Análise de Algoritmos e Técnicas: Ciclos em Grafos, *Backtracking* e *Branch and Bound*

Henrique Augusto Rodrigues[1]

[1]Instituto de Ciências Exatas e Informática - Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil

Resumo: *Divide and Conquer* é um paradigma de design de algoritmo, quebrando um problema maior em dois, ou, mais subproblemas, preferencialmente de tamanhos iguais, tornando-os simples de serem resolvidos diretamente, entrando na parte da conquista onde, concatena todos o resultados obtidos em um resultado do problema original.*Branch an bound* sendo um algoritmo para encontrar uma solução ótima em diversos problemas de otimização, em um enumeração sistemáticade todos os possíveis candidatos ao resultado. *Backtracking* sendo um algoritmo refinado da força bruta (ou enumeração exaustiva), sendo boa parte das soluções elminadas sem serem explicitamente eliminadas.

Abstract: *Divide and Conquer it is a paradigm of algorithm design, breaking a bigger problem in two, or, more subproblems, preferably of equal sizes, making them simple to be solved directly, entering in the part of the conquest where, it concatenates all the obtained results in a result of the original problem.*

1. Visão Geral

Na disciplina de Projeto e Análise de Algoritmos, entender diferentes resoluções para um mesmo problema e diferentes técnicas para dadas soluções é de suma importância. Neste trabalho será feito uma análise do problema dos pares de pontos mais próximos, utilizando a abordage divisão e conquista, problema da mochila atravé do algoritmo branch and bound e o problema turtle através do backtracking. Para checar o código, vá a branch 'Closest Pair of Points' *Github*.

2. Arquitetura da máquina

Chip: Apple M1 Memory: 8 GB macOS: Ventura 13.01

3. *BackTracking*

Backtracking é uma estratégia para resolver problemas recursivamente e incrementalmente, removendo as soluções parciais que falham em ajudar na solução do problema. Para aplicar essa técnica, a estratégia tenta encontrar a solução utilizando diversos pequenos checkpoints, para os quais o programa pode voltar se a iteração atual da resolução do problema não ajudar a encontrar a solução final. Essa estratégia é viável para resolver problemas modulares que requerem tentativas e erros, muitas, já que, ele remove "caminhos" não válidos, salvando muito tempo de processamento.

3.1. Problema do Labirinto

Simplificando o problema, podemos dividir um quadrado em pequenos outros, com obstáculos no meio no meio do caminho. Por exemplo, um robô que aspira o

chão de sua casa, ele mapeia o ambiente encontrando o melhor caminho e desviando de obstáculos, ou, até mesmo um robô onde, está em um labirinto e tenta encontrar o melhor caminho de saída.

