обратен ред.

Йода е едно от най-мъдрите същества във Вселената, затова е изпратил и алгоритъм за решаването на задачата:

1. **От клавиатурата ред да прочетеш от входни данни трябва.**
2. **Символ по символ обърни ти после реда цял. Например „АЗ СЪМ ЙОДА" става „АДОЙ МЪС ЗА".**
3. **Във всяка дума поотделно после обръщай символите ти.**
4. **И „ЙОДА СЪМ АЗ" накрая трябва да се появи.**

И нека Силата да бъде с теб!

В трансгалактическото съобщение от Йода, пише, че и авторът на най-добре написаната програма ще получи точки на състезанието и безплатно обучение за джедай.

Вход: От клавиатурата се въвежда едно изречение (не повече от 255 символа), съдържащо само главни букви от A до Z. Думите в изречението са разделени с по точно един интервал.

Изход: Програмата трябва да изведе изречението с разменени думи.

ПРИМЕР:

Вход:	**Изход:**
AZ SAM YODA	YODA SAM AZ

Решение:

Тъй като алгоритъмът е описан в задачата, той трябва просто да бъде реализиран.

```
var s,ss:string;
    i,len:byte;
    start,wlen:integer;

procedure swap(var a,b:char);
var c:char;
begin
c:=a;
a:=b;
b:=c;
end;

begin
  readln(s);
  len:=length(s);
  { Reverse the whole string }
  for i:=(len div 2) downto 1 do
    swap(s[i],s[len-i+1]);
  { Reverse each word }
  start:=-1;
  repeat
  start:=start+wlen+1;
  wlen:=0;
```

```
  while (s[start+wlen+1]<>' ') and (start+wlen<=len) do inc(wlen);
  ss:=copy(s,start+1,wlen);
  for i:=length(ss) div 2 downto 1 do
    swap(ss[i],ss[length(ss)-i+1]);
  write(ss);
  if start+wlen<=len then write(' ') else writeln;
  until start+wlen>len;
end.
```

Задача D2. ПЕРДЕ

Марийка си купила ново перде за вкъщи. За да изглежда добре окачено, тя иска кукичките, на които се закачва пердето, да са на равни разстояния една от друга. В горния край на пердето има зашити халки на равни разстояния, за които могат да се закачват кукичките. Разбира се, за да не виси пердето от двата края, първата кукичка трябва да се закачи за първата халка, а последната кукичка - за последната халка. Нека номерираме халките с целите числа и най-лявата да получи номер едно.

Помогнете на Марийка, като напишете програма **PERDE**, която, по зададен брой на халките и брой на кукичките, определя номерата на тези халки, за които са закачени кукички.

Вход: От стандартния вход се въвеждат две цели положителни числа - **A** и **B**, където **A** е броя на халките, а **B** - броя на кукичките (1<**A**<1000, 1<**B**<**A**).

Изход: На стандартния изход се извежда на екрана нарастваща редица от номерата на тези халки, за които са закачени кукички.

Тъй като не винаги е възможно всички кукички да са закачени на халки, които се намират на равни разстояния една от друга, то е необходимо поне разликата между най-голямото и най-малкото разстояние между две съседни халки да бъде минимална.

ПРИМЕР:

Вход:	**Изход:**
10 4	1 4 7 10
11 4	1 5 8 11

Решение:

Задачата се свежда до намиране на частното и остатъка от делението на A на B.

```
#include <stdio.h>

int main(int argc, char **argv) {
   int n, k;
   scanf("%d %d", &n, &k);
   printf("n=%d k=%d\n", n, k);

   int m = n - k;
   int d = k - 1;
   int l = m % d;
   int s = m / d;
   printf("m=%d d=%d l=%d s=%d\n", m, d, l, s);

   int i = 0;
```

```
    int j = 1;
    for (; i <= d; i++)
    {
        printf("%d ", j);
        j += s + 1;
        if (i >= d - l)
        {
           j++;
        }
    }
    printf("\n");

    return 0;
}
```

Задача D3. MAX3

Зададени са **N** цели числа. Напишете програма **MAX3**, която избира три от тях, произведението на които е максимално.

