A - Exploding Fireants

Problema

- **Input:** Número de casos de teste. Para cada teste, diâmetro da granada e linha com formigas (F) e espaços vazios (.)
- **Output:** Número de granadas para cada caso de teste.

Limites

- $1 \leq N \leq 1000$ Número de linhas
- $1 \leq D \leq C$ Diâmetro da explosão da granada
- $1 \leq C \leq 1000$ Número de caracteres de cada linha

Classificação

- **Categorias:** Genérico
- **Dificuldade:** Fácil

A - Exploding Fireants

Sample Input

```
3
4
....FF.FFF.F.....FFF....
6
....FFF.FF....F....F....
3
....FFF.F.F....F....F.F...
```

Sample Output

```
3
2
4
```

B - Safest Routes

Problema

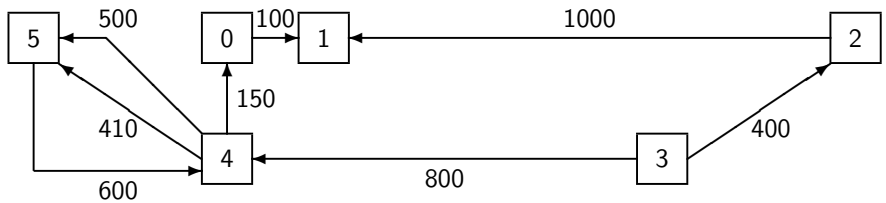
Input: P praças, ligadas por R ruas de sentido único, cada uma com um comprimento c. T testes, cada um define 2 praças (o_i e d_i).
Output: A menor distância que é possível percorrer em sentido proibido para ir de o_i para d_i.

Limites

$2 \leq P \leq 30\,000 \quad 1 \leq R \leq 150\,000$
$1 \leq c \leq 10\,000 \quad 1 \leq T \leq 10$

Classificação

Categorias: Grafos, Caminhos mais curtos, Algoritmo de Dijkstra
Dificuldade: Médio
Autor: Margarida Mamede



B - Safest Routes

- **Solução**
  - Construir um grafo (V, E), orientado e pesado.
    Quais são os vértices, os arcos e os pesos dos arcos?

B - Safest Routes

- **Solução**
  - Construir um grafo (V, E), orientado e pesado.
    Quais são os vértices, os arcos e os pesos dos arcos?
  - Para cada teste, calcular o comprimento dos caminhos mais curtos do vértice origem para o vértice destino.

B - Safest Routes

- **Solução**
  - Construir um grafo (V, E), orientado e pesado.
    Quais são os vértices, os arcos e os pesos dos arcos?
  - Para cada teste, calcular o comprimento dos caminhos mais curtos do vértice origem para o vértice destino.
- **Construção do grafo**

B - Safest Routes

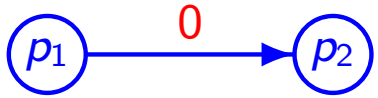
- **Solução**
  - Construir um grafo (V, E), orientado e pesado.
    Quais são os vértices, os arcos e os pesos dos arcos?
  - Para cada teste, calcular o comprimento dos caminhos mais curtos do vértice origem para o vértice destino.
- **Construção do grafo**
  - 1 vértice por cada praça p

B - Safest Routes

- **Solução**
  - Construir um grafo (V, E), orientado e pesado.
    Quais são os vértices, os arcos e os pesos dos arcos?
  - Para cada teste, calcular o comprimento dos caminhos mais curtos do vértice origem para o vértice destino.
- **Construção do grafo**
  - 1 vértice por cada praça p
  - 2 arcos por cada rua de p_1 para p_2 com comprimento c

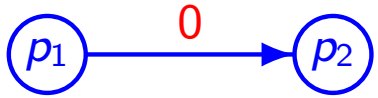
B - Safest Routes

- **Solução**
  - Construir um grafo (V, E), orientado e pesado.
    Quais são os vértices, os arcos e os pesos dos arcos?
  - Para cada teste, calcular o comprimento dos caminhos mais curtos do vértice origem para o vértice destino.
- **Construção do grafo**
  - 1 vértice por cada praça p
  - 2 arcos por cada rua de p_1 para p_2 com comprimento c
    - A distância é zero se a rua é percorrida no sentido permitido



B - Safest Routes

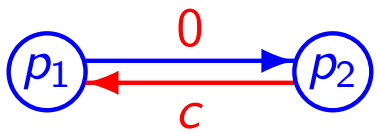
- **Solução**
  - Construir um grafo (V, E), orientado e pesado.
    Quais são os vértices, os arcos e os pesos dos arcos?
  - Para cada teste, calcular o comprimento dos caminhos mais curtos do vértice origem para o vértice destino.
- **Construção do grafo**
  - 1 vértice por cada praça p
  - 2 arcos por cada rua de p_1 para p_2 com comprimento c
    - A distância é zero se a rua é percorrida no sentido permitido



    - A distância é c se a rua é percorrida no sentido proibido

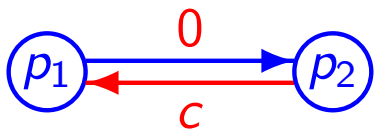
B - Safest Routes

- **Características do grafo** (V, E)
  - 1 vértice por praça; 2 arcos por cada rua (p_1, p_2) de comprimento c



B - Safest Routes

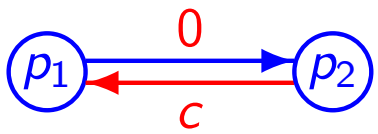
- **Características do grafo** (V, E)
  - 1 vértice por praça; 2 arcos por cada rua (p_1, p_2) de comprimento c



  - $|V| = P \leq 30\,000$ e $|E| = 2R \leq 300\,000$

B - Safest Routes

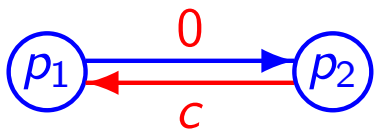
- **Características do grafo** (V, E)
  - 1 vértice por praça; 2 arcos por cada rua (p_1, p_2) de comprimento c



  - $|V| = P \leq 30\,000$ e $|E| = 2R \leq 300\,000$
  - O grafo pode ter arcos paralelos

B - Safest Routes

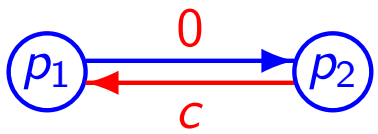
- **Características do grafo** (V, E)
  - 1 vértice por praça; 2 arcos por cada rua (p_1, p_2) de comprimento c



  - $|V| = P \leq 30\,000$ e $|E| = 2R \leq 300\,000$
  - O grafo pode ter arcos paralelos
- **Algoritmos**
  - Para cada teste, executar o algoritmo de Dijkstra (a partir de o_i), implementado com complexidade temporal $\mathcal{O}(|E| \log |V|)$

B - Safest Routes

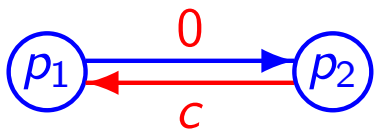
- **Características do grafo** (V, E)
  - 1 vértice por praça; 2 arcos por cada rua (p_1, p_2) de comprimento c



  - $|V| = P \leq 30\,000$ e $|E| = 2R \leq 300\,000$
  - O grafo pode ter arcos paralelos
- **Algoritmos**
  - Para cada teste, executar o algoritmo de Dijkstra (a partir de o_i), implementado com complexidade temporal $\mathcal{O}(|E| \log |V|)$
  - É necessário ter uma ED para descobrir o próximo vértice a tratar, efetuando inserções e remoções com complexidade $\mathcal{O}(\log |V|)$

B - Safest Routes

- **Características do grafo** (V, E)
  - 1 vértice por praça; 2 arcos por cada rua (p_1, p_2) de comprimento c



  - $|V| = P \leq 30\,000$ e $|E| = 2R \leq 300\,000$
  - O grafo pode ter arcos paralelos
- **Algoritmos**
  - Para cada teste, executar o algoritmo de Dijkstra (a partir de o_i), implementado com complexidade temporal $\mathcal{O}(|E| \log |V|)$
  - É necessário ter uma ED para descobrir o próximo vértice a tratar, efetuando inserções e remoções com complexidade $\mathcal{O}(\log |V|)$
  - Complexidade TOTAL: $\mathcal{O}(T \times |E| \log |V|)$

C - Seraphina's Lost Treasure

Problema

- **Input:** Mapa com as linhas desordenadas.
- **Output:** Mapa com as linhas ordenadas.

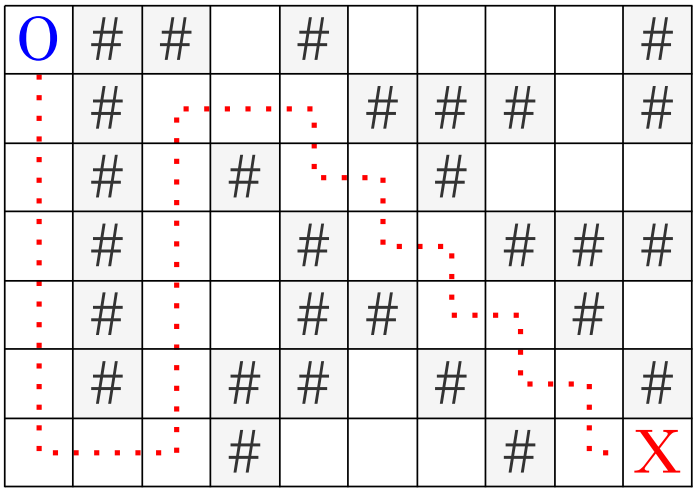
Limites

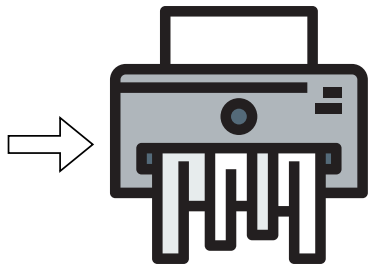
- $2 \leq W \leq 20$
- $2 \leq H \leq 10$

