Tutorial: Balinhas

Edson Alves da Costa Júnior

Como $N \leq 2 \times 10^5$, avaliar todos os N^2 pares de caixas possíveis leva ao TLE.

Uma maneira de resolver este problema é organizar as caixas em um dicionário cuja chave é o resto da divisão do número de balinhas por M. Seja x_r a maior quantidade de balinhas dentre as caixas cujas quantias deixam resto r. Assim, x_r deve fazer par com x_{M-r}, se existir.

Deve se tomar cuidado, porém, com dois casos especiais: os casos onde $r = 0$ e $r = M/2$, quando M for par. Nestes casos, as caixas escolhidas devem ser as duas melhores de suas respectivas classes. Assim, serão avaliados, no máximo, $N/2 + 1$ pares de caixas, de modo que a complexidade da solução é $O(N \log N)$, por conta da organização no dicionário e da possível ordenação das caixas em cada classe de restos.

Tutorial: Preservando o Cerrado

Daniel Saad Nogueira Nunes

Este problema pode ser resolvido em três etapas:

1. Primeiramente é necessário encontrar todas as arestas que são pontes, o que pode ser feito com uma adaptação bem conhecida do algoritmo de Tarjan em tempo $\Theta(|V| + |E|)$. Estas arestas podem ser adicionadas em uma estrutura associativa (`set` ou `unordered_set`).

2. Para cada aresta, verifique se ela é uma ponte através da estrutura associativa. Em caso afirmativo, duplique o peso da aresta. Isto leva tempo $\Theta(|V| + |E|)$ em listas de adjacências caso o `unordered_set` seja utilizado, e $\Theta((|V| + |E|) \lg |V|)$, caso o `set` utilizado.

3. Finalmente, basta aplicar o algoritmo de Dijkstra entre os pontos de interesse de Unberto, o que pode ser feito em tempo $\Theta(|E| \lg |V| + |V|)$, se uma fila de prioridades baseada na estrutura Heap for utilizada.

Tutorial: Substrings Distintas

Edson Alves da Costa Júnior

Este problema pode ser resolvido por meio de dois ponteiros L e R. O ponteiro L aponta para o início da substring a ser avaliada, e o ponteiro R avançará enquanto $S[L] = S[R]$. Inicialmente temos $L = 0, R = 1$.

Quando $S[L] \neq S[R]$, a substring $B = S[L \dots (R-1)]$, de tamanho $R - L$, é composta apenas por caracteres repetidos. Veja que as substrings de B também possuem apenas caracteres repetidos, de modo que basta armazenar, para o caracatere c, a maior substring composta por repetições de c.

Assim, o total de substrings não-vazias distintas compostas apenas por caracteres distintos será a soma destes valores máximos, para cada caractere. Como a cada iteração do laço R avança, no mínimo, uma unidade, e L avança para R, o algoritmo tem complexidade $O(N)$.

Tutorial: Fizz Busão

Guilherme Novaes Ramos

A solução envolve dois contadores, um para veículos e outro para ônibus, que são incrementados conforme a entrada. Toda vez que o contador de ônibus veículos é múltiplo de 5, Jonnie Ruquer pode dizer "busao". Toda vez que o contador de veículos é múltiplo de 3, ele pode dizer "fizz", inclusive quando também pode dizer "busao". Nos demais casos, é só mostrar o contador de veículos.

Tutorial: Soluções

Edson Alves da Costa Júnior

É possível mostrar que, dados a, b, c naturais, com $(a, b) = 1$, existem m, n naturais tais que

$$am + bn = c$$

com $0 \leq m < b$, e esta representação é única.

Se $n < 0$, não é possível escrever c como $ax + by = c$ com x, y não negativos. Isto porque a solução geral da equação diofantina $ax + by = c$ tem a forma

$$\begin{cases} x & = x_0 + bt \\ y & = y_0 - at \end{cases}$$

onde x_0, y_0 é uma solução particular. Se $x_0 = m$ e $y_0 = m$, para manter o valor de x não negativo é preciso que $t \geq 0$. Mas usar t não-negativo em y reduz o valor de y_0 em t vezes a. Assim, se n já for negativo, nunca se tornará positivo sem que m deixe de ser positivo.

Assim, como $0 \leq m < b$, é possível computar todos os valores cuja representação única tem n negativo. Basta fazer $m = 0, 1, 2, \ldots, b - 1$ e calcular os valores $am - bn > 0$ para $n = 1, 2, \ldots$. Estes valores formam o conjunto das lacunas $\mathcal{L}(a, b)$ de a e b.

É possível mostrar que $|\mathcal{L}(a, b)| = (a - 1)(b - 1)/2$, mas para este problema não é necessário conhecer este fato. Mais importante é notar que o maior elemento de $\mathcal{L}(a, b)$ (faça $m = b - 1$ e $n = 1$) é menor do que $(a - 1)(b - 1)$, de modo que é possível gerar este conjunto no tempo limite do problema.

