Trabalho Prático 2

Jean George Alves Evangelista - 2018047021

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

jeanalves@dcc.ufmg.br

1. Introdução

Conforme descrito no enunciado, o objetivo deste trabalho é implementar um programa na linguagem de programação C/C++ que resolva o problema das amigas que desejam visitar um conjunto de ilhas no Panamá. Cada ilha recebe uma pontuação e a ela é relacionada uma quantia em dinheiro que representa os custos da estadia de um dia. Dado um inteiro N, que representa o valor máximo possível de ser gasto por cada amiga e um conjunto de M ilhas, o programa deve apresentar duas soluções:

1. Utilizando um algoritmo com paradigma guloso, determinar (em complexidade de no máximo O(M log M)):
    a. A maior pontuação possível caso seja possível permanecer mais de um dia na mesma ilha.
    b. A quantidade dias que durará a viagem.
2. Utilizando um algortimo com paradigma dinâmico, determinar (em complexidade de no máximo O(M*N)):
    a. A maior pontuação possível caso não seja possível permanecer mais de um dia na mesma ilha.
    b. A quantidade de dias que durará a viagem.

O algoritmo deve primeiramente ler a o custo máximo em reais a ser gasto na viagem e a quantidade de ilhas pontuadas no problemas. Em seguida, devem ser lidos o custo de ficar um dia numa ilha e a pontuação dada para aquela ilha. Após a leitura o algoritmo deve apresentar as duas soluções descritas acima.

2. Implementação

Optou-se por realizar a implementação na linguagem de programação C++. O programa possui duas classes: *Ilha* e *Viagem*.

2.1 Classes

Ilha: Classe bastante simples criada para representar uma ilha no contexto do trabalho. Possui como atributos dois inteiros que representam o custo e a pontuação da ilha. Além disso foram criados três métodos do tipo *getter* para obter os valores do custo, da pontuação e do custo benefício (custo dividido pela pontuação).

Viagem: Classe principal do programa. Possui como atributos o custo total da viagem, a quantidade total de ilhas que as meninas desejam visitar e um vetor do tipo *Ilha*, representando as ilhas propriamente ditas. Como métodos, destacam-se dois para o cálculo de uma viagem com repetição de ilhas (com programação gulosa) e sem repetição de ilhas (com programação dinâmica), na seção abaixo está explicado melhor cada método. Foram implementados outros métodos convenientes/auxiliares dentro da classe *Viagem*, como um método *merge sort* para ordenar as ilhas por custo-benefício de forma decrescente e um método que retorna o máximo entre dois inteiros recebidos.

2.2 Descrição da execução, dos principais métodos e das escolhas de implementação

2.2.1 main.cpp

Como todo programa desenvolvido em C++, a execução inicia-se no arquivo *main.cpp*. Na função *main* a primeira coisa feita é tentar abrir o arquivo (cujo nome foi recebido como parâmetro na execução) contendo as entradas, caso o arquivo não seja aberto por qualquer motivo o programa se encerra. Em seguida, inicia-se o processo de leitura do arquivo conforme especificado: obtêm-se o custo total e a quantidade total de ilhas, respectivamente. É criado um vetor de ilhas, utilizando alocação dinâmica, de acordo com a quantidade de ilhas. Um laço do tipo *for* é executado para realizar a leitura do custo diário e da pontuação de cada ilha e esses valores são armazenados em instâncias do vetor de ilhas. Em seguida o arquivo que foi aberto para a leitura é fechado - caso não seja fechado por qualquer motivo o programa se encerra.

Nesse ponto da execução já foram lidos e armazenados o custo total, a quantidade de ilhas e todas as ilhas. É instanciado então um objeto do tipo *Viagem*, que recebe o custo, quantidade e as ilhas. São chamados os métodos *viagemComRepeticao* e *viagemSemRepeticao* do objeto do tipo *Viagem*, implementado em Viagem.hpp e Viagem.cpp.

2.2.2 Viagem.cpp

O arquivo Viagem.cpp implementa os métodos da classe *Viagem*, definidos em Viagem.hpp.

2.2.2.1 Método construtor

O construtor simplesmente define o custo total, a quantidade de ilhas e o vetor de ilhas

2.2.2.1 Método viagemComRepeticao

Conforme especificado, deveria ser feito o cálculo da pontuação máxima e dos dias que durarão uma viagem (permitindo repetições de visitas) por meio de um algoritmo guloso. Primeiramente o método ordena as ilhas por custo-benefício (decrescente) utilizando *merge sort*.[1] Em seguida basta percorrer o vetor de ilhas, verificando sempre a cada iteração se há dinheiro disponível a ser gasto (inicialmente o dinheiro disponível é o que foi lido na entrada, mas a medida das iterações essa quantia diminui). Numa iteração, caso haja dinheiro disponível para gastar é calculado quantas vezes é possível visitar a ilha em questão. Essa quantidade de vezes é somada aos dias já calculados (inicialmente a contagem de dias inicia-se com 0) e é utilizada para atualizar a pontuação total (que também se inicia com 0). No fim da iteração o dinheiro disponível é atualizado. O processo se repete até percorrer todas as ilhas. No fim a pontuação total e a quantidade de dias da viagem.

