A - Lavanderia

No problema mais simples da prova a solução consiste em verificar se o valor N está entre LA e LB, e entre SA e SB. Se sim, a resposta é possível, caso contrário a resposta é impossível.

```
if(la <= n  &&  n <= lb  &&  sa <= n  &&  n <= sb) {
        printf("possivel\n");
} else {
        printf("impossivel\n");
}
```

B - Triângulo Trinomial

Há duas potenciais soluções para este problema. A primeira se baseia na força-bruta, onde calculamos e salvamos todos os elementos do triângulo. Tal solução pode ser executada dentro dos limites de tempo, logo ela é uma solução válida.

Na segunda solução, é possível notar que, para toda linha R maior que 0, a soma dos elementos da linha R é igual a soma dos elementos da linha anterior multiplicado por 3. Logo, podemos generalizar o resultado para 3^R.

Notem que, dado que o limite da variável int na linguagem C/C++ é igual a 2^{31}-1, o resultado do caso de teste mais alto (R = 20) pode resultar em overflow ($3^{20} > 2^{31}$-1). Para representar tal valor é preciso utilizar uma variável com um limite maior, tal como o unsigned int ou o long long.

```
// tendo incluido a biblioteca "math.h"
printf("%u\n", (unsigned int)pow(3, R));
```

C - Economia Brasileira

Seja **Q** o número de brasileiros que participaram da pesquisa, e **A** o número de brasileiros que consideram o cenário atual satisfatório (a quantidade de 0's), imprima Y caso A > Q/2, e N caso contrário. Note que "maioria" aqui significa "mais que a metade", logo em casos de empate a resposta é N.

```
if(A > Q/2) {
        printf("Y\n");
} else {
        printf("N\n");
}
```

D - Ferozes e Curiosos

Editorial escrito pelo autor do problema, professor Leandro Zatesko:
https://drive.google.com/file/d/0BzkHymyjCVJebTZEWXRJV2xCSWc/view?usp=sharing

E - Cortando Canos

Este problema lembra um problema clássico chamado Problema da Mochila (Knapsack Problem), o qual pode ser resolvido com Programação Dinâmica. Basta resolvermos o problema principal (dividir e vender o cano de tamanho T) baseando-se em sub-problemas (dividir e vender canos de tamanho menor que T). A solução de tais sub-problemas podem ser combinadas para se chegar a uma solução ótima do problema principal.

Seja dp[i] o lucro máximo obtido por se dividir e vender um cano de tamanho i. Como caso base, temos que dp[0] = 0, pois não há lucro vendendo-se nada. Agora para todo i maior que 0 e menor ou igual a T, e para todo comprimento de cano Cj e valor de venda Vj desejado pelos clientes, resolvemos dp[i] ao maximizarmos a seguinte expressão:

dp[i] = max(dp[i], dp[i - C[j]] + V[j]), para todo j entre 0 e N-1, inclusive.

Logo, temos:

```
for(int i=0; i<=T; i++) {
        dp[i] = 0;
        for(int j=0; j<N; j++) {
                if(C[j] >= i) {
                        dp[i] = max(dp[i], dp[i - C[j]] + V[j]);
                }
        }
}
printf("%d\n", dp[T]);
```

Notem que esta é uma variação do Problema da Mochila onde é possível usar o mesmo cano mais de uma vez. A solução seria ligeiramente mais complexa caso isso não fosse verdade.

F - O Rato no Labirinto

Este problema é uma variação do problema de caminho mínimo em grafos. Entre os algoritmos mais indicados para se resolver este problema em questão é a Busca em Largura

(Breadth First Search). Seja bfs(A, B) a distância do caminho mínimo entre os vértices A e B, o resultado do exercício se resume a bfs(“Entrada”, “*”) + bfs(“*”, “Saida”).

A implementação pode ser dividida em duas partes: (1) leitura e conversão dos dados, e o (2) cálculo do caminho.

(1) Como os vértices aqui são representados por strings (em vez de números, como usualmente), é necessário um trabalho extra para se manipular cada vértice. Uma alternativa é o uso da estrutura Mapa (Map) da biblioteca STL (“map”) para se converter cada string em um inteiro distinto. Após tal conversão é possível construir uma lista de adjacência sem complicações.

(2) É possível implementar a Busca em Largura utilizando-se a estrutura Fila (Queue) da biblioteca STL (“queue”). Mais detalhes em breve.

G - Onde Estão Minhas Chaves

Precisamos manter registro de todas os escritórios visitados por Gabriel na última semana e, conforme ele verificar os escritórios visitados nos dois últimos dias, atualizamos tais escritórios como visitados.

