# Tutorial: Radares

Vinícius Ruela Pereira Borges

O problema pode ser resolvido utilizando a estrutura de dados Delta Encoding para realizar a leitura das aferições para todos os trechos delimitados pelo posicionamento dos radares em tempo O($Q$) e depois para determinar o trecho com a maior quantidade de infrações, que pode ser feito em O($N$).

Por isso, o problema pode ser resolvido em um custo computacional $O(N + Q)$.

# Tutorial: Presente de Dia das Mães

Daniel Saad Nogueira Nunes

Este problema pode ser resolvido via ordenação e busca binária.

1. Primeiramente ordena-se o vetor de caixas $V$ pela quantidade de chocolates contida em cada uma.
2. Em seguida, realiza-se a busca binária sobre o intervalo $[l, r]$, em que $l = 0$ e $r = V[N-1]$.
3. Enquanto $l \leq r$:
   (a) A cada iteração da busca binária, calcula-se a posição $mid = (l + r)/2$.
      i. Caso seja possível distribuir $mid$ chocolates a cada mãe faça $l \leftarrow mid + 1$.
      ii. Caso contrário, faça $r \leftarrow m - 1$.
4. Imprima $l - 1$ como resposta.

Para testar se é possível distribuir $mid$ chocolates a cada mãe, basta pegar a divisão inteira de cada elemento $V[i]$ do vetor por $mid$ e subtrair do total de mães. Se o total de mães ficar menor ou igual a 0, então é possível. Só se deve tomar cuidado no caso em que $mid = 0$ para evitar uma divisão por zero. Quando $mid = 0$ é claro que é possível distribuir 0 chocolates entre todas as mães (coitadas!).

# Tutorial: Boca de Urna

Edson Alves da Costa Junior

Este problema pode ser resolvido de duas maneiras distintas.

A primeira delas é montar um grafo não-direcionado, onde os vértices são os eleitores e as arestas as relações de voto comum. Após a montagem do grafo, basta iniciar uma busca em largura ou profundidade para cada voto conhecido $e_i$, marcando os eleitores que se encontram no mesmo componente conectado. Ao final deste processamento, para cada eleitor ainda não identificado, é preciso fazer uma nova busca, incrementando a resposta em uma unidade e marcado os eleitores que se encontram neste componente conectado. Esta solução tem complexidade $O(N + Q)$.

Outra abordagem é utilizar a estrutura *Union-Find Disjoint Set* (UFDS) para manter o registro dos componentes conectados, e unir dois componentes a cada nova relação. Uma vez processadas as relações, basta identificar o número de componentes distintos e quais deles tem ao menos um voto conhecido. A resposta será o número de componentes sem voto conhecido. Esta solução tem complexidade semelhante à da solução anterior, porém um fator multiplicativo quase constante (inversa da função de Ackerman) devido às operações da UFDS.

# Tutorial: Lista de Exercícios

Edson Alves da Costa Junior

Os casos onde $|D| \leq 4$ podem ser construídos manualmente.

Para $|D| \geq 5$ é possível usar a seguinte estratégia: Se $D > 0$, faça $x = \lfloor (D+2)/2 \rfloor, y = D - x$. Temos que $x \neq y$, $|x|, |y| > 1$ e que $x + y = D$. Assim, a matrix

$$A = \begin{bmatrix} x & -1 \\ y & 1 \end{bmatrix}$$

tem determinante igual a $D$ e coeficientes distintos.

Se $D < 0$, faça $x = \lfloor (D-2)/2 \rfloor, y = D - x$.

# Tutorial: Construindo Estradas

Pedro Henrique Lima Ferreira e José Marcos da Silva Leite

Vamos reescrever o problema para ficar mais claro o que está sendo pedido.

Dado um vetor $V$, com $N$ elementos, onde cada elemento é um par de inteiros, queremos reordenar este vetor de modo a minimizar a seguinte fórmula:

$$\sum_{i=1}^{N-1} min(V[i].first + V[i+1].first, V[i].second + V[i+1].second)$$

A primeira observação é que todos os elementos contribuem para o resultado final duas vezes, com exceção das pontas (os elementos que ficarão na primeira e última posição) que contribuem apenas 1 vez cada. Isso significa que um elemento $i$ que não é ponta contribuirá na resposta final com $2*V[i].first$ ou $2*V[i].second$ ou $V[i].first + V[i].second$.

