Caderno de Provas – *Pão de Queijo*

1ª Seletiva Interna – 2018/1

UDESC–Joinville e IFMS–Aquidauana

Servidor BOCA (Arena Joinville):
http://200.19.107.67/boca/

Organização e Realização:

Claudio Cesar de Sá (coordenação geral), Peter Laureano Brendel (coordenação técnica), Daniela Marioti, Gabriel Hermann Negri , Lucas Hermann Negri, Rogério Eduardo da Silva, Gilmário Barbosa dos Santos, Mateus Boiani, Diego Buchinger, Roberto Rosso.

Patrocinador 2018: Linx

Lembretes:

- Aos *javaneiros*: **o nome da classe deve ser o mesmo nome do arquivo a ser submetido**. Ex: **classe petrus**, nome do arquivo **petrus.java**;
- Exemplo de leitura de entradas que funcionam:

```
Java: (import java.util.Scanner)
Scanner in = new Scanner(System.in);
ou
Scanner stdin = new Scanner(new BufferedReader(new InputStreamReader(System.in)));

C: (#include <stdio.h>)
int integer1; scanf("%d", &integer1);

C++: (#include <iostream>)
int integer1; std::cin >> integer1;
```

  Exemplo de saída de entradas:

```
Java: System.out.format("%d %d\n", integer1, integer2);
C: printf("%d %d\n", integer1, integer2);
C++: std::cout << integer1 << " " << integer2 << std::endl;
```

- É permitido consultar livros, anotações ou qualquer outro material impresso durante a prova;
- A correção é automatizada, portanto, **siga atentamente as exigências da tarefa quanto ao formato da entrada e saída conforme as amostras dos exemplos**. Deve-se considerar entradas e saídas padrão;
- Todos os compiladores (Java, Python, C e C++) são padrões da distribuição Ubuntu versão 16.04 (gcc C11 habilitado);
- Procure resolver o problema de maneira eficiente. Se o tempo superar o limite pré-definido, a solução não é aceita. As soluções são testadas com outras entradas além das apresentadas como exemplo dos problemas;
- Teste seu programa antes de submetê-lo. A cada problema detectado (erro de compilação, erro em tempo de execução, solução incorreta, formatação imprecisa, tempo excedido ...), há penalização de 20 minutos. O tempo é critério de desempate entre duas ou mais equipes com a mesma quantidade de problemas resolvidos;
- Utilize o *clarification* para dúvidas da prova. Os juízes podem **opcionalmente** atendê-lo com respostas acessíveis a todos;
- Algumas interfaces estão disponíveis nas máquinas Linux, que podem ser utilizada no lugar da *Unity*. Para isto, basta dar *logout*, e selecionar a interface desejada. Usuário e senha: *udesc*;
- Ao pessoal local: **cuidado com os pés sobre as mesas para não desligarem nenhum estabilizador/computador de outras equipes**!
- O nome *Pão de Queijo* é uma homenagem e chamada para Maratona Mineira de Programação, para 26/maio. Inscrições no URI-online, em breve!

Patrocinador e Agradecimentos

- Linx – Patrocinador oficial do ano de 2018;
- DCC/UDESC;
- Aos bolsistas deste ano pelo empenho;
- Alguns, muitos outros anônimos.

Pão de Queijo

1ª Seletiva Interna da UDESC e IFMS–Aquidauana – 2018

05 de maio de 2018

Conteúdo

Atenção quanto aos nomes e números dos problemas!!!

1 Problema A: Árvores da Floresta

`Arquivo: arvores.[c|cpp|java|py]` `Tempo limite: 2 s`

O jogo "Desenhe uma Floresta" é muito popular entre os jovens. Neste jogo, o número de participantes é praticamente ilimitado, e quase todos vencem. O jogo é simples: no início, o líder do jogo descreve rapidamente a imagem de uma floresta que ele viu recentemente. Então, os jogadores recebem papéis e gizes de cera e então passam a reproduzir a imagem descrita da melhor forma possível. Cada jogador que entregar ao menos uma imagem parcial de qualquer região de qualquer floresta no planeta, não importa o estilo de representação, ganha uma pequena recompensa na forma de um chocolate ou fruta.

Neste problema, você deve implementar uma imitação deste jogo. Você é muito mais experiente que os jovens jogadores, então seu desenho deve corresponder exatamente às especificações. A imagem que você deve reproduzir é de uma clareira, isto é, uma região de árvores que não é tão populosa quanto a floresta densa que a cerca. A imagem deverá ser impressa no estilo "ascii-art".

