UNIVERSIDADE FEDERAL DE MINAS GERAIS

Departamento de Ciências da Computação

Algoritmos 1

Professora: Olga Nikolaevna

Aluno: Pedro Henrique Solano de Oliveira

1. Introdução

O problema proposto consiste em maximizar a satisfação dos viajantes conforme o valor que eles possuem disponível para gastar na viagem. Foram implementadas duas soluções diferentes: a primeira, utilizando um algoritmo guloso, no qual os viajantes podem ficar mais de um dia na mesma ilha, e; a segunda, utilizando um algoritmo de programação dinâmica, no qual os viajantes podem ficar apenas um dia em cada ilha visitada.

O algoritmo guloso escolhe a melhor opção disponível em cada iteração. O critério para otimizar a escolha das ilhas foi a proporção entre os pontos e o custo que cada ilha apresenta, ou seja, o “custo-benefício” de cada ilha.

Já o algoritmo dinâmico funciona desmembrando o problema principal em subproblemas, encontrando as soluções ótimas para esses subproblemas e armazenando-as em uma tabela, de forma que a solução de cada problema dependa apenas de subproblemas previamente calculados.

A implementação dos algoritmos é detalhada a seguir.

2. Implementação

2.1 Arquivos

Para resolver o problema, foi usada a linguagem C++, através dos arquivos funcoes.h, onde há o cabeçalho, definições e protótipos da biblioteca criada, “funcoes.cpp”, onde há a implementação das funções prototipadas em “funcoes.h”, e “main.cpp”, onde o programa principal é implementado.

No arquivo “funcoes.h”, para começar, foi utilizada a diretiva #ifndef para garantir que as definições do arquivo sejam realizadas apenas uma vez durante a execução, mesmo que o arquivo seja adicionado duas vezes ou mais.

2.2 Execução

O arquivo “main.cpp” contêm as variáveis responsáveis por registrar os dados referentes à viagem: “int orcamento” representa o valor máximo a ser gasto, “int qtde_ilhas” representa o número máximo de ilhas, “int* custos” é um vetor que armazena os custos de cada ilha e “int* pontos” é o vetor que armazena os pontos atribuídos a cada ilha. Por fim, a variável “char* arquivo” registra o nome do arquivo a ser lido que contêm os dados da viagem, informado via linha de comando.

O programa inicia com a declaração dessas variáveis, seguindo com a leitura do vetor de parâmetros por linha de comando (argv) e registrando o nome do arquivo de entrada. Em seguida, a função “le_viagem” faz a leitura da primeira linha do arquivo informado e preenche “orcamento” e “qtde_ilhas”. “qtde_ilhas” é utilizada para fazer a alocação de memória dos vetores “custos” e “pontos”, que serão preenchidos na sequência com a chamada da função “preenche_ilhas”.

Com os dados da viagem conhecidos e devidamente armazenados em suas respectivas variáveis, é chamada a função “guloso” que possui as seguintes variáveis: “float* custoXbeneficio”, que é o vetor que armazena o resultado da divisão entre os pontos e o custo de cada ilha, “int n_dias” e “int pontuacao”, sendo esses 2 últimos iniciados com o valor zero. Após a declaração das variáveis, “guloso” chama a função “preenche_cXb”, que preencherá o vetor “custoXbeneficio” de forma crescente, ou seja, a primeira posição será ocupada pelo menor resultado da divisão entre pontos e custo, representando a ilha com menor custo por ponto.

Em seguida, é chamada a função “mergeSort”, que irá ordenar o vetor custoXbeneficio e reposicionar os elementos dos vetores “custos” e “pontos”, de formar a manter a correspondência entre as posições de todos os vetores. Ao término da ordenação, as informações de custo, pontos e custoXbeneficio de uma dada ilha x estará armazenada na posição [x] desses vetores.

