UNIVERSIDADE FEDERAL DE MINAS GERAIS
Graduação em Engenharia de Sistemas
Algoritmos e Estruturas de Dados III - Trabalho Prático 1
Mortal Kontest
Matheus Silva Araujo - 2013066265

1 Introdução

No Trabalho Prático 1 da disciplina de Algoritmos e Estruturas de Dados III, o adolescente peculiar Nubby tem um novo problema e quer usar *Programação Dinâmica* para resolvê-lo.

Nubby e seus amigos irão jogar um campeonato de *Mortal Kontest*, e Nubby possui as probabilidades de vitória e derrota para cada confronto possível entre seus amigos.

No campeonato, dois participantes são sorteados para jogar a primeira rodada [1]. O competidor que perder é eliminado do campeonato e não volta mais a jogar. O vencedor continua na competição. Um novo participante é sorteado; eles jogam e o vencedor dessa nova rodada continua jogando. O campeonato prossegue assim até que haja apenas dois competidores na última rodada, o que vencer essa partida é declarado campeão.

Dadas as probabilidades de vitória e derrota de cada confronto, Nubby deseja saber quais são as chances que cada competidor tem de ser o campeão.

2 Solução do Problema

A ideia inicial para a resolução do problema é que, em um cenário onde $n = 4$ e os jogadores são A, B, C e D, a probabilidade de A ser o campeão depende das probabilidades das três possíveis *finais*: AB, AC e AD.

A é o campeão quando B, C ou D *perdem* essas *finais*.

Para que a final AB aconteça, é necessário que na rodada anterior estejam ABC ou ABD. Ou seja, AB depende da probabilidade de que C perca numa segunda rodada onde há ABC ou que D perca numa segunda rodada com ABD.

A probabilidade de um jogador perder em uma rodada é o somatório das probabilidades de uma partida acontecer multiplacando a probabilidade que o jogador tem de perder essa partida.

ABC e ABD dependem - por consequência - de $ABCD$ que tem probabilidade $P(ABCD) = 1$.

Por indução, percebe-se que a probabilidade de um subconjunto S, $|S| = i$ é função das probabilidades dos subconjuntos S', $|S'| = i + 1$ que originam S.

Portanto, calcular a probabilidade de um conjunto S' é um subproblema para calcular a probabilidade do subconjunto S.

Ainda para o exemplo $n = 4$, no momento de calcular a probabilidade de B, ela dependerá das probabilidade AB, BC e BD, no entanto, a probabilidade de AB já foi calculada quando a probabilidade de A foi calculada, portanto há a **memorização** dos cálculos. Tornando o algoritmo dinâmico mais performático do que uma abordagem de força bruta, por exemplo.

Para o cálculo de C, a memorização terá sido ainda maior. No cálculo de D, apenas um cálculo será feito.

As Figuras 1 e 2 exemplificam esse processo de memorização.

[1] As chances de sorteio de cada jogador são sempre as mesmas.