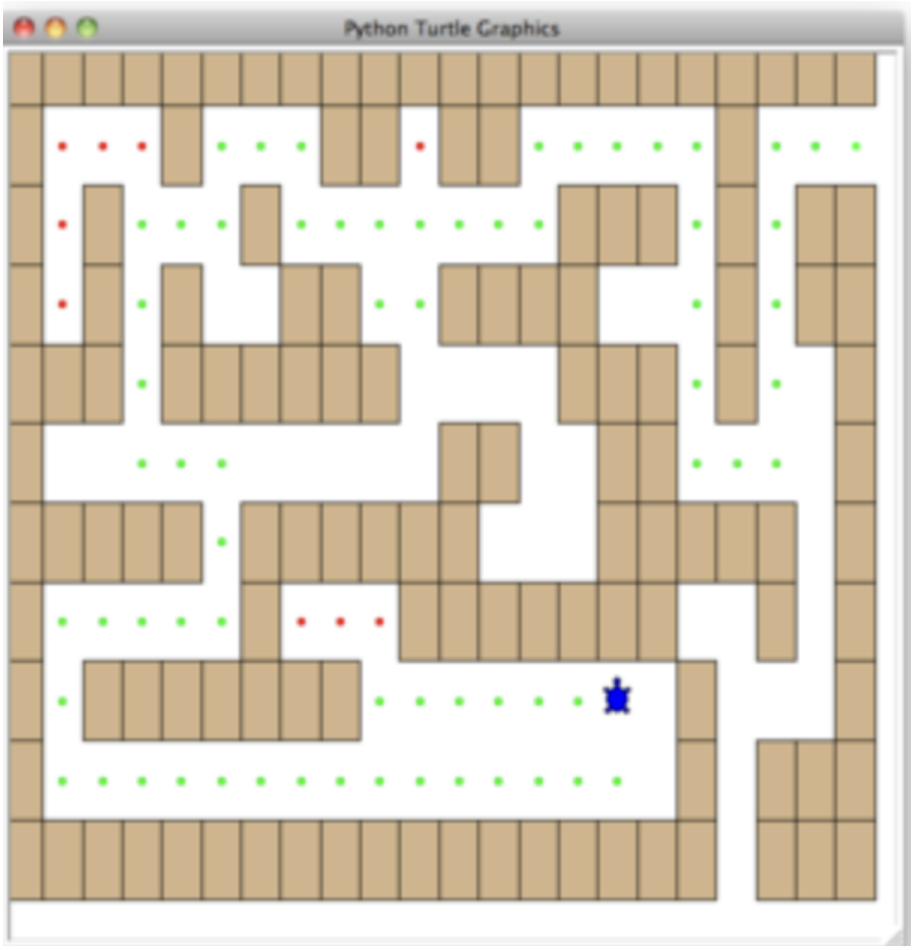

Figure 1. Representação do problema do labirinto

4. *Branch and Bound*

Branch and Bound é uma técnica para algoritmos que, tem como objetivo melhorar o tempo de processamento eliminando as soluções-candidato que, não se aplicam e/ou não ajudam a resolver o problema. Esse método geralmente é aplicado em problemas que têm soluções finitas, no qual as soluções podem ser representadas como uma sequência de opções. A primeira parte da técnica *Branch and Bound* pede que o algoritmo trate as possíveis soluções como se estivessem em uma estrutura de árvore, onde os nós da árvore são utilizados como partes de possíveis soluções garantindo que, todas soluções sejam consideradas.

4.1. Problema da Mochila

O problema escolhido para mostrar essa estratégia é o problema do 0/1 *Knapsack*. A ideia é que, após viajar por alguns dias, você decide comprar presentes na volta, mas a mochila aguentará somente alguns itens a mais, de forma que o peso total não ultrapasse o peso limite suportado pela mochila. Como escolher o que levar? Considerando valor e peso.

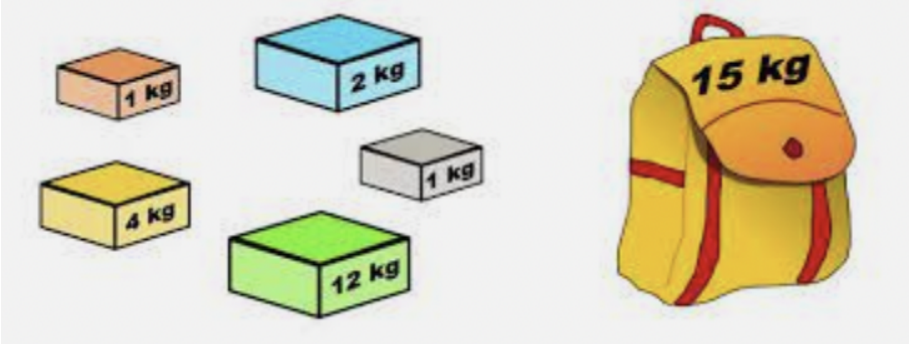

Figure 2. Representação do problema da mochila

4.2. Divisão e Conquista

Divisão e Conquista é uma técnica que pode ser dividida em três partes fundamentais: dividir um problema maior recursivamente em problemas menores(Dividir), resolver todos os sub-problemas (Conquistar) e, por fim, a solução do problema inicial é dada através da combinação dos resultados de todos os problemas menores computados(Combinar). Problemas que usam divisão e conquista são característiscos em possuir um princípio chamado Overlapping subproblems, ou seja, o problema pode ser divido em subproblemas que são reutilizados várias vezes não gerando novos subproblemas. Exemplos de problemas conhecidos que usam desta estratégia são o problema de ordenação interna(usando quicksort ou mergesort) e o problema dos pares de pontos próximos, esse último abordado no trabalho que, se encontra na subsecão a seguir.

4.3. Pares de Pontos mais Próximos

Dado um conjunto de N pontos em um espaço, é preciso encontrar os dois pontos do conjunto que possuem a menor distância um do outro. Este problema possui mais de uma implementação, porém uma das mais eficientes é utilizando divisão e conquista. Que consiste em, dividir o plano ao meio [n/2] ¡ i ¡ [n/2], ao se aproximar de i (meio do plano) analisaremos a distância entre os pontos mais próximos da esquerda para a direita e, seu correlato até o problema ser resolvido.