Inicialmente seja *visitado* um vetor de inteiros com até 1000 posições (número máximo de escritórios), onde visitado[i] = 1 caso o escritório i já tenha sido visitado na última semana, e 0 caso contrário.

Conforme Gabriel confere os escritórios visitados nos últimos dois dias, imprimimos 0 caso visitado[i] = 1, e 1 caso contrário. Atualizamos então visitado[i] para o valor 1.

H - Brincando com Números

Seja S o número de dígitos do valor X, uma solução trivial do problema seria gerar todas as S! permutações do valor X em Y, e testar se N = X+Y é um quadrado perfeito. Porém, devido ao limite de S = 12 dígitos em X, tal solução com complexidade S! é lenta demais.

Eis a interpretação de Marcelo Pinto da solução proposta pelo autor do problema, o professor Fidel Schaposnik:

Precisamos encontrar um inteiro m, tal que n = m*m e n = x+y, mas não conhecemos n, m e nem y.

Então vamos adivinhar m. Como?

O menor valor de m é $\lceil \sqrt{x} \rceil$, quando y = 0.

O maior valor de m é $\lceil \sqrt{x * 11} \rceil$, quando y = 10x.

Se y > 10x, então x e y não tem o mesmo número de dígitos.

Então, para cada inteiro g no intervalo $\left[\sqrt{x}, \sqrt{x * 11} \right]$ nós encontramos algum número y = g*g-x e testamos se este é uma permutação de x.

Se sim, nós encontramos uma solução válida.

I - Catálogo de Livros

Dado que temos 5 categorias de livros, cada uma com P, M, F, Q e B livros, respectivamente, é possível notar que temos um total de P*M*F*Q*B conjuntos distintos de livros. O exercício nos pede a soma dos valores dos K conjuntos mais caros.

Dados os limites baixos, é possível implementar uma solução “trivial” para o problema, onde geramos todos os conjuntos distintos, os ordenamos, e somamos os K maiores:

```
// tendo incluido a biblioteca “vector”
std::vector<int> v;

// gerando os valores
for(int p=0; p<P; p++) {
        …
                for(int b=0; b<B; b++) {
                        std::v.push_back(valorP[p] + valorM[m] + valorF[f] + valorQ[q]
+ valorB[b]);
                }
        …
}

// ordenando os valores
// tendo incluido a biblioteca “algorithm”
std::sort(v.begin(), v.end());
```

J - Matring

Temos aqui um problema de implementação, onde o enunciado nos diz exatamente o que fazer. Podemos dividir a solução em quatro partes: (1) leitura, (2) conversão, (3) decodificação e (4) impressão.

(1) Na linguagem C/C++, nenhum tipo de variável inteira pode conter um valor de até 82 dígitos. Um truque que podemos usar é fazer tal leitura em uma cadeira de caracteres, uma string.

(2) A conversão pode ser feita com aritmética básica, onde para cada coluna, o dígito de cima é multiplicado por 1000, o segundo é multiplicado por 100, o terceiro por 10, e o último por 1. Por exemplo, se tivéssemos os dígitos 3, 5, 2 e 7 o valor convertido seria 3*1000 + 5*100 + 2*10 + 7*1 = 3527.

Lembrem-se de converter o valor da variável em caractere para inteiro:

```
int digito = s[i]-'0';
```

(3) A decodificação pode ser feita conforme especificada no enunciado.

(4) Ao fim convertemos os valores em inteiro para seus respectivos caracteres na tabela ASCII.

```
printf("%c", char(valor));
```

K - Precisa-se de Matemáticos em Marte

Com a palavra, Carlos Vinícios:

Resumindo o problema: recebemos um vetor de **N** posições (1 <= **N** <= 10^5) e precisamos realizar consultas de dois tipos:

1. Alterar o valor da posição **i** (1 <= **i** <= N) para zero;
2. Calcular a soma do intervalo **[1,i-1]**.

Uma solução trivial consiste em fazer exatamente o que o exercício pede: manter um vetor comum contendo os valores informados. Assim, a cada consulta do tipo 1, simplesmente atualizaríamos o valor daquela posição, e a cada consulta do tipo 2, percorreríamos todo o intervalo calculando a soma. Note, no entanto, que cada consulta do tipo 2 teria complexidade **O(N)**, ocasionando em uma complexidade final de **O(Q*N)**, sendo **Q** o número de consultas. A questão não nos informa a quantidade de consultas, já que a lista termina com EOF (fim-de-arquivo), mas tendo em vista o alto limite do valor **N**, uma quantidade razoável de consultas seria suficiente para recebermos **TLE**.

