Olimpíada Brasileira de Informática 2017

Nível Sênior – Fase 2: *Upsolving*

Prof. Edson Alves - UnB/FGA

2020

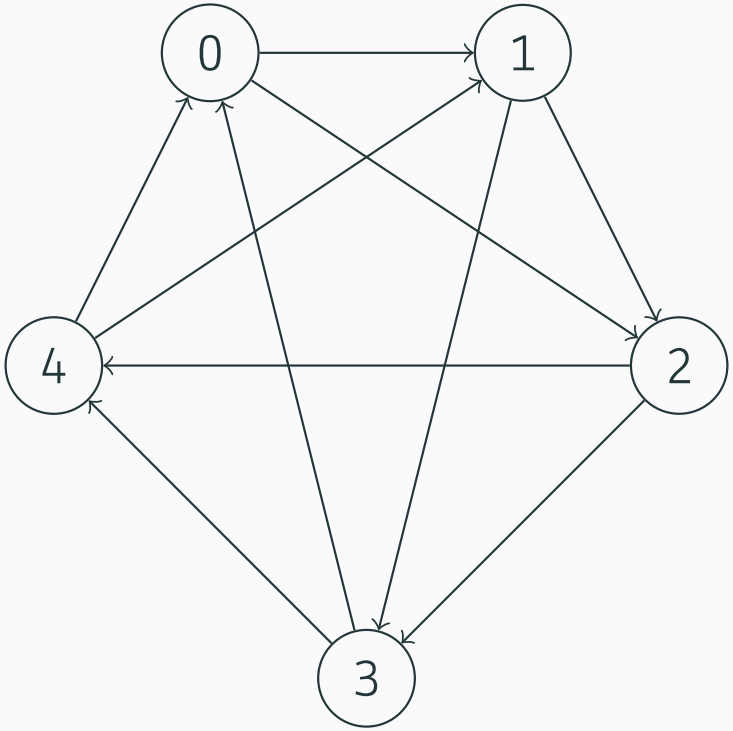
Sumário

Dario e Xerxes

Problema

A brincadeira da Pedra, Papel e Tesoura, muita gente conhece. Mas dá para fazer uma mais legal com cinco opções e não só três! Dois jogadores, `dario` e `xerxes`, jogam uma partida com N rodadas. Em cada rodada os jogadores escolhem uma "mão" entre cinco opções, que vamos representar aqui com os números 0, 1, 2, 3 e 4. A figura define exatamente quem ganha a rodada. Por exemplo, se `dario` escolheu 0 e `xerxes` escolheu 3, então `xerxes` ganha a rodada, pois existe uma seta na figura indo de 3 para 0.

Problema



Problema

Depois de N rodadas, o vencedor da partida é o jogador que ganhou mais rodadas. O número N será sempre ímpar, para não haver empate na partida. Vamos também considerar que os jogadores nunca escolhem a mesma mão numa rodada, para não haver empate na rodada. Você deve escrever um programa que determine quem venceu a partida, se foi `dario` ou `xerxes`.

Entrada e saída

Entrada

A primeira linha da entrada contém um inteiro N, o número de rodadas na partida. Cada uma das N linhas seguintes contém dois inteiros D e X, representando a mão que os jogadores dario e xerxes, respectivamente, jogaram em uma rodada.

Entrada e saída

Saída

Seu programa deve imprimir uma linha contendo o nome do jogador que venceu a partida: `dario` ou `xerxes`. Todas as letras devem ser minúsculas, sem nenhum acento!

Restrições

- $1 \leq N \leq 999$, N é impar
- $0 \leq D, X \leq 4$ e $D \neq X$

Exemplo de entradas e saídas

Entrada	Saída
3 1 3 4 2 0 2	dario
1 3 1	xerxes

Solução

- Como o problema excluí a possibilidade de ambos escolherem o mesmo número, há 20 jogadas possíveis
- Testar cada uma destas 20 possibilidades, acumulando as vitórias de cada um produz uma solução correta, porém de codificação trabalhosa e propensa a erros:

```
if (D == 0 and X == 1)
    ++dario;

if (D == 0 and X == 2)
    ++dario;

// Mais 17 ifs

if (D == 4 and X == 3)
    ++xerxes;
```

Solução

- Uma maneira de se eliminar a metade destas condicionais é observar que, caso Dario não vença, Xerxes será o vencedor
- Além disso, todas as 10 condições podem ser combinadas em uma única condição, por meio do operador lógico OU

```
if ((D == 0 && X == 1) || (D == 0 && X == 2) ||
    (D == 1 && X == 2) || (D == 1 && X == 3) ||
    (D == 2 && X == 3) || (D == 2 && X == 4) ||
    (D == 3 && X == 4) || (D == 3 && X == 0) ||
    (D == 4 && X == 0) || (D == 4 && X == 1))
        ++dario;
else
        ++xerxes;
```

Solução

- Uma vez que não há possibilidade de empate, não é preciso contabilizar as vitórias de Xerxes, pois `xerxes = N - dario`
- Além disso, Dario será o vencedor da disputa apenas se vencer mais da metade de todas as rodadas, isto é, se `dario > N/2`
- A observação cuidadosa do texto e das dez condicionais apresentadas mostra que um número x vence seus dois sucessores, se considerados apenas seus restos da divisão por 5
- Assim, a condição de vitória de Dario se reduz a

```
if (X == (D + 1) % 5 || X == (D + 2) % 5)
    ++dario;
```

Solução $O(N)$

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
    int N, D, X, dario = 0;
    cin >> N;

    for (int i = 0; i < N; ++i)
    {
        cin >> D >> X;

        if (X == (D + 1) % 5 or X == (D + 2) % 5)
            ++dario;
    }

    cout << (dario > N / 2 ? "dario" : "xerxes") << '\n';

    return 0;
}
```

Frete

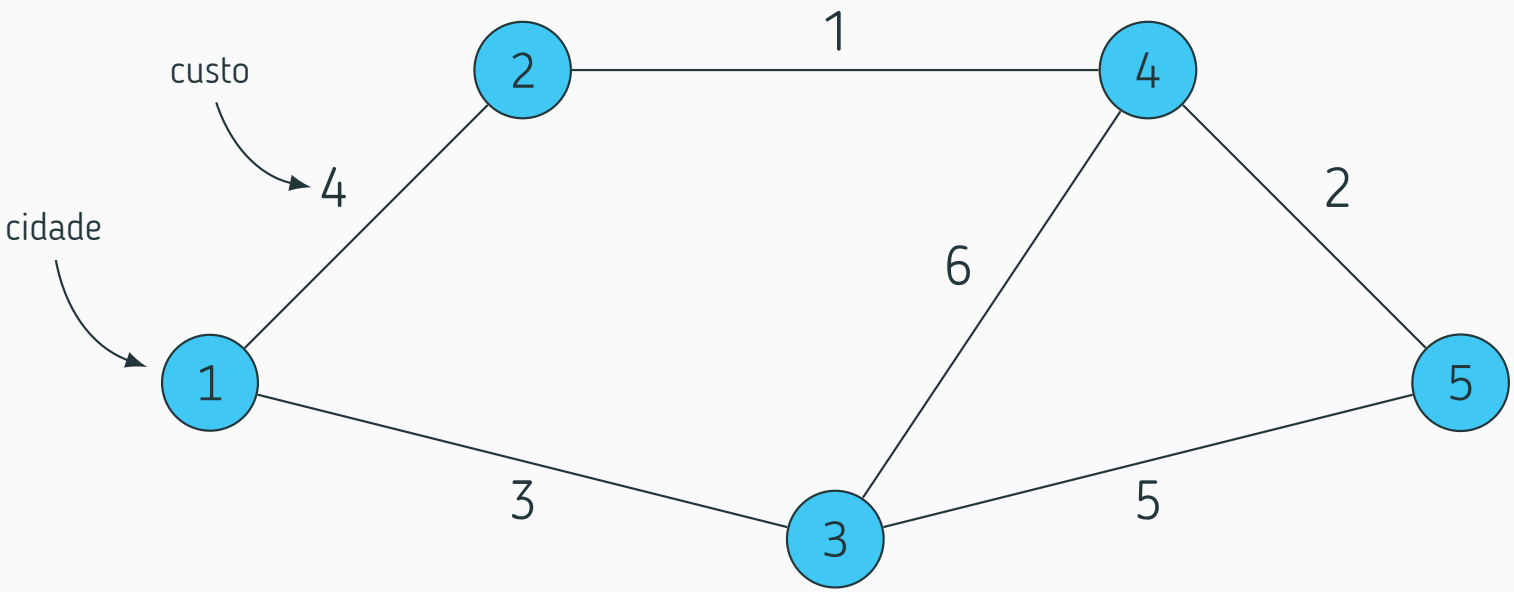
Problema

O senhor Satoshi passou anos reclamando da empresa de correios do seu país, porque ela sempre transportava suas encomendas usando um caminho que passava pelo número mínimo de cidades entre a cidade onde o senhor Satoshi mora e a cidade destino da encomenda. A empresa alegava que essa estratégia levava ao menor tempo para a entrega final da encomenda. O problema é que ele notou que essa estratégia da empresa nem sempre levava ao menor preço para o frete total. Se ele pudesse escolher o caminho por onde a encomenda deveria passar para ir da sua cidade para a cidade destino, ele poderia economizar bastante com o frete, já que não havia muita urgência para a maioria de suas encomendas.

Problema

Depois de muita reclamação, a empresa finalmente está dando aos clientes a opção de determinar o caminho por onde a encomenda deve passar. O senhor Satoshi, feliz da vida, agora quer a sua ajuda para implementar um programa que, dado o custo de transporte de uma encomenda entre vários pares de cidades pelo país, para os quais a empresa realiza entregas diretas, determine qual é o preço total mínimo para o frete entre a cidade onde ele mora e a cidade destino da encomenda.

Problema



Problema

O país tem N cidades, identificadas pelos números de 1 a N. O senhor Satoshi mora na cidade 1 e o destino da encomenda será sempre a cidade N. É garantido que sempre haverá um caminho de 1 até N. No exemplo da figura, para $N = 5$, o custo mínimo será 7, para o caminho $1 \to 2 \to 4 \to 5$.

Entrada e saída

Entrada

A primeira linha da entrada contém dois números inteiros N e M, representando o número de cidades e quantos pares de cidades possuem entrega direta de encomenda pela empresa. As M linhas seguintes contêm, cada uma, três inteiros A, B e C, indicando que a empresa realiza a entrega de uma encomenda diretamente entre as cidades A e B, cobrando o preço C.

Entrada e saída

Saída

Seu programa deve imprimir uma linha contendo um inteiro representando o preço mínimo total para o frete entre a cidade onde o senhor Satoshi mora, a cidade 1, e a cidade destino da encomenda, a cidade N.

Restrições

- $2 \leq N \leq 100$ e $1 \leq M \leq 1000$
- $1 \leq A, B \leq N$ e $A \neq B$
- $1 \leq C \leq 1000$

Exemplo de entradas e saídas

Entrada	Saída
5 6 1 2 4 1 3 3 4 3 6 4 5 2 2 4 1 3 5 5	7

Exemplo de entradas e saídas

Entrada

```
7 10
1 2 5
3 1 32
1 4 3
2 3 4
2 6 20
6 3 1
6 4 9
6 5 6
3 7 18
5 7 2
```

Saída

