Caderno de Provas – *Princesa Izabel*

1^a Seletiva Interna – 2017/1

UDESC–Joinville e IFMS–Aquidauana

Servidor BOCA (Arena Joinville):
http://200.19.107.207/boca/

Organização e Realização:

Claudio Cesar de Sá (coordenação geral), Gabriel Hermann Negri (coordenação técnica), Lucas Hermann Negri (revisão), Luciana Rita Guedes, Roberto Silvio Ubertino Rosso Jr., Rogério Eduardo da Silva (revisão)

Patrocinador 2017: Conta Azul

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;
- 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**!

Patrocinador e Agradecimentos

- Conta Azul – Patrocinador oficial do ano de 2017;
- DCC/UDESC;
- Aos bolsistas deste ano pelo empenho;
- Alguns, muitos outros anônimos.

Princesa Izabel

1ª Seletiva Interna da UDESC e IFMS–Aquidauana – 2017

13 de maio de 2017

Contents

Atenção quanto aos nomes e números dos problemas!!!

1 Problema A: Agora o Menor

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

Sempre que nosso empresário de fronteira, Roger Compulsivo Buyer, segue com suas *listinhas* para Iguaçu Paradise, ele gosta de ter a quantidade exata de dinheiro para cada uma de suas compras. Isto faz com que ele nem precise contar o dinheiro quando paga e nem receber nenhum troco de volta. Assim, ele fez vários *pacotinhos* de dinheiro para cada compra.

Claro que ao longo do dia calorento, o entra-e-sai das lojas, tudo acaba se confundindo, e misturando o dinheiro. Para amenizar o caos, o nosso empresário de fronteira gostaria de saber qual a menor quantidade exata de dinheiro (*cents* – sim lá, o dinheiro é o da Gringolândia) que ele não pode pagar, combinando as moedas de um dos seus pacotinhos.

Problema

Dada uma lista de inteiros positivos, determine qual o menor inteiro que não pode ser representado como uma soma de um subconjunto da lista.

Entrada

A primeira linha de entrada contém um único inteiro positivo, n ($1 < n \leq 100$), indicando o número de pacotes de moedas a serem avaliados. Cada um dos n conjuntos da entrada seguem abaixo. A primeira linha de cada conjunto de entrada contém apenas um inteiro, c ($1 \leq c < 31$), representando o número de moedas do pacote para cada conjunto de entrada. A linha seguinte contém exatamente c inteiros positivos, cada valor um separado por um único espaço, representando o valor de uma moeda do pacote em centavos (*cents*). A soma destes inteiros c é garantida não exceder 10^9. Observe que os valores do pacote não são necessariamente distintos, ou seja, podem existir várias moedas com o mesmo valor num mesmo pacote.

Saída

Para cada conjunto de pacotes, imprima a saída "`Conjunto #i:`" onde `i` é o número do conjunto da entrada de dados, começando com `1`. Em seguida, imprima um único inteiro positivo correspondendo ao menor valor que **não** pode ser representado como uma soma dos valores de um subconjunto do pacote de moedas fornecido na entrada.

Observe que um valor do pacote pode ser usado no máximo uma vez em um subconjunto, a menos que existam vários valores iguais neste pacote (se houver `m` ocorrências de um valor no pacote, então até `m` ocorrências desse valor podem ser usadas em um subconjunto). Deixe uma linha em branco após uma saída para cada caso de teste. Siga o formato ilustrado nos exemplos que seguem.

Exemplo de Entrada	**Exemplo de Saída**
4 6 12 8 1 2 4 100 3 1 2 3 6 3 1 3 2 3 3 4 3 1 1 3	Conjunto #1: 28 Conjunto #2: 7 Conjunto #3: 16 Conjunto #4: 9

2 Problema B: Bem-tu-ring

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

O Bem-tu-ring é uma nova espécie de pássaro recém descoberta conhecida por seus padrões de comportamentos bem regulares. Cada membro de um bando é identificável de forma única através do número de manchas coloridas nas suas asas, ou seja, não existem dois pássaros com a mesma quantidade de manchas.

Quando o bando migra para uma nova área, ele encontra uma árvore alta e constrói uma estrutura em formato de gancho na parte inferior do galho mais alto. E então, cada pássaro irá construir um ninho pra si próprio. Cada ninho irá ficar pendurado em um gancho e terá até dois outros ganchos na sua parte inferior, um na sua direita e outro na esquerda.

