



**1º**

**COMPETIÇÃO**

**DE PROGRAMAÇÃO**

○ *<!--Conq Hub de Inovação-->*

**13/08/2022**

Hub de Inovação

- Um evento de alto nível onde estudantes de graduação de instituições de ensino da nossa região serão desafiados a resolver 10 etapas de programação em ambiente controlado, sem acesso à internet e
- nem uso de celular.

# Patrocinadores

**mstech®**

**witzler**

 **FIREWORK**

 **eSapiens**

 **olabs**

  
**GENESIS**

 **FOCUS**  
consult

  
**DATA GUVI**  
Business Intelligence

**EZ.devs**



## O que encontro na Conq?



### NETWORKING

Players conectados ao ecossistema de inovação e tecnologia



WHATSAPP!  
(14)98814-7915



### ECONOMIA

Planos acessíveis a partir de R\$ 250,00



VENHA SER UM(A)  
CONQER



### FLEXIBILIDADE

Mais de 250 estações de trabalho em estações abertas e privativas













CONHEÇA NOSSOS  
produtos / serviços











# Participantes



# Estatísticas

<b>Problems</b>	<b>Total</b>	<b>Accepted</b>
A 	20	15 (75%)
B 	18	1 (6%)
C 	20	2 (10%)
D 	20	0 (0%)
E 	0	0
F 	34	2 (6%)
G 	54	3 (6%)
H 	4	2 (50%)
I 	4	2 (50%)
J 	18	2 (11%)

# Estatísticas

<b>Problems x Answers</b>	<b>NO - Compilation error</b>	<b>NO - Runtime error</b>	<b>NO - Time limit exceeded</b>	<b>NO - Wrong answer</b>	<b>YES</b>	<b>Total</b>
A 	0	0	0	5 (25%)	15 (75%)	20
B 	0	0	0	17 (94%)	1 (6%)	18
C 	0	2 (10%)	1 (5%)	15 (75%)	2 (10%)	20
D 	0	3 (15%)	15 (75%)	2 (10%)	0	20
E 	0	0	0	0	0	0
F 	0	3 (9%)	28 (82%)	1 (3%)	2 (6%)	34
G 	1 (2%)	3 (6%)	11 (20%)	36 (67%)	3 (6%)	54
H 	0	0	0	2 (50%)	2 (50%)	4
I 	0	1 (25%)	0	1 (25%)	2 (50%)	4
J 	0	13 (72%)	1 (6%)	2 (11%)	2 (11%)	18

# Estatísticas

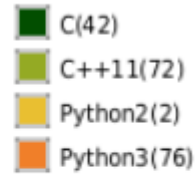
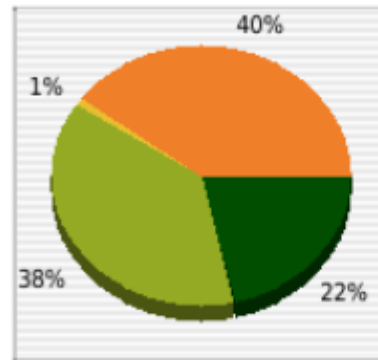
## Runs by Language

Languages	Total	Accepted
C	42	5 (12%)
C++11	72	22 (31%)
Python2	2	0 (0%)
Python3	76	2 (3%)

Powered by

BOCA

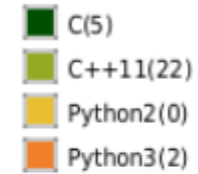
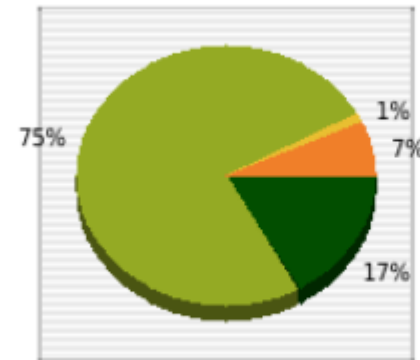
### All Runs by Language



Powered by

BOCA

### Accepted Runs by Language





# Estatísticas

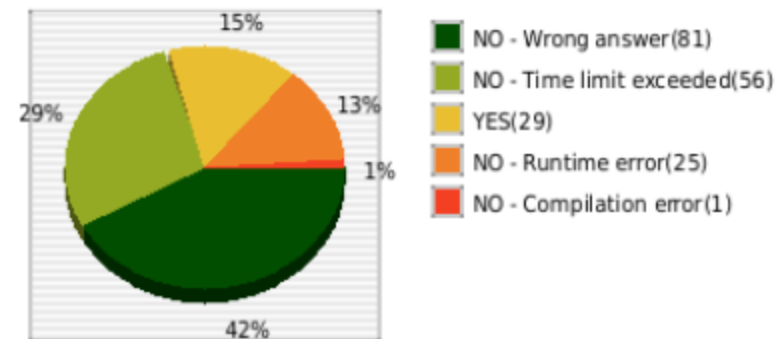
## Runs by Answer

<u>Answers</u>	Answers
NO - Compilation error	1
NO - Runtime error	25
NO - Time limit exceeded	56
NO - Wrong answer	81
YES	29

