Análise e Síntese de Algoritmos – 2º Projecto

Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526

Introdução

Este relatório aborda a solução encontrada para o problema proposto pelo corpo docente da UC como segundo projecto no ano curricular 2017/18.

Problema: Segmentação de imagens – considerando uma imagem como um rectângulo de píxeis, pretende-se segmentar os píxeis, ou seja, classificá-los como sendo de primeiro plano (**L**) ou de cenário (**C**). Para tal, foram atribuídos, por um processo estatístico, pesos a cada píxel (**lp** de primeiro plano e **cp** de cenário) e às ligações entre píxeis (**fv**) que são determinantes para o resultado final.

Descrição da solução

A implementação do algoritmo foi elaborada em linguagem C++.

Sucintamente, interpretou-se a imagem como uma **rede fluxo** (*flow network*), ou seja, como um **grafo dirigido com capacidades**. A solução consiste em encontrar o fluxo máximo e o corte mínimo do grafo, em que **L** e **C** são os conjuntos de vértices separados pelo corte. Para tal, criou-se uma fonte (*source*, vértice **s**) e um poço (*sink*, vértice **t**) universais. Ligou-se **s** a <u>todos</u> os vértices usando os pesos **lp** e <u>todos</u> os vértices a **t** usando os pesos **cp**. Os vértices da imagem ligaram-se entre si usando os pesos **fv**.

Considerando um grafo G como G = (V, E), com V o conjunto de vertices e E o conjunto de arcos, este foi representado internamente como (**a**) um array e (**b**) uma lista de adjacências: (**a**) para guardar os vértices e (**b**) como representação do grafo contendo arcos (com vértices de entrada e saída, capacidade e fluxo); (**b**) é indexada pela chave do vértice. Apesar do grafo poder ser representado num array estático, decidiu-se usar (**b**) por razões de legibilidade e flexibilidade.

O input da matriz foi dado por m (#linhas) e n (#colunas):
→ |V| = mn + 2, com 2 = **s** + **t**;
→ |E| = 2[mn + (m – 1)n + m(n – 1)] = 6(|V| - 2) - 2(m + n) = O(V)

Esta solução foi feita com base numa aplicação do algoritmo Edmonds-Karp (**EK**) para fluxos seguida de uma verificação de cada vértice para saber se foi visitado durante a última iteração da procura em largura (**BFS**). O estado de visita do vértice permite encontrar o corte mínimo (*minCut*). Formalmente, os passos incluem:

1. A leitura do input, alocação de memória e inserção de vértices e arcos. Não foram inseridos arcos com capacidade 0, visto que não seriam úteis.
2. Obtenção de um fluxo inicial:
   caminhos de aumento **p = {s → u, u → t}**, ∀u ∈ V \ {s, t};
3. Aplicação do algoritmo **EK** para encontrar o fluxo máximo;
4. Passagem pelo array de vértices e escrita do output respectivo ao estado de visita do vértice;
5. Libertação da memória alocada.

A solução **preserva** o input original mas seria necessário atribiuir de novo o valor inicial do fluxo (0) às arcos.

Análise e Síntese de Algoritmos – 2º Projecto

Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526

Análise teórica

Passos (desc. da solução)	Tempo	Espaço (memória)
(1)	O(V) → Inserção de vértices; Leitura e inserção de arcos: d = O(1) amortizado → Inserção em array dinâmico d*O(V) → (s, u) ∈ E, ∀u ∈ V \ {s, t}; d*O(V) → (u, t) ∈ E, ∀u ∈ V \ {s, t}; d*O(V) → (u, v) ∈ E, ∀u,v ∈ V \ {s, t} : u != v; **Total: d*O(V) = O(1)*O(V) = O(V)**	O(V) → Array de vértices; O(V + E) → Lista de adjacências, com E = O(V) **Total: O(V)**
(2)	Obtênção de fluxo inicial: O(V) → percorrer adjacência de **s**; **Total: O(V)**	O(1) → independente do tamanho do input; **Total: O(1)**
(3)	Algoritmo EK para fluxos provado como sendo $O(VE^2)$, com E = O(V); O(1) → Atribuição do estado de visita a cada vértice; **Total: $O(V^3)$**	O(V) → FIFO *Queue* da BFS; O(V) → Lista de arcos predecessores de um caminho; **Total: O(V)**
(4)	O(1) → Verificação do estado de visita de cada vértice; O(V) → Percorrer o array de vértices; **Total: O(V)**	O(1) → independente do tamanho do input; **Total: O(1)**
(5)	O(V + E) → Correr lista de adjacências, E = O(V) **Total: O(V)**	O(1) → independente do tamanho do input; **Total: O(1)**

Complexidade temporal da solução: $O(V^3)$
Complexidade espacial da solução: O(V)