Вход: Данните се четат от стандартния вход. На първия ред е разположено числото **N** – брой на числата от последователността ($3 \le \mathbf{N} \le 10^6$). На следващите редове са разположени и самите числа, които по абсолютна стойност не надхвърлят 30 000.

Изход: На стандартния изход се извеждат три числа, произведението на които е най-голямо. Числата се извеждат в произволен ред. Ако съществуват няколко различни тройки числа, които дават максимално произведение, се извежда коя да е от тях.

ПРИМЕР:

Вход:	**Изход:**
9 3 5 1 7 9 0 9 -3 10	9 10 9
3 -5 -30000 -12	-5 -30000 -12

Решение:

Ясно е, че поради големия брой на числата, тривиалният подход за сравняване на произведенията на всички възможни тройки числа не е добро решение, защото е много бавно.

Затова е необходимо да се приложи по-добър алгоритъм. Ако последователността съдържа само неотрицателни числа, то е ясно, че максималното произведение ще се получи от трите най-големи числа в нея. Ще ги означим с max1, max2 и max3.

Но в нашия случай последователността може да съдържа и отрицателни числа, което изисква по-подробен анализ. Произведението на две отрицателни числа е положително и затова е необходимо да намерим двете най-малки отрицателни числа - min1 и min2. Тогава максимално произведение ще получим като умножим max1 с по-голямото от max2*max3 или min1*min2. Тъй като произведението на три цели числа може да стане много голямо, за да не се получи препълване (самото произведение в задачата не се търси), удачно е да се сравняват най-напред max2*max3 и min1*min2.

Ако последователността се състои само от отрицателни числа, то максималното произведение (отрицателно) ще се получи отново от max1, max2 и max3.

```
var n, i, k, max1, max2, max3, min1, min2: longint;
begin
  max1:=-30000; max2:=max1; max3:=max1;
  min1:=30000; min2:=min1;
  readln(n);
  for i:=1 to n do begin
    readln(k);
    if k>max1 then begin
      max3:=max2; max2:=max1; max1:=k;
    end
    else if k>max2 then begin
           max3:=max2; max2:=k;
         end
         else if k>max3 then max3:=k;
    if k<min1 then begin
      min2:=min1; min1:=k;
    end
    else if k<min2 then min2:=k;
  end;
  if (max1>0) and (min1*min2>max2*max3) then writeln(max1,' ', min1,' ',min2)
  else writeln(max1,' ',max2,' ',max3);
end.
```

ТЕМА ЗА ГРУПА Е

Задача Е1. ЖИВОТИНСКА ЗАДАЧА

Кучетата и котките живеят по-кратко от нас хората, но и техният живот преминава през различни фази – бебе, дете, тийнейджър, средна възраст и старост... Интересно би било да знаем на колко човешки години се равнява възрастта на нашите домашни любимци.

Много хора смятат, че правилото 7 кучешко-котешки години са равни на 1 човешка дава точни резултати. Това, за съжаление, е погрешно и неточно изчисление... Например, повечето животни (кучета или котки) могат да имат малки на 1 година, докато е трудно си представим 7-годишно дете да бъде родител...

Всъщност, първата година се равнява на 15 човешки. Втората се равнява на 9 човешки. Всяка кучешко-котешка година от 3 до 18 се равнява на 4 години, а над 18 – на 3 години...

Така, например, ако 19-годишен домашен любимец беше човек, той щеше да е на 15+9+4+4+4+4+4+4+4+4+4+4+4+4+4+4+4+4+4+3 = 91 човешки години.

Напишете програма **DOGGY**, която по зададена възраст на домашен любимец да изчислява човешката му възраст.

Вход: От клавиатурата се въвежда едно цяло число от 0 до 25 – реалната възраст на домашния любимец.

Изход: На екрана се извежда едно единствено цяло число – човешката възраст на домашния любимец.