```
18
```

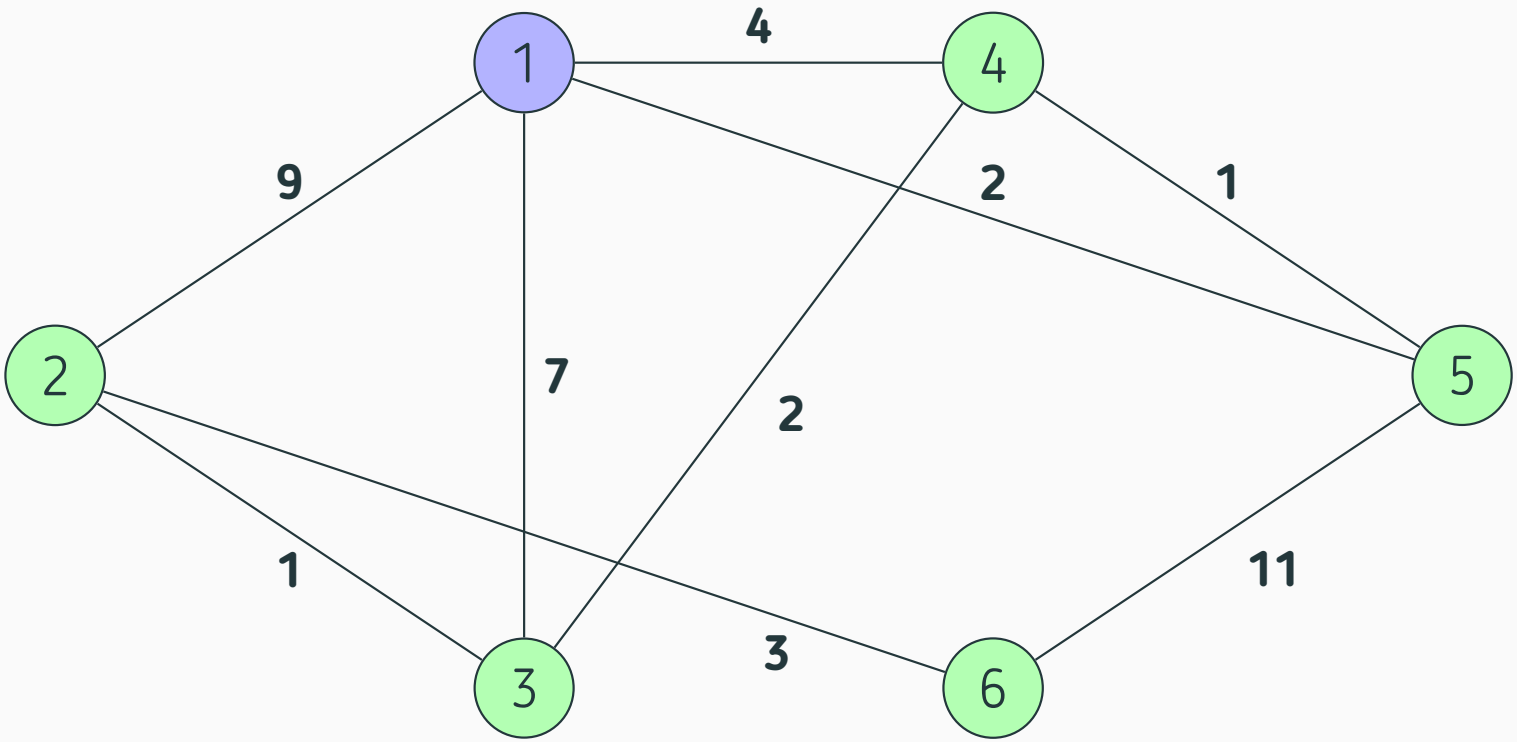
Solução

- Este é um problema de caminhos mínimos que não requer nenhum tipo de adaptação ou interpretação
- Este fato, aliado aos limites impostos quanto ao número de vértices e arestas, permite a aplicação direta de qualquer algoritmo de caminhos mínimos, sejam algoritmos de fonte única (Bellman-Ford ou Dijkstra) ou de múltiplas fontes (Floyd-Warshall)
- As diferenças entre estes algoritmos residem na complexidade assintótica de cada um e também na simplicidade ou sofisticação da implementação

Algoritmo de Bellman-Ford

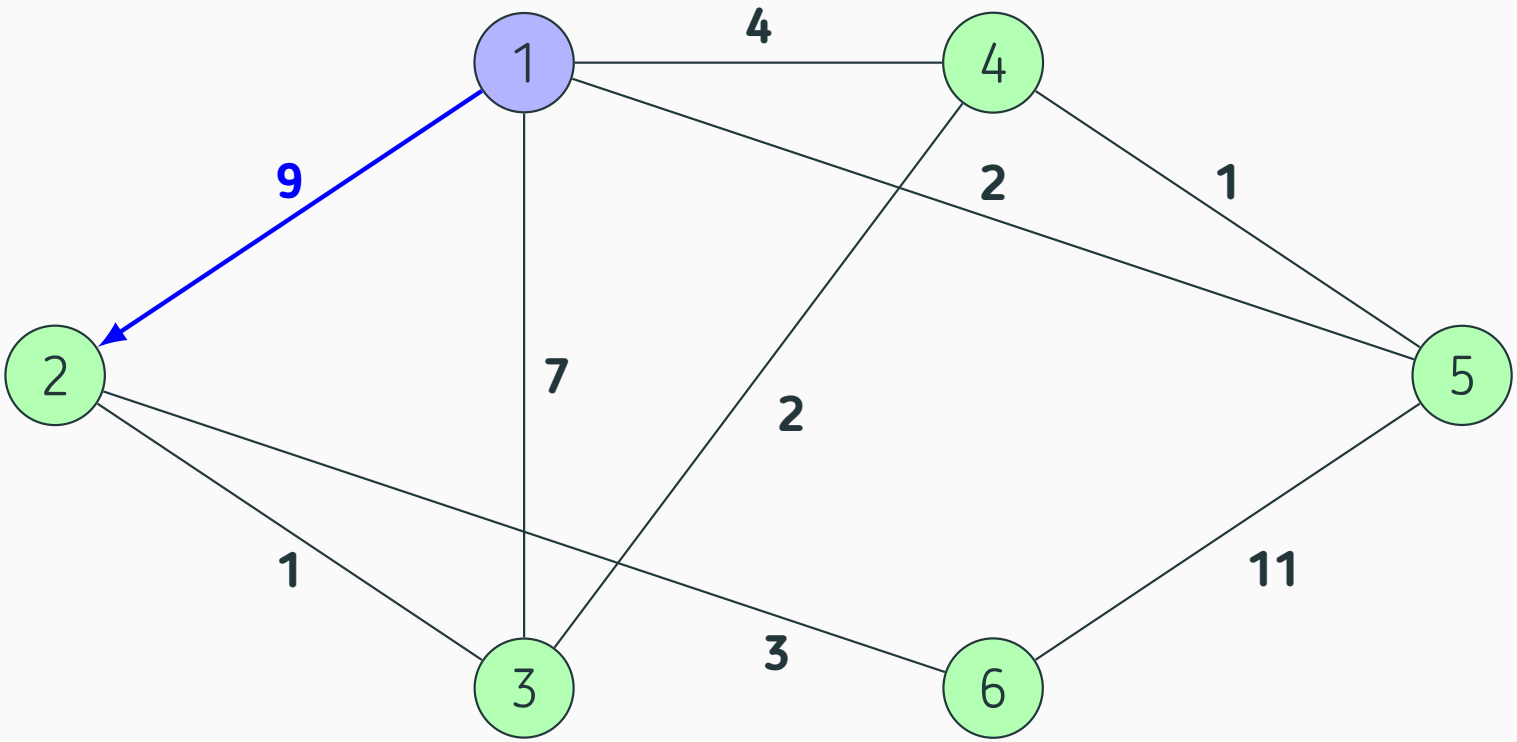
- O algoritmo de Bellman-Ford computa o caminho mínimo de todos os vértices de um grafo a um nó s dado
- Primeiramente ele inicializa a distância de s a s como zero e as demais distâncias como infinito
- A cada iteração, ele visita todas as arestas na tentativa de encurtar um caminho já existente por meio da adição de uma nova aresta a este, até que não seja mais possível realizar tal redução