Powered by















BOCA

All Runs by Answer



















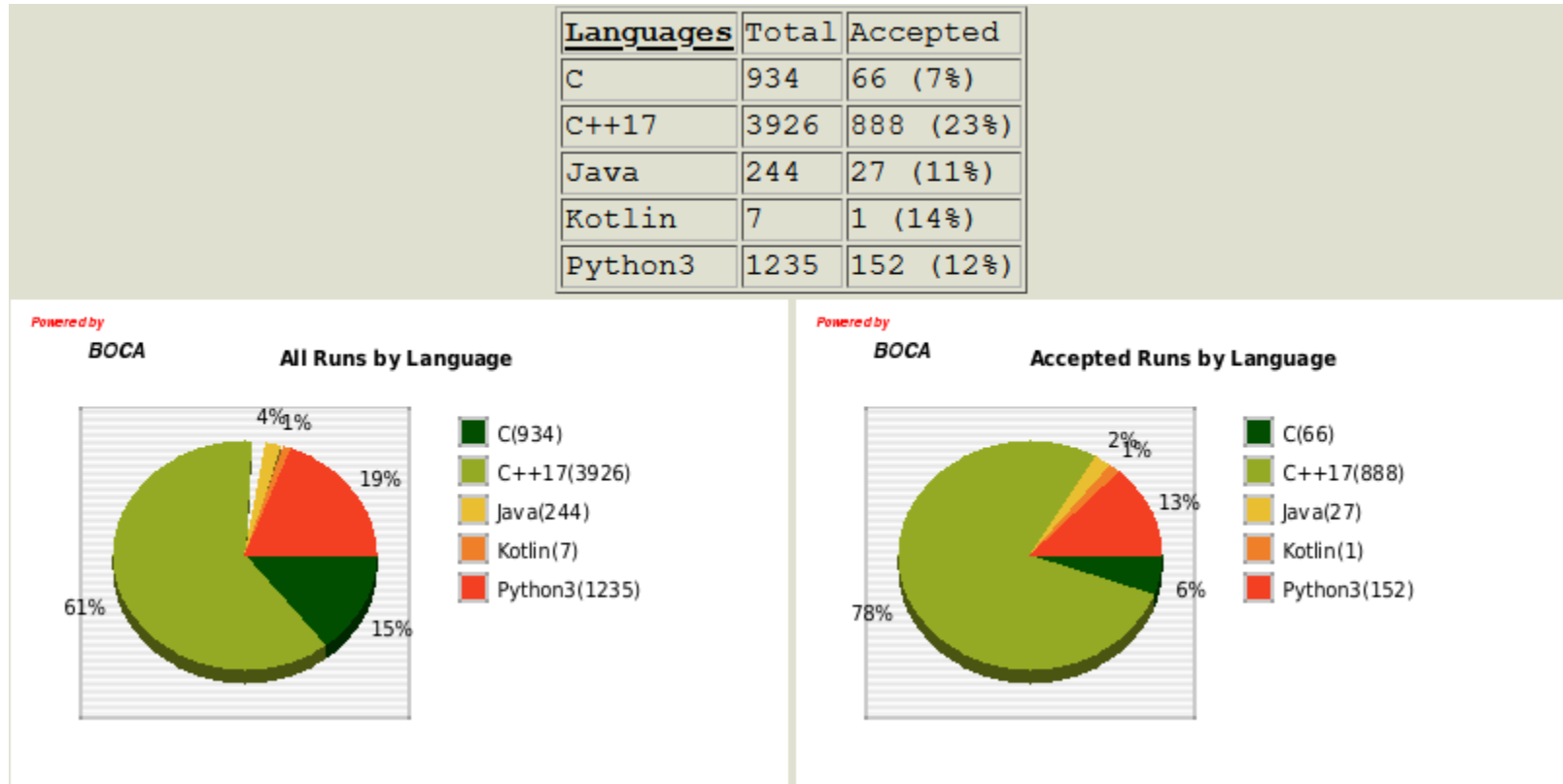
# Estatísticas - Regional 2021

Problems	Total	Accepted
A 	63	0 (0%)
B 	254	10 (4%)
C 	530	91 (17%)
D 	59	4 (7%)
E 	1008	149 (15%)
F 	16	0 (0%)
G 	361	41 (11%)
H 	1102	301 (27%)
I 	0	0
J 	3	0 (0%)
K 	1669	387 (23%)
L 	63	1 (2%)
M 	345	72 (21%)
N 	873	78 (9%)

# Estatísticas - Regional 2021

<u>Problems x Answers</u>	NO - Class name mismatch	NO - Compilation error	NO - Problem mismatch	NO - Runtime error	NO - Time limit exceeded	NO - Wrong answer	NO - Wrong language	YES	Total
A 	1 (2%)	6 (10%)	1 (2%)	5 (8%)	27 (43%)	23 (37%)	0	0	63
B 	1 (0%)	8 (3%)	0	13 (5%)	21 (8%)	200 (79%)	1 (0%)	10 (4%)	254
C 	2 (0%)	13 (2%)	0	22 (4%)	192 (36%)	210 (40%)	0	91 (17%)	530
D 	0	3 (5%)	0	7 (12%)	5 (8%)	40 (68%)	0	4 (7%)	59
E 	0	13 (1%)	0	34 (3%)	68 (7%)	742 (74%)	2 (0%)	149 (15%)	1008
F 	0	1 (6%)	0	3 (19%)	3 (19%)	9 (56%)	0	0	16
G 	0	3 (1%)	1 (0%)	9 (2%)	30 (8%)	277 (77%)	0	41 (11%)	361
H 	4 (0%)	29 (3%)	0	65 (6%)	351 (32%)	350 (32%)	2 (0%)	301 (27%)	1102
I 	0	0	0	0	0	0	0	0	0
J 	0	0	0	0	1 (33%)	2 (67%)	0	0	3
K 	10 (1%)	23 (1%)	0	94 (6%)	9 (1%)	1145 (69%)	1 (0%)	387 (23%)	1669
L 	0	3 (5%)	0	6 (10%)	19 (30%)	34 (54%)	0	1 (2%)	63
M 	0	4 (1%)	0	29 (8%)	156 (45%)	83 (24%)	1 (0%)	72 (21%)	345
N 	1 (0%)	21 (2%)	0	83 (10%)	449 (51%)	238 (27%)	3 (0%)	78 (9%)	873

# Estatísticas - Regional 2021



# Estatísticas - Regional 2021

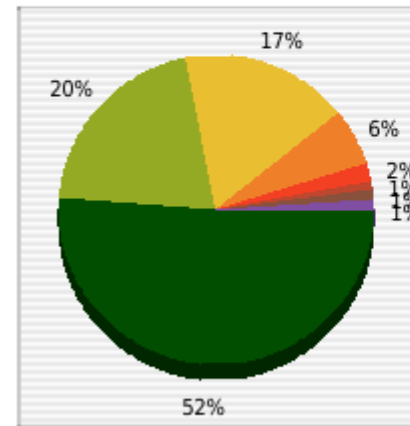
## Runs by Answer

<u>Answers</u>	Answers
NO - Class name mismatch	19
NO - Compilation error	127
NO - Problem mismatch	2
NO - Runtime error	370
NO - Time limit exceeded	1331
NO - Wrong answer	3353
NO - Wrong language	10
YES	1134

Powered by

BOCA

### All Runs by Answer



- NO - Wrong answer(3353)
- NO - Time limit exceeded(1331)
- YES(1134)
- NO - Runtime error(370)
- NO - Compilation error(127)
- NO - Wrong language(10)
- NO - Problem mismatch(2)
- NO - Class name mismatch(19)

# A - Aperto de Mão

- Problema: sabendo que em uma festa cada pessoa cumprimentou todas as demais uma única vez, e sabendo o total  $T$  de cumprimentos realizados, determinar o número de pessoas que haviam na festa.
- Exemplos:
  - 1 cumprimento  $\rightarrow$  2 pessoas
  - 15 cumprimentos  $\rightarrow$  6 pessoas



# A - Aperto de Mão

- Considerando que haviam  $x$  pessoas na festa, cada pessoa realizou  $x - 1$  cumprimentos. Sendo assim, podemos dizer que  $x \cdot (x - 1)$  é igual a  $2T$  (o dobro pois aqui estamos contando tanto quando a pessoa 1 cumprimenta a 2, como quando a 2 cumprimenta a 1, por exemplo).

$$x \cdot (x - 1) = 2T$$

$$x^2 - x = 2T$$

$$x^2 - x - 2T = 0$$

- Agora, basta solucionar esta equação de segundo grau (e apenas uma raiz irá importar, pois a outra será sempre negativa, o que não é uma solução viável para este problema).

# A - Aperto de Mão

```

#include <bits/stdc++.h>

using namespace std;

int main() {
    int N, cumprimentos;
    cin >> N;
    for (int i=1; i<=N; i++){
        double delta, x1;
        cin >> cumprimentos;
        delta = 1 - (4*1*(-2*cumprimentos));
        x1 = (1 + sqrt(delta)) / 2;
        cout << x1 << endl;
    }
}
  
```



# A - Aperto de Mão

```
import math

n = int(input())
for i in range(n):
    cumprimentos = int(input())
    delta = 1 - (4 * 1 * (-2 * cumprimentos))
    x1 = (1 + math.sqrt(delta)) / 2
    print(int(x1))
```

# G - VoGais e Consoantes

- Problema: dado um texto (em uma ou mais linhas), contar a quantidade de vogais e consoantes.

