Corso

di

Basi di Dati Spaziali

La progettazione fisica: gli indici spaziali - 1

Angelo Montanari

Donatella Gubiani

Assunzioni fondamentali

- La dimensione della collezione dei dati è molto maggiore dello spazio disponibile in memoria principale
- Tempo di accesso in memoria secondaria (disco) è molto maggiore del tempo di accesso in memoria principale
- Gli oggetti sono raggruppati in pagine/blocchi, che costituiscono l'unità di trasferimento tra memoria principale e disco

Le strutture di indicizzazione

- I dati di una base di dati sono memorizzati in file come *insiemi di record*

- Gli *indici* sono strutture dati utilizzate per rendere più efficiente il recupero dei record in presenza di specifiche condizioni di ricerca

Trattamento dei dati spaziali-1

- Tipici esempi di interrogazioni spaziali:
  - point e window query (ricerca di oggetti la cui geometria contiene un dato punto o si sovrappone ad un dato rettangolo)
  - join spaziali (dati due insiemi di oggetti, restituisce le coppie di oggetti che soddisfano una data relazione spaziale)
- L'esecuzione delle interrogazioni spaziali richiede l'esecuzione di operazioni geometriche complesse e costose e/o coinvolge grandi quantità di dati in memoria secondaria

Trattamento dei dati spaziali-2

- Le strutture d'accesso sviluppate per indicizzare i dati numerici o alfanumerici tradizionali si basano sull'*ordinamento totale* del dominio del campo chiave; ne segue che non si adattano ai dati spaziali
- Occorrono *nuove strutture di indicizzazione* (dati spaziali 2D e 3D). Ci concentreremo sul caso 2D
- Desiderata. Una relazione d'ordine per oggetti geometrici bidimensionali deve preservare la **prossimità degli oggetti**: oggetti vicini nel piano devono risultare vicini anche nell'indice.

  Una possibilità: inclusione di rettangoli

Campo di indicizzazione

- Il campo di indicizzazione degli indici spaziali è legato alle caratteristiche spaziali degli oggetti e/o fenomeni
- Diversi approcci:
  - attraverso strutture d'accesso tradizionali
  - mediante nuove strutture di indicizzazione

Indicizzazione dei dati spaziali: strutture tradizionali - 1

- Le strutture tradizionali possono essere direttamente utilizzate nel contesto spaziale

Esempio. E' possibile rappresentare il baricentro dei diversi poligoni (attraverso la concatenazione delle loro coordinate) e utilizzare il valore così ottenuto come chiave di ricerca su strutture tradizionali

Indicizzazione dei dati spaziali: strutture tradizionali - 2

- Uso diretto di indici tradizionali: poco interessante
- Vedremo, invece, una struttura dati molto utilizzata (i quadtree lineari) che usa i B^+-alberi dei DBMS tradizionali come indice per gestire opportune informazioni spaziali

Indicizzazione dei dati spaziali: nuove strutture

- Le strutture di indicizzazione per dati spaziali sono riconducibili a due categorie
  - strutture basate sullo spazio (**space-driven**): lo spazio di riferimento è suddiviso in celle rettangolari, indipendentemente dalla distribuzione degli oggetti. Gli oggetti sono associati alle celle in base a qualche criterio geometrico
  - strutture basate sui dati (**data-driven**): le strutture sono organizzate partizionando l'insieme degli oggetti, tenendo conto della loro distribuzione nello spazio di riferimento

Tipologie di indici

- Indici space-driven
  - grid file (fixed grid, grid file)
  - strutture lineari (quadtree, space-filling curve, linear quadtree, z-ordering tree)

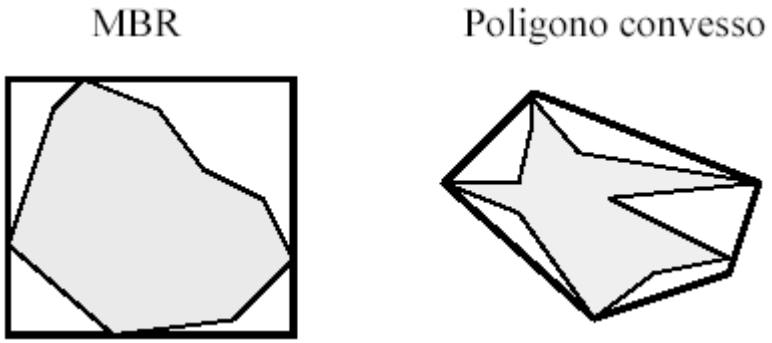
- Indici data-driven
  - R-alberi
  - R*-alberi
  - R^+-alberi

Indici spaziali in Oracle Spatial

- Oracle Spatial mette a disposizione due strutture di indicizzazione:
  - Quad-Tree (space-driven)
  - R-tree (data-driven)

Minimal Bounding Box

- Nel caso di linee e poligoni, viene usualmente utilizzata una qualche approssimazione della geometria degli oggetti: la nozione di Minimal Bounding Box (MBB)
- MBB è un descrittore che approssima il dato spaziale con un rettangolo (MBR) o con un poligono convesso



Le entry degli indici spaziali

- Vantaggi dell'uso di MBB:
  - si evita di dover valutare predicati geometrici complessi durante l'esplorazione dell'indice
  - si possono utilizzare entry di dimensione costante

- Un indice spaziale è costituito da una collezione di entry del tipo [MBB, OID].

  L'OID consente di accedere direttamente alla pagina che contiene la rappresentazione fisica dell'oggetto (attributi descrittivi e componente spaziale)

Utilizzo di MBB

- Un'operazione che coinvolge un predicato spaziale che agisce su oggetti indicizzati attraverso l'uso di MBB viene eseguita in 2 passi:
  - FILTER STEP (vengono selezionati gli oggetti il cui MBB soddisfa il predicato spaziale)
  - REFINENT STEP (il predicato spaziale viene valutato sulla geometria effettiva degli oggetti selezionati dal passo precedente)

Nel seguito ci concentreremo sul passo fondamentale: il filter step

Operazioni

- **Costruzione dell'indice**: inserimento di una entry o di una collezione di entry

- **Operazioni di ricerca**: point e window query. Assumeremo che lo spazio di ricerca sia un sottoinsieme rettangolare del piano 2D con i lati paralleli agli assi x e y

Complessità

- **Complessità temporale:** numero di operazioni di I/O (point e window query in tempo sublineare)

- **Complessità spaziale:** dimensione dell'indice (comparabile con quella della collezione dei dati: se la collezione dei dati occupa n pagine, la dimensione dell'indice dovrebbe essere dell'ordine di n)

Dinamicità

- L'indice dovrebbe gestire l'inserimento di nuovi oggetti e la cancellazione di oggetti esistenti, adattandosi alla crescita e alla riduzione della collezione di oggetti senza peggiorare le sue prestazioni (*dinamicità*)
- E' difficile ottenere strutture robuste che operino bene con qualunque distribuzione statistica dei dati
- In *media*, le prestazioni degli indici sono ottimali (numero logaritmico di operazioni di I/O). Ciò non è garantito nel *caso peggiore* (a differenza di quanto accade coi B-alberi, l'ottimalità non è garantita nel caso peggiore)

Indici space-driven

- **Fixed grid**: struttura proposta per indicizzare insiemi di punti

- **Grid file**: struttura proposta per indicizzare oggetti rispetto al valore di più attributi (indice multichiave che supporta interrogazioni che coinvolgono una qualunque combinazione di tali attributi)

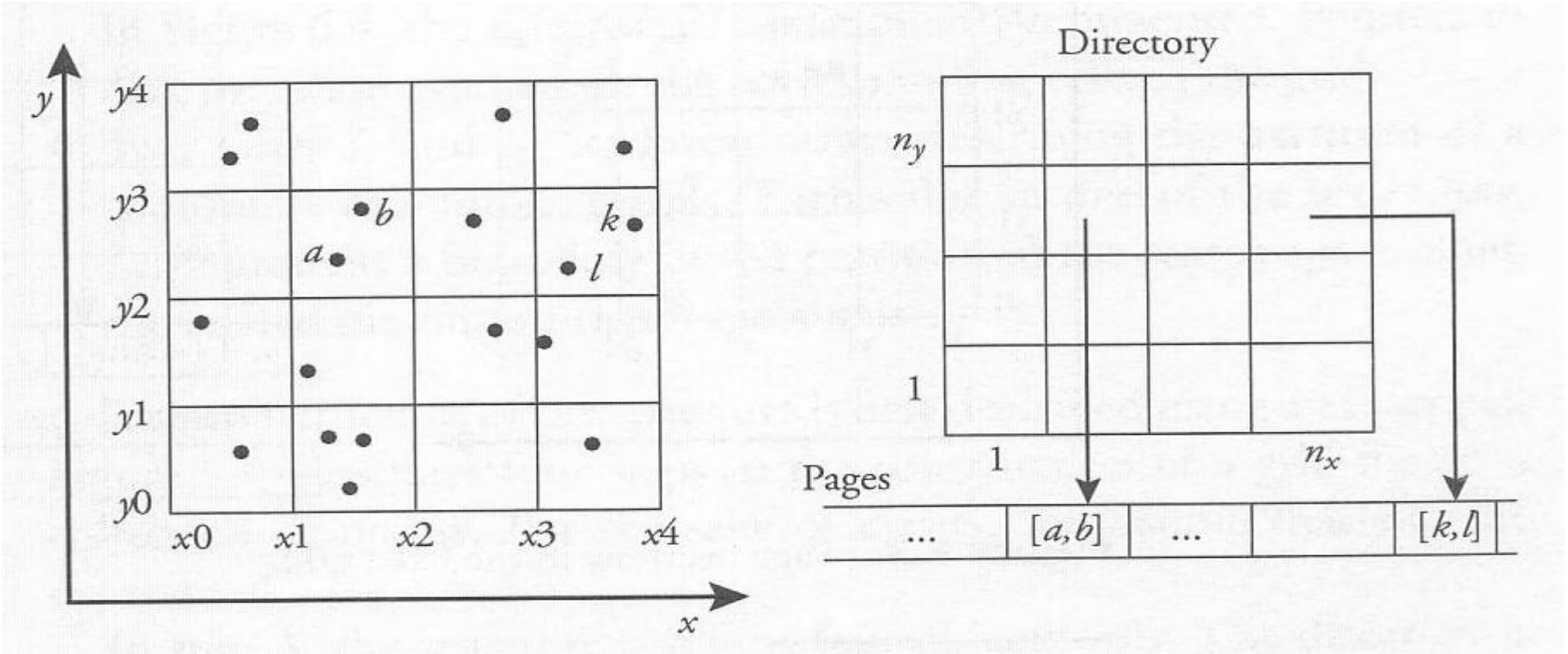
- **Strutture lineari**:usate nelle estensioni spaziali dei DBMS relazionali (e.g., Oracle Spatial); si integrano facilmente con i B^+-alberi

Fixed Grid: le caratteristiche

- Lo **spazio di ricerca** è scomposto in N_x x N_y celle rettangolari di eguale dimensione
- Ad ogni cella $c_{i,j}$ corrisponde una pagina di memoria secondaria
- Ogni punto è assegnato alla cella che lo contiene.