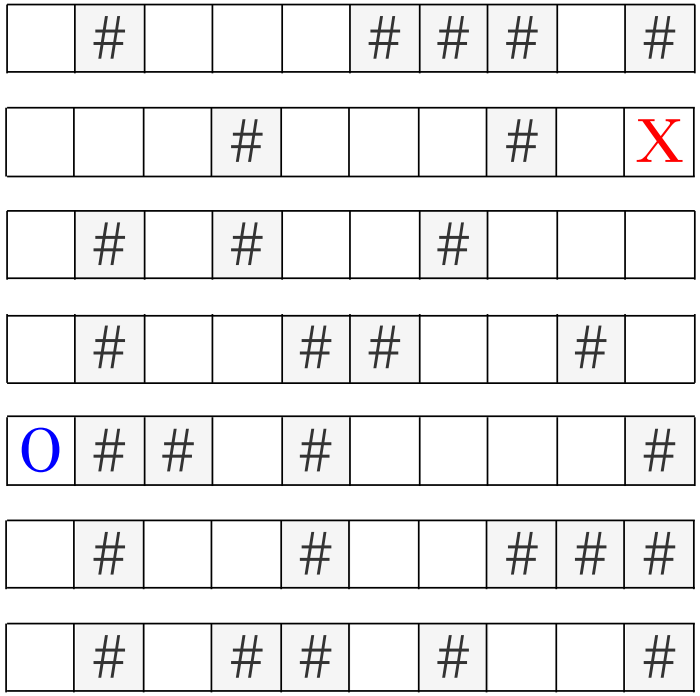
Classificação

- **Categorias:** Permutations, Grafos
- **Dificuldade:** Médio

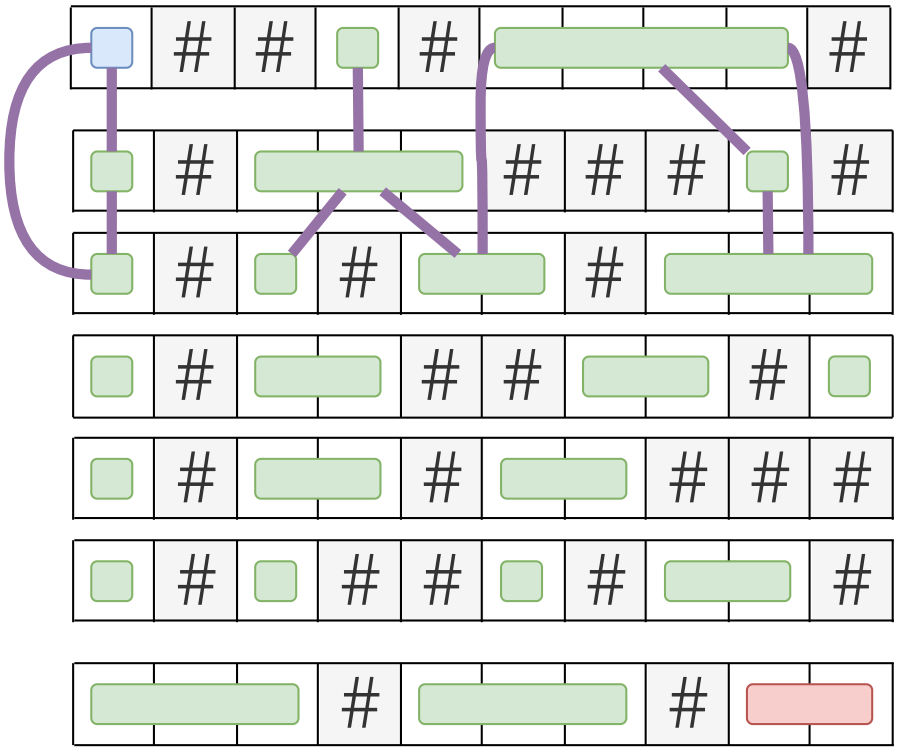
C - Seraphina's Lost Treasure

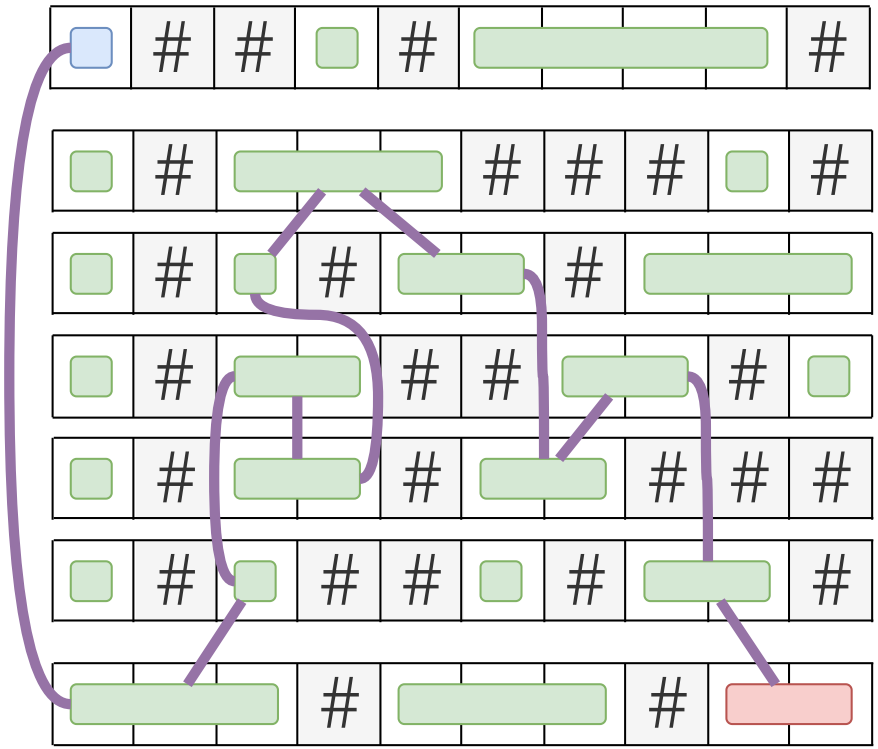






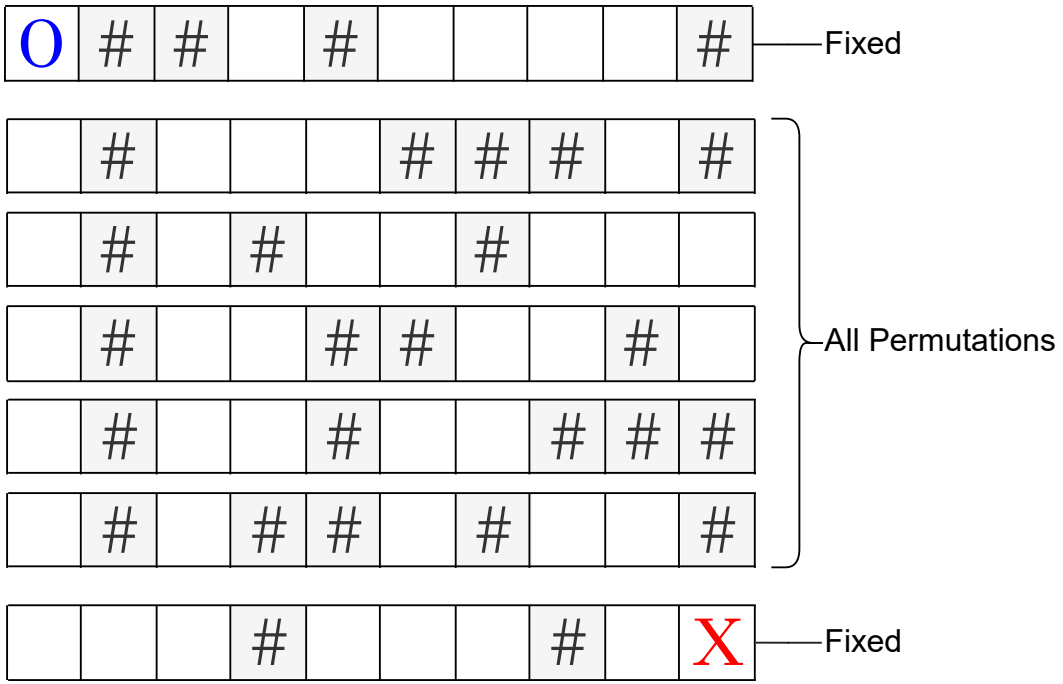
C - Seraphina's Lost Treasure

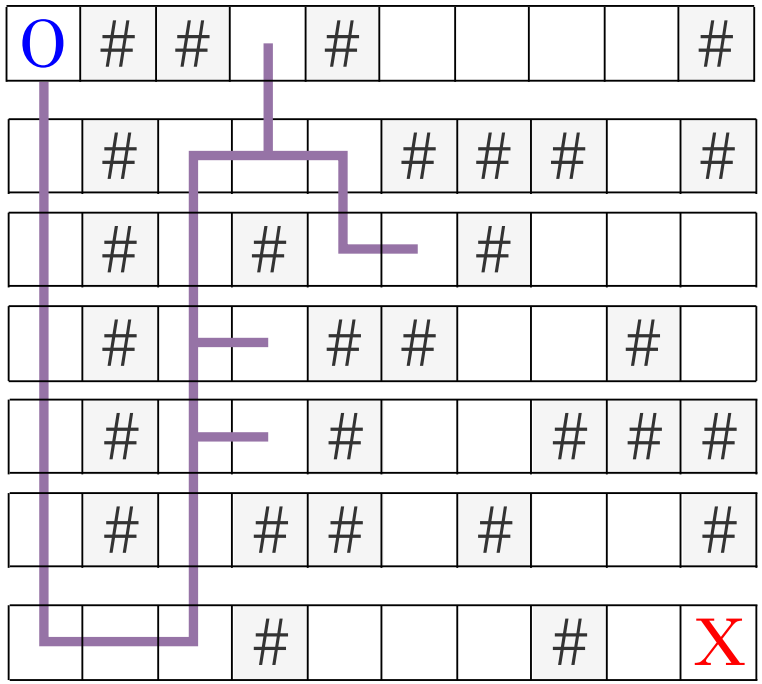




1. Simplificar o grafo juntando nodos em cada tira.
2. Procurar um caminho entre o nodo inicial e o final sem repetir tiras do mapa mas podendo voltar à tira anterior.