<b>Exemplo de entrada</b>  A soma de 2 + 2 = 4.	<b>Exemplo de saída</b>  4 vogais 3 consoantes
<b>Exemplo de entrada</b>  A CASA de papel queimou!	<b>Exemplo de saída</b>  11 vogais 8 consoantes
<b>Exemplo de entrada</b>  A e I o U	<b>Exemplo de saída</b>  5 vogais
<b>Exemplo de entrada</b>  De	<b>Exemplo de saída</b>  1 vogal 1 consoante

# G - VoGais e Consoantes

- Solução: direta, porém era fácil encontrar alguns obstáculos, especialmente aqueles que participaram do seu primeiro evento de programação competitiva. Por exemplo:
  - Leitura até o fim do arquivo
  - Leitura de linha inteira (com espaços em branco)
  - Tratamento de strings
  - Correta apresentação da saída
    - Possibilidade de 0 vogais ou 0 consoantes
    - Singular/plural

# G - VoGais e Consoantes

```

int main()
{

    int tot_vogais, tot_consoantes;
    string linha;
    tot_vogais = 0;
    tot_consoantes = 0;
    while (getline(cin, linha)) {
        for (int i = 0; i < linha.size(); i++) {
            char c = tolower(linha[i]);
            if (isalpha(c))
            {
                if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
                    tot_vogais++;
                else
                    tot_consoantes++;
            }
        }
    }
}
  
```

# G - VoGais e Consoantes

```

if (tot_vogais > 0) {
    cout << tot_vogais << (tot_vogais == 1 ? " vogal" : " vogais") << endl;
}
if (tot_consoantes > 0) {
    cout << tot_consoantes << (tot_consoantes == 1 ? " consoante" : " consoantes") << endl;
}
return 0;
}

```

# G - VoGais e Consoantes

```
import sys
tot_vogais = 0
tot_consoantes = 0
for linha in sys.stdin:
    for c in linha:
        if c.isalpha():
            if c.lower() in ['a', 'e', 'i', 'o', 'u']:
                tot_vogais += 1
            else:
                tot_consoantes += 1
if tot_vogais > 0:
    if tot_vogais == 1:
        print(tot_vogais, "vogal")
    else:
        print(tot_vogais, "vogais")
if tot_consoantes > 0:
    if tot_consoantes == 1:
        print(tot_consoantes, "consoante")
    else:
        print(tot_consoantes, "consoantes")
```

# C - Classificação Final

- Problema: Dado uma lista de competidores, seus países de origem e sua pontuação final, exibir o 3 melhores colocados, mas seguindo as seguintes restrições:
  - Um país só pode ter uma pessoa classificada, sendo assim, se o primeiro lugar for do Brasil, por exemplo, o segundo e o terceiro lugar devem ser de país diferentes.
  - Se houver empate entre dois jogadores, desempata-se pelo nome do país (preferência para o país de menor ordem lexicográfica)



# C - Classificação Final

- Problema: Dado uma lista de competidores, seus países de origem e sua pontuação final, exibir o 3 melhores colocados, mas seguindo as seguintes restrições:
  - Um país só pode ter uma pessoa classificada, sendo assim, se o primeiro lugar for do Brasil, por exemplo, o segundo e o terceiro lugar devem ser de país diferentes.
  - Se houver empate entre dois jogadores, desempata-se pelo nome do país (preferência para o país de menor ordem lexicográfica)
  - **ERRO**: e o caso de empate de jogadores do mesmo país?

# C - Classificação Final

- Solução: ordenar os jogadores em ordem decrescente pela pontuação, e desempatando pelo nome do país (de forma crescente). Por fim, percorrer a lista de jogadores encontrados os 3 primeiros colocados, sem repetir o país de origem.
  - OBS: a classificação final pode ter menos de 3 colocados, conforme exemplificado pelo segundo caso de teste do problema.
  - Utilizar STL ajuda muito com isso

# C - Classificação Final

- OBS: evite realizar sua própria implementação de algoritmo de ordenação:
  - Dependendo do método utilizado, sua solução terá complexidade maior
    - Bubble Sort:  $O(n^2)$
    - Ordenação do STL ou do Python:  $O(n \cdot \log n)$ 
      - Normalmente utiliza-se o Merge Sort, ou o Quick Sort, ou alguma versão híbrida.
  - Maior chance de ocorrência de erros
  - Mais tempo gasto escrevendo código

# C - Classificação Final

```
struct registro
{
    string competidor;
    string pais;
    int pontos;
};
```

```
bool compara(registro a, registro b)
{
    if (a.pontos != b.pontos)
        return a.pontos > b.pontos;
    return a.pais < b.pais;
}
```

# C - Classificação Final

```

int main()
{
    int N;

    cin >> N;
    vector<registro> placar;
    registro aux;
    for (int i = 0; i < N; i++)
    {
        cin >> aux.competidor >> aux.pais >> aux.pontos;
        placar.push_back(aux);
    }
    sort(placar.begin(), placar.end(), compara);

    ...

}
  
```

# C - Classificação Final

```
import functools

def compara(a, b):
    if a[2] < b[2]:
        return 1
    elif a[2] > b[2]:
        return -1
    elif a[1] < b[1]:
        return -1
    elif a[1] > b[1]:
        return 1
    elif a[3] < b[3]:
        return 1
    return -1
```

# C - Classificação Final

```
n = int(input())
placar = []
for i in range(n):
    competidor, pais, pontos = input().split()
    pontos = int(pontos)
    placar.append((competidor, pais, pontos, i))
placar = sorted(placar, key=functools.cmp_to_key(compara))

premios = [placar[0]]
achousegundolugar = False

for i in range(n):
    if (placar[i][1] != premios[0][1]) and not achousegundolugar:
        premios.append(placar[i])
        achousegundolugar = True
    elif (placar[i][1] != premios[0][1]) and (placar[i][1] != premios[1][1]):
        premios.append(placar[i])
        break

print(len(premios))
for i in range(len(premios)):
    print(i+1, premios[i][1],premios[i][0])
```



# J - Junior Programmer Jobs

- Problema: são dadas uma sequências de *queries* de dois tipos:
  - Cadastrar um novo aluno candidato com uma certa nota.
  - Consultar os N alunos com maiores notas (desempate por nome).

Exemplo de entrada	Exemplo de saída
<pre>I Catalina Urzaiz 7 I Sirio Montarelo 8.3 I Ailene Pinaud 9.4 C 2 I Pepita Fadim 9.5 I Jonhatan Heinzler 8.8 I Irma Aldegalega 7.0 C 3 C 20</pre>	<pre>9.4 Ailene Pinaud 8.3 Sirio Montarelo ----- 9.5 Pepita Fadim 9.4 Ailene Pinaud 8.8 Jonhatan Heinzler ----- 9.5 Pepita Fadim 9.4 Ailene Pinaud 8.8 Jonhatan Heinzler 8.3 Sirio Montarelo 7.0 Catalina Urzaiz 7.0 Irma Aldegalega -----</pre>

# J - Junior Programmer Jobs

- Uma solução para isso é utilizar alguma estrutura que já mantenha os dados ordenados, como o set do C++.
  - Inserir novo elemento:  $O(\log n)$
  - Iterar sobre o conjunto:  $O(n)$ , sendo  $n$  o número de elementos

# J - Junior Programmer Jobs

```
struct Pessoa {
    //Atributos
    string nome;
    float nota;

    //Construtor
    Pessoa(string nome, float nota) {
        this->nome = nome;
        this->nota = nota;
    }

    //Operador para comparar duas pessoas (e assim manter o set ordenado)
    bool operator<(const Pessoa &p) const {
        if (nota == p.nota) {
            return nome < p.nome;
        }
        return nota > p.nota;
    }
};
```

# J - Junior Programmer Jobs

```

int main(){
    multiset<Pessoa> conjunto;
    char op;    int qtde;
    string nome, sobrenome;
    float nota;
    cout << fixed << setprecision(1);
    while(cin >> op){
        if (op == 'I'){
            cin >> nome >> sobrenome >> nota;
            Pessoa aux(nome + " " + sobrenome, nota);
            conjunto.insert(aux);
        }
        else{
            cin >> qtde;
            for(auto p : conjunto){
                cout << p.nota << " " << p.nome << "\n";
                qtde--;
                if (qtde == 0)
                    break;
            }
            cout << "-----\n";
        }
    }
}
  
```

# H - Haja Interruptores!

- Problema: dados  $M$  lâmpadas e  $N$  interruptores, sendo que cada interruptor altera o estado de um subconjunto de lâmpadas, determinar se é possível apagar todas as luzes seguindo uma estratégia fixa:
  - Acionar os interruptores na sequência  $1, 2, 3, \dots, N, 1, 2, 3, \dots$
- Se for possível, imprimir a quantidade de vezes que o zelador vai acionar os interruptores.
- Caso contrário, imprimir -1

# H - Haja Interruptores!

- Ideia mais simples possível: simular o problema, ou seja, obedecer a sequência de acionamentos de interruptores, até que todas as lâmpadas estejam apagadas.