- L'indice utilizza una matrice DIR[1.. N_x, 1.. N_y] come **directory**.
- Ogni elemento DIR[i,j] contiene l'indirizzo PageID della pagina che memorizza i punti assegnati alla cella $c_{i,j}$
- Se [S_x, S_y] è la dimensione dello spazio di ricerca, ogni cella rettangolare ha dimensione [S_x/N_x, S_y/N_y]

Fixed Grid: la struttura



La **struttura** del fixed grid

Fixed Grid: gli algoritmi

- **Inserimento di P(a,b):** si calcolano $i = (a-x_0)/(S_x/N_x) + 1$ e $j = (b-y_0)/(S_y/N_y) + 1$ e si inserisce il punto P nella pagina DIR[i,j].PageID

- **Point Query**: dato un punto P(a,b), si trova la pagina come nel caso dell'inserimento e si verifica se contiene P

- **Window Query**: si calcola l'insieme S delle celle c tali che c.rect si sovrappone alla finestra di input. Per ogni cella $c_{i,j}$ in S, si determinano i punti appartenenti alla pagina DIR[i,j].PageID contenuti nella finestra di input

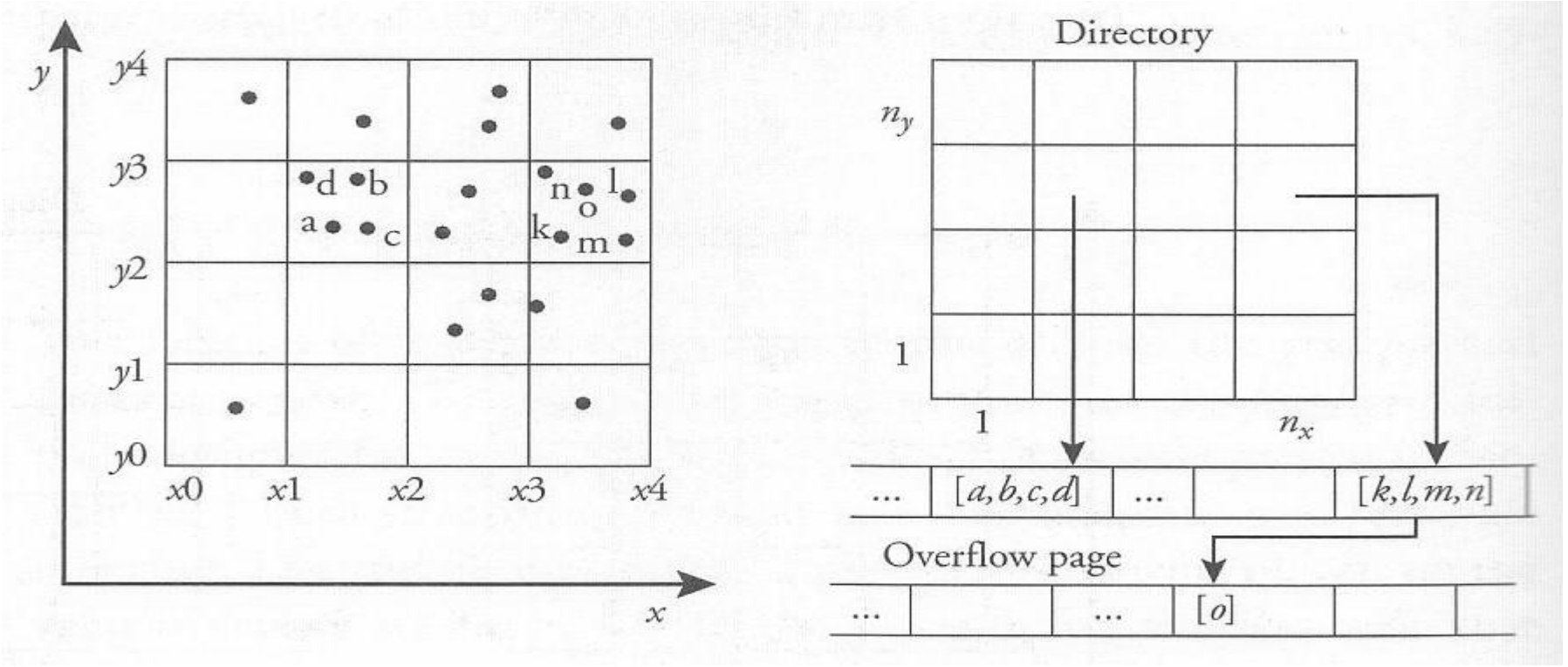
Fixed Grid: complessità

- **Point Query**: efficiente (se la directory risiede in memoria principale, richiede una singola operazione di I/O)
- **Window Query**: numero di operazioni di I/O dipende dal numero di celle che si sovrappongono alla finestra di input (proporzionale alla dimensione della finestra)

- La risoluzione della griglia dipende dal numero N di punti da indicizzare (se una pagina ha una capacità di M punti, si crea una griglia con almeno N/M celle). Ogni cella contiene mediamente meno di M oggetti. Una cella cui sono assegnati più di M punti va in **overflow**

Fixed Grid: pagine di overflow

- **Gestione degli overflow**: si utilizzano pagine di overflow. Una point query può richiedere fino a q operazioni di I/O se il punto è memorizzato nel q-esima pagina di una catena di overflow (caso peggiore: lista lineare di pagine)

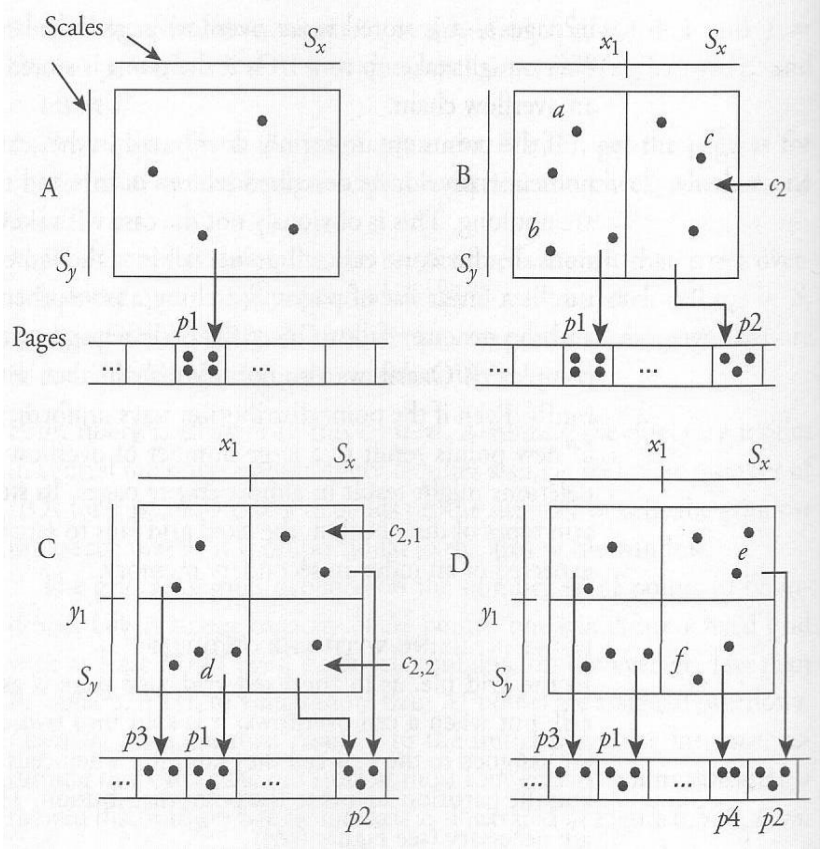


Grid File: i punti

- Una pagina è ancora associata ad una singola cella, ma quando si verifica un overflow la cella viene spezzata in due (**split**) e i punti assegnati alla nuova cella che li contiene
- Le **celle** possono essere di **dimensioni differenti**; la suddivisione dello spazio si adatta alla distribuzione dei punti
- Vengono utilizzate 3 strutture dati:
  - una **directory** DIR (matrice che contiene gli indirizzi delle pagine associate alle celle; due **celle adiacenti** possono indirizzare la **stessa pagina**)
  - due **scale** S_x e S_y (vettori che descrivono la suddivisione degli assi x e y in intervalli)

Grid File: inserimento - 1

- Un esempio di costruzione della struttura assumendo che la capacità di una pagina sia 4



Grid File: inserimento - 2

- Inserimento di un punto, tre possibili casi:
  - NO SPLIT - punto appartiene a una pagina non piena
  - SPLIT delle CELLE/PAGINE, ma non della DIRECTORY - il punto P da inserire cade nella cella c e la pagina p indirizzata da c è piena, ma p è indirizzata più celle. Viene allocata una nuova pagina p' che viene assegnata a c. Gli oggetti in p contenuti in c.rect, assieme al punto P, vengono inseriti in p'. L'entry di DIR corrispondente a c viene aggiornata con p'
  - SPLIT delle CELLE/PAGINE e della DIRECTORY - la pagina p indirizzata dalla cella $c_{i,j}$, in cui cade il punto da inserire P, è piena e non ci sono altre celle che indirizzano p. Il rettangolo $c_{i,j}$.rect va spezzato lungo l'asse x o l'asse y

Grid File: inserimento - 3

- SPLIT delle CELLE/PAGINE e della DIRECTORY (continua) -

  Supponiamo che venga spezzato rispetto a x. La proiezione della cella su S_x viene spezzata in 2 intervalli (nel caso più semplice, di eguale dimensione) e un nuovo valore viene inserito in S_x di rango i+1. La vecchia cella $c_{i,j}$ dà origine a due nuove celle $c_{i,j}$ e $c_{i+1,j}$. Viene creata una nuova pagina p' assegnata a $c_{i+1,j}$. I punti della pagina piena vanno distribuiti tra p (punti che cadono in $c_{i,j}$.rect) e p' (punti che cadono in $c_{i+1,j}$.rect).

