Resolução dos Exercícios

Exercícios F, H, I

F - Buns

- TokiDeBuns, uma chefe de renome, vai fazer vários pãezinhos doces com diferentes recheios;
- Vamos chamar esses pãezinhos doces de donuts;
- Dada uma **quantidade de massa N**, qual é o **máximo de lucro** que ela consegue com deles ?

F - Buns

- Ela vai produzir **M donuts**, cada uma com recheio diferente;

- Para cada $donut_i$, ela tem **a_i de recheio dispo**
ela **precisa de b_i unidades de recheio** e
unidades de massa e pode **vendê-los pelo valor d_i**;

- Ela também pode fazer donuts **sem reche**
c_0 unidades de massa e vender por d_0.

F - Buns

Massa disponível: 10 unidades

F - Buns


```
Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores
```

F - Buns

```
Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores
```

	Recheio Disponível	**Recheio (1 Un.)**	**Massa (1 Un.)**	**Custo de Venda**
Donuts 0	Infinito	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	Infinito	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	Infinito	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

Vamos testar fazer donuts com quantidades variáveis de massa disponível, dividindo o problema em subproblemas até chegar na quantidade N.

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

Vamos testar fazer donuts com quantidades variáveis de massa disponível, dividindo o problema em subproblemas até chegar na quantidade N.

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10

Vetor da DP, contendo todas as possíveis quantidades de massa até N

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

Vamos supor que temos **8 unidades** de massa disponível

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	**8**	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

1 Unidade do Donuts 0

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

1 Un. do Donuts 0 = 2 Un. de massa
Voltamos para a situação **dp[8 - 2] = dp[6]**

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	**8**	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

Guardamos na dp[8] o **máximo** entre ela e a dp[6]
somado do valor de venda de **1 unidade**

0	0	0	0	0	0	0	0	1	0	0
0	1	2	3	4	5	6	7	**8**	9	10

dp[8] = max(dp[8], dp[8 - 2] + **1** * 1)

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

2 Unidades do Donuts 0

0	0	0	0	0	0	0	0	1	0	0
0	1	2	3	4	5	6	7	**8**	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

2 Un. do Donuts 0 = 4 Un. de massa
Voltamos para a situação **dp[8 - 4] = dp[4]**

0	0	0	0	0	0	0	0	1	0	0
0	1	2	3	4	5	6	7	**8**	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

Guardamos na dp[8] o **máximo** entre ela e a dp[4] somado do valor de venda de **2 unidades**

0	0	0	0	0	0	0	0	2	0	0
0	1	2	3	4	5	6	7	**8**	9	10

dp[8] = max(dp[8], dp[8 - 4] + **2** * 1)

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

Vamos repetir o processo para cada tipo de Donuts em nosso problema, resolvendo seus subproblemas que consistem em variar a quantidade de massa total disponível.

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

PROBLEMA!

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

PROBLEMA!

3 Unidades do Donuts 1

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

PROBLEMA!

3 Unidades do Donuts 1

0	1	2	3	4	5	6	7	8	9	10
0	0	1	1	2	2	3	3	4	4	5

Podemos resolver essa situação?

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

PROBLEMA!

3 Unidades do Donuts 1

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

Podemos resolver essa situação? NÃO!

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

Além da quantidade de massa disponível, temos que nos preocupar com a quantidade de recheio disponível também.

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

Neste caso, precisaríamos utilizar 9 Un. de recheio para fazer 3 Un. do Donuts 1, porém, temos disponível apenas 7 Un. de recheio.

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	**9**	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

PROBLEMA!

3 Unidades do Donuts 1

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

PROBLEMA!

3 Unidades do Donuts 1

0	0	1	1	2	2	3	3	4	4	5

0 1 2 3 4 5 6 7 8 9 10

Podemos resolver essa situação?

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

PROBLEMA!

3 Unidades do Donuts 1

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

Podemos resolver essa situação? NÃO!

F - Buns

Massa disponível: 10 unidades
Tipos de Donuts com recheio: 2 sabores

Simplesmente, para o subproblema em que temos disponível apenas 1 Un. de Massa, não podemos fazer Donuts do tipo 1, pois precisamos de 2 Un.

	Recheio Disponível	Recheio (1 Un.)	Massa (1 Un.)	Custo de Venda
Donuts 0	1000	1	2	1
Donuts 1	7	3	2	100
Donuts 2	12	3	1	10

0	0	1	1	2	2	3	3	4	4	5
0	1	2	3	4	5	6	7	8	9	10

F - Buns

- Podemos observar a seguinte recorrência:
  - dp[i] = max(dp[i], dp[i - w[j] * k] + c[j] * k);
  - c[j] * k <= i;
  - k variando de 0 até a[j] / b[j].

- Onde o índice **i** são todos as **possíveis quant** **de massa até N**, **j** são todos os **tipos de do** **k** a **quantidade de donuts i** que podem ser com o **recheio disponível**.