# H - Haja Interruptores!

- Ideia mais simples possível: simular o problema, ou seja, obedecer a sequência de acionamentos de interruptores, até que todas as lâmpadas estejam apagadas.
- Problema com essa ideia: há casos em que não temos solução, o que nos levaria a uma busca sem fim.



# H - Haja Interruptores!

- Vamos pegar o segundo caso de teste do enunciado (que não possui solução), simulá-lo por algum tempo e ver se encontramos algum padrão.

Exemplo de entrada	Exemplo de saída
3 3 2 2 3 1 3 2 1 2 1 2	-1

# H - Haja Interruptores!

Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Ações:

# H - Haja Interruptores!

Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Ações: 1

# H - Haja Interruptores!

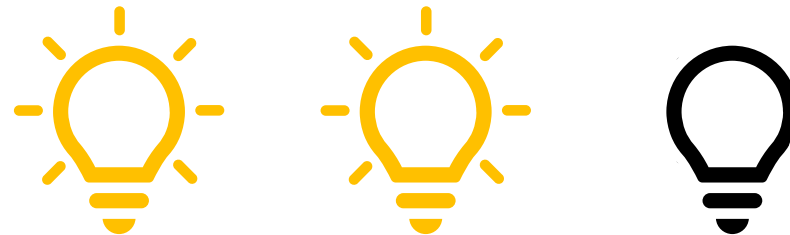
Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Ações: 1, 2

# H - Haja Interruptores!

Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Ações: 1, 2, 3

# H - Haja Interruptores!

Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Ações: 1, 2, 3, 1

# H - Haja Interruptores!

Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Ações: 1, 2, 3, 1, 2

# H - Haja Interruptores!

Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Ações: 1, 2, 3, 1, 2, 3



# H - Haja Interruptores!

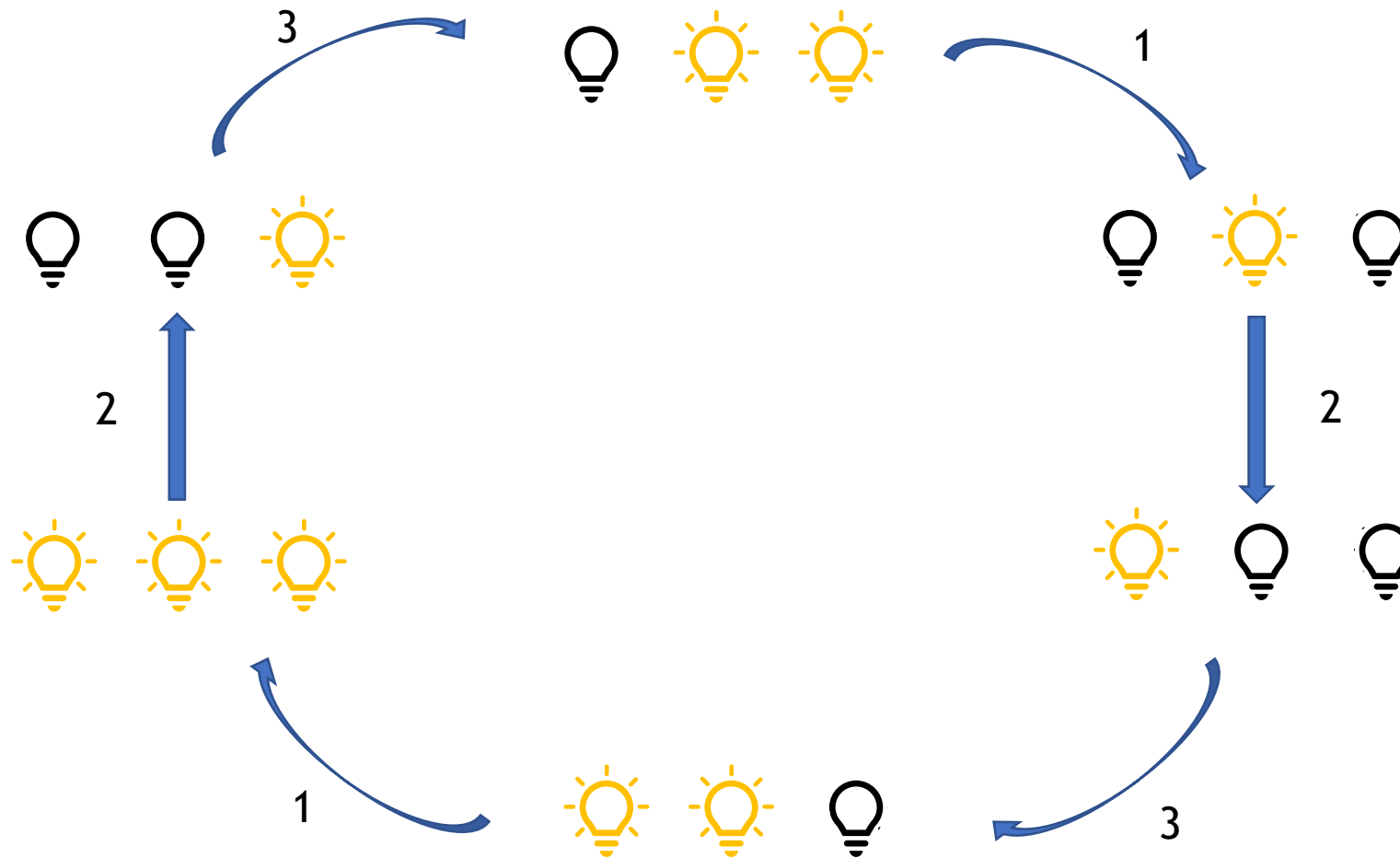
Interruptor	Lâmpadas
1	3
2	1, 2
3	2



Voltou ao estado original!

Ações: 1, 2, 3, 1, 2, 3

# H - Haja Interruptores!



# H - Haja Interruptores!

- Com este exemplo, podemos perceber que depois de passar duas vezes pela sequência de interruptores voltamos ao estado inicial, e a partir daí entramos em um ciclo.
- Sendo assim, não podemos realizar este processo apenas duas vezes, e se não encontrarmos uma situação em que todas as lâmpadas estejam apagadas, então não há solução.