Inicia-se, então, a execução do loop guloso:

```
enquanto o saldo for positivo e não exceder o número máximo de ilhas
{
 se (saldo - custo da ilha atual) não negativo:
  {
   subtrai o custo da ilha atual do saldo;
   soma os pontos da ilha atual à pontuação total;
   incrementa em 1 o número de dias;
  }senão
  {
   avança para a próxima ilha;
  }
}
```

Figura 1: pseudo-código do algoritmo guloso.

Conforme mostrado na figura 1, o loop inicia a iteração verificando os dados da primeira ilha, que, necessariamente, será aquela com melhor custo-benefício, devido à ordenação realizada previamente. Caso o saldo seja suficiente para ficar um dia nessa ilha, o algoritmo somará os pontos dessa ilha à variável "pontuacao", debitará o custo dessa ilha da variável "saldo" e incrementará em 1 a variável "n_dias". Caso o saldo não seja suficiente, o contador "i" será incrementado, fazendo com que sejam acessados os dados da próxima ilha. As iterações serão executadas até que o saldo remanescente seja insuficiente para qualquer ilha adicional.

Concluído o loop, serão impressos na tela a pontuação total obtida, armazenada em "pontuacao", e o número de dias da viagem, armazenado em "n_dias".

Retornando ao "main", após a execução da função "guloso" é chamada a função "dinamico", que executará o algoritmo de programação dinâmica.

A solução com programação dinâmica foi baseada no problema da mochila visto em sala de aula, e implementada da seguinte forma:

Para qualquer ilha podemos fazer a seguinte afirmação:

1 - a ilha está no subconjunto ótimo, ou;

2 – a ilha não está no subconjunto ótimo.

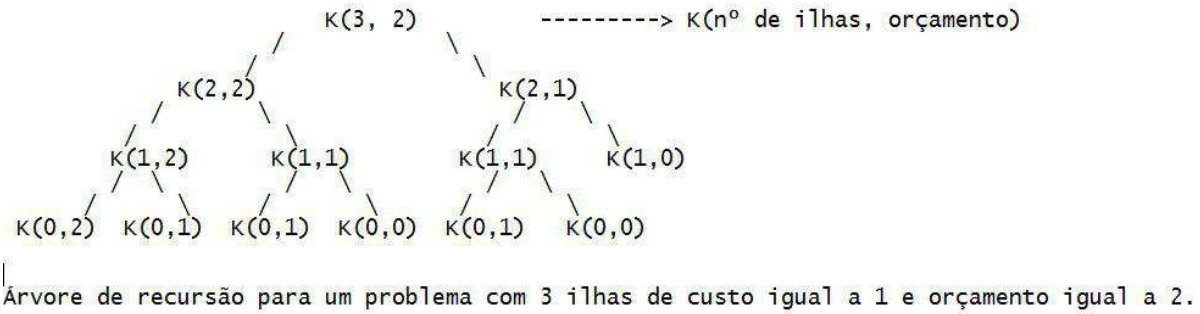
Portanto, a pontuação máxima que pode ser obtida de "*i*" ilhas será determinada pelo maior dos dois valores a seguir:

1- a pontuação máxima obtida por *i*-1 ilhas com orçamento *O* (excluindo a ilha *i*);

2 - a pontuação da *i*-ésima ilha mais a pontuação obtida com n-1 ilhas e o orçamento restante (orçamento menos o custo da ilha *i*).

Se o custo da i-ésima ilha é maior que o orçamento disponível, o caso 1 será a única possibilidade.

Ao simular uma árvore de recursão para esse problema, percebemos que existe o cálculo repetido de subproblemas já resolvidos, conforme ilustrado na figura a seguir:



Árvore de recursão para um problema com 3 ilhas de custo igual a 1 e orçamento igual a 2.

Figura 2: Árvore de recursão

Para solucionar essa ineficiência, armazenamos os resultados dos subproblemas já computados em uma matriz, de tamanho igual a [número de ilhas+1] vezes [orcamento +1]. Essa matriz é declarada na variável *k*, logo no início da função “dinâmico”. A seguir, a matriz é preenchida com os valores de pontuação das ilhas através de dois laços de repetição, que executam a lógica descrita acima.