Uma vez gerado o conjunto das lacunas, basta remover de $[1, N]$ os elementos deste conjunto. Vale observar que a construção acima só vale para $(a, b) = 1$: se $(a, b) = d > 1$, é preciso simplificar a equação, observando que a equação diofantina só tem solução quando c é múltiplo de d, de forma que os números que não são múltiplos de N deve ser excluídos da contagem.

Assim, a complexidade da solução é $O(ab)$.

Tutorial: Quantos Movimentos?

Daniel Saad Nogueira Nunes

Seja $\Delta x = |x_c - x|$ e $\Delta y = |y_c - y|$. Tome d' como:

$$d' = \max\left\{\frac{\Delta x}{2}, \frac{\Delta y}{2}, \frac{\Delta x + \Delta y}{3}\right\}$$

O número de movimentos, no caso geral, necessários para o cavalo sair de (x_c, y_c) e chegar em (x, y) pode ser calculado como.

$$d = d' + ((d' + \Delta x + \Delta y) \mod 2)$$

.

Contudo, existem exceções na região 5×5 de alcance do cavalo. Estas exceções estão dispostas a seguir:

$$d = \begin{cases} 4, & \Delta x = \Delta y = 2 \\ 3, & \Delta x + \Delta y = 1 \\ 4, & \Delta x + \Delta y = 1 \end{cases}$$

Tutorial: Distribuidora de Bebidas

Vinicius Ruela Pereira Borges

O problema pode ser resolvido simulando o processo de carregamento e descarregamento das caixas de bebidas pelos caminhões. Podemos definir duas estruturas de dados:

- pilha para simular o estocamento de caixas de bebidas no espaço U;
- fila para simular a atividade dos N caminhões que chegam ao galpão.

Após preparar a fila e a pilha, a ideia é pegar o caminhão que está na frente da fila e deixá-lo realizar suas funções nos espaços U e V, respeitando os critérios estabelecidos no enunciado. Portanto:

- ao ser carregado, deve-se utilizar somente as garrafas das caixas do espaço U, sendo que não se pode colocar parcialmente as garrafas de uma caixa. Se ele conseguir desempenhar suas funções, retire-o da fila. Caso o caminhão não consiga ser carregado com caixas, deve-se enfileirar ele novamente para uma última tentativa.
- deve-se descarregar as garrafas do caminhão (sempre cheio), formar as caixas, empilhá-las no espaço U. As garrafas remanescentes ficam no espaço V. Ao final retire o caminhão da fila.

Desta maneira, pode-se pensar no seguinte pseudo-código abaixo:

```
enquanto fila de caminhoes nao for vazia, faça

    {c,f,tentativa} = informacoes do caminhao da frente da fila;

    caixas_max = c/B;
    sobra_garrafas = c mod B;

    se f == 1

        se o espaco U estiver vazio

            se tentativa == 2
                retira caminhao da fila;

            senao
                desenfileira o caminhao e o enfileira novamente (c,f,tentativa+1);

            fim_se
        senao

            enquanto existirem caixas no espaço U e caixas_max > 0

                desempilha uma caixa do espaço U
                caixas_max = caixas_max -1;
```

```
            fim_enquanto

            o caminhao sai definitivamente da fila;
        fim_se
    senao

        empilha caixas_max no espaço U;

        o caminhao sai definitivamente da fila;

        adicione as garrafas remanescentes no espaco V;

        se(espacoV >= B)

            empilha uma caixa no espaço U;
        fim_se
    fim_se

fim_enquanto

imprime a quantidade de garrafas no espaco V somada a quantidade de garrafas nas caixas do espaço U;
```

Tutorial: Projetando Iniciadores

Daniel Saad Nogueira Nunes

Este problema corresponde ao problema NP-difícil *Closest String* e pode ser resolvido por uma abordagem de busca completa.

Gere todas as possíveis sequências sobre o alfabeto $\{A, C, G, T\}$ de comprimento L e, para cada sequência S gerada, compute a distância máxima de S considerando todas as sequências de entrada. Caso esta distância seja menor que a distância global encontrada, atualize a distância global e guarde a sequência S. A complexidade desta solução é $\Theta(4^L \cdot N^2)$.

As palavras podem ser geradas utilizando *backtracking*, e a distância de hamming pode ser computada mais rapidamente utilizando máscaras de bit e tabelas pré-computadas, apesar disto não ser necessário neste problema. Caso utilize-se tabelas pré-computadas, é possível reduzir a complexidade para $\Theta(4^L \cdot N)$.

Tutorial: Rali de Regularidade

Daniel Saad Nogueira Nunes

Uma forma de resolver este problema é utilizar o `scanf` para ler cada parte inteira da *string* `HH:MM:SS`, como o código a seguir:

```
int hh,mm,ss;
scanf("%d:%d:%d",&hh,&mm,&ss);
```

Dado que as horas estão dispostas em `hh`, os minutos em `mm` e os segundos em `ss`, resta transformar tudo para segundos através de operações de multiplicação e soma:

```
total_segundos = hh*3600 + (mm*60) + ss;
```

Com todos os tempos convertidos em segundos, podemos, para cada trecho, calcular quantos segundos acima ou abaixo do tempo ideal uma determinada equipe ficou em determinado trecho e assim, calcular a penalidade da equipe.