# H - Haja Interruptores!

```
def resolver(lampAcesas, interruptores, qtdeLampAcesas):
    qtdeInt = len(interruptores)
    qtdeLamp = len(lampAcesas)
    qtdeLampApagadas = qtdeLamp - qtdeLampAcesas

    if qtdeLampApagadas == qtdeLamp:
        return 0

    mov = 0
    for k in range(2):
        for i in range(qtdeInt):
            mov += 1
            for pos in interruptores[i]:
                lampAcesas[pos] = not lampAcesas[pos]
                if lampAcesas[pos]:
                    qtdeLampApagadas -= 1
                else:
                    qtdeLampApagadas += 1
            if qtdeLampApagadas == qtdeLamp:
                return mov

    return -1
```

# H - Haja Interruptores!

```

qtdeInt, qtdeLamp = [int(x) for x in input().split()]

entrada = [int(x) for x in input().split()]
qtdeLampAcesas = entrada[0]
lampAcesa = [False for x in range(qtdeLamp)]
for i in range(qtdeLampAcesas):
    lampAcesa[entrada[i+1]-1] = True

interruptores = []
for i in range(qtdeInt):
    entrada = [int(x)-1 for x in input().split()]
    interruptores.append(entrada[1:])

print(resolver(lampAcesa, interruptores, qtdeLampAcesas))
  
```

## B - Batistando

- Problema: José percorre uma certa sequência de lojas, iniciando o percurso com  $X$  reais na carteira. Em cada loja  $i$ , ele verifica o preço de um único produto  $c_i$ , e se  $c_i \leq X$  ele compra. Sabendo que José voltou para a casa com pelo menos  $K$  itens, qual o menor valor possível que José deveria ter na carteira no início do percurso.

# B - Batistando

Exemplo de entrada	Exemplo de saída
6 3 1 1 2 3 7 10	4

Exemplo de entrada	Exemplo de saída
5 3 10 8 9 3 4	17

# B - Batistando

- Estratégias que NÃO funcionam mas que nos ocorrem durante a prova:
  - Estratégia gulosa: talvez exista alguma estratégia gulosa que solucione o problema, mas algoritmos mais simples como somar os  $k$  menores valores não funciona em todos os casos (caso de teste 2, por exemplo)
  - Busca binária: este exercício possui uma estrutura bastante tradicional de problemas de busca binária, porém sem caráter monótono, o que inviabiliza a aplicação de busca binária.



## B - Batistando

- Importante se atentar ao fato que há no máximo 100 lojas, e o valor de cada item é no máximo 100 (ou seja, a soma total dos valores é no máximo  $100 \cdot 100 = 10.000$ )
- Isso permite que aplicado um algoritmo de força bruta que passe no tempo:

para  $X = 1$  até 10.000:

verificar se José consegue comprar pelo menos  $K$  itens com  $X$  reais  
se sim, esta é a resposta

# B - Batistando

```
int consegueComprar(vector<int> &lojas, int x)
{
    int resp = 0;
    for(int c : lojas)
    {
        if (c <= x){
            resp++;
            x -= c;
        }
    }
    return resp;
}
```

# B - Batistando

```
int main()
{
    int n, k, x, aux, soma = 0;
    vector<int> lojas;
    cin >> n >> k;

    for(int i = 0; i < n; i++)
    {
        cin >> aux;
        soma += aux;
        lojas.push_back(aux);
    }

    for(x = 1; x <= soma; x++)
    {
        if (consegueComprar(lojas, x) >= k){
            cout << x << "\n";
            break;
        }
    }
}
```

# B - Batistando

```
def consegueComprar(lojas, x):
    resp = 0
    for c in lojas:
        if c <= x:
            x -= c
            resp += 1
    return resp

n, k = [int(x) for x in input().split()]
lojas = [int(x) for x in input().split()]
soma = sum(lojas)
for x in range(1, soma+1):
    if consegueComprar(lojas, x) >= k:
        print(x)
        break
```

# F - Fazendo Economia

- Problema: dado uma vetor  $v$ , responder diversas consultas do tipo  $(i, j)$ , imprimindo a soma dos valores  $v[i \dots j]$ .

Exemplo de entrada	Exemplo de saída
6	4
1 1 2 3 7 10	13
3	17
1 3	
2 5	
5 6	

# F - Fazendo Economia

- Solução por força bruta: para cada consulta, realizar a soma de todos os elementos entre as posições  $i$  e  $j$ .
- Porém, essa solução tem complexidade muito alta:
  - $Q$  consultas, com  $Q \leq 10^5$ .
  - Cada consulta tem complexidade, no pior caso,  $O(N)$ , com  $N \leq 10^5$ .
- Ou seja, o problema todo tem complexidade  $O(Q.N)$ , que devido ao tamanho que estas variáveis podem assumir, não passa no tempo.

# F - Fazendo Economia

$n$	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g. Enumerating permutations (Section 3.2)
$\leq [15..18]$	$O(2^n \times n^2)$	e.g. DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g. DP with bitmask technique (Section 8.3.1)
$\leq 100$	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, $nC_{k=4}$
$\leq 400$	$O(n^3)$	e.g. Floyd Warshall's (Section 4.5)
$\leq 2K$	$O(n^2 \log_2 n)$	e.g. 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g. Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 1M$	$O(n \log_2 n)$	e.g. Merge Sort, building Segment Tree (Section 2.3)
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	Most contest problem has $n \leq 1M$ (I/O bottleneck)

Table 1.4: Rule of thumb time complexities for the ‘Worst AC Algorithm’ for various single-test-case input sizes  $n$ , assuming that your CPU can compute  $100M$  items in 3s.

# F - Fazendo Economia

- Para resolver isso, utilizaremos um vetor de soma de prefixo. Com isso, cada consulta terá complexidade  $O(1)$ .



# F - Fazendo Economia

- Para resolver isso, utilizaremos um vetor de soma de prefixo. Com isso, cada consulta terá complexidade  $O(1)$ .

	0	1	2	3	4	5	6
<b>v</b>		1	1	2	3	7	10
<b>P</b>	0	1	2	4	7	14	24

# F - Fazendo Economia

- Para resolver isso, utilizaremos um vetor de soma de prefixo. Com isso, cada consulta terá complexidade  $O(1)$ .

	0	1	2	3	4	5	6
v		1	1	2	3	7	10
P	0	1	2	4	7	14	24

sum(2, 5)

# F - Fazendo Economia

- Para resolver isso, utilizaremos um vetor de soma de prefixo. Com isso, cada consulta terá complexidade  $O(1)$ .

	0	1	2	3	4	5	6
v		1	1	2	3	7	10
P	0	1	2	4	7	14	24

$$\text{sum}(2, 5) = p[5] - p[1] = 14 - 1 = 13$$

# F - Fazendo Economia

- Assim, podemos generalizar uma consulta  $q$  como sendo:

$$q(l, r) = P[r] - P[l - 1]$$

- Por este método, temos as seguintes complexidades:
  - Pré-processamento:  $O(n)$  (feito apenas uma vez, antes das consultas)
  - Alteração:  $O(n)$  (não precisamos para este exercício)
  - Consulta:  $O(1)$
- Complexidade do problema:  $O(N + Q)$

# F - Fazendo Economia

```

#include <bits/stdc++.h>
#define ll long long int
#define MAXN 100001

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n, q, a, l, r;
    int p[MAXN];
    cin >> n;
    p[0] = 0;
    for(int i = 0; i < n; i++){
        cin >> a;
        p[i+1] = p[i] + a;
    }
    cin >> q;
    while(q--){
        cin >> l >> r;
        cout << (p[r] - p[l-1]) << "\n";
    }
}

```

# F - Fazendo Economia

```
n = int(input())
a = [int(x) for x in input().split()]

p = [0]
for i in range(n):
    p.append(p[i] + a[i])

q = int(input())
for i in range(q):
    l, r = (int(x) for x in input().split())
    print(p[r]-p[l-1])
```

# I – 3D Monster Maze

- Problema: encontrar um caminho até a saída de um labirinto.
- Observações:
  - Sempre existe um caminho até a saída.
  - Existe apenas um caminho até a saída.

