SEMINÁRIO DE ESTRUTURA DE DADOS II

ESTRUTURA ESPACIAL GRID

GRUPO:

ANANDA MENDES SOUZA - 19.1.4030

ALESSANDRO D'ANGELO - 18.2.4054

LAURA MARTINS DA COSTA COURA MARINHO - 19.1.4168

MARIA LAURA MOREIRA RAIMUNDO - 19.1.4144

RAFAEL COELHO MONTE ALTO - 19.1.4111

RÔMULO DE OLIVEIRA CARNEIRO - 19.1.4107

VINÍCIUS COSSIO DE OLIVEIRA - 19.1.4004

SUMÁRIO

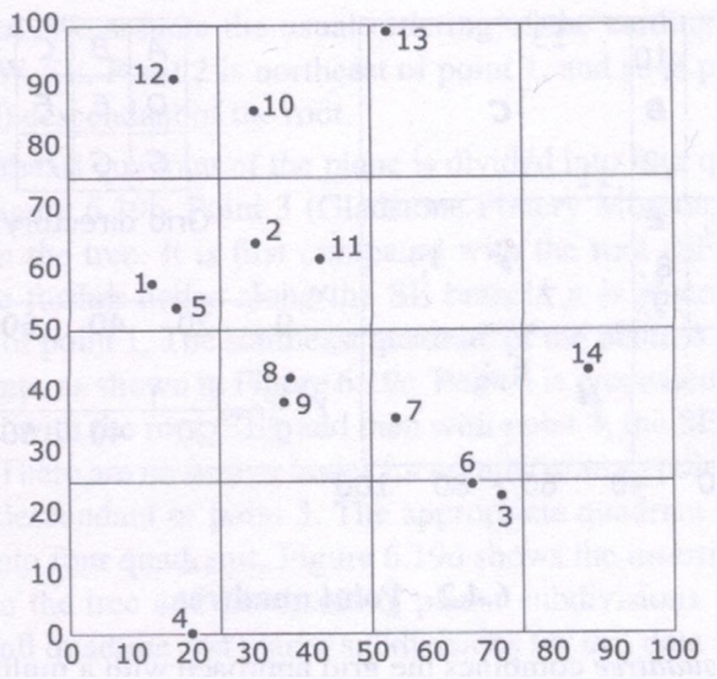
ORIGEM

Há dificuldades de encontrar conteúdo relacionado a Grid, mas, segundo nossa pesquisa, o conceito de *Grid Files* (estrutura baseada em árvores de busca multidimensional, tal como *quad trees*, *b trees* e *k-d trees,* mas que será apresentado em mais detalhes posteriormente) foi introduzido em 1984 por Jürg Nievergelt, Hans Hinterberger e Kenneth C. Sevcik em seu trabalho "*The grid file: an adaptable, symmetric multikey file structure*". Portanto, a partir deles, outros pesquisadores desenvolveram seus próprios algoritmos e trabalhos, como, por exemplo, Klaus Helmer Hinrichs, que em 1985 implementou um sistema de *Grid Files*.

O QUE É?

O *Grid* é uma estrutura que que particiona o espaço de trabalho em uma "grade" de células de mesmo tamanho, sendo implementado, normalmente, através de *arrays* ou listas encadeadas. Suas células são referidas como *buckets*, sendo responsáveis pela representação da localização física em que os itens podem ser recuperadas.

Além disso, o *Grid* é organizado de forma que os pontos próximos fiquem em um mesmo *bucket*. Por exemplo, na imagem ao lado, o ponto caiu nos limites de um *bucket*. Ele é então inserido no mesmo *bucket* que o ponto 7, já que, por convenção, os *buckets* possuem pontos nos limites sul e oeste.



Em relação ao tamanho ideal de *bucket*, o cálculo é feito com base no número de pontos (quanto mais pontos, mais *buckets*) e na magnitude da faixa média das range query suportadas pelo sistema.

Também tem-se que os algoritmos que utilizam da estrutura *Grid* podem ser divididos com base nas seguintes características:

- A estrutura de dados utilizada para representar o Grid.

