Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

TRABALHO PRÁTICO: VIAGEM ÀS ILHAS

Victor Gabriel Ferreira Moraes

Belo Horizonte,

2019

Descrição do problema

O contexto deste trabalho é referente ao projeto de um grupo de amigas, as quais pretendem realizar uma viagem ao Arquipélago de San Blas, que se trata de um conjunto de 365 ilhas pertencentes ao Panamá. Durante o planejamento da viagem, as amigas se reuniram para dar uma pontuação e estipular o preço necessário para passar um dia em cada ilha.

Devido ao orçamento limitado, surgiu a necessidade de determinar dois tipos de roteiro baseado nesse limite, um que fosse capaz de determinar a maior pontuação possível dado um conjunto de ilhas, podendo ocorrer repetição de ilhas (ficar mais de um dia na mesma ilha), e outro roteiro que considerasse a maior pontuação possível dado um conjunto de ilhas, sem que haja a repetição das mesmas.

Posto isso, a solução proposta neste trabalho, consiste em um sistema capaz de realizar uma simulação, baseada no valor total do orçamento de cada amiga e na quantidade de ilhas (juntamente do preço de estadia diário e da pontuação referente a cada ilha), que calcule a melhor combinação possível de ilhas a serem visitadas, para os dois tipos de roteiro (com e sem repetição), tendo como resultado a pontuação máxima possível e a quantidade de dias que durará a viagem para os dois casos.

Princípios de projetos adotados

O sistema desenvolvido foi estruturado de modo a conter as seguintes classes: main.cpp, Viagem.cpp e Ilha.cpp, as quais fizeram uso de estruturas de dados auxiliares (std::vector) e métodos (std::sort) da biblioteca padrão da linguagem C++.

Em relação aos algoritmos utilizados para realizar o objetivo central do problema, esses foram desenvolvidos respeitando a complexidade limite estabelecida pelo enunciado, sendo O(n log(n)) para o algoritmo guloso (com repetição), e O(nm) para o algoritmo de programação dinâmica (sem repetição),

onde n corresponde a quantidade de ilhas, e m representa o valor máximo possível a ser gasto por cada amiga.

Além disso, os conhecimentos teóricos a respeito de algoritmos gulosos, programação dinâmica e, em específico, sobre o problema da mochila, foram essenciais no processo de desenvolvimento do projeto.

Classes

Ilha.cpp

A classe Ilha.cpp corresponde a uma estrutura de dados desenvolvida para representar as ilhas do problema e possui quatro atributos, sendo eles custoDiario (responsável por representar o custo de estadia diário da ilha); pontuacao (representando a pontuação da ilha em específico); custoPorPontuacao (razão entre o custoDiario e a pontuacao); e custoMaximo (representando o orçamento máximo a ser gasto na viagem).

Em relação aos métodos, a classe possui apenas um, calculaCustoPorPontuacao(), responsável por inicializar o valor de custoPorPontuacao. Por fim a classe possui um modificador do operador de comparação ‘<’, a fim de realizar a ordenação dos conjuntos de ilhas com base nos atributos de custoPorPontuacao e custoMaximo.

Viagem.cpp

A classe Viagem.cpp, corresponde ao componente principal para a solução do problema em questão, pois ela é responsável por executar os algoritmos destinados para a resolução de ambos os dois tipos de roteiro (com e sem repetição). Essa classe é composta basicamente por apenas dois métodos: viagemComRepeticaoIlha(...) e viagemSemRepeticaoIlha(...), esse dois métodos recebem como parâmetro um vetor de ilhas, a quantidade de ilhas contidas neste vetor, o custo máximo a ser gasto (orçamento), e mais dois parâmetro passados por referência para representar os resultados, referentes a pontuação máxima e a quantidade de dias.

Main.cpp

Por último, temos a classe main.cpp, que é encarregada de: realizar a leitura dos parâmetros de entrada; inicializar as estruturas de dados e criar o vetor responsável por representar o conjunto de ilhas; chamar os métodos de simulação do resultado para o roteiro de viagem com e sem repetição de ilhas; e por fim, realizar a escrita do resultado obtido.

Problema da mochila

Os dois principais métodos desenvolvidos nesse projeto, foram ambos baseados em um problema denominado “problema da mochila”, que corresponde a uma determinada situação, onde se tem uma mochila com um dado limite de peso, e diferentes objetos com valores e pesos respectivos. Nessa situação, tem-se como objetivo preencher essa mochila com os objetos que totalizam o maior valor possível, não ultrapassando o limite de peso estabelecido. Dessa forma, tem-se duas abordagens desse tipo de problema, por meio do algoritmo guloso e pelo algoritmo de programação dinâmica.