# I - 3D Monster Maze

- Solução: *backtracking* ou busca em profundidade.
  - A solução é simples, mas pode ser difícil de chegar nela caso estes conteúdos ainda não tenham sido vistos durante o curso.
  - Além disso, a implementação pode ser um pouco trabalhosa.



# I – 3D Monster Maze

- Algoritmo base para *backtracking*:

```

backtracking(S, k){
    if (!ehValida(S, k))
        return;
    if (ehSolucao(S, k))
        processa(S, k);
    for( $c_{k+1}$  in  $P_{k+1}$ ){
         $S' = S + c_{k+1}$ ;
        backtracking( $S'$ , k+1);
    }
}
  
```

# I - 3D Monster Maze

```
string lab[101];
int n;

pair<int,int> mov[] = { {0, 1}, {0, -1}, {1, 0}, {-1, 0} };

pair<int, int> movimentar(pair<int,int> pos, int sentido){
    pos.first += mov[sentido].first;
    pos.second += mov[sentido].second;
    return pos;
}

bool ehValida(pair<int, int> pos){
    if (pos.first < 0 || pos.first >= n || pos.second < 0 || pos.second >= n)
        return false;
    return ((lab[pos.first][pos.second] == '.') || (lab[pos.first][pos.second] == 'S'));
}
```

# I - 3D Monster Maze

```
bool percorrerLab(pair<int, int> pos){
    for(int sentido = 0; sentido < 4; sentido++){
        pair<int, int> newPos = movimentar(pos, sentido);

        if (!ehValida(newPos))
            continue;

        if (lab[newPos.first][newPos.second] == 'S')
            return true;

        lab[newPos.first][newPos.second] = 'C';
        if (percorrerLab(newPos))
            return true;
        lab[newPos.first][newPos.second] = '.';
    }
    return false;
}
```

# I - 3D Monster Maze

```
lab = []
mov = [(0,1), (0,-1), (1, 0), (-1, 0)]

def movimentar(pos, sentido):
    newPos = [pos[0], pos[1]]
    newPos[0] += mov[sentido][0]
    newPos[1] += mov[sentido][1]
    return newPos

def ehValida(pos):
    global lab
    if pos[0] < 0 or pos[0] >= n or pos[1] < 0 or pos[1] >= n:
        return False
    return lab[pos[0]][pos[1]] == '.' or lab[pos[0]][pos[1]] == 'S'
```

# I - 3D Monster Maze

```
def percorrerLab(pos):
    global lab

    for sentido in range(4):
        newPos = movimentar(pos, sentido)
        if not ehValida(newPos):
            continue

        if lab[newPos[0]][newPos[1]] == 'S':
            return True

        lab[newPos[0]][newPos[1]] = 'C'
        if percorrerLab(newPos):
            return True
        lab[newPos[0]][newPos[1]] = '.'

    return False
```

# D - Dígitos Finais

- Problema: imprimir os últimos  $k$  dígitos da potência  $a^b$ .
- Porém...
  - $1 \leq a \leq 10^{1000}$
  - $1 \leq b \leq 922 \times 10^{15}$
  - $1 \leq k \leq 8$

# D - Dígitos Finais

- Dificuldades:
  - $a$  pode ser um número muito grande, que extrapola os limites de variáveis do tipo int ou mesmo long long int (BigInteger)
  - $b$  não é um BigInteger, mas é um expoente muito grande, calcular uma potência com um expoente tão grande, em tempo linear, deve estourar o tempo limite.
    - Mesmo se a multiplicação fosse em tempo constante, o que não acontece com BigInteger.
- Não esquecer que queremos apenas os  $k$  dígitos finais da potência, sendo que  $k$  é um valor pequeno.

# D - Dígitos Finais

- Solução:
  - Exponenciação por divisão e conquista
  - Aritmética modular



# D - Dígitos Finais

- Exponenciação por divisão e conquista

$$x^n = \begin{cases} 1, & n = 0 \\ x^{\frac{n}{2}} \cdot x^{\frac{n}{2}}, & n \text{ é par} \\ x^{\frac{n}{2}} \cdot x^{\frac{n}{2}} \cdot x, & n \text{ é ímpar} \end{cases}$$

# D - Dígitos Finais

- Exponenciação por divisão e conquista

```

int potencia(int x, int n) {
    if (n == 0)
        return 1;
    int y = potencia(x, n/2);
    if (n % 2 == 0)
        return y*y;
    return y*y*x;
}
  
```

# D - Dígitos Finais

- Aritmética modular:

$$(x + y) \% n = ((x \% n) + (y \% n)) \% n$$

$$(x - y) \% n = ((x \% n) - (y \% n)) \% n$$

$$(x * y) \% n = ((x \% n) * (y \% n)) \% n$$

$$(x ^ y) \% n = ((x \% n) ^ y) \% n$$

# D - Dígitos Finais

- Exponenciação por divisão e conquista + Aritmética modular:

```

long long pow(long long x, long long y, long long mod) {
    if (y == 0)
        return 1;

    long long p = pow(x, y/2, mod);
    if (y % 2 == 0)
        return (p * p) % mod;
    else
        return (((p * p) % mod) * (x % mod)) % mod;
}
  
```

# D - Dígitos Finais

```
#include<bits/stdc++.h>
#define ll long long int

using namespace std;

ll pot(ll a, ll b, int mod){
    if (b == 0)
        return 1;
    ll x = pot(a, b/2, mod);
    if (b % 2 == 0)
        return (x * x) % mod;
    return (((x * x) % mod) * (a % mod)) % mod;
}

ll pot(ll a, ll b){
    if (b == 0)
        return 1;
    int x = pot(a, b/2);
    if (b % 2 == 0)
        return x * x;
    return x * x * a;
}
```

# D - Dígitos Finais

```
int main()
{
    int k;
    string a;
    ll b;
    cin >> a >> b >> k;

    if (k < a.size())
        cout << pot(stoll(a.substr(a.size()-k, k)), b, pot(10, k)) << "\n";
    else
        cout << pot(stoll(a), b, pot(10, k)) << "\n";
}
```

# D - Dígitos Finais

```
(a, b, k) = (int(x) for x in input().split())  
print(pow(a, b, pow(10, k)))  
#Sim, sério...
```

# E - Entregadores de Balões

- Problema: determinar o menor custo para passar por um certo conjunto de vértices de um grafo (e voltar para o vértice de partida)
  - Uma espécie de caixeiro viajante, mas com um subconjunto de vértices
- Máximo de vértices no grafo: 150
- Máximo de arestas: 10000
- Máximo de vértices a serem visitados: 13



