PAA – Relatório do Trabalho Prático 2

Danilo Ferreira e Silva[1]

[1]Departamento de Ciência da Computação – UFMG

danilofs@dcc.ufmg.br

1. Introdução

Neste trabalho foram projetados e implementados algoritmos para solucionar o problema do redimensionamento de imagens usando a técnica conhecida como *Seam Carving*.

Para tal, é necessário encontrar um caminho de pixels $(1, j_1), (2, j_2), ..., (h, j_h)$ cuja soma de suas energias é a menor possível. Além disso, existe a restrição que dois pixels subsequentes deste caminho não podem estar em colunas que não sejam a mesma ou adjacentes. Ou seja, se $j_1 = 3$, j_2 poderia ser 2, 3 ou 4, mas não poderia ser 5.

A energia de cada pixel $e(i, j)$ é definida como a norma do vetor gradiente de um pixel, que pode ser calculado pelo operador de Sobel.

2. Solução em programação dinâmica

O problema do caminho de menor energia pode ser resolvido combinando subproblemas menores, beneficiando-se da propriedade de substrutura ótima.

Se $S = (1, j_1), (2, j_2), ..., (h, j_h)$ é o caminho de menor energia de uma imagem $n \times m$, o caminho $(1, j_1), (2, j_2), ..., (h-1, j_{h-1})$ também é o de menor energia terminando em $(h - 1, j_{h-1})$. Caso não fosse, existiria uma solução melhor que S e teríamos uma contradição.

Seja $e(i, j)$ a função que define a energia do pixel (i, j) da imagem. Podemos definir a função recursiva $lep(i, j)$ que é a energia do caminho ótimo que termina no pixel (i, j):

$$lep(i,j) = \begin{cases} e(i,j) & \text{se } i = 1 \\ e(i,j) + min \begin{cases} lep(i-1, j-1) \\ lep(i-1, j) \\ lep(i-1, j+1) \end{cases} & \text{se } i > 1 \end{cases}$$

Baseado nessa definição recursiva foi projetado um algoritmo de programação dinâmica que preenche uma matriz D de tamanho $n \times m$, conputando em cada célula da mesma $lep(i, j)$. Esta matriz pode ser preenchida de cima para baixo. Os valores de uma linha dependem apenas das soluções computadas na linha imediatamente acima. Note que $j - 1$ e $j + 1$ podem estar fora dos limites da imagem e nestes casos eles não são alternativas, embora na definição recursiva isso tenha sido omitido por simplicidade.

Além da matriz D é necessário armazenar uma matriz π com os antecessores de cada pixel para que seja possível reconstruir o caminho ao final do algoritmo. Na tabela 1 temos um exemplo destas matrizes preenchidas.

Tabela 1. Tabela de soluções para um subproblema específico

	$e(i,j)$			$lep(i,j)$			$\pi(i,j)$		
	1	2	3	1	2	3	1	2	3
1	111.79	536.37	869.10	111.79	536.37	869.10	nil	nil	nil
2	182.10	192.23	681.08	293.90	304.02	1217.45	1	1	2
3	330.28	333.00	93.76	624.19	626.90	397.78	1	1	2

3. Solução usando grafos

Uma segunda forma de resolver este problema é modelá-lo atravez de um grafo. Seja $G(V, E)$ um grafo contruído de tal forma que cada vértice $v_{i,j}$ corresponde a um pixel (i, j) da imagem original. Saindo de cada vértice $v_{i,j}$ são criadas arestas $(v_{i,j}, u_{i+1,j-1}), (v_{i,j}, u_{i+1,j}), (v_{i,j}, u_{i+1,j+1})$ cujos pesos correspondem a energia do pixel u.

Além disso, introduzimos um vértice S conectado a cada pixel da primeira linha da imagem, e um vértice T, ao qual cada pixel da última linha se conecta. Na figura 1 é mostrada essa transformação.