Tutorial: Soma de quadrados

Edson Alves da Costa Júnior

Seja $I(n)$ o maior inteiro i tal que $i^2 \leq n$. A soma dos n primeiros quadrados é dada por

$$S_n = \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Deste modo, a solução é dada por $S = S_R - S_{L-1}$. É preciso, porém, tomar cuidado com a aritmética modular em dois pontos: em primeiro lugar, é preciso computar o inverso multiplicativo de 6 módulo $p = 10^9 + 7$, que é dado por

$$6^{-1} \equiv \ 6^{p-2} \pmod{p}$$

Em segundo lugar, a diferença pode ser negativa. Para evitar isso, basta fazer

$$S = S_R - S_{L-1} + p \pmod{p}$$

Assim, a solução tem complexidade $O(\log p)$, por conta do cálculo do inverso multiplicativo. Se ele já estiver pré-computado, a solução tem complexidade $O(1)$.

Tutorial: Onde está Wally?

Edson Alves da Costa Júnior

A solução do problema é simples: percorrer a matriz da entrada, linha a linha, e cada coluna de cada linha, até encontrar o caractere `'W'`. A solução tem complexidade $O(NM)$.

Tutorial: Postos de Combustível

Vinicius Ruela Pereira Borges

Para resolver o problema, temos que levar em consideração os seguintes fatos do enunciado:

- o professor Guilherme sempre sai de tanque cheio de Brasília;
- sempre que pára para abastecer, o professor Guilherme enche o tanque de combustível de seu veículo;
- o professor Guilherme nunca chega a um posto i ou à cidade litorânea sem combustível no tanque;
- o preço do combustível é calculado por quilômetro de autonomia.

Seja A a autonomia máxima do veículo, a_i a autonomia do veículo ao passar pelo i-ésimo posto da rodovia, ou a cidade litorânea, e p_i o preço do combustível e km_i o quilômetro desse posto na rodovia. Podemos pensar nos seguintes passos:

1. Se Guilherme está em km_i e o carro está com o tanque cheio ($a_i = A$), mas ele não é capaz de chegar ao km_{i+1}, deve-se retornar -1;
2. Se a_i é suficiente para sair de um posto i até chegar ao posto $i+1$, o professor Guilherme pode escolher abastecer ou não seu veículo. Caso ele abasteça, ele terá um custo de $(A - a_i) * p_i$, sendo que p_i é o preço do combustível.
3. Caso contrário, o professor Guilherme obrigatoriamente abastece seu carro a um custo de $(A - a_i) * p_i$.

Pode-ser ver que devemos testar todas as possibilidades possíveis do professor Guilherme abastecer ou não seu carro nos N postos distribuídos pela rodovia, e verificar se ele consegue chegar ou não à cidade litorânea. Nesse sentido, uma solução baseada em força bruta extrapola o limite de tempo do problema (o famoso TLE).

Podemos adaptar o algoritmo utilizando uma estratégia baseada em programação dinâmica. Para isso, criamos uma tabela `tab[1...K][1...A]` para armazenar os resultados intermediários do problema, isto é, se em um determinado $1 \leq km_i \leq K$ da rodovia de parada (posto ou a cidade litorânea) e uma conhecida autonomia $1 \leq a_i \leq A$, qual é a solução do problema: se Guilherme chega a km_i, qual o custo de combustível obtido até então, ou -1 caso não seja possível chegar a um posto ou a cidade litorânea.

Com base nisso, o pseudo-código que resolve o problema acima pode ser elaborado como:

```
int tab[100005][805];
const INF 1e9;

int solucao(int km, int autonomia)
inicio
    se km > N
       retorna 0;
    fim_se

    se tab[km][autonomia] já foi calculado
```

```
      retorna tab[km][autonomia];
   fim_se

   se a distancia para o proximo posto > A
      tab[km][autonomia] = INF;
      retorna tab[km][autonomia];
   senao

      abastece = solucao(km+1,A-posto[km+1]) + (A-autonomia)*p[km];

      se a distancia para o proximo posto < autonomia

          naoAbastece = solucao(km+1,autonomia-posto[km+1]);

          tab[km][autonomia] = min(abastece,naoAbastece);
          retorna tab[km][autonomia];
      senao

          tab[km][autonomia] = abastece;
          retorna tab[km][autonomia];
      fim_se
   fim_se
fim
```

Antes de chamar a função solução pela primeira vez, deve-se inicializar `tab` apropriadamente com valores que signifiquem que uma solução ainda não foi obtida. A primeira chamada da função fica como:

```
   ans = solucao(0,A);

   se (ans < INF)
       imprime ans;
   senao
       imprime -1;
   fim_se
```