UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Daniele Cassia Silva Diniz
2020076874

TRABALHO PRÁTICO 3

Algoritmos 1

Belo Horizonte

2024

1. Introdução

Este trabalho prático visa aplicar conceitos de otimização em grafos temporais, especificamente focando na maximização de pontuação em uma pista de skate na comuna fictícia de Radlândia. O problema exige o desenvolvimento de um algoritmo que utilize programação dinâmica para calcular a melhor sequência de manobras em diferentes seções da pista, levando em consideração fatores como bonificação, penalizações e restrições de tempo.

Radlândia, uma sociedade baseada no Deboísmo, possui uma pista de skate onde os habitantes buscam realizar as manobras mais radicais. Diná, uma moradora recente, deseja encontrar a sequência de manobras que maximize sua pontuação em uma pista composta por várias seções. Cada seção permite a realização de um conjunto limitado de manobras, cada uma com uma pontuação base e um tempo necessário para sua execução. Além disso, manobras consecutivas podem sofrer penalizações, o que aumenta a complexidade do problema.

2. Metodologia

Os algoritmos foram implementados em C++ e testados para garantir eficiência na maximização da pontuação. A estrutura principal utilizada é a programação dinâmica MEM, um vetor de Chip, que armazena e utiliza resultados intermediários para evitar recomputações desnecessárias.

Estruturas Utilizados

- **Chip:** Armazena o estado de uma seção específica na DP.
    - action: representa a combinação de manobras realizadas.

    - record: pontuação máxima obtida para essa combinação de manobras.
- **Sections**: Um typedef que representa um vetor de pair de pares bônus e time para cada seção da pista.
    - bonus: bonificação da seção.
    - time: tempo disponível na seção para realizar manobras.
- **Tricks**: Um typedef que representa um vetor de pair de pares bonus e time para cada manobra possível.
    - bonus: bonificação da manobra.
    - time: tempo necessário para executar a manobra.

Técnicas Empregadas

Programação Dinâmica (DP): A MEM armazena os melhores resultados para cada seção e combinação de manobras, otimizando a computação da pontuação máxima.

Combinação de Manobras: As combinações de manobras são testadas para verificar se podem ser realizadas no tempo disponível da seção, maximizando a pontuação com base no estado anterior.

Representação do Campo de Bits: A representação de um conjunto de manobras escolhidas (action) é representada por um inteiro, que é interpretado como bits que representam se a manobra foi realizada. Isso é possível pois K é no máximo 10 e um inteiro tem 32 bits.

3. Implementação

A Programação Dinâmica (DP) é utilizada para calcular a melhor sequência de manobras em cada seção da pista, com o objetivo de maximizar a pontuação total. Para isso dividimos em duas funções:

dp(int id, int lastAction)

Que responsável por calcular a pontuação máxima possível para a seção *id*, dado que a última manobra realizada foi *lastAction*. Ela utiliza Programação Dinâmica (DP) para armazenar e reutilizar resultados de subproblemas, otimizando assim o processo de cálculo. A função começa verificando se o resultado para o estado atual (*id* e *lastAction*) já foi previamente calculado e armazenado na matriz *MEM*. Se sim, retorna imediatamente o resultado armazenado, evitando cálculos redundantes.

Caso contrário, a função explora todas as combinações possíveis de manobras para a seção atual, representadas por bitmasks. Para cada combinação de manobras, a função calcula o tempo total necessário e verifica se ele se encaixa no tempo disponível da seção. Se a combinação é válida, a função calcula a pontuação total para essa combinação, considerando o bônus de cada manobra e aplicando penalizações se uma manobra é realizada consecutivamente. Se a seção não for a última, a função faz uma chamada recursiva para calcular a pontuação da próxima seção, combinando o resultado obtido com o atual. O melhor resultado encontrado é então armazenado na matriz *MEM* e retornado.

updateScore(int id, int action, int points, int lastAction, ll r, ll nextR)

Essa função atualiza a matriz de memória *MEM* com a pontuação máxima encontrada para uma combinação específica de seção (*id*) e a última manobra (*lastAction*). A função calcula a pontuação atual considerando o bônus da seção e o número de manobras realizadas, ajustando a pontuação total com base na pontuação da próxima seção (*nextR*). Se a pontuação calculada (*currentScore*) for superior à pontuação armazenada previamente (*r*), a função atualiza a matriz *MEM* com a nova manobra (*action*) e a pontuação (*currentScore*). Essa atualização garante que a matriz de memória sempre contém o melhor resultado possível para cada combinação de estado, otimizando a solução geral do problema.