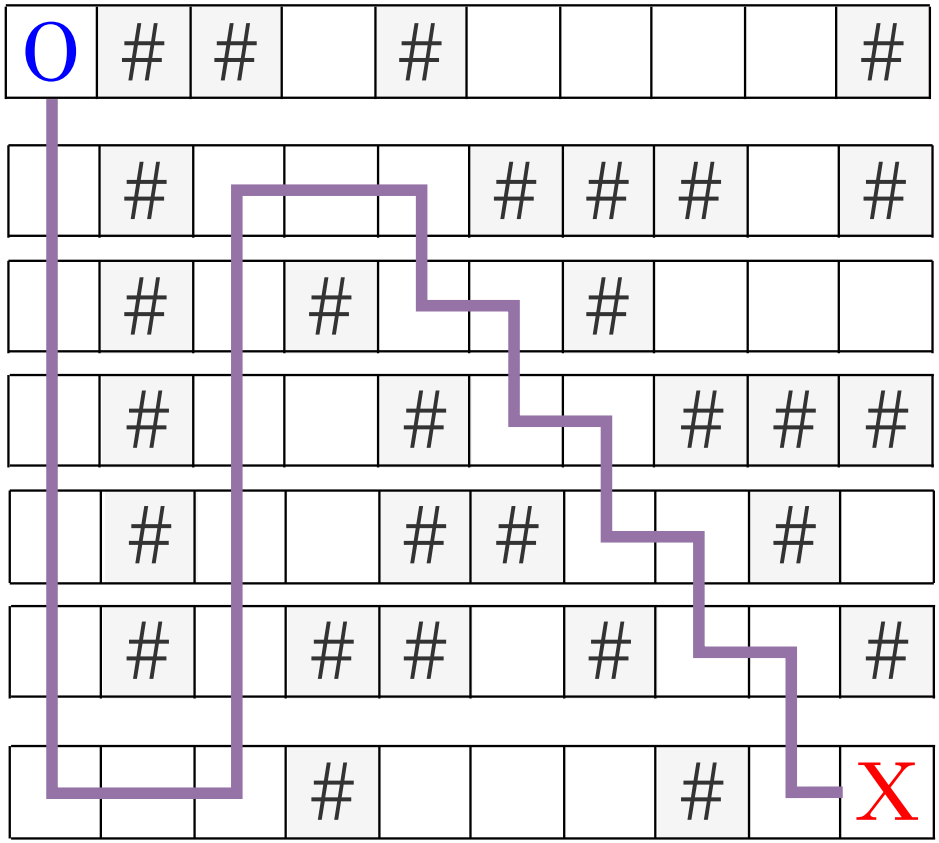
C - Seraphina's Lost Treasure





1. Fixar a primeira e última tira; passar por todas as permutações das tiras centrais.
2. Usar, por exemplo DFS, para verificar se a solução é válida.

C - Seraphina's Lost Treasure



Eventualmente uma das permutações vai ser a solução.

D - This is the Way

Problema

- **Input:** N intervalos $[a, b]$
- **Output:** Quantos números no intervalo têm mais uns (1) que zeros (0) na sua representação em binário

Limites

- $1 \leq N \leq 100$
- $1 \leq a \leq b \leq 10^{18}$

Classificação

- **Categorias:** Programação Dinâmica
- **Dificuldade:** Médio

D - This is the Way

- Enumerar todos os números válidos não é exequível (100×10^{18})

D - This is the Way

- Enumerar todos os números válidos não é exequível (100×10^{18})
- Será que conseguimos escrever a solução em função de **subproblemas iguais mais pequenos**?
- Será que o número de subproblemas pequenos é suficientemente pequeno para **calcular, guardar e reutilizar resultados**?

D - This is the Way

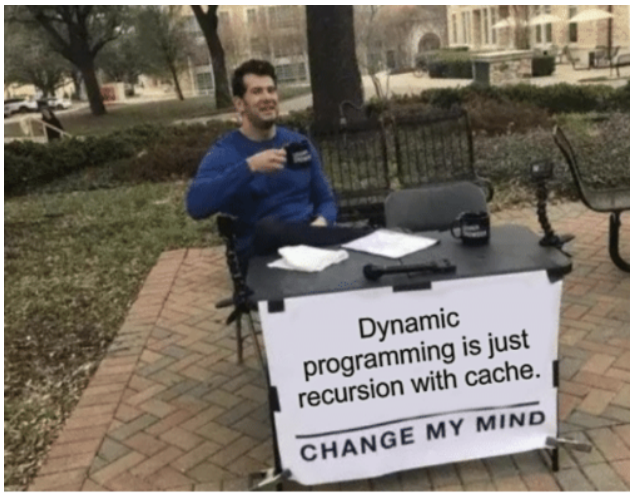
- Enumerar todos os números válidos não é exequível (100×10^{18})
- Será que conseguimos escrever a solução em função de **subproblemas iguais mais pequenos**?
- Será que o número de subproblemas pequenos é suficientemente pequeno para **calcular, guardar e reutilizar resultados**?

Programação Dinâmica

D - This is the Way

- Enumerar todos os números válidos não é exequível (100×10^{18})
- Será que conseguimos escrever a solução em função de **subproblemas iguais mais pequenos**?
- Será que o número de subproblemas pequenos é suficientemente pequeno para **calcular, guardar e reutilizar resultados**?

Programação Dinâmica



It is so much easier to write dp
if you think this way.

D - This is the Way

- Seja *amount(x)* quantidade de números válidos:
  Queremos no fundo **amount(b) - amount(a-1)**

D - This is the Way

- Seja *amount(x)* quantidade de números válidos:
  Queremos no fundo **amount(b) - amount(a-1)**

- Existem muitos **"estados" possíveis** para resolver o problema
  - *4 soluções dos juízes, todas ligeiramente diferente*
- Vou dar um **estado mais geral** do que o necessário
  (para ser enquadrado como *digit dp*)

D - This is the Way

- Seja *amount(x)* quantidade de números válidos:
  Queremos no fundo **amount(b) - amount(a-1)**

- Existem muitos **"estados" possíveis** para resolver o problema
  - *4 soluções dos juízes, todas ligeiramente diferente*
- Vou dar um **estado mais geral** do que o necessário
  (para ser enquadrado como *digit dp*)

- **(pos, diff, hasone, tight)**
  - *pos* - posição onde vamos na representação em binário
  - *diff* - diferença entre quantidade de 1s e 0s
  - *hasone* - já apareceu o primeiro 1?
  - *tight* - só podemos ir até ao número inicial
- Queremos descobrir $amount(num\_digits, 0, false, true)$

D - This is the Way

- $(pos, diff, hasone, tight)$
  - **[caso base]**
    se $pos = -1$ então devolver 1 se $diff > 0$ ou 0 caso contrário

D - This is the Way

- $(pos, diff, hasone, tight)$
  - **[caso base]**
    se $pos = -1$ então devolver 1 se $diff > 0$ ou 0 caso contrário
  - $count = 0$

D - This is the Way

- $(pos, diff, hasone, tight)$
  - **[caso base]**
    se $pos = -1$ então devolver 1 se $diff > 0$ ou 0 caso contrário
  - $count = 0$
  - **[usar dígito 1]**
    se $tight = false$ OU $num[pos] = 1$ então
    $count+ = (pos - 1, diff + 1, true, tight)$

D - This is the Way

- $(pos, diff, hasone, tight)$
  - **[caso base]**
    se $pos = -1$ então devolver 1 se $diff > 0$ ou 0 caso contrário
  - $count = 0$
  - **[usar dígito 1]**
    se $tight = false$ OU $num[pos] = 1$ então
    $count+ = (pos - 1, diff + 1, true, tight)$
  - **[usar dígito 0]**
    - Se $hasone = false$ $count+ = (pos - 1, diff, hasone, false)$
    - Senão se $v[pos] = 1$ $count+ = (pos - 1, diff - 1, hasone, false)$
    - Senão $count+ = (pos - 1, diff - 1, hasone, tight)$

D - This is the Way

- $(pos, diff, hasone, tight)$
  - **[caso base]**
    se $pos = -1$ então devolver 1 se $diff > 0$ ou 0 caso contrário
  - $count = 0$
  - **[usar dígito 1]**
    se $tight = false$ OU $num[pos] = 1$ então
    $count+ = (pos - 1, diff + 1, true, tight)$
  - **[usar dígito 0]**
    - Se $hasone = false$ $count+ = (pos - 1, diff, hasone, false)$
    - Senão se $v[pos] = 1$ $count+ = (pos - 1, diff - 1, hasone, false)$
    - Senão $count+ = (pos - 1, diff - 1, hasone, tight)$

E - Shipping Containers

Problem

- For each test case find the maximum profit that can be obtained shipping containers with the given deadline.
- Ships can only take a certain number of containers within this deadline. Each container can only be sent once but there may be several ships that can transport it.