F - Buns

```
int main() {
    int c0, d0;
    cin >> n >> m >> c0 >> d0;
    a = b = c = d = vi(m + 1);
    a[0] = INF;
    b[0] = 1;
    c[0] = c0;
    d[0] = d0;

    for (int i = 1; i <= m; i++)
        cin >> a[i] >> b[i] >> c[i] >> d[i];

    cout << knapsack() << "\n";

    return 0;
}
```

F - Buns


```
int main() {
    int c0, d0;
    cin >> n >> m >> c0 >> d0;
    a = b = c = d = vi(m + 1);
    a[0] = INF;
    b[0] = 1;
    c[0] = c0;
    d[0] = d0;

    for (int i = 1; i <= m; i++)
        cin >> a[i] >> b[i] >> c[i] >> d[i];

    cout << knapsack() << "\n";

    return 0;
}
```

Essas duas linhas permitem que sejam criadas grandes quantidades de donuts sem recheio.

F - Buns

```cpp
int main() {
    int c0, d0;
    cin >> n >> m >> c0 >> d0;
    a = b = c = d = vi(m + 1);
    a[0] = INF;
    b[0] = 1;
    c[0] = c0;
    d[0] = d0;

    for (int i = 1; i <= m; i++)
        cin >> a[i] >> b[i] >> c[i] >> d[i];

    cout << knapsack() << "\n";

    return 0;
}
```

```cpp
int knapsack() {
    vi dp(n + 1);

    for (int j = 0; j <= m; j++) {
        for (int i = n; i >= 0; i--) {
            for (int k = 0; b[j] * k <= a[j]; k++) {
                if (c[j] * k <= i)
                    dp[i] = max(dp[i],
                        dp[i - c[j] * k] + d[j] * k);
            }
        }
    }

    return dp[n];
}
```

F - Buns


```
int main() {
    int c0, d0;
    cin >> n >> m >> c0 >> d0;
    a = b = c = d = vi(m + 1);
    a[0] = INF;
    b[0] = 1;
    c[0] = c0;
    d[0] = d0;

    for (int i = 1; i <= m; i++)
        cin >> a[i] >> b[i] >> c[i] >> d[i];

    cout << knapsack() << "\n";

    return 0;
}
```

Complexidade: O(N*M*A)
N -> total de massa
M -> total de donuts
A -> recheio disponível

```
int knapsack() {
    vi dp(n + 1);

    for (int j = 0; j <= m; j++) {
        for (int i = n; i >= 0; i--) {
            for (int k = 0; b[j] * k <= a[j]; k++) {
                if (c[j] * k <= i)
                    dp[i] = max(dp[i],
                        dp[i - c[j] * k] + d[j] * k);
            }
        }
    }

    return dp[n];
}
```

F - Buns

```
int main() {
    int c0, d0;
    cin >> n >> m >> c0 >> d0;
    a = b = c = d = vi(m + 1);
    a[0] = INF;
    b[0] = 1;
    c[0] = c0;
    d[0] = d0;

    for (int i = 1; i <= m; i++)
        cin >> a[i] >> b[i] >> c[i] >> d[i];

    cout << knapsack() << "\n";

    return 0;
}
```

Complexidade: O(N*M*A)
N -> total de massa
M -> total de donuts
A -> recheio disponível


```
int knapsack() {
    vi dp(n + 1);

    for (int j = 0; j <= m; j++) {
        for (int i = n; i >= 0; i--) {
            for (int k = 0; b[j] * k <= a[j]; k++) {
                if (c[j] * k <= i)
                    dp[i] = max(dp[i],
                        dp[i - c[j] * k] + d[j] * k);
            }
        }
    }

    return dp[n];
}
```

H - Longest Regular Bracket Sequence

Objetivo: Contar o tamanho da maior sequência correta de parênteses.

-> Devemos contar quantas sequências desse tamanho máximo existem.

-> Se não houver nenhuma sequência correta printar “0 1”.

H - Longest Regular Bracket Sequence

Exemplo:

Sequência ->)((()))((())))

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((()))

A pilha está vazia então quando aparece um parênteses para a esquerda, apenas continuamos

Pilha -> []

H - Longest Regular Bracket Sequence

Exemplo:

) ((( ))) ((( ))))

Quando for um parênteses para a direita, sua posição é adicionada na pilha

Pilha -> [2]

H - Longest Regular Bracket Sequence

Exemplo:

```
)((()))((()))))
```

Pilha -> [2, 3]

H - Longest Regular Bracket Sequence

Exemplo:

```
)((()))((()))))
```

```
Pilha -> [2, 3, 4]
```

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((())))

Temos um parênteses para a esquerda, temos que retirar um elemento da pilha.