  Tutte le precedenti colonne di rango l, con l>i, diventano di rango l+1 e viene creata una nuova colonna di rango i+1. Per ogni k diverso da j, alla nuova cella $c_{i+1,k}$ è assegnata la stessa pagina di $c_{i+1,k}$

Grid File: point query

- Ogni point query si può eseguire con 2 operazioni di I/O (una per accedere alla pagina p indirizzata dalla directory e una per accedere alla pagina del file dei dati associata ad una delle entry in p)
- La struttura è dinamica e normalmente utilizza lo spazio in modo ragionevole

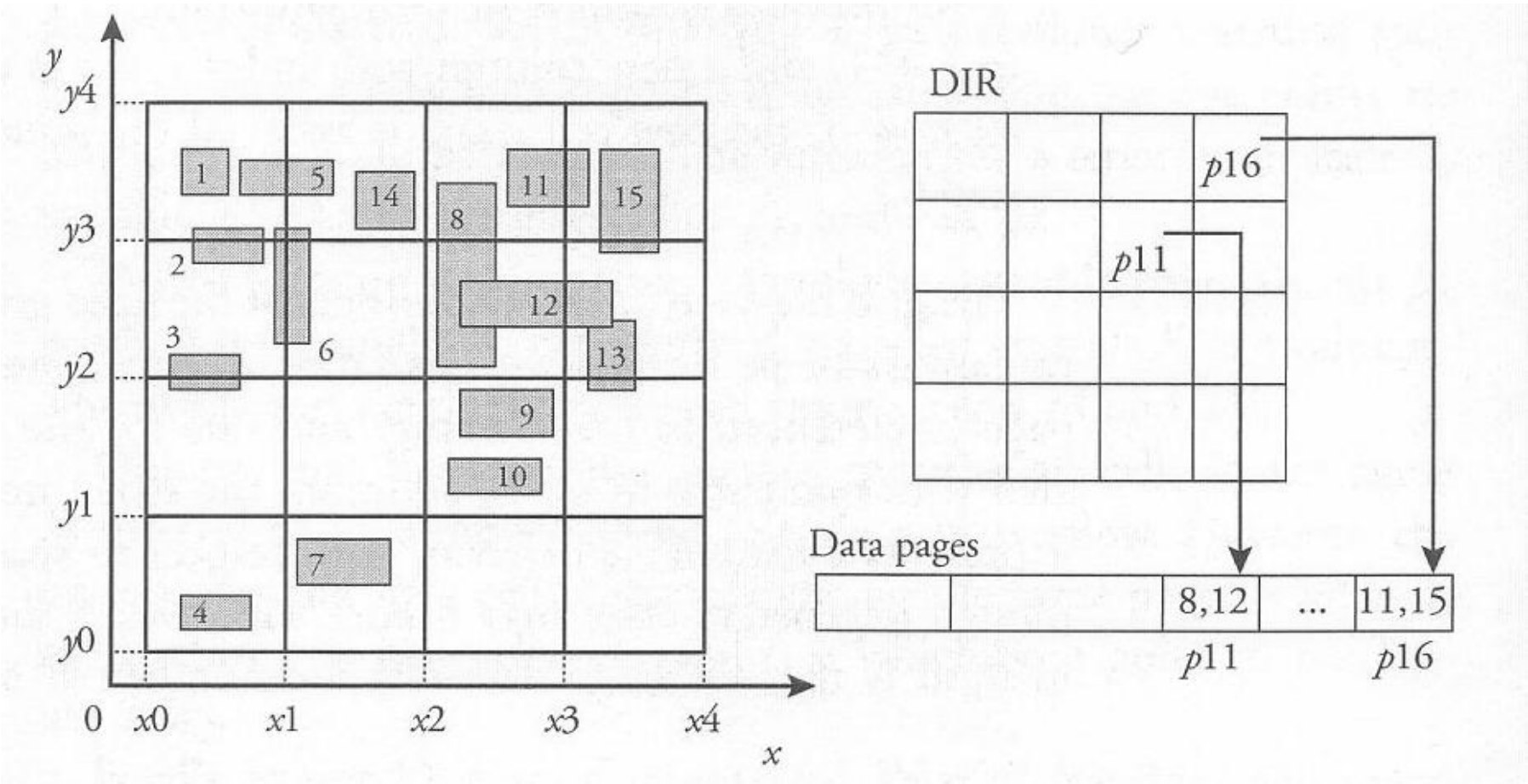
- **Limite**: per grandi insiemi di dati il numero di celle nella DIRECTORY cresce a tal punto che la essa non può più essere interamente contenuta in memoria principale

Il caso dei rettangoli (MBR)

- Criterio più semplice: assegnare un rettangolo alle celle cui si sovrappone
- Tre casi:
  - il rettangolo contiene la cella
  - il rettangolo e la cella si intersecano
  - il rettangolo è contenuto nella cella

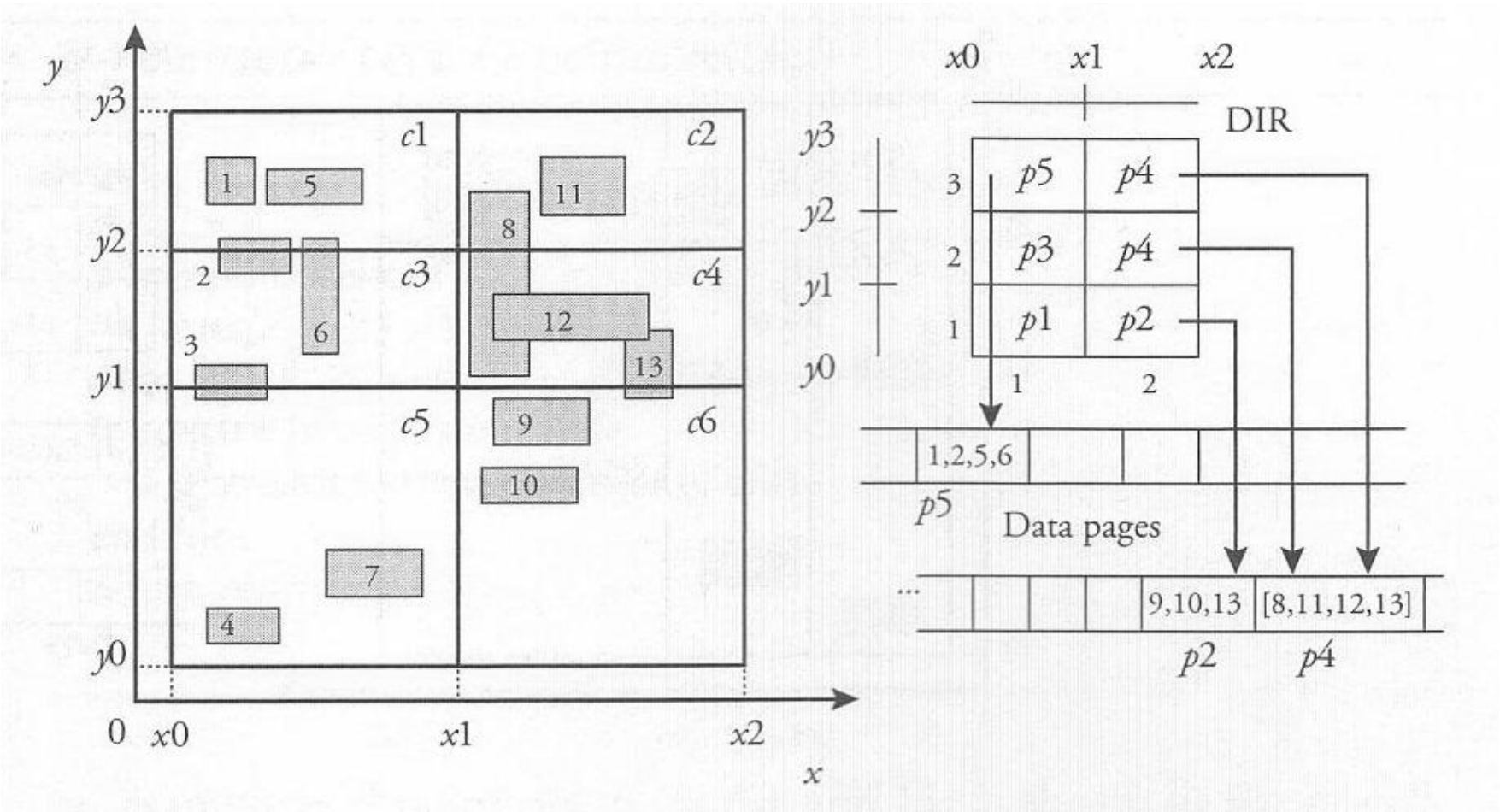
Nei primi due casi il rettangolo è assegnato a più celle, l'indice cresce di dimensioni e le prestazioni peggiorano

Fixed Grid e MBR



La soluzione con la fixed grid (capacità 4)

Grid File e MBR



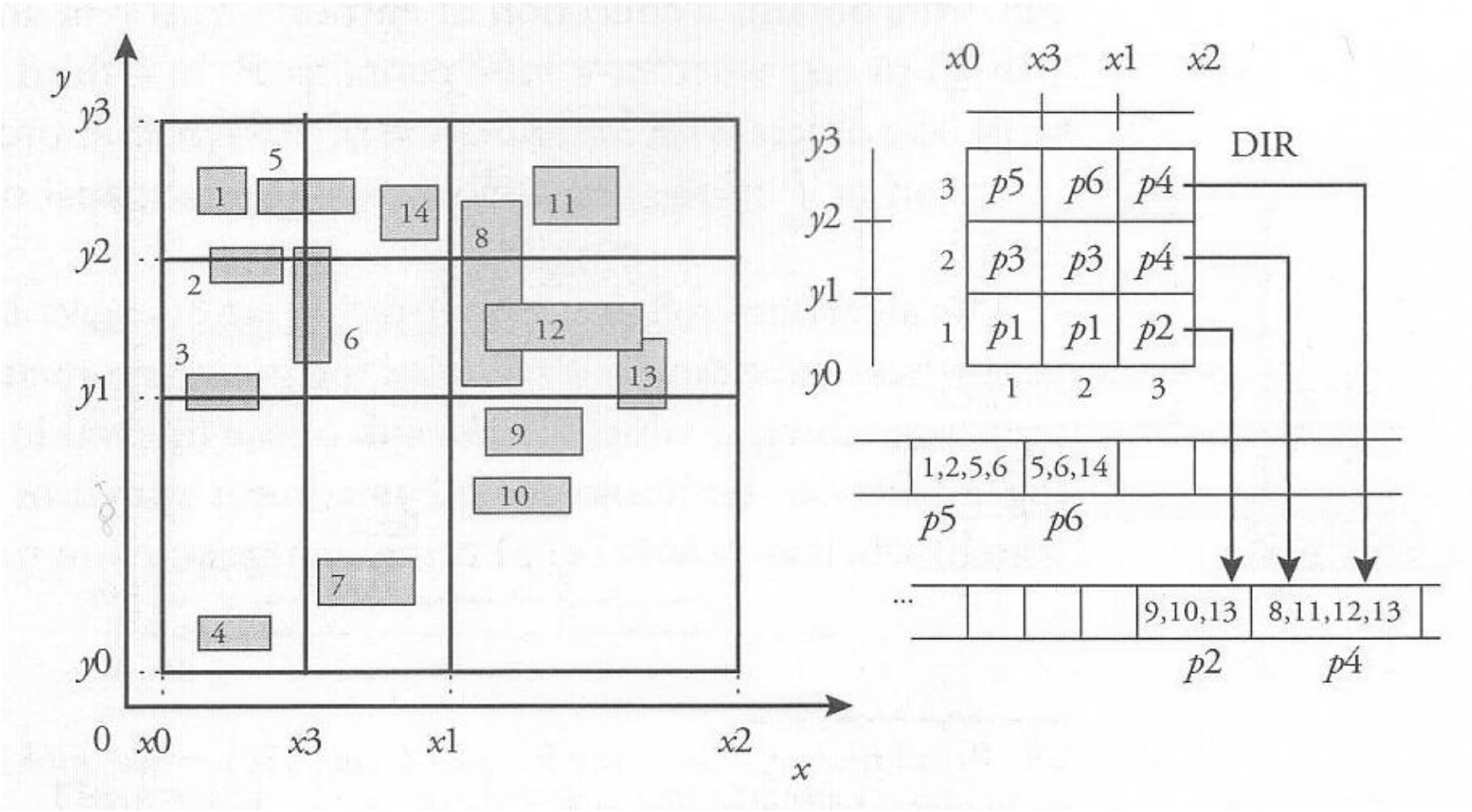
La soluzione con il grid file (capacità 4)