- A complexidade é $O(NM)$, pois o número máximo de arestas em um caminho mínimo é igual a $N - 1$

Visualização do algoritmo de Bellman-Ford



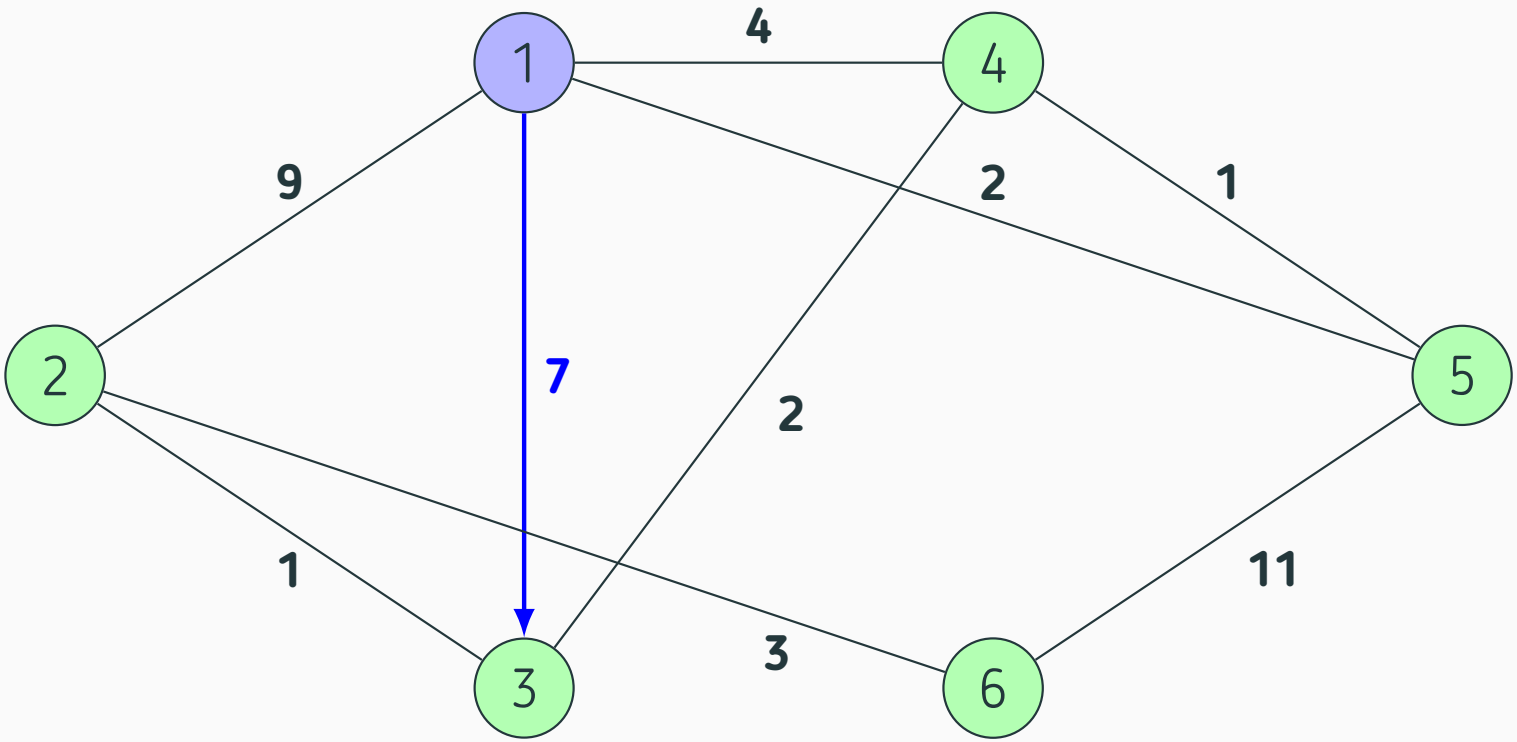
	1	2	3	4	5	6
Distâncias:	0	∞	∞	∞	∞	∞

Visualização do algoritmo de Bellman-Ford



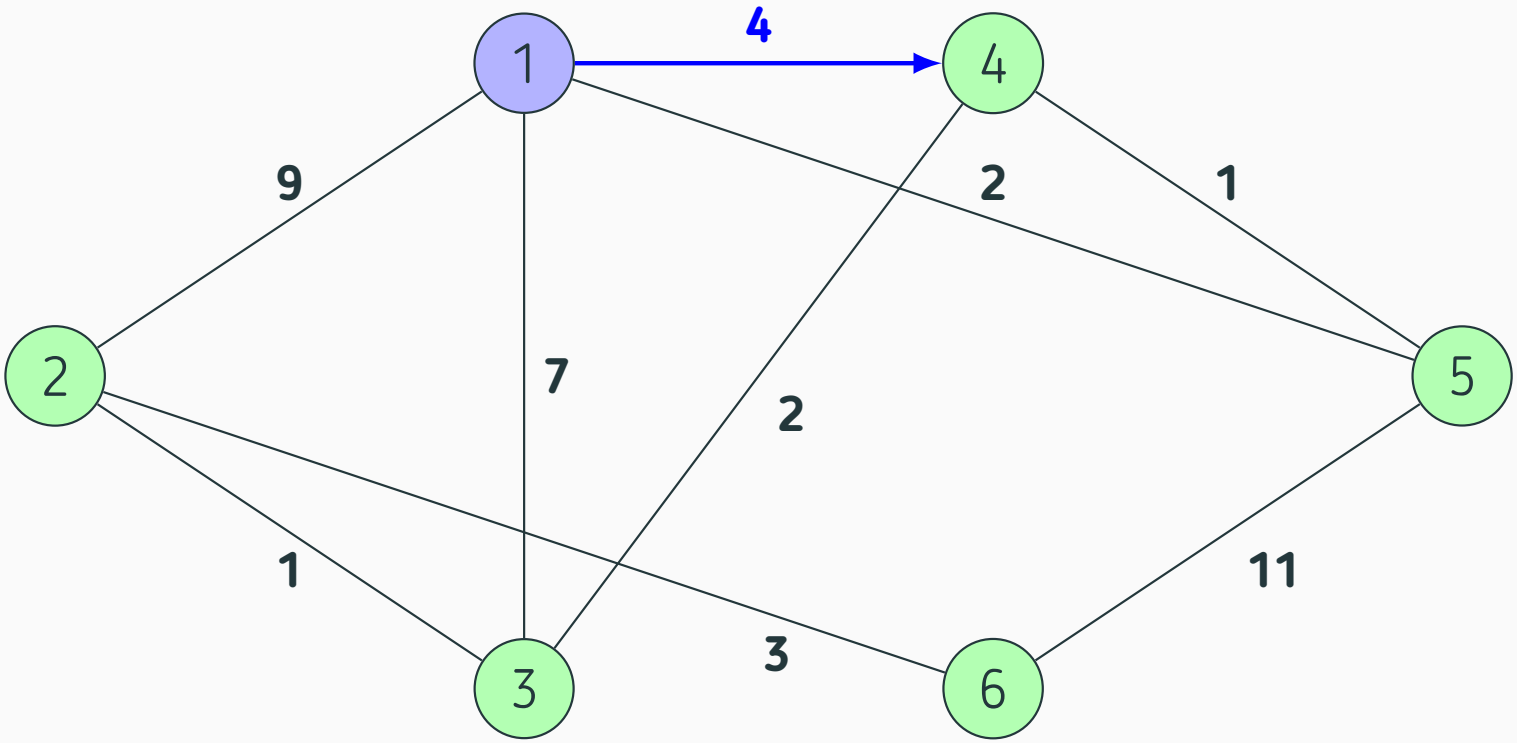
	1	2	3	4	5	6
Distâncias:	0	9	∞	∞	∞	∞

Visualização do algoritmo de Bellman-Ford



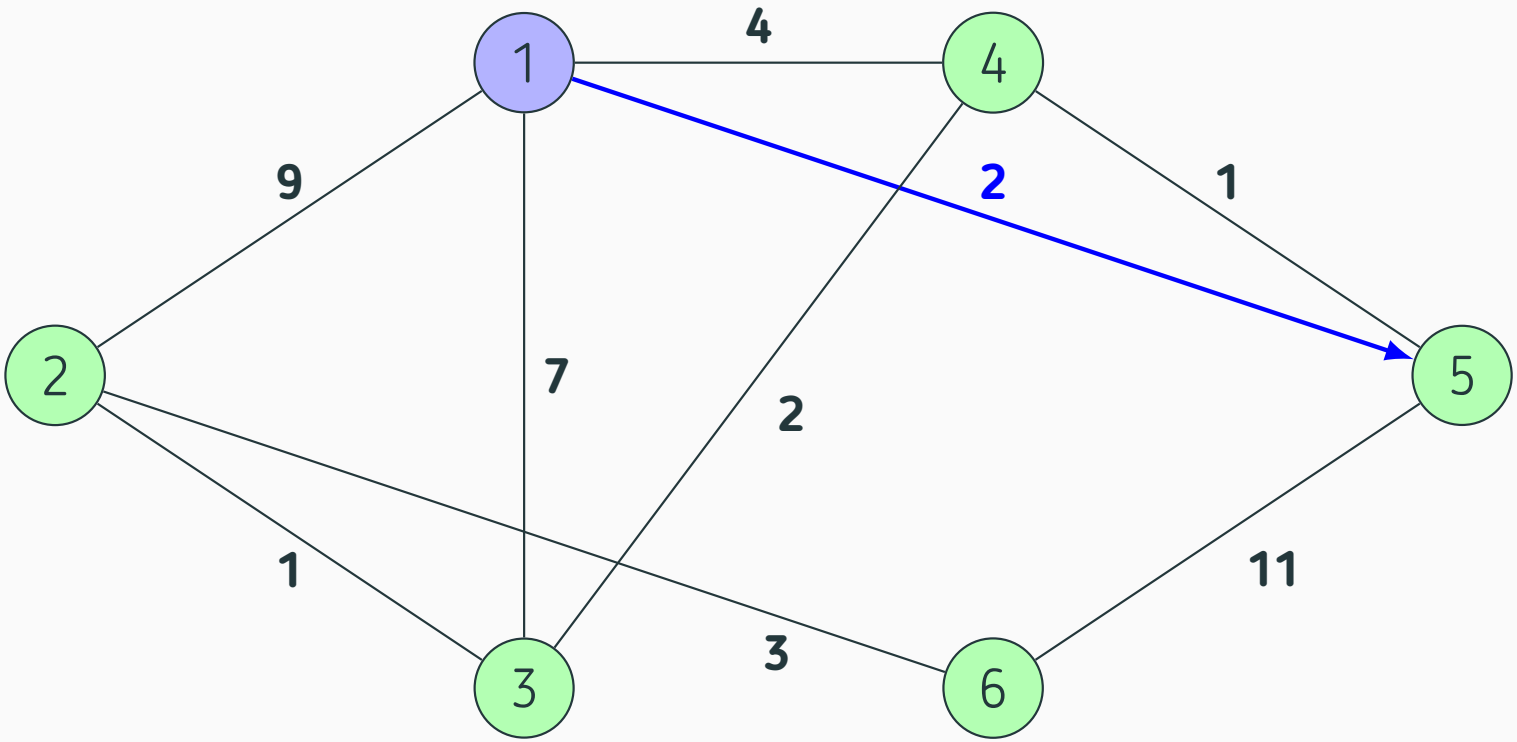
	1	2	3	4	5	6
Distâncias:	0	9	7	∞	∞	∞

Visualização do algoritmo de Bellman-Ford



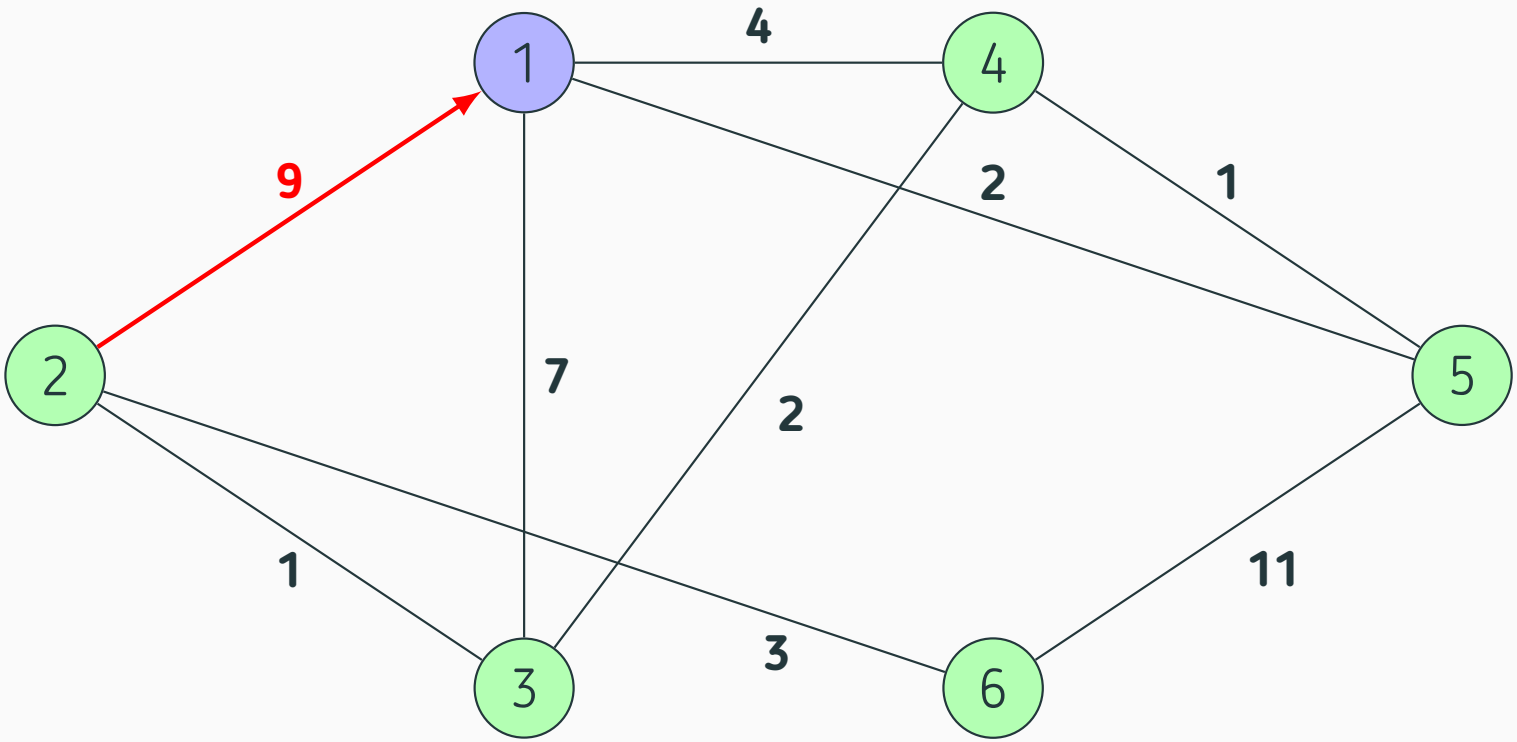
	1	2	3	4	5	6
Distâncias:	0	9	7	4	∞	∞

Visualização do algoritmo de Bellman-Ford



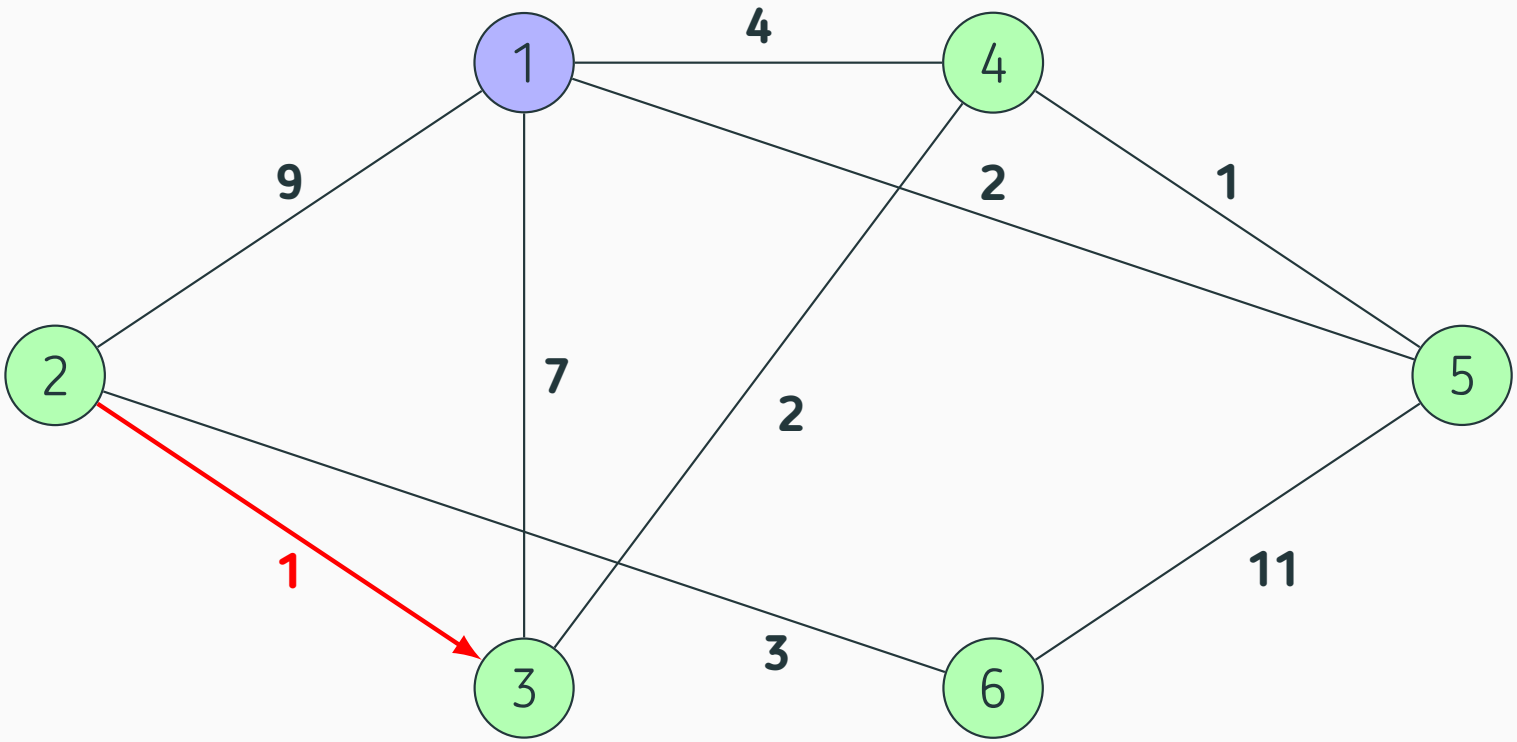
	1	2	3	4	5	6
Distâncias:	0	9	7	4	2	∞

Visualização do algoritmo de Bellman-Ford



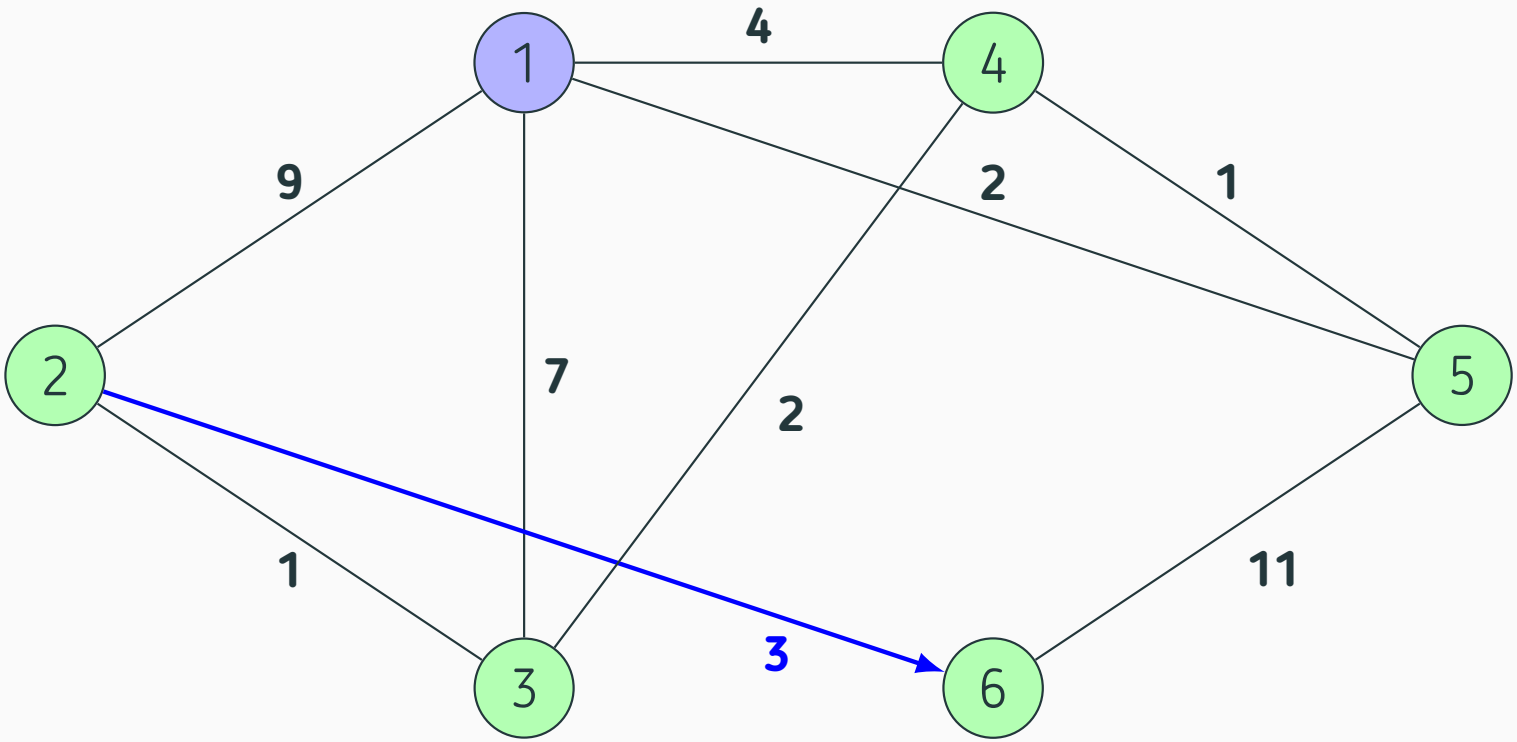
	1	2	3	4	5	6
Distâncias:	0	9	7	4	2	∞

Visualização do algoritmo de Bellman-Ford



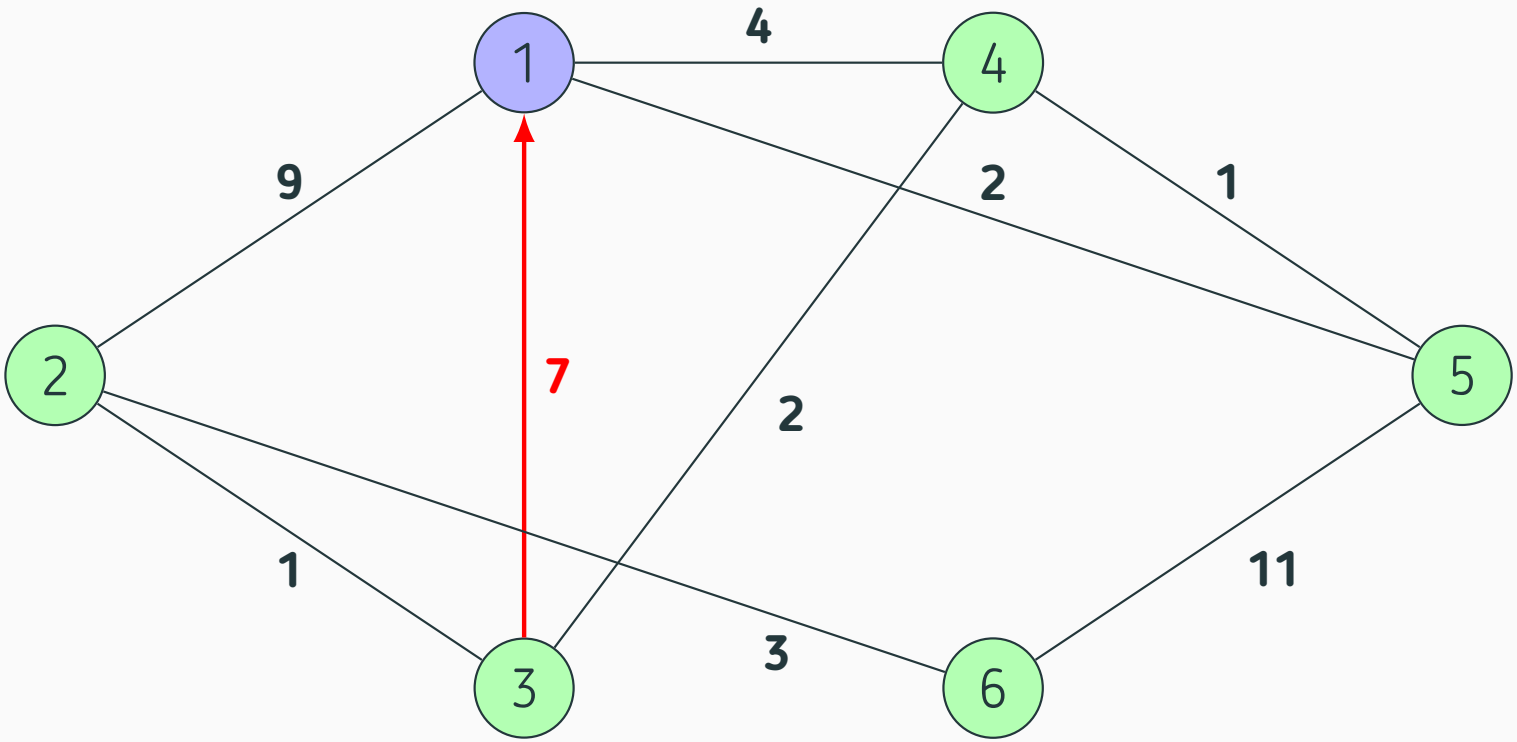
	1	2	3	4	5	6
Distâncias:	0	9	7	4	2	∞

Visualização do algoritmo de Bellman-Ford



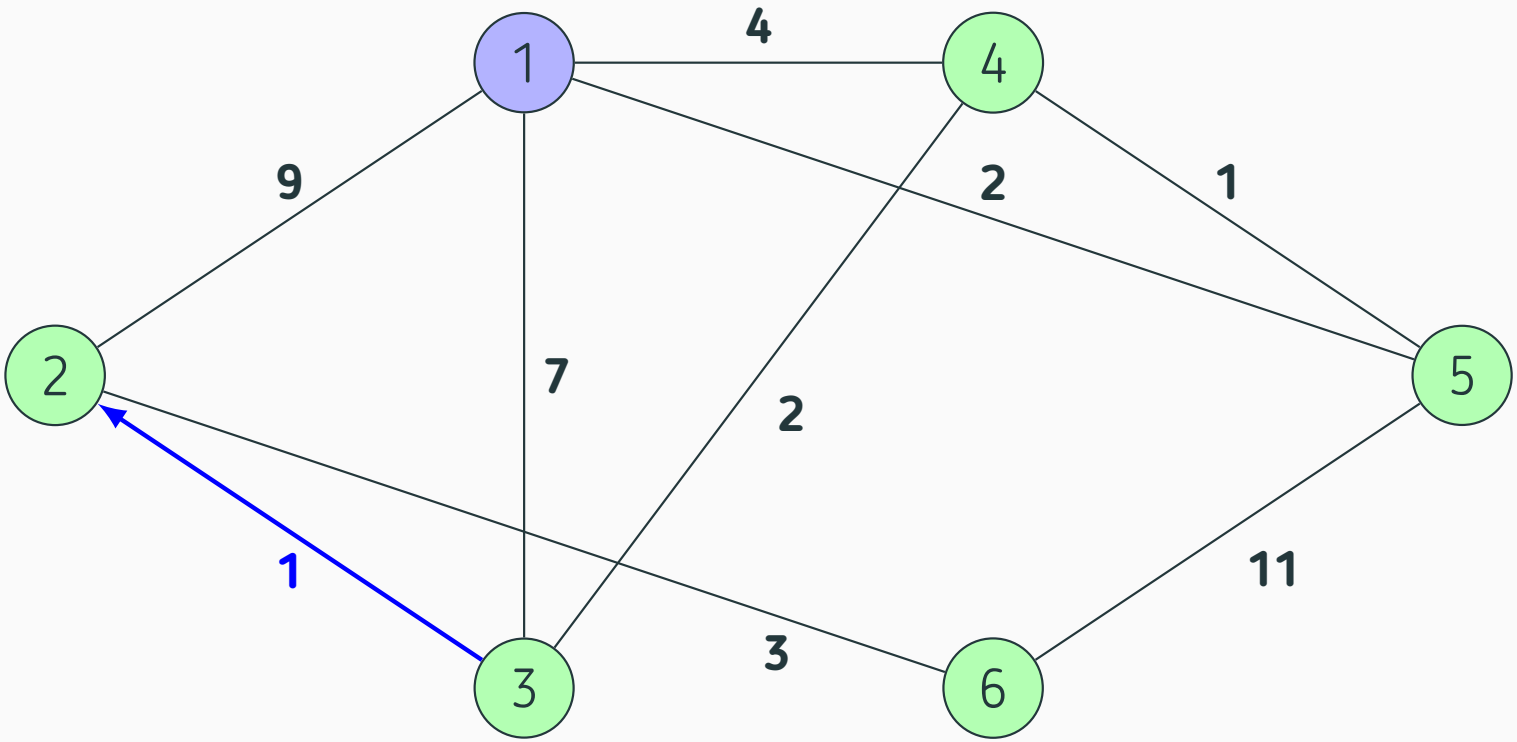
	1	2	3	4	5	6
Distâncias:	0	9	7	4	2	12