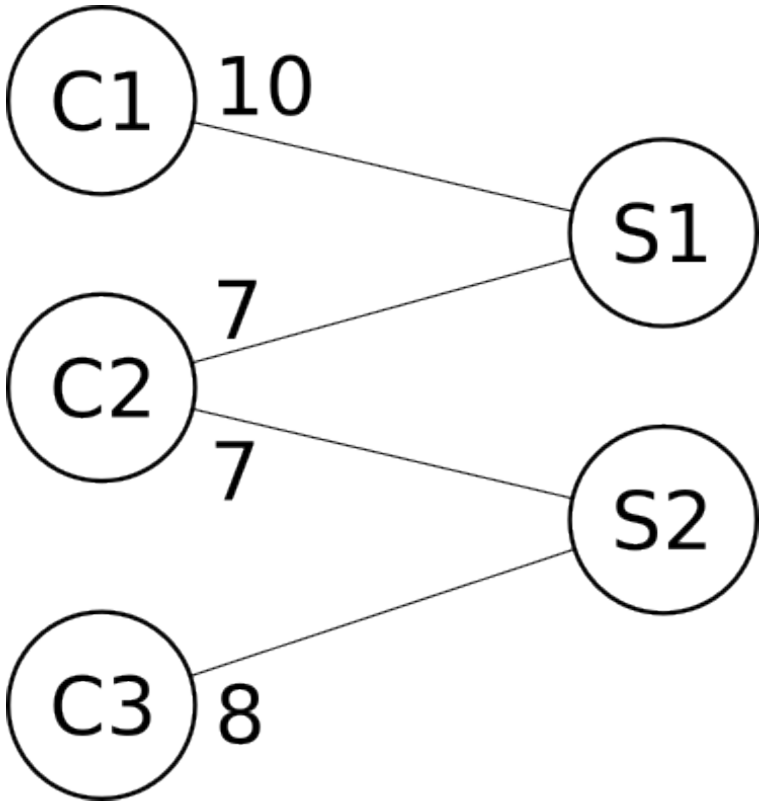
Limits

$1 \leq D \leq 1000$	Deadline
$1 \leq N \leq 20$	Number of ships
$1 \leq C \leq 2500$	Number of containers
$C \leq M \leq \min(N \times C, 2 \times 10^4)$	Number of ship/container pairs
$1 \leq q_i \leq 10$	Ship's capacity
$1 \leq t_i \leq D$	Ship's travel time
$1 \leq p_j \leq 100$	Container's profit

E - Shipping Containers

Solving

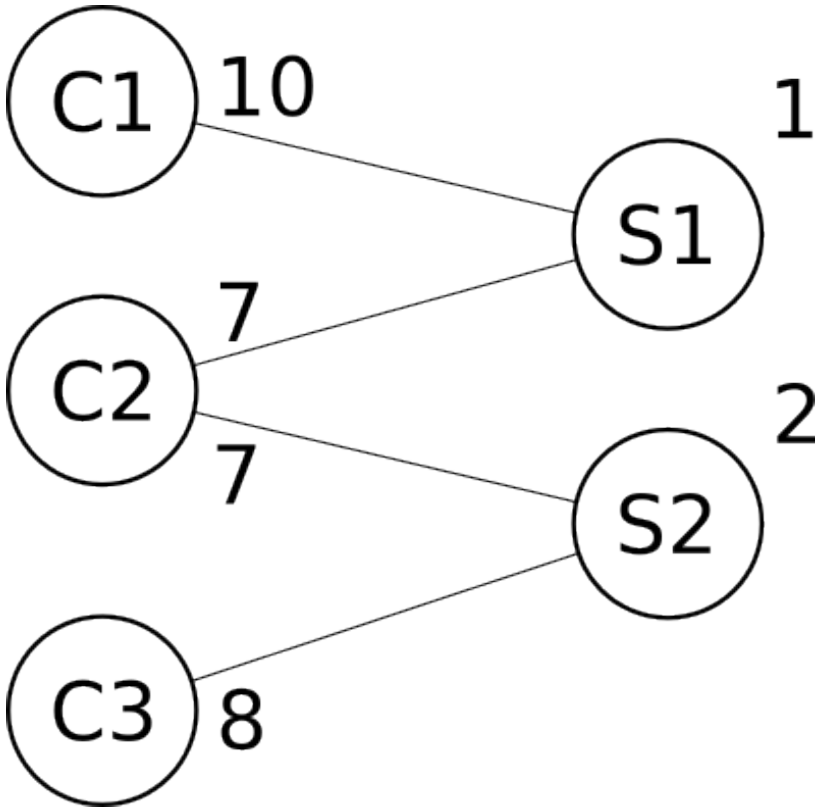
We can think about the problem has a bipartite graph. Consider the case where there are three containers with profits 10, 7, and 8 respectively, two ships, and the following connections between them.



E - Shipping Containers

Solving

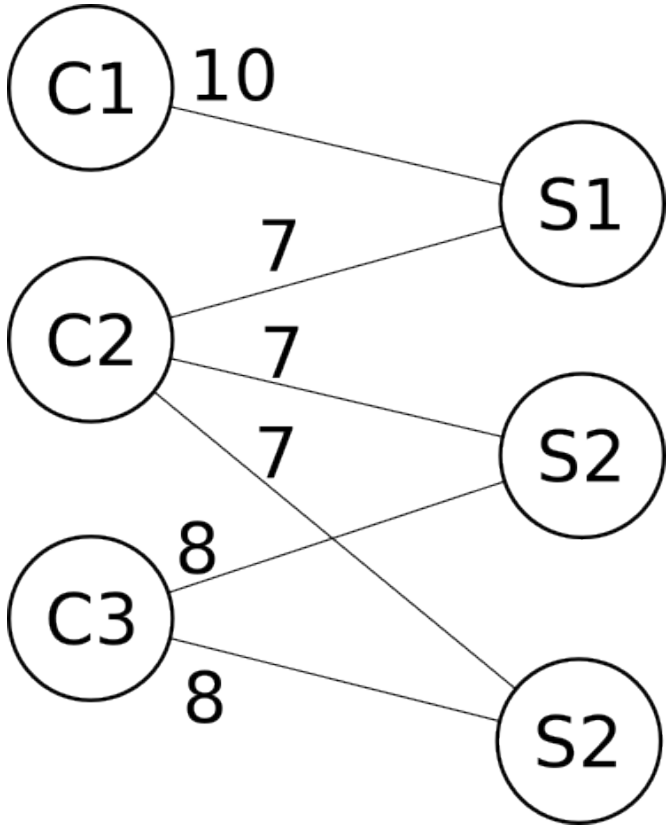
Given the traveling time of each ship we can pre-compute how many containers it can take before the deadline.



E - Shipping Containers

Solving

If we duplicate the nodes with capacity greater than 1, this is a (maximizing) linear assignment problem.



E - Shipping Containers

Solving - Hungarian Algorithm

The Hungarian algorithm can be used to solve the (minimizing, but we can negate the costs) linear assignment problem.

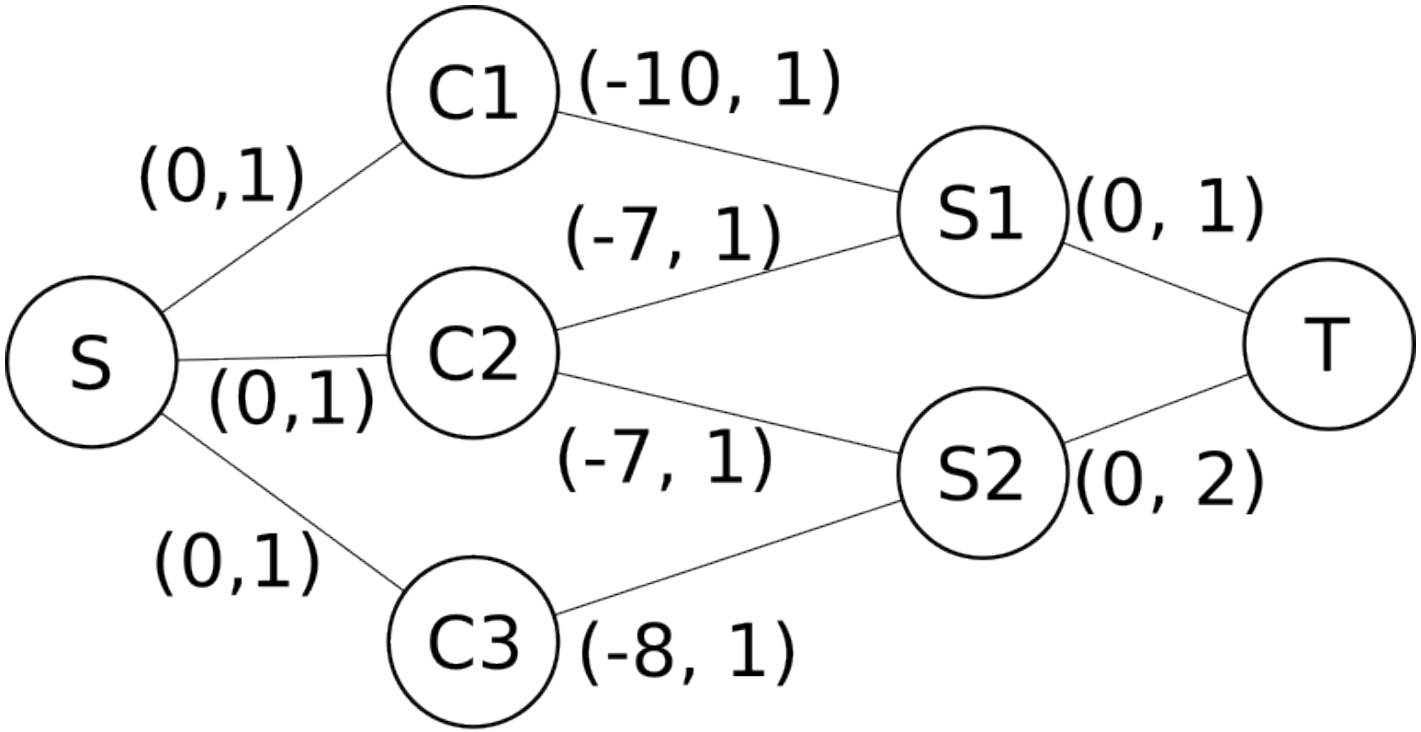
Complexity

A typical implementation would have runtime complexity $\mathcal{O}(C^2N')$ assuming $C \leq N'$ where N' is the number of ships after "splitting" the nodes. Since N' can be quite large ($C \times N$ at most if we are careful), this was expected to give TLE.

E - Shipping Containers

Solving - Min-cost Max-flow

The minimizing assignment problem can also be solved with a min-cost max-flow algorithm by transforming our graph a little bit. This also avoids "splitting" the ship nodes.



E - Shipping Containers

Solving - Min-cost Max-flow

The Min-cost Max-flow can be solved with successive shortest-path, which is a defined implementation of the Ford-Fulkerson algorithm. Essentially, on each iteration we compute the shortest-path (relative to the costs) that adds flow the graph. We stop once no more flow can be added.

Complexity (Bellman-Ford)

If we use the Bellman-Ford algorithm to find the shortest-path we get a complexity of $\mathcal{O}(C^2N^2)$. This would be enough to get Accepted.

Complexity (Dijkstra's)

If we use Dijkstra's shortest-path algorithm we can get a complexity of $\mathcal{O}(C^2N)$. However, since there can be negative weights, we need to keep an up to date list of potential values for each node such that we can have an equivalent graph with only positive weights.

F - Stickers

Problema

Input: N cromos ordenados, possivelmente repetidos, de uma coleção com C cromos diferentes; P perguntas (identificadores) diferentes.
Output: Quantos exemplares de cada identificador existem?