- Uma imagem $M \times M$ é representada por uma tela com M linhas, cada linha com M caracteres. Cada *pixel* na imagem é representado por um caractere ASCII imprimível nesta tela.
- As coordenadas dos *pixels* na imagem correspondem às coordenadas dos caracteres na tela. **As coordenadas dos *pixels* no canto inferior esquerdo e no canto superior direito da imagem são** $(0, 0)$ **e** $(M-1, M-1)$, respectivamente. A coordenada x do pixel no canto inferior direito da imagem é $M-1$.
- Cada *pixel* na imagem representa ou grama ou uma parte de uma árvore ou de um toco de árvore. Um *pixel* representando grama é descrito pelo caractere "." (ponto, código ASCII 46) na tela. Árvores e tocos de árvores são representados por mais *pixels*, como definido a seguir.
- Uma árvore possui uma altura positiva S e consiste de quatro partes: as **raízes**, o **tronco**, os **galhos** e a **copa**.
- As raízes são representadas por três caracteres adjacentes: O *underscore*, a barra vertical e o *underscore* ("_|_", códigos ASCII 95, 124, 95).
- O tronco é representado por S barras verticais adjacentes ("|", código ASCII 124), e é localizado imediatamente acima do centro das raízes das árvores.
- Os galhos consistem de S galhos imediatamente à esquerda do tronco e S galhos imediatamente à direita do tronco. Todos os galhos são adjacentes ao tronco da árvore.
- Cada galho esquerdo é representado por uma barra ("/", código ASCII 47), enquanto que cada galho direito é representado por uma barra invertida ("\", código ASCII 92).
- A copa é representada por um único caractere, o acento circunflexo ("^", código ASCII 94), localizado imediatamente acima do caractere mais alto do tronco da árvore.
- Um toco de árvore consiste em três *pixels* adjacentes representados pelos caracteres *undercore*, a letra minúscula "*o*" e *undercore* ("_o_", códigos ASCII 95, 111, 95).

Note que uma árvore ou toco de árvore podem aparecer na imagem somente de forma parcial ou podem até não aparecer na imagem, dependendo das suas coordenadas. Veja um exemplo de dados abaixo para uma melhor ilustração deste caso.

Entrada

A entrada é composta por vários casos de teste. Cada caso inicia com uma linha contendo os inteiros M e N, separados por um espaço ($1 \leq M \leq 100, 1 \leq N \leq 10^5$). A seguir encontram-se N linhas, cada linha contendo uma tripla de inteiros S, X, Y, separados por espaços, que descrevem uma árvore ou toco de árvore. Os valores de X e Y representam as coordenadas do centro da uma raiz de uma árvore ou de um toco de árvore. No caso de $S = 0$, a tripla descreve um toco de árvore. No caso de $S > 0$, a tripla descreve uma árvore de altura S ($0 \leq S \leq 9, -10^9 \leq X, Y, \leq 10^9$).

Garante-se que nenhuma parte de diferentes árvores ou tocos de árvores estejam sobrepostas em um mesmo *pixel*.

O fim da entrada é dado por EOF

Saída

Para cada caso de teste, imprima a tela com a imagem da clareira. A impressão deverá ser decorada com uma borda feita de asteriscos (*). Imprima uma linha entre cada caso de teste.

Exemplo de Entrada	Exemplo de Saída
3 2 0 5 5 9 1 0 8 10 3 3 2 0 2 1 1 -1 -1 0 -1 2 3 0 6 6 4 7 0 7 4 3 8 -1 5 5 -5 9 2 -10	***** */|* */|* *_|_* ***** ********** *|\._|_..* *|_.^....* *../|\...* *../|\._o* *../|\...* *_._|_../* *._o_.^./* *\.^./|\/* **********

Explicando os exemplos

- Nas primeiras 3 linhas ocorre o primeiro caso de testes.
- A primeira linha, o número **3** indica o tamanho da imagem(**3**×**3**) e em seguida o número **2** indica que haverá duas árvores ou tocos.
- A segunda linha o número **0** representa um toco de árvore, e os números **5** e **5** a coordenada do centro do toco. (Note que o toco está fora do campo de visão da imagem)
- A terceira linha o processo se repete, o primeiro número indica a altura da árvore, os próximos dois, a coordenada do centro da raíz ou centro do toco.

2 Problema B: Blocos de Lego

`Arquivo: blocos.[c|cpp|java|py]` `Tempo limite: 2 s`

Quando Claudinho ainda era uma criança e estava aprendendo português, possuía 5 tipos de bloquinhos de brinquedo, cada um correspondendo a uma das 5 primeiras letras do alfabeto (maiúsculas).

Cada bloquinho pode ser representado como uma grade 5×3 composta pelos caracteres . e *. Os 5 tipos de bloquinhos se parecem com isso:

```
*** *** *** *** ***
*.* *.* *.. *.* *..
*** *** *.. *.* ***
*.* *.* *.. *.* *..
*.* *** *** *** ***
```

Claudinho está com uma palavra S com N letras, e gostaria de saber como ficaria a sequência de blocos correspondente, quando dispostos lado a lado.