Visualização do algoritmo de Bellman-Ford



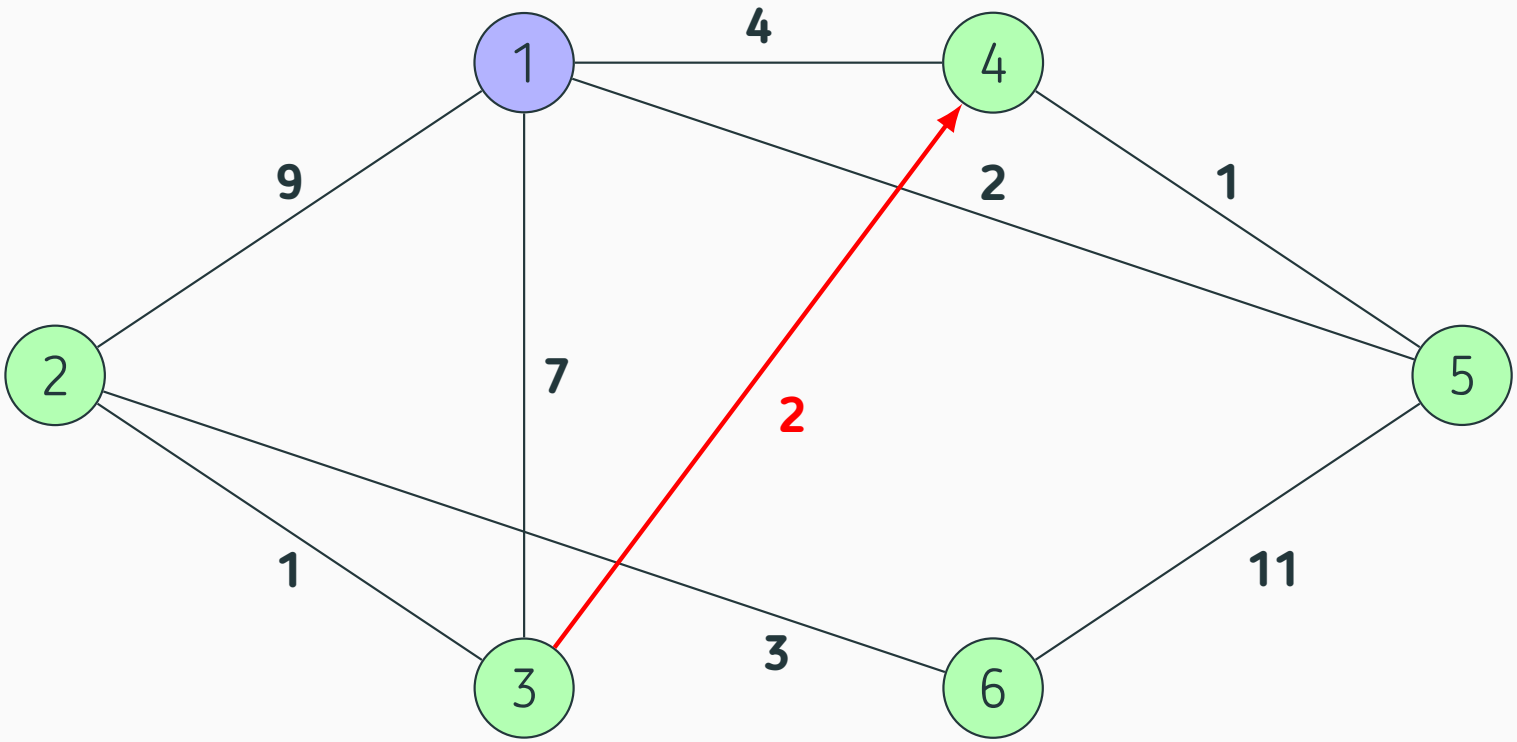
	1	2	3	4	5	6
Distâncias:	0	9	7	4	2	12

Visualização do algoritmo de Bellman-Ford



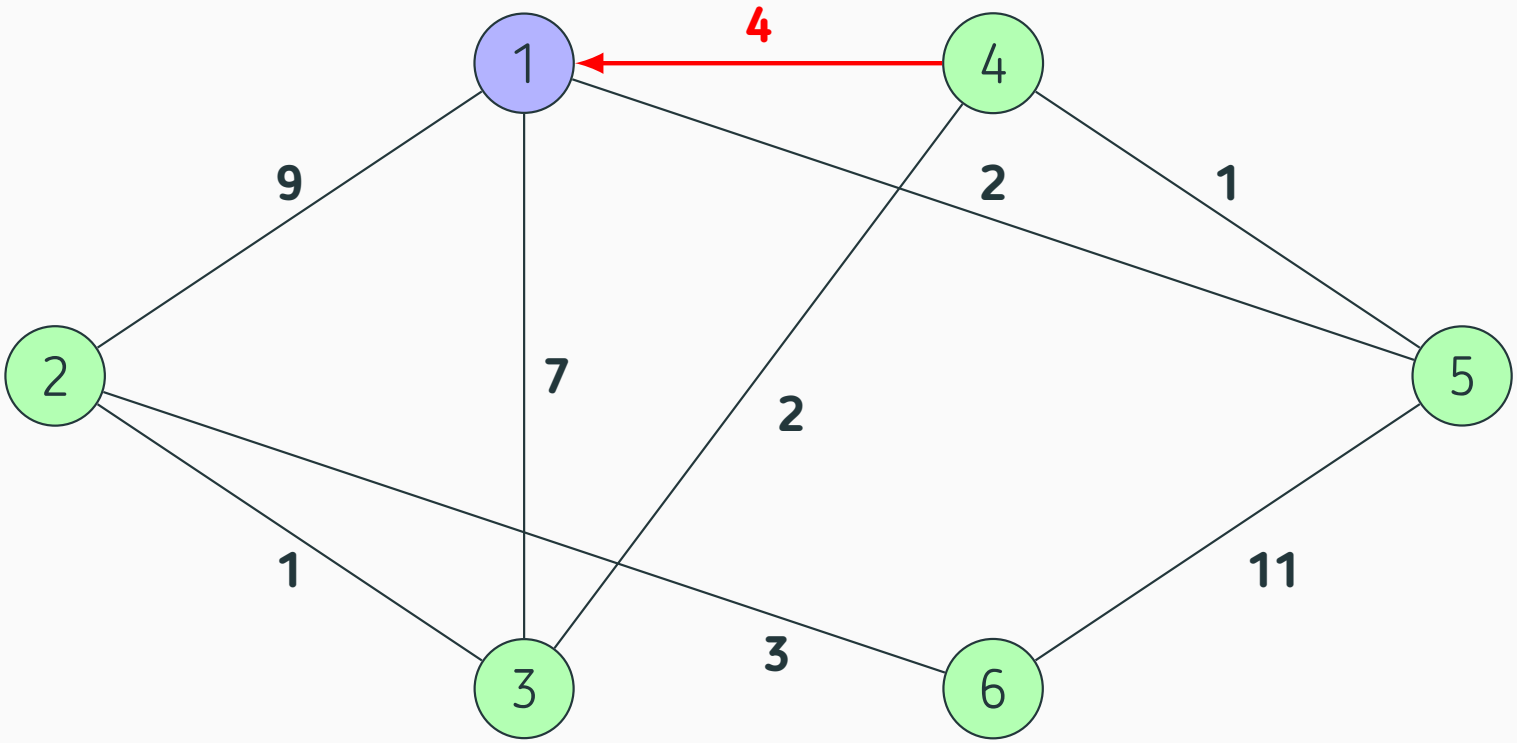
	1	2	3	4	5	6
Distâncias:	0	8	7	4	2	12

Visualização do algoritmo de Bellman-Ford



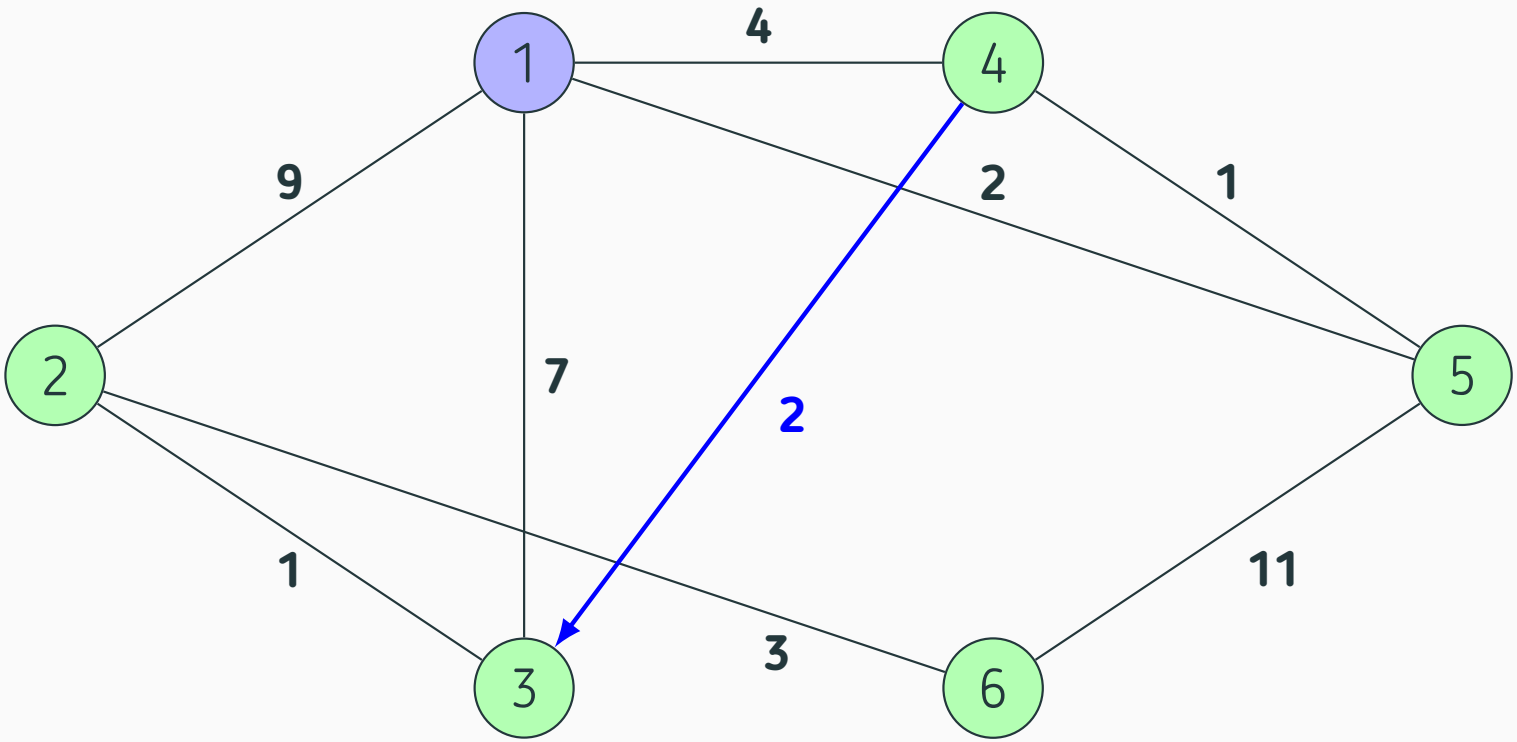
	1	2	3	4	5	6
Distâncias:	0	8	7	4	2	12

Visualização do algoritmo de Bellman-Ford



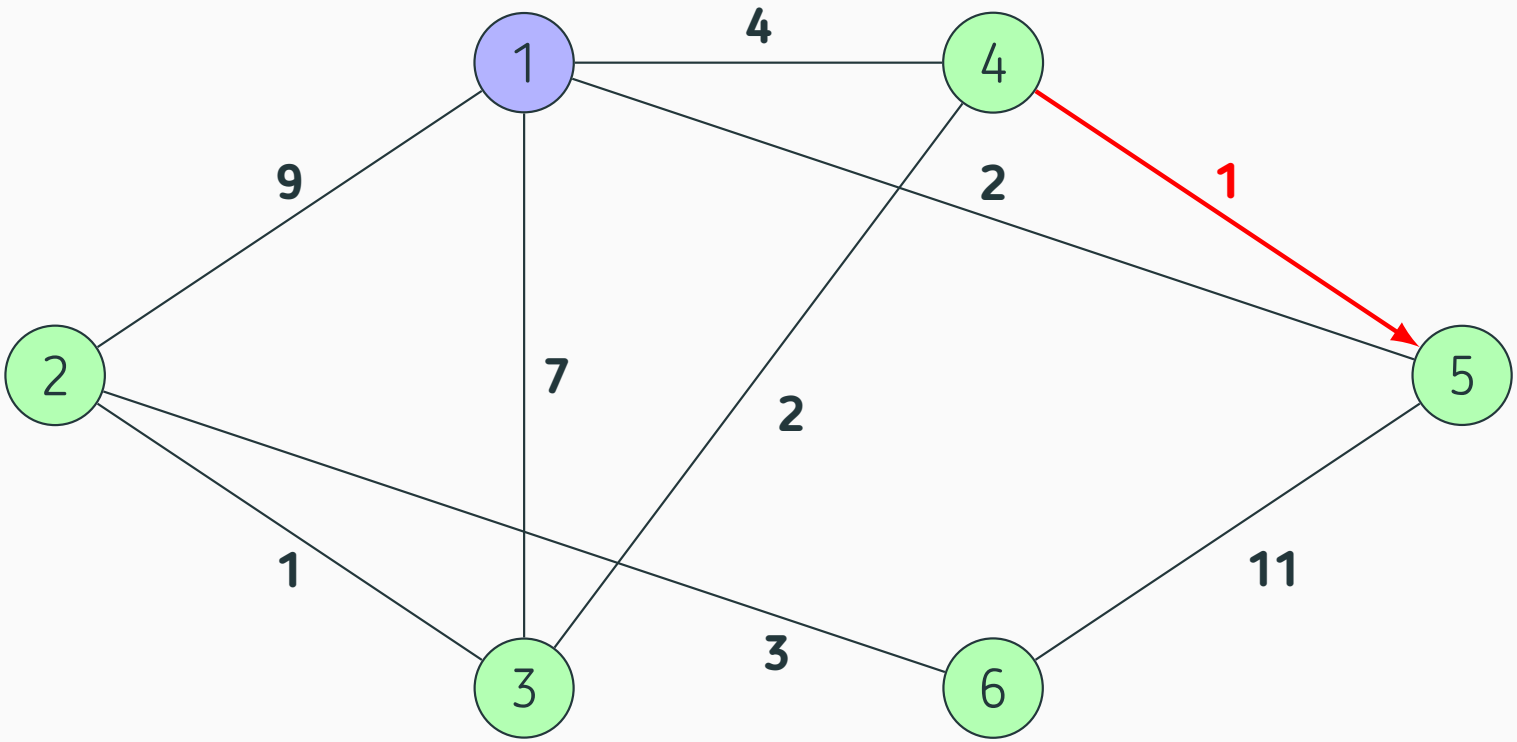
	1	2	3	4	5	6
Distâncias:	0	8	7	4	2	12

Visualização do algoritmo de Bellman-Ford



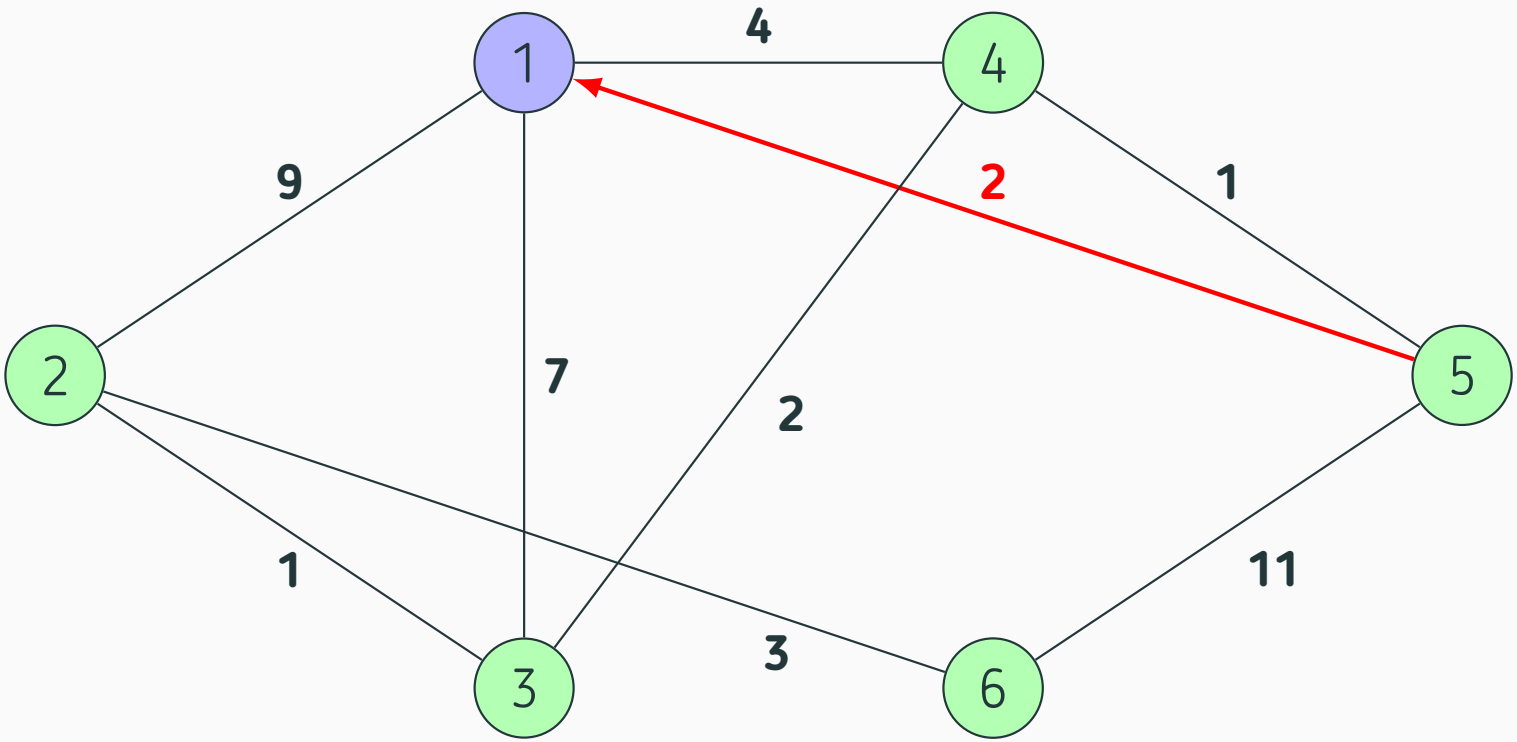
	1	2	3	4	5	6
Distâncias:	0	8	6	4	2	12

Visualização do algoritmo de Bellman-Ford



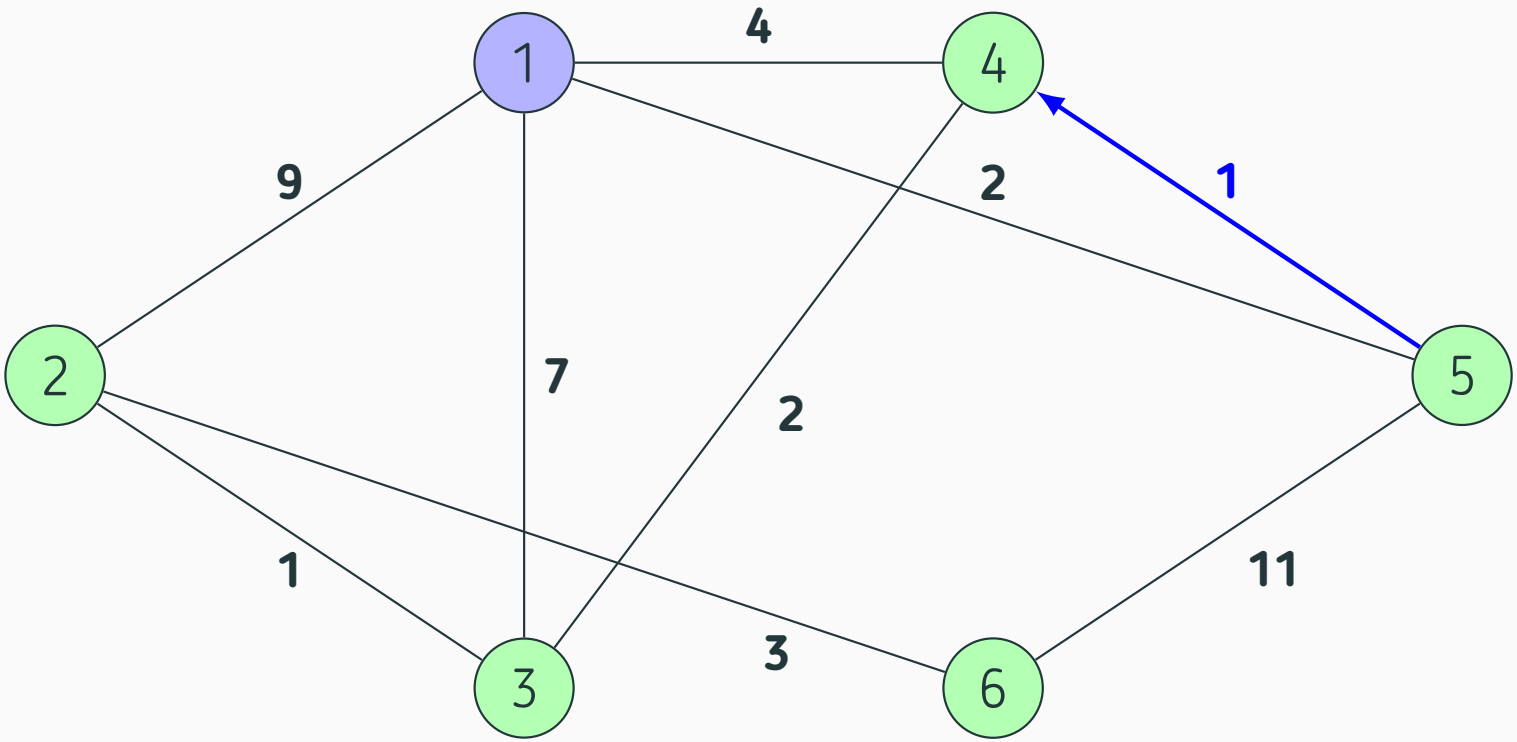
	1	2	3	4	5	6
Distâncias:	0	8	6	4	2	12

Visualização do algoritmo de Bellman-Ford



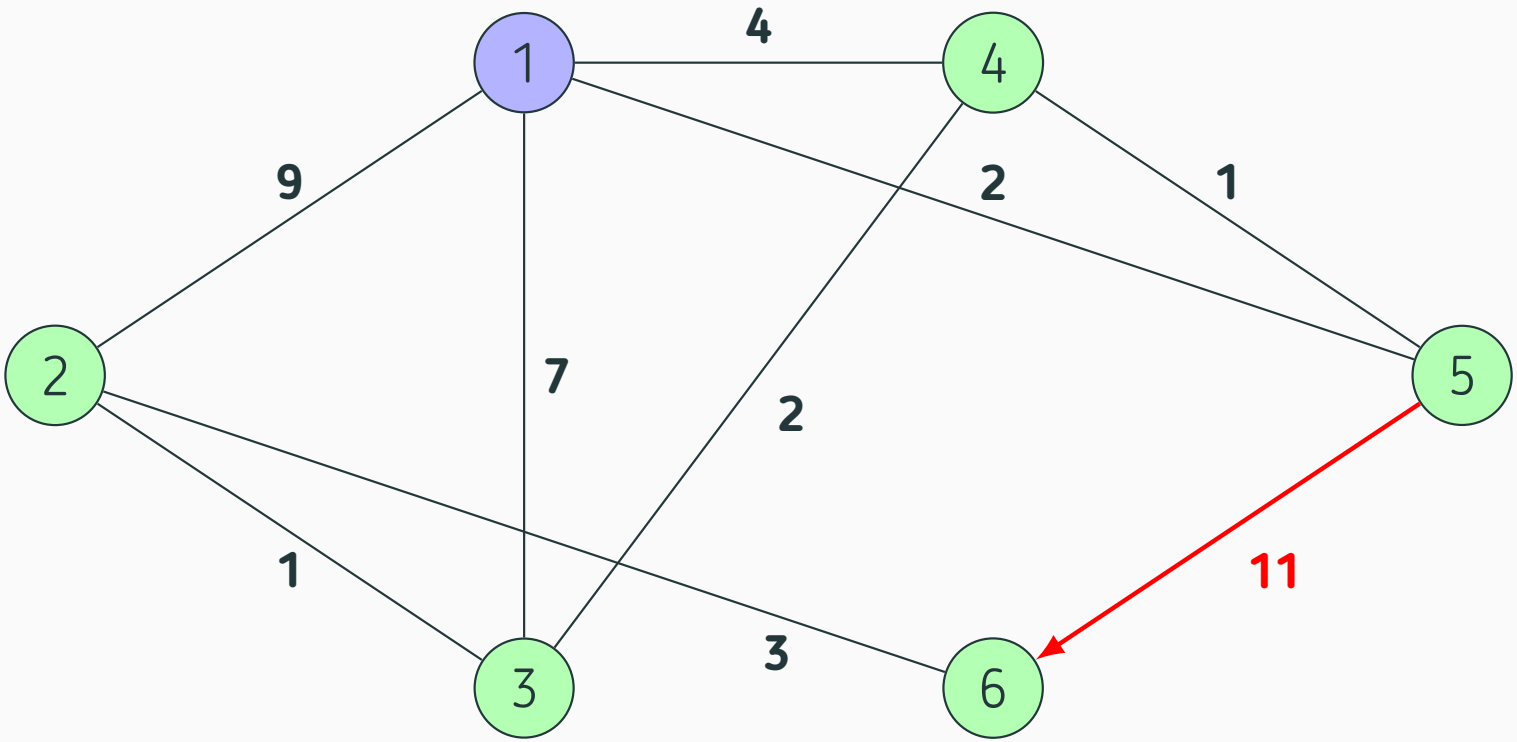
	1	2	3	4	5	6
Distâncias:	0	8	6	4	2	12

Visualização do algoritmo de Bellman-Ford



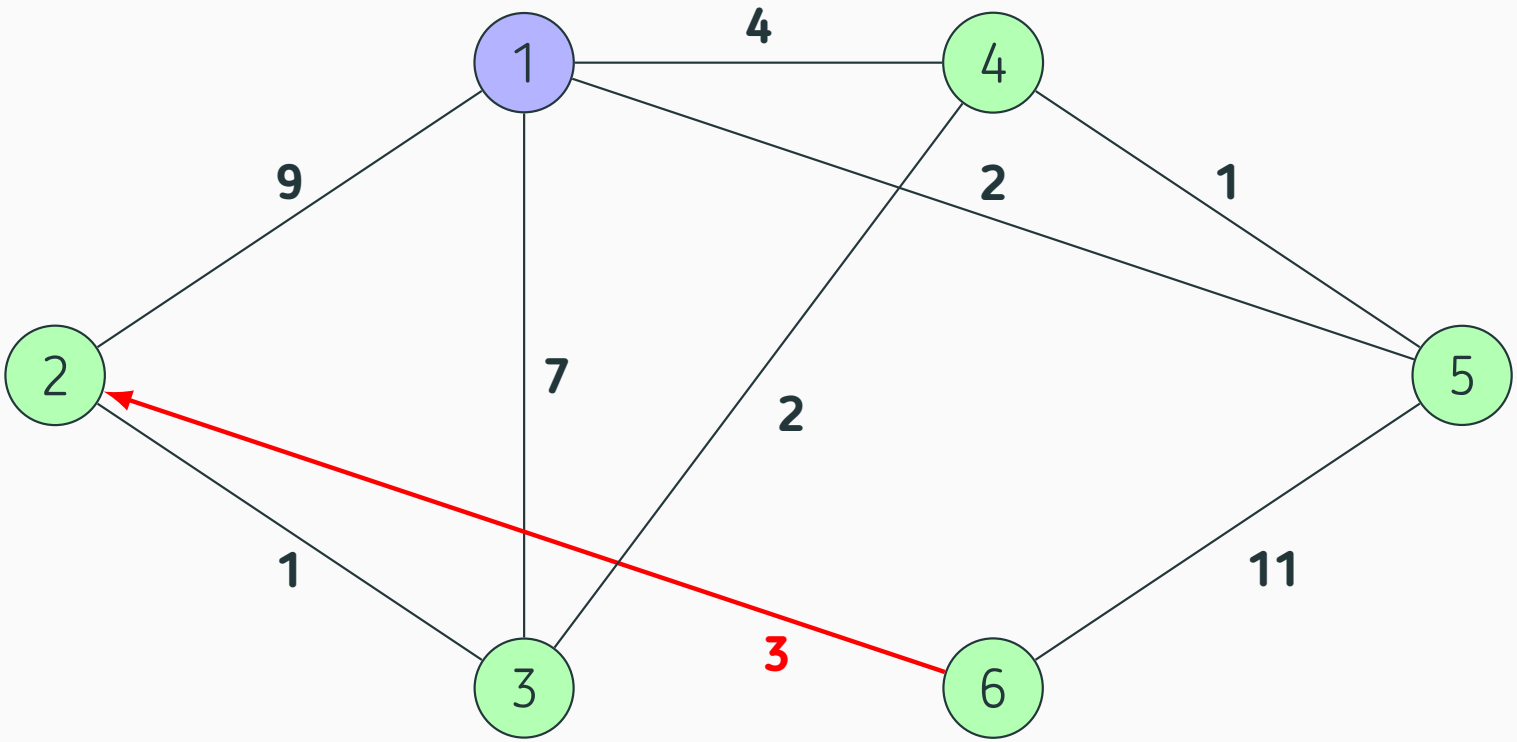
	1	2	3	4	5	6
Distâncias:	0	8	6	3	2	12

Visualização do algoritmo de Bellman-Ford