Algoritmo Guloso

A abordagem do algoritmo guloso nesse trabalho, corresponde a implementação do problema da mochila onde é possível adicionar itens fracionados. Dessa forma, essa abordagem foi utilizada para implementar a função de simulação da viagem com repetição de ilhas, visto que permite a adição de mais de um item (ilhas) na mochila (orçamento).

O algoritmo funciona da seguinte forma: de forma gulosa, calcula a razão entre o valor da diária e a pontuação de cada ilha, e em seguida ordena o conjunto de ilhas com base nessa razão. Então, é selecionado a ilha com menor valor por pontuação e a mesma é adicionada à solução o máximo de vezes possível, em seguida repete-se essa etapa para a próxima ilha do conjunto ordenado, até não ser possível adicionar mais nenhuma ilha. Dessa forma, a solução encontrada é sempre

ótima para essa abordagem e por isso foi selecionada para realizar a simulação do roteiro de viagem com repetição de ilhas.

Algoritmo Programação Dinâmica

No caso da abordagem do algoritmo de programação dinâmica, essa é referente ao problema da mochila onde não é possível adicionar itens repetidos à mesma, dessa forma os itens são tratados como únicos. Sendo assim, fez-se o uso dessa abordagem para solucionar o problema de simulação de viagem sem repetição de ilhas.

A solução foi construída da seguinte forma: para cada ilha, ou essa ilha é inclusa no conjunto solução, ou não. Sendo n o número total de ilhas, temos os dois cenários:

1. A pontuação máxima obtida pelas n-1 ilhas e o limite de orçamento m, de modo que a ilha de índice n, não é incluída na solução.
2. A pontuação da ilha de índice n mais a pontuação máxima obtida pelas n-1 ilhas e o limite de orçamento m subtraído do valor diário da ilha n, nesse caso, inclui-se a ilha de índice n à solução.

Conforme apresentado, em teoria esse algoritmo corresponde a uma estrutura que divide a solução em subproblemas menores até resolver o caso base e recursivamente soluciona os casos anteriores chegando ao resultado. O problema dessa abordagem recursiva, é que na prática, pode ocorrer de um dado subproblema ser computado mais de uma vez, gerando o que se chama de sobreposição de subproblemas.

A fim de evitar essa situação, foi adotado na solução desse trabalho uma abordagem desse algoritmo, denominada bottom-up, onde não se faz uso de recursão, mas sim de uma matriz auxiliar de dimensões n por m, sendo n a quantidade de ilhas e m o limite do orçamento, que guarda os resultados calculados a partir do caso base.

Análise de Complexidade

Tempo

O sistema desenvolvido pode ser analisado em relação à sua complexidade, através das classes que o compõe. Sendo assim, começamos analisando a classe Ilha.cpp, a qual corresponde a uma classe criada para representar uma estrutura de dados auxiliar para o problema, possuindo um único método, calculaCustoPorPontuacao(), que apenas inicializa um de seus atributos, tem-se portanto, complexidade constante O(1).

Em relação à classe Viagem.cpp, essa possui dois métodos que devem ser analisados de forma separada, viagemComRepeticaoIlha(...) e viagemSemRepeticaoIlha(...). O primeiro, como já mencionado, é baseado no algoritmo da mochila fracionado, sendo assim, logo de início, o método realiza uma ordenação no conjunto de ilhas recebidos, gerando uma complexidade da ordem de O(n log n), por fim o método realiza um ‘for’ iterando sobre as n ilhas, a fim de calcular a melhor pontuação possível com base no custo total de estadia, possuindo portanto complexidade O(n), logo a complexidade assintótica resultante desse método é da ordem de O(n log n).

Já no caso do segundo método, esse se baseia no algoritmo da mochila sem repetição o qual de início realiza um conjunto de dois for aninhados, que percorrem as n ilhas e a cada ilha iteram até o valor de custoMaximo (orçamento limite) m, gerando uma complexidade O(nm). Em seguida, é realizado um for de modo a fazer um ‘backtracking’ na matriz de pontuações, a fim de obter as ilhas selecionadas e a quantidade de dias, nesse processo, são realizados no máximo n iterações (quando se percorre todas as ilhas), logo sua complexidade é O(n). Posto isso, a complexidade assintótica resultante do segundo método é da ordem de O(nm).

