UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Algoritmos e Estruturas de Dados III
Trabalho Prático 1 – Mortal Kontest

Orlando Enrico Liz Silvério Silva

Belo Horizonte
18 de outubro de 2017

1 Introdução

A maioria dos problemas da computação podem ser otimizados aplicando conceitos de paradigmas de projetos de algoritmos. Às vezes, a forma como um algoritmo aborda um problema pode levar a um desempenho ineficiente ou não resolve o problema em tempo viável. Sendo assim, este trabalho tem como objetivo utilizar o paradigma de programação dinâmica para calcular a probabilidade de cada um dos N amigos de Nubby ganhar o campeonato de *Mortal Kontest* baseado nos seus registros de partidas de campeonatos anteriores.

2 Solução do Problema

Esta seção tem como objetivo apresentar a solução implementada. Para tanto, inicialmente será introduzida a modelagem proposta, ilustrando as etapas da resolução do problema. Em seguida, serão apresentados detalhes da programação dinâmica. Por fim, a estrutura da solução é mostrada.

2.1 Modelagem do Problema

A entrada consiste, na primeira linha, de um inteiro positivo N ($1 \leq N \leq 25$), indicando o número de amigos de Nubby que participarão do campeonato. Cada uma das N linha seguintes possui N números reais que representam a matriz de probabilidade P^{NxN} ($0 \leq a_{i,j} \leq 1$), sendo essa a probabilidade de um amigo i vencer um amigo j. Portanto, toda a diagonal principal da matriz será zero, uma vez que um amigo não pode jogar consigo mesmo. Já para as demais células $A_{ji} = 1 - A_{ij}$.

O campeonato, por sua vez, apresenta algumas regras. Ele terá $n - 1$ rodadas em que cada uma possui uma única partida. Para cada partida, dois amigos, v_i e v_j, com $i \neq j$ sendo que a probabilidade de qualquer par de amigos ser sorteado é a mesma. O vencedor de uma partida se mantem no campeonato e o perdedor é eliminado definitivamente. Por fim, na rodada de número $n - 1$ sobram apenas dois amigos.

Portanto, as configurações de campeonato possíveis que resultam em vitória de um amigo A dentre três participantes é:

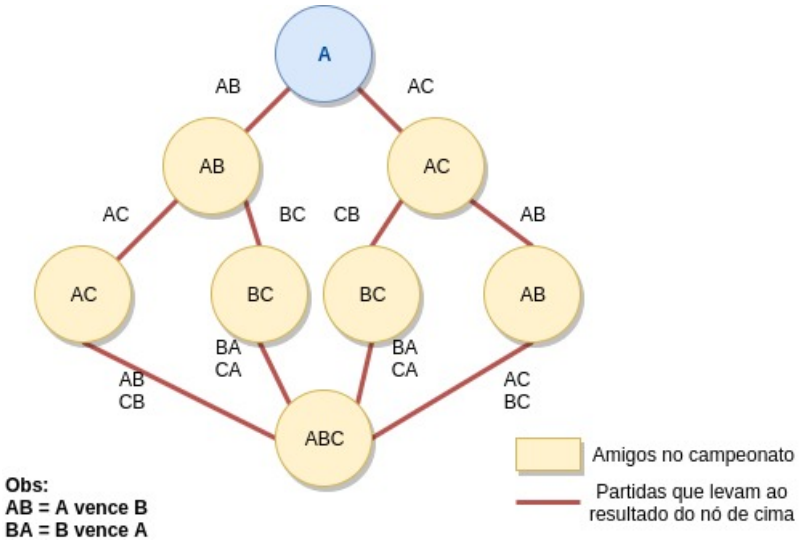

Figura 1 – Configurações possíveis para a vitória de um amigo A

.

A partir da ilustração de uma das 3 possibilidades de campeão feita acima é possível notar

que na mesma árvore existem subconjuntos se repetindo (AB, BC e AC). Ou seja, de acordo com o paradigma de programação dinâmica, podemos evitar o recálculo desses subproblemas comuns através da memorização.