Eis as regras para a construção dos ninhos:

1. Pássaros sempre constroem ninhos na ordem em que eles chegam
2. Cada ninho ficará pendurado em um gancho pertecente a outro ninho ou no gancho inicial, preso à árvore. Cada gancho pode conter, no máximo, um ninho.
3. Cada pássaro tentará pendurar seu ninho diretamente na estrutura base (primeiro gancho, preso diretamente à árvore). Caso este gancho esteja ocupado, ele usará a regra (4)
4. Se já existir um ninho pendurado em um gancho que o pássaro está tentando usar, ele irá comparar suas manchas com as do dono do ninho pendurado naquele gancho. Se ele tiver menos manchas, ele irá tentar construir seu ninho no gancho da esquerda, caso contrário ele tentará usar o gancho da direita. Este procedimento irá se repetir até que ele encontre um gancho vazio.

O Problema

O bem-tu-ring é uma espécie em risco de extinção e pesquisadores recentemente descobriram o porquê. Quando bandos muito grandes de pássaros constroem seus ninhos, a estrutura resultante tem a tendência a ficar instável. Inevitavelmente um ninho cai da estrutura, levando consigo todos os demais ninhos que estavam pendurados nele. (Apesar de que 'cair' geralmente não é um problema para pássaros, voar acaba se tornando um problema difícil quando meia dúzia de ninhos e seus ocupantes caem sobre sua cabeça no meio da noite).

Define-se como um *cabide* de ninhos a sequência de ninhos conectados por ganchos, começando pelo ninho e movendo-se um ninho abaixo a cada passo (nunca lateralmente) até que se alcance um ninho no nível mais inferior. o *comprimento de um cabide* é apenas a quantidade de ninhos em um cabide.

A *altura de um ninho* é definida pelo comprimento do seu cabide mais longo. O *fator de instabilidade* de um ninho é a diferença absoluta da altura do ninho pendurado à sua esquerda com a altura do ninho pendurado à sua direita. (se não existirem ninhos em um gancho, a altura para aquele 'ninho nulo' é tratada como zero).

Seu programa, dada a descrição de um bando de pássaros, deverá exibir o número de manchas do pássaro que vive no ninho com maior fator de instabilidade. Se existirem mais de um pássaro vivendo nas mesmas condições críticas, exiba aquela que chegou mais recentemente à estrutura.

Entrada

A primeira linha da entrada contém um número positivo, n, indicando a quantidade de bandos de pássaros. Cada caso de entrada usa duas linhas. A primeira linha contém um inteiro $f(1 \leq f \leq 5000)$, indicando o número de pássaros naquele bando. A próxima linha consiste de f números inteiros positivos, cada um representando a quantidade de manchas de um pássaro, na ordem em que eles chegam à estrutura. Você pode assumir que não existem dois pássaros com mesma quantidade de manchas. Cada pássaro tem no mínimo 1 mancha e no máximo 5000.

Saída

Para cada caso de teste, exiba uma única linha na forma "`Bando #k: p`" onde k é o número do bando de pássaros, e p é a quantidade de manchas do pássaro vivendo no ninho mais perigoso. Deixe uma linha em branco após cada resultado do caso de teste. Siga o formato apresentado no exemplo a seguir.

Exemplo de entrada	**Exemplo de saída**
3 10 10 14 12 4 8 19 18 17 9 16 5 4 5 3 2 1 15 50 67 19 42 18 66 168 1 52 57 37 64 78 22 130	Bando #1: 14 Bando #2: 4 Bando #3: 66

3 Problema C: Calendário Dobrado

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

O professor Rogério possui um calendário (de 365 páginas) em sua mesa. Todas as manhãs, ele arranca uma página e, enquanto lê as anotações na página seguinte, ele dobra (por força do hábito) a folha que foi arrancada. O professor Rogério percebeu que ele sempre dobra o papel retangular pelo lado maior. Isto é, se um lado tem 80 cm e outro tem 60 cm, ele dobra o papel pelo lado maior, resultando em uma folha de 40 cm por 60 cm. Se ele dobrar novamente, a folha passa a ter 40 cm por 30 cm. Para testar seus alunos de IC, ele propôs um desafio. Usar a poderosa ferramenta da computação para predizer o tamanho final de um papel a ser dobrado por ele. Será que você consegue ser aprovado para uma bolsa com o professor Rogério?

O Problema

Dado um pedaço de papel retangular e o número de dobraduras efetuadas, determine o tamanho final do papel. Quando o lado a ser dobrado tiver tamanho ímpar, ignora-se o resto da divisão por dois, ou seja, um lado de tamanho 7 se torna de tamanho 3 quando dobrado.