Por fim, temos a classe main.cpp, que realiza a leitura dos dados e inicializa o conjunto das n ilhas, imprimindo o resultado da simulação ao final, portanto possuindo uma complexidade linear (O(n)).

Dessa forma, é possível perceber que a complexidade de tempo exigida neste trabalho foi respeitada, sendo O(n log n) para o algoritmo guloso, e O(nm) para o algoritmo de programação dinâmica.

Espaço

Referente a complexidade de espaço do sistema desenvolvido, é possível observar que o mesmo faz uso da estrutura de dados vector, padrão da biblioteca, a fim de representar o conjunto de n ilhas, gerando uma complexidade linear O(n). Além disso, é utilizado uma matriz de pontuações de dimensões n por m, no caso da abordagem bottom-up do algoritmo de programação dinâmica do problema da mochila, portanto sua complexidade de espaço é da ordem O(nm). Dessa forma, a complexidade assintótica de espaço do sistema apresentado é da ordem de O(nm).

Prova de corretude do algoritmo

O algoritmo utilizado na construção do sistema em questão, foi baseado no “problema da mochila”, de modo que o método de resolução do problema das ilhas retratado, utiliza uma abordagem análoga ao algoritmo da mochila. Dessa forma, pode-se provar a corretude do algoritmo deste trabalho, através da prova do algoritmo do “problema da mochila”. Sendo assim, temos:

- Por definição, considere OPT(i, w) como o subconjunto ótimo de maior valor dentre os itens 1 até i, com peso limite w.
- Pretende-se provar OPT(n, W).
- Dessa forma, temos dois casos para o problema:
    - Caso 1: OPT(i, w) não seleciona o item i (possível visto que pode ser que wi > w, onde wi corresponde ao peso do item i).
        - Sendo assim, OPT(i, w) seleciona o maior valor do subconjunto {1, 2, …, i-1}, usando o peso limite w.
    - Caso 2: OPT(i, w) seleciona o item i.
        - Adiciona-se o valor do item i, vi.
        - Recalcula o peso limite tal que w = w - wi.

    - OPT(i, w) seleciona o maior valor do subconjunto {1, 2, …, i-1}, usando o novo peso limite w calculado.
- Logo, temos provado OPT(n, W), por meio de uma prova de troca de argumentos, portanto o algoritmo é ótimo.

Avaliação experimental

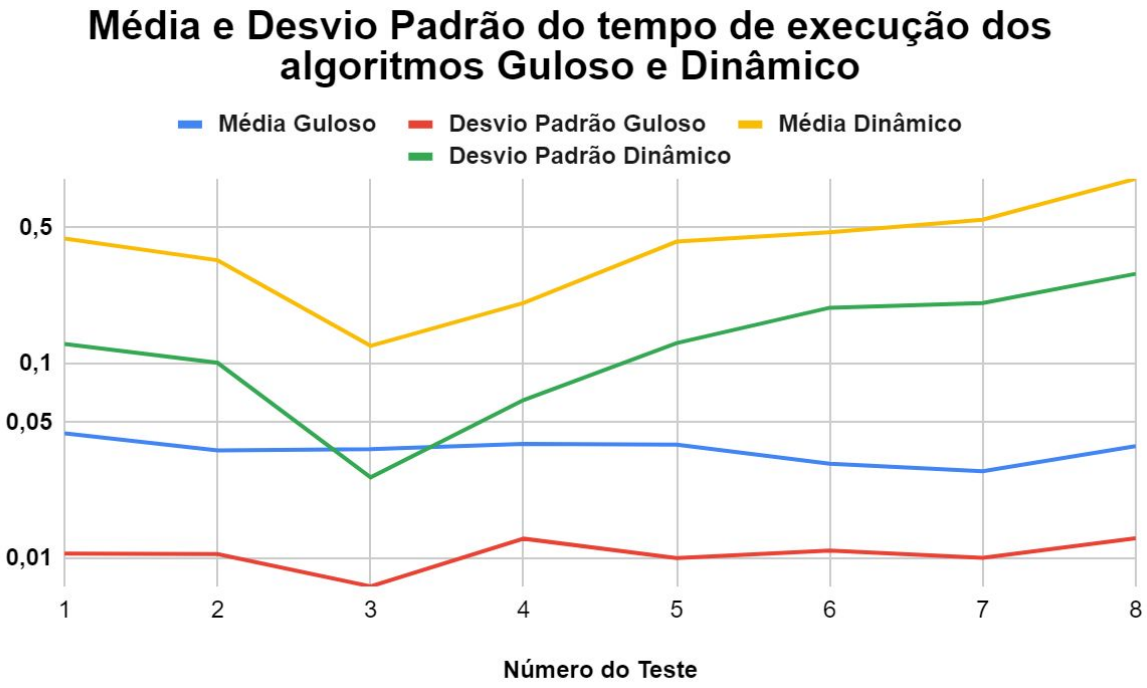
Análise de tempo de execução

Teste	Limite Orçamento (m)	Quantidade Ilhas (n)
1	6000	5
2	6000	5
3	2250	4
4	5000	3
5	5000	8
6	5000	12
7	5000	16
8	5000	20