- O tamanho da célula.

- E se um mesmo elemento pode ser mapeado por mais de uma célula.

PARA QUE SERVE?

Grid é útil para solucionar dois tipos de problemas:

- **Point Query:** encontrar um ponto que atende uma condição específica.

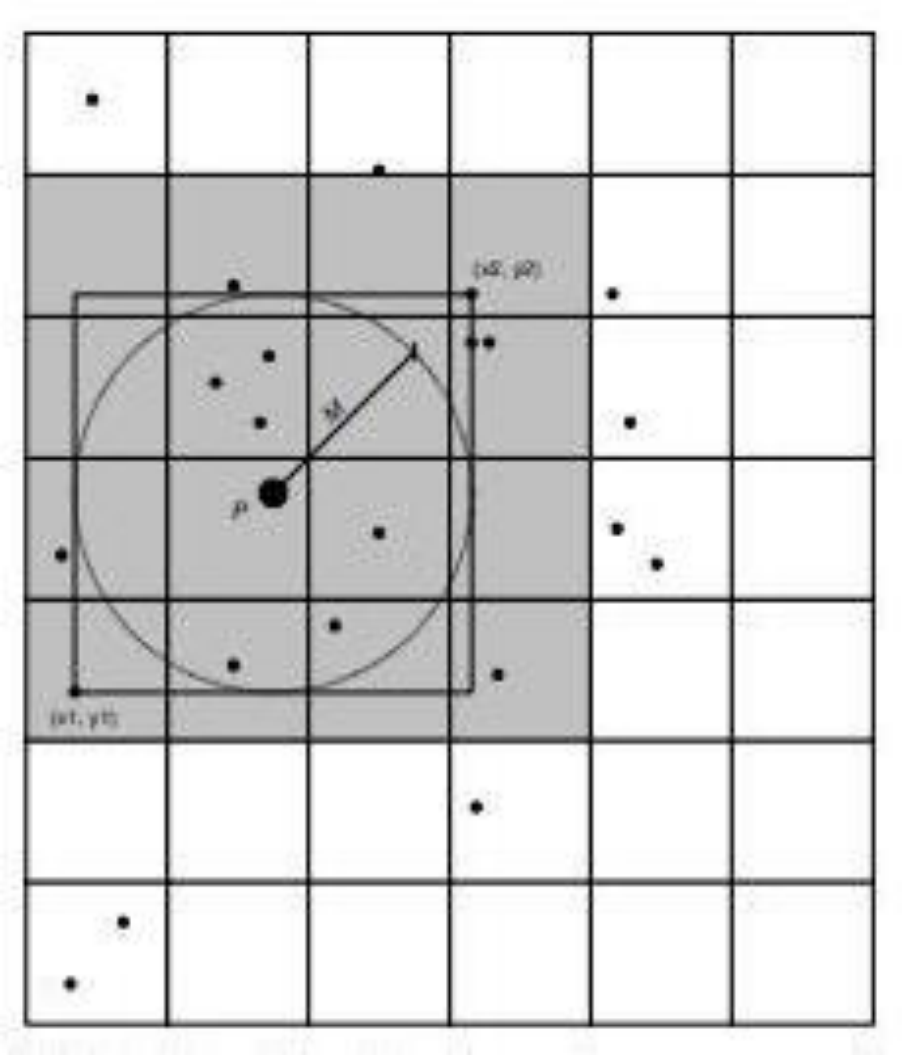
- **Range Query:** encontrar um conjunto de pontos que se encaixam em um determinado range.

Já um exemplo de utilização da estrutura *Grid* é para localizar restaurantes num raio de 50m:

Para isto, basta calcular as coordenadas de dois pontos auxiliares, um deles subtraindo 50 de ambas as coordenadas, e no outro somando:

P1(x1, y1): x1 = xP - 50 e y1 = yP – 50

P2(x2, y2): x2 = xP + 50 e y2 = yP + 50



Depois, determina-se qual é a linha e a coluna dos quadrados que contêm os pontos P1 e P2 e todos os pontos procurados estarão nos quadrados compreendidos entre os limites expressos pelas linhas e colunas encontradas.