Limites

$1 \leq N \leq 500\,000$
$1 \leq C \leq 100\,000$
$1 \leq P \leq 100\,000$

Classificação

Categoria: Estruturas de Dados
Dificuldade: Simples
Autor: João Neves (UBI)

F - Stickers

Algoritmo (mais frequente)

Construir um histograma (identificador, #exemplares)

- Vetor de pares / Matriz com 2 colunas **com** os IDs que existem
  Pesquisa binária

F - Stickers

Algoritmo (mais frequente)

Construir um histograma (identificador, #exemplares)

- Vetor de pares / Matriz com 2 colunas **com** os IDs que existem
  Pesquisa binária

- Mapa (int, int)
  (tabela de dispersão ou árvore binária de pesquisa)

F - Stickers

Algoritmo (mais frequente)

Construir um histograma (identificador, #exemplares)

- Vetor de pares / Matriz com 2 colunas **com** os IDs que existem
  Pesquisa binária
- Mapa (int, int)
  (tabela de dispersão ou árvore binária de pesquisa)
- Vetor (int[100000])

G - Snake Oil Salesperson

Problema

Encontrar a área do(s) maior(es) rectângulo(s)

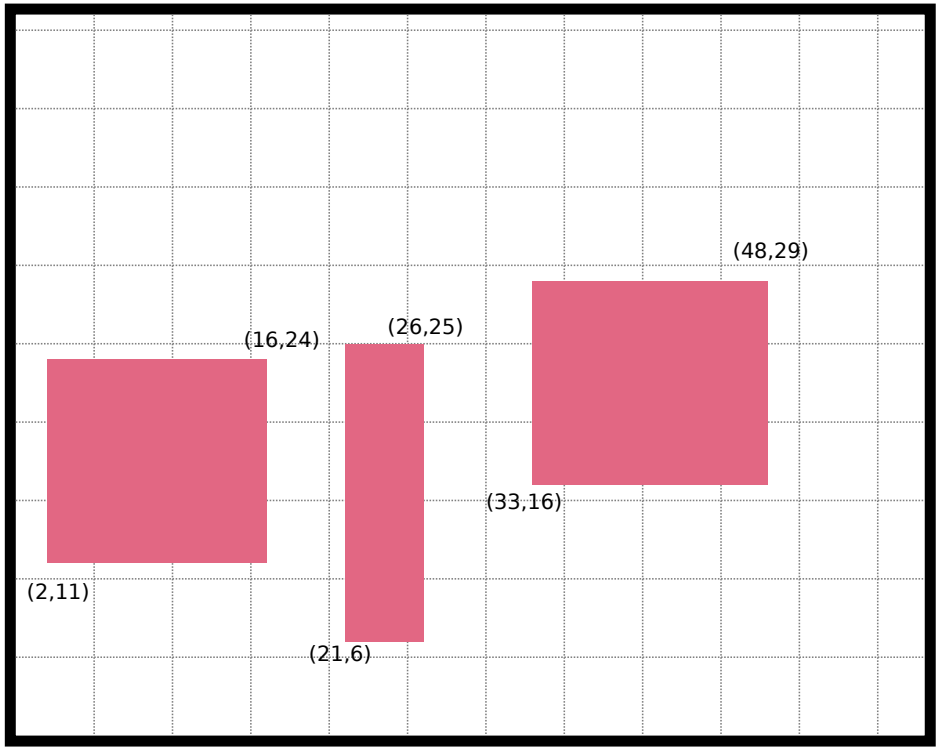
Restrições

$10 \leq W \leq 2 \cdot 10^5$	Comprimento da paisagem
$10 \leq H \leq 2 \cdot 10^5$	Altura da paisagem
$1 \leq N \leq 100$	Número de obstáculos rectangulares
$0 \leq X_1^r < X_2^r \leq W$	Coordenadas horizontais dos cantos dos rectângulos
$0 \leq Y_1^r < Y_2^r \leq H$	Coordenadas verticais dos cantos dos rectângulos

Classificação

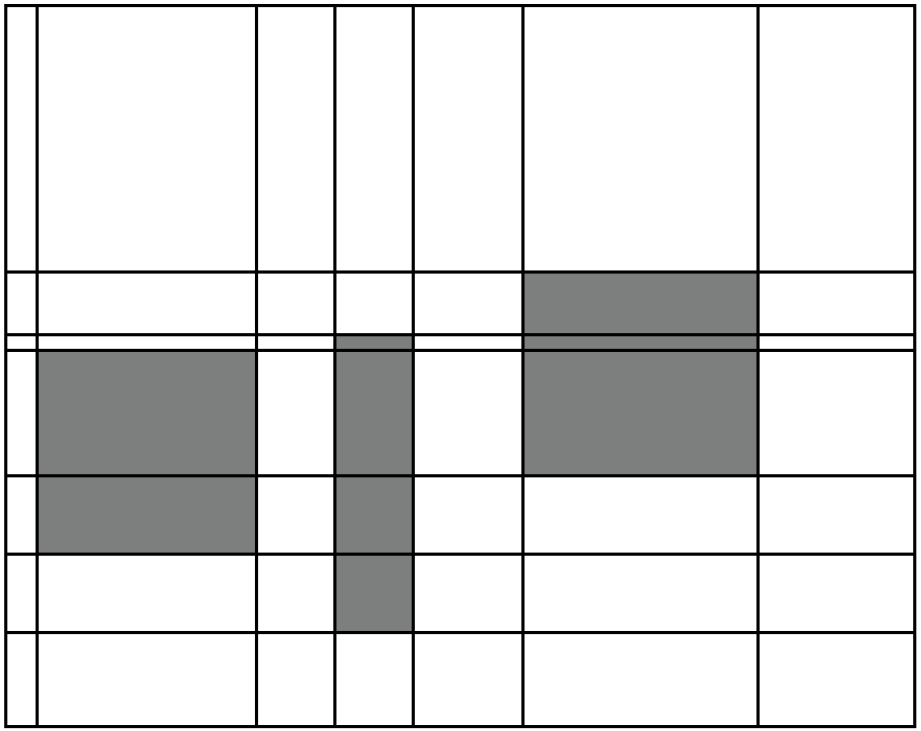
- **Categorias:** Geometria
- **Dificuldade:** Médio+

