Documentação do TP1 de Algoritmos 2

Lucas Resende Pellegrinelli Machado

Outubro 2019

1 Introdução

O trabalho proposto diz respeito a amigas que querem viajar juntas para algumas ilhas. A decisão deve envolver tanto o preço da estadia de cada ilha tanto o quanto elas querem ir para aquela ilha e a resposta deve ser o melhor balanço dessas variáveis de modo a elas gastarem o possível para serem o mais felizes possíveis. Duas opções devem ser dadas a elas: uma opção que não deixa repetir a estadia em uma ilha e uma outra opção onde elas podem ou não ficar mais de um dia em cada uma das ilhas.

2 Modelagem

O problema pode ser modelado como um problema similar ao famoso problema "Knapsack" onde existe uma mochila com uma certa capacidade e vários itens, cada um com um valor e um peso. A ideia é encontrar quais e quanto de cada um dos itens deve ser levado na mochila de forma a não exceder o peso limite e maximizar o valor dos itens levados. Ao mudar esse problema para servir para o problema do trabalho, consideramos o peso limite da mochila igual ao dinheiro que as meninas tem disponível (a ideia é a mesma para os dois, limitar a quantidade de itens a ser levado) e o valor de cada item será o quanto elas querem ir para aquela ilha (que tem a mesma ideia visto que ambos servem para priorizar um item à outro). Dessa forma podemos utilizar os mesmos algoritmos utilizados para a resolução do problema do "Knapsack" para esse trabalho.

3 Implementação

Para a implementação, foi requerido o uso de dois paradigmas da computação: uma resolução usando um algoritmo guloso e outro usando um algoritmo dinâmico. Para a armazenagem dos dados, foi utilizado um `std::vector` que guarda informações do tipo `std::pair<int, int>`, que correspondem a cada uma das ilhas. O primeiro valor do `std::pair` corresponde ao custo da ilha e o segundo valor corresponde à pontuação dada pelas amigas.

3.1 Algoritmo Guloso

No algoritmo guloso a ideia era ordenar as ilhas pelo seu custo-benefício, ou seja, ordenar de forma decrescente em relação a $\frac{pontuacao}{custo}$. Isso foi feito utilizando o `std::sort` da biblioteca padrão do C++, que pode ser modificado a partir de uma função de comparação para comparar os itens de um `std::vector` da forma que quisermos. Dessa forma criei uma função lambda que definia que a ordenação deveria ser feita em função de $\frac{2 \text{ valor do pair}}{1 \text{ valor do pair}}$.

Depois de ordenado, passamos por cada uma das posições do vetor e, caso o preço da ilha pudesse ser pago, as meninas ficariam mais um dia na ilha e então caso não fosse possível mais ficar naquela ilha devido aos custos, passariamos para a próxima, até o fim do vetor. Esse algoritmo não é perfeito e pode falhar dependendo das entradas, mas com um algoritmo guloso é o mais próximo de uma resposta ótima que podemos chegar.

3.2 Algoritmo Dinâmico

Já no algoritmo dinãmico foi escolhido usar a recursão visto que o código ficaria mais limpo e fácil de entender. Para memeizar os valores já calculados, ao invés de uma matriz, foi utilizado um `std::unordered_map`, que é essencialmente um Hashmap, ou seja, ele guarda uma chave associada a um valor em uma posição de memória que pode ser inferida a partir do valor da chave, tornando a busca por essa chave independente do número de valor guardados nesse Hashmap. Isso é interessante pois faz com que tanto a inserção quanto a busca de valores seja $O(1)$. Essa complexidade é igual a de uma matriz de resultados parciais, porém melhor otimizado espacialmente visto que é possível que na matriz, grande parte das posições não seja utilizada, e no mapa, só é salvo valores que serão utilizados. Note que a estrutura de dados usada foi de fato a `std::unordered_map` e não a `std::map` visto que a posterior é a implementação de uma árvore, ou seja, tanto a inserção quanto a busca são $O(log(n))$.

Para utilizar esse mapa, também foi necessário criar um estrutura que tem como função definir a hash para os `std::pair` que serviriam de chave para o Hashmap. Isso foi definido no struct `hash_pair`.

O algoritmo em sí é bastante simples. Caso as meninas não podem pagar o preço da ilha, simplesmente chame a própria função denovo sem mudar o dinheiro disponível mas retirando da lista de ilhas a ilha que elas não podem pagar (o que foi controlado com um parâmetro `int i` na função que definia qual era o índice da última ilha disponível). Caso elas pudessem pagar o preço da ilha, duas chamadas são feitas: uma que simplesmente exclui essa ilha e não muda o dinheiro disponível (chamada correspondente a ignorar a ilha e resolver o problema com o mesmo dinheiro e menos uma ilha) e uma chamada que diminui o dinheiro disponível e também exclui essa ilha da lista (chamada que corresponde a elas passarem um dia nessa ilha e resolver o problema com dinheiro disponível menor e uma ilha a menos). Ao final comparamos essas duas opções em relação à soma de felicidade para escolher qual caminho é o melhor para retornar.

4 Análise de Complexidade

4.1 Algoritmo Guloso

A complexidade de espaço do problema é bem simples de calcular visto que usamos apenas uma estruturas de armazenamento de dados: o `std::vector` responsável por guardar as informações de cada uma das ilhas, ou seja, é linear em função da quantidade de ilhas, ou seja $O(n)$.

