Universidade Federal de Minas Gerais — UFMG

Instituto de Ciencias Exatas — ICEx

Departamento de Ciencia da Computacao — DCC

Trabalho Pratico 4

Disciplina:
Algoritmo e Estrutura de Dados 3

Aluno:
Julio Cesar Tadeu Guimaraes

1. Introdução

Neste trabalho teremos que solucionar um problema que envolve programação dinâmica e paralelização, esse problema é definido da seguinte forma. Para um conjunto de vértices T, onde cada vértice é uma arvore que possui peras de qualidades Q e peso P, nesse pomar de pêssego não é possível caminhar livremente entre todas as arvores, existe um conjunto C de caminhos possíveis entre cada arvore, e cada caminho possui uma distância D.

Zambis, nosso personagem, terá uma mochila com capacidade W e ele estará andando de cavalo, esse cavalo só tem a capacidade de andar uma distancia total D.

Meu trabalho é desenvolver um algoritmo que diga qual é a qualidade máxima das peras que posso colocar na minha mochila, e por fim dizer quais as árvores visitadas e quantas peras pegou de cada árvore visita para atingir essa qualidade máxima.

2. Solução Proposta

Para solucionar usaremos três estruturas de dados, a primeira é uma matriz de adjacência que conterá o grafo onde cada vértice é uma árvore e caso exista caminho entre duas árvores, as arestas desses dois vértices terão como peso sua distância. Teremos dois vetores que guardará os pesos e as qualidades das peras de cada árvore. Por último será necessário a criação de uma matriz de três dimensões, onde cada dimensão representa distancia disponível para percorrer, as arvores do pomar e a capacidade disponível na mochila.

Para criação dessa matriz3d é preciso entender que existem algumas variáveis capaz de mudar qual é o valor máximo da capacidade de cada célula, ou seja, caso a distancia = 0, arvore = 1 e capacidade = 0, tem-se que o valor máximo = 0, já que o cavalo não pode andar para nenhuma arvore e sua mochila não cabe nenhuma pera. Logo é possível ver que esses 3 valores são responsáveis para mudar esse valor máximo, assim essa função terá como parâmetro D, V e C, onde D a distancia, V é o vértice e C é a capacidade.

Dessa maneira a criação da matriz 3d fica da seguinte forma:

Para D indo de 0 à Distancia que o cavalo pode percorres:
Para V indo de 1 à maior arvore:
Para C indo de 0 à capacidade da mochila:
Tem-se um valor máximo para MAT[D][V][C]

Esse valor máximo é calculado da seguinte forma.

Para qtdPegada indo de 0 a C/peso[V]
para cada arvore vizinha e alcançável:
Temp = qtdPegada*qualidade[V] +
MAT[D-caminhoEntreAsArvores][vizinha][C-qtdPegada*peso[arvore]]

Pegar o maior valor de temp, para todas essas interações, este é o valor máximo da matriz.

Dessa maneira é possível preencher toda a matriz3d, agora teremos que encontrar qual é o valor máximo do problema, para isso tem-se que verificar na maior capacidade disponível e na maior distancia disponível qual arvore que tem maior valor máximo, assim para essa arvore pode-se percorrer qual caminho Zambis fez entre as arvores para conseguir tal valor máximo.

Essa solução utiliza o paradigma da programação dinâmica, tendo como base a matriz3d, pois essa matriz possui alguns casos bases descritos abaixo:

C = 0
D < 0
C < P[arvore] & D <= 0

Nesses casos o valor máximo será 0, dessa maneira para calcular o restante da matriz, onde não se encaixa esse caso base, teremos o somatório denominado Temp acima, para calcular esse Temp serão utilizados valores da Matriz que já foram calculados, que no caso serão esses casos bases. Dessa maneira, a criação da matriz torna-se mais eficiente.

Esta solução também contempla o paradigma da programação em paralelo, para isso é necessário criar threads para execução em paralelo do programa, essas threads serão criadas na função main, no lugar dos três for's de D, V, C. Para cada distancia, será criada V threads que serão responsáveis pelo calculo do valor máximo para todas as capacidades de cada árvore, para isso será utilizado a biblioteca pthread.

Essas threads serão criadas e assim começaram a preencher a linha correspondente a arvore que elas estão executando, o próximo nível de distancia só será calculado, quando todas as threads do nível um forem executadas, para que isso seja possível é necessário utilizar a função pthread_join, para todas as

threads criadas. Ou seja, depois que todas as threads criadas retornarem de sua execução a distancia será incrementada e novas threads serão geradas para o calculo da nova linha de cada árvore, para a nova distancia e todas as capacidades possíveis.