A segunda observação importante é que se um elemento tem $V[i].first \leq V[i].second$ e não é uma das pontas, idealmente, vamos querer que ele contribua na resposta com $2 * V[i].first$. De maneira análoga, se $V[i].second \leq V[i].first$, gostaríamos que o elemento contribuísse com $2 * V[i].second$. Isso, obviamente, não é possível sempre, pois pode ocorrer de ter vários elementos do tipo $V[i].first \leq V[i].second$ e do tipo $V[j].second \leq V[j].first$ em $V$. Logo, pode ocorrer de precisarmos usar algum elemento $k$ para fazer a transição de só usar $2 * V[i].first$ para $2 * V[j].second$ $(i < k < j)$. Este elemento $k$, que chamaremos de elemento de transição, contribui $V[k].first + V[k].second$ para a resposta final.

A observação final é que é possível reordenar o vetor $V$, de modo a minimizar o somatório do problema, utilizando no máximo 2 elementos de transição.

Para resolver o problema então, precisamos verificar 3 casos:

- caso onde o vetor $V$ tem 0 elementos de transição:

  Neste caso, ou todos os elementos de $V$ usarão seus first, ou usarão seus second. Para checar isto, basta apenas somar todos os first, multiplicar a soma por 2 e subtrair da soma o maior e o segundo maior first uma única vez (pois como dito antes, todos os elementos contribuirão 2 vezes na resposta, com exceção das pontas). Este mesmo check também deve ser feito para os second.

- caso onde o vetor $V$ tem 1 elementos de transição:

  Primeiramente, vamos fixar as pontas ($p1$ e $p2$) do nosso vetor, assumindo que vamos começar utilizando o first da ponta inicial e que acabaremos no second da ponta final. Agora, vamos analisar os elementos restantes. Todos eles contribuirão duas vezes para a resposta, mais especificamente com $2 * min(V[i].first, V[i].second)$, com exceção do elemento de transição $k$, que irá contribuir com $V[k].first + V[k].second$. Para escolhê-lo, basta apenas pegarmos o elemento $k$ onde $k$ não é uma ponta e $abs(V[k].first - V[k].second)$ é mínimo. A resposta para este caso será:

  $V[p1].first + V[p2].first -$abs(V[k].first - V[k].second)$+ \sum_{i=1, i \neq p1, i \neq p2}^{N} 2 * min(V[i].first, V[i].second)$

- caso onde o vetor $V$ tem 2 elementos de transição:

  Este caso é bem semelhante ao caso anterior. Fixamos uma ponta inicial $p1$ e uma final $p2$. Aqui temos 2 pequenos casos que devemos tratar. O caso em que começamos com o first da ponta inicial e acabamos no first da ponta final e, o caso em que começamos com o second de uma ponta e acabamos no second da outra. Teremos dois elementos de transição $k1$ e $k2$. De maneira semelhante à anterior, pegamos os 2 elementos que minimizam $abs(elem.first - elem.second)$ e que não são pontas. A resposta aqui será:

$V[p1].first + V[p2].first -$ abs(V[k1].first - V[k1].second) $-$ abs(V[k2].first - V[k2].second) $+ \sum_{i=1, i \neq p1, i \neq p2}^{N} 2 * min(V[i].first, V[i].second)$

Esta solução é $O(N^2)$, pois para cada par de pontas, verificamos os 3 casos acima. Para melhorar essa solução podemos ver o quanto a soma é modificada para cada elemento que escolhemos como ponta.

Quando um elemento não é ponta, ele contribuiu para soma em 2*min(v[i].first, v[i].second) ou em v[i].first + v[i].second. Se um elemento que contribui com 2*min(v[i].first, v[i].second) passa a contribuir com v[i].first para soma, a variação é de v[i].first - 2*min(v[i].first, v[i].second). Se um elemento que contribui com v[i].first + v[i].second passa a contribuir com v[i].second, a variação é de v[i].second - (v[i].first + v[i].second). Os outros casos são análogos.

Queremos minimizar então sempre escolhemos os elementos com menor contribuição para as pontas. Podemos montar um conjunto $S$ com os 4 melhores elementos de cada uma das formas de calcular variação. Assim $|S| \leq 16$.

Assim podemos deixar de testar todos os $O(N^2)$ pares de pontas possível para só testar cada par de pontas de $S$ que é no máximo $16^2$.

Há também uma outra solução, utilizando programação dinâmica, com os seguintes estados:

f(idx_atual, quantidade_elementos_transição, se_ponta_inicial_usada, se_ponta_final_usada, se_ponta_inicial_usa_first, se_ponta_final_usa_first)

- **idx_atual**: qual elemento de V está sendo analisado agora ($\leq N$);
- **quantidade_elementos_transição**: quantos elementos de transição podem ser usados ($\leq 2$);
- **se_ponta_inicial_usada**: flag que indica se algum elemento já foi escolhido como ponta inicial ($\leq 1$);
- **se_ponta_final_usada**: flag que indica se algum elemento já foi escolhido como ponta final ($\leq 1$);
- **se_ponta_inicial_usa_first**: flag que indica se ponta inicial usa first ($\leq 2$);
- **se_ponta_final_usa_first**: flag que indica se ponta final usa first ($\leq 2$);

Para fazer as transições, basta apenas analisar as flags e tratar cada caso. Como na solução anterior, resposta é o mínimo dos 3 casos (considerando 0, 1 e 2 elementos de transição).

