Programando com recursão vetores, busca e ordenação

Vamos continuar aplicando a recursão na resolução de problemas com vetores.

Checando se todos os elementos de um vetor são pares

Iterativo

A ideia é que se encontramos pelo menos um número ímpar podemos dizer que nem todos são pares.

```
int todos_pares(int v[], int n){
    for(int i = 0; i < n; i++){
        if(v[i]%2==1) return 0;
    }
    return 1;
}
```

Recursivo

```
int todos_pares_recursivo(int v[], int n){
    if(n==1){
        return (v[0]%2==0);
    }else{
        //ultimo precisa ser par
        //e o restante também
        int r = todos_pares_recursivo(v, n-1);
        return (v[n-1]%2==0) && r==1;
    }
}
```

Existe pelo menos um par

Iterativo

```
int existe_par(int v[], int n){
    for(int i = 0; i < n; i++){
        if(v[i]%2==0) return 1;
    }
    return 0;
}
```

Recursivo

```
int existe_par_recursivo(int v[], int n){
    if(n==1){
        return (v[0]%2==0);
```

```
    }else{
        return (v[n-1]%2==0) ||
            existe_par_recursivo(v, n-1);
    }
}
```

Busca vetor não ordenado

```
int busca(int v[], int n, int k){
    if(n==1){
        return v[0]==k;
    }else{
        int encontrou = busca(v, n-1, k);
        if(encontrou == 1) return 1;
        else return v[n-1]==k;
    }
}
```

Busca binária vetor não ordenado

```
int __bin(int v[],int inicio, int fim, int k){
    if(inicio > fim)
        return 0;
    else {
        int meio = (inicio+fim)/2;
        if(v[meio]==k)
            return 1;
        else if(v[meio]>k)
            return __bin(v, inicio, meio-1, k);
        else
            return __bin(v, meio+1, fim, k);
    }
}

int buscabin(int v[], int n, int k){
    return __bin(v, 0, n-1, k);
}
```

Checa se duas palavras são iguais

Iterativo

```
int palavras_iguais_iterativo(char * s1, char * s2){
    int i = 0;
    while( s1[i] != '\0' && s2[i] != '\0'){
        if(s1[i] != s2[i]) return 0;
```

```
    }
    return s1[i] == '\0' && s2[i] == '\0';
}
```

Recursivo

```
int __palavra_iguais(char * s1, char * s2, int i){
    //se pelos menos um palavra chegou ao fim estamos no caso base
    if( s1[i] == '\0' || s2[i] == '\0'){
        if(s1[i] == '\0' && s2[i] == '\0') return 1;
        else return 0;
    }else {
        if(s1[i] == s2[i] )
            return __palavra_iguais(s1, s2, i+1);
        else
            return 0;
    }
}

int palavras_iguais(char * s1, char * s2){
    return __palavra_iguais(s1, s2, 0);
}

// palavra_iguais("ABC","ABC", 0) =
```

Palindrome

Iterativo

```
int palindromeIterativo(char * s){
    int n = strlen(s);
    int i = 0;
    int j = n-1;

    while(i < j){
        if(s[i] != s[j]) return 0;
        i++;
        j--;
    }

    return 1;
}
```

Recursivo

```
//funcao auxiliar
int __palindrome(char * s, int i, int j){
```

```
    if(s[i] == s[j])
        return __palindrome(s, i+1, j-1);
    else
        return 0;
}


int palindrome(char *s){
    int n = strlen(s);
    return __palindrome(s, 0, n-1);
}
```

Subsequencia

Iterativo

```
int subsequenceIterativo(char *s1, char * s2){
    int j = 0;
    for(int i = 0; s1[i] != '\0'; i++){
        while( s2[j] != '\0' && s2[j] != s1[i] ) {
            printf("testando s1 %c s2 %c\n", s1[i], s2[j]);
            j++;
        }
        if( s2[j] == '\0') return 0;
        else {
            printf("casa s1 %c s2 %c\n", s1[i], s2[j]);
            j++;
        }
    }
    return 1;
}
```

