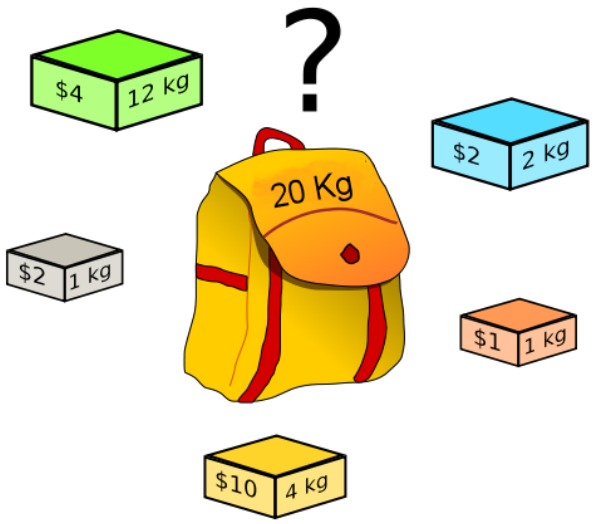
UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO
CURSO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMAÇÃO DE COMPUTADORES

TRABALHO PRÁTICO 1

Joãozinho, apesar de estar no início do curso de computação, já andou pesquisando e se deparou com um problema interessante de otimização chamado de Problema da Mochila. O problema consiste em levar em uma mochila os objetos que somem o maior valor possível, mas sem extrapolar sua capacidade de peso. Cada objeto possui um peso e um valor associado.



Esse problema pode ser resolvido por várias técnicas, incluindo programação dinâmica e algoritmos gulosos. Mas como Joãozinho tem interesse pela área de Inteligência Artificial, ele resolveu usar a técnica de **algoritmos genéticos**.

Joãozinho não será capaz de resolver completamente o problema, pois ele ainda está iniciando o aprendizado de programação de computadores, mas ele já pode construir a base do sistema necessário para a solução.

Vamos assumir que a mochila suporta 20Kg e que temos 16 objetos, cada um com peso e valor mostrados abaixo:

Objeto	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Peso (Kg)	12	3	5	4	9	1	2	3	4	1	2	4	5	2	4	1
Valor ($)	4	4	8	10	15	3	1	1	2	10	20	15	10	3	4	12

A solução usando algoritmos genéticos vai representar cada objeto por um bit. Vamos construir sequencias de 16 bits para indicar quais objetos estão na mochila e quais não estão. A sequência abaixo poderia representar uma solução para o problema.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	0	1	1	0	0	0	0	0	1	1	0	0	0	1	1

O primeiro bit igual a 1 significa que o objeto A está na mochila, o segundo bit igual a zero

significa que o objeto B não está na mochila, e assim por diante. Cada sequência de 16 bits representa uma solução para o problema. Nem todas as soluções representadas assim são possíveis, muitas vão extrapolar o peso máximo da mochila, outras não representarão a melhor solução. Por essa razão precisamos criar uma função de avaliação de soluções. Dada uma solução, a função deve retornar o valor total da mochila, ou zero, caso o peso máximo seja extrapolado.

Estamos interessados em encontrar a sequência de bits que nos dê uma solução ótima para o problema. A técnica de algoritmos genéticos faz isso através de tentativas. Ela parte de um conjunto inicial de soluções que são fornecidas pelo usuário. No nosso caso, as soluções serão números:

```
Entre com 6 soluções iniciais (números entre 0 e 65535):
60504
25000
12329
38054
1259
732
```

Cada número entre 0 e 65535 pode ser armazenado em uma variável inteira positiva com capacidade para 16 bits, fornecendo assim 6 soluções iniciais para o problema. A partir daí aplicaremos alguns **operadores genéticos** em cima das soluções iniciais para obter novas soluções.

A finalidade dos operadores genéticos é obter uma variação das soluções iniciais com o objetivo de chegar mais próximo de uma solução ótima para o problema. Normalmente os operadores são aplicados sobre soluções "promissoras", mas aqui aplicaremos eles indiscriminadamente para simplificar o problema.

1) Cruzamento

   a. Ponto único: o cruzamento tipo ponto único gera uma nova solução a partir de duas soluções existentes, copiando parte dos bits da primeira solução e o restante da segunda solução.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Solução A	1	1	1	0	1	1	0	0	0	1	0	1	1	0	0	0
Solução B	0	1	1	0	0	0	0	1	1	0	1	0	1	0	0	0
Nova	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0	0

   b. Aritmético (AND): o cruzamento tipo aritmético gera uma nova solução a partir de duas existentes, usando o bit 1 nas posições em que ambas as soluções iniciais sejam 1 e 0, caso contrário.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Solução A	0	0	1	1	0	0	0	0	0	0	1	0	1	0	0	1
Solução B	1	0	0	1	0	1	0	0	1	0	1	0	0	1	1	0
Nova	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

2) Mutação
   a. Simples: a mutação simples modifica um bit de uma solução existente, obtendo assim uma nova solução.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Solução	0	0	0	0	0	1	0	0	1	1	1	0	1	0	1	1
Nova	0	0	0	0	0	1	1	0	1	1	1	0	1	0	1	1

   b. Dupla: a mutação dupla tem o mesmo comportamento da simples, mas a nova solução tem dois bits modificados, no lugar de apenas um.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Solução	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	0
Nova	0	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0