G - Snake Oil Salesperson

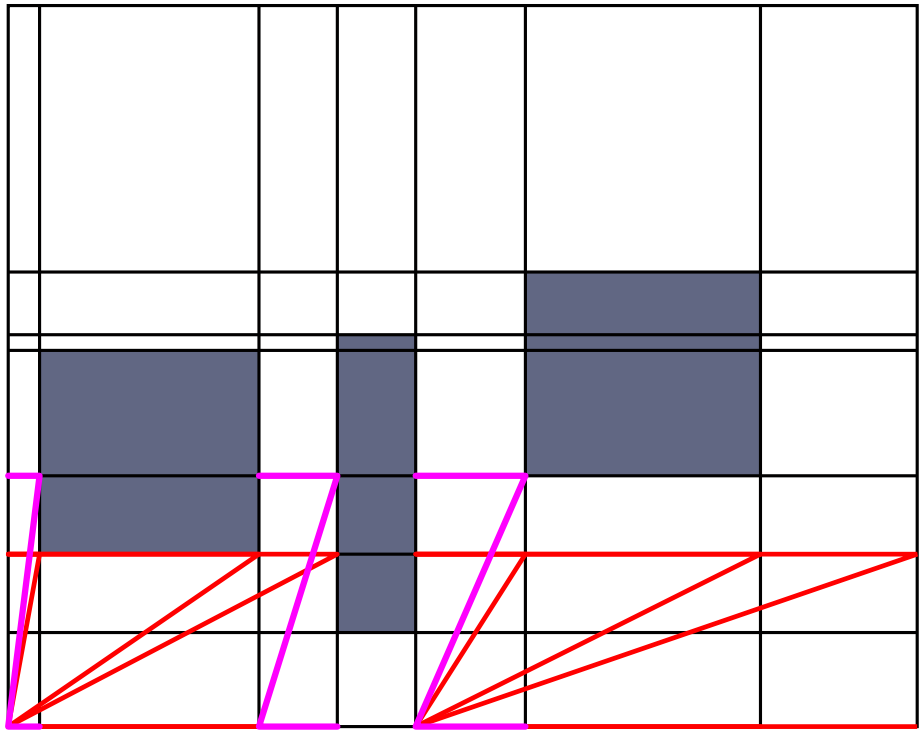


Criar uma grelha com as coordenadas dos obstáculos

G - Snake Oil Salesperson

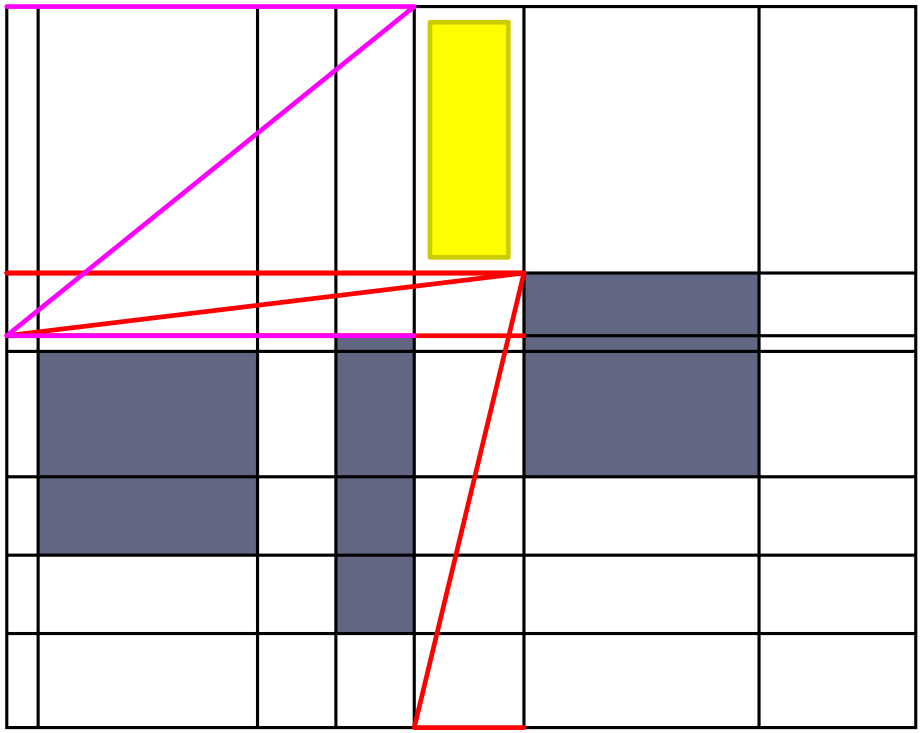


Marcar células da grelha ocupadas pelos obstáculos

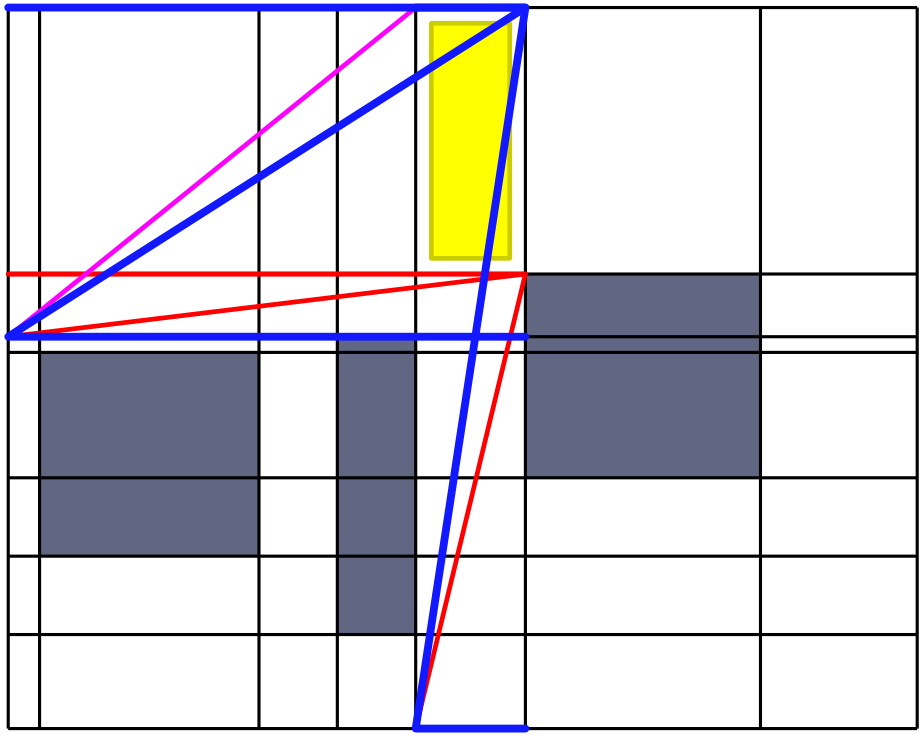


Varrer a grelha acumulando as áreas das células livres

G - Snake Oil Salesperson



Os retângulos na célula livre atual dependem dos retângulos nas células em baixo e à esquerda



Atualizar a lista dos maiores retângulos encontrados até então

H - MagicStone Deckbuilder

Problema

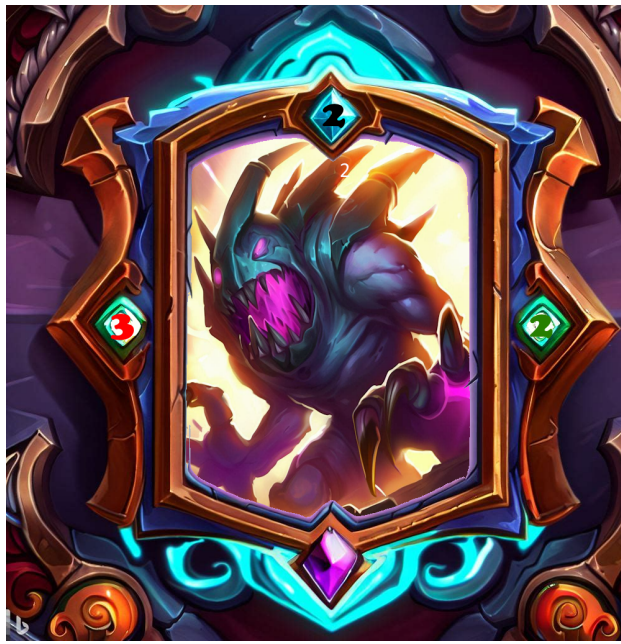
- **Input:**
  - S - Tamanho do baralho a construir
  - C - Custo máximo do baralho
  - Informação sobre as cartas disponíveis (custo, ataque, defesa)
- **Output:** Determinar o conjunto de tamanho S que maximiza o poder (ataque + defesa) do baralho sem exceder o custo máximo

Restrições

- $1 \leq S \leq 30$
- $1 < C \leq 300$
- $1 < N < 50$

Classificação

- **Categorias:** Combinatório
- **Dificuldade:** Médio



H - MagicStone Deckbuilder

Solução 1

- Abordagem Branch & Bound
- Geramos todas as combinações possíveis, e escolhemos a de maior valor
  - Em caso de empate escolher por ordem lexicográfica das cartas no baralho
- Cortamos as combinações que sabemos serem inválidas (ou que prevemos serem inválidas)

H - MagicStone Deckbuilder

Representação

[0 0 1 0 1 0 0 .... 0 1]

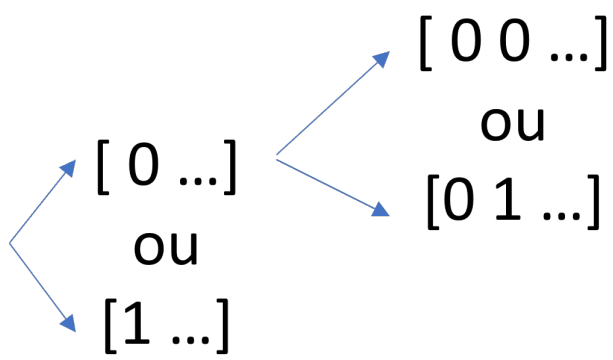
array que representa a seleção de cartas
1 – carta escolhida para o baralho
0 – carta não escolhida

H - MagicStone Deckbuilder

Geração

Podemos representar este problema como uma sequência de decisões a fazer. 2 decisões possíveis:

- incluir a carta da posição i
- não incluir a carta da posição i



H - MagicStone Deckbuilder

Fim de um caminho

Terminamos um caminho quando já não há mais cartas de onde escolher. Nesta situação, verificamos se o baralho atual é válido, e se é melhor que o que temos até agora.

H - MagicStone Deckbuilder

Cortes

- Não devemos explorar combinações para as quais já sabemos à partida que não são válidas.
  - $cartasbaralho > S$
  - $custobaralho + custocarta_i > maxcost$
- devemos usar heurísticas para detetar um caminho inválido o mais rapidamente possível
  - já não é possível ter o número de cartas necessárias
  - O custo de adicionar as próximas cartas necessárias ultrapassa o custo máximo

H - MagicStone Deckbuilder

Solução 2

- Abordagem DP
- Este problema também pode visto como um problema da mochila com múltiplos objetivos
- Dynamic Programming com uma tabela de 3 dimensões [N,S,C]
- Temos de ter cuidado com o critério de ordenação lexicográfica

I - Talking with Aliens

Problema

- **Input:** Uma *String* inicial e uma sequência de instruções `JOIN` e `CUT`.
- **Output:** Qual a *String* final gerada? (uma única linha)

Limites

- $1 \leq N \leq 2500$
- $1 \leq \texttt{length}(S_0) \leq 1000$
- $1 \leq \texttt{length}(S_N) \leq 1000$

Classificação

- **Categoria:** Estruturas de dados
- **Dificuldade:** Médio+

Descrição do problema

Input

- Uma *String* inicial

```
HELLOWORLD
```

- Uma sequência de operações `JOIN` e `CUT`

```
CUT 0 0 5
JOIN 1 1
```

- Instrução `CUT`: obter uma sub-*String*

$$\texttt{HELLOWORLD} \Longrightarrow \texttt{CUT 0 0 5} \Longrightarrow \texttt{HELLOWORLD} \Longrightarrow \texttt{HELLO}$$

- Instrução `JOIN`: concatenação de duas *Strings*

$$\texttt{JOIN 1 1} \Longrightarrow \texttt{JOIN HELLO HELLO} \Longrightarrow \texttt{HELLOHELLO}$$

Implementação ineficiente

Resolução aparentemente simples

```
String[] strings = new String[N+1];
strings[0] = in.readLine();
for (int i = 1; i <= N; i++) {
  String[] line = in.readLine().split(" ");
  String instr = line[0];
  if (instr.equals("CUT")) {
    int j = Integer.parseInt(line[1]);
    int l = Long.parseLong(line[2]);
    int u = Long.parseLong(line[3]);
    strings[i] = strings[j].substring(l, u);
  }
  else {
    int j = Integer.parseInt(line[1]);
    int k = Integer.parseInt(line[2]);
    strings[i] = strings[j] + strings[k];
  }
```

Implementação ineficiente

CUT: operação constante.