Entrada

A primeira linha de entrada contém um inteiro positivo, n, indicando o número de conjuntos de dados de teste. Os conjuntos seguem nas próximas n linhas, um conjunto por linha. Cada conjunto contém três inteiros positivos menores ou iguas a 10000. Os dois primeiros indicam as dimensões do retângulo e o terceiro é o número de dobraduras a serem feitas pelo professor Rogério.

Saída

No início de cada caso de teste, imprima: `Conjunto: a b c`, em que `a b c` são os valores fornecidos para cada caso. Então, na próxima linha, imprima as dimensões finais para o retângulo (imprima primeiro o valor maior). Deixe uma linha em branco após a saída para cada caso de teste. Siga o formato ilustrado no exemplo a seguir.

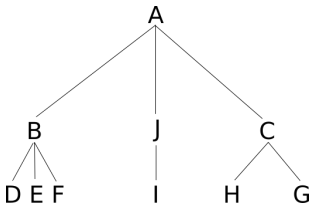
Exemplo de entrada	**Exemplo de saída**

`3`
`60 51 4`
`3 2 50`
`3 2 1`
	`Conjunto: 60 51 4`
	`15 12`
	
	`Conjunto: 3 2 50`
	`0 0`
	
	`Conjunto: 3 2 1`
	`2 1`

4 Problema D: Deslocamento

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

O professor Gilmário gosta de fazer caminhadas, e como ele acabou de ensinar sobre árvores na disciplina AED - Algoritmos e Estruturas de Dados, ele acaba de achar uma nova trilha que tem exatamente o formato de uma árvore. Para efeito de simplicidade, o professor Gilmário sempre usa A como referência para a raiz da árvore. Sendo assim, um exemplo de trilha pode ser:



Professor Gilmário decidiu seguir uma abordagem de **Busca em Amplitude (Breadth First Search - BFS)** para se deslocar pela trilha e, para efeito de simplicidade, Gilmário visita cada filho de um nó em ordem alfabética. A travessia BFS do exemplo acima fica:

```
A B C J D E F G H I
```

Porém Gilmário rapidamente percebeu que se deslocar de um nó para seu irmão, ele precisa antes voltar até o pai daquele nó novamente (porque não há uma trilha direta entre um filho e seu irmão). Assim, para a árvore-exemplo acima, o professor Gilmário irá na prática se deslocar da seguinte forma:

```
A--B--A--C--A--J--A--B--D--B--E--B--F--B--A--C--G--C--H--C--A--J--I--J--A
```

Note que Gilmário retorna sempre até a raiz no final de sua caminhada. Pois bem, isto representa muito mais deslocamento que ele tinha antecipado em uma BFS mas ele não se importa afinal de contas sua esposa o convenceu sobre os benefícios de se fazer exercícios e comer de forma saudável!

O Problema

Dada a descrição de uma trilha (árvore), determine a distância total (deslocamento) do algoritmo de deslocamento através da busca em amplitude (BFS) na trilha, incluindo a distância de retorno em cada passo.

Entrada

A primeira linha da entrada contém um número positivo, t, indicando o número de trilhas (árvores) a ser checado. Cada trilha é definida por uma série de caminhos que conectam junções e pontos de interesse (cada qual é representado por uma única letra maiúscula). A linha seguinte da entrada conterá um número inteiro, $n (1 \leq n \leq 26)$, fornece o número de nós da árvore (junções e pontos de interesse); assuma que esses nós serão representadas pelas n primeiras letras maiúsculas. Em cada uma das próximas $n-1$ linhas da entrada conterão duas letras maiúsculas, indicando que existe uma trilha entre essas duas localidades (o primeiro nó sendo o pai do segundo), e um número positivo representando o comprimento desta trilha (a distância entre duas localidades); os três valores nessa entrada são separados por um único espaço em branco.

Gilmário sempre inicia na entrada da trilha, rotulado pela letra "A", a qual é garantida pertencer à entrada.

Saída

No início de cada trilha, exiba a mensagem "`Trilha #i:`" em uma linha separada, onde i representa o número da trilha na entrada (iniciado por 1). Na próxima linha da saída, exiba a mensagem "`Deslocamento total: b`" onde b é a distância total que o professor Gilmário se deslocou através de sua caminhada usando uma abordagem BFS (isto deve incluir as distâncias de retorno). Assuma que o total da distância irá caber em um número inteiro.

Notem que, em cada localidade, Gilmário sempre visitará cada uma das próximas localidades em ordem alfabética. Deixe uma linha em branco após exibir o resultado para cada trilha. Siga o formato apresentado no exemplo a seguir.