Concluído o preenchimento da matriz *k*, imprime-se na tela o valor contido na última posição da matriz, que representa a maior pontuação possível. Em seguida, inicia-se um loop para verificar quais ilhas foram selecionadas; esse loop parte da última posição da tabela, fazendo um processo inverso ao da construção da matriz. Para cada ilha selecionada, incrementa-se a variável contadora de dias “n_dias”, que ao final do loop é impressa na tela, concluindo a função “dinâmico”.

O “main” termina após a execução da função “dinâmico”.

3. Análise de complexidade

3.1 Algoritmo guloso

3.1.1 Complexidade de tempo

O algoritmo guloso consiste em três etapas sequenciais:

1 - Preenchimento do vetor “custoXbeneficio”, que ocorre em tempo linear com o número *m* de ilhas, uma vez que armazena o resultado da divisão entre os elementos do vetor “custo” pelos elementos de “beneficio”, ambos com *m* posições.

2 - Ordenação merge sort do vetor “custoXbeneficio”. Essa ordenação ocorre a partir de duas chamadas recursivas para dividir o vetor inicial, seguidas do procedimento para unir os subvetores.

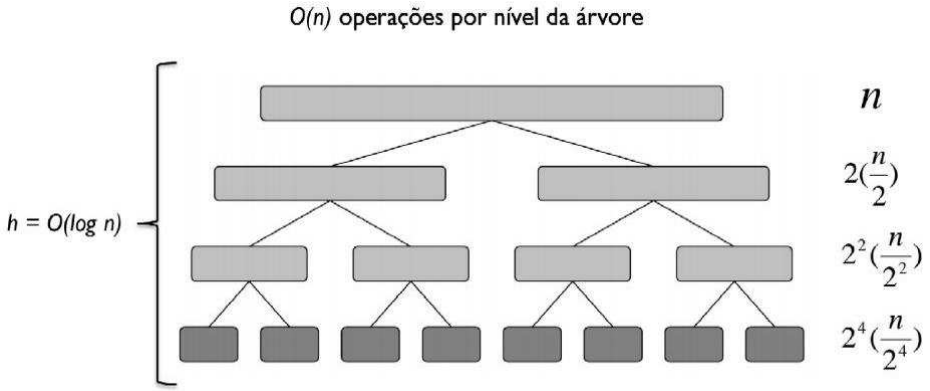


Figura 3: Divisão do vetor principal em subvetores

Cada chamada recursiva tem custo *m*/2 e a operação de merge tem custo linear. Assim, a equação de recorrência pode ser escrita da seguinte forma: T(m) = 2T(m/2) + m. Aplicando o caso 2 do Teorema Mestre, temos:

$$a = 2,\ b = 2,\ f(n) = m \text{ e } m^{\log_b a} = m^{\log_2 2} = m$$

que resolve a equação para T(m) = *O*(m log m).

3 - Loop while para seleção das ilhas. Esse loop ocorre em tempo linear com o número *m* de ilhas, que será percorrido linearmente até, no máximo, sua última posição.

Assim, a complexidade de tempo do algoritmo guloso é dada por *O*(m) + *O*(m log m) + *O*(m), o que é equivalente a *O*(m log m).

3.1.2 Complexidade de espaço

O algoritmo guloso manipula 3 vetores (custos, pontos e custoXbeneficio) de tamanho *m*. Para cada um desses vetores são criados 2 subvetores auxiliares, cujos tamanhos são a metade do vetor original. Assim, a complexidade de espaço é igual a *O* (3n), que é equivalente a *O*(n).

3.2 Algoritmo dinâmico

3.2.1 Complexidade de tempo

Para obter a pontuação máxima evitando a repetição de cálculos já realizados, o algoritmo dinâmico constrói uma matriz de tamanho (nº de ilhas + 1) x (orçamento disponível para viagem + 1), que é preenchida através de dois loops *for*.