Recursivo

```
int __subsequence(char *s1, char *s2, int i, int j, int n1, int n2){
    if(i == n1){
        // a primeira palavra está vazia
        return 1;
    }else if(j == n2){
        //a segunda palavra está vazia e
        //a primeira não está vazia
        return 0;
    }else{
        if(s1[i] == s2[j]){
            return __subsequence(s1, s2, i+1, j+1, n1, n2);
        }else{
            return __subsequence(s1, s2, i, j+1, n1, n2);
```

```
        }
    }

}

int subsequence(char *s1, char * s2){
    int n1 = strlen(s1);
    int n2 = strlen(s2);

    return __subsequence(s1, s2, 0, 0, n1, n2);
}
```

Coloca os pares no começo e os ímpares no final do vetor

Separa versão 1

A função `separa(v,n)` devolve um inteiro k tal que `v[0..k]` é formado por números pares e `v[k+1..n-1]` é formado por números ímpares.

```
int separa1(int v[], int n){
    int k = -1;
    for(int i = 0; i < n; i++){
        if(v[i] % 2 == 0){
            k++;
            int t = v[i];
            v[i] = v[k];
            v[k] = t;

        }
    }
    return k;
}

int checa(int v[], int n, int k){
    for(int i = 0; i < n; i++){
        if(i<=k && v[i]%2==1) return 0;
        if(i>k  && v[i]%2==0) return 0;
    }
    return 1;
}
```

Separa versão 1 Recursivo

```
int __separa1_rec(int v[], int n, int i, int k){
    if(i >= n) return k;
    printf("(i,k) = (%d, %d)\n", i, k);
```

```
    if(v[i]%2 == 0){
        k++;
        int t = v[k];
        v[k] = v[i];
        v[i] = t;
        return __separa1_rec(v, n, i+1, k);
    }else{
        return __separa1_rec(v, n, i+1, k);
    }
}

int separa1_rec(int v[], int n){
    return __separa1_rec(v, n, 0, -1);
}
```

Separa versão 2

```
int separa2(int v[], int n){
    int i = 0; //procurando um ímpar
    int j = n-1; //procurando um par

    while(i<=j){
        if( v[i] % 2 == 0) i++;
        else if( v[j] %2 == 1) j--;
        else {
            int t = v[i];
            v[i] = v[j];
            v[j] = t;
        }
    }
    printf("i %d j %d\n", i, j);
    return i-1;
    // para no primeiro ímpar direita -> esquerda
    // para no primeiro par esquerda -> direita
}
```

Separa versão 2 Recursivo

```
int __separa2_rec(int v[], int n, int i, int j){
    if( i > j){
        return i-1;
    }else{
        if(v[i]%2==0) return __separa2_rec(v, n, i+1, j);
        else if(v[j]%2==1) return __separa2_rec(v, n, i, j+1);
        else {
            int t = v[i];
```

```
            v[i] = v[j];
            v[j] = t;
            return __separa2_rec(v, n, i+1, j-1);
        }
    }
}

int separa2_rec(int v[], int n){
    return  __separa2_rec(v, n, 0, n-1);
}
```

Ordenação

Particiona

A função particiona recebe três parâmetros v, inicio e fim. A função escolhe o primeiro elemento como pivô. Em seguida, o vetor é particionado da seguinte maneira:

- v[0..k-1] <= pivô
- v[k] pivô
- v[k+1..n] >= pivô

Iterativo

```
int particiona(int v[], int inicio, int fim){
    int k = inicio;
    int pivo = v[inicio];
    for(int j = inicio+1; j <= fim; j++){
        if(v[j] <= pivo){
            k++;
            troca(v, j, k);
        }
    }
    troca(v, inicio, k);
    return k;
}
```

Quicksort Recursivo

```
void _qsort(int v[], int i, int f){
    if(f-i>0){
        int k = particiona(v, i, f);
        _qsort(v, i, k-1);
        _qsort(v, k+1, f);
    }
```

```
}

void quicksort(int v[], int n){
    return _qsort(v, 0, n-1);
}
```