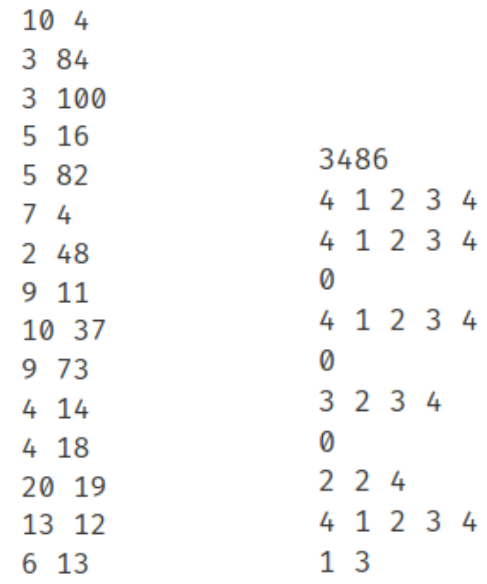
4. Testes

Para testar a implementação, foram utilizadas entradas com diferentes números de seções e manobras. Alguns dos testes feitos foram os presentes no enunciado do trabalho. Além desses implementamos mais dois testes, sendo eles:

Teste a

Entrada	Saída
10 4 10 67 8 50 10 87 10 74 10 59 9 38 4 4 8 39 5 20 8 70 1 5 8 7 12 11 17 19	7150 4 1 2 3 4 0 4 1 2 3 4 0 4 1 2 3 4 3 2 3 4 0 3 4 3 2 0 4 1 2 3 4

Teste b



Entrada	Saída
10 4 3 84 3 100 5 16 5 82 7 4 2 48 9 11 10 37 9 73 4 14 4 18 20 19 13 12 6 13	3486 4 1 2 3 4 4 1 2 3 4 0 4 1 2 3 4 0 3 2 3 4 0 2 2 4 4 1 2 3 4 1 3

Além de resolver os testes com o algoritmo, decidi fazer o teste 1 do enunciado no papel. Não só para verificar a exatidão, mas também para pensar em como o problema deveria ser resolvido.

4. Discussão dos Resultados

Os resultados dos testes indicaram que o algoritmo é eficiente em identificar a combinação de manobras que maximiza a pontuação em cada seção da pista de skate. A precisão do algoritmo foi avaliada com base na capacidade de encontrar a sequência ótima de manobras, levando em consideração as restrições de tempo e as bonificações associadas a cada seção e manobra. O tempo de execução mostrou-se adequado, na maioria dos casos, mesmo para pistas com um grande número de seções e manobras possíveis.

Embora o algoritmo tenha sido otimizado, a abordagem adotada, **top-down**, não é a mais rápida quando comparada à abordagem **bottom-up**. Devido à escolha dessa abordagem, não é possível otimizar o código além do que já está, o que resultou na **falha do código no teste 22** do VPL.

Ao tentar uma nova otimização, alterando o valor de retorno para MEM a fim de evitar a criação de uma nova estrutura, o código acabou falhando em mais testes do que antes. Com esse resultado, decidi reverter o código para sua forma original e o enviei novamente. Com isso, **todos os testes foram aprovados**, incluindo o teste 22, que havia falhado anteriormente. Isso sugere que o resultado desse teste pode variar de execução para execução, indicando que ele é possivelmente um **flaky test**.

5. Conclusão

A implementação deste trabalho permitiu aplicar conceitos de Programação Dinâmica e otimização em sistemas complexos, como o problema de maximização de pontuação na pista de skate. O algoritmo desenvolvido foi capaz de otimizar a pontuação de Diná, levando em consideração múltiplos fatores e restrições, como o tempo disponível em cada seção e as penalizações para manobras consecutivas.

Embora a natureza exponencial do problema represente um desafio significativo, a utilização de Programação Dinâmica contribuiu para melhorar a eficiência do algoritmo, ainda que ele não tenha um tempo de execução linear. A solução

desenvolvida demonstrou ser robusta e eficiente, respeitando os limites de tempo e memória definidos e oferecendo uma resposta eficaz ao problema proposto.

6. Bibliografia

➔ Count Ways To Assign Unique Cap To Every Person. Disponível em:

https://www.geeksforgeeks.org/bitmasking-and-dynamic-programming-set-1-count-ways-to-assign-unique-cap-to-every-person/

Acesso em: 20 de Agosto de 2024.

➔ Algoritmo Dynamic Programmer. Disponível em:

https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/DP/dcDp.cpp

Acesso em: 20 de Agosto de 2024.