Exemplo de entrada	**Exemplo de saída**
3 3 A B 5 A C 3 8 A B 1 A C 7 B D 5 B E 3 B F 2 C G 4 C H 2 3 A B 1000 B C 2000	Trilha #1: Deslocamento total: 16 Trilha #2: Deslocamento total: 64 Trilha #3: Deslocamento total: 6000

5 Problema E: Espaços – A Fronteira Final

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

O capitão Pickláudio, o comandante chefe da nave estelar *Aventura*, é também um famoso xeno-arqueologista. Nas suas raras férias, ele geralmente pode ser encontrado em algum obscuro planeta escavando artefatos de civilizações antigas. Na sua mais recente expedição, ele encontrou uma série de inscrições em uma linguagem antiga de Zigzaguiano. Ele pede a sua ajuda para decifrá-las.

Uma característica estranha da escrita *zigzaguiana* é que ela não usa espaços entre as palavras. Consequentemente, decifrá-la é geralmente uma tarefa bastante difícil, devido ao fato de que não se consegue ter certeza sobre quando uma palavra termina e a próxima começa. Isto faz com que dividir uma inscrição em suas palavras componentes se torne um problema difícil, pois pode haver uma grande quantidade de possíveis divisões.

Felizmente, o sempre prestativo comandante Theta conseguiu reativar o Codex zigzaguiano, um artefato que, dada uma inscrição, associa um valor numérico para um conjunto de palavras em zigzaguiano. Após estudarem estes pares palavra–valor, o capitão Pickláudio supõe que o valor de uma divisão seja a soma dos valores das suas palavras componentes. E mais, inscrições são escritas de tal forma que a divisão correta é aquela com o valor mais alto. Perceba que a mesma palavra pode ocorrer múltiplas vezes em uma inscrição, e que cada ocorrência contribui para o valor. Então se a palavra `ra` tem um valor de 5, e aparece 3 vezes, isto conta como $5 \times 3 = 15$.

O Problema

O capitão quer um programa capaz de encontrar o maior valor da divisão de cada inscrição. Ajudar o capitão representa um passo mais perto para a escolha da sua equipe como representante da UDESC na Maratona de Programação e toda a glória que isso significa, então vamos ajudá-lo?

Entrada

A primeira linha da entrada contém um número positivo, n indicando o número de inscrições a dividir. Isto é seguido por n descrições de inscrições e o Codex associado.

A primeira linha de cada descrição é uma única string S com comprimento entre 1 e 1000. É garantido que esta string contenha somente letras minúsculas sem espaços.

A próxima linha da entrada contém um número inteiro, c $(1 \leq c \leq 1000)$, que indica o número de pares palavra–valor geradas pelo Codex para esta inscrição. A seguir a entrada tem c linhas, cada uma contém uma palavra w (contendo apenas letras minúsculas de comprimento entre 1 e 1000) e seu respectivo valor v $(1 \leq v \leq 1000)$, separados por um único espaço.

Saída

Para cada uma das descrições na entrada, imprima "`Inscricao #k:`" (sem til e cedilha) onde k é o número da inscrição. A próxima linha da entrada deve conter o maior valor da divisão da inscrição. Na próxima linha, você deve imprimir essa divisão através da inserção dos espaços em branco na inscrição nos lugares apropriados. Você pode assumir que pelo menos uma divisão válida sempre exista.

No caso de existirem múltiplas divisões possíveis com valor máximo, escolha aquela onde o primeiro espaço *diferenciador* for posicionado primeiro na string. Um espaço diferenciador

é aquele que não aparece em ambas as divisões. Veja o exemplo de saída para maiores esclarecimentos.

Deixe uma linha em branco após cada descrição. Siga o formato apresentado no exemplo a seguir.

Exemplo de entrada	**Exemplo de saída**
2 olamundo 7 a 5 ola 9 la 2 mu 3 mun 6 mund 4 mundo 8 atatata 5 a 7 at 3 ta 6 t 4 att 5	Inscricao #1: 17 ola mundo Inscricao #2: 40 a t a t a t a

6 Problema F: Fiscal de Gabarito

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

O renomado professor de teoria dos números Caesar Claudius (CC), após chegar à conclusão que seu tempo não deveria ser gasto corrigindo provas discursivas, decidiu que todas as suas provas teriam 9 problemas, e que a resposta de cada problema seria um inteiro módulo 10, ou seja, um inteiro entre 0 e 9, inclusive. Fascinado por padrões, CC resolveu que as respostas no gabarito de suas provas seguiriam um determinado padrão. Ainda, sendo um professor muito exigente, determinou que a) se o aluno acertasse todas as questões, tiraria nota máxima, ou b)se errasse qualquer número maior ou igual a 1 de questões, tiraria zero (mais tarde, passou a distribuir balões a quem gabaritasse...). O padrão a ser seguido é: a primeira, a quinta e a nona questão teriam respostas independentes. As demais questões, teriam respostas seguindo o seguinte padrão, de acordo com as respostas da primeira, da quinta e da nona questão, sendo A o valor da reposta da primeira questão, B o valor da resposta da segunda questão e C o valor da resposta da terceira questão:

- Resposta da segunda questão: ((A+C)× 2)%10
- Resposta da terceira questão: (B× 3)%10
- Resposta da quarta questão: (A+B)%10
- Resposta da sexta questão: (C× 4)%10
- Resposta da setima questão: (A× 8)%10
- Resposta da oitava questão: (A+B+C)%10

O Problema

Dada uma sequencia de 9 inteiros, correspondente ao gabarito entregue por um aluno, determine se o aluno gabaritou ou nao a prova, de acordo com o padrao descrito, considerando que suas respostas para a primeira, a quinta e a nona questao estao corretas (o professor ja descartou as provas com erros em tais questoes). Note que as provas sao diferentes para cada aluno; logo, havera diferencas nas respostas entre diferentes alunos.

Entrada

A entrada sera composta por um inteiro $n \leq 1000$, indicando o numero de gabaritos a serem verificados. Na sequencia, sao dados n gabaritos, indicando, sequencialmente, a resposta para cada questao.

Saída

Para cada gabarito, imprima `SIM` caso o padrão seja verificado no gabarito ou `NAO` caso o padrão não seja verificado.

Exemplo de entrada	**Exemplo de saída**
4 3 8 8 9 6 4 4 0 1 9 2 2 2 3 8 2 4 1 4 4 1 1 7 2 2 9 8 5 8 3 5 0 6 0 9 3	SIM NAO SIM NAO

7 Problema G: Gerador de Estatística

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

Como acabamos de ver, no problema anterior, o professor Caesar Claudius adotou uma abordagem binária de correção de provas (afinal, não existe resposta meio certa na Maratona de Programação!). Em contrapartida, seu arqui-rival, o renomado professor Negris Lukas, prefere analisar cuidadosamente o desempenho de seus alunos. Para tanto, Lukas utiliza a análise estatística, através do cálculo da média aritmética (não ponderada) e do desvio padrão do conjunto de notas dos alunos (possivelmente sob influência de seu orientador de mestrado).

Como todos sabem, a média aritmética não ponderada de um conjunto de número é dada por:

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i$$

sendo n o número de amostras no conjunto, $\hat{x}$ o valor médio e x_i a i-ésima amostra.

O desvio padrão adotado por Lukas é o populacional, afinal, ele utiliza todas as notas da turma para análise, e é dado por:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2}$$

O Problema

Seria muito simples calcular a média e o desvio padrão caso Lukas soubesse extamente o número de alunos em sua turma. Entretanto, devido à inevitável desistência de alunos ao longo do semestre, o número de notas a serem observadas pode variar (Lukas não quer simplesmente atribuir zero aos faltantes, pois isso destruiria sua rigorosa análise estatística!). Dado um conjunto de notas e sabendo que em todas as provas aplicadas por Rogerius, há pelo menos `5` e no máximo `20` alunos presentes, você consegue calcular a média e o desvio padrão para auxiliar o professor?

Entrada

A primeira linha da entrada contém um número positivo, n, indicando a quantidade de turmas a serem analisadas. Cada uma das n linhas subsequentes contém m ($5 \leq m \leq 20$ notas, representadas por números inteiros, quando a parte decimal for nula (ex: `7`) ou por um número com uma casa decimal (ex: `9.2`), utilizando o padrão internacional (ponto) para identificação da parte fracionária do número.

Saída

Para cada caso de teste, exiba uma única linha na forma "`Turma i: media X.Y desvio A.B`" em que `i` é o número da turma avaliada, começando por `1`, e `X.Y` e `A.B` são a média e o desvio padrão com uma casa decimal (um espaço apenas para separar os termos da saída; não há espaço entre o número da turma e o sinal de dois pontos). Siga o modelo exemplificado a seguir:

Exemplo de entrada	**Exemplo de saída**
3 9 9.5 7 5.5 7 8 8 8 8 8 8 8 8 8 8 8 1 2 3 4 5 6 7 8 9 10	Turma 1: media 7.6 desvio 1.5 Turma 2: media 8.0 desvio 0.0 Turma 3: media 5.5 desvio 2.9

Comandos em C/C++ aleatoriamente escritos aqui (os javaneiros que me desculpem :))

- `double atof(char* str)`: converte uma string "`123.456`" para o número `123.456`, por exemplo;
- `void char * fgets ( char * str, int num, FILE * stream )`: lê uma sequência de caracteres (espaços inclusive) e armazena em um vetor de char (num é o número máximo de caracteres a serem lidos). `FILE* stream`: `stdin` é uma opção. Pode ser interessante usar `scanf("\n")` após o `fgets`, para consumir o pulo de linha que não é lido pelo `fgets`.
- atribuir o caracter `'\0'` em uma determinada posição de uma cadeia de caracteres indica o fim da "`palavra`" escrita na cadeia.