JOIN: concatenação de *Strings*, linear no tamanho do primeiro argumento.

```
A1B2C3
JOIN 0 0
JOIN 1 1
JOIN 2 2
...
JOIN 60 60
CUT 61 1 800
```

- Último `JOIN`: tamanho do primeiro argumento $\sim 2^{60}$ caracteres.
- *java.lang.OutOfMemoryError: Overflow: String length out of range*
- *C++: killed* (MLE)

Implementação esperada

Referência

Ropes: an Alternative to Strings
Boehm, Atkinson, Plass, 1995

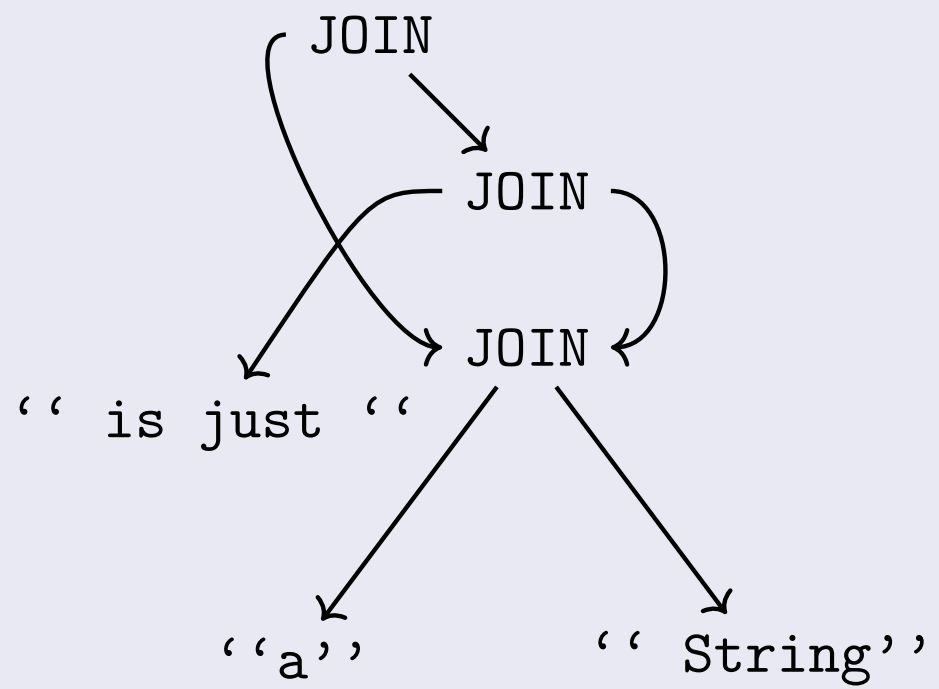
Ideia

- Não concatenar *Strings*; construir **árvores binárias**
- Estrutura de dados **persistente**: *sharing* é possível

Implementação

- `JOIN`: $\mathcal{O}(1)$
- `CUT`: se for uma folha $\mathcal{O}(1)$; senão, recursivamente em $\mathcal{O}(n)$.

Exemplo



J — The Enigmatic Connection

Problema

- **Input:** Duas sequências de pares de coordenadas (x, y), com comprimentos M e N
- **Output:** Maior número de elementos em comum, respeitando a ordem, e número de maneiras de obter esse número de elementos em comum

Limites

$$0 \leq M, N \leq 4\,000$$
$$1 \leq x, y \leq 100$$

Classificação

Categoria: Programação dinâmica (LCS)
Dificuldade: Médio **Autor:** Vasco Pedro

J — The Enigmatic Connection

Comprimento de uma LCS (sequências s e t)

$$\ell_{st}(i,j) = \begin{cases} 0 & \text{se } i = 0 \vee j = 0 \\ 1 + \ell_{st}(i-1, j-1) & \text{se } i,j > 0 \wedge s_i = t_j \\ \max\{\ell_{st}(i-1,j), \ell_{st}(i,j-1)\} & \text{se } i,j > 0 \wedge s_i \neq t_j \end{cases}$$

Símbolos

$$\text{símbolo}((x,y)) = (x-1) + (y-1) \times 100$$

J — The Enigmatic Connection

Comprimento e número de construções de uma LCS

(comprimento, número)

$$\ell c_{st}(i,j) = \begin{cases} (0,1) & \text{se} \quad i = 0 \vee j = 0 \\ \ldots & \text{se} \quad i,j > 0 \wedge s_i = t_j \\ \ldots & \text{se} \quad i,j > 0 \wedge s_i \neq t_j \end{cases}$$

J — The Enigmatic Connection

Match

		A
$\ldots$		
	(ℓ, a)	(ℓ, b)
A	(ℓ, c)	$(\ell+1, a)$

		A
$\ldots$		
	(ℓ, a)	$(\ell+1, b)$
A	(ℓ, c)	$(\ell+1, a+b)$

		A
$\ldots$		
	(ℓ, a)	(ℓ, b)
A	$(\ell+1, c)$	$(\ell+1, a+c)$

		A
$\ldots$		
	(ℓ, a)	$(\ell+1, b)$
A	$(\ell+1, c)$	$(\ell+1, a+b+c)$

J — The Enigmatic Connection

No match ($A \neq B$)

		A
$\ldots$		
	(ℓ, a)	(ℓ, b)
B	(ℓ, c)	$(\ell, b + c - a)$

		A
$\ldots$		
	(ℓ, a)	$(\ell + 1, b)$
B	(ℓ, c)	$(\ell + 1, b)$

		A
$\ldots$		
	(ℓ, a)	(ℓ, b)
B	$(\ell + 1, c)$	$(\ell + 1, c)$

		A
$\ldots$		
	(ℓ, a)	$(\ell + 1, b)$
B	$(\ell + 1, c)$	$(\ell + 1, b + c)$

K - Periodic Quadratic Equations

Problema

- **Input:** Equação do segundo grau.
- **Output:** Tamanho do ciclo.

Limites

- $1 \leq \mathtt{N} \leq 1000000$
- $1 \leq \mathtt{A} \leq 9223372036854775807$
- $-9223372036854775807 \leq \mathtt{B} \leq 9223372036854775807$
- $-9223372036854775807 \leq \mathtt{C} \leq 9223372036854775807$
- $1 \leq a \leq 9$

Classificação

- **Categorias:** Matemática
- **Dificuldade:** Médio+

Transformação Exemplo

$$3x_0^2 \qquad -18x_0 \quad +26 = 0, \qquad x_0 = 3 + 1/x_1$$

Transformação Exemplo

$$3x_0^2 \quad -18x_0 \quad +26 = 0, \qquad x_0 = 3 + 1/x_1$$

$$3(3 + 1/x_1)^2 \; -18(3 + 1/x_1) \quad +26 = 0$$

Transformação Exemplo

$$
\begin{aligned}
3x_0^2 \quad & -18x_0 \quad +26 = 0, \qquad x_0 = 3 + 1/x_1 \\
3(3 + 1/x_1)^2 \; & -18(3 + 1/x_1) \quad +26 = 0 \\
3(9 + 6/x_1 + 1/x_1^2) \; & -18(3 + 1/x_1) \quad +26 = 0
\end{aligned}
$$

Transformação Exemplo

$$
\begin{aligned}
3x_0^2 \quad & -18x_0 \quad +26 = 0, \qquad x_0 = 3 + 1/x_1 \\
3(3 + 1/x_1)^2 \quad & -18(3 + 1/x_1) \quad +26 = 0 \\
3(9 + 6/x_1 + 1/x_1^2) \quad & -18(3 + 1/x_1) \quad +26 = 0 \\
27 + 18/x_1 + 3/x_1^2 \quad & -54 - 18/x_1 \quad +26 = 0
\end{aligned}
$$

Transformação Exemplo

$$
\begin{aligned}
3x_0^2 \quad & -18x_0 \quad +26 = 0, \qquad x_0 = 3 + 1/x_1 \\
3(3 + 1/x_1)^2 \; & -18(3 + 1/x_1) \quad +26 = 0 \\
3(9 + 6/x_1 + 1/x_1^2) \; & -18(3 + 1/x_1) \quad +26 = 0 \\
27 + 18/x_1 + 3/x_1^2 \quad & -54 - 18/x_1 \quad +26 = 0 \\
27x_1^2 + 18x_1 + 3 \; & -54x_1^2 - 18x_1 \; +26x_1^2 = 0
\end{aligned}
$$

Transformação Exemplo

$$
\begin{aligned}
3x_0^2 \quad & -18x_0 \quad +26 = 0, \qquad x_0 = 3 + 1/x_1 \\
3(3 + 1/x_1)^2 \; & -18(3 + 1/x_1) \quad +26 = 0 \\
3(9 + 6/x_1 + 1/x_1^2) \; & -18(3 + 1/x_1) \quad +26 = 0 \\
27 + 18/x_1 + 3/x_1^2 \quad & -54 - 18/x_1 \quad +26 = 0 \\
27x_1^2 + 18x_1 + 3 \quad & -54x_1^2 - 18x_1 \; +26x_1^2 = 0 \\
(27 - 54 + 26)x_1^2 \quad & +(18 - 18)x_1 \qquad +3 = 0
\end{aligned}
$$

Transformação Exemplo

$$
\begin{aligned}
3x_0^2 - 18x_0 + 26 &= 0, & x_0 &= 3 + 1/x_1 \\
3(3 + 1/x_1)^2 - 18(3 + 1/x_1) + 26 &= 0 \\
3(9 + 6/x_1 + 1/x_1^2) - 18(3 + 1/x_1) + 26 &= 0 \\
27 + 18/x_1 + 3/x_1^2 - 54 - 18/x_1 + 26 &= 0 \\
27x_1^2 + 18x_1 + 3 - 54x_1^2 - 18x_1 + 26x_1^2 &= 0 \\
(27 - 54 + 26)x_1^2 + (18 - 18)x_1 + 3 &= 0 \\
x_1^2 + 0x_1 - 3 &= 0
\end{aligned}
$$

Transformação geral

$$Ax_0^2 \qquad +Bx_0 \quad +C = 0, \qquad x_0 = a + 1/x_1$$

Transformação geral

$$
\begin{aligned}
Ax_0^2 \quad &+Bx_0 \quad &+C = 0, \qquad & x_0 = a + 1/x_1 \\
A(a + 1/x_1)^2 \; &+B(a + 1/x_1) \quad &+C = 0 &
\end{aligned}
$$

Transformação geral

$$
\begin{aligned}
Ax_0^2 \quad & +Bx_0 \quad +C = 0, \qquad x_0 = a + 1/x_1 \\
A(a + 1/x_1)^2 \quad & +B(a + 1/x_1) \quad +C = 0 \\
A(a^2 + 2a/x_1 + 1/x_1^2) \quad & +B(a + 1/x_1) \quad +C = 0
\end{aligned}
$$

Transformação geral

$$
\begin{aligned}
Ax_0^2 \quad & +Bx_0 \quad +C = 0, \qquad x_0 = a + 1/x_1 \\
A(a + 1/x_1)^2 \quad & +B(a + 1/x_1) \quad +C = 0 \\
A(a^2 + 2a/x_1 + 1/x_1^2) \quad & +B(a + 1/x_1) \quad +C = 0 \\
Aa^2 + 2Aa/x_1 + A/x_1^2 \quad & +Ba + B/x_1 \quad +C = 0
\end{aligned}
$$