Aplicaremos **cruzamento de ponto único** entre as duas primeiras soluções fornecidas pelo usuário, **cruzamento aritmético** entre a terceira e quarta solução, **mutação simples** sobre a quinta solução e, finalmente, **mutação dupla** sobre a sexta e última solução. Isso significa que após aplicação dos operadores genéticos teremos mais 4 soluções novas para avaliar. A aplicação dos operadores em cima do exemplo inicial resultaria nos seguintes números:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
60504	1	1	1	0	1	1	0	0	0	1	0	1	1	0	0	0
25000	0	1	1	0	0	0	0	1	1	0	1	0	1	0	0	0
12329	0	0	1	1	0	0	0	0	0	0	1	0	1	0	0	1
38054	1	0	0	1	0	1	0	0	1	0	1	0	0	1	1	0
1259	0	0	0	0	0	1	0	0	1	1	1	0	1	0	1	1
732	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	0
60584	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0	0
4128	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
1771	0	0	0	0	0	1	1	0	1	1	1	0	1	0	1	1
4820	0	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0

Ajude Joãozinho a resolver esse problema. Construa um programa para ler os 6 valores iniciais, aplicar os operadores genéticos e avaliar as soluções obtidas. A saída do programa deve ser **formatada e colorida** como no exemplo abaixo:

```
Resultado da Avaliação
----------------------
60504 – $69 – 40Kg – X
25000 – $45 – 22Kg – X
12329 – $60 – 17Kg – OK
38054 – $46 – 29Kg – X
 1259 – $61 – 18Kg – OK
  732 – $41 – 18Kg – OK
----------------------
60584 – $66 – 41Kg – X
 4128 – $30 –  6Kg – OK
 1771 – $62 – 20Kg – OK
 4820 – $41 – 17Kg – OK
```

INSTRUÇÕES

O programa deverá conter uma função para cada uma das tarefas abaixo:

1. **Função de Avaliação**: recebe um valor inteiro, exibe o valor e peso total da solução e retorna um booleano indicando se o peso está dentro do limite da mochila;
2. **Cruzamento Ponto único**: recebe dois valores inteiros representando as soluções e retorna um valor inteiro representando o cruzamento de ponto único entre as soluções;
3. **Cruzamento Aritmético**: recebe dois valores inteiros representando as soluções e retorna um valor inteiro representando o cruzamento aritmético entre as soluções;
4. **Mutação Simples**: recebe um valor inteiro representando uma solução e retorna um valor inteiro representando a mutação simples da solução;
5. **Mutação Dupla**: recebe um valor inteiro representando uma solução e retorna um valor inteiro representando a mutação dupla da solução.

Estas funções devem ser implementadas nos arquivos ***genetico.h*** e ***genetico.cpp***. As funções que implementam os operadores genéticos precisarão manipular os bits dos números inteiros. Essa manipulação de bits deve ser feita através de funções separadas, a serem implementadas nos arquivos ***binario.h*** e ***binario.cpp***:

1. **Ligar Bit**: recebe um valor inteiro e a posição do bit a ser ligado, retornando o valor inteiro resultante da modificação do bit;
2. **Desligar Bit**: recebe um valor inteiro e a posição do bit a ser desligado, retornando o valor inteiro resultante da modificação do bit;
3. **Testar Bit**: recebe um valor inteiro e a posição do bit a ser testado, retornando um booleano para indicar se o bit está ou não ligado;
4. **AND Binário**: recebe dois valores inteiros e retorna um inteiro representando o resultado da operação AND bit a bit entre os números recebidos.
5. **OR Binário**: recebe dois valores inteiros e retorna um inteiro representando o resultado da operação OR bit a bit entre os números recebidos.
6. **Bits Baixos**: recebe um valor inteiro e retorna outro contendo apenas os 8 bits de mais baixa ordem do valor original, com os demais bits colocados para zero.
7. **Bits Altos**: recebe um valor inteiro e retorna outro contendo apenas os 8 bits de ordem mais alta do valor original, com os demais bits colocados para zero.

O programa final deverá conter cinco arquivos, sendo o arquivo **problema.cpp** aquele que conterá a função main e se encarregará de ler os dados pelo teclado e exibir os resultados do programa.

EXIGÊNCIAS

1) Não é permitido usar variáveis globais
2) A entrada e saída de dados só podem ser feitas no arquivo problema.cpp, salvo quando instruído explicitamente de outra forma (função de avaliação)
3) O programa deve ter comentários e estar bem organizado

ENTREGA DO TRABALHO

Grupos: Trabalho individual

Data da entrega: 23/02/2023 (até a meia noite)

Valor do Trabalho: 3,0 pontos (na 1a Unidade)

Forma de entrega: enviar apenas os arquivos fonte (.cpp) e os arquivos de inclusão (.h) compactados no formato **zip** através da tarefa correspondente no SIGAA.

O não cumprimento das orientações resultará em **penalidades:**

- Programa não executa no Visual Studio 2022 (3,0 pontos)
- Programa contém partes de outros trabalhos (3,0 pontos)
- Atraso na entrega (1,5 pontos por dia de atraso)
- Arquivo compactado em outro formato que não zip (0,5 ponto)
- Envio de outros arquivos que não sejam os .cpp e .h (0,5 ponto)
- Programa sem comentários e/ou desorganizado (0,5 ponto)