2.2.2.1 Método viagemSemRepeticao

Além de um método que permita repetições de visitas diárias a ilhas, deveria ser implementado outro que calcula-se a pontuação e o total dos dias sem repetições, por meio de um algoritmo em programação dinâmica. O método *viagemSemRepeticao* faz isso utilizando uma estratégia bem similar à realizada no *Knapsack Problem* visto em sala de aula. Foi empregada uma estratégia do tipo *bottom-up*, que calcula todos os valores até chegar no valor de entrada. É feita uma matriz M*N, sendo M o número total de ilhas e No custo total. A matriz é populada avaliando-se três possibilidades:

1. Caso estejamos na primeira linha na primeira coluna da matriz é armazenado 0. Isso pode ser interpretado como, “se eu tenho 0 ilhas, qual será meu custo?” ou “se eu tenho um custo 0, quantas ilhas tenho?”.
2. Caso o custo daquela ilha se encaixe, avaliamos se a solução ótima contém ou não contém aquela ilha. Isso é feito avaliando o máximo entre a solução sem aquela ilha (já calculado e com valor armazenado na matriz) e uma solução com ela (custo da solução sem essa ilha + pontuação dessa ilha).

[1] Explicação da implementação foi omitida por ser julgado que não se fazia necessária. Foi utilizado *merge sort* para obter-se a melhor complexidade possível na ordenação: O(M log M), sendo M o número de ilhas.

3. Caso o custo daquela ilha não se encaixe é possível concluir que tal ilha não estará na solução, então é armazenado a solução previamente calculada.

Após o preenchimento, o valor presente na última linha e última coluna é a solução (pontuação máxima). Para calcular a duração da viagem é preciso fazer o caminho reverso (*backtracking*), avaliando quais linhas estão presentes na solução. A matriz é percorrida da última posição para a primeira. Um valor da matriz será desconsiderado caso seja igual ao valor presente na mesma coluna e uma linha cima. Será considerado caso contrário, sendo acrescido 1 ao número total de dias (que inicia-se com 0). No fim, teremos o total de dias da viagem.

3. Instruções de Compilação de Execução

Compile com make e rode com ./tp2 <nomedoarquivodeentrada>.

4. Análise de Complexidade

Será utilizado M para abreviar "quantidade total de ilhas" e N para abreviar "custo total da viagem".

4.1 Complexidade de Espaço

Avaliando o programa como um todo, vemos que a maior quantidade de memória alocada é para a variável que representa a matriz utilizada no cálculo com o algoritmo dinâmico. Essa matriz possui dimensões M*N, portanto a complexidade total do programa será O(M*N). Também é possível realizar uma análise de complexidade de espaço de outras partes do programa:

- Leitura da entrada: O(M), por conta do vetor de tamanho igual à quantidade total de ilhas.
- Cálculo com o método guloso: O(M), considerando tanto a memória alocada para ordenação quanto a memória dentro do próprio método.

4.1 Complexidade de Tempo

A complexidade tempo de cada parte considerada importante no programa é analisada abaixo:

- Leitura da entrada: O(M), tendo em vista que há um laço *for* de 0..M.
- Cálculo da solução com o método guloso: O(M log M), valor que obtido avaliando o custo do *merge sort* (O(M log M)) mais o custo para percorrer o vetor ordenado (O(M)).
- Cálculo da solução com o método dinâmico: O(M*N), tendo em vista que são executados dois laços do tipo *for* aninhados (um laço 0..N dentro de um 0..M) e tudo o que é feito dentro dos laços tem custo constante ou O(1).

5. Prova de corretude

Para provar a corretude do método guloso e do método dinâmico é preciso provar que ambos terminam e retornam a solução ótima.