Inserimenti e cancellazioni

- Inserimenti e cancellazioni sono gestiti come nel caso dei punti. A causa della duplicazione degli oggetti nelle celle vicine, aumenta la frequenza degli split

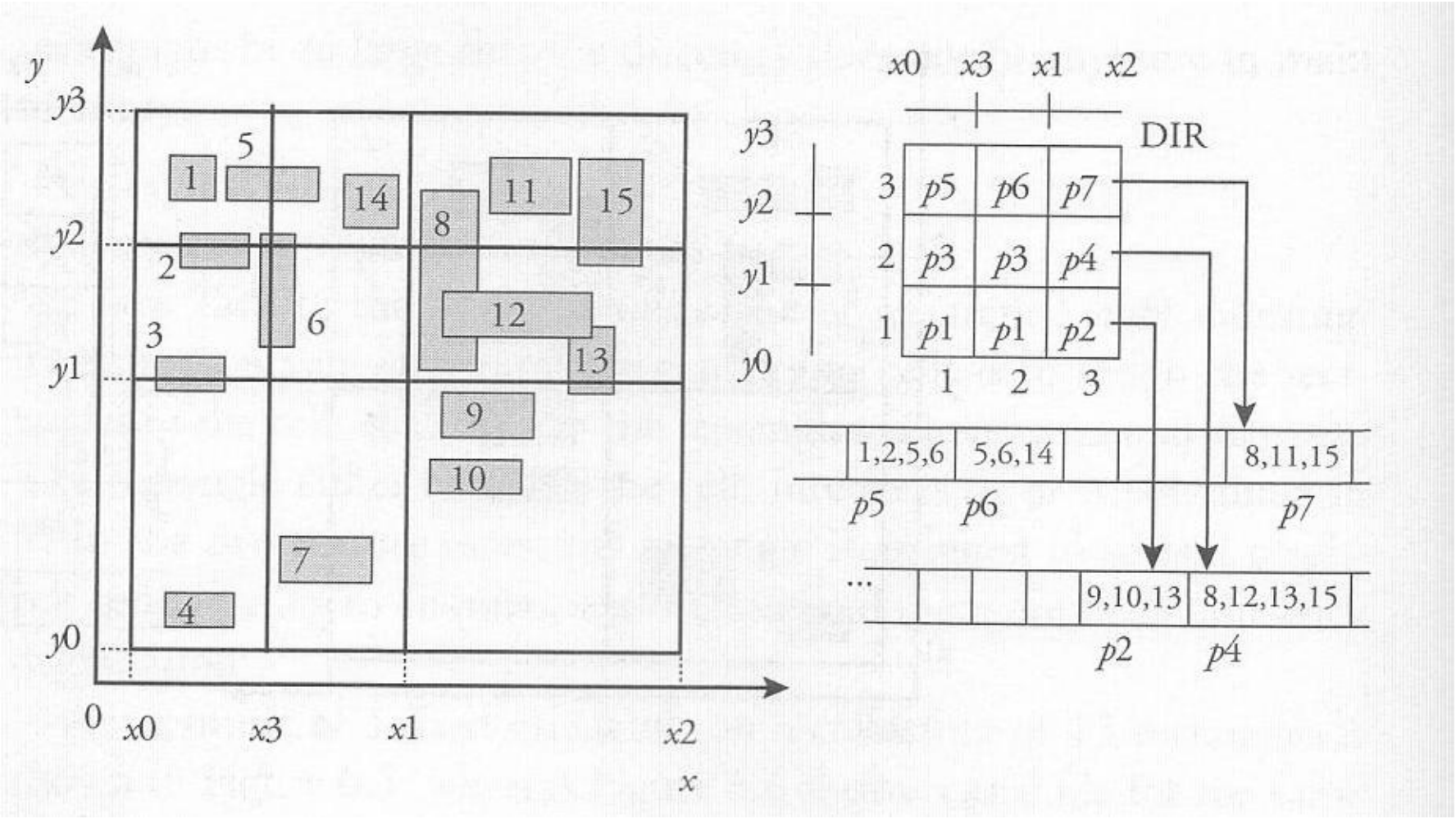
- Con riferimento alla figura precedente, si possono dare più casi in cui è necessario uno split delle celle illustrati graficamente nel seguito

Inserimenti - 1



Inserimento dell'oggetto 14
(oggetti 5 e 6 duplicati in p5 e p6!)

Inserimenti - 2



Inserimento dell'oggetto 15
(split di celle/pagine, non della directory)

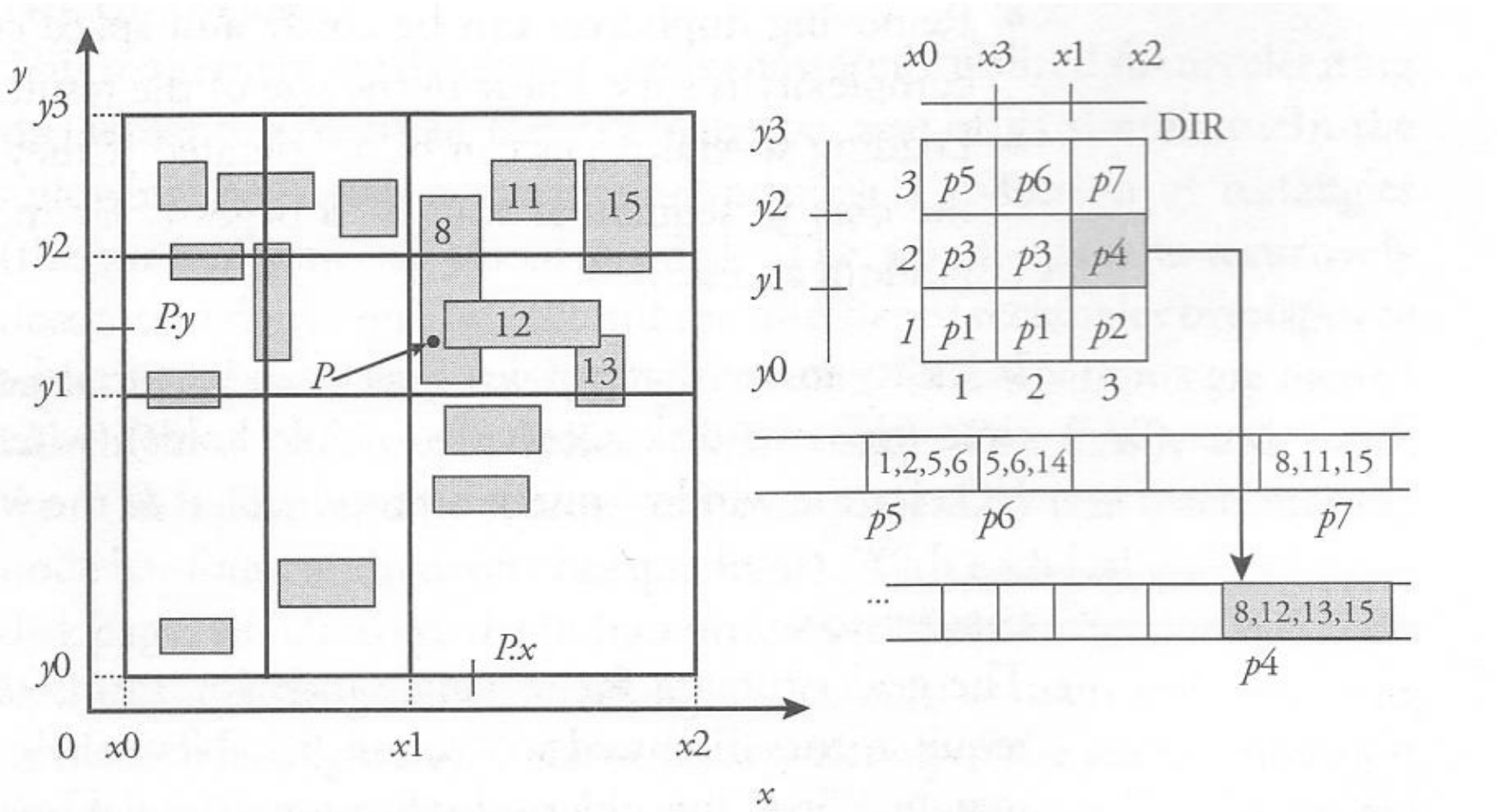
Point query: algoritmo

```
GF-POINTQUERY (P(a,b):point):set(oid)
   begin
    result = ∅
    #individuazione della cella che contiene P
    i = RANK(a, Sx); j = RANK(b, Sy)
    page = READPAGE(DIR[i,j].PageID)
    for each e in page do
        if P ∈ e.mbb then result+= {e.oid}
    end for
    return result
   end
```

Complessità

- Nel caso della fixed grid, il costo (tempo di CPU) è costante, nel caso del grid file è logaritmico (ricerca dicotomica sulle scale)
- Successivamente, è necessario un accesso a disco per leggere la pagina indirizzata dalla cella, che restituisce un insieme E di entry della forma [MBB,OID]
- Per ogni entry e $\in$ E, va verificato se P $\in$ e.mbb
- Per tutte le entry che superano il test, va effettuato il passo di raffinamento (verificare l'appartenza di P all'effettiva geometria dell'oggetto spaziale)

Un esempio di point query



Point query (punto P) con grid file

Window query: algoritmo - 1

```
GF-WINDOWQUERY (W(x1,y1,x2,y2):rectangle):set(oid)
   begin
    result = ∅
    #individuazione della cella più in basso a sinistra
    #che interseca W
    i1 = RANK(x1, Sx); j1 = RANK(y1, Sy)
   #individuazione della cella più in alto a destra
   #che interseca W
     i2 = RANK(x2, Sx); j2 = RANK(y2, Sy)
    #scansione delle celle della griglia
    for i=i1;i≤i2;i++ do
        for j=j1;j≤j2;j++ do
         #leggi la pagina e controlla ogni entry
        page = READPAGE(DIR[i,j].PageID)
```

Window query: algoritmo - 2

```
            for each e in page do
              if W ∩ e.mbb ≠ ∅ then result+= {e.oid}
            end for
        end for
    end for
    # ordina il risultato e rimuovi i duplicati
    SORT(result); REMOVEDUPLICATE(result);
    return result
end
```

Osservazioni.

