Instruções

1. A prova vale 100;
2. Você tem uma hora e 30 minutos para resolver as questões;
3. O acesso à Internet será então restabelecido para a submissão das respostas;
4. A solução do exercício 1 deve estar no arquivo *1.cpp*;
5. A solução do exercício 2 deve estar no arquivo *2.cpp*;
6. A solução do exercício 3 deve estar no arquivo *3.cpp*;
7. Comprima os arquivos fontes com o comando `tar -czvf pp3.tar.gz 1.cpp 2.cpp 3.cpp`
8. Envie o arquivo `pp3.tar.gz` para `gcom.tp.sub@gmail.com`;
9. O envio deve ser feito obrigatoriamente de seu e-mail institucional da UFV;
10. O assunto (*subject*) do e-mail deve ser apenas seu número de matrícula (sem o ES);
11. Nenhum tipo de plágio será tolerado;
12. **A interpretação faz parte da avaliação.**

1. (30 Pontos) Faça um programa, que use operadores em bits, para verificar se um número (inteiro de 32 bits sem sinal) fornecido pela linha de comando é divisível por alguma potência de 2 de expoente inteiro e positivo (ou seja, você deve verificar se existe $i > 0$ tal que o número fornecido seja divisível por 2^i). Em caso positivo, seu programa deve imprimir 1 na tela. Em caso contrário, deve imprimir 0.

   Observação você não está autorizado a usar operadores aritméticos (-, +, *, / e %) e comandos de repetição.

   Exemplo:

```
g++ -o 1 1.cpp
./1 1
0
./1 64
1
```

   Explicação: veja que 1 é divisível apenas por 1 e que 64 é divisível por 2^6.

2. (35 Pontos) Um aluno da disciplina de CAP CXII teve a tarefa de implementar o *RadixSort* na versão MSD (*Most Significant Digit*) para inteiros não negativos de 32 bits (utiliza o *RadixSort* começando pelo bit mais significativo). Após algumas dicas do professor, o aluno chegou a seguinte implementação:

```
#include <iostream>

using namespace std;

// está correta
// Veja que (1 << w) é uma máscara para o bit de índice w
int msdRadixPartition(unsigned int *v, int l, int r, int w) {
    int i = l, j = r;
    while (j != i) {
        while ((v[i] & (1 << w)) == 0 && (i < j)) i++;
        while ((v[j] & (1 << w)) != 0 && (j > i)) j--;
        swap(v[i], v[j]);
    }
    if ((v[r] & (1 << w)) == 0) j++;
    return j;
}

// Ordena o arranjo v[l], ..., v[r]
// w é o índice do bit sendo analisado
void msdRadixRec(unsigned int *v, int l, int r, int w) {
    if (r <= l || w < 32)
        return;
    int pos = msdRadixPartition(v, l, r, w);
    msdRadixRec(v, l   , pos - 1, w + 1);
    msdRadixRec(v, pos, r      , w + 1);
}

void msdRadixSort(unsigned int *v, int n) {
    msdRadixRec(v, 0, n-1, 0);
}

// está correta
int main() {
    int n;
    int i;
    cin >> n;
    unsigned int *x = new unsigned int[n];
    for (i = 0; i < n; i++)
        cin >> x[i];
    msdRadixSort(x, n);
    for (i = 0; i < n; i++)
        cout << x[i] << endl;
    delete []x;
    return 0;
}
```

Ao perceber que o código não funcionava, o aluno foi ao professor pedir ajuda. O professor disse ao aluno: "Bom, você fez praticamente tudo certo. Só está confundindo MSD e LSD. Se perceber isso, mudando poucos caracteres, seu código vai funcionar".

Siga a dica do professor de CAP CXII e conserte o código.

3. (35 Pontos) Seja S um conjunto com n elementos distintos (inteiros não negativos de 32 bits) e k um número (também um inteiro não negativo de 32 bits). Desenvolva um programa, baseado em backtracking **com poda**, para descobrir se, dados S e k, S possui algum subconjunto cuja soma dos elementos seja k. Para lhe ajudar, abaixo o código que vimos em sala para gerar combinações.

```
void imprimeCombinacoes(bool combs[], int begin, int n) {
    if (begin >= n) {
        for(int i=0;i<n;i++) {
            cout << combs[i];
        }
        cout << endl;
    } else {
        combs[begin] = 0;
        imprimeCombinacoes(combs, begin+1, n);
        combs[begin] = 1;
        imprimeCombinacoes(combs, begin+1, n);
    }
}
int main(int argc, char **argv) {
    int n = atoi(argv[1]);
    bool combs[n];
    imprimeCombinacoes(combs,0,n);
}
```

Seu programa deve primeiro ler k, então n e depois o valor de cada um dos elementos. A saída do programa deve ser 1, se o subconjunto de soma k existe e 0 caso contrário.

Exemplo:

```
g++ -o 3 3.cpp
./3
1
3
2 3 4
0

./3
1
5
1 2 3 4 5
1
```

Explicação: No primeiro exemplo, é impossível encontrar um subconjunto de $\{2, 3, 4\}$ que some 1. No segundo, o subconjunto $\{1\}$ tem soma 1.

Observação: Você pode usar variáveis globais nesse exercício, mas com moderação!