5.1 Método Guloso

- Término - O método guloso terminará se:
    a. A função do *merge sort* para ordenação terminar. Isso fatalmente acontecerá, tendo em vista que foi implementado um *merge sort* clássico.
    b. Não entrar num loop infinito. Isso facilmente pode ser verificado, tendo em vista que o único laço de repetição do método itera exatamente M vezes, sendo M o número total de ilhas.
- Solução Ótima - O método retornará a solução ótima se:
    a. A função de ordenação ordenar corretamente o vetor de ilhas por custo benefício em ordem decrescente, fato que podemos assumir que acontece por se tratar de um *merge sort* clásssico.
    b. A solução final deve for ótima. Sabemos que as ilhas estão ordenadas por custo-benefício, como a repetição é permitida a solução ótima deverá conter o maior número possível da ilha com menor custo benefício, em seguida o maior possível da ilha com segundo menor custo benefício e assim por diante. O que foi descrito é exatamente o que o método guloso implementa.

5.1 Método Dinâmico

- Término - O método dinâmico terminará se:
    a. Não entrar num loop infinito. Não ocorre loop infinito devido ao fato de todos os laços de repetições serem executados uma quantidade linear (varia com a entrada) de vezes.
- Solução Ótima - O método retornará a solução ótima se:
    a. Cada posição [i][j] da matriz A, de dimensões M*N deve conter a solução ótima até a quantidade de ilhas i e custo total j. Assuma que A[i][j] seja a solução ótima até os índices i,j. Precisaremos então calcular o valor de A[i+1][j], que pode ser dado avaliando-se o máximo entre escolher esse elemento ([Ai][j-w] + v, sendo w o peso e v o valor) e não escolher o elemento (A[i][j]). Tanto A[i][j-w] como A[i][j] já foram calculados no momento que estivermos avaliando A[i+1][j] e ambas são ótimas (por indução, visto que A[i][j] é solução ótima), podemos concluir então que em A[i+1][j] teremos também uma solução ótima. Esse foi o raciocínio utilizado para resolução, que como dito se assemelha bastante ao *Knapsack Problem*.

6. Avaliação Experimental

Foram realizados testes com entradas com 4, 8, 12, 16 e 20 ilhas. Para cada teste, mediu-se 10 vezes os tempos de execução do método guloso e dinâmico (foram medidos separadamente). No fim calculou-se a média do tempo e o desvio-padrão do tempo. Abaixo temos representações dos resultados obtidos em forma de tabela e de gráfico.

Tabela 1. Resultados obtidos para o método guloso.

LINHAS	MÉDIA	DESVIO PADRÃO
4	6,22	0,60
8	8,20	0,63
12	9,80	0,87
16	12,30	0,90
20	14,30	0,90

Gráfico 1. Resultados obtidos para o método guloso.