Análise e Síntese de Algoritmos – 2º Projecto

Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526

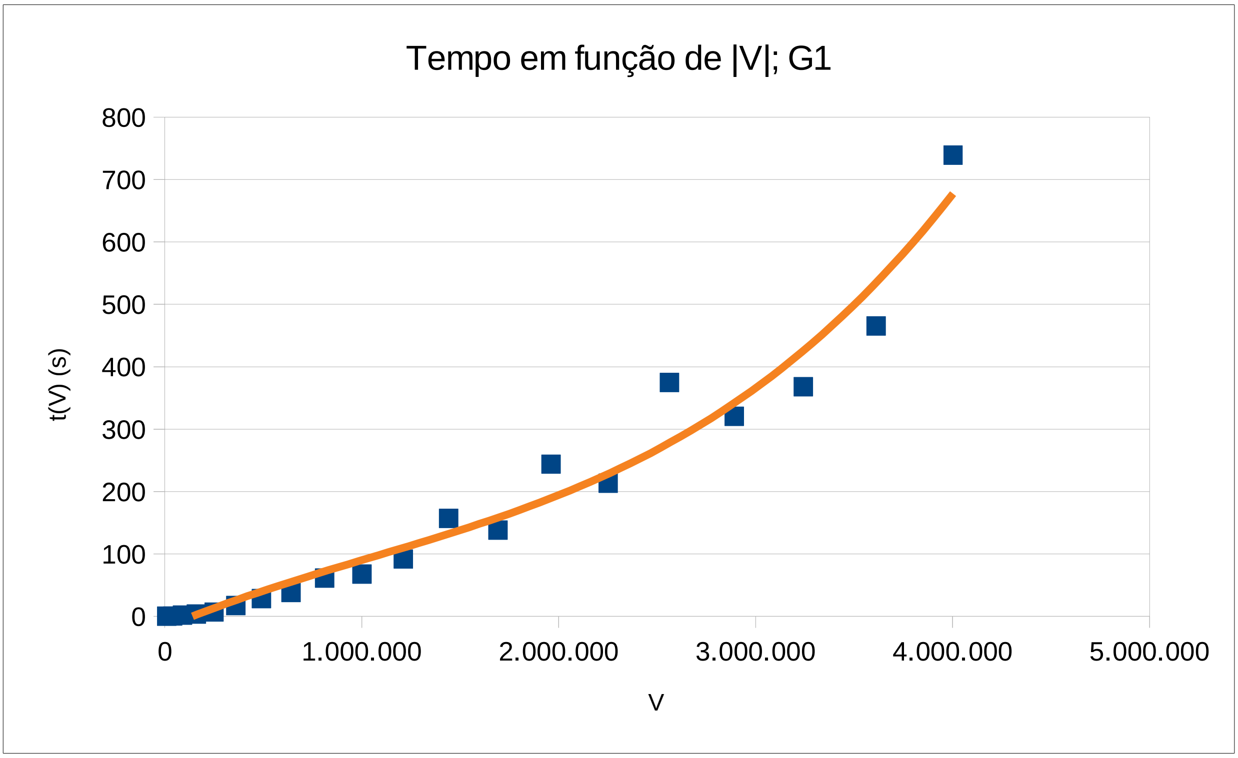
Análise experimental dos resultados

Os testes foram realizados em 2 tipos de ambiente:

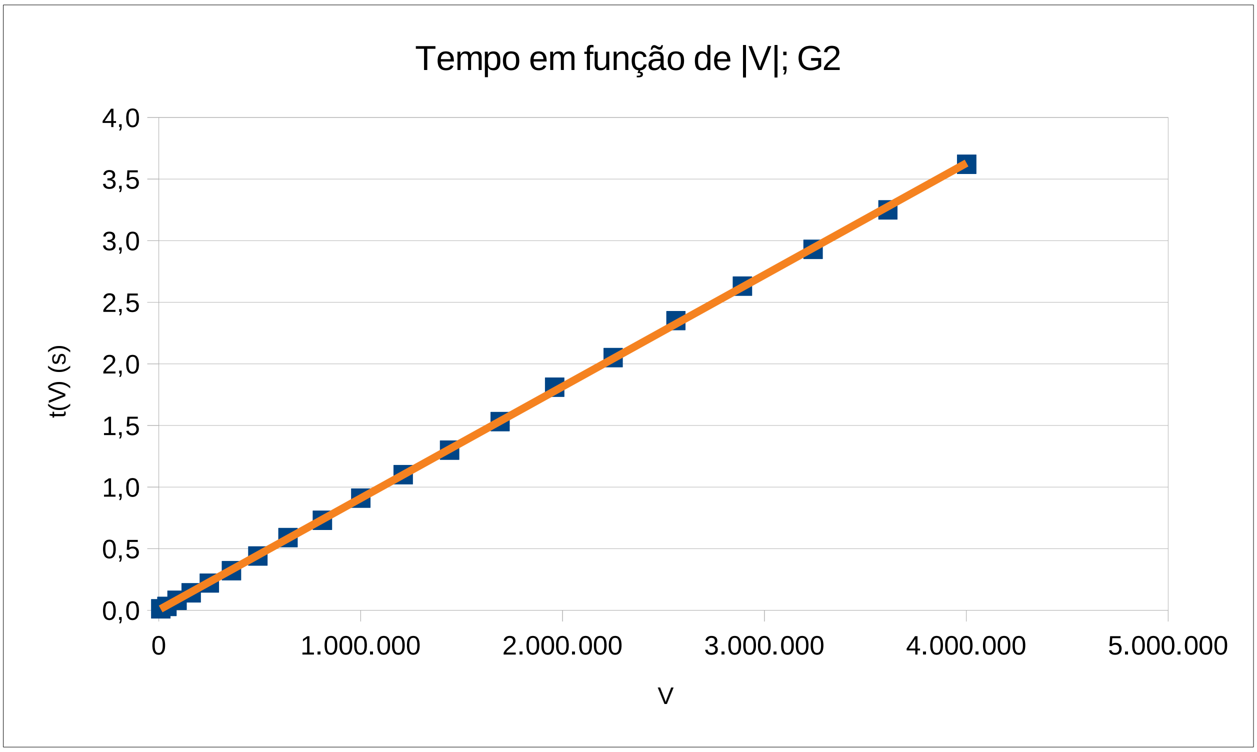
Grafos com:
→ Muitos arcos (s, u), (u, t) ∈ E, ∀u ∈ V \ {s, t} com capacidade zero (G1);
→ Arcos (s, u), (u, t) ∈ E, ∀u ∈ V \ {s, t} todos com capacidade diferente de zero (G2);
→ Em ambos os ambientes, existem alguns arcos (u, v) ∈ E, ∀u,v ∈ V \ {s, t} : u != v com capacidade zero.

