MAXIMIZAÇÃO DE LUCRO

Algoritmo guloso e programação dinâmica

Ana Júlia Dias, Bianca Rangel, Gabriel Ferreira,
Lucas Gabriell Ferreira e Pedro Henrique Alves

INTRODUÇÃO

Qual combinação de pratos maximiza o lucro?

- Custo e lucro dos pratos conhecidos.
- Lucro de um prato reduzido pela metade caso esteja sendo cozinhado pela segunda vez seguida.
- Lucro de um prato zerado caso esteja sendo cozinhado pela terceira vez seguida em diante.

METODOLOGIA: PARADIGMA GULOSO

- Otimização local por meio da relação lucro/custo.

- A cada dia, procura-se o prato que oferece a melhor relação lucro/custo, considerando os pratos dos dois últimos dias para reajustar o lucro.

- Não garante solução ótima:
  - Uma decisão localmente ótima pode resultar em uma redução dos lucros dos dias seguintes.
  - Orçamento pode não ser eficientemente aproveitado devido ao limite de dias.

ALGORITMO GULOSO

- **Inicialização:** Configura variáveis para armazenar a combinação de pratos, a escolha atual, o custo acumulado, e o lucro alcançável, e faz uma cópia das relações lucro/custo.

- **Iteração diária:** Para cada dia no menu, busca o prato com a melhor relação lucro/custo que cabe no orçamento restante.

- **Escolha do prato:** Se encontra um prato que maximiza o lucro dentro do orçamento, ele é escolhido e suas informações são atualizadas.

```
class Greedy {
  greedyAnalysis() {

    for (let menu = 0; menu < this.numMenu; menu++) {
      let combination = [];
      let choice = -1;
      let cp = -1; // Valor de lucro por custo
      let sumCost = 0;
      let achievableProfit = 0;
      let ppcCopy = this.planning.dishes.profitPerCost[menu].slice(); //

      // Itera por cada dia do planejamento
      for (let day = 0; day < this.planning.days[menu]; day++) {
        // Encontra o prato com a melhor relação lucro/custo que cabe no
        for (let dish = 0; dish < this.planning.numDishes[menu]; dish++)
          if (cp < ppcCopy[dish]) {
            if (
              sumCost + this.planning.dishes.cost[menu][dish] <
              this.planning.budget[menu]
            ) {
              cp = ppcCopy[dish];
              choice = dish + 1;
            }
          }
        }
```

METODOLOGIA: PROGRAMAÇÃO DINÂMICA

- Tabela multidimesional: **table[day][lastDish][count][budget]**
- Cada estado da tabela representa uma combinação possível de dias, pratos, contagem consecutiva e orçamento.
- Para cada dia, decide-se qual prato cozinhar, levando em consideração o custo, o lucro e a penalidade por cozinhar o mesmo prato consecutivamente.
- Preenche-se a tabela para todos os dias e, no último dia, busca-se o estado que proporciona o maior lucro.

ALGORITMO DINÂMICO

- **Inicialização:** Inicializa a tabela dinâmica com lucro e custo zero no estado inicial.

- **Iteração:** O algoritmo itera sobre cada dia, considerando todos os pratos possíveis como o último prato cozinhado no dia anterior. Em seguida, percorre todas as contagens consecutivas possíveis e valores de orçamento para calcular os lucros acumulados.

- **Cálculo de lucro e custo:** Calcula e atualiza lucro e custo na tabela dinâmica considerando o orçamento e as penalidades.

```
// Inicializa o estado base
table[0][0][0][m] = { profit: 0, cost: 0 };

// Preenche a tabela dinâmica
for (let day = 0; day < k; day++) {
  // Itera sobre todos os pratos possíveis como último prato cozinhado
  for (let lastDish = 0; lastDish <= n; lastDish++) {
    // Itera sobre todas as contagens consecutivas possíveis
    for (let count = 0; count < 2; count++) {
      // Itera sobre todos os valores de orçamento possíveis
      for (let budget = 0; budget <= m; budget++) {
        // Pula estados inválidos
        if (table[day][lastDish][count][budget].profit === -Infinity)
          continue;

        // Tenta cozinhar cada prato disponível no dia atual
        for (let dish = 1; dish <= n; dish++) {
          const { cost, profit } = dishes[dish - 1];
          // Verifica se há orçamento suficiente para cozinhar o prato
          if (budget >= cost) {
            let newProfit = table[day][lastDish][count][budget].profit;
            let newCost = table[day][lastDish][count][budget].cost + c
            let newCount = 0;
```

SOLUÇÃO

- Linguagem utilizada: JavaScript.
- Interface web feita com Svelte.

CONCLUSÃO

O algoritmo com programação dinâmica, apesar de possuir uma pior ordem de complexidade ($O\left(n^4\right)$), com relação ao guloso ($O\left(n^3\right)$), considerando os limites da entrada de dados, é mais adequado para o problema por garantir a solução ótima global.

REFERÊNCIAS

ROCHA, Anderson; DORINI, Leyza Baldo. Algoritmos gulosos: definições e aplicações. Campinas, SP, v. 53, 2004.

VASCONCELOS, Amália Soares Vieira. Programação Dinâmica. 2024. Betim. Notas de aula.

OBRIGADO!