MARATONA DE PROGRAMAÇÃO

InterFatecs

Taquaritinga, 2017

Soluções

Apoio:

Realização:

www.interfatecs.com.br

6ª MARATONA DE PROGRAMAÇÃO
InterFatecs2017

Problema A
Classificados para a final

Arquivo fonte: interfatecs.{ c | cpp | java }

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Problema basicamente de ordenação, requeria que se criasse a classificação final da prova e selecionasse as equipes. Uma solução rápida requeria 3 ordenações: uma primeira utilizando como critério principal a escola (em ordem crescente ou decrescente) seguida da quantidade de acertos (em ordem decrescente) e de tempo total (em ordem crescente). Com isso processava-se as equipes campeãs de cada escola, copiando-as para um outro vetor ou marcando-as como classificadas apenas. Em seguida, faz-se nova ordenação, agora ignorando a escola e as equipes já classificadas: ordem decrescente dos problemas resolvidos e ordem crescente do tempo. Processa-se as equipes melhores classificadas até a quantidade de vagas ser atingida, marcando-as como classificadas ou copiando o código da equipe para um outro vetor. Finalmente, ordena-se o conjunto de equipes classificadas em ordem crescente do código da equipe e faz-se a impressão, tomando o cuidado de colocar as vírgulas separadoras e os espaços em branco requeridos, com o ponto final após a última equipe.

Considerando a disponibilidade de rotinas de ordenação pré-definidas nas linguagens utilizadas na prova, não havia uma maior complexidade no problema.

Problem B
Urinals

Source file: urinals.{ c | cpp | java }

Author: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Era um dos problemas realmente fáceis da prova. Como entre duas pessoas que ocupam mictórios no banheiro é necessário que haja um mictório vazio, se temos N pessoas, precisaremos de N – 1 mictórios vagos. Assim, bastava receber a quantidade N da entrada padrão do computador e imprimir o resultado correspondente à soma N + (N – 1).

Problema C
Boleto

Arquivo fonte: boleto.{ c | cpp | java }

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Esse problema requeria uma certa atenção ao se mapear os pesos de cada dígito do código de barras para o correspondente na linha digitável. Isso poderia ser conseguido por meio de um vetor de inteiro, com um elemento para cada dígito a ser processado contendo o peso referente àquele dígito. Esse vetor poderia ser inicializado manualmente, com muita atenção aos valores de cada elemento. Após lido o conteúdo da entrada padrão, bastava multiplicar o valor do dígito lido pelo peso da posição correspondente no vetor e acumular esse produto, aplicando as regras de cálculo ao final do processamento.

6ª MARATONA DE PROGRAMAÇÃO InterFatecs 2017

Problema D
Bingo

Arquivo fonte: bingo.{ c | cpp | java }
Autor: Leandro Luque (Fatec Mogi das Cruzes)

Este é um problema simples que envolve o uso de matrizes. Entre as possíveis soluções, pode-se criar matrizes de números inteiros para armazenar as cartelas conforme elas são informadas na entrada. Em seguida, para cada bola sorteada, pode-se iterar pelos números das cartelas e alterá-los para 0 (zero), que é equivalente ao valor da figura na cartela, se forem iguais ao número sorteado. Em seguida, pode-se passar por cada linha e coluna de cada cartela e verificar se alguma delas contém apenas valores iguais a 0 (zero). Se esse for o caso, algum jogador ganhou. Por meio de um contador, pode-se contar quantos jogadores ganharam na rodada e, em outra variável, pode-se armazenar o número da rodada. Nas rodadas seguintes (até o fim do jogo), não é necessário fazer nada além de ler os dados. Esta seria uma solução válida. De toda forma, ela poderia se tornar mais rápida por meio de um índice invertido. Este índice armazenaria, para cada possível valor de bola sorteada, as cartelas, linhas e colunas onde este número aparece. Isto poderia ser feito durante a entrada do problema. Desta forma, não seria necessário procurar os números na cartela, o que reduziria bastante a complexidade da solução. Ainda, seria possível, para cada cartela, manter uma linha e coluna extras, onde seriam armazenas as somas dos valores em cada linha/coluna. Estas fileiras seriam iniciadas durante a entrada e atualizadas após o sorteio de cada bola (subtraindo o valor da bola encontrada de cada fileira onde ela se encontra). Fazendo desta forma, não é mais necessário passar por cada linha/coluna para verificar se ela só possui valores iguais a 0, bastando verificar se a soma de cada linha/coluna é igual a 0.