8 Problema H: Harmonia

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

A vida do universitário esforçado não é fácil. Além do currículo do curso, este precisa dar atenção à saúde, ao companheiro (quando há), aos amigos, à família, além dos estudos e atividades extra-curriculares. Muitas vezes o estudante presta atenção somente a uma destas, deixando as outras para trás. A chave está na harmonia: dedicar-se para uma vida completa. Este é o lema da *startup* onde você trabalha 12 horas por dia, sendo pago com *experiência* e *promessas de contatos futuros*. Nesta empresa, você foi encarregado de escrever uma rotina que analise a rotina dos usuários, classificando-a como adequada ou não adequada. Escreva, para ontem, a rotina requisitada pelo seu chefe.

O Problema

Classifique a rotina dos usuários como *adequada* ou *não adequada*. A rotina é fornecida de forma simplificada na entrada, onde o usuário informa o número de horas gastas para o estudo, para diversão e para o trabalho, sendo o restante das horas implicitamente atribuído para descanso (sono). Assim, a rotina é composta por quatro atividades distintas.

Uma rotina é adequada se e somente se respeita todas as regras abaixo (caso contrário é não adequada):

- Nenhuma atividade utiliza mais do que 12 horas;
- É respeitado o mínimo de 6 horas para descanso;
- Somente uma atividade pode ter 0 hora atribuída a ela.

Entrada

A entrada é composta por vários casos de teste, cada um correspondendo a um usuário diferente. A primeira linha contém um inteiro n ($1 \leq n \leq 10000$) que indica o número de casos de teste. Na sequência encontram-se n linhas, cada uma descrevendo um caso de teste. Cada uma destas linhas contém 3 inteiros (a, b e c) separados por um espaço, sendo $0 \leq a, b, c \leq 24$ e $a + b + c \leq 24$. Quanto a a, b e c correspondem ao número de horas diárias alocadas para o estudo, diversão e trabalho, respectivamente. O restante das horas é atribuída para a atividade denominada descanso. Veja os exemplos a seguir.

Saída

Para cada caso de teste, imprima uma linha contendo "`adequada`", se a rotina for adequada, ou "`nao adequada`", caso contrário.

Exemplo de entrada	**Exemplo de saída**
4 6 6 6 0 0 12 7 7 7 8 8 0	adequada nao adequada nao adequada adequada

9 Problema J: Jogo do Calabouço

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

Gabriel e Lucas gostavam de jogar um jogo chamado "**Problemas no Calabouço!**" quando eram crianças. A mecânica do jogo é bastante simples. Você começa com um mapa de calabouços não-numerados em uma caverna, onde alguns são conectados entre si através de passagens secretas. O objetivo é associar um números entre 1 e 5 a cada calabouço de tal maneira que dois calabouços adjacentes não tenham o mesmo número. A última pessoa que consiga, com sucesso, numerar um calabouço sem violar a regra do jogo é declarado o vencedor.

Gabriel adora estatísticas a mantém um meticuloso registro de todas as partidas jogadas. Para cada partida jogada, ele desenha um grafo descrevendo as conexões dos calabouços na caverna e o número que foi associado a eles. Note que alguns dos calabouços podem não ter sido numerados devido ao fato de que um dos jogadores tenha vencido antes de todos os calabouços terem sido numerados.

Recentemente, Gabriel digitou todos esses jogos antigos no seu super poderoso computador pessoal, o "`Prime Analyzer 9000`". Infelizmente, o computador dele é muito especializado e apenas reconheceu os números primos que ele forneceu para os calabouços numerados; os dados de todos os calabouços numerados que não eram primos foram perdidos!

O Problema

Gabriel quer sua ajuda para analizar as partidas para ver qual informação pode ser determinada a partir dos dados que sobraram. Você ainda tem o mapeamento completo do calabouço original e suas conexões, e os números dos calabouços que estão ainda numerados com 2, 3 e 5 (os números primos). Porém, você não sabe os números dos calabouços que originalmente tinham valor 1 ou 4, e ainda não se sabe quais calabouços não tinham sido numerados.