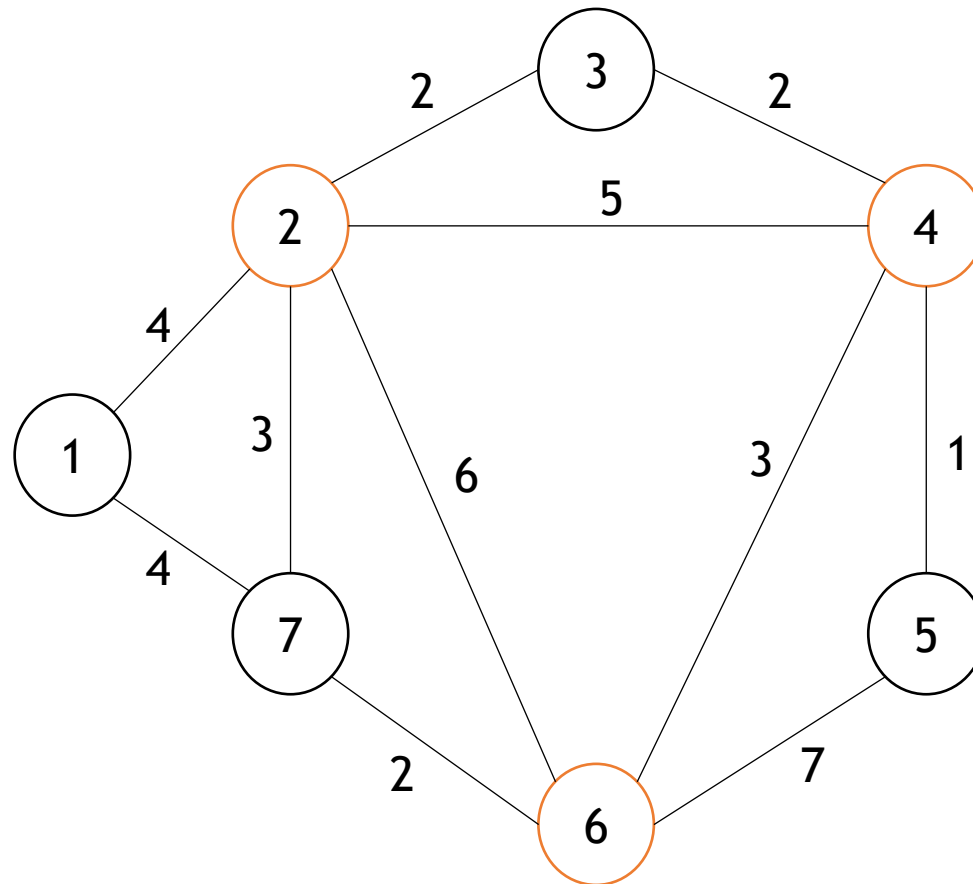
# E - Entregadores de Balões

- Problema do Caixeiro Viajante
  - NP-Completo
  - Sem solução polinomial conhecida
  - Solução por força-bruta:  $O(n!)$
  - Solução com PD:  $O(2^n n)$

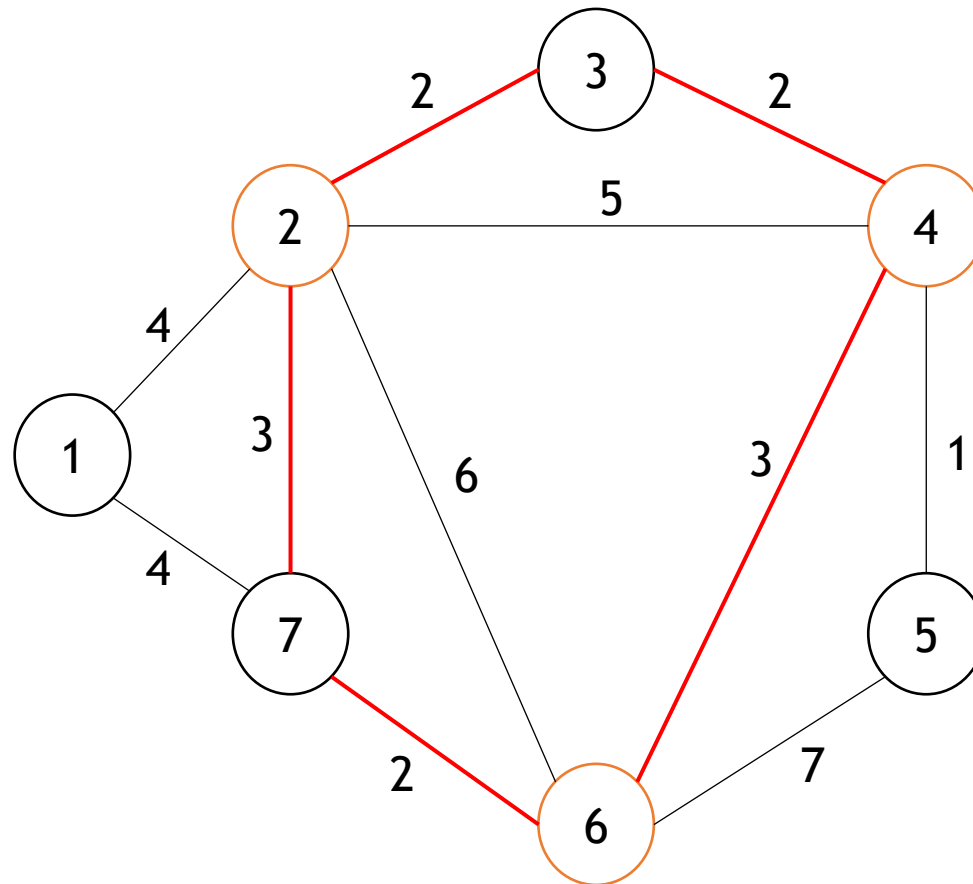
# E - Entregadores de Balões

- Porém, como só queremos visitar um subconjunto de vértices, temos que refletir um pouco mais sobre o problema:
  - Podemos passar por outros vértices fora desse subconjunto, mas apenas se eles conduzem a uma distância menor.
  - Considerar todos os  $N$  vértices para o caixeiro viajante estouraria o tempo.

