Universidade Federal de Ouro Preto - UFOP
Departamento de Ciência da Computação - DECOM

Relatório atividade 2 - Yahtzee

BCC402 - ALGORITMOS E PROGRAMACAO AVANCADA

Kayo Xavier Nascimento Cavalcante Leite - 21.2.4095

Professor: Rafael Alves Bonfim

Ouro Preto
31 de março de 2025

Sumário

Lista de Códigos Fonte

1 Código e enunciado.

Na Atividade 2 - o problema selecionado foi o Yahtzee. O Yahtzee é um jogo de dados que envolve estratégia para maximizar a pontuação total ao longo de 13 rodadas. Cada rodada deve ser atribuída a uma das 13 categorias de pontuação, cada uma com regras específicas. O objetivo é escolher a melhor categoria para cada conjunto de dados visando a maior pontuação possível. O código comentado e documentado, casos de teste e executável pré compilado se encontram no .zip da atividade. O código foi feito com base na referência encontrada no site:

`https://github.com/evandrix/UVa/blob/master/10149.cpp`

Caso queira, para rodar e compilar o código, é necessário ter o compiler g++ e utilizar o seguinte comando no terminal dentro do diretório da pasta da atividade específica:

Compilando e rodando o exercício

```
para compilar:
g++ Yahtzee.cpp -o executavel

e para rodar basta utilizar .\executavel no cmd.

para utilizar os cenários de teste:
.\executavel < sampleinput.txt
.\executavel < testinput.txt
.\executavel < test.txt
```

2 Pseudocódigo e descrição do problema

Entrada: 13 rodadas com 5 dados cada
Saída: Pontuação máxima e alocação ótima

1. **Pré-Cálculo**:
   - Para cada rodada k e categoria j, calcule `score[k][j]`
   - Ordenação dos dados para facilitar verificações (ex: `sort(DICE[k])`)
2. **Inicialização da DP**:
   - Tabela `dp[mask][sum_upper]` onde:

$$\texttt{mask} \in [0, 2^{13} - 1] \quad \text{(bitmask para categorias usadas)}$$
$$\texttt{sum\_upper} \in [0, 63] \quad \text{(soma das categorias 1-6)}$$

   - Estado inicial: `dp[0][0] = 0`
3. **Preenchimento da Tabela**:
   - Para cada estado `(mask, sum_upper)`:

$$\texttt{new\_mask} = \texttt{mask} \mid (1 \ll j) \quad \text{(adiciona categoria } j\text{)}$$

   - Atualizações:

$$\texttt{new\_sum} = \min(\texttt{sum\_upper} + \texttt{score}, 63)$$
$$\texttt{dp[new\_mask][new\_sum]} \leftarrow \max(\text{valor atual}, \texttt{dp[mask][sum\_upper]} + \texttt{score})$$

4. **Backtracking**:
   - Reconstrução da solução usando `arg_dp[mask][sum_upper]`

- Determinação das categorias usadas em cada rodada

5. **Bônus**:

$$\text{Bônus} = \begin{cases} 35, & \text{se } \sum_{j=1}^{6} \text{Pontos}(j) \geq 63 \\ 0, & \text{caso contrário} \end{cases}$$

2.1 Otimizações-Chave

- **Ordenação de Dados**: Permite verificações rápidas de combinações (ex: `dice[0] == dice[2]` para trincas)
- **Máscara de Bits**: Representação compacta de estados (13 categorias → 13 bits)
- **Limitação do Bônus**: `sum_upper` limitado a 63 para reduzir espaço de estados