Problema E
Nomes

Arquivo fonte: nomes.{ c | cpp | java }
Autor: Danilo Ruy Gomes (Fatec Itapetininga)

Problema envolvendo cadeias de caracteres bem simples, onde bastava comparar o nome do pai ou da mãe, caractere a caractere pelo menos até a 4° posição, caso encontrasse algum nesta condição, bastaria imprimir o termo verificar, senão normal.

Exemplo

INDICE	0	1	2	3	4	5	6	7	8
NOME	F	L	A	V	I	O			
FILHO	F	L	A	V	I	E	L	Y	
FILHO	F	L	A	V	I	A			

6ª MARATONA DE PROGRAMAÇÃO InterFatecs 2017

Problema F

Mario

Arquivo fonte: mario.{ c | cpp | java }

Autor: Anderson V. de Araujo/Alessandro F./Lucas T. (UFMS)

Esse problema pode ser resolvido com busca em largura em uma grid. Para isso você pode armazenar o mapa como uma matriz de chars. Uma sugestão é armazenar a matriz a partir da linha e coluna 1 e preencher as linhas 0 e H+1 e colunas 0 e C+1 com obstáculos.

Exemplo:

```
S.
.x
xT

    xxxx
xS.x
x.xx
xxTx
```

Fazendo isso no momento da busca, você não irá precisar validar se você está saindo do mapa e acessando posição inválida da memória, deixando as condicionais mais simples e diminuindo a chance de Runtime Error.

Um detalhe que você deve prestar atenção ao problema é que ele não indica a quantidade de T, ou seja, pode haver nenhum ou vários T no mapa. No caso de existir você deve priorizar o mais próximo.

Após isso, a partir da posição S você roda uma busca em largura em todo o mapa, até encontrar uma célula T. Uma possível abordagem para essa busca é você definir uma struct em C, que armazena a linha e coluna de cada célula:

```
typedef struct
{
    int x, y;
} cell;

cell queue[1010 * 1010];
int beg, end;
```

E partindo de S você enfileira todas as posições adjacentes à sua célula atual que represente uma célula livre e que você não tenha visitado anteriormente:

```
v = queue[beg++];
if(grid[v.x+1][v.y] == '.' && d[v.x+1][v.y] == INF)
  …
if(grid[v.x-1][v.y] == '.' && d[v.x-1][v.y] == INF)
  …
if(grid[v.x][v.y+1] == '.' && d[v.x][v.y+1] == INF)
  …
if(grid[v.x][v.y-1] == '.' && d[v.x][v.y-1] == INF)
  …
```

Caso a célula atual seja um ponto de tele transporte você retorna a distância daquele ponto a S.

```
if(grid[v.x][v.y] == 'T')
 return  d[v.x][v.y];
```

Por fim, como ele gasta um coração inteiro a cada 4 passos na caverna, você deve verificar se a distância do tele transporte é menor ou igual a 4*C.

```
dist = bfs(...);
printf("%s\n", (dist <= c*4) ? "SIM" : "NAO");
```

Você pode notar que você irá enfileirar uma célula apenas na primeira vez que você a visite. Sendo assim a complexidade do algoritmo no pior caso é O(H*C), que é linear no número de células do mapa.