# E - Entregadores de Balões

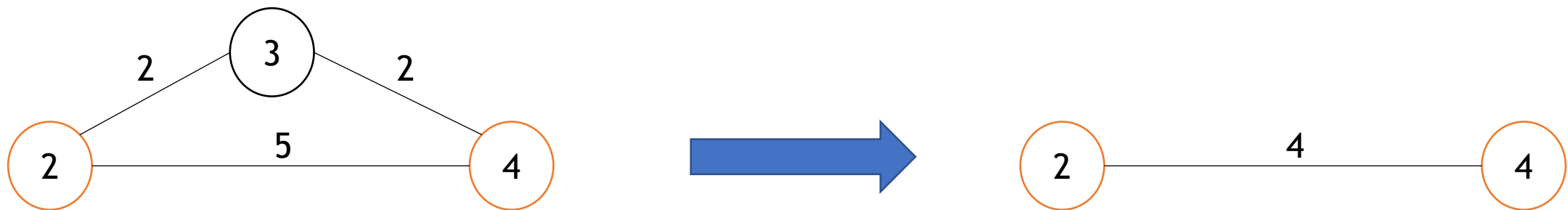


# E - Entregadores de Balões

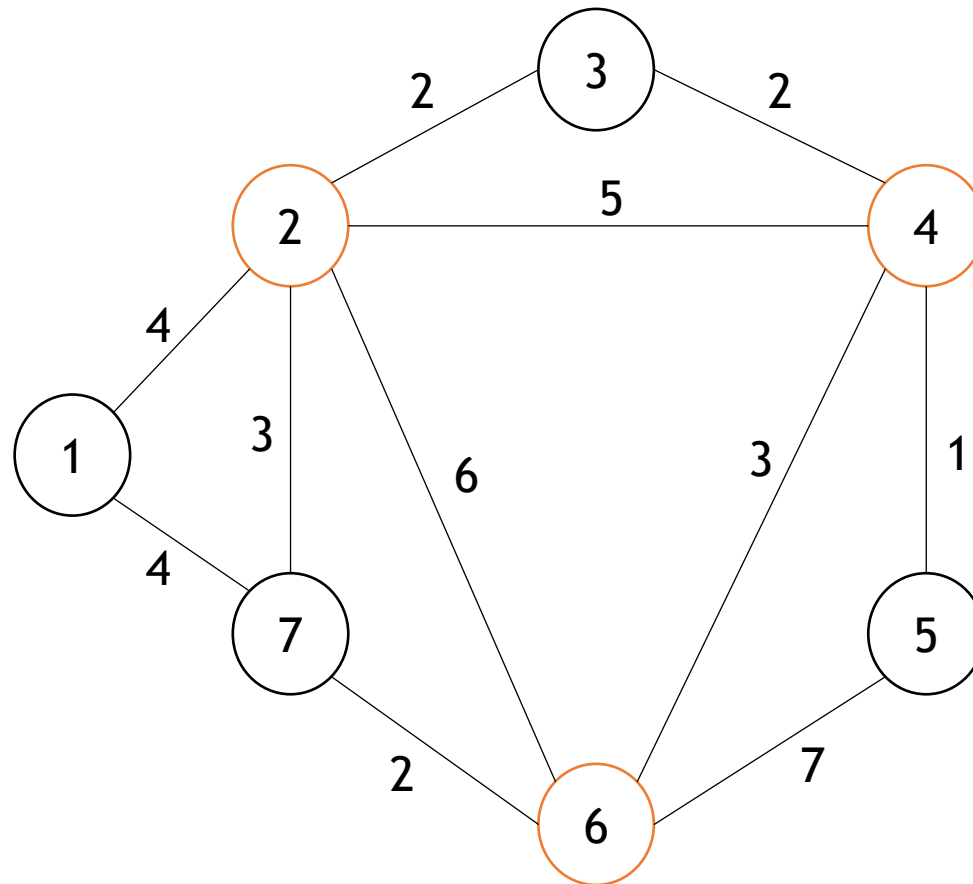


# E - Entregadores de Balões

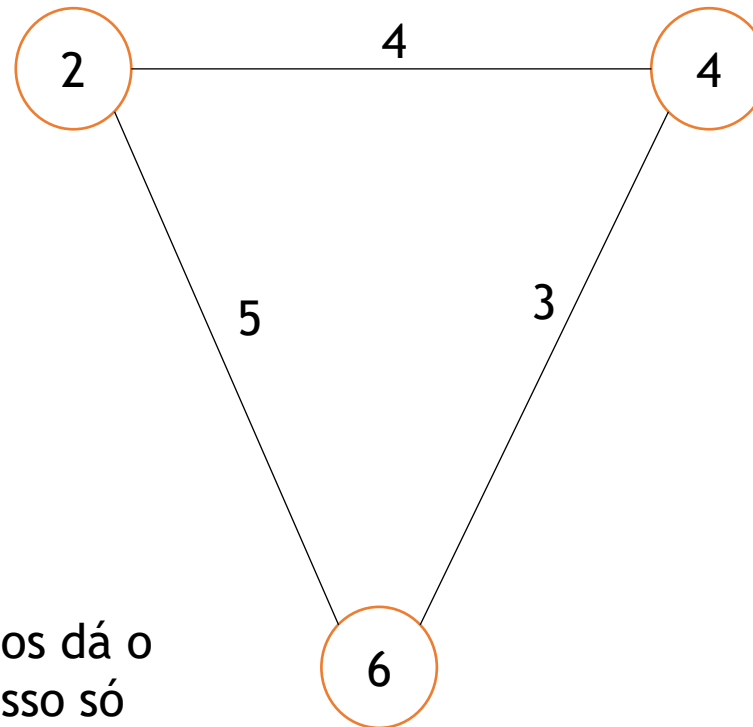
- Solução: compactar o grafo e então aplicar o algoritmo do caixeiro viajante.
  - Compactar o grafo: construir um novo grafo, apenas com as cidades do subconjunto, mas criando novas arestas entre elas que identifiquem o menor caminho entre as mesmas.



# E - Entregadores de Balões



# E - Entregadores de Balões



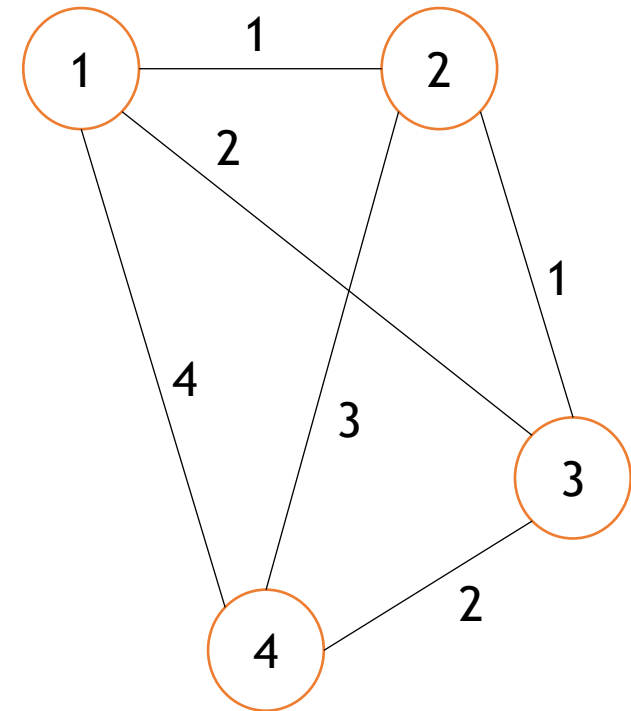
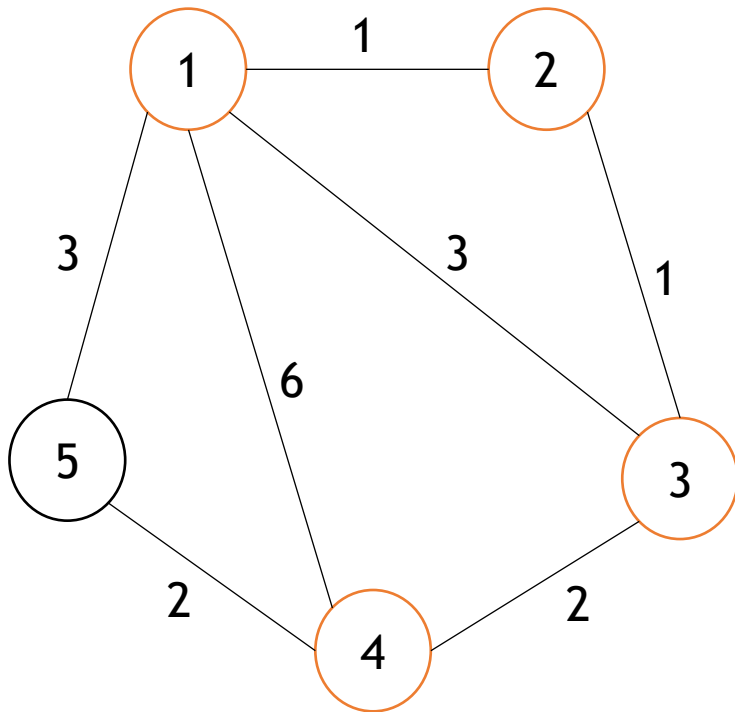
Neste caso o grafo compactado já nos dá o caminho do caixeiro viajante, mas isso só acontece porque temos apenas 3 vértices para visitar.

# E - Entregadores de Balões

- CUIDADO: ao compactar o grafo, ao calcular a menor distância de um vértice  $u$  até  $v$  não podemos passar por nenhum vértice  $w$  que também esteja no subconjunto de vértices a serem visitados.
  - Isso faria que um mesmo vértice fosse visitado duas vezes durante o percurso do caixeiro viajante, o que quebra as restrições do problema.



# E - Entregadores de Balões



# E - Entregadores de Balões