1. Rimuovere i duplicati può essere costoso (più che lineare nella dimensione del risultato)
2. Il caricamento delle pagine può essere più efficiente se esse sono consecutive

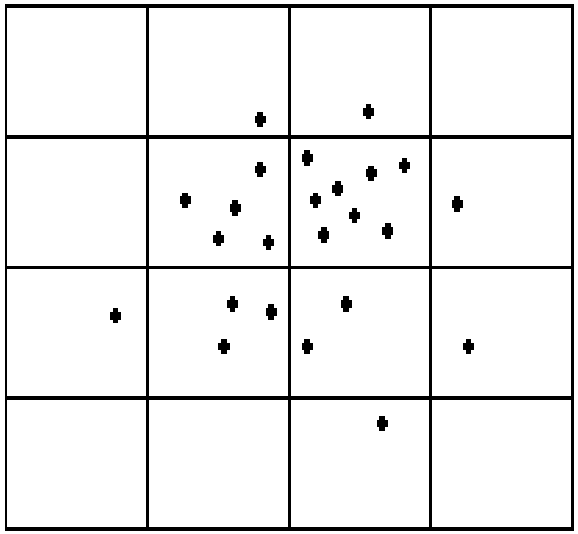
Complessità

- A differenza delle point query, che si possono eseguire in tempo costante (2 operazioni di I/O), il numero di operazioni di I/O per le window query è **proporzionale** all'area della finestra

- **Limiti** dei grid file:
  - la **duplicazione** degli oggetti nelle celle vicine causa un aumento del numero di entry dell'indice e, quindi, un aumento della dimensione dell'indice (specie quando la dimensione delle celle diventa comparabile a quella degli mbb)
  - la **rimozione** dei duplicati è costosa specie se il risultato è di grandi dimensioni
  - se le dimensioni della directory impediscono di tenerla in memoria principale, le prestazioni degradano rapidamente (la gestione della directory risulta complessa e costosa)

Strutture lineari: introduzione - 1

- Come nel caso dei grid file, l'idea iniziale è quella di una struttura dati basata sulla struttura a griglia fissa
- La struttura a griglia fissa partiziona una regione piana in celle di uguale dimensione



Strutture lineari: introduzione - 2

- Come già osservato, una corrispondenza cella/blocchi di memoria in una struttura a griglia fissa non è ragionevole
  - I blocchi di memoria hanno una capacità costante, mentre la distribuzione dei dati può non essere uniforme (certe celle possono essere poco popolate, altre molto popolate)

- La struttura a griglia fissa deve essere raffinata per consentire una suddivisione dinamica delle celle

I quadtree - 1

- La struttura Quad-Tree si basa sulla decomposizione ricorsiva dello spazio in celle/quadranti (quadtree decomposition): i rettangoli da indicizzare sono mappati in celle ottenute attraverso una decomposizione ricorsiva dello spazio in celle/quadranti

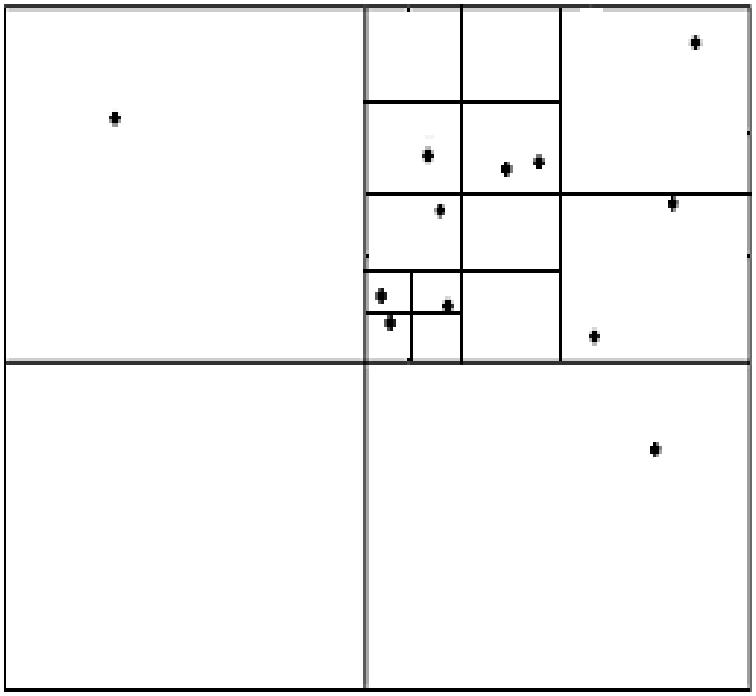
- Fissato un numero massimo di elementi contenuti in un quadrante, ogni volta che si verifica un overflow, il quadrante interessato viene suddiviso il quattro quadranti

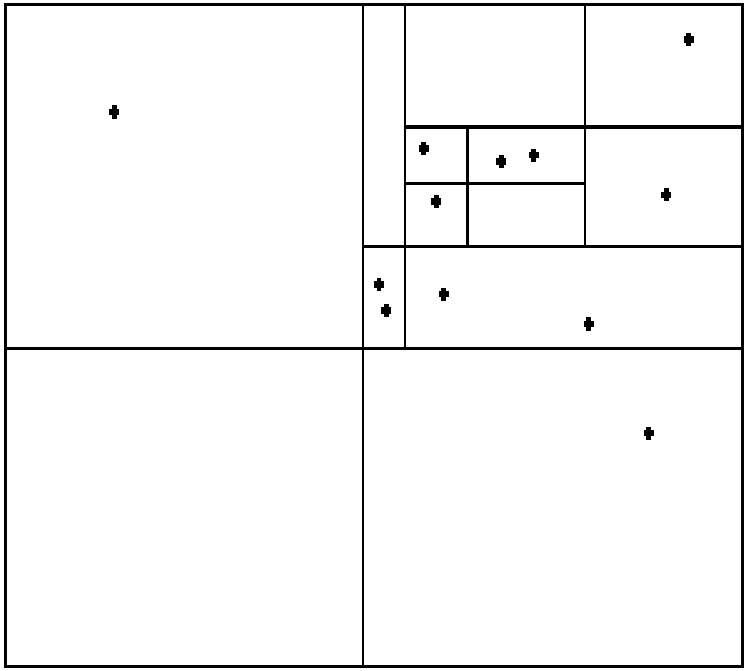
I quadtree - 2

- Sono memorizzati attraverso strutture ad albero multilivello in cui ogni nodo interno ha 4 figli

- La celle e i rettangoli ad esse associati sono indicizzati per mezzo di un B^+-albero che utilizza il rango della cella come chiave