Entrada

Em cada caso de testes a entrada é composta por duas linhas. A primeira contém um inteiro N ($1 \leq N \leq 2018$). A segunda linha contém uma palavra S de comprimento N, constituída somente pelas letras A, B, C, D e E.

Saída

Imprima a representação dos blocos que formam a palavra fornecida (a resposta conterá 5 linhas, cada uma com $3N$ caracteres).

Exemplo de Entrada	Exemplo de Saída
6 ABCDEA	********** *.**.**..*.**..*.* *******..*.******* *.**.**..*.**..*.* *.**********.*
10 ECADBECADB	****************************** *..*..*.**.**.**..*..*.**.**.* ****..****.********..****.**** *..*..*.**.**.**..*..*.**.**.* *******.**********.*******

3 Problema C: Combina

`Arquivo: combina.[c|cpp|java|py]` `Tempo limite: 2 s`

No mais novo jogo da Naointendo, Merio Odyssey, lançado em 2018, nosso querido carpinteiro finalmente pôde mudar a aparência clássica! Ao decorrer do jogo é possível concluir desafios e através deles liberar novas vestimentas.

Uma vestimenta consiste em um chapéu e uma camisa. Quando são liberados em um desafio, ambos são comuns a um tema.

JP é um jogador muito *hipster* e acredita que duas peças de mesmo tema não combinam e tornam Merio deselegante.

Problema

No início era fácil manter controle de quantas aparências diferentes e elegantes Merio poderia ter, mas após N desafios tornou-se inviável manter controle de todas as combinações possíveis. Para contornar este problema, Merio pediu para que você criasse um programa que mostre de quantas maneiras diferentes ele pode se manter elegante.

Entrada

A entrada consiste em um inteiro N ($1 \leq N \leq 2018$) que representa a quantidade de desafios concluídos por JP.

Saída

A saída esperada é a quantidade de combinações possíveis de vestimentas que não tornem Merio deselegante.

Exemplo de Entrada	Exemplo de Saída
1	0
2	2
5	20

4 Problema D: Dinossauros

`Arquivo: dinossauro.[c|cpp|java|py]` `Tempo limite: 5 s`

Yushi é um pequeno dinossauro, um dos últimos de sua espécie, ele mora embaixo de um tronco de madeira trazido de uma floresta de clima equatorial. O tronco é grande e úmido e atrai moscas, o que deixa Yushi muito satisfeito.

Ali também há uma linha de pedras que atravessa a zona úmida em frente ao tronco. Existem algumas manchas escuras nessas pedras e às vezes Yushi gosta de observá-las e imaginar que não são apenas manchas, mas sim moscas gigantes.

Ontem, um amigo de Yushi foi vê-lo e sugeriu que jogassem um jogo.

"*Está vendo aquelas manchas nas pedras?*- Perguntou o amigo. "*Eu te desafio a começar na pedra mais a esquerda e então ir pulando de pedra em pedra a fim de chegar o mais longe possível, mas com uma condição. Você só pode pular de uma pedra A para outra B se a soma das manchas na pedra A e na pedra B for igual à distância entre essas duas pedras!*"

"*Tudo bem, mas você sabe que eu só sei contar até vinte e três*- Disse Yushi.

"*Não tem problema, eu te ajudo com os números maiores- Respondeu o amigo.*

"*Então vai ser fácil!*-- Disse Yushi, enquanto se posiciona na primeira pedra e observa confuso para o resto da linha.

Problema

Você receberá uma sequência de quantidade de manchas escuras nas pedras da linha. Sua tarefa é encontrar qual a pedra mais distante possível de alcançar através de saltos consecutivos seguindo as regras do jogo.

O primeiro pulo inicia na primeira pedra, e um pulo entre duas pedras só será possível se e somente se a soma do número de manchas das pedras for igual à distância entre elas. Suponha que a linha de pedras é reta e que a distância entre duas pedras vizinhas é de exatamente uma UDD (Unidade de Distância Dinossauresca).

Entrada

Uma entrada é composta por vários casos de teste. A primeira linha de cada caso de teste é um inteiro N ($1 \leq N \leq 10^6$) representando a quantidade de pedras. A linha seguinte contem uma lista de N inteiros. A ordem desta lista é a mesma ordem das pedras na zona úmida, cada inteiro representa a quantidade de manchas escuras na pedra correspondente. Nenhuma pedra tem mais de 10^9 manchas. Suponha que o amigo de Yushi conheça todos os diferentes pares de pedras onde Yushi pode pular durante a sua sequencia de pulos. É garantido que essa quantidade de pares de pedras nunca ultrapassa 10^6.