6ª MARATONA DE PROGRAMAÇÃO InterFatecs 2017

Problema G
Questão de Lógica

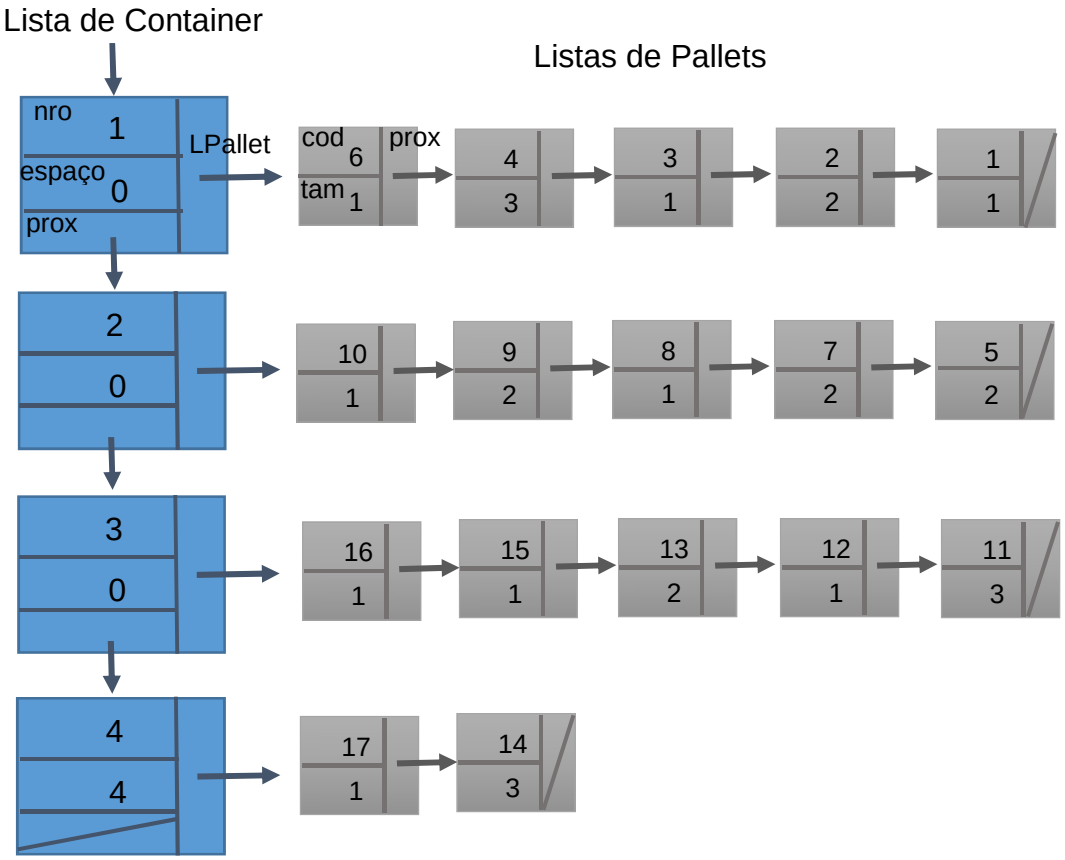
Arquivo fonte: logica.{ c | cpp | java }
Autor: Lucio Nunes de Lira (Fatec São Paulo)

Um dos problemas mais simples da competição! Poderia ser resolvido com duas linhas de código. Os exemplos de entrada/saída (além da própria ilustração) deixavam claro que a base do funil era sempre o número *N*×2-2.

Problema H
Logística de Container

Arquivo fonte: logistica.{ c | cpp | java }
Autor: Julio Lieira (Fatec Lins)

Este problema exigia conhecimentos de Estrutura de Dados, especificamente, o conceito de Lista Encadeada. Um pouco trabalhoso, mas depois de estruturado corretamente, o problema se resume em inserir e localizar em Lista Encadeada. Assim, teríamos uma Lista de Containers, onde cada nó aponta para uma Lista de Pallets.