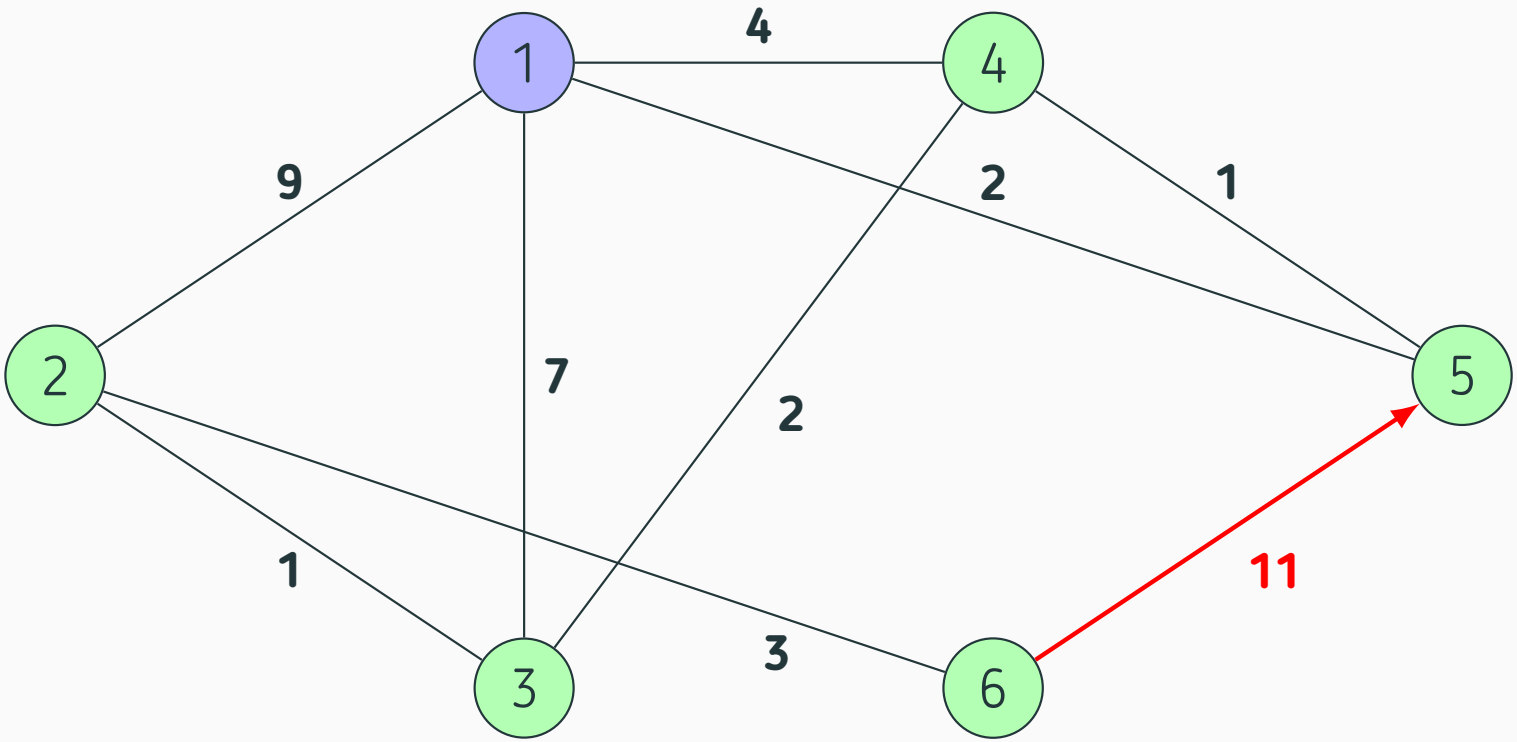
	1	2	3	4	5	6
Distâncias:	0	8	6	3	2	12

Visualização do algoritmo de Bellman-Ford



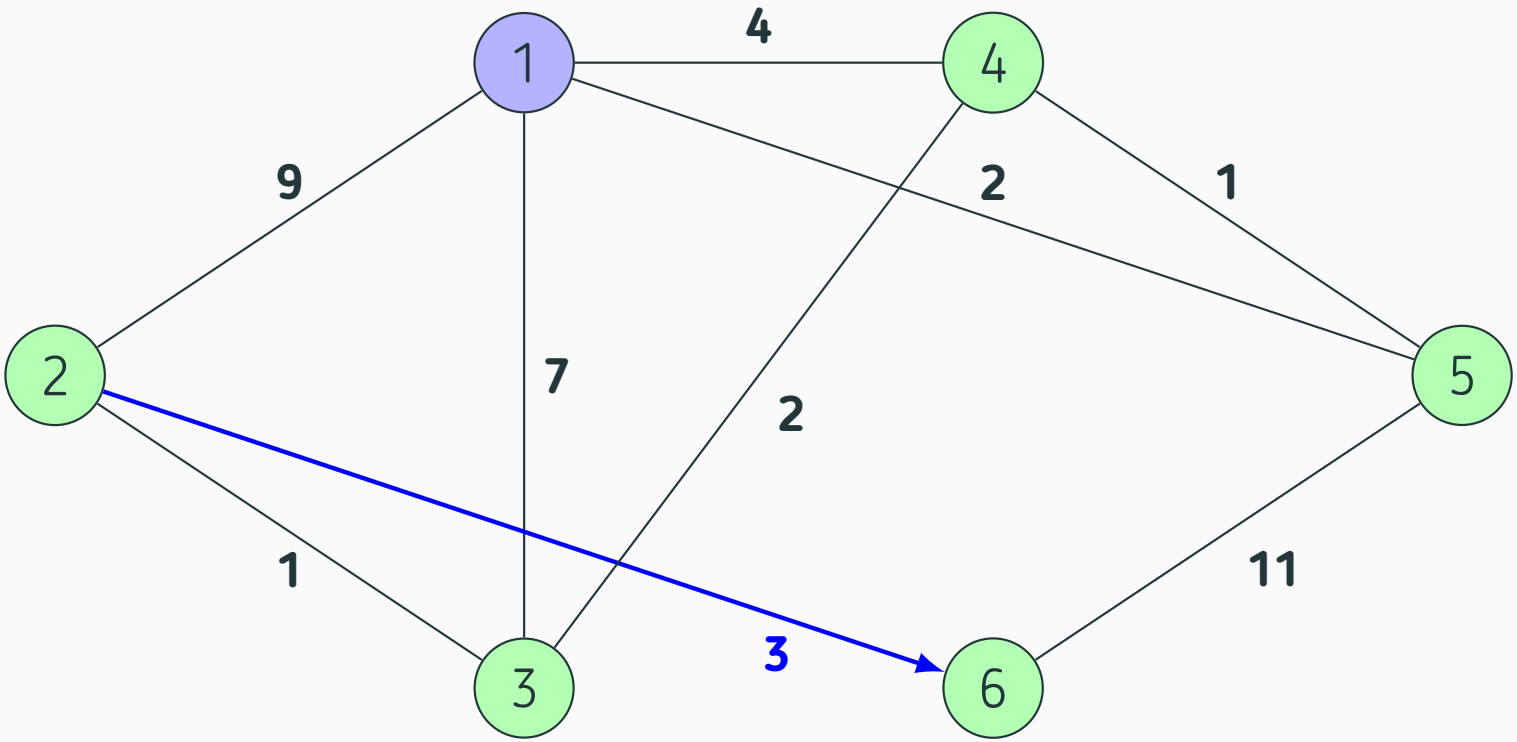
	1	2	3	4	5	6
Distâncias:	0	8	6	3	2	12

Visualização do algoritmo de Bellman-Ford



	1	2	3	4	5	6
Distâncias:	0	8	6	3	2	12

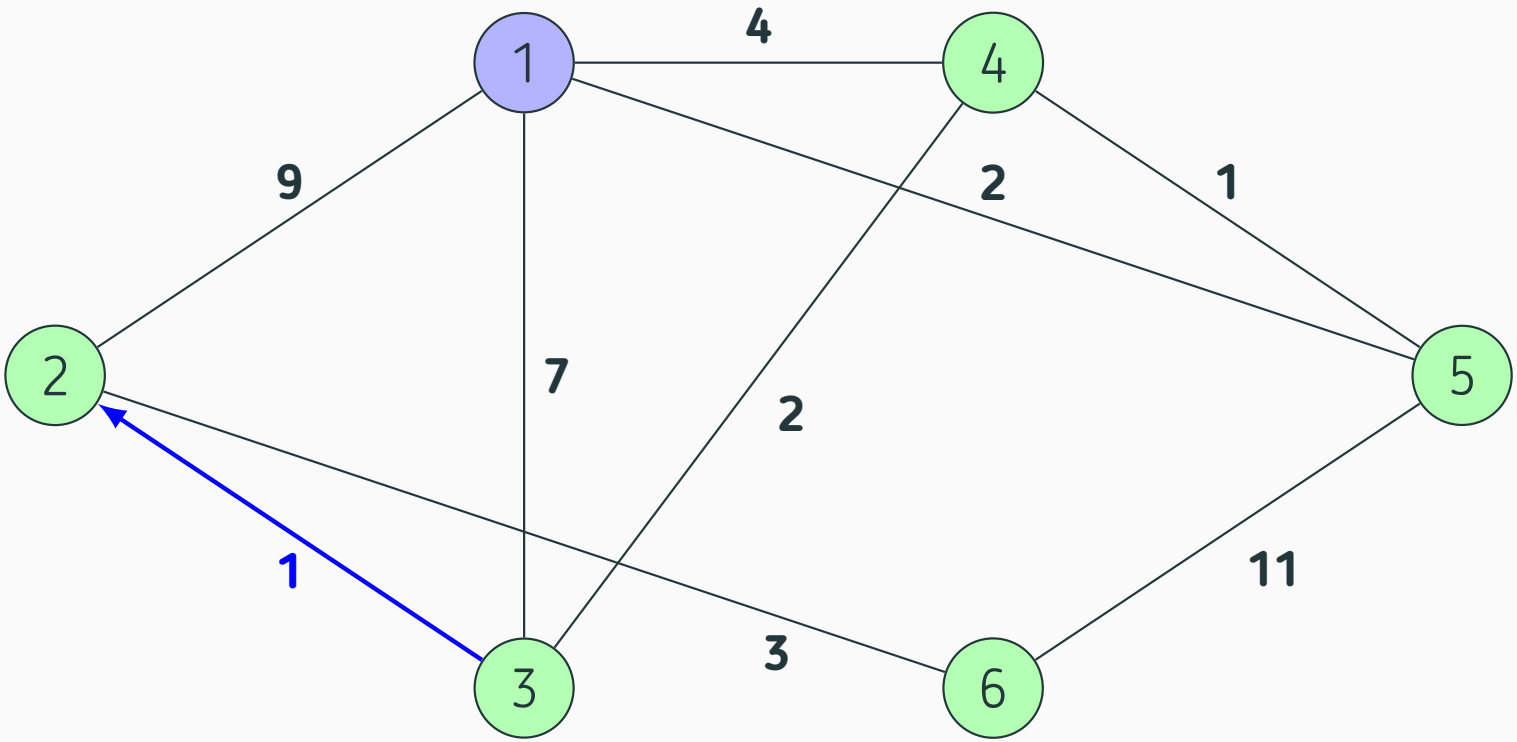
Visualização do algoritmo de Bellman-Ford



Round #2

	1	2	3	4	5	6
Distâncias:	0	8	6	3	2	11

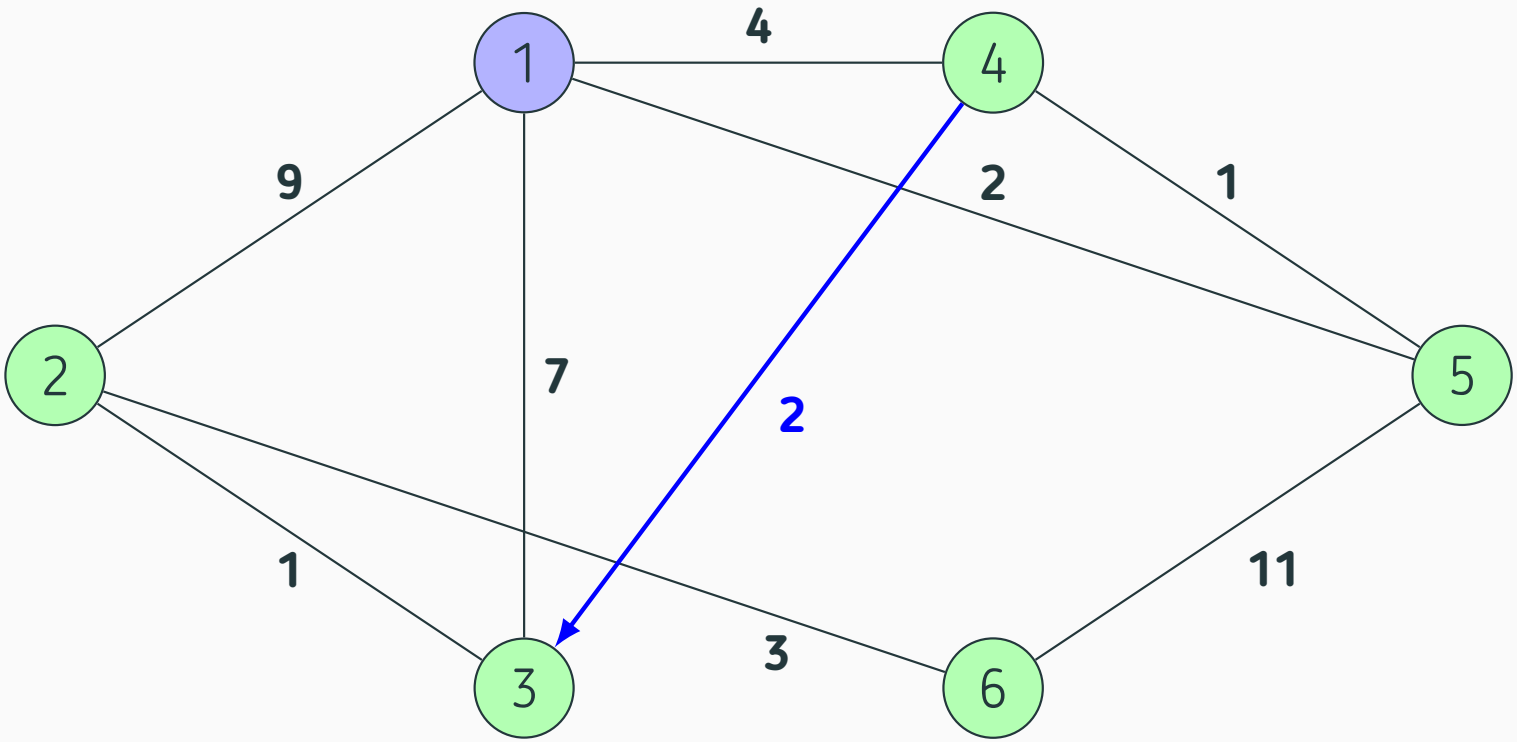
Visualização do algoritmo de Bellman-Ford



Round #2

	1	2	3	4	5	6
Distâncias:	0	7	6	3	2	11

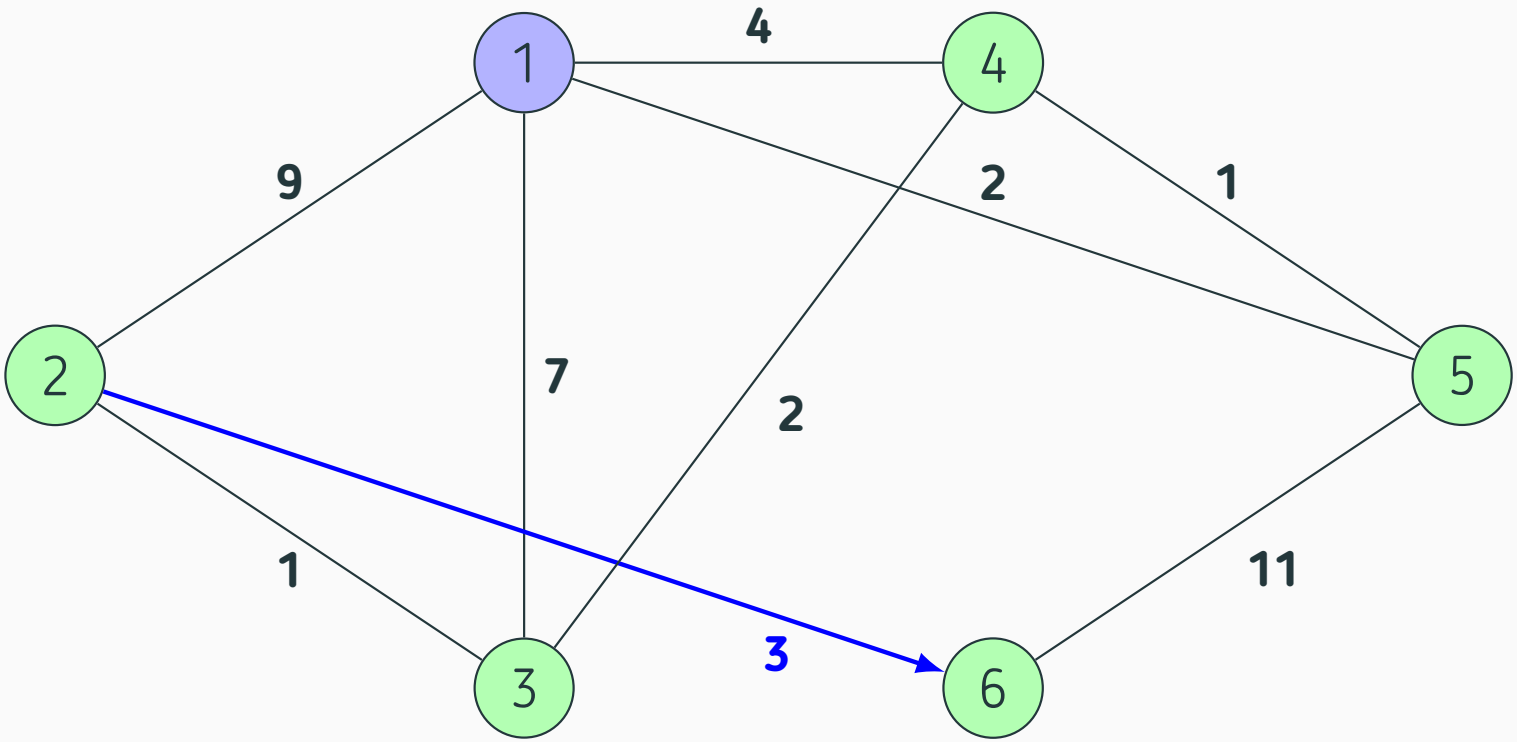
Visualização do algoritmo de Bellman-Ford



Round #2

	1	2	3	4	5	6
Distâncias:	0	7	5	3	2	11

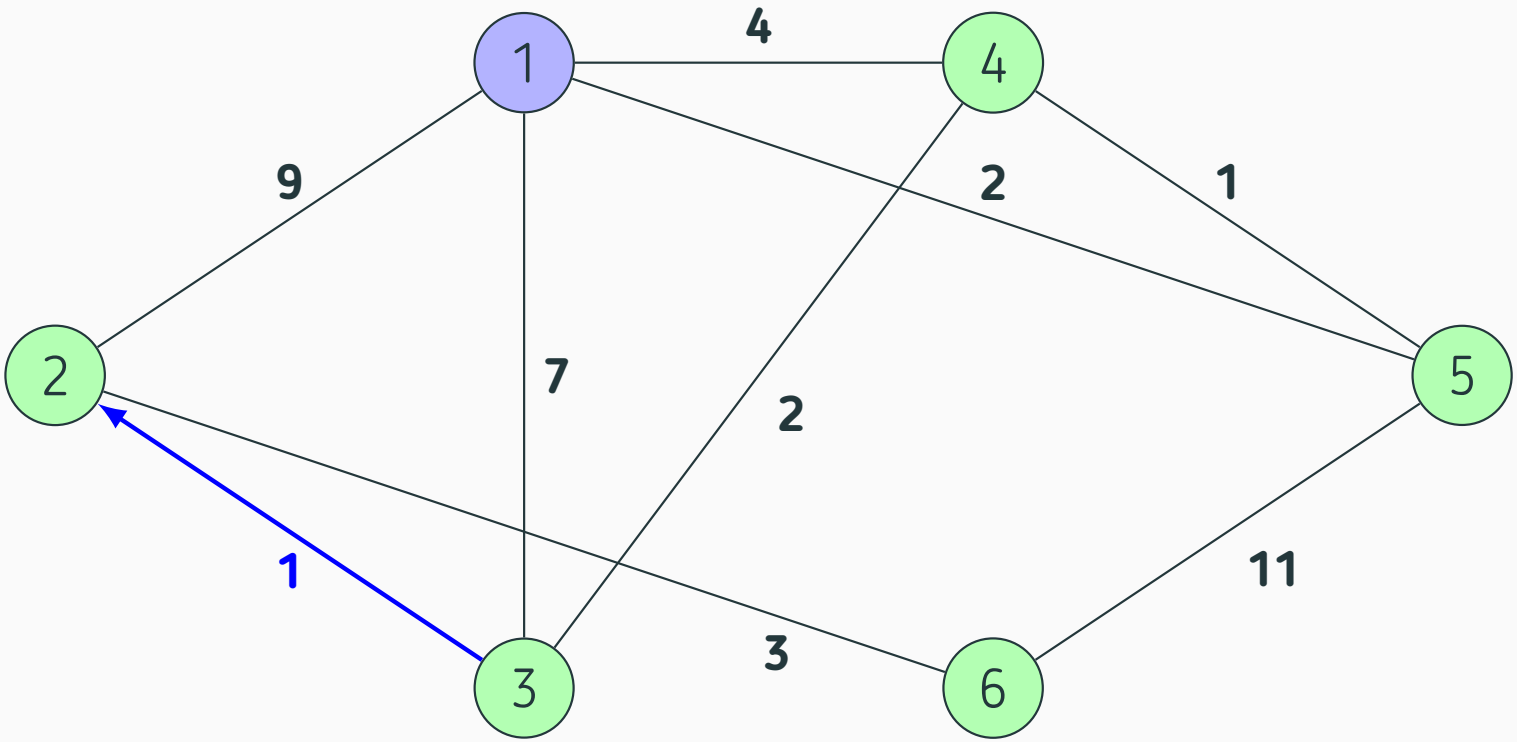
Visualização do algoritmo de Bellman-Ford



Round #3

	1	2	3	4	5	6
Distâncias:	0	7	5	3	2	10

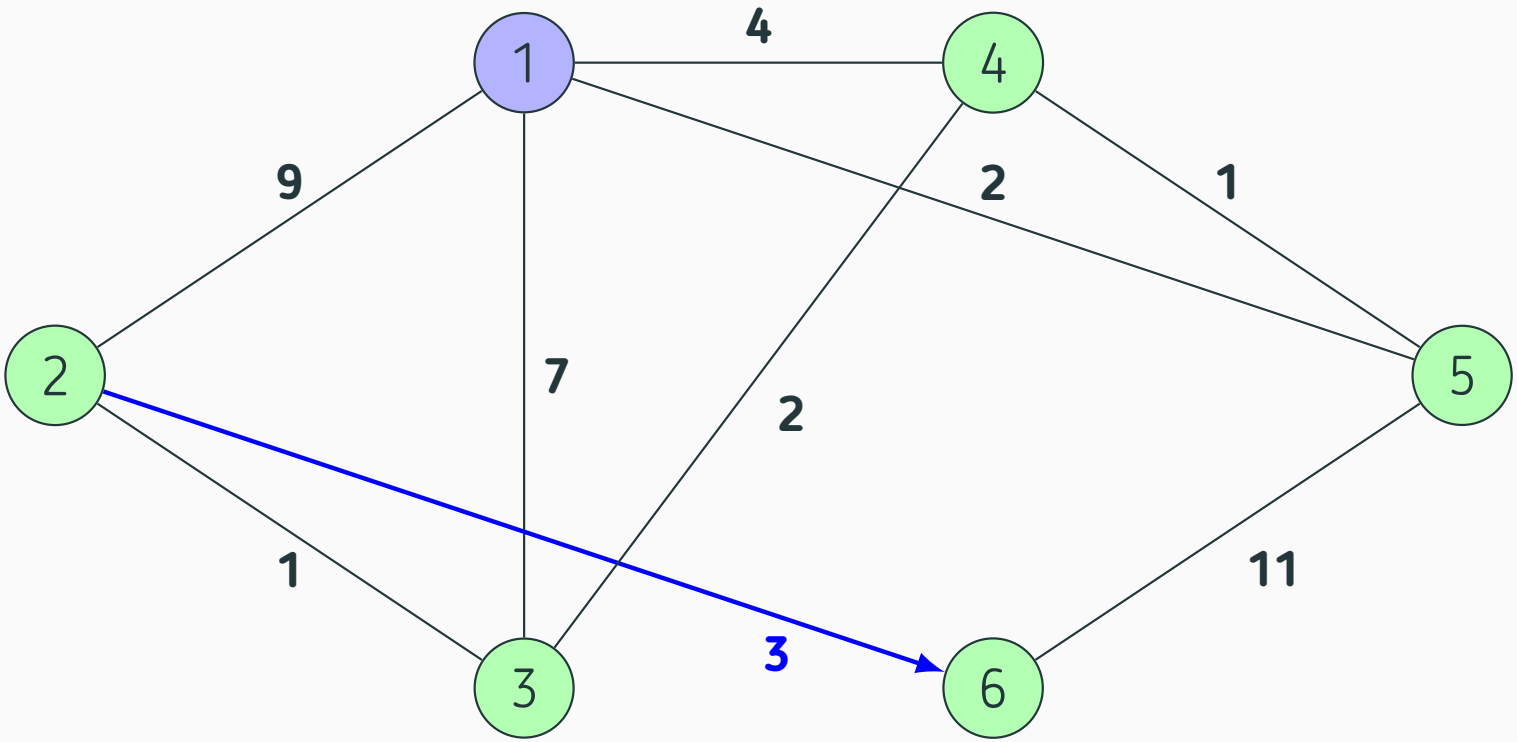
Visualização do algoritmo de Bellman-Ford



Round #3

	1	2	3	4	5	6
Distâncias:	0	6	5	3	2	10

Visualização do algoritmo de Bellman-Ford



Round #4

	1	2	3	4	5	6
Distâncias:	0	6	5	3	2	9

Solução $O(NM)$

