Trabalho 3 - Branch and Bound

Letícia Americano Lucas - 691290

Problema:

O Problema de Sequenciamento de Trabalhos com Prazo e Penalidade é um desafio que se concentra em organizar uma lista de tarefas ou trabalhos, cada uma com um prazo limite para ser concluída e uma penalidade associada ao seu atraso.

Em termos mais específicos, cada trabalho tem um tempo estimado para ser realizado e uma data limite para sua conclusão. Se o trabalho for finalizado após o prazo estabelecido, uma penalidade é aplicada. O objetivo principal é encontrar a ordem mais eficiente para realizar esses trabalhos, minimizando o acúmulo total das penalidades decorrentes de atrasos.

Essa questão se torna um desafio de otimização combinatória, pois a determinação da melhor sequência de execução exige considerar todas as combinações possíveis para os trabalhos. O objetivo final é encontrar uma ordem de execução.

Algoritmo de Branch and Bound:

Pensando na forma de implementação de um algoritmo que resolva tal problema, temos o algoritmo de Branch and Bound, que utiliza da estratégia de busca usada para encontrar a melhor solução em problemas complexos. Ele funciona dividindo o problema em partes menores, explorando-as e descartando soluções que não vão levar a resultados melhores do que os já encontrados. É como se fosse um explorador que segue por caminhos diferentes, mas inteligentemente desiste de alguns caminhos quando percebe que não levarão ao destino desejado. Isso ajuda a economizar tempo e esforço, encontrando a resposta certa de forma mais rápida e eficiente. Essa técnica é útil em muitos tipos de problemas, como encontrar a melhor rota em viagens, organizar tarefas para evitar atrasos ou resolver quebra-cabeças, simplificando a busca por soluções ótimas.

Resolução:

```python
best_sequence = None
best_penalty = None

def calculate_penality(jobs):
    current_time = 0
    total_penalty = 0

    for job in jobs:
        current_time += job.time

        if current_time > job.deadline:
            penalty = current_time - job.deadline
            total_penalty += penalty

    return total_penalty


def branch_and_bound(jobs, current_jobs):
    global best_sequence
    global best_penalty

    if len(jobs) == 0:
        best_sequence = current_jobs
        best_penalty = calculate_penality(current_jobs)
        return

    for job in jobs:
        jobs_without_current = [j for j in jobs if j != job]
        new_current_jobs = current_jobs + [job]

        if best_penalty != None and best_penalty <= calculate_penality(new_current_jobs):
            return

        branch_and_bound(jobs_without_current, new_current_jobs)

    return best_sequence, best_penalty

```

Inicialmente, as variáveis best_sequence e best_penalty são usadas para armazenar a melhor sequência de trabalhos e sua penalidade total associada de forma global. Logo abaixo, a função calculate_penalty avalia essa penalidade, considerando o tempo total e aplicando penalidades por atraso nos trabalhos.

Na função principal, branch_and_bound, implementada de forma recursiva, a verificação inicial é se não há mais trabalhos na lista. Se isso ocorrer, calcula-se a penalidade da sequência atual e, caso seja a melhor solução até então, atualizam-se best_sequence e best_penalty.

Em seguida, explora-se os trabalhos restantes, criando novas listas sem o trabalho atual e formando novas sequências ao adicioná-los. Durante essa busca, é feita uma comparação entre a penalidade da melhor sequência encontrada até então e a penalidade da nova sequência. Se a penalidade já encontrada for menor ou igual à da nova sequência, a exploração nesse ramo é interrompida.