2.1.1 Cálculo da Probabilidade

Para o amigo de Nubby, Bryan (denotado como B e responsável por ajudar Nubby com uma certa árvore de segmentação em trabalho anterior), por exemplo, permanecer no campeonato é preciso vencer outro amigo. Nesse caso, a probabilidade dele ser sorteado para uma partida é o número de possibilidades de escolher um outro amigo, que não o Bryan, (pois já pressupõe que foi) dentre o total de maneiras de se escolher dois jogadores entre todos os possíveis.

$$= \frac{|c|-1}{\binom{|c|}{2}} = \frac{|c|-1}{\frac{|c|!}{2!(|c|-2)!}} = \frac{2}{|c|}$$

Já a de Bryan encontrar com qualquer amigo $x \in c'$ tal que $c' = c - \{B\}$ é $\frac{1}{|c|-1}$. Somando todos os amigos, obtemos a expressão da probabilidade para esse caso:

$$= \frac{2}{|c|} \sum \frac{1}{|c|-1}\ a_{B,x}\ p[c - \{x\}][B]$$

sendo $p[c - \{x\}][B]$ a probabilidade de Bryan se manter no torneio entre os amigos presentes no conjunto $c - \{x\}$.

Além disso, há a possibilidade de Bryan não ser sorteado e, assim, prosseguir no campeonato. Logo, a probabilidade de não sair para jogar com qualquer amigo é $= 1 - \frac{2}{|c|} = \frac{|c|-2}{|c|}$. Nessa vertente, teremos um amigo x e y se enfrentando. Em ambos os casos, suas respectivas probabilidades são contabilizadas, uma vez que, ou x ganha de y, ou y ganha de x. A probabilidade de se sortear esses dois amigos é dada por 1 entre todas as combinações 2 a 2 de elementos em c sem Bryan (que não foi sorteado), ou seja, $= \frac{1}{\binom{|c|-1}{2}}$. E, somando todos os pares de amigos respeitando as restrições do caso chega-se em:

$$= \frac{|c|-2}{|c|} \sum \frac{1}{\binom{|c|-1}{2}} [a_{x,y}\ p[c - \{y\}][B] + [a_{y,x}\ p[c - \{x\}][B]]$$

sendo $p[c - \{y\}][B]$ a probabilidade de Bryan se manter no torneio dentre os amigos presentes no conjunto $c - \{y\}$ e $p[c - \{x\}][B]$ a probabilidade de Bryan se manter no torneio dentre os amigos presentes no conjunto $c - \{x\}$.

Portanto, a probabilidade de Bryan ganhar é dada pela soma dos casos citados anteriormente:

$$p[c][B] = \frac{2}{|c|} \sum \frac{1}{|c|-1} a_{B,x}\ p[c - \{x\}][B] + \frac{|c|-2}{|c|} \sum \frac{1}{\binom{|c|-1}{2}} [a_{x,y}\ p[c - \{y\}][B] + [a_{y,x}\ p[c - \{x\}][B]]$$

$$p[c][B] = \frac{2}{|c|(|c|-1)} \sum a_{B,x}\ p[c-\{x\}][B] + \frac{|c|-2}{|c|} \sum \frac{2}{(|c|-1)(|c|-2)} [a_{x,y}\ p[c-\{y\}][B] + [a_{y,x}\ p[c-\{x\}][B]]$$

$$p[c][B] = \frac{2}{|c|(|c|-1)} \sum a_{B,x}\ p[c - \{x\}][B] + \frac{2}{|c|(|c|-1)} \sum [a_{x,y}\ p[c - \{y\}][B] + [a_{y,x}\ p[c - \{x\}][B]]$$

$$p[c][B] = \frac{2}{|c|(|c|-1)} (\sum a_{B,x}\ p[c - \{x\}][B] + \sum a_{x,y}\ p[c - \{y\}][B] + a_{y,x}\ p[c - \{x\}][B])$$

