**LISP**

**Descrição do trabalho:**

Existem diversas linguagens de programação atualmente, cada uma com suas particularidades e diferenciais, algumas mais voltadas para alguns tipos de problemas, outros para problemas de outras natureza, cada uma respeitando seus paradigmas suportados. Este trabalho tem por objetivo instigar nos alunos o interesse em novas linguagens, talvez desconhecidas até então, e aumentarem seus conhecimentos, sabendo quais tipos de problemas podem ser implementados em cada linguagem. Cada aluno da turma ficou responsável por estudar uma ou duas linguagens e tentar implementar e realizar estudos a respeito dos problemas: Fibonacci recursivo, ParseInt, Quicksort, Conjunto de Mandelbrot, Geração de Pi de Euler, Estatística em Matriz Randômica, Método de Relaxação Sucessiva (SOR) e o Método de Newton.

**História da linguagem**

Lisp é uma linguagem de paradigma funcional criada por John McCarthy em 1958, que apesar de ter excelente desempenho, possui uma alta complexidade de implementação devido a dependência do formato recursivo.

A linguagem funcional debatida é muito utiliza para problemas de Inteligência Artificial(IA) e a recursão que é um dos focos de seu paradigma (funcional), se emprega muitas vezes como estrutura de controle, o que resta eficiência às execuções.

Algo importante a se ressaltar é a deficiência da linguagem em relação à recursos para aplicação de métodos numéricos, impossibilitando a implementação de muitos problemas propostos.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Linguagem | Fib. Rec. | Parse int | Quick Sort | Mandelbrot | Pi Euler | Est. Matriz Rand | SOR | Mét. De Newton |
| Lisp | 0,09 | - | 0,31 | - | - | - | - | - |

**Figura 1 - Quadro com o tempo de execução dos algoritmos dos problemas sugeridos.**

**Código Fib. Recursivo:**

(defun fibonacci (n &optional (a 0) (b 1) (acc ()))

(if(zerop n)

(nreverse acc)

(fibonacci (1-n) b (+ a b) ( cons a acc))))

ou

|  |
| --- |
| (defun fib-loop (num) |
|  | (do ((n 0 (1+ n)) |
|  | (cur 0 next) |
|  | (next 1 (+ cur next))) |
|  | ((= num n) cur))) |

Quick Sort em Lisp:

|  |
| --- |
| (defun quicksort (lst) |
|  | (if (null lst) |
|  | nil |
|  | (let\* ((x (car lst)) |
|  | (r (cdr lst)) |
|  | (fn (lambda (a) (< a x)))) |
|  | (append (quicksort (remove-if-not fn r)) |
|  | (list x) |
|  | (quicksort (remove-if fn r)))))) |

**Conclusão do trabalho:**

Com a evolução da computação, diversas linguagens de programação evoluíram e novas linguagens vem surgindo a cada ano. Com o intuito de avaliar o desempenho de algumas dessas linguagens foi proposto uma serie de problemas matemáticos clássicos, envolvendo desde cálculos simples a manipulações avançadas com matrizes. Cada linguagem possui sua particularidade e vantagens de utilização. Um dos principais fatores que diferenciam a aplicação dessas linguagens ˜ e seu paradigma. Linguagens como Lisp, Haskell e OCaml que operam sobre o paradigma funcional possuem uma alta complexidade de implementação, na maioria das vezes dependentes de uma modelagem recursiva, pilhas e listas. Por outro lado são altamente eficientes em problemas que exigem um alto esforço computacional. As linguagens voltadas para uma abordagem direta com a matemática, como o Scilab, Octave e Matlab possuem uma linguagem extremamente simples e praticas, facilitando muito a implementação de problemas que envolvem principalmente manipulação de matrizes. Linguagens de baixo nível como C/C++ muitas vezes precisam de dezenas de linhas para representar algo que pode ser feito com apenas em uma linha no Matlab/Octave. Ao contrário das linguagens matemáticas que apresentaram maior tempo que os demais em diversos problemas testados. Apesar de se mostrarem mais lentas, as linguagens matemáticas são extremamente recomendadas para aplicações que não exijam um alto esforço computacional e que envolvam problemas relacionados ao calculo numérico, sua sintaxe simplificada e vasta biblioteca matemática pode facilitar diversas tarefas durante a implementação.