Transformação geral

$$
\begin{aligned}
Ax_0^2 \quad\quad\quad\quad\quad\quad + Bx_0 \quad + C &= 0, \qquad x_0 = a + 1/x_1 \\
A(a + 1/x_1)^2 \; + B(a + 1/x_1) \quad + C &= 0 \\
A(a^2 + 2a/x_1 + 1/x_1^2) \; + B(a + 1/x_1) \quad + C &= 0 \\
Aa^2 + 2Aa/x_1 + A/x_1^2 \quad + Ba + B/x_1 \quad + C &= 0 \\
Aa^2x_1^2 + 2Aax_1 + A \; + Bax_1^2 + Bx_1 + Cx_1^2 &= 0
\end{aligned}
$$

Transformação geral

$$
\begin{aligned}
Ax_0^2 \quad & +Bx_0 \quad +C = 0, \qquad x_0 = a + 1/x_1 \\
A(a + 1/x_1)^2 \; & +B(a + 1/x_1) \quad +C = 0 \\
A(a^2 + 2a/x_1 + 1/x_1^2) \; & +B(a + 1/x_1) \quad +C = 0 \\
Aa^2 + 2Aa/x_1 + A/x_1^2 \quad & +Ba + B/x_1 \quad +C = 0 \\
Aa^2x_1^2 + 2Aax_1 + A \; & +Bax_1^2 + Bx_1 \; +Cx_1^2 = 0 \\
(Aa^2 + Ba + C)x_1^2 \; & +(2Aa + B)x_1 \quad +A = 0
\end{aligned}
$$

Transformação geral

$$
\begin{aligned}
Ax_0^2 \quad & +Bx_0 \quad +C = 0, \qquad x_0 = a + 1/x_1 \\
A(a+1/x_1)^2 \; & +B(a+1/x_1) \quad +C = 0 \\
A(a^2 + 2a/x_1 + 1/x_1^2) \; & +B(a+1/x_1) \quad +C = 0 \\
Aa^2 + 2Aa/x_1 + A/x_1^2 \quad & +Ba + B/x_1 \quad +C = 0 \\
Aa^2x_1^2 + 2Aax_1 + A \; & +Bax_1^2 + Bx_1 \; +Cx_1^2 = 0 \\
(Aa^2 + Ba + C)x_1^2 \; & +(2Aa + B)x_1 \quad +A = 0 \\
\pm(Aa^2 + Ba + C)x_1^2 \; & \pm(2Aa + B)x_1 \quad \pm A = 0
\end{aligned}
$$

Ciclo

$$3x_0^2 - 18x_0 + 26 = 0, \qquad x_0 = 3 + 1/x_1$$

Ciclo

$$3x_0^2 - 18x_0 + 26 = 0, \qquad x_0 = 3 + 1/x_1$$

$$x_1^2 + 0x_1 - 3 = 0, \qquad x_1 = 1 + 1/x_2$$

Ciclo

$$
\begin{aligned}
3x_0^2 - 18x_0 + 26 &= 0, & x_0 &= 3 + 1/x_1 \\
x_1^2 + 0x_1 - 3 &= 0, & x_1 &= 1 + 1/x_2 \\
2x_2^2 - 2x_2 - 1 &= 0, & x_2 &= 1 + 1/x_3
\end{aligned}
$$

Ciclo

$$
\begin{aligned}
3x_0^2 - 18x_0 + 26 &= 0, & x_0 &= 3 + 1/x_1 \\
x_1^2 + 0x_1 - 3 &= 0, & x_1 &= 1 + 1/x_2 \\
2x_2^2 - 2x_2 - 1 &= 0, & x_2 &= 1 + 1/x_3 \\
x_3^2 - 2x_3 - 2 &= 0, & x_3 &= 2 + 1/x_4
\end{aligned}
$$

Ciclo

$$
\begin{aligned}
3x_0^2 - 18x_0 + 26 &= 0, & x_0 &= 3 + 1/x_1 \\
x_1^2 + 0x_1 - 3 &= 0, & x_1 &= 1 + 1/x_2 \\
2x_2^2 - 2x_2 - 1 &= 0, & x_2 &= 1 + 1/x_3 \\
x_3^2 - 2x_3 - 2 &= 0, & x_3 &= 2 + 1/x_4 \\
2x_4^2 - 2x_4 - 1 &= 0, & x_4 &= x_2
\end{aligned}
$$

Ciclo

$$
\begin{aligned}
3x_0^2 - 18x_0 + 26 &= 0, & x_0 &= 3 + 1/x_1 \\
x_1^2 + 0x_1 - 3 &= 0, & x_1 &= 1 + 1/x_2 \\
2x_2^2 - 2x_2 - 1 &= 0, & x_2 &= 1 + 1/x_3 \\
x_3^2 - 2x_3 - 2 &= 0, & x_3 &= 2 + 1/x_2
\end{aligned}
$$

Floyd's Algorithm

$$
\begin{array}{llll}
3x_0^2 - 18x_0 + 26 = 0, & & x_0 = 3 + 1/x_1 & \texttt{t} \\
x_1^2 + 0x_1 - 3 = 0, & & x_1 = 1 + 1/x_2 & \texttt{h} \\
2x_2^2 - 2x_2 - 1 = 0, & & x_2 = 1 + 1/x_3 & \\
x_3^2 - 2x_3 - 2 = 0, & & x_3 = 2 + 1/x_2 &
\end{array}
$$

Floyd's Algorithm

$$
\begin{array}{llll}
3x_0^2 - 18x_0 + 26 = 0, & x_0 = 3 + 1/x_1 & \texttt{t} \\
x_1^2 + 0x_1 - 3 = 0, & x_1 = 1 + 1/x_2 & \\
2x_2^2 - 2x_2 - 1 = 0, & x_2 = 1 + 1/x_3 & \texttt{h} \\
x_3^2 - 2x_3 - 2 = 0, & x_3 = 2 + 1/x_2 &
\end{array}
$$

Floyd's Algorithm

$$
\begin{array}{llll}
3x_0^2 - 18x_0 + 26 = 0, & x_0 = 3 + 1/x_1 & \texttt{t} \\
x_1^2 + 0x_1 - 3 = 0, & x_1 = 1 + 1/x_2 & \\
2x_2^2 - 2x_2 - 1 = 0, & x_2 = 1 + 1/x_3 & \\
x_3^2 - 2x_3 - 2 = 0, & x_3 = 2 + 1/x_2 & \texttt{h}
\end{array}
$$

Floyd's Algorithm

$$
\begin{array}{llll}
3x_0^2 - 18x_0 + 26 = 0, & & x_0 = 3 + 1/x_1 & \\
x_1^2 + 0x_1 - 3 = 0, & & x_1 = 1 + 1/x_2 & \texttt{t} \\
2x_2^2 - 2x_2 - 1 = 0, & & x_2 = 1 + 1/x_3 & \\
x_3^2 - 2x_3 - 2 = 0, & & x_3 = 2 + 1/x_2 & \texttt{h}
\end{array}
$$

Floyd's Algorithm

$$
\begin{array}{llll}
3x_0^2 - 18x_0 + 26 = 0, & & x_0 = 3 + 1/x_1 & \\
x_1^2 + 0x_1 - 3 = 0, & & x_1 = 1 + 1/x_2 & \texttt{t} \\
2x_2^2 - 2x_2 - 1 = 0, & & x_2 = 1 + 1/x_3 & \texttt{h} \\
x_3^2 - 2x_3 - 2 = 0, & & x_3 = 2 + 1/x_2 &
\end{array}
$$

Floyd's Algorithm

$$
\begin{array}{llll}
3x_0^2 - 18x_0 + 26 = 0, & & x_0 = 3 + 1/x_1 & \\
x_1^2 + 0x_1 - 3 = 0, & & x_1 = 1 + 1/x_2 & \texttt{t} \\
2x_2^2 - 2x_2 - 1 = 0, & & x_2 = 1 + 1/x_3 & \\
x_3^2 - 2x_3 - 2 = 0, & & x_3 = 2 + 1/x_2 & \texttt{h}
\end{array}
$$

Floyd's Algorithm

$$
\begin{array}{llll}
3x_0^2 - 18x_0 + 26 = 0, & x_0 = 3 + 1/x_1 & & \\
x_1^2 + 0x_1 - 3 = 0, & x_1 = 1 + 1/x_2 & & \\
2x_2^2 - 2x_2 - 1 = 0, & x_2 = 1 + 1/x_3 & & \texttt{t} \\
x_3^2 - 2x_3 - 2 = 0, & x_3 = 2 + 1/x_2 & & \texttt{h}
\end{array}
$$

Floyd's Algorithm

$$
\begin{aligned}
3x_0^2 - 18x_0 + 26 &= 0, & x_0 &= 3 + 1/x_1 \\
x_1^2 + 0x_1 - 3 &= 0, & x_1 &= 1 + 1/x_2 \\
2x_2^2 - 2x_2 - 1 &= 0, & x_2 &= 1 + 1/x_3 \qquad \texttt{ht} \\
x_3^2 - 2x_3 - 2 &= 0, & x_3 &= 2 + 1/x_2
\end{aligned}
$$

Teorema Lagrange

[J. Steining. A proof of Lagrange's theorem on periodic continued fractions. *Archiv der Mathematik*, 1992.]

Teorema Lagrange

[J. Steining. A proof of Lagrange's theorem on periodic continued fractions. *Archiv der Mathematik*, 1992.]

$$
\begin{array}{rrrl}
Ax_0^2 & Bx_0 & C & D = B^2 - 4AC \\
3x_0^2 & -18x_0 & +26 = 0, & 18^2 - 4 \times 3 \times 26 = 12 \\
x_1^2 & +0x_1 & -3 = 0, & 0^2 + 4 \times 1 \times 3 = 12 \\
2x_2^2 & -2x_2 & -1 = 0, & 2^2 + 4 \times 2 \times 1 = 12 \\
x_3^2 & -2x_3 & -2 = 0, & 2^2 + 4 \times 1 \times 2 = 12 \\
2x_4^2 & -2x_4 & -1 = 0, & 2^2 + 4 \times 2 \times 1 = 12
\end{array}
$$