Orçamento: 10 Quantidade de ilhas: 5

ILHA	PONTOS	Custo
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Orçamento (o+1 posições)

Nº DE ILHAS (i+1 posições)	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29
5	0	1	6	7	7	18	22	28	29	34	35

Figura 4: exemplo da matriz criada para um arquivo com 5 ilhas e orçamento = 10

A cada iteração de cada loop *for* é realizada uma atribuição de valor e, no máximo, 2 comparações. Assim, a complexidade de espaço pode ser determinada como *O* (o * i), sendo “o” o valor disponível e “*i*” o número de ilhas.

3.2.1 Complexidade de espaço

Conforme ilustrado na figura X, o algoritmo dinâmico utiliza um espaço *O* (o * i), para armazenar a matriz calculada.

4. Resultados experimentais:

Foram testados sete arquivos de entrada diferentes, com o número de ilhas variando entre 4 e 50 e o valor disponível variando entre 2250 e 40000. Cada arquivo foi executado 10 vezes. Os resultados dos tempos de execução, média e desvio padrão estão representados nos gráficos e na tabela a seguir:

Nº Ilhas	4	10	15	20	25	30	50
Valor Total	2250	2000	5000	10000	10000	20000	40000

Tempo de execução (ms): Guloso

Nº de execuções							
1	0,019	0,024	0,020	0,031	0,021	0,024	0,040
2	0,016	0,026	0,016	0,055	0,021	0,033	0,027
3	0,026	0,022	0,040	0,021	0,078	0,022	0,030
4	0,018	0,022	0,023	0,020	0,030	0,023	0,026
5	0,017	0,022	0,022	0,047	0,025	0,048	0,031
6	0,024	0,018	0,020	0,024	0,024	0,020	0,032
7	0,021	0,018	0,020	0,024	0,023	0,023	0,104
8	0,023	0,018	0,021	0,025	0,026	0,026	0,054
9	0,025	0,015	0,018	0,033	0,022	0,022	0,027
10	0,018	0,017	0,020	0,027	0,021	0,026	0,027

Média:	0,021	0,020	0,022	0,031	0,029	0,027	0,040
Desvio padrão:	0,003	0,003	0,006	0,011	0,017	0,008	0,023

Tempo de execução (ms): Dinâmico

Nº de execuções							
1	0,069	0,102	0,493	1,656	1,775	4,587	14,998
2	0,068	0,100	0,499	1,432	1,812	4,746	14,740
3	0,070	0,103	0,571	1,472	1,771	4,462	15,018
4	0,068	0,101	0,499	1,425	1,967	4,421	14,811
5	0,098	0,102	0,538	1,435	1,765	4,523	14,833
6	0,072	0,102	0,565	1,435	1,760	4,430	15,466
7	0,071	0,157	0,494	1,432	1,760	4,388	14,896
8	0,071	0,132	0,521	1,468	1,762	4,420	14,983
9	0,072	0,139	0,493	1,417	1,782	4,497	15,326
10	0,106	0,101	0,526	1,573	1,762	4,454	14,787

Média:	0,077	0,114	0,520	1,475	1,792	4,493	14,986
Desvio padrão:	0,013	0,020	0,028	0,074	0,060	0,101	0,226