3. Analise de complexidade

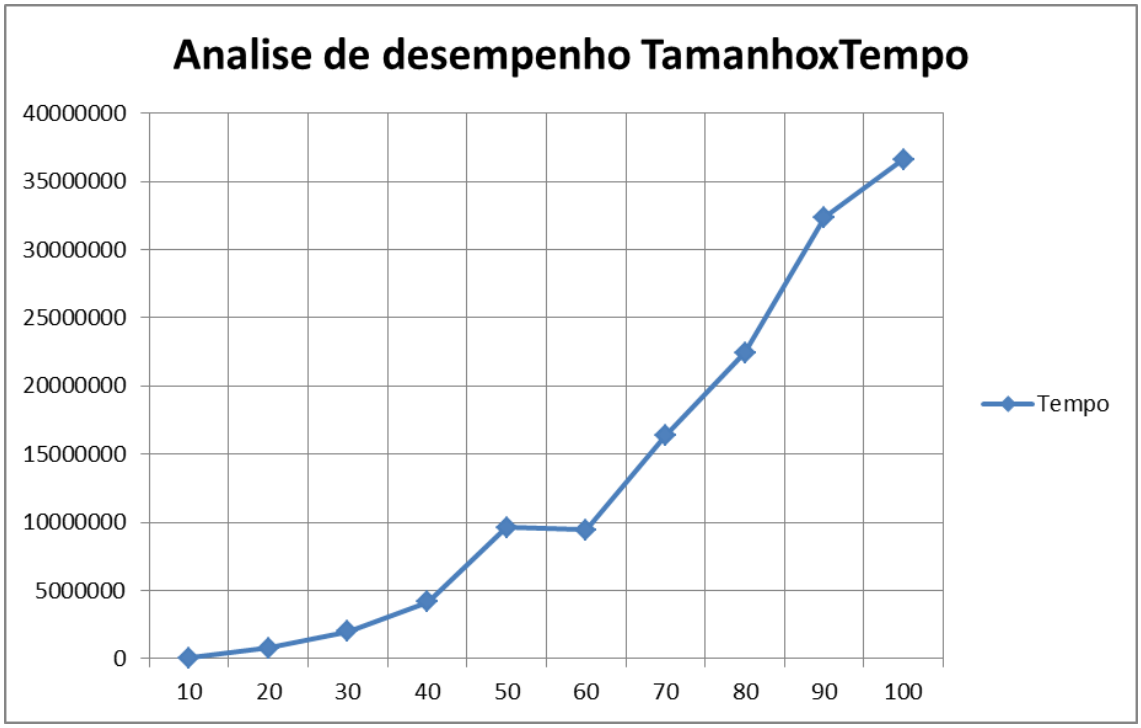
A complexidade de espaço desse problema será a soma dos espaços gastos pelo grafo, pelos vetores peso e qualidade e pela matriz3d, como o grafo é uma matriz nxn, onde n = qtdvertices, a complexidade é igual O(n*n). Os vetores terá um tamanho também dado pela quantidade de árvores, logo cada um será O(n). E por ultimo teremos a matriz3d, que terá como dimensões a distancia disponível, a quantidade de arvores e a capacidade da mochila, logo sua complexidade é dada por O(D*n*C).

A complexidade de tempo dessa solução é limitado pelo tempo de criação da matriz3D, que como foi mostrado é O(D*n*C*(C/peso[arvore])*arvore[adjs]), pois para cada distancia de 0 a máxima, para cada árvore e para cada capacidade de 0 a máxima, tem-se que calcular o valor máximo, esse valor máximo é comparado para caso eu pegue nenhuma pera da primeira arvore e ande apenas, ou eu pego até a quantidade máxima de peras que cabem na mochila dessa arvore, levando em consideração que eu posso andar para todas as árvores vizinhas da árvore inicial desde que o cavalo consiga alcançar. Dado por tanto como $O(C^2*V^2*D)$.

Essa complexidade não contempla a implementação da paralelização do código.

4. Experimentos

Os experimentos realizados foram exemplificados na tabela abaixo:



Estes testes não contempla o código com a paralelização implementada. Vide conclusão.

5. Máquina Utilizada

A máquina utilizada para os testes tem a seguinte configuração (notebook próprio):

Ubuntu 13.04
Memória 7,7 GiB
Processador Intel Core i5-2410m CPU @2,30 GHz x 4
Tipo de sistema 64-bit

6. Conclusão

Neste trabalho foi implementado do problema descrito, para isso tive um contato direto com programação dinâmica e programação em paralelo. Esses módulos são extremamente importante para a criação de soluções com uma eficiência maior por melhor utilização do processamento da máquina e a reutilização de valores já criados anteriormente para criar os próximos valores, tendo assim que realizar menos instruções para chegar a solução exata.

O objetivo do trabalho foi atingido, pois para a resolução deste problema foi utilizado à linguagem de programação C, para criar as estruturas de dados e as funções necessárias para obter os resultados em tempo ótimo, como descrito na analise de complexidade.

Com tudo, considero de grande importância os conhecimentos obtidos com a elaboração desse código. Tendo que utilizar vários módulos estudados anteriormente na disciplina como caminhamento por grafo, além de desenvolver e

solidificar os conhecimentos a cerca do tema do tp, paradigma de programação em paralelo e paradigma de programação dinâmica.

Como dito sobre o estudo e a prática do paradigma de programação em paralelo neste tp, essa parte não foi implementada no código, enfrentei muitos problemas para a criação das threads e passagem de parâmetros da funcao_thread e da struct_thread de parâmetros necessários para essa função. Ao utilizar o método pthread_create e passar como parâmetro o nome da função, este avisava erro pois não pode reconhecer a função como uma função para pthread e dizia que o nome passado por parâmetro não foi declarado. Assim, a implementação do código não contem a paralelização, o código simplesmente roda em serial.

7. Referências

Foi utilizado bastante as referencias da linguagem c, para entender os parâmetros e os retornos de algumas funções como malloc, free entre outros.