O termo $\frac{2}{|c|(|c|-1)}$ colocado em evidência nada mais é do que a divisão dos termos dos somatórios pela combinação 2 a 2 dos $|c|$ elementos do conjunto.

$$\frac{1}{\binom{|c|}{2}} = \frac{1}{\frac{|c|!}{2!(|c|-2)!}} = \frac{2!(|c|-2)!}{|c|!} = \frac{2}{|c|(|c|-1)}$$

Portanto, a probabilidade de um amigo x ganhar dentro de um conjunto que só tem ele mesmo é 1. Já a probabilidade de um amigo x ser campeão em um conjunto ou subconjunto formado por ele e um amigo y correponde a $a_{x,y}$

2.1.2 Bitmask

A utilização do paradigma de programação dinâmica nesse caso pressupõe uma memorização das probabilidades. Cada probabilidade memorizada representa um subconjunto do conjunto c.

Para cada amigo do conjunto c estão disponíveis duas escolhas: ou ele é colocado no subconjunto ou não. Essas escolhas são todas independentes. Logo, há $2^{|c|}$ subconjuntos. Devido à isso, foram utilizados *unsigned int* como tipos, pois, uma vez que o maior valor de N pode ser 25, seria preciso indexar 2^{25} posições e o tipo em questão vai de $[0, 2^{32} - 1]$.

Como o problema pede uma lista de controle de participantes do torneio, uma opção é utilizar um vetor indicando com 1 o amigo que ainda está no campeonato e 0 o que foi eliminado. Porém, a solução demoraria a ser calculada por ser necessário fazer consultas ao vetor todo.

Uma alternativa melhor seria a utilização do *Bitmask*. Essa técnica nada mais é que um número binário representando algo. Considerando um conjunto $A = \{1, 2, 3, 4, 5\}$ como exemplo, é possível representar qualquer subconjunto de A usando um *bitmask* de tamanho 5. O subconjunto $\{1, 2, 4\}$ é representado por 01011.

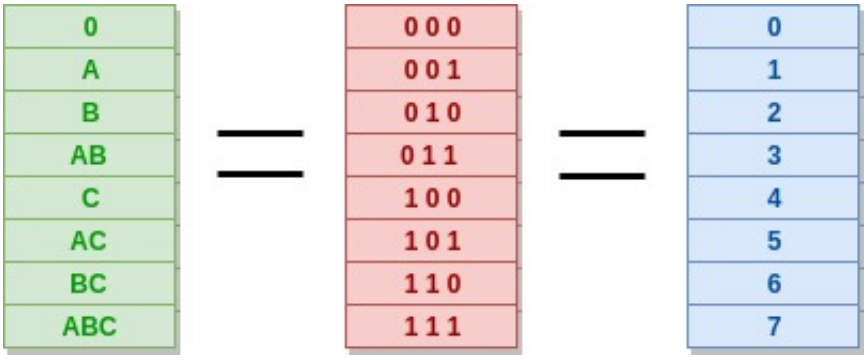

Figura 2 – *Exemplo de bitmask.*

2.1.3 Solução

Para resolver o problema, a probabilidade do maior subconjunto possível (ou seja, o próprio conjunto) é inicializado com 1. A partir disso, os nós de cada nível são calculados a partir dos nós filhos, da probabilidade de enfrentamento dos times participantes do torneio até o momento e da combinação desses times tomados 2 a 2.

Uma vez que a combinação de n elementos é feita tomando amigos 2 a 2, a fórmula pode ser simplificada:

$$= \binom{n}{2} = \frac{n\ (n-1)\ (n-2)!}{2\ (n-2)!} = \frac{n\ (n-1)}{2}$$

Como demonstrado anteriormente, os subconjuntos que indicam a probabilidade de cada time ser campeão são representados por potências de 2. Ou seja, num torneio com 3 times, o A possui índice 2^0, o B 2^1 e o C 2^2.