```cpp
#include <bits/stdc++.h>

using namespace std;
struct Edge { int a, b, c; };

const int MAX { 110 }, oo { 1000000010 };
int dist[MAX];

int solve(int s, int N, const vector<Edge>& es)
{
    for (int i = 1; i <= N; ++i)
        dist[i] = oo;

    dist[s] = 0;

    for (int i = 1; i <= N - 1; i++)
        for (const auto& e : es)
            dist[e.b] = min(dist[e.b], dist[e.a] + e.c);

    return dist[N];
}
```

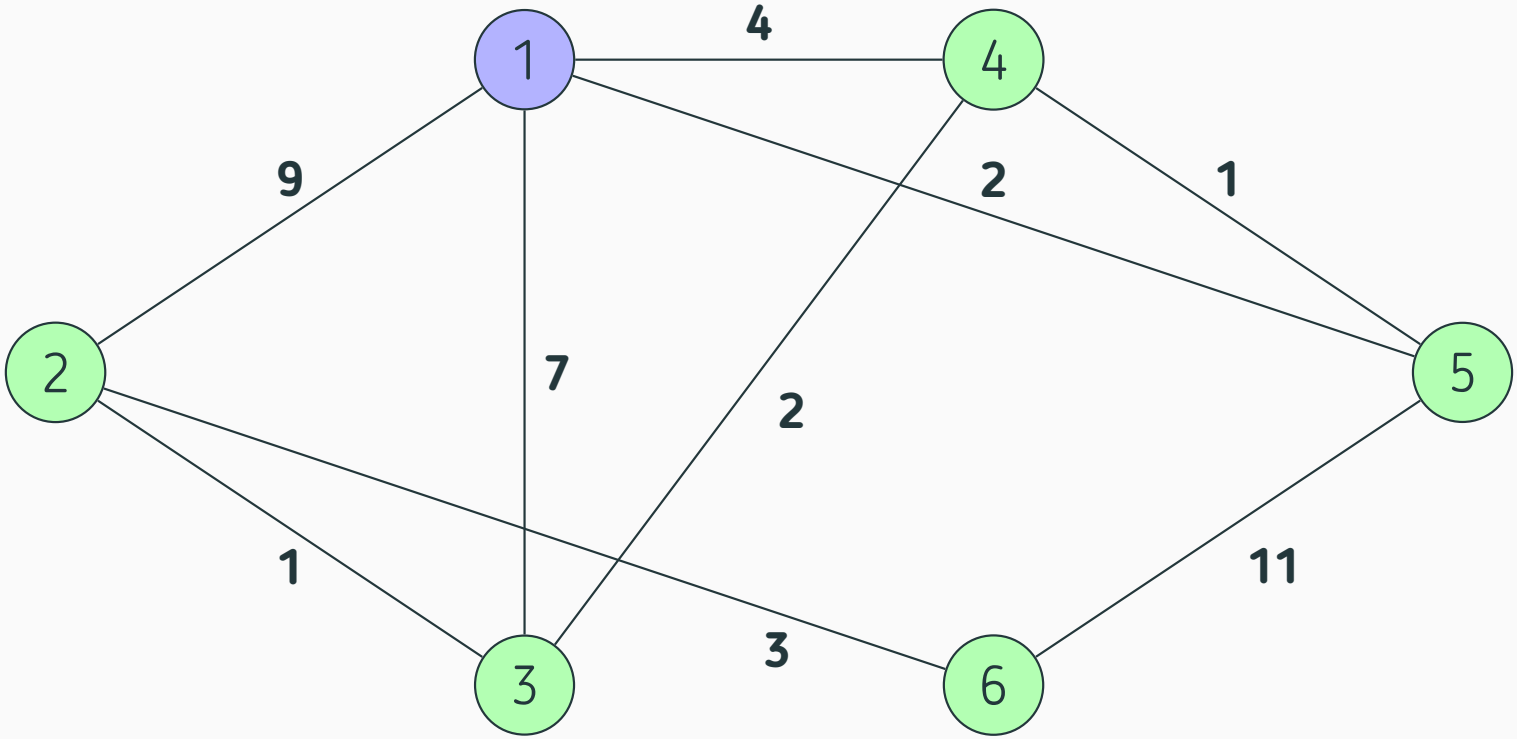
Solução $O(NM)$

```
22
23 int main()
24 {
25     int N, M;
26     cin >> N >> M;
27
28     vector<Edge> es(2*M);
29
30     for (int i = 0; i < M; ++i)
31     {
32         cin >> es[i].a >> es[i].b >> es[i].c;
33         es[M + i] = Edge { es[i].b, es[i].a, es[i].c };
34     }
35
36     auto ans = solve(1, N, es);
37
38     cout << ans << '\n';
39
40     return 0;
41 }
```

Algoritmo de Dijkstra

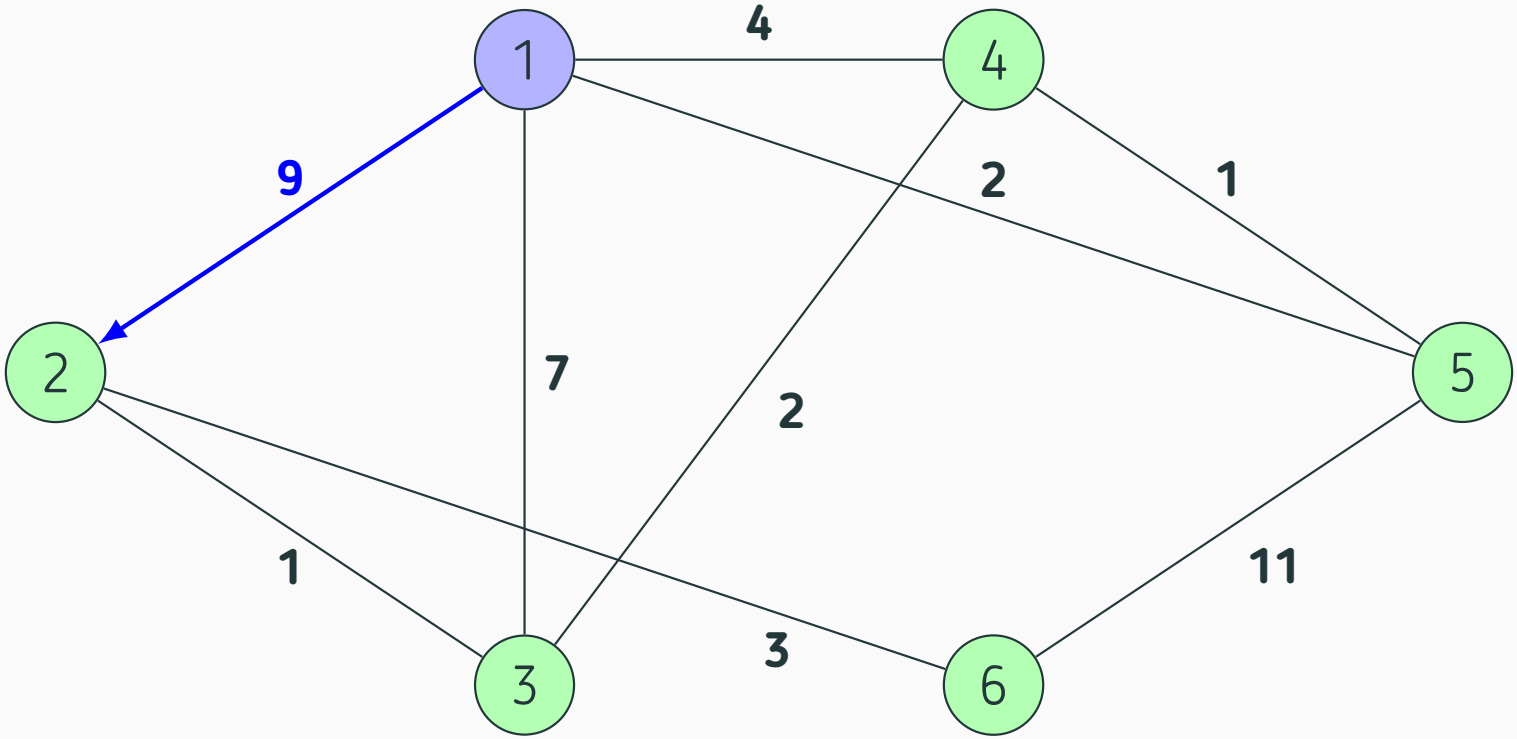
- Inicialmente, a distância de s a s é igual a zero, e todas as demais distâncias são iguais a infinito
- A cada iteração, o algoritmo escolhe o nó u mais próximo de s que ainda não foi processado
- Todas as arestas de partem de u então são processadas, atualizando as distâncias quando possível
- Esta operação de atualização de distância é chamada relaxamento
- Para escolher o próximo nó a ser processado de forma eficiente pode ser utilizada uma fila com prioridade
- Desta forma, a complexidade do algortimo é $O(N + M \log M)$

Visualização do algoritmo de Dijkstra



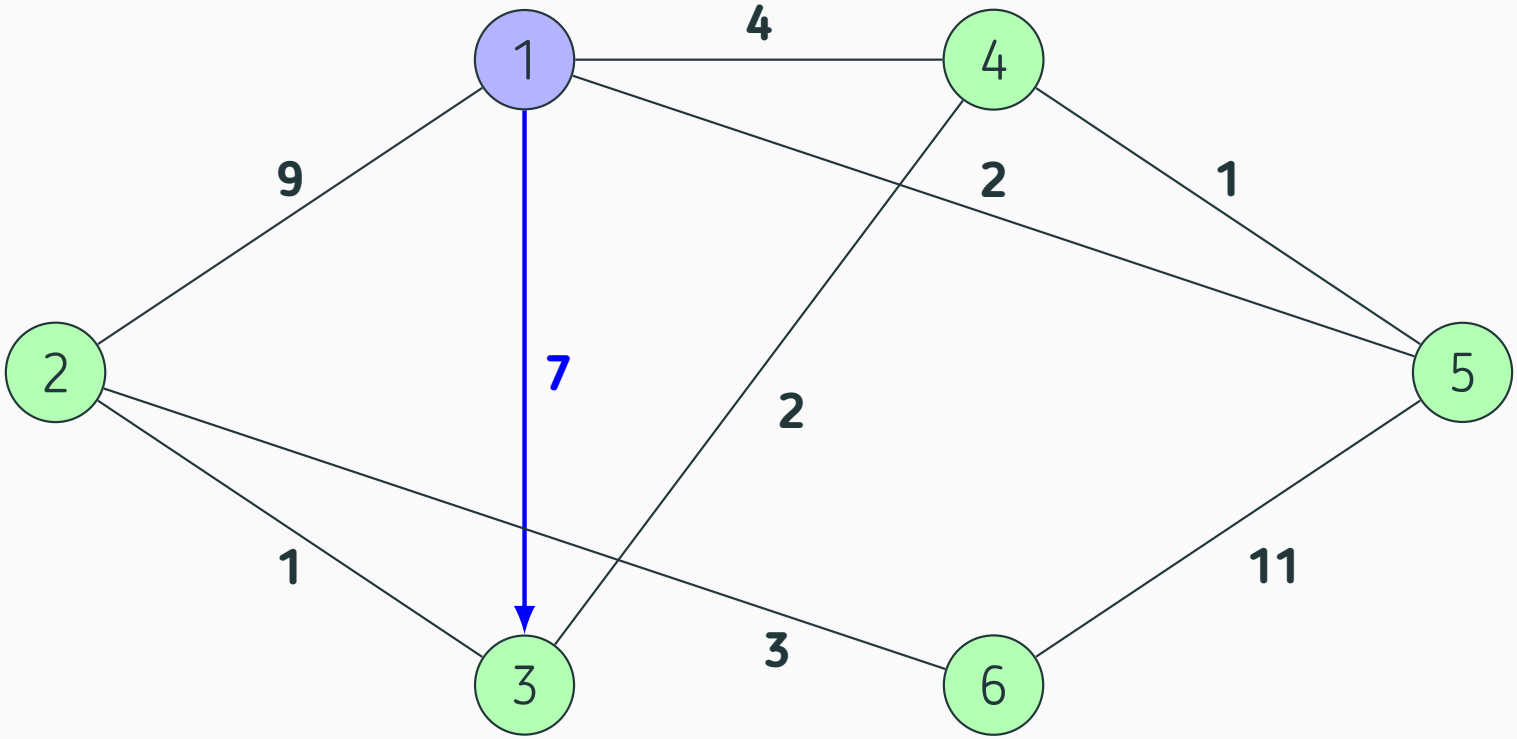
	1	2	3	4	5	6
Distância ao nó 1:	0	∞	∞	∞	∞	∞

Visualização do algoritmo de Dijkstra



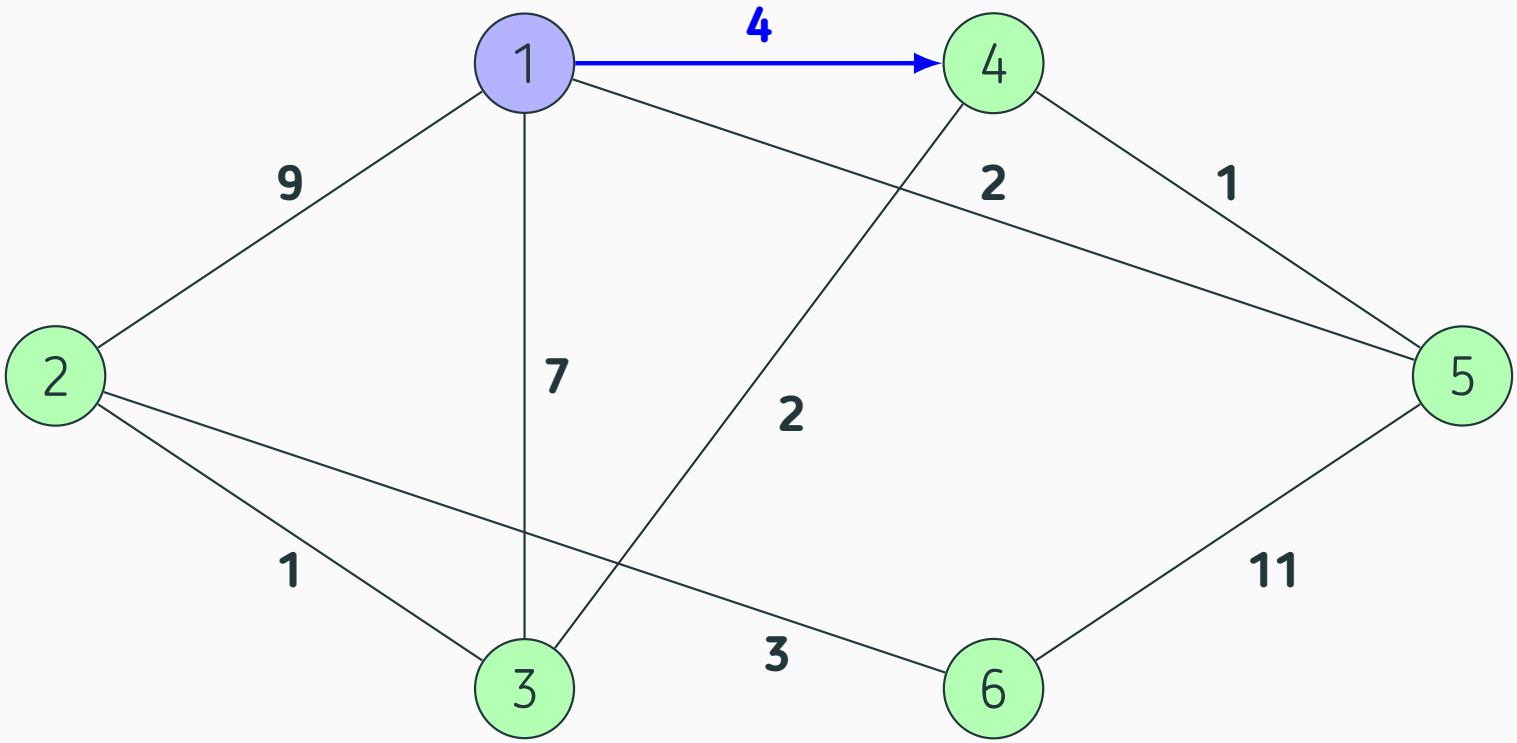
	1	2	3	4	5	6
Distância ao nó 1:	0	9	∞	∞	∞	∞

Visualização do algoritmo de Dijkstra



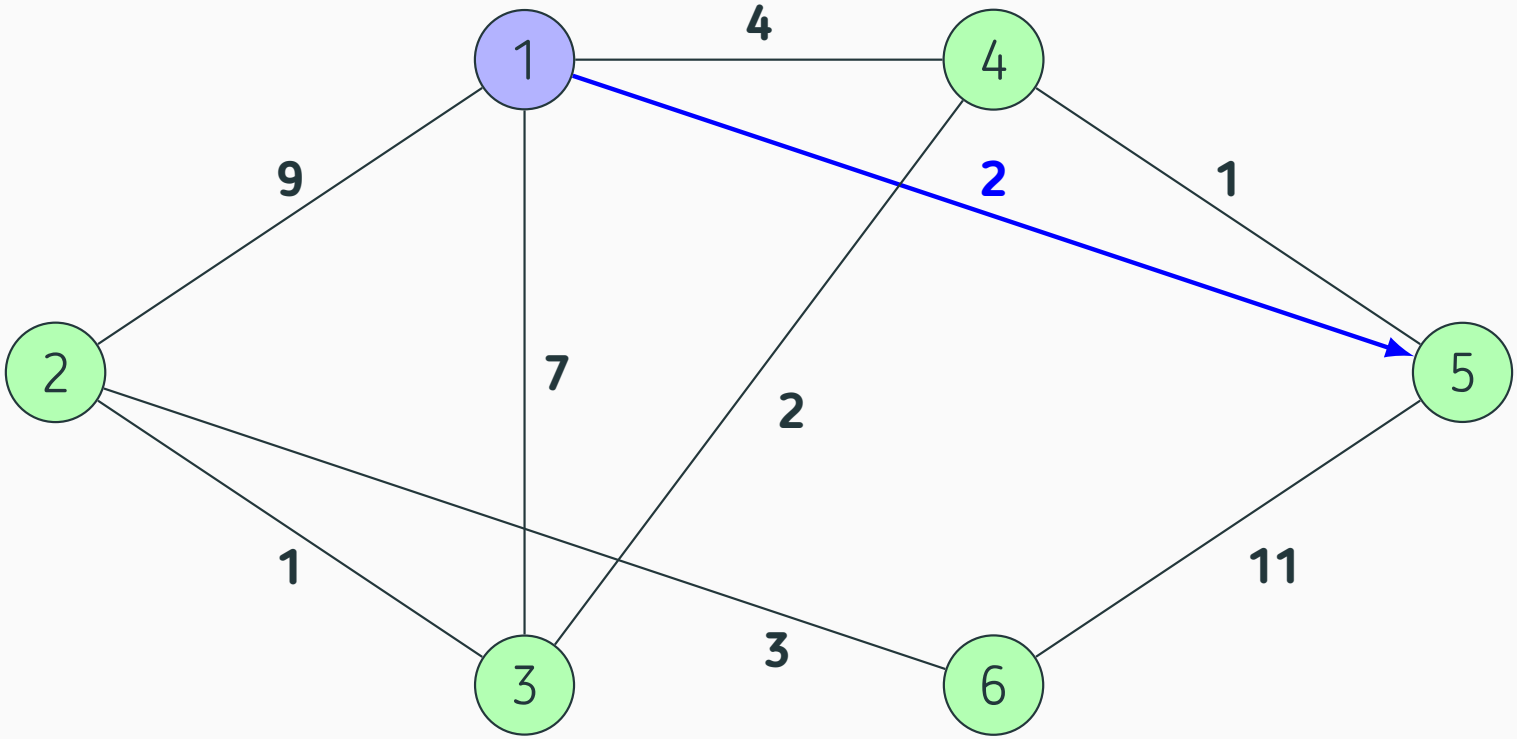
	1	2	3	4	5	6
Distância ao nó 1:	0	9	7	∞	∞	∞

Visualização do algoritmo de Dijkstra



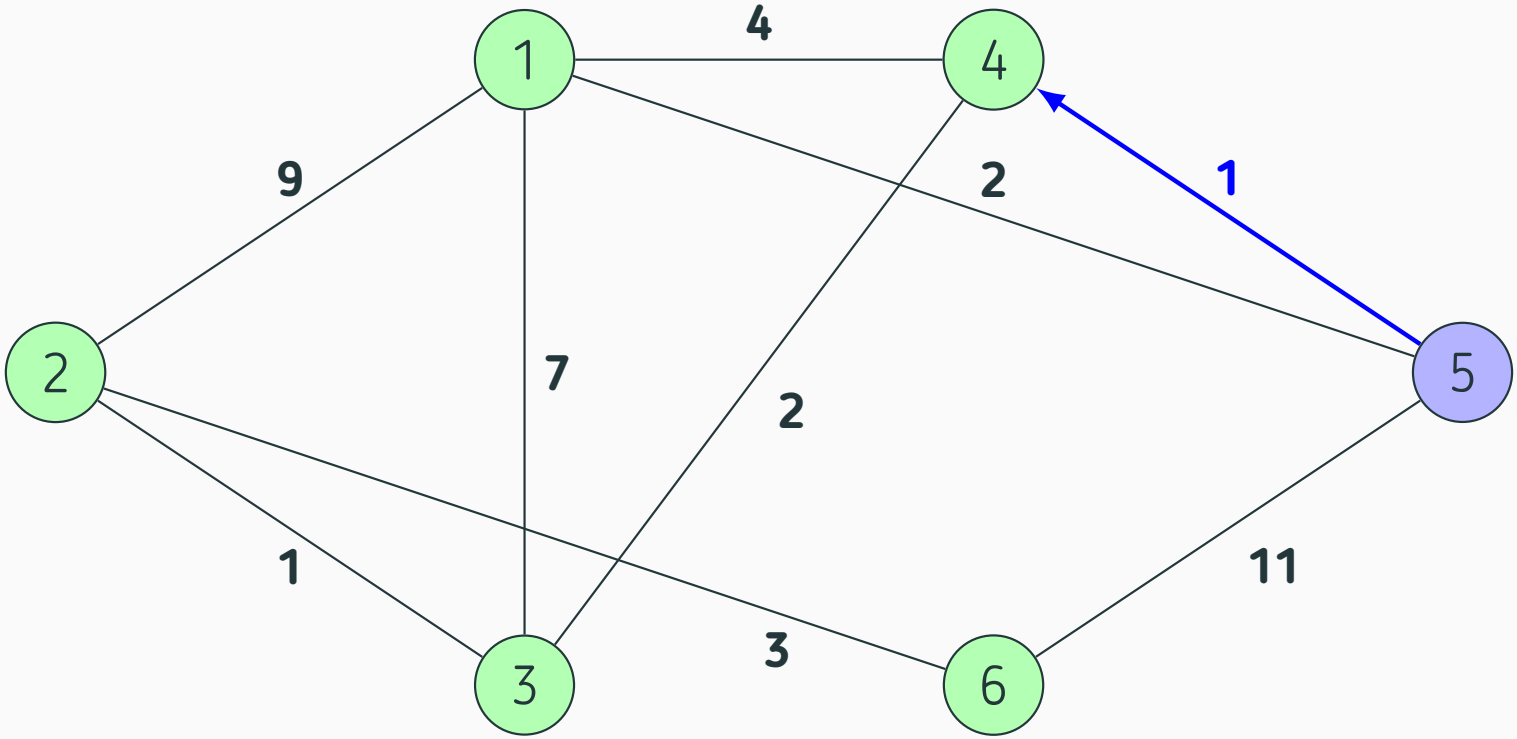
	1	2	3	4	5	6
Distância ao nó 1:	0	9	7	4	∞	∞

Visualização do algoritmo de Dijkstra



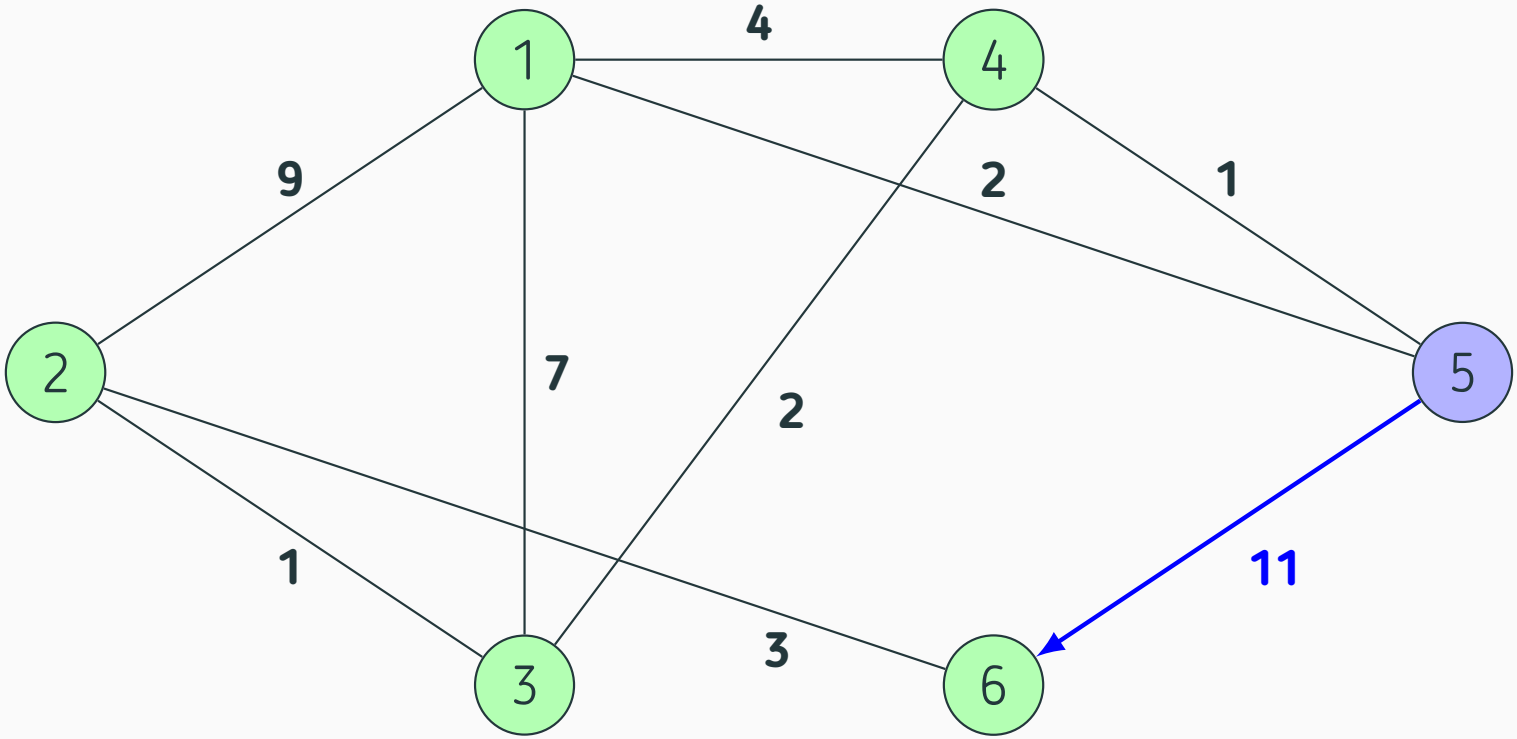
	1	2	3	4	5	6
Distância ao nó 1:	0	9	7	4	2	∞

Visualização do algoritmo de Dijkstra



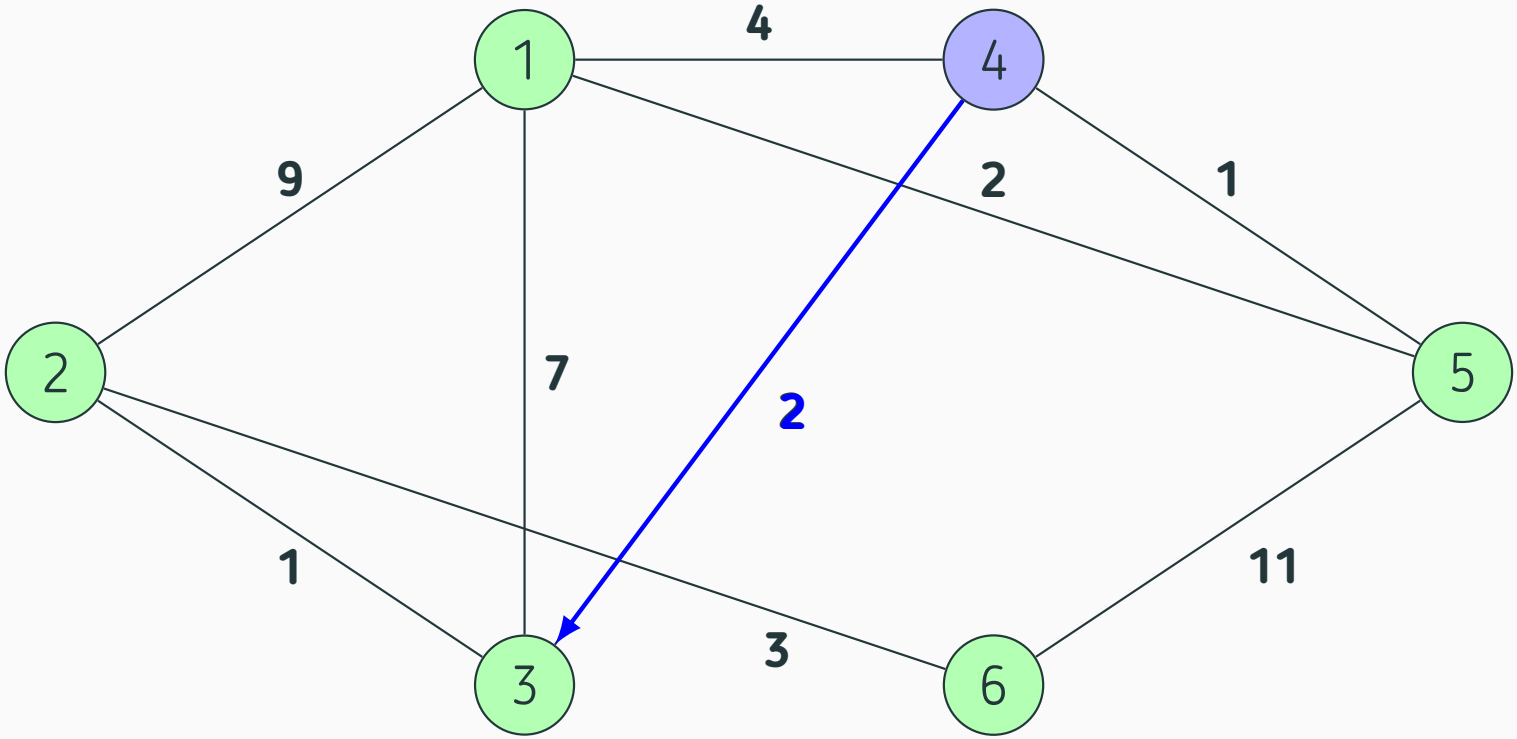