A última entrada é uma linha contendo o número zero e indica o fim dos casos de teste.

Saída

Para cada caso de teste, imprima um único inteiro representando a distância da pedra mais distante possível de ser alcançada por Yushi seguindo as regras do jogo.

Exemplo de Entrada	**Exemplo de Saída**
7 2 1 0 1 2 3 3 11 7 6 1 4 1 2 1 4 1 4 5 0	5 10

5 Problema E: Esta *Sharon Stone*

`Arquivo: engenheira.[c|cpp|java|py]` `Tempo limite: 2 s`

A orla marítima de Balneário de Camboriú é repleta de prédios (ou edifícios), numerados de 1 a N. Para cada prédio $i \geq 2$, há uma passarela de mão dupla do edifício i a outro edifício x_i. Dois edifícios são chamados *vizinhos* se houver uma ponte ou passarela entre eles.

Cada edifício tem um valor de beleza estético único, que é definido por um número inteiro qualquer. Este valor de beleza aos edifícios é importante para avaliação imobiliária, pois pessoas famosas estão investindo na cidade. Inclusive, a garota propaganda de alguns prédios é a (secretamente engenheira) Sharon Stone. Sharon é ávida também em avaliar a estética destes prédios.

É difícil determinar o exato valor estético de cada edifício, então Sharon gostaria de encontrar *um belo* edifício, isto é, que tenha um valor estético mais alto do que todos os seus vizinhos.

Sharon gasta t_i segundos para inspecionar o edifício i e as pontes para todos os seus vizinhos. Após inspecionar os j vizinhos do prédio i, Sharon saberá se para cada vizinho j do prédios i, qual dos dois tem o maior valor estético. Afinal, de beleza Sharon entende.

Problema

Qual é o tempo total mínimo em segundos, de modo a garantir que Sharon encontre um belo edifício? Note que o valor efetivo de estética dos edifícios não é necessário. Sharon pode decidir qual prédio inspecionar em seguida, com base nos resultados de inspeções anteriores. Ignore o tempo necessário para viajar entre edifícios.

Entrada

- Linha 1 contém um inteiro N $(2 \leq N \leq 16)$
- Linha 2 informa os valores do tempo de inspeção: contém N inteiros $t_1, ..., t_N$ $(1 \leq t_i \leq 10^8)$
- Linha 3 informa as pontes de ligação entre os edifícios, isto é, a vizinhança: contém $N - 1$ inteiros $x_2, ..., x_N$ $(1 \leq x_i < i)$.

Saída

Imprima uma linha com um inteiro, com o tempo total mínimo, que garanta que um belo edifício possa ser encontrado.

Exemplo de Entrada	**Exemplo de Saída**
3 10 40 20 1 2	30
5 2 4 1 2 1 1 1 2 2	5
3 90432658 64136824 24255460 1 2	64136824
5 90432658 666 24255460 88840 4255460 1 2 3 4	89506

Explicando os exemplos

- Para o primeiro exemplo, a inspeção dos edifícios 1 e 3 garante que um belo edifício será encontrado.
- Para o segundo exemplo, uma estratégia ótima é sempre inspecionar primeiro o edifício 3.

6 Problema F: Formando as Retas

`Arquivo: formando.[c|cpp|java|py]` `Tempo limite: 2 s`

Você talvez não saiba, mas Sharon Stone é fissurada por retângulos! Recentemente ela elaborou um novo desafio envolvendo retângulos. Considere que Sharon tem N retângulos, sendo que o i-ésimo retângulo tem seus vértices das extremidades opostas (diagonais) em (a_i, b_i) e (c_i, d_i). Agora pense na união de todos esses N retângulos. Essa união gera um desenho que é denotado por U.

O desafio de Sharon, entretanto, começa agora: considere um conjunto de retas diagonais definidas por $y = s - x$ que podem ou não interseccionar U em segmentos de linha disjuntos (possivelmente degenerados). A soma dos comprimentos desses segmentos de linha que interseccionam U é denotada por $f(s)$, podendo ser zero caso a interseção seja vazia.

No desafio de Sharon, ela fala dois valores, L e R que representam o intervalo de valores inteiros que podem ser assumidos por s na equação da reta diagonal. Logo, devem ser consideradas R–$L + 1$ retas que possivelmente interseccionam U. Dado os inteiros L e R, define-se $S = \sum_{s=L}^{R} f(s)$, ou seja, a soma do comprimento de todas as interseções das linhas diagonais geradas com U. Sharon quer saber de você qual é o número (inteiro) de diagonais V que são interseccionadas (note que $S = V\sqrt{2}$).