Esta solução roda em $O(N)$.

**Por que sempre é possível reordenar o vetor V, de modo a minimizar o somatório, com no máximo 2 elementos de transição?**

Ideia da prova: Vamos supor que temos um vetor V com exatamente 3 elementos de transição (k1, k2 e k3) desta forma:

- **v[1] -¿ first**
- ... (grupoA que usa 2*first)
- **v[k1] -¿ first second** -¿ usa first + second
- ... (grupoB que usa 2*second)
- **v[k2] -¿ second first** -¿ usa second + first
- ... (grupoC que usa 2*first)
- **v[k3] -¿ first second** -¿ usa first + second
- ... (grupoD que usa 2*second)

- **v[n] -¿ second**

Vamos mostrar que tem como obter uma reordenação de V, utilizando apenas 1 elemento de transição, com um resultado menor ou igual ao anterior.
Fixando **k1** como elemento de transição (vamos realocar **k2** e **k3**), as pontas continuam se comportando do mesmo modo (**v[1]** utiliza o first e **v[n]** utiliza o second). Além disso, antes de realocar **k2** e **k3**, podemos observar que é possível realocar os elementos do grupoC para o grupoA, e os elementos do grupoD para o grupoB sem alterar o resultado final.

Agora, para realocar **k2** é só fazer o seguinte:

se **v[k2].first ¡ v[k2].second**, é mais vantajoso a gente utilizar 2*v[k2].first, logo ele vai pro grupoA, senão ele vai pro grupoB. podemos realocar **k3** da mesma maneira. Deste modo, nossa soma inicial é diminuída em $(abs(V[k2].first - V[k2].second) + abs(V[k3].first - V[k3].second))$.

# Tutorial: Espetinho do Barbosinha

Daniel Saad Nogueira Nunes

Este problema pode ser resolvido via uma abordagem gulosa.

Utilizamos dois vetores, $S$ e $F$ que armazenam respectivamente os tempos (em segundos) de início e fim das estadias de cada cliente.

Após isso, cada um dos vetores é ordenado em ordem crescente.

Em seguida, o seguinte procedimento é aplicado:

```
max_overlap = 0;
overlap = 0;
while(i<n && j<n){
   if(S[i]<=F[j]){
     i++;
     overlap++;
     max_overlap = max(max_overlap,overlap);
   }
   else{
     overlap--;
     j++;
   }
}
```

A justificativa é a seguinte. Suponha que sabemos o tamanho da sobreposição atual e o tamanho da sobreposição máxima. Caso $S[i] <= F[j]$, então o i-ésimo ponto de início está se sobrepondo com o j-ésimo ponto do fim e o tamanho da sobreposição atual deve ser incrementada, caso ela supere o tamanho da sobreposição global, a última deve ser ajustada. Caso $S[i] > F[j]$, então não há sobreposição entre o intervalo com início em $S[i]$ e o intervalo com fim em $F[j]$, portanto o número de sobreposições atual é decrementado. Esta solução possui complexidade $\Theta(N \lg N)$ pois é dominada pelos passos de ordenação.

Outra solução, baseada em soma de prefixos, consiste em inicializar um vetor $V[0, T+1]$, com zeros, em que $T$ representa o maior tempo de término dentre todos os clientes. Para cada tempo de início $b$ e fim $e$ (em segundos) informado pelos clientes, fazemos $V[b]++$ e $V[e]--$. Em seguida, computa-se a soma de prefixos para cada $V[i]$. A resposta será o maior valor de $V[i]$ dentre todos os $i$. A complexidade desta solução é $\Theta(N+T)$, e como existe a restrição de que $T \leq 24 \cdot 3600 - 1$, obtém-se um algoritmo rápido.

# Tutorial: Propagação de Worms

Daniel Saad Nogueira Nunes

Esse problema corresponde a um problema similar ao do conjunto dominante mínimo de vértices de um grafo $G = (V, E)$.

Estamos interessados em um conjunto $S$ mínimo tal que, para $v \in V$:

- $v \in S$ ou
- existe uma aresta $(u, v)$ tal que $u \in S$.

Ou seja, estamos interessados no conjunto $S$ que é a união de um conjunto dominante mínimo com o conjunto dos vértices isolados.