	1	2	3	4	5	6
Distância ao nó 1:	0	9	7	3	2	∞

Visualização do algoritmo de Dijkstra



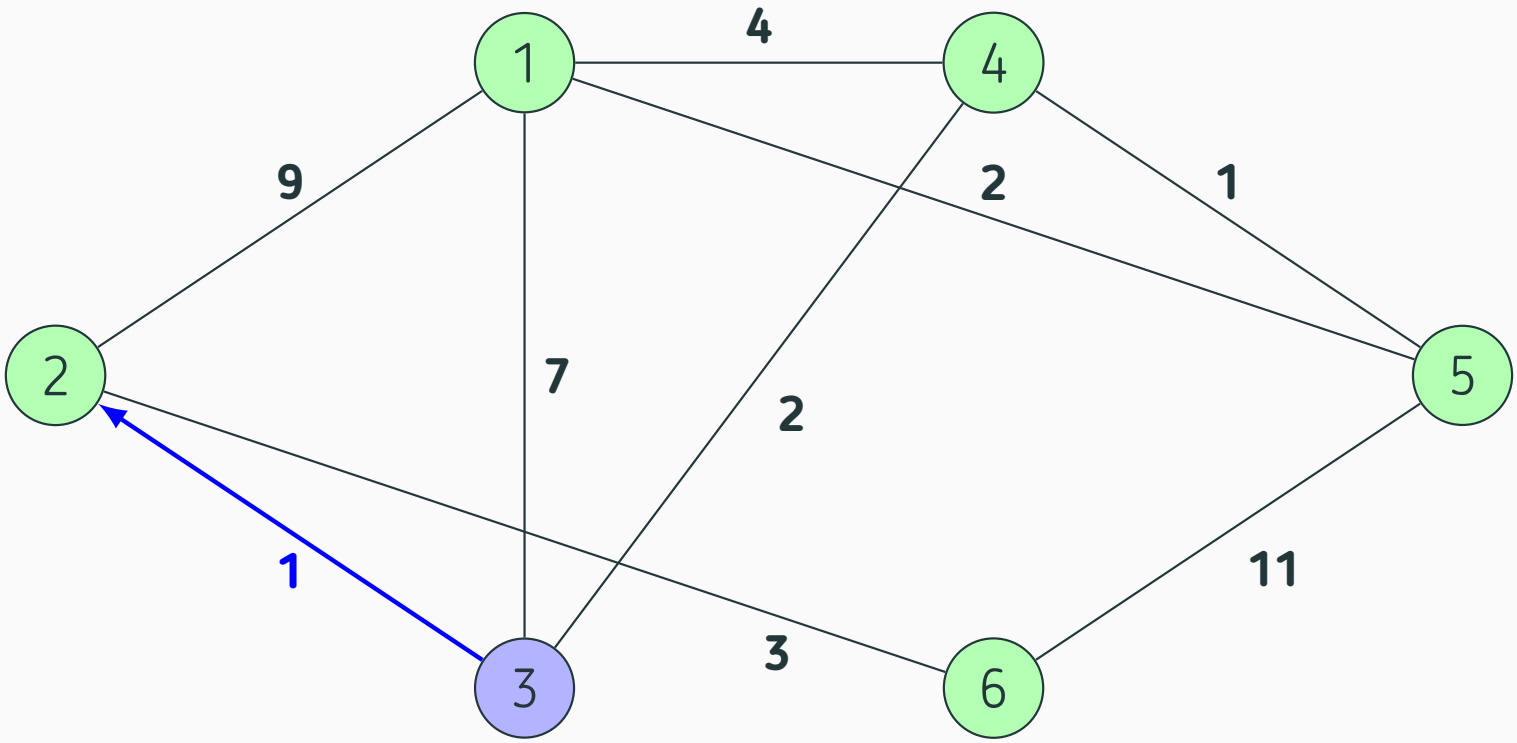
	1	2	3	4	5	6
Distância ao nó 1:	0	9	7	3	2	13

Visualização do algoritmo de Dijkstra



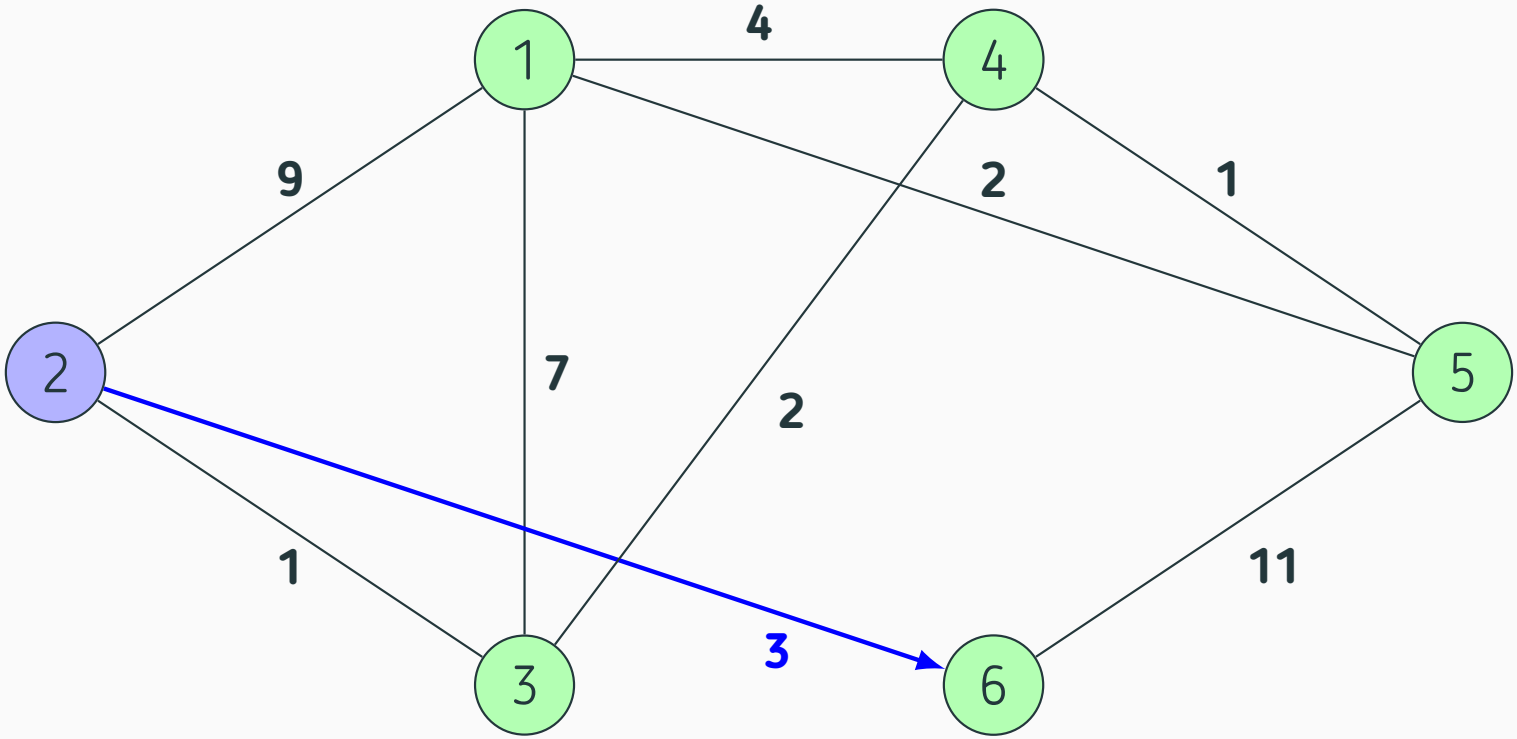
	1	2	3	4	5	6
Distância ao nó 1:	0	9	5	3	2	13

Visualização do algoritmo de Dijkstra



	1	2	3	4	5	6
Distância ao nó 1:	0	6	5	3	2	13

Visualização do algoritmo de Dijkstra



	1	2	3	4	5	6
Distância ao nó 1:	0	6	5	3	2	9

Solução $O(N + M \log M)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;

const int MAX { 110 }, oo { 1000000010 };

int dist[MAX];
vector<ii> adj[MAX];
bitset<MAX> processed;

void dijkstra(int s, int N)
{
    for (int i = 1; i <= N; ++i)
        dist[i] = oo;

    dist[s] = 0;
    processed.reset();

    priority_queue<ii, vector<ii>, greater<ii>> pq;
    pq.push(ii(0, s));
```

Solução $O(N + M \log M)$

```
23      while (not pq.empty())
24      {
25          auto d = pq.top().first, u = pq.top().second;
26          pq.pop();
27
28          if (processed[u])
29              continue;
30
31          processed[u] = true;
32
33          for (const auto& e : adj[u])
34          {
35              auto v = e.first, w = e.second;
36
37              if (dist[v] > d + w) {
38                  dist[v] = d + w;
39                  pq.push(ii(dist[v], v));
40              }
41          }
42      }
43  }
```

Solução $O(N + M \log M)$

```
45 int solve(int N)
46 {
47     dijkstra(1, N);
48     return dist[N];
49 }
50
51 int main()
52 {
53     ios::sync_with_stdio(false);
54
55     int N, M;
56     cin >> N >> M;
57
58     while (M--)
59     {
60         int a, b, c;
61         cin >> a >> b >> c;
62
63         adj[a].push_back(ii(b, c));
64         adj[b].push_back(ii(a, c));
65     }
```

Solução $O(N + M \log M)$

```
66
67      auto ans = solve(N);
68
69      cout << ans << '\n';
70
71      return 0;
72  }
```

Mapa

Problema

Harry ganhou um mapa mágico no qual ele pode visualizar o trajeto realizado por seus amigos. Ele agora precisa de sua colaboração para, com a ajuda do mapa, determinar onde Hermione se encontra.