Problema

Calcule o valor de V.

Entrada

A linha 1 contém os inteiros $N, L, R (1 \leq N \leq 10^3, -2 \times 10^8 \leq L < R \leq 2 \times 10^8)$. A i-ésima linha contém os inteiros $a_i, b_i, c_i, d_i (-10^8 \leq a_i < c_i \leq 10^8, -10^8 \leq b_i < d_i \leq 10^8)$.

Saída

Imprima uma linha, com um valor inteiro V.

Exemplo de Entrada	Exemplo de Saída
3 -1 3 -2 -1 0 2 -1 0 1 1 1 -2 2 -1	7
3 -1 3 -2 -1 0 2 -1 0 1 1 0 -2 2 0	10
1 2 3 -1 1 0 2	0

Explicando os exemplos

A figura abaixo ilustra o primeiro exemplo quando $s = 0$. $f(0) = 3\sqrt{2}$ é a soma dos comprimentos dos dois segmentos de linha em negrito que intersectam U, isto é, os retângulos de Sharon. Note que $S = 7\sqrt{2} \approx 9.899$.

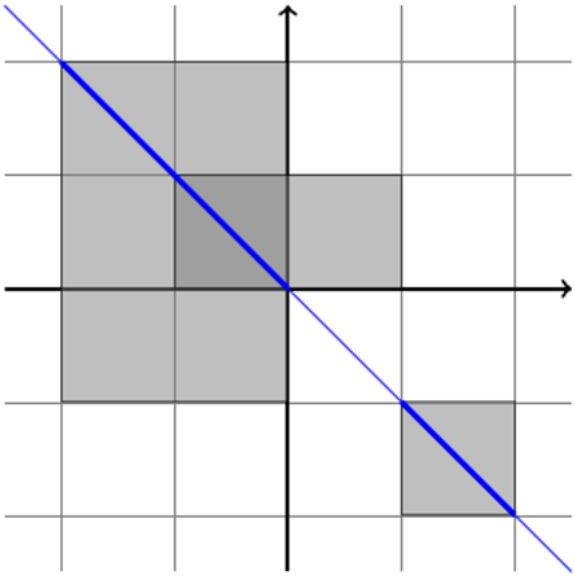

Figura 1

7 Problema G: Geometria

`Arquivo: geometria.[c|cpp|java|py]` `Tempo limite: 2 s`

Daniela é uma aluna do ensino médio que recentemente se interessou por linguagens de programação após participar de uma olimpíada de informática em Aquidauana. Em uma das recentes aulas de matemática, sua professora apresentou um pequeno desafio: contar o número de triângulos que podem ser formados a partir de um conjunto de palitos. Daniela enxergou nesse desafio uma oportunidade de praticar suas habilidades com programação. Sua equipe pode ajudá-la a implementar a solução para essa tarefa?

Problema

Daniela tem 5 palitos de diferentes comprimentos $l_i, 1 \leq i \leq 5$, e ela pode escolher diferentes conjuntos de 3 palitos dentre os 5 disponíveis para formar triângulos. Quantos triângulos diferentes podem ser criados? Cada triângulo deve ter área positiva e os palitos não podem ser cortados ou dobrados. Um mesmo conjunto de 3 palitos pode fazer no máximo um triângulo, ou seja, espelhar um triângulo não configura a criação de outro triângulo.

Entrada

A entrada consiste em uma linha contendo 5 números inteiros, que representam o comprimento de cada um dos 5 palitos $l_i, 1 \leq i \leq 5$, sendo $1 \leq l_i \leq 1000$.

Saída

Imprima uma linha com um inteiro: o número de diferentes triângulos que podem ser formados.

Exemplo de Entrada	**Exemplo de Saída**
`1 2 3 4 5`	`3`
`1 2 4 8 16`	`0`
`1 999 2 1000 3`	`2`

Explicando os exemplos

No primeiro caso de testes é possível formar três triângulos escolhendo os palitos 2, 3, 4 ou 2, 4, 5 ou 3, 4, 5.

8 Problema I - Inventando um Drama

`Arquivo: inventando.[c|cpp|java|py]` `Tempo limite: 2 s`

Problema

AdilsonJJ tem um tabela com H linhas e N colunas. As linhas estão numeradas de 1 a H enquanto as colunas estão numeradas de 1 a N. A célula na r-ésima linha e na c-ésima coluna está representada na forma (r, c). As células são coloridas em branco e preto. Uma coloração é chamada de *pirâmide* se:

- Exatamente N células são pretas;
- (1,1) é preta;
- Se (r,a) e (r,b) são pretas, então (r,k) é preta para $a < k < b$;
- Se (r,c) é preta, então (r-1, c), se ela existir, é preta;
- Se (r,c) é preta e não há $k < c$ tal que (r, k) é preta, então $(r + 1$, c), se ela existir, é branca.