Gabriel gostaria que você analizasse cada jogo e dissesse pra ele se o jogo poderia ter sido jogado até sua conclusão ou não. Um jogo é considerado jogado até a conclusão se todos os calabouços tiveram um número associado entre 1 e 5. Note que um jogador pode ganhar um jogo antes que todos os calabouços tenham sido numerados, mas nós consideramos um jogo completo se todos os calabouços foram numerados. Gabriel não se importa com quem ganhou; ele apenas quer saber se ele e seu irmão puderam jogar o jogo até sua conclusão, assumindo que ambos jogaram de forma ótima e ainda que ambos tinham objetivo de completar o jogo. Note que, para o propósito desse programa, o objetivo não é ganhar a partida; ao invés disso é jogar até sua conclusão.

Entrada

A primeira linha da entrada contém um número positivo, t indicando o número de calabouços a processar. Cada calabouço começa com uma linha contendo dois números inteiros positivos, n $(1 \leq n \leq 26)$, que é a quantidade de calabouços no mapa e m $(1 \leq m \leq 325)$, que é o número de passagens secretas conectandos os calabouços. A seguir, temos m linhas indicando as conexões entre os calabouços. Cada linha é da forma i j $('A' \leq i \leq j \leq' Z')$, que indicam a conexão de uma passagem secreta entre os calabouços i e j (passagens secretas podem ser atravessadas em qualquer direção). Em seguida, temos uma linha contendo um número inteiro k $(0 \leq k \leq n)$, representando quantos calabouços já possuem números. A seguir, são apresentadas k linhas da forma i c $('A' \leq i \leq' Z')$ e $c = \{2, 3, 5\}$. É garantido que cada uma dessas k linhas fornecem um calabouço único que existe em algum mapa, e a numeração

do calabouço não viola qualquer regra do jogo. Também é garantido que exista um caminho entre quaisquer dois calabouços no mapa, ou seja, alguém pode ir de qualquer calabouço a qualquer outro usando as passagens secretas (conexões) da caverna.

Saída

Para cada calabouço na entrada, imprima a linha "`Jogo #x: s`", onde x é o número do calabouço, começando em 1, e s indica uma das seguintes mensagens "`Gabriel e Lucas jogaram este jogo ateh o fim!`" ou "`Gabriel e Lucas NAO completaram este jogo!`", dependendo do resultado da análise da partida.

Deixe uma linha em branco após cada caso de teste. Siga o formato apresentado no exemplo a seguir.

Exemplo de entrada	**Exemplo de saída**
3 3 3 A B A C B C 1 A 3 3 3 A B A C B C 0 4 5 A B A C B C B D C D 2 A 5 D 2	calabouco #1: Gabriel e Lucas jogaram este jogo ateh o fim! calabouco #2: Gabriel e Lucas NAO completaram este jogo! calabouco #3: Gabriel e Lucas jogaram este jogo ateh o fim!

10 Problema N: Ninguém Merece este *Mala*!

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

Recentemente foi implantado o estacionamento privativo para professores e servidores no CCT-UDESC. Afinal, estes necessitam chegar no horário de seus compromissos com aulas e reuniões e não podem perder tempo procurando vagas num estacionamento que é distribuído amplamente por todo o campus universitário.

Considerando que as pessoas que estudam na universidade (pública e gratuita) são educadas e conscientes, a Direção não colocou cancelas ou barreiras no acesso à área de estacionamento exclusivo de professores e servidores. Ao invés disso, de forma educada e conscientizadora, fez uma campanha junto aos estudantes para que os mesmo parem seus carros nas vagas não-privativas ou até mesmo não venham de carro! Afinal, transporte público, bicicletas e andar a pé são alternativas de mobilidade mais sustentáveis!

No início da implantação, tudo parecia correr bem ... mas com o passar do tempo, alguns estudantes esqueceram de tais vagas destinadas a professores e servidores e começaram a estacionar seus carros nestes locais.

Preocupado o problema que estava ocorrendo, o professor Claudius Virus resolveu ajudar no processo de conscientização e começou a anotar diariamente as placas dos carros de alunos que se aproveitavam das vagas privativas! Várias outras pessoas interessadas no bem comum passaram a ajudar neste movimento. Ficou decidido que, no final do ano, será exposto um *ranking* de todos alunos que infringiram a lei por mais de 3 vezes, até 9. A partir de 10 infrações, os alunos receberão um rótulo especial por gerarem problemas de convivência social: `NINGUÉM MERECE ESTE MALA !`

Problema

O professor Claudius Virus solicita à sua equipe de programadores que gere um relatório de infratores a partir de uma listagem das placas dos infratores. Os infratores mais insistentes (dez ou mais infrações) deverão ser marcados com o rótulo `NINGUÉM MERECE`.