Naturalmente, deve-se calcular a distância de cada ponto encontrado a P, para verificar o atendimento à restrição de distância, pois a pesquisa na realidade foi feita usando um quadrilátero, e não com um círculo.

ESTRUTURA DE DADOS

```c
#define TAM_GRID 5

typedef struct {
    double x;
    double y;
    // Outros atributos
    // ...
} Item;

typedef struct {
    Item *fila;
} Bucket;

typedef struct {
    Bucket grid[TAM_GRID][TAM_GRID];
} Grid;
```

INSERÇÃO

```
void adicionarItem(Grid *g, Item *i) {
    // Encontrar posição de inserção
    int insertX, insertY;
    acharPosicao(i, &insertX, &insertY);

    // Adicionar no bucket da posição insertX, insertY
    adicionarFila(&g->grid[insertX][insertY], i);
}


void acharPosicao(Item i, int *posicaoX, int *posicaoY) {
    // Neste caso, usamos o valor inteiro arredondado para baixo
    *posicaoX = floor(i.x);
    *posicaoY = floor(i.y);
}
```

REMOÇÃO

```
int removerItem(Grid *g, Item *i) {
    if(!pesquisa(*g, i)){
        printf("Ponto nao encontrado\n");
        return 0;
    }

    // Encontrar posição de remoção
    int posicaoX, posicaoY;
    acharPosicao(i, &posicaoX, &posicaoY);

    // Remover no bucket da posição
posicaoX, posicaoY
    retirarFila(&g-
>grid[posicaoX][posicaoY], i);

    return 1;
}
```

```
int pesquisa(Grid *g, Item *i) {
    // Achar posição inteira da onde estaria
o item desejado
    int posicaoX, posicaoY;
    acharPosicao(i, &posicaoX, &posicaoY);

    // Procurar dentro do bucket
    int resultado =
pesquisarFila(&g.grid[posicaoX][posicaoY]);

    if(resultado) return 1;
    else return 0;
}
```

VANTAGENS

- Pode ser utilizado para pesquisa de uma única chave.
- Retorna somente os resultados corretos.
- Melhora significativa no tempo de processamento de consulta de múltiplas chaves.

DESVANTAGENS

- Cálculo do tamanho ideal do bucket (hash).
- Limitação de utilização: apenas pontos ou objetos que caibam inteiramente em quadrados.
- A população de pontos nas células varia muito (overflow, underflow, células vazias).
- Se os pontos forem menos uniformemente distribuídos este problema piora.