Duas *pirâmides* são diferentes se houver uma célula que é branca em uma *pirâmide* e preta em outra. Compute o número de diferentes *pirâmides* módulo $10^9 + 7$.

Entrada

Cada caso de teste consiste em apenas uma linha contendo dois inteiros H e N ($1 \leq H, N \leq 10^5$).

Saída

Imprima uma linha com um inteiro, o número de diferentes *pirâmides* módulo $10^9 + 7$.

Exemplos

Exemplo de Entrada	Exemplo de Saída
2 6	7
3 20	487

Explicando os exemplos

Para o primeiro exemplo, as sete *pirâmides* são:

```
###### ####.. ####.. #####. #####. #####. #####.
...... .##... ..##.. .#.... ..#... ...#.. ....#.
```

9 Problema J: Jantar

`Arquivo: jantar.[c|cpp|java|py]` `Tempo limite: 2 s`

Você conhece 26 receitas, cada uma das receitas é representada por uma letra minúscula de a à z. Você está preparando um banquete que consiste de N pratos organizados em uma mesa circular. Como você é muito preguiçoso, cada prato é independente e uniformemente distribuído entre uma de suas 26 receitas. O banquete pode ser representado como uma *string* S de tamanho N onde S_i é a receita do prato i ($1 \leq i \leq N$). Na mesa, o prato j está posicionado em sentido horário ao prato $j-1$ para $2 \leq j \leq N$ e o prato 1 está posicionado no sentido horário ao prato N.
Um exemplo de sequência de receitas pode ser uma seção consecutiva de pratos em ambos sentidos (horário e anti-horário).
Você precisa de ajuda e conta com um milagre para descobrir quantas sequências diferentes existem, ou seja, quantas possibilidades de receitas você pode fazer. Dois exemplos de receitas são iguais se eles tiverem o mesmo comprimento e a mesma receita em cada posição.

Entrada

A entrada consiste em apenas duas linhas:

- Linha 1 contém um inteiro N ($2 \leq N \leq 50000$).
- Linha 2 contém uma *string* S de comprimento N composta de letras minúsculas. É garantido que S representa um banquete criado pelo processo descrito.

Saída

Imprima uma linha com um único inteiro indicando o número total de diferentes receitas possíveis.

Exemplo de Entrada	Exemplo de Saída
3 aba	8
6 ondrej	66
8 yakisoba	116

Explicando os exemplos

Para o primeiro exemplo, as oito diferentes receitas são: **a, b, aa, ab, ba, aba, aab, baa**.

Para o segundo exemplo, **rejo**, **drejon** são sentido horário e **nojer**, **dnojer** são sequências em sentido anti-horário.

10 Problema L: Legítimo ou Não

`Arquivo: legitimo.[c|cpp|java|py]` `Tempo limite: 2 s`

A empresa Brendel é responsável por um serviço de comunicação segura entre clientes. Atualmente, o app da empresa tem apenas 20000 usuários cadastrados e já sofre com um enorme problema. Alguns usuários mal intencionados estão criando contas *fakes* para fazer *spam* e atacar outros usuários.

O administrador do banco de dados da empresa ficou preocupado, foi verificar as contas cadastradas no banco e percebeu que os usuários estavam livres para cadastrar qualquer CPF, sem qualquer tipo de verificação. Revoltado com a situação ele pediu para vocês, calouros da empresa, que solucionassem o problema! Não deve ser permitido o cadastro de novo clientes com CPF inválido ou clonado!

É necessário verificar se o CPF digitado já existe no sistema, caso já exista não deve ser cadastrado novamente, além disso, ainda é preciso verificar se o CPF é legítimo.

Através de uma profunda busca na internet, foi possível encontrar um modo para validar um número de CPF. Dado um CPF (sem símbolos de verificação), é possível calcular seus dígitos verificadores (aqueles dois após o hífen) da seguinte maneira:

Tome como exemplo o CPF sem os dígitos verificadores 111444777.

- Para calcular o primeiro digito verificador distribua os nove primeiros dígitos em um quadro, da esquerda para a direita, atribuindo um peso para cada número.

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

- Multiplique coluna por coluna e some os valores, o resultado será $10+9+8+28+24+20+28+21+14=162$.
- Considere r, o resto da divisão do resultado por 11, no nosso exemplo, r=162%11 = 8.
- O primeiro dígito é calculado da seguinte maneira:
  - se $(r < 2)$, o dígito será 0.
  - senão, o dígito será $(11 - r)$. (Neste exemplo, $r = 8$, logo o primeiro dígito será $11 - 8 = \mathbf{3}$)
- Para calcular o segundo dígito realizamos o mesmo processo, porém, adicionamos o dígito encontrado no final da lista, alterando os pesos. Ou seja:

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

- **ATENÇÃO:** O segundo dígito é calculado da mesma maneira que o primeiro. (Neste exemplo, o segundo dígito é **5**, logo, os dígitos verificadores são **35**)

Dado um CPF, verifique se este é válido ou se já está cadastrado.