Precisamos, portanto, de uma estrutura que processe consultas desse tipo de forma mais eficiente, tal como uma "**Segment Tree**" ou uma "**Binary Indexed Tree**". Utilizando uma dessas estruturas, podemos obter uma complexidade **O(log N)** para cada uma das consultas de ambos os tipos. Para essa questão, apresentarei uma solução envolvendo a última, já que seu código é consideravelmente menor. A complexidade final é **O(Q*log N)**.

Segue o código: http://ideone.com/Sb8E78

L - Soma Natural

Uma das soluções do problema consiste na força-bruta, ou seja, somamos todos os valores entre A e B, um por um. Porém, dados os limites altos do problema, tal solução ultrapassa o limite de tempo.

Uma fórmula bem conhecida por estudantes de matemática resolve nosso problema: Seja F(N) a soma dos inteiros entre 1 e N, inclusive, temos que F(N) = (N * (N+1))/2.

Com isso, podemos combinar o conceito de inclusão-exclusão, onde somamos todos os inteiros entre 1 e B, inclusive, e então subtraímos todos os inteiros entre 1 e (A-1), inclusive, resultando na soma de todos os inteiros entre A e B, inclusive.

Logo, temos como resposta F(B) - F(A-1).

M - Agente 004

Com a palavra, Thalyson:

Esse problema tem solução simples, porém está um pouco disfarçada. Para chegar na solução, temos que notar 3 coisas:

1º: Os criminosos podem interceptar Bino em qualquer lugar da cidade (Em todas as rotas em todos os lugares, inclusive, nos lugares onde estão as sedes de treinamento inicial e a destino), porém, é fácil notar que se um criminoso pode interceptar Bino no meio de uma rota, também poderia interceptar em um local. Então podemos considerar que os criminosos não interceptam no meio das rotas.

2º: Seja B um caminho que Bino tomou. Para um criminoso conseguir interceptar Bino em alguma local nesse caminho, o caminho deve ter pelo menos um local em que a menor distância do criminoso até esse local seja menor ou igual a distância percorrida por Bino até ele.

3º: Seja B um caminho que Bino tomou, e ele é formado por uma sequência de locais onde Bino esteve (L_1, L_2, L_3, ... , L_n). É fácil notar que se a menor distância de um criminoso para algum local L_x é menor ou igual a distância que Bino percorreu até L_x, também será verdade para qualquer L_y ($x \leq y$), mas nem sempre será verdade para L_k($x > k$). Logo, um criminoso alcança Bino se a menor distância dele para o destino final for menor ou igual a distância que Bino percorreu no caminho.

Com essas ideias, é fácil perceber que o melhor caminho para Bino é o caminho mais curto até o destino final. E o melhor caminho para cada criminoso também é o menor caminho até o destino final. Portanto, basta saber se a menor distância de cada criminoso para o destino final é menor ou igual a menor distância de Bino ao destino final. Para isso basta utilizar um dijkstra no grafo de Bino (Grafo com apenas as rotas conhecidas pelo Bino), e um dijkstra no grafo dos criminosos (Grafo com todas as rotas possíveis). Depois verificar quais criminosos conseguem chegar com tempo igual ou menor que Bino no destino final.

N - Triângulo Trinomial, a Vingança

Com a palavra, Thalyson:

Como já explicado no problema B, a solução do problema é 3^R, mas dessa vez, (3^R)%(2^31 - 1).

Para resolver esse problema:

1º Calculamos o módulo: MOD = 2^31 - 1 = 2147483647.

2º Elevamos 3 à potência R utilizando exponenciação por quadrado para resolver em tempo O(log R). A ideia básica do algoritmo é:

```
long long MOD = 2147483647; // 2^31 - 1 = 2147483647
scanf("%lld", &R);
long long ans = 1LL; // Inicializa resposta como 1
long long P = 3LL; // Inicializa a potência atual como 3
while(R){ // Enquanto R maior que zero
        if(R&1) ans = (ans*P)%MOD; // Se k for impar, multiplica a resposta pela potência e tira o módulo.
        P = (P*P)%MOD; // Eleva a potência ao quadrado e tira o módulo.
        R = (R >> 1); // R é dividido por dois
}
printf("%lld\n", ans);
```

Escrito por: Cristhian Bonilha.
Colaboração: Marcelo Pinto, Thalyson Nepomuceno, Carlos Vinícios.