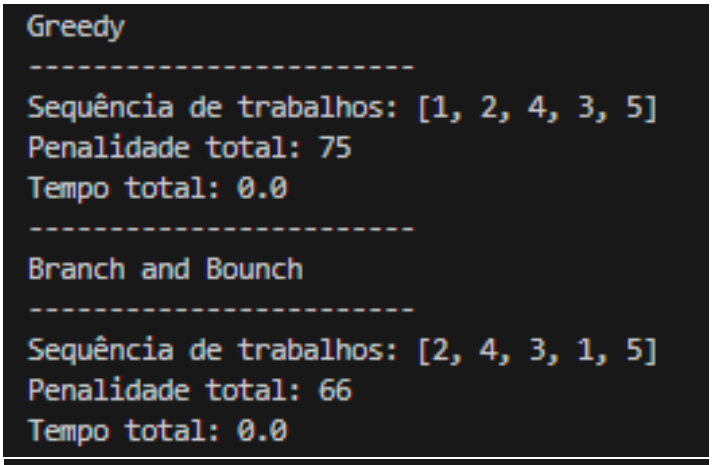
Ao concluir a exploração de todas as ramificações possíveis, a função retorna a melhor sequência best_sequence e sua correspondente penalidade best_penalty, atualizando as variáveis globais. Essa abordagem visa encontrar a solução mais eficiente para o problema de sequenciamento de trabalhos com prazo e penalidade.

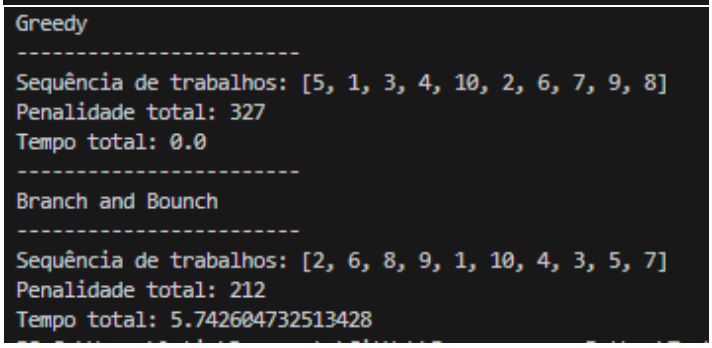
Análise de Dados:

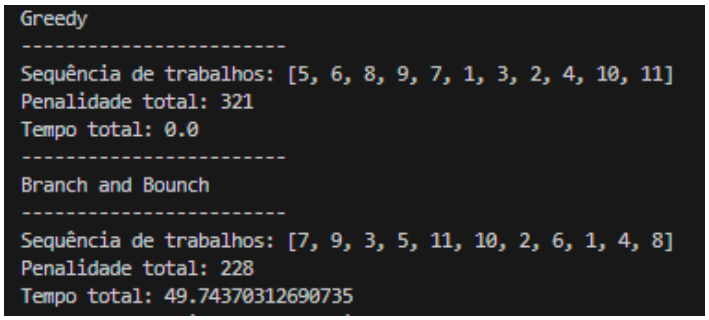
Na análise de dados utilizando o algoritmo Branch and Bound, foi adotada uma abordagem complementar ao integrar a função greedy. Essa função foi estrategicamente aplicada para oferecer uma visão inicial mais simplificada do conjunto de dados, em vista disso, a função greedy ordena os arrays de forma em que o array de maior penalidade esteja no início da lista e processa todos os dados de acordo com essa lista encontrando um ideal de penalidade mais baixa possível. A utilização do algoritmo greedy enfrenta críticas devido à sua abordagem de tomada de decisão imediata. Ao tomar as melhores escolhas localmente em cada passo, o algoritmo pode não garantir a solução ótima globalmente. Essa tendência em selecionar a opção mais vantajosa no momento pode resultar em soluções subótimas, já que o algoritmo não considera o panorama completo. Além disso, a eficácia do algoritmo é altamente influenciada pelos critérios de seleção, e uma escolha inadequada pode levar a resultados insatisfatórios. Outro aspecto relevante é a falta de capacidade do algoritmo em revisar decisões anteriores, já que não há mecanismos para retroceder e reconsiderar uma escolha feita. Portanto, apesar da sua simplicidade e rapidez, o algoritmo greedy pode apresentar limitações na obtenção de soluções ótimas em problemas complexos.

```python
import time

def greedy(jobs):
    start_time = time.time()
    jobs.sort(key=lambda x: x.penalty, reverse=True)
    current_time = 0
    total_penalty = 0
    sequence = []

    for job in jobs:
        current_time += job.time

        if current_time > job.deadline:
            penalty = current_time - job.deadline
            total_penalty += penalty
        sequence.append(job.id)

    end_time = time.time()
    execution_time = end_time - start_time

    return sequence, total_penalty, execution_time
```