Entrada

A primeira linha da entrada tem um inteiro **N** ($1 \leq N \leq 1000$) que representa a quantidade de cadastros. As próximas **N** linhas contém dois inteiros, **C** de 9 dígitos e **D** ($0 \leq C \leq 99$), representando um CPF (sem os dígitos) e seu possível dígito verificador, respectivamente.

Saída

Para cada um dos **N** cadastros imprima "Clonado"caso o CPF já exista, "Invalido"caso os dígitos estejam incorretos, e "Sucesso"caso contrário.

Exemplo de Entrada	**Exemplo de Saída**
4 111444777 35 948102232 10 948102232 30 111444777 35	Sucesso Invalido Sucesso Clonado

11 Problema M: *Marelinha* para Calouros

`Arquivo: marelinha.[c|cpp|java|py]` `Tempo limite: 2 s`

Todos já brincaram um dia de amarelinha. Aqui é um pouco diferente do usual, pois o jogo começará do fim, chamado de *céu*, e termina neste mesmo lugar, ver figura 2.

O básico é descobrir quantos saltos são necessários para o término do jogo, sendo que a pedrinha se encontre em uma célula (ou casa: 1 a 10) qualquer. Basicamente, o jogo consiste em sair do "céu", saltando de casa em casa, até onde se encontra a pedrinha, avançar a pedrinha de **uma casa** em direção ao "céu", e voltar para o "céu". Repetir este procedimento até a pedrinha alcançar o "céu". A marcação das casas do jogo de *marelinha* é a mesma do jogo tradicional. Ver figura 2:

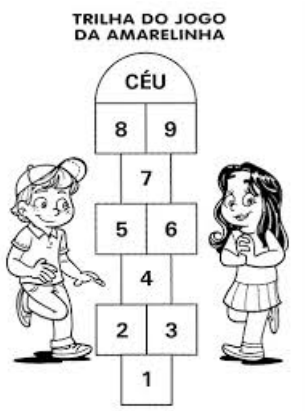

Figura 2: Clássico da Amarelinha

ou:

```
   [2]    [5]    [8]
[1]    [4]    [7]    [10]--(Céu)
   [3]    [6]    [9]
```

Quanto ao "céu" é o ponto de partida para esta "*amarelinha ao contrário*". O jogo se dá pelo avanço da pedrinha a partir de uma casa qualquer de 1 a 10, de acordo com a figura acima. Neste jogo modificado, o jogador começa e termina no "*céu*".

Se a pedrinha estiver na casa 10 basta o jogador ir até a casa 10, com um salto simples (só vale este tipo de avanço), pegar a pedrinha e arremessar para o céu e voltar! Neste caso, foram necessários 2 saltos e o jogo termina.

Caso a pedrinha esteja nas casas 8 ou 9 (o conceito é o mesmo para as casas 5 ou 6 e 2 ou 3), o jogador vai até esta casa, pega a pedrinha e avança para a casa seguinte, que no caso é a casa 10. Ou seja, a pedrinha sempre é arremessada para a casa subsequente, em direção ao "*céu*". Feito isso, o jogador deve sempre retornar ao céu e recomeçar a busca.

Assim, se a pedrinha está nas casas 8 ou 9, o jogador gastou 4 saltos para ir e voltar a partir do céu, e deixou a pedrinha na casa 10. Para o término do jogo, o jogador gastou estes 4 saltos para ir e para voltar até a casa 8 ou 9, mais os 2 saltos exigidos com a pedrinha na casa 10. Essas regras do jogo valem até a casa 1, a mais distante. Este processo se repete iterativamente até a casa 10; mas observe o tabuleiro, as células 2 e 3, 5 e 6, e 8 e 9, se encontram na mesma coluna ou equidistantes do "*céu*".

Problema

Sua tarefa é encontrar o número total de saltos dada a posição inicial X da pedrinha no tabuleiro.

Entrada

A entrada contém vários casos de teste. Um caso por linha. Em cada caso de testes é dado um valor X ($1 \leq X \leq 10$), que representa a posição inicial da pedrinha no tabuleiro. Os casos de testes se encerram com `-1`. Ou seja, `-1`, voce não lê mais nada e nem calcula.

Saída

Para cada caso de teste imprima um inteiro por linha, o qual corresponde a quantidade de saltos para finalizar o jogo.