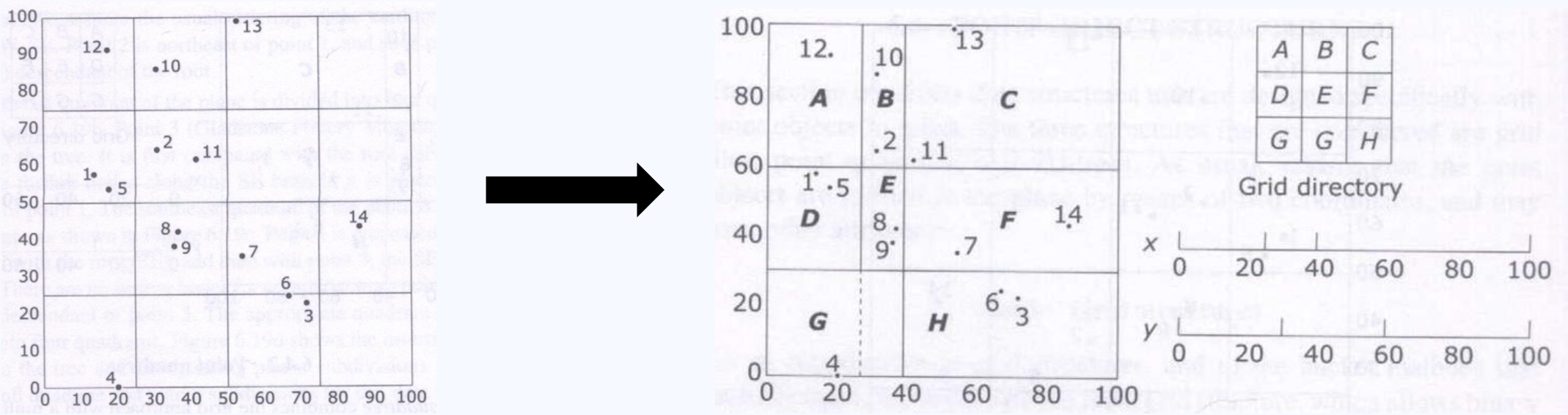
SOLUÇÃO: GRID FILES

GRID FILES

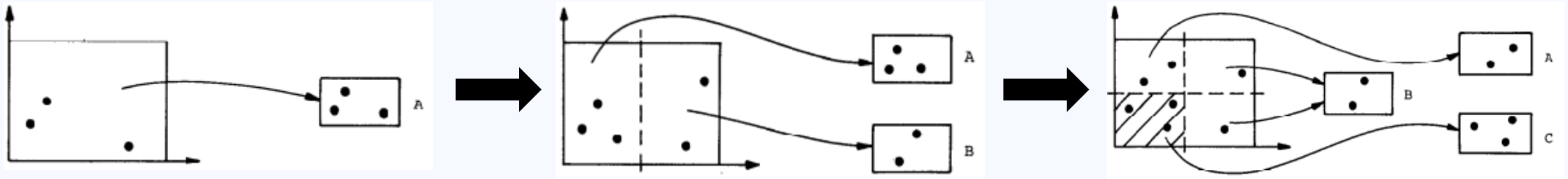
Extensão de grid que permite as linhas de divisão das células terem tamanhos arbitrários, ou seja, não sendo mais restrito que as células tenha todas o mesmo tamanho. Portanto, ao invés de *n buckets*, teremos uma grade que cobre todo o espaço de pesquisa.

Desse modo, a inserção e remoção de elementos dos *buckets* afeta seus tamanhos. Se uma célula está muito vazia, ela pode se juntar com outra, ou, se está muito cheia, pode se dividir em duas. Ademais, destaca-se que os Grid Files utilizam também uma estrutura (normalmente um vetor de 2 dimensões) para armazenar os endereços dos *buckets*, sendo esta chamada de **grid directory**.

Também, tal como Hinrichs descreveu em seu trabalho "*The grid file system implementation and case studies of applications*", objetos espaciais podem ser transformados em pontos e, então, armazenados nos *Grid Files*. Assim, essa extensão permite armazenar informações geométricas, resolvendo um dos outros problemas citados no *Grid*.



No exemplo acima, tem-se que os 16 *buckets* anteriores de mesmo tamanho foram transformados em 8 *buckets* de tamanhos distintos. Já no exemplo abaixo, observa-se uma situação similar, onde os *buckets* com poucos pontos se juntam e os com muitos são divididos.



PERGUNTAS?

BIBLIOGRAFIA

LINS, Bruno Normande. **Avaliação de desempenho de algoritmos paralelos de busca de vizinhos em cenários com distribuições espaciais distintas.** 2016.

NIEVERGELT, Jürg; HINTERBERGER, Hans; SEVCIK, Kenneth C. **The grid file: an adaptable, symmetric multikey file structure.** 1984.

HINRICHS, Klaus Helmer. **The grid file system implementation and case studies of applications.** 1985.

BAPTISTA, Cláudio. **Sistemas de Informações Geográficas.** 2010.

DAVIS JUNIOR, Clodoveu Augusto; QUEIROZ, Gilberto Ribeiro de. **Métodos de acesso para dados espaciais.** 2005.

Grids and Grid Files - Basic Concepts. Disponível em: <https://www.unidata.ucar.edu/software/mcidas/current/mclearn/grids-1.html>. Acesso em 15 de abril de 2021.

FERREIRA, Artur, et al. **Estrutura de Dados Espaciais: GRID**. Disponível em: <https://prezi.com/1wfvudsxrein/grid>. Acesso em 15 de abril de 2021.