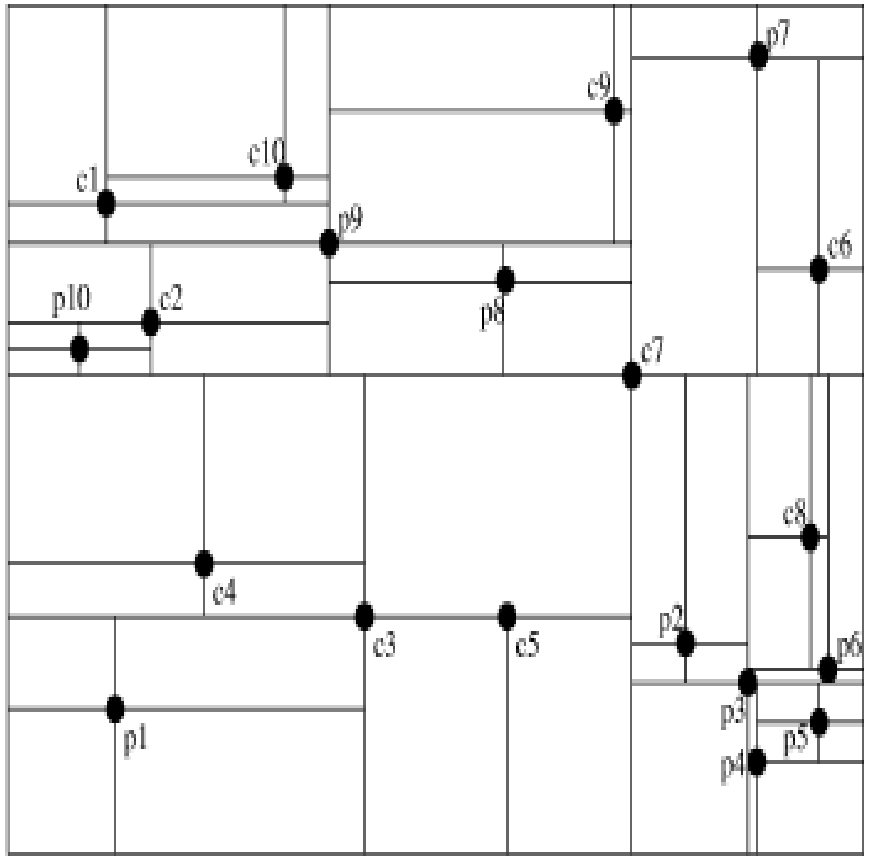
Esempi di quadtree

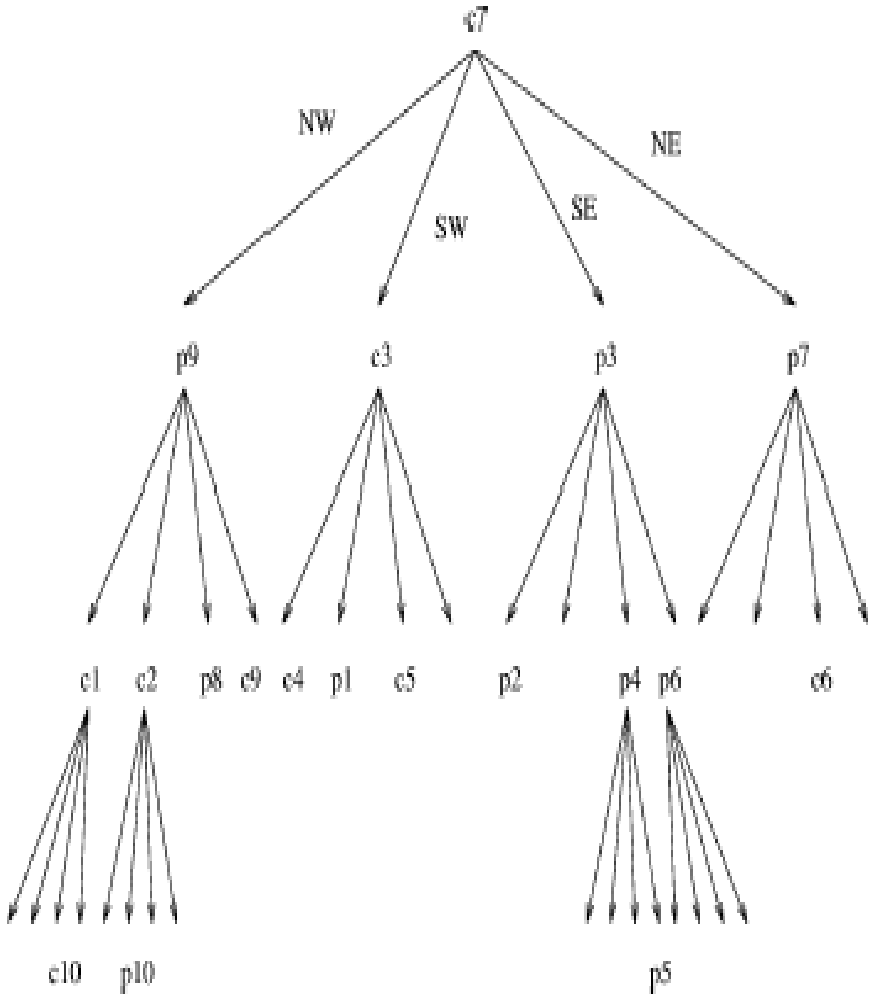




La dimensione dei sottoquadranti ottenuti dalla suddivisione di un quadrante non deve essere necessariamente la stessa, anche se questo è ciò che accade normalmente (come nel caso dei grid file)

Un’altra possibilità: point quadtree





Tipologie di quadtree

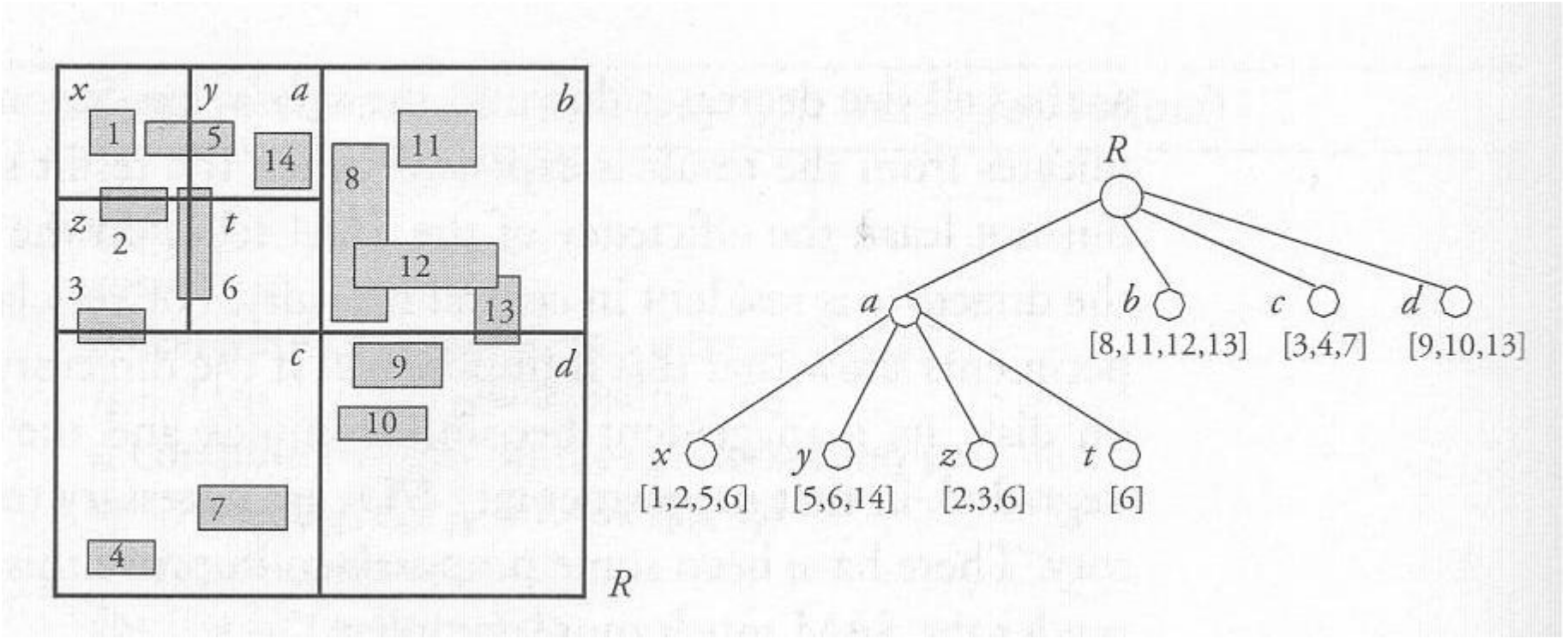
- 2 approcci principali:
  - quadtree lineari (fanno riferimento agli MBB degli oggetti spaziali)
  - z-ordering tree (approssima la geometria dell'oggetto non attraverso l'MBB, ma attraverso le celle della quadtree decomposition)

  Ci concentreremo sul primo approccio

I quadtree (semplici)

- Lo spazio di ricerca è ricorsivamente decomposto in quadranti, fino a quando il numero di rettangoli che intersecano ciascun quadrante è inferiore alla capacità di una pagina
- I quadranti sono denominati NW, NE, SW e SE. L'indice è un albero quaternario. Ogni nodo interno ha 4 figli, uno per quadrante. Ogni foglia è associata ad una pagina di memoria secondaria che memorizza le entry dell'indice
- Ogni rettangolo appare in tutti i quadranti foglia con cui ha intersezione non nulla

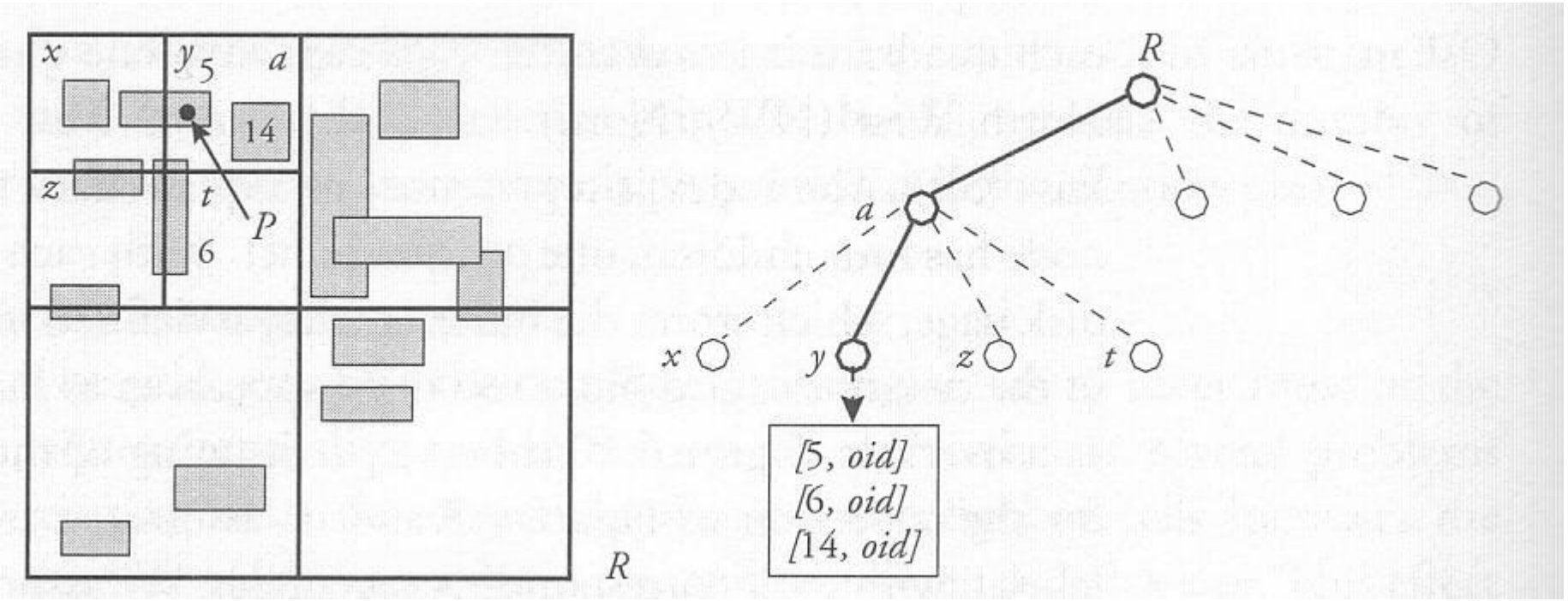
Un esempio



Un quadtree semplice
(capacità: 4 entry per pagina)

Point query in un quadtree

- Si segue un cammino dalla radice ad una foglia. Ad ogni livello, si sceglie il quadrante che contiene il punto ricevuto in input. Una volta raggiunta una foglia, questa viene letta e scandita come nel caso del grid file