Figura 1: Memorização em $P(A)$ e $P(B)$

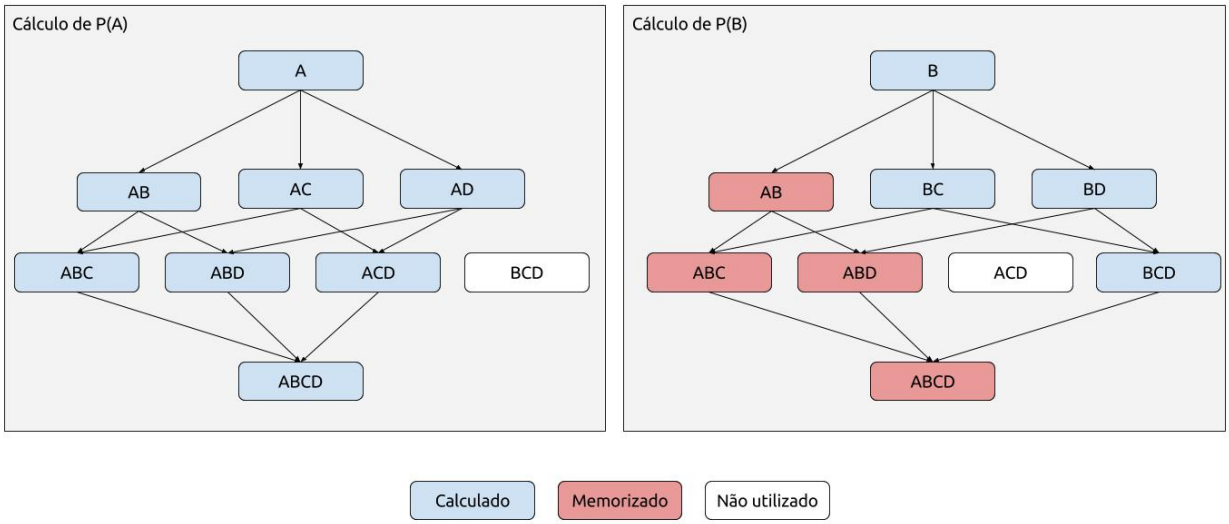
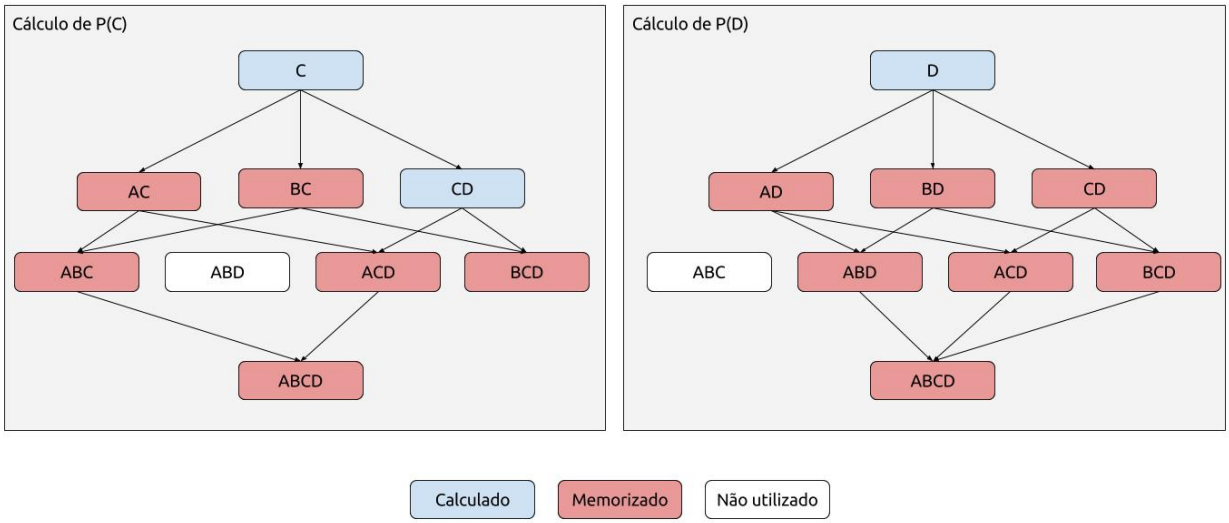


Figura 2: Memorização em $P(C)$ e $P(D)$



2.1 Equação de Recorrência

A equação de recorrência para o problema é:

$$P(S) = P_j(|S'|) \times \sum_{i \notin S} \Big(P(S') \times \sum_{j \in S'} P(j \bigtriangledown i) \Big)$$

Onde:

- S' é um conjunto que originou S tendo apenas um elemento i removido
- $P_j(|S'|)$ é a probabilidade de um jogo no conjunto S', calculada pela expressão:

$$P_j(|S'|) = \frac{2}{|S'| \times (|S'| - 1)}$$

- $P(j \bigtriangledown i)$ é a probabilidade de um jogador j vencer o jogador i.

3 Análise de Complexidade

Com o algoritmo de *Programação Dinâmica* apresentado, a probabilidade de cada subconjunto possível é calculada apenas uma vez, devivo à memorização. Existem 2^n subconjuntos possíveis.

Para calcular a probabilidade de cada subconjunto há um custo de, no pior caso, n operações.

Portanto, a complexidade do algoritmo é de $O(n \cdot 2^n)$.

4 Uso de memória

Para armazenar as probabilidades de cada subconjunto é possível utilizar apenas um vetor, cada elemento desse vetor apresenta a probabilidade de um subconjunto, portanto esse vetor tem tamanho 2^n.

