Tutorial: Raio de Visão

Daniel Saad

Para resolver este problema, é necessário verificar se o item está presente no círculo de visão dos jogadores. Para isto, basta verificar se a distância do item até cada jogador é menor ou igual ao raio deste círculo.

A distância entre dois pontos a e b por sua vez pode ser calculada através da equação:

$$d(a, b) = \sqrt{(a.x - b.x)^2 + (a.y - b.y)^2}$$

Tutorial: Corretor Automático

Daniel Saad

Para resolver este problema, basta realizar a leitura, comparar a resposta do aluno com a do gabarito e contar quantas questões foram acertadas.

Tendo esta informação, basta dividir o número de questões acertadas pelo total de questões e multiplicar por 10. Na hora da impressão, deve-se limitar a 2 casas de precisão.

Em `C++`, podemos fazer da seguinte forma:

```
cout << "Nota: " << fixed << setprecision(2)
     << (double)10 * ((double)correct_answers_n / n) << endl;
```

Em `C`, podemos usar o `printf` como disposto abaixo:

```
printf("Nota: \%.2f",nota_final);
```

Tutorial: Popstar

Daniel Saad

Uma observação inicial que pode ser feita é que há no máximo um aluno *popstar*, pois se houvesse mais de um, cada *popstar* teria que conhecer o outro, o que é impossível de acordo com a definição de *popstar*.

Seja i o identificador de um aluno `popstar`. Logo, o índice da matriz a que ele se refere é $i-1$, pois a identificação dos alunos começa de 1. Como i é *popstar*, todos o conhecem, logo a coluna $i-1$ deve estar toda preenchida com 1. Analogamente, o aluno i não conhece ninguém, com exceção de si mesmo, logo a linha $i-1$ deve estar toda preenchida com 0, com exceção da célula $M[i-1][i-1]$.

Assim, para encontrar o aluno *popstar*, basta realizar essa verificação para todos os identificadores de aluno.

Caso essa verificação falhe para todos os alunos, não há aluno *popstar*.

Tutorial: Middle Square

Daniel Saad

Neste problema, basta selecionar os dígitos centrais de uma string de 8 caracteres e elevar o número gerado ao quadrado, para obtenção da próxima string de 8 caracteres.

Isso pode ser facilmente obtido com a função sprintf, da linguagem C, que pode imprimir um inteiro em uma string preenchendo com zeros à esquerda quando necessário.

Em C++ o equivalente pode ser obtido da seguinte forma:

```
// declaração de um streamstring
ostringstream out;
// Insere o número next ma stringstream completando com '0's mais à esquerda
out << std::internal << std::setfill('0') << std::setw(8) << next;
// Seleciona os caracteres centrais para compor o próximo número e os converte para inteiro
next = stoi(out.str().substr(2,4));
```

Tutorial: Pisca-Pisca

Matheus Faria

Para resolver esta questão é necessário analisar a frequência das luzes queimadas, para relacionar a posição da luz com a cor que ela pertence basta utilizar a operação de módulo com o N. Desta forma você poderia contabilizar cada ocorrência com $freq[pos\%N]++$,

Tutorial: Reservatório

Vinicius Borges

Com uma solução $\Theta(n^2)$, que obtém a soma de cada intervalo $[i, j]$ do vetor, é possível determinar a soma máxima dos intervalos, conforme disposto abaixo:

```
int64_t max_sum = 0;
for(size_t i=0;i<v.size();i++){
    int64_t sum = 0;
    for(size_t j=i;j<v.size();j++){
        sum+= v[j];
        max_sum = max(max_sum,sum);
    }
}
cout << max_sum << endl;
```

O algoritmo de Kadane [1], consegue resolver o problema em tempo $\Theta(n)$, mas, pelo tamanho da entrada, não é necessário utilizá-lo.

[1] https://en.wikipedia.org/wiki/Maximum_subarray_problem#Kadane's_algorithm

Tutorial: Financiamento

Edson Alves

Há duas soluções possíveis para o problema. Usando a fórmula da soma dos termos da progressão geométrica, obtemos

$$V = \frac{P\left(1 - (1+i)^{-n}\right)}{j}$$

Esta expressão nos permite computar P em $O(\log n)$. Daí basta fazer uma busca binária iniciando com $a = 1, b = 10^9$ e atualizar o resultado sempre que $P \leq M$. Esta solução tem complexidade $O(\log^2 n)$.

Outra solução é isolar o valor de n na fórmula acima, usando logaritmos. Assim temos uma expressão fechada para a solução em $O(1)$ (tome cuidado de usar a função maior inteiro para obter o resultado correto).

Tutorial: Matemática Prefixada

Matheus Faria

Para transformar uma expressão prefixa para infixa é necessário analisar a estrutura das operações disponíveis. Nesta questão apenas quatro operações eram permitidas: $+ - */$, todas elas são operações binárias, ou seja, elas tem 2 operandos.

A estrutura de uma expressão prefixa se dá do seguinte modo

```
A OP B
```

Se torna:

```
OP A B
```

Onde OP é uma dos operadores e A, B os operandos. Sendo assim, uma expressão

```
Y + X * Z
```

Podemos fazer as seguintes relações: $A = Y$ e $B = X * Z$, assim conseguimos reduzir a expressão para $A + B$ que em forma infixa fica $+AB$. Esta relações devem ser expandidas, e para cada expansão você faz o mesmo processamento para tradução em forma prefixa.