Para calcular o conjunto $S$ de menor cardinalidade, pode-se utilizar uma abordagem de busca completa:

1. Gere todos os subconjuntos de vértices.
2. Para cada subconjunto $S \subseteq V$, verifique se atende as restrições do problema. Para cada $v \in S$, faça:
   (a) Marque $v$ e os nós adjacentes.
   (b) Se todos os vértices de $V$ foram marcados a partir de $S$ e o tamanho de $S$ é menor do que uma solução prévia, armazene $|S|$ e os hospedeiros que fazem parte de $|S|$.
3. Imprima $|S|$ e os hospedeiros que fazem parte de $S$.

Para gerar todos os subcojuntos de $V$ eficientemente podem ser utilizadas técnicas de manipulação de bits. Se $V$ possui $n$ elementos, uma forma de gerar todos os seus subconjuntos é:

```
for (int i = 0; i < 1 << n; i++) {
    check(i);
}
```

Aqui, a representação binária de $i$ corresponde a um subconjunto $S \subseteq V$. Assim, um vértice $v$ está neste subconjunto se e somente se $\texttt{i}\&(\texttt{1} << \texttt{v}) \neq 0$.

A complexidade total da solução é $\Theta((|V| + |E|) \cdot 2^{|V|})$.

# Tutorial: Quantos Caminhos?

Lucas Vasconcelos Mattioli

Em breve.

# Tutorial: Sequência Binomial Central

Edson Alves da Costa Junior

A tentativa de construir todos os termos da sequência leva ao TLE ou ao *overflow*, pois $\binom{200000}{100000}$ não cabe em um tipo primitivo da linguagem C++ e o tempo necessário para calcular todos os termos em Python leva ao TLE.

Para saber se um número é ou não múltiplo de um primo $p$, é preciso determinar quantas vezes o $p$ aparece na fatoração do numerador $i!$ e na fatoração do denominador $\lfloor i/2 \rfloor ! \lfloor (i+1)/2 \rfloor !$.

Se no numerador houver mais incidências de $p$ do que o numerador, o número binomial será múltiplo de $p$.

Para determinar o número de ocorrências de $p$ no fatorial de $n$, basta usar a função $E_p$:

$$E_p(n) = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \ldots + \left\lfloor \frac{n}{p^k} \right\rfloor,$$

onde $\left\lfloor \frac{n}{p^{k+1}} \right\rfloor = 0$.

Esta função tem complexidade $O(\log N)$, de modo que a solução tem complexidade $O(N \log N)$.

# Tutorial: Fabricação de Caixas

Edson Alves da Costa Junior

Observe que $V = b^2h$. O menor valor possível para $h$ é 1, de modo que $b \leq \sqrt{V}$. Portanto, o valor de $b$ pertence ao intervalo $[1, 10^7]$. Assim, basta fazer uma busca completa neste intervalo, observando que $b^2$ tem que ser um divisor de $V$.

Logo, a solução tem complexidade $O(\sqrt{V})$.

# Tutorial: Máquina de Refrigerante

Daniel Saad Nogueira Nunes

Este problema pode ser resolvido com uma modelagem via cadeias de Markov.

Tome um vetor $V^k[0, n-1]$ de modo que $V[i]$ contém a probabilidade de atingir o estado $i$ com $k$ moedas. $V^0[i] = 1$ se e somente se $i = U$, caso contrário, $V^0[i] = 0$.

Dado $V^k[0, n-1]$ é possível calcular $V^{k+1}[0, n-1]$ da seguinte forma:

$$V^{k+1}[i] = \begin{cases} \sum\limits_{x \in \{0,\ldots,n-1\}} V^k[x] \cdot M[x][i], & k = 0 \\ \sum\limits_{x \in \{0,\ldots,n-1\} \setminus \{F\}} V^k[x] \cdot M[x][i], & k > 0 \end{cases}$$

Ou seja, a probabilidade de atingir o estado $i$ é a soma das probabilidades de, partindo de um estado $x$, chegar no estado $i$ utilizando a matriz de transição entre estados.

É importante ressaltar que, como queremos saber a probabilidade de Epaminondas não receber o seu refrigerate favorito, excluímos da conta o estado $F$ quando $k > 0$.

Ao final, basta realizar a soma dos elementos do vetor $V^M$, excluindo $V^M[F]$, para calcular a probabilidade pedida.

O custo total desse algoritmo é $\Theta(N^2 \cdot M)$.

# Tutorial: Almoço em Manhattan

José Marcos da Silva Leite

Temos que calcular o custo total de almoçar em cada restaurante e depois informar qual o menor destes custos. O custo de almoçar num restaurante é (preço da refeição) + 2 * (distancia de Chico ate o restaurante). Só é preciso tomar cuidado por caso haja empate em custos, precisamos informar o restaurante de menor índice.