Tabela 1:tempos de execução

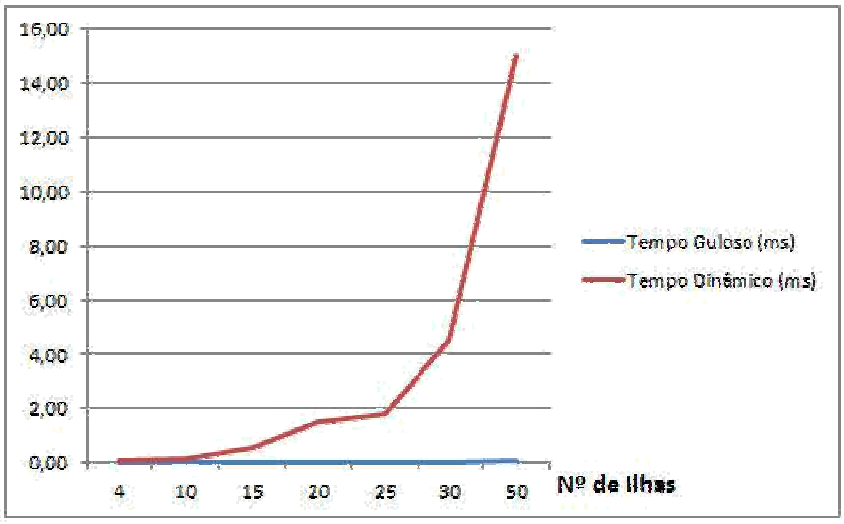


Gráfico 1:tempo de execução (ms) X Nº de ilhas

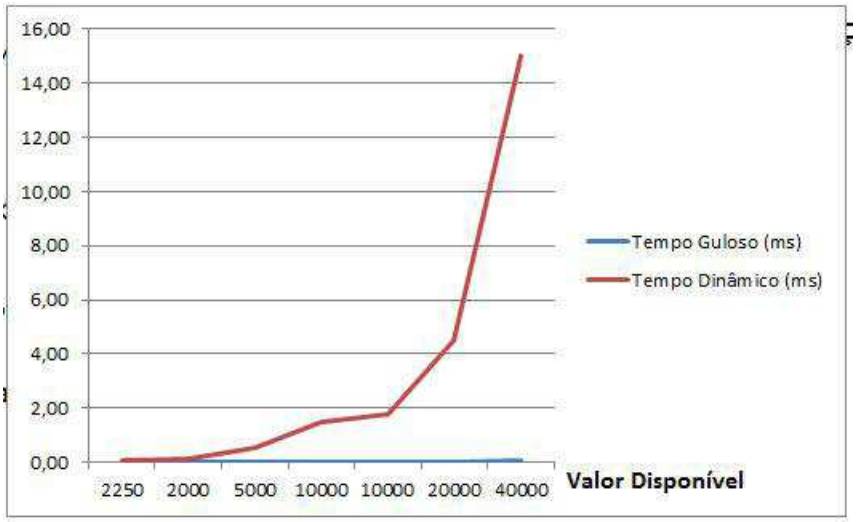


Gráfico 2: tempo de execução (ms) X Valor disponível

O algoritmo guloso apresentou um pequeno aumento no tempo de execução (de 0,021ms para 0,040ms) em função do aumento do tamanho da entrada, enquanto o algoritmo dinâmico sofreu uma variação quase 100x maior (de 0,077ms para 14,986ms). É possível observar pelos gráfico que o tempo de execução do dinâmico é sensível tanto ao aumento do número de ilhas quanto ao aumento do valor disponível, enquanto o guloso é sensível apenas ao número de ilhas.

Podemos concluir que os resultados encontrados atendem às expectativas, uma vez que o dinâmico precisa de um tempo *O* (n * m) para preencher sua matriz, enquanto o guloso gasta *O* (m log m) para realizar sua ordenação.

5. Conclusão

O problema proposto no enunciado foi resolvido como sugerido, isso é, na linguagem C++, de forma modularizada e bem documentada, tanto nesse arquivo quanto em forma de comentários ao longo dos códigos fonte e respeitando os tempos de execução requeridos.

Podemos observar que, para os arquivos fornecidos no Dataset, bem como para diversas outras entradas testadas, o algoritmo guloso retorna valores de pontuação e de dias de viagem maiores que aqueles obtidos pelo algoritmo dinâmico. Isso se deve ao fato de que o algoritmo guloso pode repetir uma mesma ilha, enquanto o dinâmico não. Por outro lado, por esse mesmo motivo o dinâmico garante a maior variabilidade possível de ilhas, enquanto o guloso poderia passar toda a viagem em uma mesma ilha, dependendo dos parâmetros de entrada.

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). I*ntroduction to algorithms.*MIT press.

Ford,W.,William, F., and Topp,W. (1995). *Data structures with C++*. Simon & Schuster, Inc.

https://www.geeksforgeeks.org/. Consultado em 25 de outubro de 2019.