Escola Superior de Tecnologias e Gestão de Beja

k-d Trees

Miguel de Campos Rodrigues

5959

18 De Junho de 2012

Índice

Introdução

Este projeto da unidade curricular de Estrutura de dados e Algoritmos debate-se sobre a utilização de métodos de estruturação e organização de dados de um modo mais viável e robusto, assim como dar a entender a o funcionamento dos mesmos. Neste caso descrito neste relatório, estes dados são armazenados e orientados utilizando uma estrutura com base em árvore binária com particionamento de espaço denominado por k-D Tree.

O objetivo deste trabalho nomeia-se pela obtenção de noções e formas de estruturação de dados e suas aplicações; conceção de uma estrutura de árvore binária mais especificamente sobre k-D tree e criação de métodos e funções que habilitam a sua manipulação.

O seguinte relatório descreve de forma detalhada como o algoritmo foi desenvolvido, ao estar estruturado da seguinte forma:

Teoria; Aqui é demonstrado como o algoritmo funciona na sua teoria:

Parte experimental; Neste ponto é descriminado todos os aspetos relativos ao sistema em si, resultados de testes e execuções, inclusive a plataforma no qual foi desenvolvido e em que máquina foi desenvolvido e testado. Este Ponto é subdividido em três subcontextos nomeando por Realização Experimental, Sistema Experimental e Resultados Experimentais.

Seguidamente o Relatório inclui uma conclusão ao qual resume os tópicos apontados neste relatório e uma bibliografia com todas as referências ao qual este projeto se baseou.

Teoria

Uma k-D Tree trata-se nada mais, nada menos que uma árvore binária utilizada para estruturação de dados e particionamento de espaço. Esta estrutura decompõe-se de forma multidimensional, isto é, em que cada nó representa uma posição dentro de um determinado espaço dimensional, independentemente da sua dimensão.

A dimensão deste tipo de árvores é definida através da sua chave que no qual contém K posições em que K é igual à dimensão da árvore no qual está contida. Por exemplo, numa k-D Tree com três dimensões cada nó terá uma chave com três valores (X, Y e Z respetivamente). Como se trata de uma árvore binária cada nó subdivide-se em dois respetivos nós, um à esquerda com um valor menor e outro à sua direita com um valor maior ou igual.

Em cada camada ou "profundidade" os nós são representados de forma independente, de acordo com que dimensão estes se assentam, isto é, se a árvore contém três dimensões e o nó for a raiz da árvore então a sua profundidade será '1' (Um) e a sua dimensão será X (1D). Caso o nó esteja adjunto à raiz ou posterior este assenta-se sobre dimensões superiores com Y (2D) ou inclusive Z (3D), voltando de novo à primeira dimensão, sendo sempre assim alternado. Ver ilustração 1

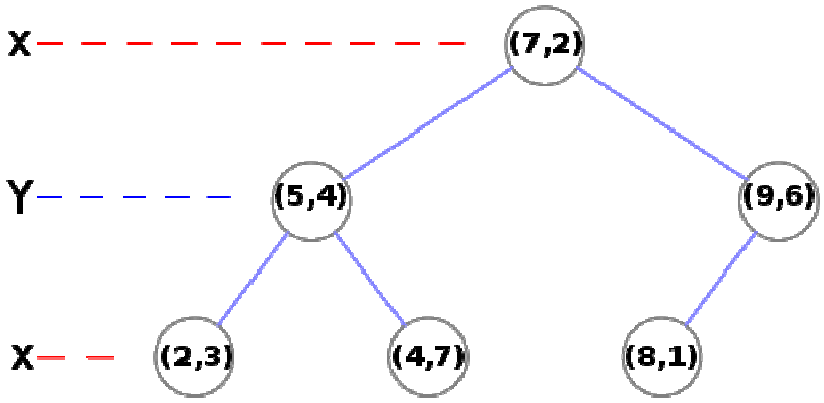

Ilustração 1[1]

Para identificar qual é o número da chave a avaliar o valor é aquele que se encontra na posição K em que K é o número da dimensão em que o nó se assenta. Por exemplo, na ilustração anterior, no nó (5, 4) utiliza-se o 4 para verificar qual o valor que fica adjunto a esta mesma. Como 3 do nó (2, 3) é menor que 4 então vai para a esquerda. Consequentemente, 7 do nó (4, 7) é maior que 4 e como tal vai para a direita.

[1] Fonte: Wikipédia.org - http://upload.wikimedia.org/wikipedia/commons/thumb/2/25/Tree_0001.svg/500px-Tree_0001.svg.png

Parte Experimental

1. Realização Experimental

Linguagem de programação: Python

Ambiente de desenvolvimento:

- Eclipse IDE + PyDev
- Python 2.7
- IDLE
- Tortoise SVN (Subversion)

Sistema operativo: Microsoft Windows 7 Professional 64 bits

Hardware:

- Notebook Samsung R580
- CPU: Intel Core i5 M430 @ 2.27GHz
- Memória: 4GB DDR2
- NVIDIA GeForce GT 330M

2. Sistema Experimental

O código desta estrutura está subdividido em duas ficheiros, dos quais se identifica:

- main.py
- kDTree.py

No ficheiro main.py encontra-se o código fonte de execução e teste para as funcionalidades que a estrutura permite.