Exemplo

Exemplo de Entrada	Exemplo de Saída
10 9 6 2 -1	2 6 20 42

12 Problema N: Navios de Transporte

`Arquivo: navios.[c|cpp|java|py]` `Tempo limite: 5 s`

O ano é 2077. As calotas polares derreteram, e a civilização humana passou a morar em plataformas-cidade dispostas no Mar Eterno, o oceano que agora cobre mais de 95% da superfície da Terra. Você trabalha para a Viação Peixinho, uma companhia que realiza o transporte de mercadorias de entre plataformas.

Sua principal função na Viação é traçar rotas que possibilitem que os marinheiros cheguem na sua plataforma destino no menor número possível de dias, respeitando as regras do sindicato dos marinheiros. O sindicato só permite que os marinheiros dirijam por no máximo **10** horas diárias, em vias pré-definidas (devido ao perigo dos piratas) entre plataformas. A Viação quer encontrar uma rota que vá de uma plataforma inicial até uma plataforma destino de maneira que o marinheiro-piloto possa passar a noite em uma plataforma de confiança da Viação (lembre-se, o marinheiro pode dirigir por no máximo **10** horas entre duas plataformas de confiança, ou a Viação será multada e você será demitido!).

Entrada

A entrada é composta por diversos casos de teste. Cada caso inicia com uma linha contendo um inteiro N ($2 \leq N \leq 10000$), correspondendo ao número de plataformas no mapa marítimo. As plataformas são numeradas de 1 até N, onde 1 é a plataforma incial e N é a plataforma de destino. A próxima linha contém um inteiro H seguido pelos números c_1, c_2, ..., c_h, indicando o número das plataformas de confiança da Viação. Você pode assumir que $0 \leq h \leq min(N, 100)$. A terceira linha de cada caso contém um inteiro M ($1 \leq M \leq 105$) correspondendo ao número de vias existentes entre plataformas. Cada uma das M linhas seguintes descrevem uma via. Cada via é descrita por uma linha contendo 3 inteiros a, b e t, ($1 \leq a$, $b \leq N$ e $1 \leq t \leq 600$), onde a e b são as duas plataformas conectadas pela via, e t é o tempo em minutos necessário para navegação.

A entrada é terminada com um caso onde $N = 0$, que não deverá ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo o menor número de paradas que a empresa precisa realizar ao percorrer a rota ótima da cidade 1 até N. Se não for possível encontrar uma rota que satisfaça as regras do sindicato, imprima -1.

Exemplo de Entrada	**Exemplo de Saída**
6 3 2 5 3 8 1 2 400 3 2 80 3 4 301 4 5 290 5 6 139 1 3 375 2 5 462 4 6 300 3 0 2 1 2 371 2 3 230 0	2 -1

13 Problema S: Somas e Gauss

`Arquivo: gauss.[c|cpp|java|py]` `Tempo limite: 2 s`

Johann Carl Friedrich Gauss (1777–1855) foi um dos mais importantes matemáticos da Alemanha. Para aqueles que se lembram do Marco alemão, antes do Euro, a foto de Gauss estava estampada na nota de 10 DM (*Deutsche Mark*).

Gauss quando estava na escola primária, seu professor J.G. Buttner tentou ocupar os seus pupilos fazendo com com que eles somassem os inteiros de 1 até 100. O jovem Gauss surpreendeu a todos produzindo uma resposta correta em poucos segundos: 5050. Pode voce escrever um programa de computador tal que faça esta soma realmente de maneira muito rápida?

O Problema

Dado dois números inteiros n e m, voce poderia computar a soma de todos inteiros de n até m.

Entrada

A primeira linha contém o número de casos no arquivo de entrada. O número de casos, k, é dado por: $1 < k < 1000$. Em seguida, as k linhas seguintes são os k casos. Cada caso consiste de um par de números n e m ($-10^9 \leq n \leq m \leq 10^9$).

Saída

A saída de cada caso começa com uma linha contendo a palavra "`Caso #i:`", onde i é o número de cada caso começando no 1. Então imprima a soma de todos inteiros de n até m.

Exemplo de Entrada	**Exemplo de Saída**

`3`
`1 100`
`-11 10`
`-89173 938749341`
	`Caso #1: 5050`
	`Caso #2: -11`
	`Caso #3: 440625159107385260`

Dicas

- Em geral o prof Claudius Virus Linux, usa esta série/sequência para ilustrar o uso de laços de repetições aos calouros;
- Sempre há um mais *antenado*, que se lembra desta fórmula de seus tempos de cursinho!
- A dificuldade é voce se lembrar e deduzir o que Gauss fez, mas, cuidado: há números grandes!