2.2 Complexidade

- **Tempo**: $O(13 \times 2^{13} \times 64) \approx 6.7 \times 10^6$ operações
- **Espaço**: $O(2^{13} \times 64) \approx 524,288$ estados

```
// Problema: Maximizar pontuacao tipo Yahtzee com 13 rodadas de dados fixas (DICE[13][5]).
// Cada rodada deve ser usada em uma categoria unica (1-13). Considerar bonus de 35 pontos se soma das categorias 1-6 >= 63.

// Funcao score_cat(dados[5], categoria): Retorna pontuacao da categoria para os dados (assume dados ordenados).

// Funcao cnt_bit(mascara): Retorna numero de bits ligados na mascara.

// Globais:
//   DICE[13][5]: Dados de entrada.
//   dp[1<<13][64]: Tabela DP. dp[mascara][soma_bonus] = max_pontos. 'soma_bonus' e a soma das categorias 1-6 (limitada a 63).
//   score[13][13]: score[k][j] = pontuacao pre-calculada dos dados da rodada k para categoria j.
//   arg_dp[][][]: Tabela para backtracking (guarda ultima categoria adicionada e soma_bonus anterior).

// Procedimento sol_dp():
// 1. Pre-calcular score[k][j] para todas as rodadas k e categorias j usando score_cat.
// 2. Inicializar dp[][] = -1, dp[0][0] = 0.
// 3. Loop k de 0 a 12 (representa adicionar a k-esima rodada/categoria):
// 4.   Loop i sobre todas as mascaras (0 a 2^13-1):
// 5.     Se cnt_bit(i) == k: // Processa estados com k categorias usadas
// 6.       Loop j de 0 a 12 (representa a categoria a ser adicionada com dados k):
// 7.         Se categoria j nao esta em i: // Se a categoria j esta livre
// 8.           pontos_cat_j = score[k][j]
// 9.           nova_mascara = i | (1 << j)
// 10.          incremento_bonus = (j < 6) ? pontos_cat_j : 0
// 11.          Loop p de 0 a 63 (soma_bonus anterior):
// 12.            Se dp[i][p] >= 0: // Se estado anterior e valido
// 13.              nova_soma_bonus = min(p + incremento_bonus, 63)
// 14.              Se dp[i][p] + pontos_cat_j > dp[nova_mascara][nova_soma_bonus]:
// 15.                Atualizar dp[nova_mascara][nova_soma_bonus]
// 16.                Salvar j e p em arg_dp para backtracking.
```

```
// 17. Encontrar max_pontos em dp[2^13 - 1][p] para p de 0 a 62. Guardar p_final.
// 18. Calcular pontos_com_bonus = dp[2^13 - 1][63] + 35.
// 19. Se pontos_com_bonus > max_pontos, atualizar max_pontos, p_final=63, bonus=35.
// 20. Fazer Backtracking usando arg_dp a partir de (mascara=2^13-1, p_final) para encontrar qual rodada (i) foi usada para qual categoria (cat) e a pontuacao (pontos_finais[cat] = score[i][cat]).
// 21. Imprimir pontos_finais[0..12], bonus, max_pontos.

// Procedimento main():
// 1. Loop infinito:
// 2.   Ler 13x5 dados em DICE[][]. Se falhar (EOF), terminar.
// 3.   Ordenar os 5 dados de cada rodada k: sort(DICE[k]).
// 4.   Chamar sol_dp()
```

Código 1: Pseudocódigo do problema.

3 Casos teste - Input e output esperado.

Para os casos de teste do problema, foi disponibilizado junto a pasta do mesmo os seguintes arquivos :sampleinput.txt, test.txt e testinput.txt, sendo o primeiro o próprio caso de teste disponibilizado pelo exercício e o segundo e terceiro casos de teste encontrados na plataforma https://www.udebug.com/. Além disso, encontra-se também o arquivo com os outputs esperados para cada input. Ambos os resultados foram validados e tiveram o output esperado.

sampleinput.txt

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 1 1 1 1
6 6 6 6 6
6 6 6 1 1
1 1 1 2 2
1 1 1 2 3
1 2 3 4 5
1 2 3 4 6
6 1 2 6 6
1 4 5 5 5
5 5 5 5 6
4 4 4 5 6
3 1 3 6 3
2 2 2 4 6
```

output esperado

```
1 2 3 4 5 0 15 0 0 0 25 35 0 0 90
3 6 9 12 15 30 21 20 26 50 25 35 40 35 327
```

Os demais testes se encontram no diretório da atividade