ПРИМЕР:

Вход:	**Изход:**

3	28
19	91

Решение:
Вариант 1 - с използване на масив:

```
const  ay: array[1..25] of byte: ( 15,9,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3);
var i, aa, ha: byte;
begin
  readln(aa); ha:=0;
  for i:=1 to aa do ha:=ha+ay[i];
  writeln(ha);
end.
```

Вариант 2 - без масиви.

```
var i, aa, ha: byte;
begin
  readln(aa); ha:=0;
  if aa>0 then ha:=ha+15;
  if aa>1 then ha:=ha+9;
  if aa>2 then for i:= 3 to aa do ha:=ha+4;
  if aa>18 then for i:=19 to aa do ha:=ha+3;
  writeln(ha);
end.
```

Задача Е2. МНОЖЕСТВА

Разглеждаме 6 множества, в които са включени някои от числата от 1 до 63:

{1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63}

{2,3,6,7,10,11,14,15,18,19,22,23,26,27,30,31,34,35,38,39,42,43,46,47,50,51,54,55,58,59,62,63}

{4,5,6,7,12,13,14,15,20,21,22,23,28,29,30,31,36,37,38,39,44,45,46,47,53,54,55,56,60,61,62,63}

{8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31,40,41,42,43,44,45,46,47,56,57,58,59,60,61,62,63}

{16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63}

{32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63}

Всяко от тези числа може да бъде еднозначно определено като е известно на кои от горните множества то принадлежи.

Напишете програма **SETS.EXE**, която прави това. Принадлежността на едно число към всяко от множествата се задава с 1, а непринадлежността - с 0.

Вход: От клавиатурата последователно се въвежда информацията за принадлежността на предварително избрано число към горните множества (0 или 1 на отделен ред).

Изход: Определеното от програмата число.

ПРИМЕР:

Вход:	**Изход:**
1 1	19

0 0 1 0	
0 0 1 0 0 1	36

Решение:

Последователността от нули и единици, които задават принадлежността на избраното число към всяко от дадените множества, всъщност е двоичното представяне на това число. Ето защо е достатъчно да преобразуваме това двоично число в десетично и да получим търсеното число.

```
var A, B, C, i: byte;
begin
  C:=1;
  for i:=1 to 6 do begin
     readln(A); B:=B+A*C; C:=C * 2;
  end;
  writeln(B)
end.
```

Задача Е3. СНЕЖИНКА.

Зима е… Навсякъде вали сняг и професор Снежин Снежинков е решил да си направи машина за снежинки с различни размери… Проблемът е, че му трябва програма, която изчертава снежинки с размери от 1 до 10 символа. Напишете програма **SNOW** в помощ на професора.

Вход: От клавиатурата се въвежда едно число **N** – големина на снежинката

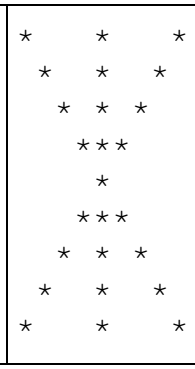
Изход: Снежинка с големина **N**.

ПРИМЕР:

Вход:	**Изход:**
1	`  *`

2	`***`
	`  *`
	`***`

3	`*  *  *`
	`  ***`
	`   *`
	`  ***`
	`*  *  *`

5	

Решение:

```
var n:byte;

procedure SnowFlake(x:byte);
{ Draws a snowflake with size x }
var i:byte;
begin
  for i:=1 to n-x do
    write (' ');
  if x=1 then writeln('*') else
    begin
      write('*');
      for i:=1 to x-2 do
      write (' ');
      write('*');
      for i:=1 to x-2 do
      write (' ');
      writeln('*');
      SnowFlake(x-1);
      for i:=1 to n-x do
      write (' ');
      write('*');
      for i:=1 to x-2 do
      write (' ');
      write('*');
      for i:=1 to x-2 do
      write (' ');
      writeln('*');
    end;
end;

begin
readln(n);
SnowFlake(n);
end.
```