Após executar os casos de teste acima, 10 vezes cada um, foi possível obter uma média e desvio padrão do tempo de execução para cada abordagem de algoritmo executado (Guloso e Dinâmico), compondo a seguinte tabela:

	Guloso		Dinâmico	
Teste	Média Guloso	Desvio Padrão Guloso	Média Dinâmico	Desvio Padrão Dinâmico
1	0,0439	0,01055619881	0,4397	0,1263751470
2	0,0358	0,01049656028	0,3400	0,1010060504
3	0,0364	0,00716782936	0,1235	0,0260266957
4	0,0387	0,01260555433	0,2051	0,0650255505
5	0,0384	0,01002441464	0,4239	0,1275813031
6	0,0306	0,01094633373	0,4733	0,1938934702
7	0,0280	0,01003327796	0,5499	0,2047852262
8	0,0377	0,01267587385	0,8918	0,2897641953

Ao plotar esses resultados obtidos, é possível gerar o seguinte gráfico:



Dessa forma, de acordo com o que foi apresentado e ao realizar uma análise do gráfico, é possível observar que os valores obtidos na média e desvio padrão do algoritmo dinâmico, dominam assintoticamente os valores obtidos na média e desvio padrão do algoritmo guloso, respectivamente. Além disso, é possível notar que a partir do caso de teste número 4 (onde m se mantém fixo e n passa a crescer), no caso dinâmico, o comportamento da função é análogo à f(nm), já no caso guloso, temos uma tendência à função f(n log n). Dessa forma, a complexidade de tempo gerada por essas duas abordagens é exatamente conforme o esperado, onde o algoritmo guloso é da ordem de O(n log n) e o algoritmo dinâmico se comporta conforme O(nm), dominando assintoticamente a abordagem gulosa.

Discussão das abordagens

Conforme observado ao longo do trabalho, foi possível traçar as diferenças entre as distintas abordagens (Algoritmo Guloso e Algoritmo de Programação Dinâmica) do problema apresentado. De tal forma, tornou-se perceptível que o roteiro de viagem que permitia repetição de ilhas, abordagem gulosa, favorecia a

estadia em um mesmo local por um período maior de tempo. Já no caso do roteiro de viagem sem repetições de ilhas, abordagem dinâmica, esse por consequência, permitia que fosse visitado uma maior quantidade de lugares diferentes.

Como conclusão da análise dos resultados dos testes realizados, foi constatado que na maior parte dos casos, o roteiro do algoritmo guloso, resultava em uma pontuação e duração da viagem maior do que a do roteiro do algoritmo dinâmico, porém, esse por sua vez, permitia viajar por uma quantidade maior de ilhas distintas.

Conclusão

Conclui-se então que foi possível alcançar o objetivo inicial do trabalho, ao realizar o desenvolvimento de um sistema que soluciona o problema de viagem proposto sob a perspectiva de dois roteiros distintos, guloso e dinâmico, atendendo às especificações de complexidade O(n log n) e O(nm), respectivamente.

Ademais, o trabalho proporcionou a oportunidade de reforçar conceitos teóricos já trabalhados como o “problema da mochila”, algoritmos gulosos e programação dinâmica no geral, aliando a parte teórica à prática e servindo assim como forma de consolidar os conhecimentos referentes à disciplina de Algoritmos 1.

Referências

- GeeksForGeeks. Fractional Knapsack Problem. Disponível em: <https://www.geeksforgeeks.org/fractional-knapsack-problem/>. Último acesso em: 24 de Outubro de 2019.
- GeeksForGeeks. 0-1 Knapsack Problem | DP-10. Disponível em: <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>. Último acesso em: 24 de Outubro de 2019.
- GeeksForGeeks. Printing Items in 0/1 Knapsack. Disponível em: <https://www.geeksforgeeks.org/printing-items-01-knapsack/>. Último acesso em: 24 de Outubro de 2019.