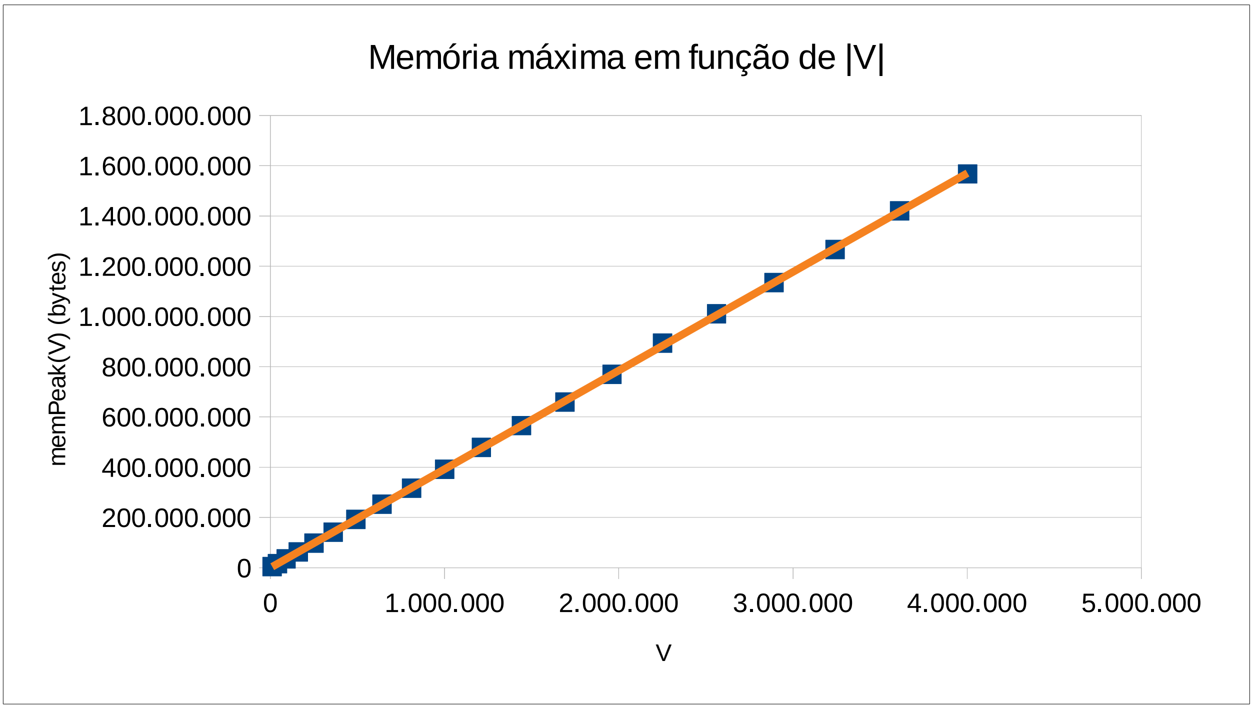
O binário utilizado não escrevia output, para calcular apenas o tempo de computação. Os testes foram realizados em modo linha de comando.

Todos os testes foram automatizados através de um *script* e correram na mesma máquina, cujas especificações estão listadas abaixo:
CPU → AMD Ryzen 7 1700X @ 3.8 GHz (8C / 16T)
RAM → DDR4 2 x 8GB @ 3200 MHz 14-14-14-36



Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526





Como seria de esperar, testes G1 geraram uma função cúbica, tal como a complexidade teórica. Testes G2 geraram uma função linear pois grande parte do trabalho foi feita durante o passo (2) da solução.

Referências:

https://en.wikipedia.org/wiki/Edmonds%E2%80%93Karp_algorithm
http://en.cppreference.com/w/
http://www.cplusplus.com/