Segue, na figura abaixo, um exemplo de como a solução é obtida:

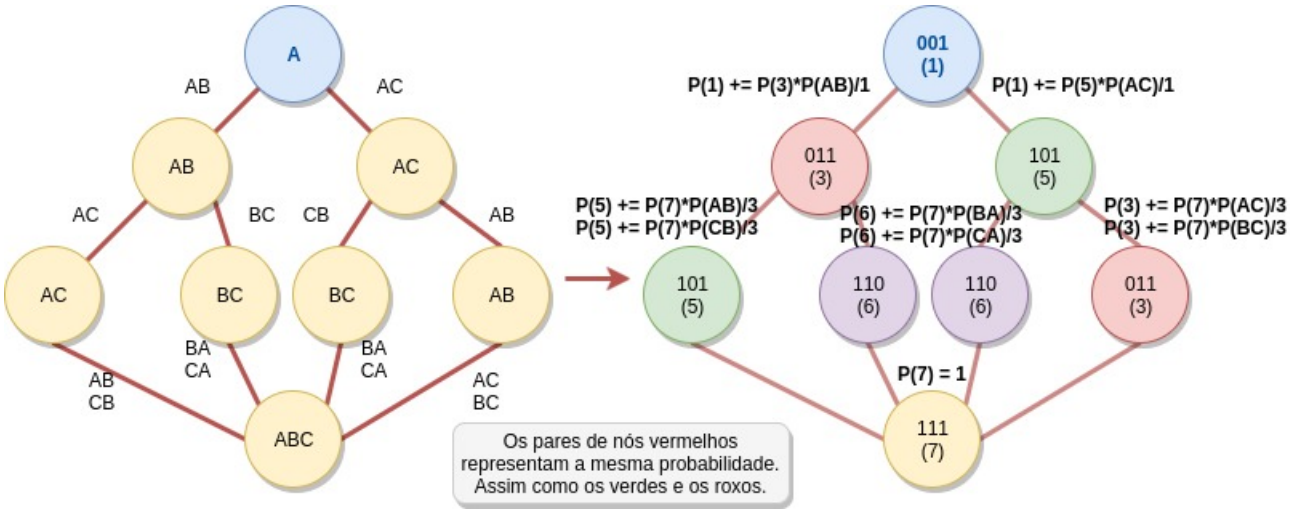

Figura 3 – Exemplo de cálculo de uma amigo A campeão num torneio de 3 amigos.

2.2 Organização do Código

A organização do código, em termos de arquivos, ficou do seguinte modo:

- **matriz.h e matriz.c**: fornecem uma matriz dinamicamente alocada encapsulada em uma struct *Matrix* que também possui o tamanho. Além disso, possui as operações de criação (alocação) e liberação da memória da matriz.
- **main.c**: contem a solução para as probabilidades de cada time ser vencedor do *Mortal Kontest* utilizando o paradigma de programação dinâmica.

3 Análise de Complexidade

Nesta seção serão apresentadas as análises de complexidade de tempo e espaço dos algoritmos implementados.

3.1 Complexidade Temporal

A biblioteca *matriz.h* é responsável por construir a matriz. A alocação é feita em $O(n^2)$. E, como explicado anteriormente, as probabilidades referentes a cada um dos 2^n subconjuntos precisa ser calculada. Sendo assim, há um contador de participantes para o cálculo das combinações 2 a 2 que, no pior caso, é $O(n)$ e uma variação de amigos em uma partida que, no pior caso, é $O(n^2)$ (leva mais tempo que o contador de participantes). Logo, a complexidade é dada por $O(2^n n^2)$ ou $O^*(2^n)$. Para liberar a matriz ao final do programa, a complexidade temporal é a mesma da alocação, ou seja, $O(n^2)$. Para a consulta de uma determinada probabilidade de um amigo x ganhar de um amigo y, o custo é $O(1)$ uma vez que é necessário somente indexar a matriz.