Com isso toda vez que temos um operador e dois termos, podemos trazer eles na forma infixa de volta. Para fazer esta análise na questão, podemos utilizar uma estrutura semelhante a uma pilha, onde colocamos todos os elementos da expressão, até que achemos um padrão de 1 operador e 2 termos. Quando achar o padrão, resolva a operação, e continue procurando o padrão com as últimas ocorrências da sua pilha. Quando não achar mais o padrão, volte a adicionar os termos da expressão na pilha.

Tutorial: Quem Eliminar?

Lucas Mattioli

Se os inimigos não se mexessem, o campo de visão de um inimigo i com spawn $S_i = (X_i, Y_i)$ qualquer seria sempre o quadrado determinado pelos 4 pontos: $(X_i-L/2, Y_i-L/2)$, $(X_i-L/2, Y_i+L/2)$, $(X_i+L/2, Y_i-L/2)$ e $(X_i + L/2, Y_i + L2)$. Nesse problema mais simples, basta checar quais inimigos têm marlin em seu campo de visão; isto é, pra cada inimigo basta checar se o ponto em que marlin se encontra está dentro do dito quadrado. Para checar se um dado ponto P está dentro de um quadrado Q, basta que a coordenada X do ponto P esteja dentro do segmento determinado pela menor coordenada X de Q e pela maior coordenada X de Q e que a coordenada Y do ponto P esteja dentro do segmento determinado pela menor coordenada Y de Q e pela maior coordenada Y de Q.

O problema não muda muito caso haja a movimentação dos inimigos: supondo um inimigo i com movimentação horizontal e levando em conta que o mesmo ande por um tempo infinito, a área em que será ameaçada por ele é agora um retângulo dado pelos 4 pontos: $(X_i-L/2-D_i, Y_i-L/2)$, $(X_i-L/2-D_i, Y_i+L/2)$, $(X_i+L/2+D_i, Y_i-L/2)$ e $(X_i+L/2+D_i, Y_i+L/2)$. A solução continua o mesmo para o caso do quadrado: basta checar para quantos inimigos o ponto em que marlin está se encontra dentro de seus respectivos retângulos, utilizando o mesmo método descrito no parágrafo anterior (afinal, um quadrado também é um retângulo). A solução para o caso da movimentação vertical é análoga, mudando apenas onde o D_i é somado: neste caso, nas coordenadas Y.

Tutorial: Conquista de Territórios

Felipe Duerno

Para resolver este problema, emprega-se uma técnica denominada *flood-fill*, que é basicamente uma busca em profundidade.

A ideia é, partindo de cada soldado, efetuar uma busca em profundidade marcando as regiões de terra alcançadas por ele.

Uma curiosidade é que esta técnica é o que está por trás do “balde de tinta da ferramenta `Paint`”, responsável por colorir toda uma região delimitada por *pixels* que não são brancos. Aqui, temos uma situação parecida: a técnica irá marcar todas as regiões de terra delimitadas por água alcançáveis por cada soldado.

Ao fim do processo, o número de regiões de terras marcadas é contado e a resposta é dada.

Tutorial: Noob Authenticator

Jeremias Gomes

O problema utiliza um conceito chamado Desarranjo (representado por $!n$), em que busca-se o número de permutações sem pontos-fixo de uma determinada sequência de elementos. No problema, $n = |S|$ e a fórmula do desarranjo para uma coleção de n elementos é definida recursivamente como:

$$!n = (n - 1) * (!(n - 1)+!(n - 2))$$

Para resolver o problema, a solução recursiva não é suficiente, pois é necessário memorizar os desarranjos já calculados. Assim, uma simples tabela de programação dinâmica auxilia na memorização e resolução do problema.

```
int desarranjo(n):
    tab = [0] * n
    tab[1] = 0
    tab[0] = tab[2] = 1

    for (i = 3; i <= n; i++) {
        tab[i] = (i - 1) * (tab[i - 1] + tab[i - 2] mod 1000000007)  mod 1000000007
    }

    return tab[n]
```

Tutorial: Presente de Natal Palindrômico

Arthur Komatsu

Vamos analisar quantos números palíndromos existem menores que 10^{12}. Existem várias formas de gerar números palíndromos até um número 10^d. A seguir, serão discutidas duas formas distintas:

1. **Força bruta:** Passar por todos os números de 0 até 10^d e verificar se cada um é palíndromo. Essa estratégia é muito lenta para os limites do problema.

2. **Por construção:** Suponha que d seja par. Para cada número de 0 até $10^{d/2}$, observe que é possível gerar um número palíndromo novo de d dígitos concatenando-se com sua representação reversa. Note que ainda é possível gerar outro palíndromo de $d-1$ dígitos concatenando-se com sua representação reversa e ignorando o primeiro dígito. Por exemplo, para o número 123, podemos criar os palíndromos 12321 e 123321. Para o caso d ímpar, basta analisar o caso $d+1$ e verificar se o palíndromo gerado não excede d dígitos.

Observe que, para cada número de 0 até $10^{d/2}$, geram-se unicamente 2 números palíndromos (com exceção do 0). Dessa forma, para 10^{12}, temos $d = 12$ e logo existem $2 \cdot 10^6 - 1$ números palíndromos menores que 10^{12}. Portanto, para resolver o problema, para cada palíndromo p construído, basta verificar se $n - p$ é não-negativo e palíndromo ou se $n - p$ existe numa lista de palíndromos pre-processados.