5 Avaliação Experimental

O algoritmo de *Programação Dinâmica* foi avaliado, e adicionalmente foi feita uma comparação de seu desempenho com um algoritmo de *Força Bruta*.

5.1 Algoritmo *Programação Dinâmica*

Para avaliar a implementação do algoritmo de *Programação Dinâmica* foram criados 25 casos de teste com n variando de 1 a 25 e tomadas as medidas de tempo e uso de memória. Os testes foram feitos em ambiente *Linux Ubuntu*.

Para medir o tempo, a aplicação foi executada três vezes para cada valor de n e os tempos medidos através do comando *time*.

Para medir a memória, foi usada a ferramenta *valgrind*.

A Tabela 1 apresenta os valores medidos. Na Figura 3 é apresentado um gráfico do tempo em função de n. Já na Figura 4 é apresentado um gráfico do uso de memória, os valores da tabela são apresentados em função de log_2.

Tabela 1: Medidas de tempo e memória.

n	**T1 (s)**	**T2 (s)**	**T3 (s)**	**T méd. (s)**	**Memória (bytes)**
1	0.002	0.002	0.002	0.0020	5132
2	0.002	0.002	0.002	0.0020	5152
3	0.002	0.002	0.002	0.0020	5188
4	0.002	0.002	0.002	0.0020	5248
5	0.002	0.003	0.002	0.0023	5348
6	0.002	0.002	0.002	0.0020	5520
7	0.002	0.002	0.002	0.0020	5828
8	0.002	0.002	0.001	0.0017	6400
9	0.003	0.002	0.003	0.0027	7492
10	0.001	0.003	0.003	0.0023	9616
11	0.004	0.004	0.004	0.0040	13796
12	0.005	0.003	0.006	0.0047	22080
13	0.010	0.006	0.007	0.0077	38564
14	0.015	0.016	0.015	0.0153	71440
15	0.018	0.018	0.019	0.0183	137092
16	0.031	0.032	0.033	0.0320	268288
17	0.062	0.061	0.062	0.0617	530564
18	0.123	0.124	0.126	0.1243	1054992
19	0.261	0.269	0.287	0.2723	2103716
20	0.729	0.721	0.712	0.7207	4201024
21	1.772	1.813	1.75	1.7783	8395492
22	4.014	3.956	3.999	3.9897	16784272
23	8.978	8.925	9.022	8.9750	33561668
24	18.612	18.806	19.857	19.0917	67116288
25	41.552	40.269	41.13	40.9837	134225348

Figura 3: Medidas de tempo

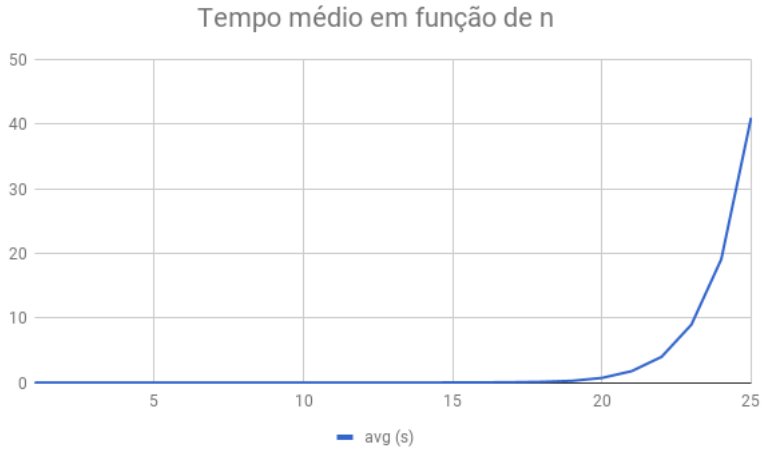
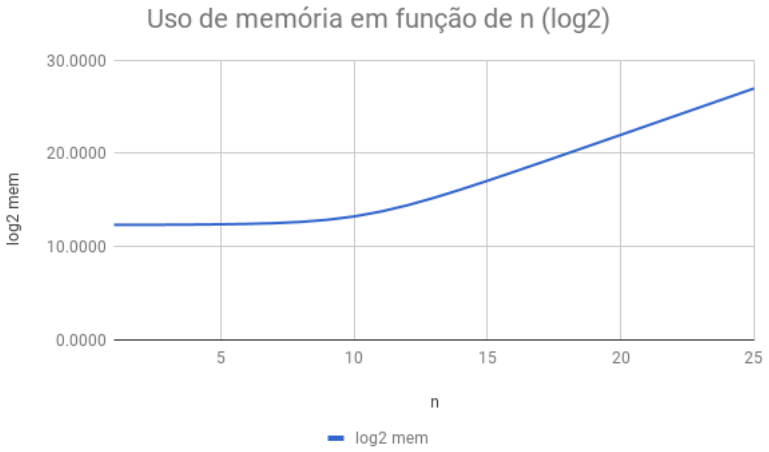


