SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho 2 - AEDS 1

Desempenho de algoritmo

Luan Barros Reis [6576]
Thomaz Augusto Araújo Silva [6577]
Yasmim Pereira Delfino[5919]

Florestal - MG
2025

Sumário

1. Código e Algoritmo

1.1 Algoritmo utilizado

O algoritmo de combinação utilizado foi obtido no site Rosetta Code [1], onde já estava implementado em linguagem C, não havendo necessidade de conversão.

```c
typedef unsigned long marker;
marker one = 1;

void comb(int pool, int need, marker chosen, int at)
{
    if (pool < need + at) return; /* not enough bits left */

    if (!need) {
        /* got all we needed; print the thing.  if other actions are
         * desired, we could have passed in a callback function. */
        for (at = 0; at < pool; at++)
            if (chosen & (one << at)) printf("%d ", at);
        printf("\n");
        return;
    }
    /* if we choose the current item, "or" (|) the bit to mark it so. */
    comb(pool, need - 1, chosen | (one << at), at + 1);
    comb(pool, need, chosen, at + 1);  /* or don't choose it, go to next */
}
```

Esse algoritmo gera todas as combinações possíveis de tamanho **pool** a partir de um conjunto de **need** elementos, sem repetição e sem considerar a ordem. Ele faz isso recursivamente, utilizando operações bit a bit para marcar quais elementos estão incluídos em cada subconjunto.

A variável **chosen** atua como um indicador de bits: cada bit representa se um determinado pacote está ou não presente na combinação atual.

A cada chamada recursiva, o algoritmo decide entre incluir ou não incluir o item atual **at**, explorando assim todas as possibilidades possíveis.

O caso base ocorre quando o número de elementos necessários **need** chega a zero — ou seja, quando uma combinação completa é formada.

1.2 Adaptação do código

Na adaptação feita para este trabalho, o comportamento do algoritmo foi alterado, para que, a cada subconjunto gerado, o código:

```
if (!need) {
    double pesoT = 0.0, prioT = 0.0;
    for (int i = 0; i < pool; i++) {
        if (chosen & (one << i)) {
            pesoT += vet[i].peso; /* 1 */
            prioT += vet[i].prioridade; /* 1 */
        }
    }

    /* 2 */
    if (pesoT <= pesoMAX && prioT > com->melhor_prioridade) {
        com->melhor_prioridade = prioT; /* 3 */
        com->melhor_comb = chosen; /* 3 */
    }
    return;
}
```

1. Calcule a **soma total dos pesos** e das **prioridades** dos pacotes.
2. Verifique se o **peso total** respeita o limite máximo do drone e se a prioridade é maior que a melhor combinação encontrada até o momento.
3. Armazene o subconjunto como o melhor, através de um ponteiro **com** do tipo **struct combinacao**.

```
void comb(int pool, int need, marker chosen, int at,
     pacote* vet, double pesoMAX, combinacao* com);
```

Esse processo é repetido para subconjuntos de tamanho 1 até N, garantindo que todas as possibilidades sejam avaliadas — caracterizando um **algoritmo de força bruta** com **complexidade exponencial $O(2^n)$.**

1.3 Funções auxiliares implementadas para o TP2

Para atender às necessidades do Problema de Carregamento, foram criadas algumas funções adicionais além das já implementadas no TP1 :

- `contarpacotes(lista* l)`: retorna a quantidade de pacotes na lista encadeada, utilizada para determinar o tamanho do conjunto de entrada antes de gerar combinações.
- `listaparaarray(lista* l, pacote* vetor)`: converte a lista encadeada em um array, permitindo que o algoritmo de combinações opere diretamente sobre o vetor de pacotes.
- `removerpacote(lista* l, pacote p)`: remove um pacote específico da lista encadeada, garantindo que pacotes já carregados em uma viagem anterior não sejam considerados novamente nas próximas combinações. Possui complexidade O(n), desprezível frente à complexidade exponencial $O(2^n)$ do algoritmo de combinação.

2. Configurações de Hardware e Software

Hardware:

- Processador: AMD Ryzen 5 5500 (6 núcleos, 12 threads, frequência base 3,6 GHz, turbo até 4,2 GHz)
- Placa de vídeo: AMD RX 5700 XT
- Memória RAM: 32 GB DDR4 3600 MHz CL18
- Armazenamento: SSD NVMe 512 GB (velocidade de leitura: 2200 MB/s, gravação: 1600 MB/s)

Software:

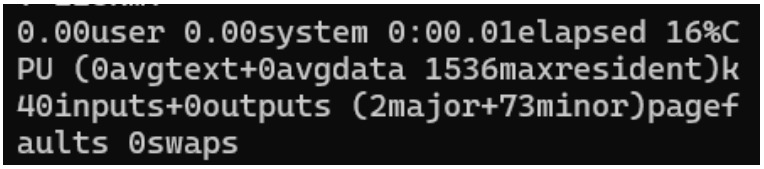
- Sistema operacional: Ubuntu via WSL2 no Windows 11 Pro (versão 25H2)
- Kernel Linux: 6.6.87.2-microsoft-standard-WSL2
- WSL: versão 2.6.1.0
- Compilador C: GCC 6.3.0 (MinGW.org GCC-6.3.0-1) - Arquitetura 64 bits
- IDE/editor: Visual Studio Code

3. Resultados e Tempo de Execução

Os testes foram realizados utilizando um Makefile, que compila o programa e executa automaticamente, ou por teclado, os testes para diferentes valores de N (10, 15, 20 e 30). Para rodar o programa, basta digitar **make run** na raiz do projeto, e então seguir as orientações. A execução foi feita com otimizações de compilação **-O2** e o tempo de execução foi medido internamente com a biblioteca **time**. Os tempos reais de execução foram os seguintes:

N	Tempo de Execução	CPU(%)	Memória Máx
10	0.000128s	6%	1536 kB
15	0.001180s	16%	1536 kB
20	0.032433s	76%	1536 kB
30	36.007415s	100%	1536 kB

Os valores de CPU e memória foram obtidos usando uma execução alternativa com o comando “`time ./$(BIN) $(TEST_DIR)/testeN=15.txt`” dentro do Makefile. O tempo real nesta medição é menos preciso que os valores da execução principal utilizando medição interna através da biblioteca “`<time.h>`”.



Vale ressaltar que os tempos podem variar alguns milésimos de segundo dependendo de aplicativos em execução em segundo plano, se foi dado make clean antes de rodar, entre outros fatores. No geral, a média de execução (rodando 3 vezes) foi a mostrada na tabela.

4. Seria razoável executar o seu algoritmo para valores de N=50? Justifique a resposta.

O algoritmo é força bruta que gera todas as combinações de 1 a N elementos, logo sua complexidade é $O(2^n)$. Usando como base a execução medida:

- Tempo medido para N = **30**: 36.007415s (execução real).
- $2^{50} / 20^{30} = 2^{20} \approx 1.048.576$
- Estimativa de tempo para N = **50**:
- 36.007415s × 1.048.576 ≈ 37756511.191 ≈ 436.9 dias ≈ 1,19 anos
- Ou seja, levaria mais de um ano pro código terminar de rodar.

Logo, não é razoável para N=50 com a implementação atual.

5. Referências

[1] ROSETTACODE. Combinations. Disponível em: <https://rosettacode.org/wiki/Combinations#C>. Acesso em: 19 out. 2025.

6. GitHub

https://github.com/essashr/tp2-C