PosAtual = 5

TopoDaPilha = 4

Pilha -> [2, 3, 4]

H - Longest Regular Bracket Sequence

```
Exemplo:

Tamanho da sequência de parênteses = PosAtual - TopoDaPilha + 1
```

```
)((()))((()))
```

```
Pilha -> [2, 3]
```

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((())))

Pilha -> [2, 3]

H - Longest Regular Bracket Sequence

Exemplo:

Tamanho da sequência de parênteses = 6 - 3 + 1 = 3

)((()))((()))

Pilha -> [2]

H - Longest Regular Bracket Sequence

Exemplo:

) ((( ))) ((( ))))

Pilha -> [2]

H - Longest Regular Bracket Sequence

Exemplo:

Tamanho da sequência de parênteses = 7 - 2 + 1 = 6

)((()))((())))

Pilha -> []

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((())))

Pilha -> [8]

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((())))

Pilha -> [8, 9]

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((())))

Pilha -> [8, 9, 10]

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((())))

Pilha -> [8, 9, 10]

H - Longest Regular Bracket Sequence

Exemplo:

Tamanho da sequência de parênteses = 11 - 10 + 1 = 2

)((()))((()))

Pilha -> [8, 9]

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((())))

Pilha -> [8, 9]

H - Longest Regular Bracket Sequence

Exemplo:

Tamanho da sequência de parênteses = 12 - 9 + 1 = 4

)((()))((()))

Pilha -> [8]

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((()))))

Pilha -> [8]

H - Longest Regular Bracket Sequence

Exemplo:

Tamanho da sequência de parênteses = 13 - 8 + 1 = 6

)((()))((())))

Pilha -> []

H - Longest Regular Bracket Sequence

Exemplo:

)((()))((()))

Pilha -> []

H - Longest Regular Bracket Sequence

- O máximo valor que consegui foi 6.
- Só que a maior sequência correta era 12
- Precisamos somar as duas sequências

)(((())))((()))))

H - Longest Regular Bracket Sequence

---

dp -> Armazena a maior sequência correta que tem até aquela posição.

Toda vez que fechar um parênteses pegar a maior sequência que começava uma posição antes dele.

)((()))((())))

dp[13] = dp[7] + tamanho

H - Longest Regular Bracket Sequence

Exemplo:

Tamanho da sequência de parênteses = 13 - 8 + 1 = 6

)((()))((()))))

dp[posAtual] = dp[TopoDaPilha - 1] + tamanho

dp[13] = dp[8 - 1] + 6

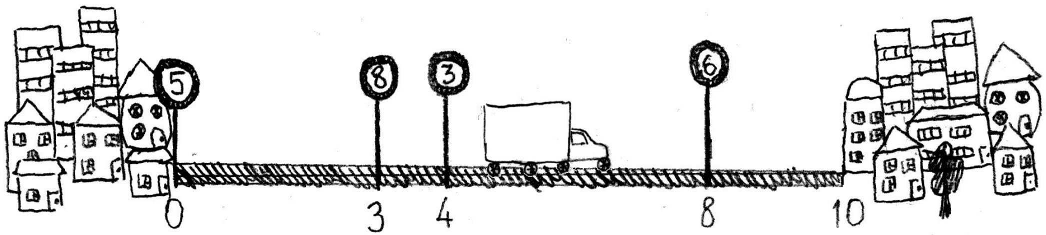
```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

int main(){
    string s;
    cin >> s;
    s = "+" + s;
    ll tam = s.size();
    vector<ll> dp (tam+1, 0);

    stack<ll> pq;
    ll ma = 0;

}
```


```cpp
for(int i = 1; i <= tam; i++){
    if(s[i] == ')'){
        if(pq.empty())continue;
        ll val = pq.top(); pq.pop();
        ll range = i - val + 1;
        dp[i] = range + dp[val-1];
        ma = max(ma, dp[i]);
    }
    else{
        pq.push(i);
    }
}
ll cont = 0;
for(int i = 1; i <= tam; i++){
    if(ma == dp[i])cont++;
}
if(ma == 0)cout << "0 1\n";
else cout << ma << " " << cont << "\n";
```

I - Road Optimization

O Governo de Marte está interessado em otimizar o sistema de rodovia!

Dado o sistema de uma rodovia de tamanho l, que possui uma sequência de n placas sendo d_i sua posição e a_i o limite de velocidade (min), sendo possível remover até k placas, qual o tempo mínimo para se deslocar do ponto 0 até l.



I - Road Optimization

```
d.push_back(l);
a.push_back(0);

dp[0][0] = 0;

for (int i = 1; i <= n; i++){
    for (int j = 0; j <= k; j++){
        for (int m = 0; m < i; m++){
            ll removed = j - (i - m - 1);
            if (removed >= 0)
            {
                dp[i][j] = min(dp[i][j], dp[m][removed] + (d[i] - d[m]) * a[m]);
            }
        }
    }
}
```