Figura 4: Alocação de memória



5.2 Comparação com *Força Bruta*

Durante o desenvolvimento do Trabalho Prático, foi implementado um algoritmo de *Força Bruta* que também resolvia o problema. A fim de comparação, os dois algoritmos foram executados através dos casos de teste *toys* fornecidos e os tempos de execução foram comparados.

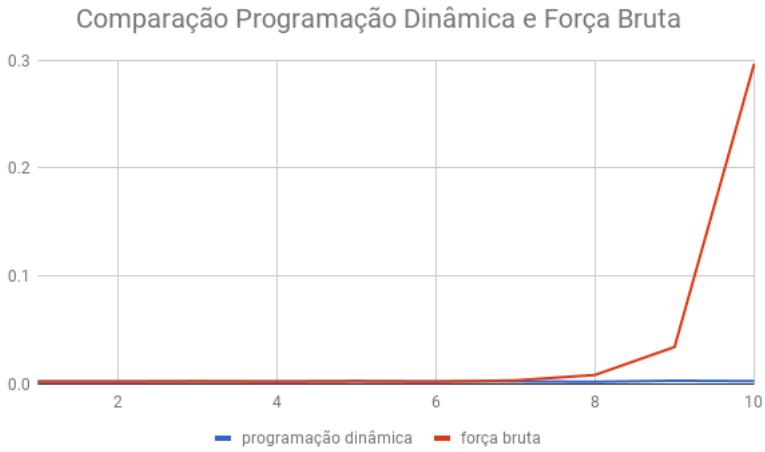
Esse algoritmo *Força Bruta* tem complexidade temporal $O(n!)$.

Os resultados são apresentados na Tabela 2, adicionamente um gráfico é apresentado na Figura 5.

Tabela 2: Comparação com algoritmo de *Força Bruta*

	Programação dinâmica				***Força Bruta***			
n	**T1 (s)**	**T2 (s)**	**T3 (s)**	**T méd. (s)**	**T1 (s)**	**T2 (s)**	**T3 (s)**	**T méd. (s)**
1	0.002	0.002	0.002	0.0020	0.002	0.001	0.002	0.0017
2	0.002	0.002	0.002	0.0020	0.002	0.002	0.001	0.0017
3	0.002	0.002	0.002	0.0020	0.002	0.002	0.002	0.0020
4	0.002	0.002	0.002	0.0020	0.002	0.002	0.001	0.0017
5	0.002	0.003	0.002	0.0023	0.002	0.002	0.002	0.0020
6	0.002	0.002	0.002	0.0020	0.002	0.001	0.002	0.0017
7	0.002	0.002	0.002	0.0020	0.003	0.003	0.003	0.0030
8	0.002	0.002	0.001	0.0017	0.009	0.004	0.011	0.0080
9	0.003	0.002	0.003	0.0027	0.031	0.036	0.035	0.0340
10	0.001	0.003	0.003	0.0023	0.297	0.296	0.297	0.2967

Figura 5: Comparação de algoritmos



6 Conclusão

Durante a construção do algoritmo e a implementação foi possível notar as vantagens da programação dinâmica.

Foi possível sair de um algoritmo com ordem de complexidade temporal de $O(n!)$ para um com $O(n \cdot 2^n)$.

Só foi possível *encontrar* o algoritmo *dinâmico* após a implementação do algoritmo *força bruta* e perceber as repetições que permitiam a memorização. O que mostra que algoritmos dinâmicos podem ser entendidos como a combinação de outros algoritmos e memorização.