O mapa tem L linhas e C colunas de caracteres, que podem ser '`.`' (ponto), a letra '`o`' (minúscula) ou a letra '`H`' (maiúscula). A posição inicial de Hermione no mapa é indicada pela letra '`o`', que aparece exatamente uma vez no mapa. A letra '`H`' indica uma posição em que Hermione pode ter passado – o mapa é impreciso, e nem toda letra '`H`' no mapa representa realmente uma posição pela qual Hermione passou. Mas todas as posições pelas quais Hermione passou são representadas pela letra '`H`' no mapa.

Problema

A partir da posição inicial de Hermione, Harry sabe determinar a posição atual de sua amiga, apesar da imprecisão do mapa, porque eles combinaram que Hermione somente se moveria de forma que seu movimento apareceria no mapa como estritamente horizontal ou estritamente vertical (nunca diagonal). Além disso, Hermione combinou que não se moveria de forma a deixar que Harry tivesse dúvidas sobre seu caminho (por exemplo, Hermione não passa duas vezes pela mesma posição). Considere o mapa abaixo, com 6 linhas e 7 colunas:

Problema

	1	2	3	4	5	6	7
1	.	.	.	**H**	**H**	**H**	.
2	H	H	H	.	.	.	**H**
3	H	.	H	H	H	.	.
4	H	.	.	.	H	H	.
5	H	.	o	.	.	.	.
6	H	H	H	.	.	**H**	**H**

Problema

A posição inicial de Hermione no mapa é (5,3), e sua posição atual é (4,6). As posições marcadas em negrito ('**H**') são erros no mapa.

Dado um mapa e a posição inicial de Herminone, você deve escrever um programa para determinar a posição atual de Herminone.

Entrada e saída

Entrada

A primeira linha contém dois números inteiros L e C, indicando respectivamente o número de linhas e o número de colunas. Cada uma das seguintes L linhas contém C caracteres.

Saída

Seu programa deve produzir uma única linha na saída, contendo dois números inteiros: o número da linha e o número da coluna da posição atual de Hermione.

Entrada e saída

Restrições

- $2 \leq L, C \leq 100$
- Apenas os caracteres '.', 'o' e 'H' aparecem no mapa.
- A letra 'o' aparece exatamente uma vez no mapa.
- A letra 'H' aparece ao menos uma vez no mapa.
- O caminho de Hermione está totalmente contido no mapa.
- Na posição da letra 'o' no mapa, há apenas uma letra 'H' como vizinho imediato na vertical ou horizontal.
- Na posição atual de Hermione no mapa, há apenas uma letra 'H' como vizinho imediato na vertical ou horizontal.
- Em cada uma das posições intermediárias do caminho de Hermione, há exatamente duas letras 'H' como vizinhas imediatas na vertical ou horizontal.

Entrada e saída

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 20 pontos, $N \leq 8$

Exemplo de entradas e saídas

Entrada

```
3 4
HHHH
H...
o.HH
```

Saída

```
1 4
```

Exemplo de entradas e saídas

Entrada

```
6 7
...HHH.
HHH....
H.HHH..
H...HH.
H.o....
HHH.HH.
```

Saída

```
4 6
```

Solução

- O conjunto de restrições dadas para o problema simplifica a solução
- Isto porque as restrições delimitam um caminho único entre a posição inicial de Harry e a posição final de Hermione
- Assim, a partir da posição inicial, basta consultar os quatro vizinhos (norte, sul, leste e oeste) a procura de um 'H'
- De acordo as restrições, haverá apenas um vizinho com tal marcação
- Daí, basta se mover para esta posição e apagar o 'H', de modo que ele não seja mais levado em consideração

Solução $O(LC)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;

ii solve(int L, int C, vector<string>& M)
{
    int x = -1, y = -1;

    for (int i = 0; i < L; ++i)
        for (int j = 0; j < C; ++j)
            if (M[i][j] == 'o')
            {
                x = i;
                y = j;
            }

    const vector<ii> dirs { ii(0, -1), ii(0, 1), ii(1, 0), ii(-1, 0) };

```

Solução $O(LC)$

```
20      while (true)
21      {
22          bool move = false;
23
24          for (auto d : dirs)
25          {
26              auto dx = d.first, dy = d.second;
27              auto u = x + dx, v = y + dy;
28
29              if (0 <= u and u < L and 0 <= v and v < C and M[u][v] == 'H')
30              {
31                  M[u][v] = '.';
32                  move = true;
33                  x = u;
34                  y = v;
35              }
36          }
37
38          if (not move)
39              break;
40      }
```

Solução $O(LC)$

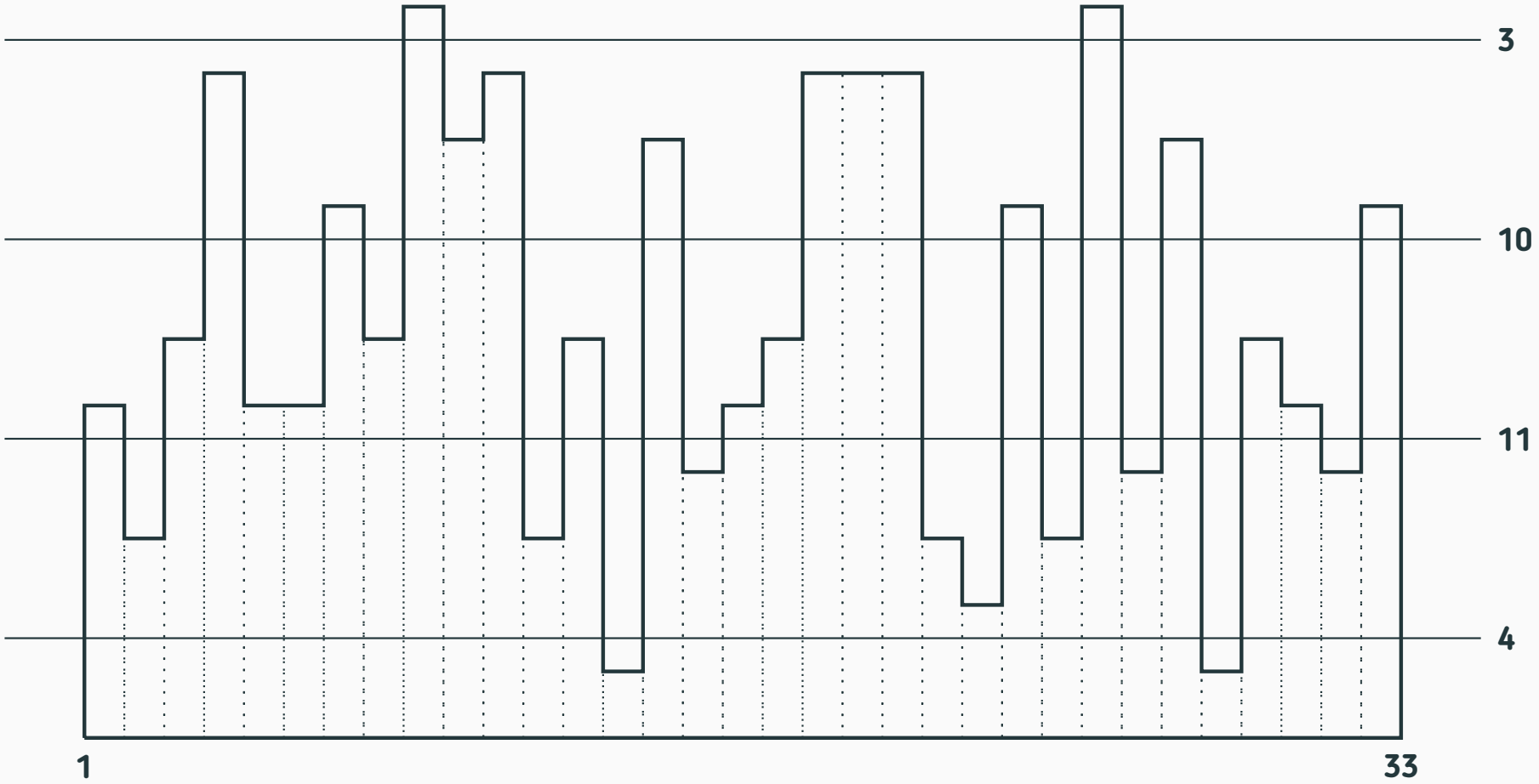
```
42      return ii(x + 1, y + 1);
43  }
44
45  int main()
46  {
47      ios::sync_with_stdio(false);
48
49      int L, C;
50      cin >> L >> C;
51
52      vector<string> M(L);
53
54      for (int i = 0; i < L; ++i)
55          cin >> M[i];
56
57      auto ans = solve(L, C, M);
58
59      cout << ans.first << ' ' << ans.second << '\n';
60
61      return 0;
62  }
```

Cortando o Papel

Problema

Uma folha de papel é composta de uma sequência de retângulos com diferentes alturas mas com larguras fixas, tal que as bases dos retângulos estão assentadas em uma linha horizontal. A figura ilustra uma folha exemplo com 33 retângulos. Nós gostaríamos de fazer um único corte horizontal, com a ajuda de um estilete e uma régua, que maximize o número resultante de pedaços separados pelo corte. A figura mostra quatro diferentes cortes que resultariam, respectivamente, em 4, 11, 10 e 3 pedaços.

Problema



Entrada e saída

Entrada

A primeira linha da entrada contém um inteiro N, representando o número de retângulos na folha de papel. A segunda linha contém N inteiros A_i, $1 \leq i \leq N$, representando a sequência de alturas dos retângulos.

Saída

Seu programa deve imprimir uma linha contendo um inteiro representando o número máximo de pedaços possível, com um único corte horizontal.

Entrada e saída

Restrições

- $2 \leq N \leq 10^5$
- $1 \leq A_i \leq 10^9$, para $1 \leq i \leq N$

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 40 pontos, $N \leq 1000$

Exemplo de entradas e saídas

Entrada	Saída
10 20 5 10 5 15 15 15 5 6 22	5
5 10 20 30 40 50	2

Solução

- Uma parte importante do problema consiste em identificar o número de pedaços resultantes de um corte na altura x
- Conforme pode ser observado na figura, um pedaço consiste em uma série de retângulos contíguos cujas alturas são todas maiores do que x
- Além disso, há o pedaço formado por todos os retângulos, ou partes de retângulo, que ficaram abaixo de x
- É possível usar a técnica de dois ponteiros para identificar os pedaços resultantes de um corte em x

Rotina que computa os pedaços para um corte em x

```
1  int pieces(double x, int N, const vector<int>& hs)
2  {
3      auto L = 0, res = 0;
4
5      while (L < N)
6      {
7          auto R = L + 1;
8
9          if (hs[L] > x)
10         {
11             while (R < N and hs[R] > x)
12                 ++R;
13
14             ++res;
15         }
16
17         L = R;
18     }
19
20     return res + 1;
21 }
```

Solução

- A rotina $pieces(x)$, que computa o número de pedaços resultantes de um corte em x, tem complexidade $O(N)$
- Uma solução para o problema consiste em computar o valor máximo de $pieces(x)$ para $x \in [1, M]$, onde $M = \max\{h_1, h_2, \ldots, h_N\}$
- Tal solução tem complexidade $O(NM)$, e como $N \leq 10^5$ e $M \leq 10^9$, tal abordagem resulta em um veredito TLE

Solução TLE $O(MN)$

```cpp
#include <bits/stdc++.h>

using namespace std;

int pieces(double x, int N, const vector<int>& hs)
{
    auto L = 0, res = 0;

    while (L < N)
    {
        if (hs[L] <= x)
        {
            ++L;
            continue;
        }

        auto R = L + 1;

        while (R < N and hs[R] > x)
            ++R;

```

Solução TLE $O(MN)$

```
22          ++res;
23          L = R;
24      }
25
26      return res + 1;
27  }
28
29  int solve(int N, const vector<int>& hs)
30  {
31      int M = *max_element(hs.begin(), hs.end()), ans = 2;
32
33      for (int i = 1; i <= M; ++i)
34          ans = max(ans, pieces(i, N, hs));
35
36     return ans;
37  }
38
39  int main()
40  {
41      ios::sync_with_stdio(false);
42
```

Solução TLE $O(MN)$

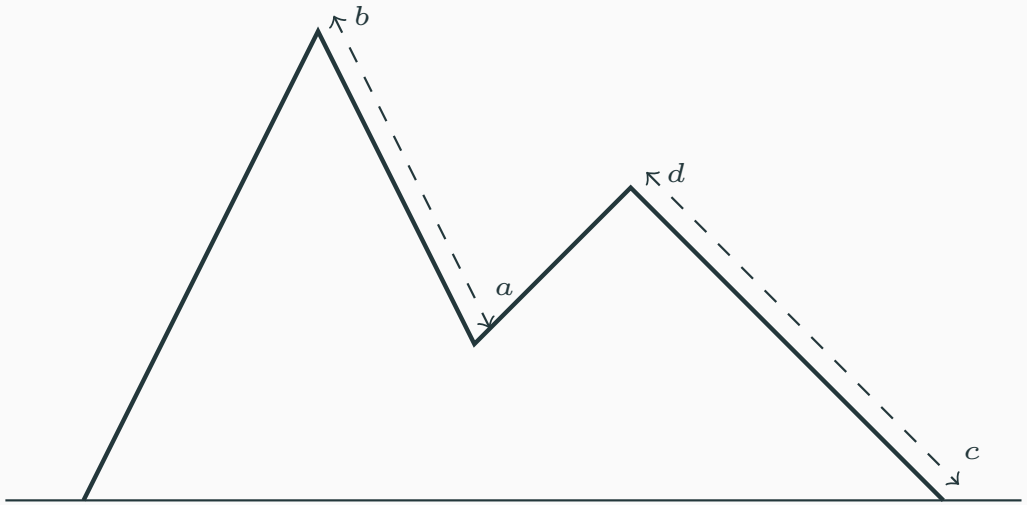
```
43      int N;
44      cin >> N;
45
46      vector<int> hs(N);
47
48      for (int i = 0; i < N; ++i)
49          cin >> hs[i];
50
51      auto ans = solve(N, hs);
52
53      cout << ans << '\n';
54
55      return 0;
56  }
```

Solução

- Por meio de uma observação cuidadosa dos cortes para valores consecutivos de x leva a conclusão que os valores de $pieces(x)$ só se modificam nos valores que correspondem às alturas dos retângulos
- Isto reduz a quantidade máxima de alturas a serem verificadas de 10^9 para 10^5
- Embora o veredito continue sendo TLE, esta redução permite resolver com sucesso o conjunto de casos de testes que correspondem a 40 pontos
- Para somar todos os 100 pontos, é preciso uma abordagem distinta que identifique, de forma eficiente, os valores de x que levam aos cortes com o número máximo de pedaços

Solução

- Considere a figura abaixo



- Um corte com $x \in [a, b)$ divide o primeiro pico em duas partes
- Já um corte com $x \in [c, d)$ divide o segundo pico em duas partes

Solução

- Agora, se $x \in [a, b) \cap [c, d)$, ambos serão divididos
- Considerando que o i-ésimo pico pode ser caracterizado por um ponto de máximo local b_i e um ponto de mínimo local a_i, o problema passa a ser determinar a maior interseção possível entre todos os intervalos $[a_i, b_i)$
- Para identificar de forma mais simples e eficiente os pontos de máximo e mínimo locais, a entrada pode ser comprimida, eliminando-se alturas iguais e consecutivas:

```
auto it = unique(hs.begin(), hs.end());

N = (int) distance(hs.begin(), it);
hs.resize(N);
```

Solução

- Após a compressão, há um ponto de máximo local em h_i se $h_{i-1} < h_i$ e $h_{i+1} < h_i$
- De forma análoga, há um ponco de mínimo local em h_i se $h_{i-1} > h_i$ e $hi + 1 > h_i$
- Por fim, a maior interseção possível entre todos os intervalos $[a_i, b_i)$ pode ser determinada por meio de um algoritmo de *sweep line*
- Para cada intervalo a_i deve ser criados dois eventos: um evento de abertura em a_i e um e fechamento em b_i
- Os eventos devem ser ordenados pelo ponto de ocorrência
- Em caso de empate, os eventos de fechamento devem vir antes dos de abertura

Solução

- Os eventos devem ser processados em ordem
- Inicialmente não há nenhum intervalo aberto
- Cada abertura incrementa em uma unidade o número de intervalos abertos
- Os eventos de fechamento reduzem o número de intervalos abertos em uma unidade
- A solução do problema é o maior número de intervalos abertos registrado durante o processamento dos eventos
- Assim, como a compressão tem complexidade $O(N)$ e a ordenação dos eventos $O(N \log N)$, a solução tem complexidade $O(N \log N)$

Solução $O(N \log N)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;

int solve(int N, vector<int>& hs)
{
    // Compressão dos adjacentes iguais
    auto it = unique(hs.begin(), hs.end());

    N = (int) distance(hs.begin(), it);
    hs.resize(N);

    // Identificação dos pontos críticos
    vector<int> cs;

    for (int i = 1; i <= N - 2; ++i)
        if ((hs[i - 1] < hs[i] and hs[i] > hs[i + 1]) ||
            (hs[i - 1] > hs[i] and hs[i] < hs[i + 1]))
                cs.push_back(hs[i]);

```

Solução $O(N \log N)$

```
22      // Finaliza com um mínimo
23      if (cs.size() % 2)
24          cs.push_back(0);
25
26      // Geração dos eventos para o sweep line
27      vector<ii> es;
28      int OPEN = 1, CLOSE = -1, ans = 1, open = 0;
29
30      for (size_t i = 0; i < cs.size(); i += 2)
31      {
32          es.push_back(ii(cs[i + 1], OPEN));
33          es.push_back(ii(cs[i], CLOSE));
34      }
35
36      sort(es.begin(), es.end());
37
38      for (auto e : es)
39      {
40          open += e.second;
41          ans = max(ans, open);
42      }
```

Solução $O(N \log N)$

```
43
44      return ans + 1;
45  }
46
47  int main()
48  {
49      ios::sync_with_stdio(false);
50
51      int N;
52      cin >> N;
53
54      vector<int> hs(N + 2, 0);
55
56      for (int i = 1; i <= N; ++i)
57          cin >> hs[i];
58
59      cout << solve(N, hs) << '\n';
60
61      return 0;
62  }
```

Referências

1. Dario e Xerxes
2. Frete
3. Mapa
4. Cortando o Papel