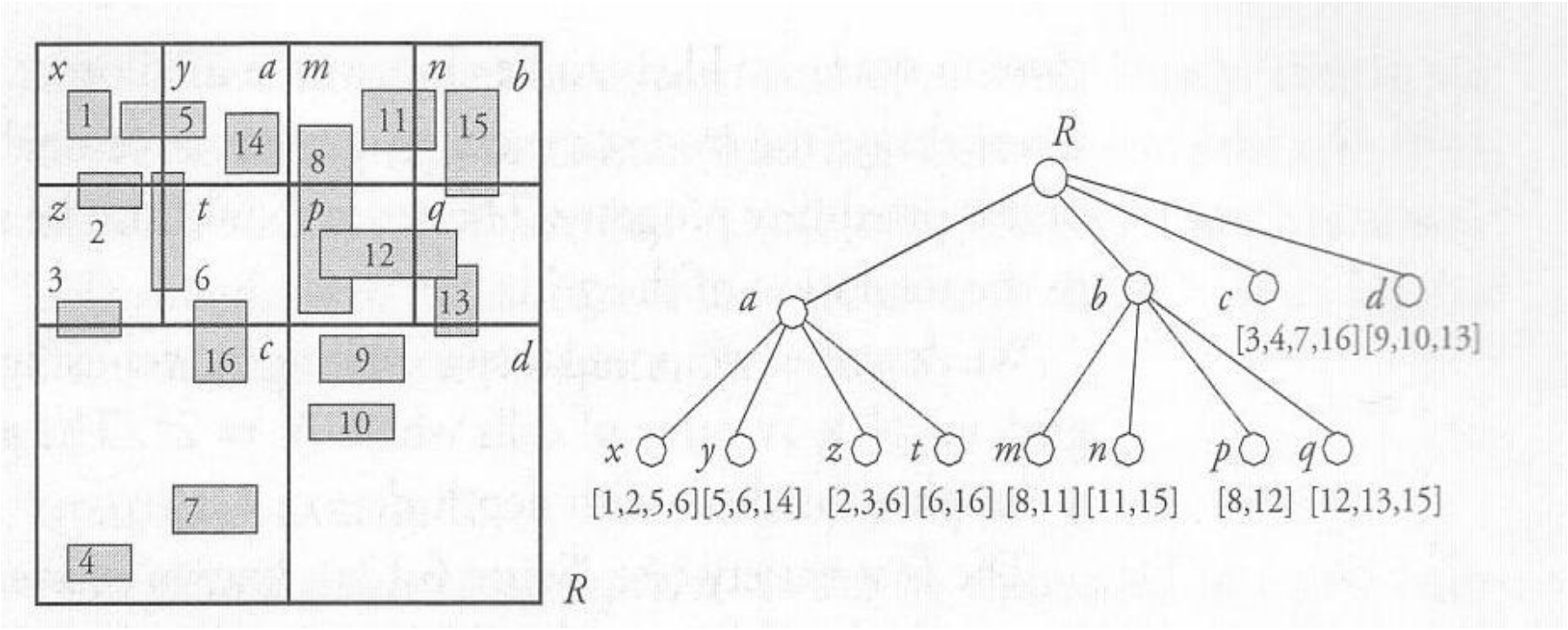
Window query in un quadtree

- Simile alla point query nella struttura fondamentale
- Vanno, però, recuperati tutti i quadranti che hanno intersezione non nulla col rettangolo ricevuto in input (*dettagli omessi*)

Inserimento (dinamico) in un quadtree

- Un rettangolo viene inserito in ogni quadrante foglia cui si sovrappone
- Occorre, quindi, seguire tutti i cammini che portano a tali foglie
- Viene letta la pagina p associata ad ogni foglia.
- Due casi:
  - se p non è piena, viene inserita una nuova entry
  - se p è piena, il quadrante viene suddiviso in 4 nuovi quadranti e tre nuove pagine vengono allocate al quadtree. Le vecchie entry e la nuova vengono distribuite fra le 4 pagine: un'entry e viene inserita in ogni pagina il cui quadrante interseca e.mbb

Un esempio



Inserimento dei rettangoli 15 e 16

Limiti - 1

- A differenza dei B-alberi e dei B^+-alberi il fan-out è molto piccolo (4): i 4 figli di un nodo occupano una piccola parte di una pagina (è difficile mappare i quadtree nelle pagine di memoria secondaria)
- Il tempo di accesso è legato alla profondità dell'albero che può essere elevata. Nel caso peggiore, ogni nodo interno si trova in una pagina diversa e il numero di operazioni di I/O è pari alla profonditá dell'albero

Limiti - 2

- Se la collezione è statica i nodi si possono impaccare nelle pagine, se è dinamica non è possibile farlo in modo efficiente
- Come nel caso dei grid file, vi è il problema della duplicazione degli oggetti

Space-filling curve

- Una Space-filling curve definisce un ordinamento totale delle celle di una griglia 2D che preserva parzialmente la prossimità

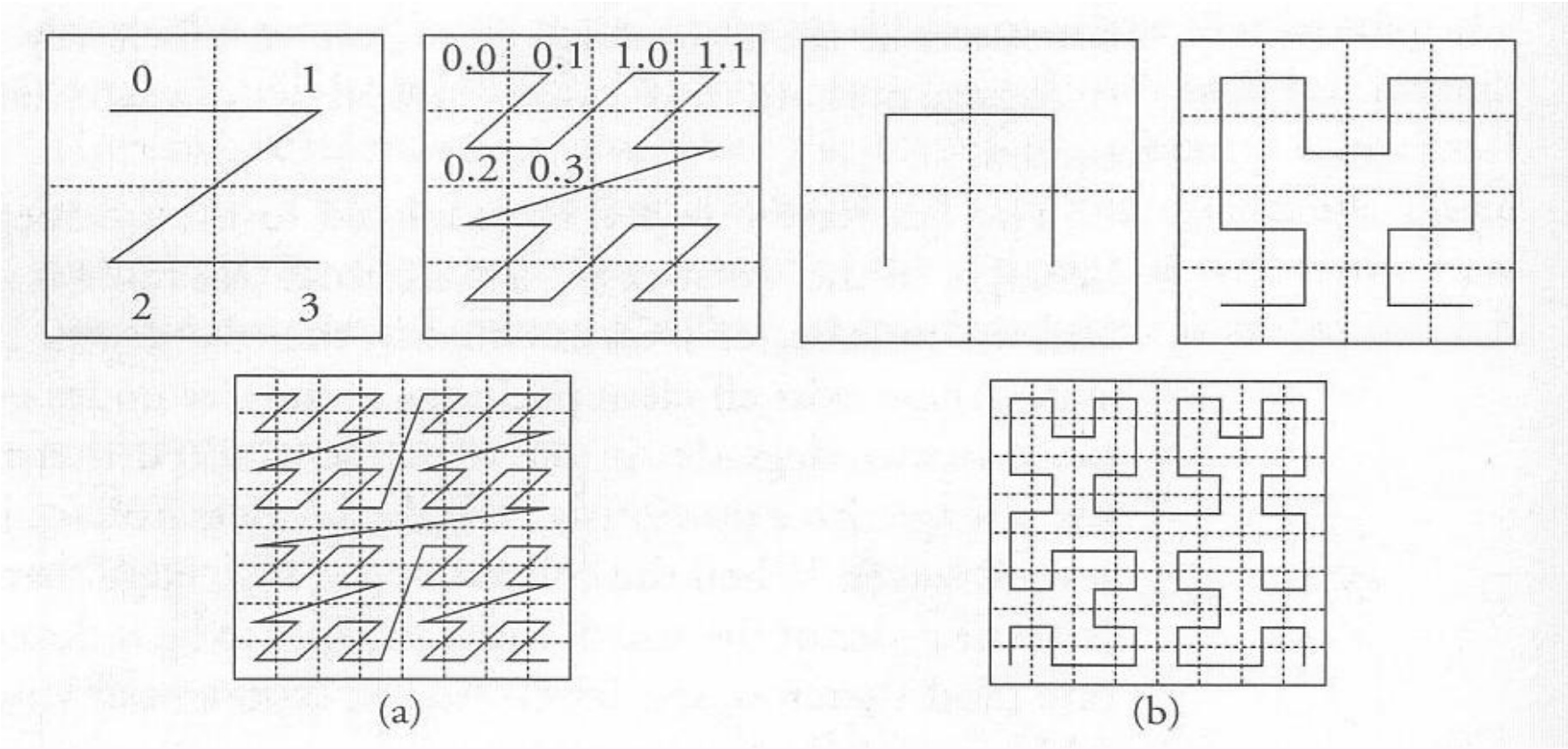
- Tale ordine deve essere stabile rispetto alla risoluzione della griglia

- 2 space-filling curve che usano una matrice di dimensione N x N celle, dove $N=2^d$ (un quadrante completo di profondità 2^d):
  - z-ordering
  - Hilbert curve

Z-ordering

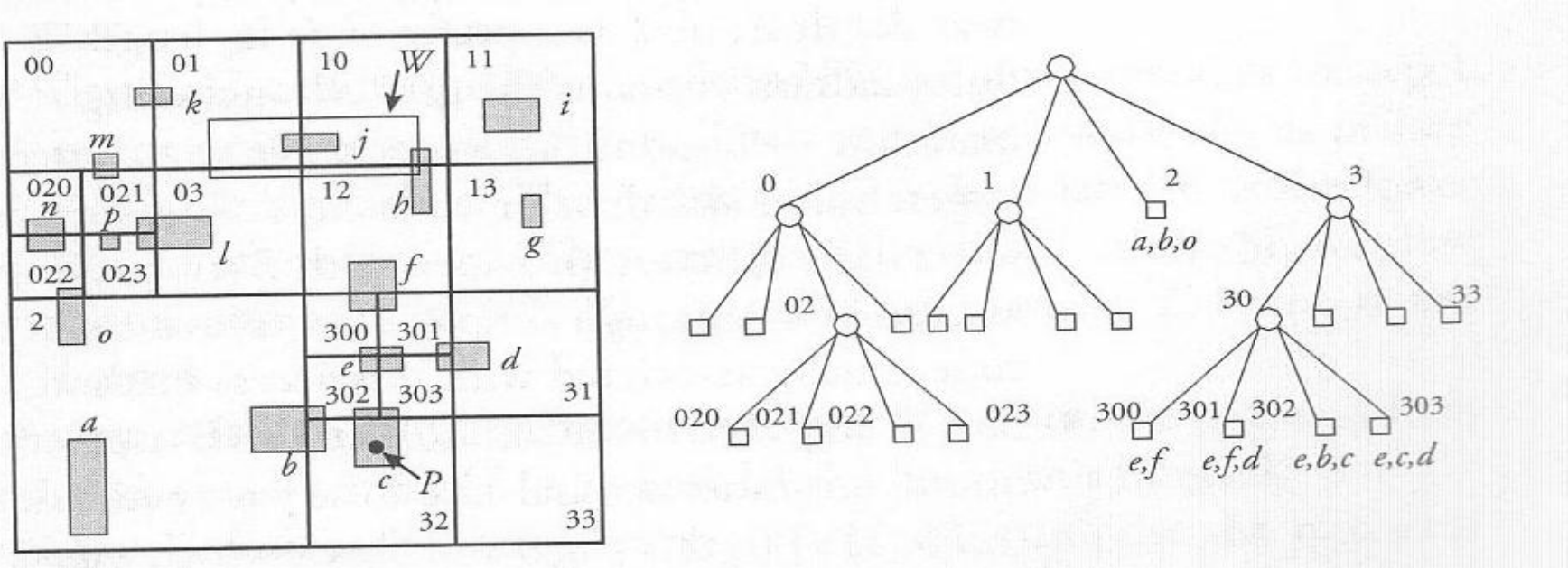
- Ad ogni nodo è associata un'etichetta scelta tra le stringhe sull'alfabeto {0, 1, 2, 3}
- La radice usa come etichetta la stringa vuota
- Il figlio NW (rispettivamente, NE, SW, SE) di un nodo interno con etichetta k ha etichetta k·0 (risp., k·1, k·2, k·3)
- Le celle sono etichettate con stringhe di lunghezza d
- Le celle sono ordinate in base alle loro etichette (ordinamento lessicografico)