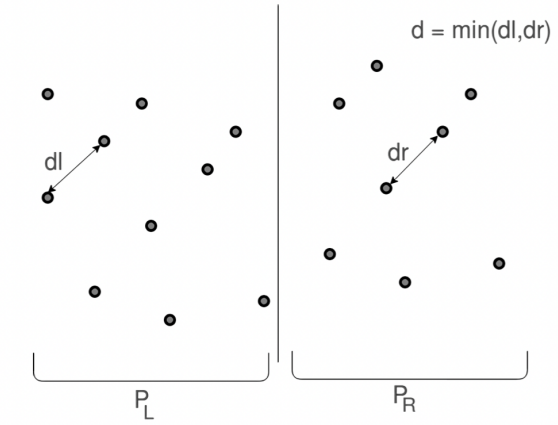

Figure 3. Representação menor distância entre dois pontos

A entrada para o problema é um conjunto de pontos com um dos eixos já ordenado, encontramos o ponto do meio, separamos o conjunto ao meio, e então recursivamente descobrimos as menores distâncias entre os conjuntos separados, para então achar a solução geral que queremos.

5. Resultados

Utilizando 6 pares de pontos no plano cartesiano (2D), e fazendo o comparativo com as instâncias dadas no enunciado do problema.

5.1. Programa Original

0.00s user 0.00s system 0/100 cpu 0.590 total

5.2. Programa com 1000 instâncias

0.00s user 0.00s system 58/100 cpu 0.008 total

5.3. Programa com 100000 instâncias

0.00s user 0.00s system 51/100 cpu 0.002 total

5.4. Programa com 1000000 instâncias

0.00s user 0.00s system 56/100 cpu 0.003 total

5.5. Branch and Bound

Por um conflito entre o compilador GCC e o chip M1 cuja arquitetura é ARM 64, sendo assim não foi possível obter o tempo de execução.

5.6. BackTracking

0,00s user 0,00s system 78/100 cpu 0,010 total

6. Análise Comparativa entre as Abordagens

Nesta seção, mostrar-me-ei um comparativo entre as abordagens, divide and conquer, branch and bound e o backtracking.

• Devido a natureza da estratégia de *Backtracking*, tais algoritmos costumam ter implementações recursivas, enquanto algoritmos gulosos tentam buscar a melhor escolha local de forma iterativa.

• *Branch and Bound*, assim como *Backtracking*, consegue voltar a estados anteriores da árvore para encontrar a solução, enquanto isso, algoritmos gulosos tentam buscar a melhor escolha local de forma iterativa e não dividem o problema em ”*checkpoints*”.

6.1. *Backtracking* Vs *Branch and Bound*

• Ambos algoritmos usam uma árvore de estados(*state space tree*) para encontrar a solução, porém, enquanto *Backtracking* procura por ela exaustivamente até encontrar a solução, *Branch and Bound* é forçado a passar pela árvore inteira para encontrar a solução otimizada.

• A pesquisa feita na árvore de estados do *Backtracking* utiliza um *DFS(Depth first search)*, enquanto que o *Branch and Bound* pode utilizar *DFS*, ou até mesmo *BFS(Breadth first search)*.

• Como *Backtracking* não precisa percorrer por todas as possíveis soluções, esta abordagem é mais eficiente do que *Branch and Bound*, que precisa percorrer todas as ramificações da árvore de estados.

• O resultado do algoritmo de *Backtracking* pode ser a solução optimizada, porém, existem problemas na implementação de *backtracking* em que, o resultado será uma solução sub-optimizada. Como *Branch and Bound* varre e procura todas as soluções, este consegue achar a solução otimizada com maior frequência.

6.2. Divisão e Conquista Vs *Branch and Bound*

• Tanto *Branch and Bound* e *Divide and Conquer* são estratégias que partem dos mesmos princípios: ambas querem dividir o problema em partes menores para resolver o problema todo.

6.3. Divisão e Conquista Vs *Backtracking*

• Devido à maneira como ambos são conceitualizados, ambos possuem a natureza de serem recursivos.

• Em Divisão e Conquista, precisamos analisar a entrada inteira do usuário para resolver o problema, enquanto que *Backtracking* pode ou não analisar a todas as possibilidades dentro da entrada do usuário.

7. Conclusão

O paradigma da divisão e conquista, resolve inúmeros problemas maiores em um tempo da execução satisfatório contendo poucas instâncias, a partir do momento em que esse número sobe, começa a exigir mais do sistema e tais resultados começam a variar de tal forma que, é preciso uma análise mais aprofundada para afirmar se, o aumento do número de instâncias são computadas em sua totalidade, ou, chega em um ponto que causa uma falha no sistema e não segue adiante na procura dos pares de pontos mais próximos. Cada algoritmo, exceto o backtracking, atendeu os resultados esperado pelo altor.

8. Contribuições

O autor (Henrique Augusto Rodrigues), construiu, compilou, executou e analisou os resultados do programa e também, construiu o texto.