Abaixo a declaração das estruturas dos nós das Listas conforme desenho:

```
struct noPallet {   //No da Lista de Pallets
 int cod;  //Código do Pallet
 int tam; //Comprimento do Pallet
 struct noPallet *prox; //Ponteiro para o Próximo Pallet
};

struct noCcontainer {  //No da Lista de Containers
 int nro; //Número do Container
 int espaco; //Espaco restante neste container
 struct noPallet *LPallet; //Ponteiro para a Lista de Pallet do Container
 struct noCcontainer *prox; //Ponteiro para o Próximo Container
};
```

A cada tamanho de Pallet da entrada, percorrer a lista de Containers para encontrar o primeiro Container com espaço para acomodar o Pallet. Se encontrar, inserir o Pallet no início da Lista de Pallets do referido Container. Não esquecer de atualizar o campo espaço do nó do Container onde o Pallet foi inserido. Caso não encontre um Container com espaço suficiente para armazenar o Pallet, criar um novo Container e inserir o Pallet.

Após efetuar a entrada dos Pallets, basta ler o código do Pallet sendo procurado e localizá-lo na Lista de Containers.

Problema I
Funil de Estrelas

Arquivo fonte: funil.{ c | cpp | java }

Autor: Lucio Nunes de Lira (Fatec São Paulo)

Um problema bem simples de resolver para as equipes que perceberam os detalhes do enunciado. Uma breve contextualização é dada para explicar justamente os dois tópicos que deverão ser levados em conta para a construção da solução:

- A soma de números ímpares e pares e as consequências disso;
- A explicação do que é *"A Sequência de Fibonacci"*.

Essencialmente o problema exigia a leitura de um número relativamente grande (até 2^{64}-1), que é o máximo que uma variável unsigned long long int da linguagem C comporta, e verificar se o número é divisível por três.

Mas temos detalhes! Esses detalhes se referem justamente a lembrar do tipo da variável adequada para o armazenamento, como fazer a leitura desse valor e entender (o que bastava ler o enunciado, diga-se de passagem) que o valor que deve ser verificado é o próprio *N* e não o *N*-ésimo termo da *Sequência de Fibonacci,* o que seria impraticável, pois exigiria mais de dezoito quintilhões de operações no pior caso, dependendo da forma como fosse calculado.

6ª MARATONA DE PROGRAMAÇÃO InterFatecs 2017

Problema J
Celular do Juvenal

Arquivo fonte: celular.{ c | cpp | java }

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Este problema requeria que se recebesse uma palavra da entrada e a processasse caracter a caracter, totalizando a quantidade de toques no teclado do aparelho celular. Se o caracter encontrado era um espaço em branco, acrescentava-se um toque no total, se fosse uma letra minúscula, a quantidade de toques correspondente àquela letra era determinada pela figura contida no enunciado. Se a letra fosse em maiúsculas, a quantidade de toques era a quantidade de toques para a minúscula correspondente acrescida da quantidade de letras disponível naquela tecla.

Uma solução bem simples seria criar dois vetores de inteiros, com 26 posições cada um. Um vetor conteria a quantidade de toques para cada minúscula (elemento 0 com a quantidade de toques para a letra ‘a’, elemento 1 com a quantidade de toques para a letra ‘b’, etc) e outro para as maiúsculas (elemento 0 com a quantidade de toques para a letra ‘A’, elemento 1 com a quantidade de toques para a letra ‘B’, etc). Ao processar a letra, testar se é maiúscula ou minúscula e obter no vetor correspondente a quantidade de toques a acrescentar no total. Nesta solução cada vetor tem a quantidade específica de cada letra, na ordem alfabética, então para encontrar o elemento correspondente bastava calcular Letra – ‘a’ para obter a posição no vetor de minúsculas e Letra – ‘A’ para o vetor de maiúsculas.