Para a complexidade de tempo também temos um caso bastante simples pois existem apenas duas partes: a ordenação que toma $O(nlogn)$ e a passagem pelas ilhas, que é linear em função do número de ilhas, ou seja, $O(n)$. Como esses passos acontecem um após o outro, temos que a maior complexidade domina a complexidade de tempo da função, ou seja, o algoritmo é $O(nlogn)$.

4.2 Algoritmo Dinâmico

A complexidade de espaço para o algoritmo dinâmico é um pouco diferente visto que usamos duas estruturas de armazenamento de dados: o `std::vector` responsável por guardar as informações de cada uma das ilhas, que é linear em função da quantidade de ilhas, ou seja $O(n)$ e também utilizamos o `std::unordered_map` para guardar o resultado das chamadas, que depende tanto da quantidade de dinheiro disponível quanto do número de ilhas. Porém, como o mapa só terá valores novos em chamadas que não foram chamadas ainda, é seguro assumir que no pior caso, o número de valores armazenado pela mapa é igual ao número de chamadas da recursão, que é $O(nm)$ onde n é o número de ilhas e m é o dinheiro disponível pelas meninas, como será mostrado a seguir na complexidade de tempo.

Na complexidade de tempo, temos que uma chamada só precisará ser processada caso ela não foi ainda processada devido ao mapa de memeização. Como cada chamada depende de duas coisas: o número de ilhas ainda disponíveis e o dinheiro ainda disponível, no pior caso, o algoritmo terá que testar todas as combinações de dinheiro disponível com número de ilhas restantes. Como temos m de dinheiro e n ilhas, então esse número máximo será $O(nm)$.

5 Prova de Corretude

5.1 Algoritmo Guloso

O algoritmo guloso NÃO é ótimo para o problema apresentado. Um contra exemplo para a otimalidade dele é a entrada:

```
10 2
6 6
5 4
```

Esse teste falha pois a entrada `6 6` é a de melhor custo beneficio, então o algoritmo guloso irá dar preferência a ele, porém ele só poderá comprar um dia

de estadia nessa ilha, gerando uma "felicidade" de apenas 6. Caso ele escolhesse o de pior custo benefício só que duas vezes (que cabe no bolso das amigas) a felicidade seria de 8, gerando assim uma resposta melhor.

Não existe um meio de fazer um algoritmo guloso que seja ótimo para esse problema.

5.2 Algoritmo Dinâmico

Como o algoritmo dinâmico testa todas as possibilidades plausíveis (todas as combinações de quantidade de dinheiro e número de ilhas que é possível obter em alguma das ordens de ilhas a serem visitadas) e as compara de modo a sempre retornar a que gera maior pontuação com as amigas, é impossível que exista uma combinação de lugares a serem visitados melhor que o retornado pelo algoritmo.

6 Análise Experimental

Os testes feitos foram aumentando o número de ilhas (de maneira informada em cada uma das explicações de análise de cada paradigma) e a quantidade de dinheiro disponível pelas meninas sendo sempre $\#ilhas \times 10$. O preço da estadia de cada ilha foi criado por um número aleatório entre 10 e o número de ilhas.

6.1 Algoritmo Guloso

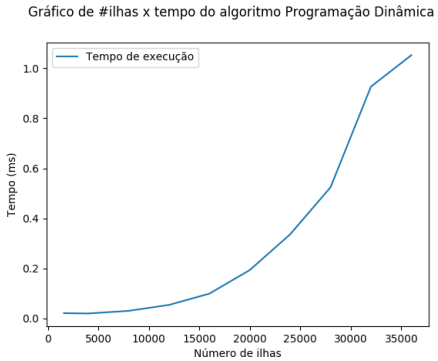
Para o algoritmo guloso, era esperado que o gráfico de número de ilhas por tempo fosse em um formato parecido com o da curva de $f(n) = nlogn$ visto que essa é sua complexidade de tempo. O que foi obtido foi o seguinte gráfico.

# ilhas	Média	Desvio Padrão
10	0.01974	0.00346
50000	0.02368	0.00248
100000	0.03185	0.00517
150000	0.04231	0.01720
200000	0.04976	0.01169
250000	0.05897	0.01608
300000	0.06972	0.01853
350000	0.07993	0.04652
400000	0.08672	0.02412
450000	0.09648	0.03884
500000	0.10483	0.03517

6.2 Algoritmo Dinâmico

Já para o algoritmo dinamico, era esperado que o gráfico de número de ilhas por tempo fosse um gráfico de uma curva quadrática pois como sua complexidade de tempo é $O(nm)$ e o número de ilhas está aumentando linearmente e o dinheiro disponível pelas amigas cresce linearmente junto com o número de ilhas, quando multiplicarmos um pelo outro vai resultar em uma curva quadrática em função do número de ilhas.

# ilhas	Média	Desvio Padrão
3500	0.02013	0.00396
7000	0.01977	0.00235
10500	0.02573	0.00436
14000	0.04821	0.01225
17500	0.09152	0.02865
21000	0.18189	0.08406
24500	0.29238	0.13678
28000	0.41967	0.12278
31500	0.80147	0.20114
35000	1.09412	0.35354



7 Conclusão

O problema foi modelado e implementado com sucesso e os testes representaram o que era esperado quando analisamos o algoritmo de forma teórica

O trabalho compriu o propósito de nos fazer praticar o uso de paradigmas da programação para resolver um problema, tanto na parte de modelagem quanto na implementação, que é algo bem diferente do que estamos acostumados a lidar no curso até então no curso.

Acredito que as amigas tem uma preocupação a menos tentando planejar sua viagem.