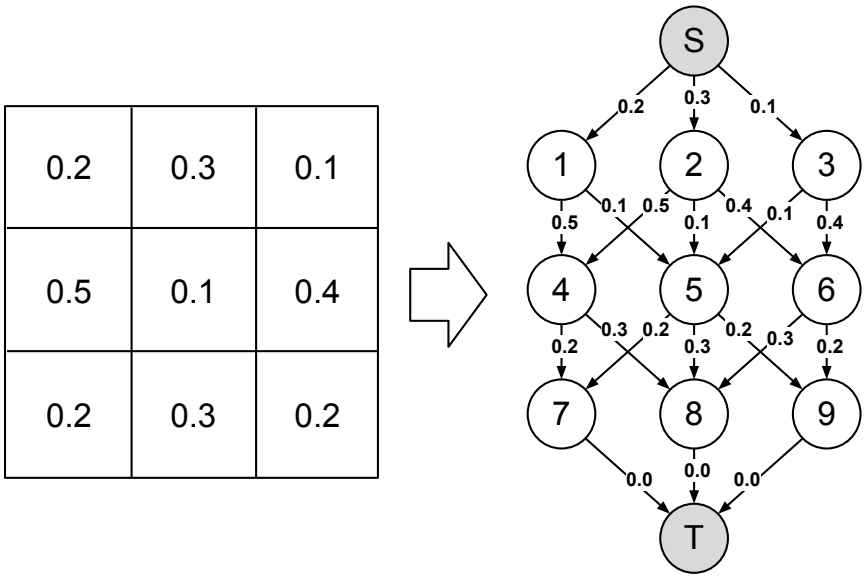

Figura 1. Transformação da imagem no grafo de energia dos caminhos

Contruído este grafo podemos reformular o problema como encontrar o menor caminho entre S e T, o que pode ser computado pelo algoritmo de Dijkstra, visto que o grafo não possui arestas com pesos negativos.

4. Análise de complexidade

Ambos os algoritmos precisam computar a energia de cada pixel uma vez. O custo desta operação depende do tamanho da matriz de pesos usados no cáculo do gradiente. Considerando que ela é uma matriz quadrada $d \times d$, o custo total do cálculo da energia de cada pixel é $O(w \times h \times d^2)$.

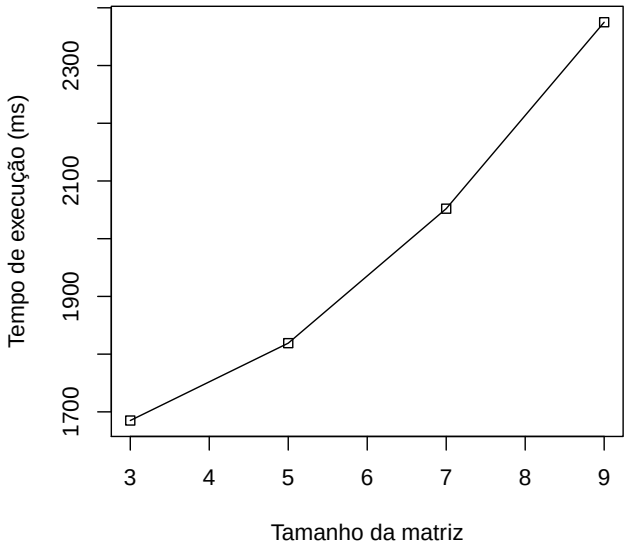

Figura 2. Tempo de execução por tamanho da matriz ($d \times d$)

Na figura 2 é traçado um gráfico do tempo de execução total da redução de 10 pixels de uma imagem fixa, variando-se a dimensão d da matriz de pesos.

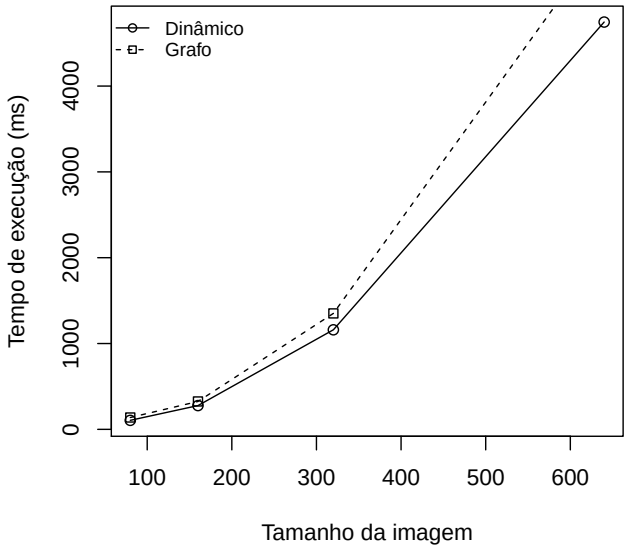

Figura 3. Tempo de execução por tamanho da imagem ($n \times n$)

O algoritmo usando programação dinâmica computa o valor da função $lep(i, j)$ para cada pixel da imagem. Considerando que a matriz gradiente tem tamanho constante,

esta operação é $O(1)$ e o custo total de encontrar o caminho mínimo é $O(h \times w)$.