Entrada

A primeira linha de entrada contém um único inteiro positivo, n ($1 < n \leq 200$), indicando o número de conjuntos de dias que foram anotado infrações. Cada um dos n dias da entrada seguem abaixo. O primeiro inteiro positivo de cada linha na entrada p ($1 \leq p < 100$), representa o número das placas que cometeram uma infração naquele dia. Como o estacionamento da UDESC tem a capacidade para umas 120 vagas, estima-se num pior caso 100 infratores num pior dia.

O restante da linha contém p valores que representam os números das placas dos carros infratores naquele dia. Estes p números por linha são valores de $0 \leq n \leq 9999$. Todas as placas tem 4 dígitos; exemplo: a placa `127` é entendida como `0127`, idem para a placa `7`, que é entendida como `0007` (quase que o Jaime Bond aparece de novo aqui na UDESC!)

Saída

O relatório final deve ser gerado em ordem crescente de placa, seguindo os seguintes critérios:

- Não imprimir nada para infratores com até 3 penalidades;

- Imprimir o número da placa e o número de infrações para quem cometeu mais de 3 infrações e menos de 10;
- Imprimir o número da placa e o número de infrações para quem cometeu mais de 10 infrações com uma mensagem especial para estes energúmenos;

Veja os exemplos a serem seguidos bem como a formatação dos números das placas (4 dígitos).

Exemplo de Entrada	**Exemplo de Saída**
12 5 3 9999 2 000 4500 4 6 1 5 9999 4 1 2 9999 0 8 3 1 9999 2 13 11 999 21 4 6 1 5 9999 8 1 2 9999 0 11 22 99 100 4 3 1 9999 2 4 6 1 5 9999 4 1 2 9999 0 4 3 1 9999 2 4 6 1 5 9999 1 9999	Placa: 0000 tem: 4 infracoes! Placa: 0001 tem: 10 infracoes! (NINGUEM MERECE) Placa: 0002 tem: 7 infracoes! Placa: 0003 tem: 4 infracoes! Placa: 0005 tem: 4 infracoes! Placa: 0006 tem: 4 infracoes! Placa: 9999 tem: 12 infracoes! (NINGUEM MERECE)

Vamos cantar?

Este problema me lembra uma música que a professora do jardim cantava, para ensinar o alfabeto à turminha:

```
A batata é amarela - A
Vou andar de bicicleta - B
E depois do canto - C
Vou imprimir com %04d -D ...
```

e assim seguia a música!

11 Problema T: Tanque Cheio?

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

Depois de verificar os recibos de sua viagem de carro pela Europa neste verão, você percebeu que os preços de gasolina variaram entre as cidades que visitou. Talvez você pudesse ter economizado algum dinheiro se fosse um pouco mais esperto sobre o local onde completou seu combustível?

Para ajudar outros turistas (e economizar seu próprio dinheiro da próxima vez), você quer escrever um programa para encontrar a forma mais barata de viajar entre cidades, enchendo seu tanque no caminho. Assumimos que todos os carros usam a mesma unidade de combustível por unidade de distância, e iniciam com um tanque de gasolina vazio.

Entrada

O conjunto de entrada é formado por apenas **uma instância de problema**, com diversas consultas. A primeira linha de entrada é formada por $1 \leq n \leq 1000$ e $0 \leq m \leq 10000$, o número de cidades e estradas. Então segue uma linha com n inteiros $1 \leq p_i \leq 100$, onde p_i é o preço do combustível na i-ésima cidade. Então seguem m linhas com três inteiros $0 \leq u, v < n$ e $1 \leq d \leq 100$, dizendo que há uma estrada entre u e v com extensão d. Em seguida, uma linha com o número $1 \leq q \leq 100$, referente ao número de consultas, e q linhas com três inteiros $1 \leq c \leq 100$, s e e, onde c é a capacidade de combustível do veículo, s é a cidade de partida, e e é o destino.

Saída

Para cada consulta, exiba o preço da viagem mais barata a partir de s até e, usando um carro com a capacidade dada, ou "`IMPOSSIVEL`" se não há nenhuma maneira de sair de s e chegar em e com um dado carro.

Exemplo de entrada	**Exemplo de saída**
5 5 10 10 20 12 13 0 1 9 0 2 8 1 2 1 1 3 11 2 3 7 6 10 0 3 20 1 4 20 2 3 20 1 3 15 3 1 15 3 7	170 IMPOSSIVEL 100 80 96 IMPOSSIVEL