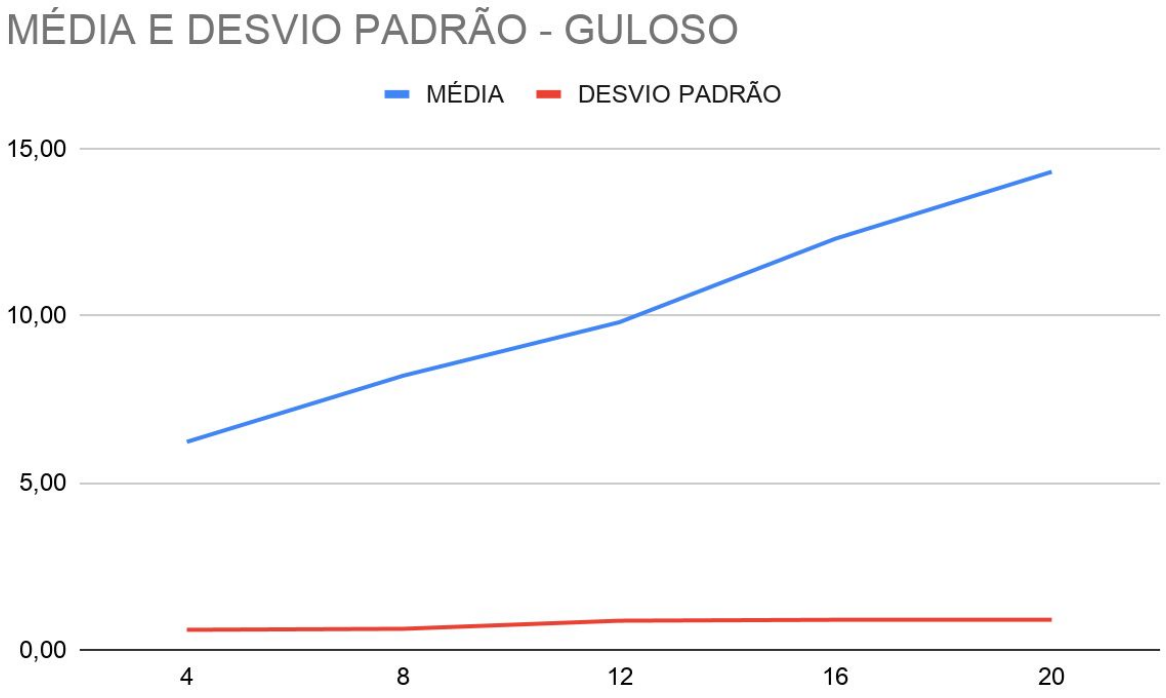
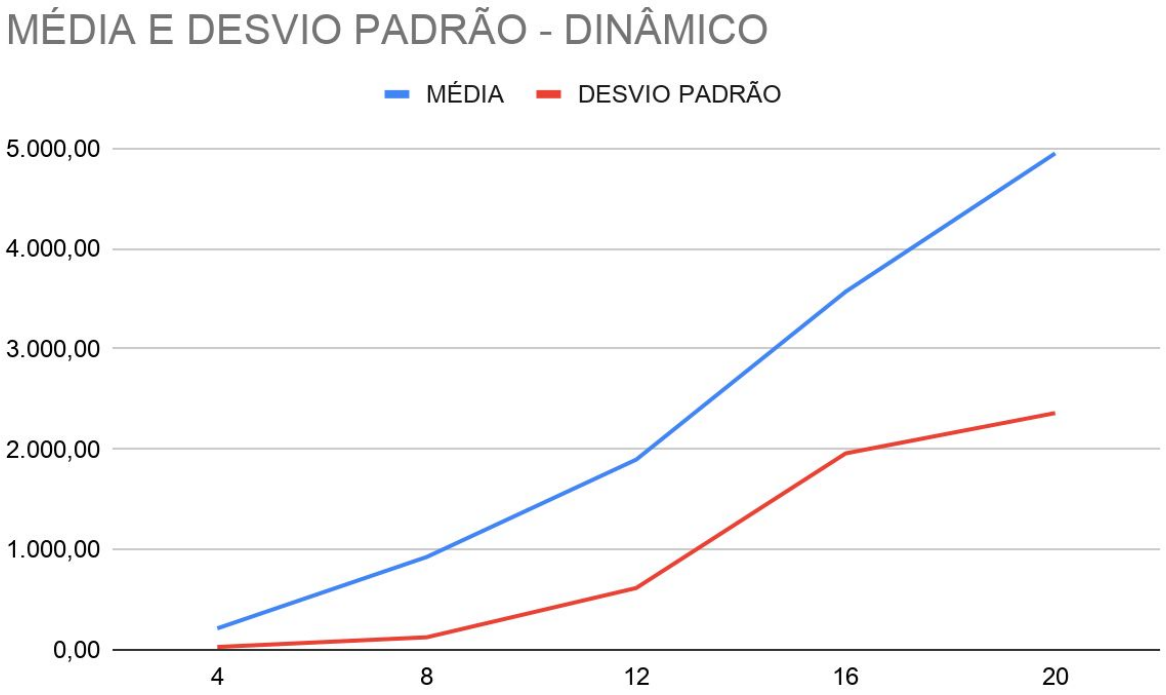


Tabela 2. Resultados obtidos para o método guloso.

Linhas	MÉDIA	DESVIO PADRÃO
4	214,22	29,03
8	926,33	125,72
12	1.897,22	616,81
16	3.571,56	1.959,25
20	4.950,40	2.361,24

Gráfico 2. Resultados obtidos para o método guloso.



Em todos os testes o valor do custo das ilhas é bem maior que o valor da pontuação, então os valores observados nas tabelas e gráficos acima já eram esperados. Observa-se também que ambos os métodos tendem a ter um tempo de execução maior ao aumentar o tamanho da entrada, o que também já era esperado. Por último, vale citar o fato do método dinâmico ter um tempo de execução muito maior se comparado ao método guloso, o que é compreensível visto que no método guloso é utilizada uma matriz M*N para o cálculo.

7. Conclusão

Conclui-se que foi possível implementar um programa funcional que apresente uma solução para o problema descrito no enunciado, respeitando as restrições de complexidade. Por último, é possível afirmar que o trabalho foi bastante interessante e proveitoso, por permitir compreender e colocar em prática conceitos e técnicas de diferentes paradigmas de programação.

Referências

StackOverflow. Disponível em: <https://stackoverflow.com/>. Acesso em: 26 out. 2019.

C++ Language. Disponível em: <http://www.cplusplus.com/doc/tutorial/>. Acesso em: 26 out. 2019.

Geeks for Geeks. Disponível em: <https://www.geeksforgeeks.org/selection-sort/>. Acesso em: 26 out. 2019.

Lecture Slides for Algorithm Design by Jon Kleinberg And Éva Tardos. Princeton Univeristy. Disponível em: <https://www.cs.princeton.edu/~wayne/kleinberg-tardos/>. Acesso em: 26 out. 2019.