No ficheiro kDTree.py o código fonte é composto por três elementos identificados por duas classes e um objeto, que por sua vez se distinguem por:

- Classe Tree – É a árvore em si. Dentro desta apresenta-se a raiz e a sua dimensão. Esta classe é composta pelos seguintes métodos:
    - __init__(self, root, k)
        - (Construtor) Inicializa uma nova instancia de uma árvore binária tipo k-D Tree.
    - insert(self, z)
        - Insere um nó na árvore. A sua posição é aplicada automaticamente, dependendo do peso da sua chave.
    - nearestSearch(self, k = [], x=None)
        - Procura um nó utilizando o método de pesquisa do elemento mais próximo (NN - nearest-neighbor search).
    - pitagorasTheorem(self, point1, point2)
        - Calcula a diagonal entre dois pontos multidimensionais.
    - minimum(self, x=None)
        - Devolve o nó mais à esquerda (menor) na árvore.
    - maximum(self, x=None)
        - Devolve o nó mais à direita (maior) na árvore.
    - sucessor(self, x)
        - Devolve o nó sucessor relativamente à sua posição.
    - inorder_walk(self, x=None, nodeList = [])
        - Cria uma lista com todas os nós ordenados de forma sequencial.
    - transplant(self, u, v)
        - Transplanta (troca) um nó para outra posição na árvore.
    - delete(self, z)
        - Remove um nó da árvore.
    - sort(self, nodeToSort, parentNode = None, depth=0)
        - Reordena a árvore de forma a ficar balanceada.
    - __getMedian(self, nodeList, depth)
        - (privado) Obtém o nó mediano (o mais a meio) da lista.
    - getDim(self, depth)
        - Devolve a dimensão do nó em relação à profundidade entre o mesmo sua raiz da árvore.
    - self.root – O nó de raiz da árvore.

O projeto inclui também os seguintes elementos:

- Classe Node – Representa um nó na árvore. Nesta apresenta uma chave composta por K posições, valor, profundidade, nó subjacente ou "pai" e dois nós adjacentes ou "filhos", sequentemente da esquerda (nó menor) e da direita (nó maior).
    - __init__(self, key=[0], value="", parent=None, left=None, \ right=None, k=IND_D, depth=IND_D)
        - Inicializa uma nova instância de um nó de uma k-D Tree.
    - def __str__(*self*)
        - *Converte o nó para uma representação em texto.*
    - def compareKeys(self, keyToCompare)
        - Compara duas chaves na dimensão da chave a ser comparada (instanciada na própria classe).

- Objeto "NIL" do tipo Node – Representa um nó vazio, indicando assim o limite da árvore e as suas "folhas" ou nós de limite.

3. Resultados Experimentais

Neste trabalho utilizei um método de teste linear (em sequencia) para poder avaliar e testar os resultados do código, no qual compus uma sequência de operações sobre a estrutura, no qual se apresentam, a criação de uma árvore a partir de uma lista de dados, pesquisa de 4 chaves (3 válidas e uma inválida), verificação de nós extremos (máximo e mínimo), nó sucessor, listagem ordenada (in order walk) da árvore, a transplantação e remoção de nós, balanceamento / reordenação da árvore e procura do valor mais próximo (Nearest-Neighbor search).

Tempos de execução (sem depuração ativada):

Criar Estrutura: 0.00140 segundos

Procurar (três) nós: 0.00059 segundos

Procurar extremos: 0.00013 segundos

Achar nó sucessor à raiz e aos extremos: 0.00023 segundos

Listar Nós: 0.00066 segundos

Transplante: 0.00068 segundos

Remoção: 0.00074 segundos

Balancear árvore: 0.00146 segundos

Procurar por mais próximo: 0.00048 segundos

Relativamente a estes dados obtido pode-se reparar que a criação da estrutura e o rebalanceamento da árvore requerem mais recurso da máquina pois ser um processo com uma certa complexidade, quer aritmética, quer de lógica.

Conclusões

Neste trabalho conclui que se uma k-d Tree trata-se de uma árvore binária de estrutura bastante interessante, quer a nível de complexidade, quer a nível de usabilidade, visto que por exemplos que investiguei contém potencial em várias áreas, como pro exemplo, em deteção de colisões em ambientes gráficos avançados e robótica.

Neste trabalho investiguei como esta estrutura funciona e, se viabilidade for certa, futuramente esta estrutura será utilizada para potenciais projetos.

Bibliografia

Wikipédia.org - k-d Tree – http://en.wikipedia.org/wiki/K-d_tree

Idots.org - http://ldots.org/kdtree

Introduction to Algorithms, Third Edition, "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein", The MIT Press

Youtube.com - http://youtu.be/kcIokRndG4A, “susanmhaynes”