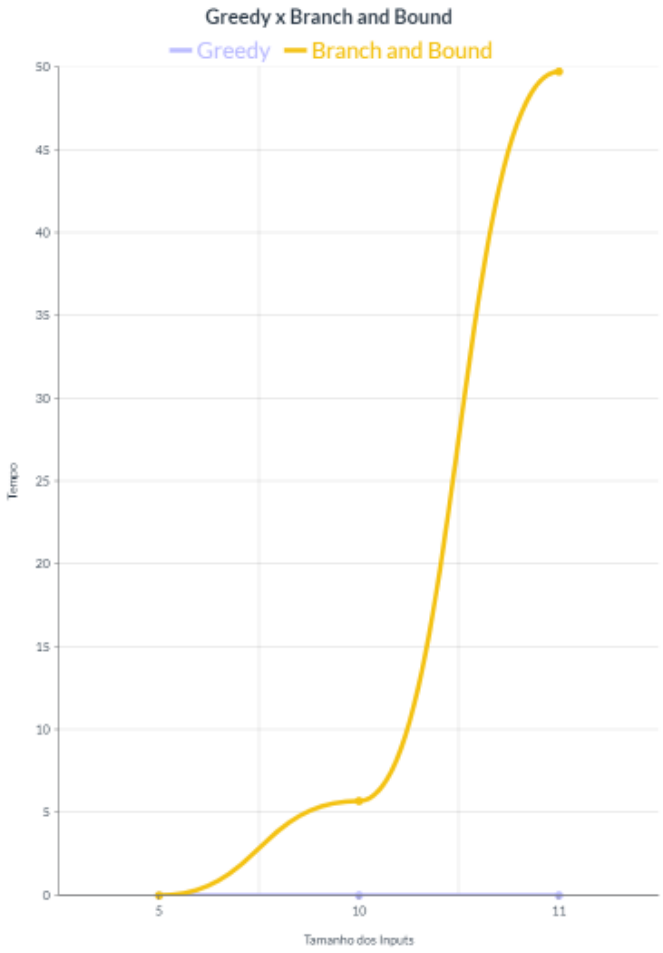
Dessa forma, ao gerarmos em sequência 5, 10 e 11 inputs o tempo acaba escalonando muito depressa sendo muito dificil trabalhar com uma coleção de inputs maiores.







Ao empregar o algoritmo greedy e o branch and bound, é evidente que ambos têm suas vantagens e limitações na resolução de problemas. O algoritmo greedy, embora rápido e simples, tende a oferecer soluções subótimas devido à sua abordagem imediata de seleção das melhores opções locais, sem considerar o quadro completo. Por outro lado, o branch and bound, apesar de alcançar resultados mais precisos e superar o greedy em termos de otimalidade, enfrenta desafios de escalabilidade. Conforme observado nas métricas de tempo, o branch and bound apresenta uma escalada exponencial no tempo de execução, enquanto o greedy permanece estável.



Conclusão:

Dessa forma, diante das particularidades e demandas variadas de cada problema, a escolha entre o algoritmo greedy e o branch and bound revela-se uma decisão estratégica. O branch and bound destaca-se por sua precisão em problemas menores, mesmo que isso implique em um aumento significativo no tempo de execução. Em contrapartida, o algoritmo greedy se mostra uma opção viável para problemas de maior escala, onde a agilidade na obtenção de uma solução aceitável é mais premente do que a busca pela otimalidade absoluta. Portanto, a seleção do algoritmo mais apropriado torna-se uma questão de ponderar as demandas específicas do problema em relação às prioridades estabelecidas, considerando tanto a qualidade da solução quanto a eficiência computacional, a fim de alcançar resultados satisfatórios de acordo com os objetivos traçados.