Por outro lado, a solução baseada em grafos usa o algoritmo de Dijkstra, usando um heap binário como implementação de fila de prioridades, que resulta em um custo $O((V + E) \log V)$. Como o número de arestas E é limitado por V (com exceção de S, saem no máximo 3 arestas de cada vértice, podemos dizer que $E = O(3V) = O(V)$. Além disso, $V = O(hw)$, portanto a complexidade total é $O(hw \log hw)$.

O custo de memória de ambos os algoritmos é $O(hw)$. Note que no caso do algoritmo de grafo não é necessário de fato armazenas as arestas, pois elas podem ser inferidas pelos índices i, j associados ao vértice.

5. Qualidade do redimensionamento

A solução desenvolvida é capaz de reduzir as imagens sem distorções perceptíveis em algumas situações. Nas figuras 4 e 5 temos um exemplo onde os objetos de interesse foram totalmente preservados e apenas a água foi retirada.

Figura 4. Imagem original

Figura 5. Imagem com largura reduzida pela metade

Como contra exemplo, temos nas figuras 6 e 7 um caso onde o algoritmo provoca deformações severas na imagem. Intuitivamente podemos explicar este comportamento pelo fato de que o objeto de interesse, neste caso a bola, possui uma superfície lisa e com pouca variação de tonalidade, enquanto o gramado de fundo possui uma textura com variações. Por este motivo vários caminhos que atravessam a bola são retirados, deformando-a completamente.

Figura 6. Imagem original

Figura 7. Imagem com largura reduzida pela metade

A distorção da imagem 7 foi amenizada ao usar uma matriz de gradiente com dimensão 9×9, como podemos observar na figura 8.

Figura 8. Imagem com largura reduzida pela metade usando gradiente 9×9

Nas figuras 9 a 12 temos exemplos onde a redução não causou a deformações na imagem. Já as imagens 13 a 16 mostram mais casos de redimensionamentos com resultados ruins.

Figura 9. Imagem original

Figura 10. Imagem com largura reduzida pela metade

Figura 11. Imagem original

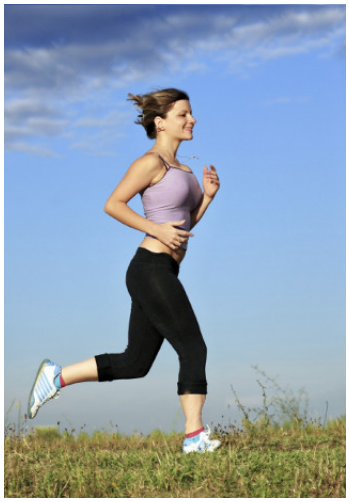

Figura 12. Imagem com largura reduzida pela metade

Figura 13. Imagem original

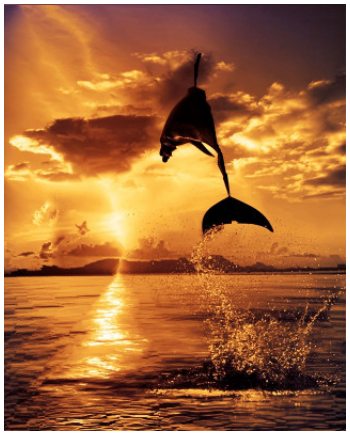

Figura 14. Imagem com largura reduzida pela metade

Figura 15. Imagem original

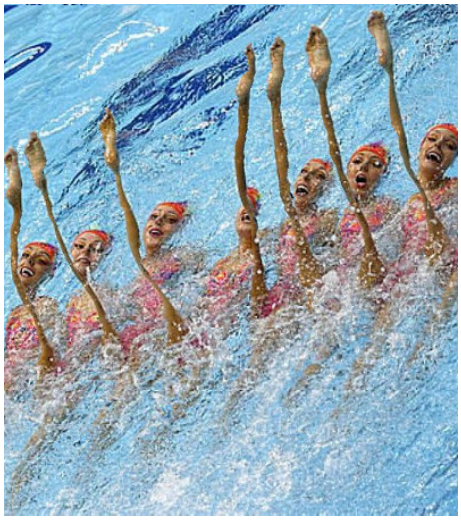

Figura 16. Imagem com largura reduzida

6. Conclusão

Neste trabalho foram projetadas e desenvolvidas duas implementações diferentes do algoritmo Seam Carving para redução de imagens, usando programação dinâmica e grafos. Ambas as soluções fornecem os mesmos resultados, por buscarem a solução ótima, sendo a solução com programação dinâmica mais eficiente em termos de custo de tempo.

Além disso foi observado que a redução da imagem com Seam Carving pode deformar objetos de interesse se eles possuirem regiões de pouca variação na tonalidade se comparados ao restante da imagem.