3.2 Complexidade Espacial

Com relação ao vetor usado para memorização, seu tamanho corresponde ao número de subconjuntos possíveis de um determinado conjunto. E, já que um elemento pode ou não estar num subconjunto, o número de possibilidades é 2^n e, portanto, a complexidade é dada por $O(2^n)$.

Já a complexidade de espaço da matriz utilizada para armazenar as probabilidades com base nos jogos anteriores é $O(n^2)$.

4 Análise Experimental

Para analisar o comportamento dos algoritmos desenvolvidos, foi implementado um gerador de testes para verificar o que ocorre com o crescimento do número de amigos (n) participantes do campeonato de *Mortal Kontest* com n variando no intervalo $[2, 25]$.

A Figura 4 ilustra o comportamento assintótico analisado experimentalmente para o crescimento do número de amigos (n). Como esperado, o comportamento do algoritmo confere com a complexidade encontrada anteriormente ($O(2^n n^2)$).

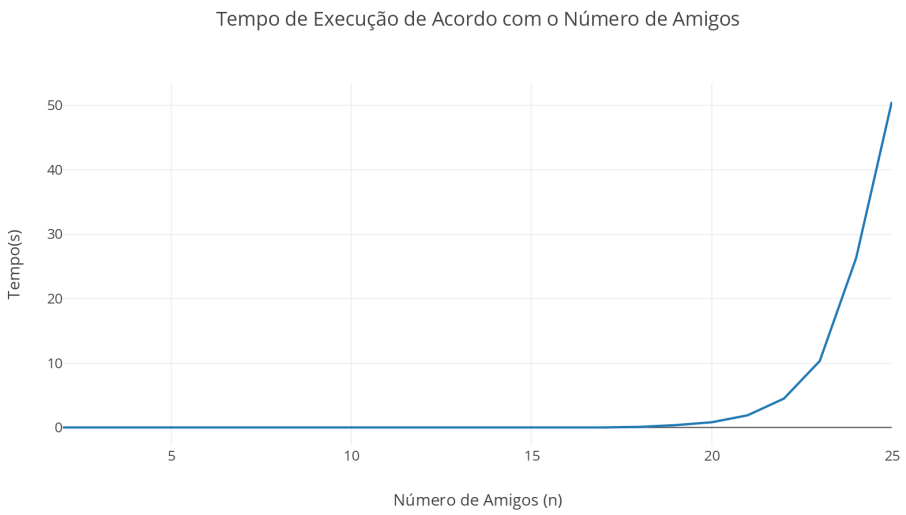

Figura 4 – Análise experimental do algoritmo de acordo com o número de amigos (n) crescente.

5 Conclusão

Esse trabalho apresentou a solução de um problema utilizando o paradigma de programação dinâmica para projetos de algoritmos. Uma vez que o problema dado pode ser dividido em subproblemas menores e são observadas equivalências entre eles, basta memorizar cálculos para serem utilizados posteriormente. Além disso, as soluções ótimas para os subproblemas contribuem para a solução ótima do problema dado (denominado propriedade da Estrutura Ótima). Vale ressaltar também que o uso do *Bitmask* torna a implementação bem mais eficiente.

O paradigma de programação dinâmica, por ser uma técnica refinada, permitiu a obtenção da solução em tempo muito menor se comparada com uma solução construída a partir de tentativa e erro, por exemplo.

6 Referências Bibliográficas

Cormen, Thomas H. Introduction to Algorithms. Cambridge, Mass: MIT Press, 2001.

ZIVIANI, Nivio. Projeto de Algoritmos com implementações em PASCAL e C, 3 ed. Cengage Learning, 2011.

Lecture Slides for Algorithm Design. J. Kleinberg, E. Tardos. Acesso em: 12 out 2017. Disponível em: <http://www.cs.princeton.edu/ wayne/kleinberg-tardos/>.

Algorithms. Jeff Erickson. Acesso em: 12 out 2017. Disponível em: <http://jeffe.cs.illinois.edu/teaching/algorithms/>