Z-ordering e Hilbert curve



Etichettatura del quadtree - 1

- Un quadtree di profondità d può essere immerso in una griglia N x N con $N=2^d$
- Ogni quadrante foglia sarà etichettato con una stringa di lunghezza ≤ d



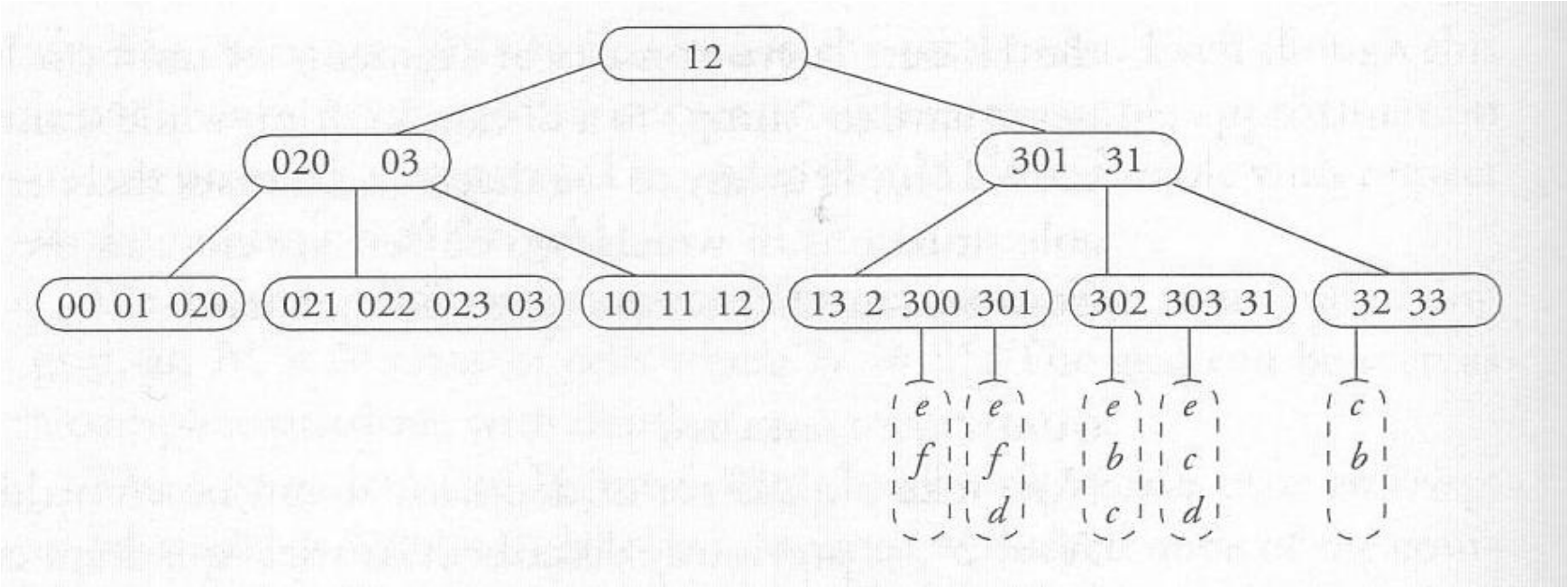
Etichettatura del quadtree - 2

- L'ordine delle foglie corrisponde alla scansione delle foglie da sinistra a destra
- Le etichette possono essere di lunghezza diversa (la loro lunghezza è pari alla loro profondità nell'albero)
- L'etichetta di una foglia può essere vista come l'etichetta del cammino dalla radice alla foglia

Linear QuadTree

- Assegnate le entry [MBB,OID] ad una foglia del quadtree con etichetta l, memorizzata in una pagina con indirizzo p, la collezione di coppie (l,p) sarà indicizzata in un B^+-albero con chiave l'etichetta l della foglia
- Tale struttura realizza un'organizzazione compatta delle etichette del quadtree nelle foglie del B^+-albero
- Tale organizzazione è dinamica e si mantiene a fronte di inserimenti/cancellazioni di oggetti della collezione

Esempio



Le foglie del quadtree indicizzate con un B^+-albero

NOTA. Tale soluzione presenta ancora il problema delle ridondanze: mbb che si sovrappongono a diverse foglie del quadtree sono presenti nelle pagine associate a tali foglie

Point Query con quadtree lineari - 1

- Viene eseguita in tre passi:
  1. CALCOLO DELL'ETICHETTA DEL PUNTO - Viene calcolata l'etichetta l (stringa di lunghezza d) della cella della griglia che contiene P (funzione POINTLABEL)
  2. RECUPERO DELLA FOGLIA DEL QUADTREE NEL B^+-ALBERO (funzione MAXINF(l)) - Sia L l'etichetta della foglia del quadtree che contiene P. Sono possibili due casi:
     - L=l (la foglia dell'albero è a profondità d)
     - L è un prefisso di l (la lunghezza di L è minore di d e il quadrante associato alla foglia contiene la cella della griglia cui appartiene P; tale quadrante corrisponde all'entry del B^+-albero la cui chiave è il valore più grande minore o uguale a l)

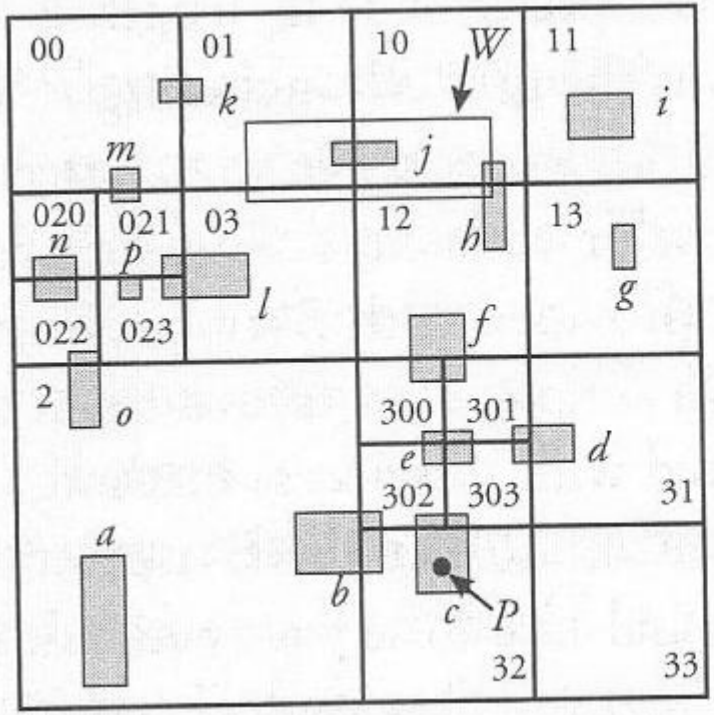
Point Query con quadtree lineari - 2

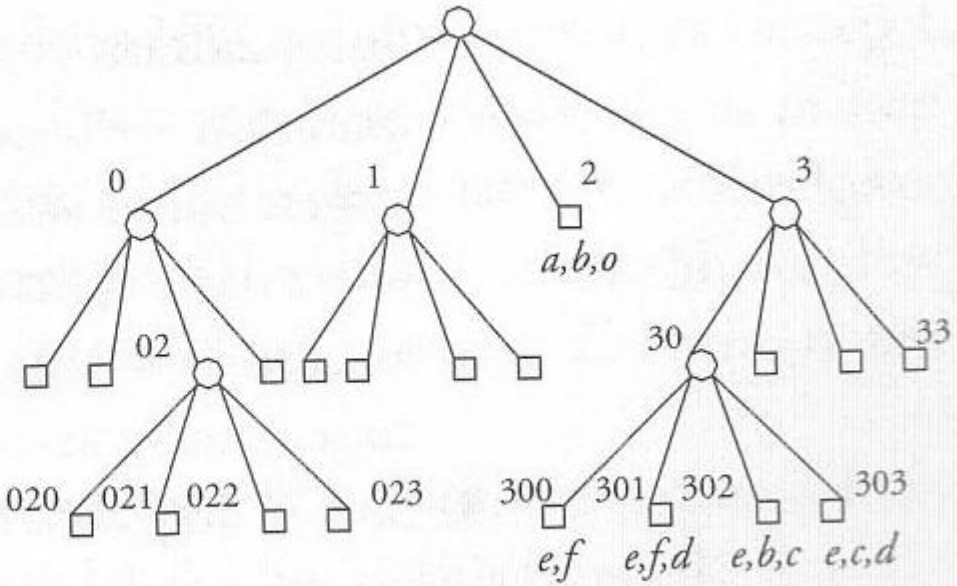
3. ACCESSO ALLA FOGLIA E SCANSIONE - Una volta individuata l'entry [L, p], si accede alla pagina all'indirizzo p, si scandiscono tutte le coppie [MBB, OID] in essa contenute e si individuano gli MBB che contengono P (uno o più). Il risultato è una lista di OID.

Point query: algoritmo

```
LQ-POINTQUERY (P:point):set(oid)
  begin
   result = ∅
   #Step 1: calcolo dell'etichetta del punto
   l = POINTLABEL(P)
   #Step 2: entry [L,p] si ottiene attraversando
   #il B+-albero con la chiave l
   [L,p] = MAXINF(l)
   #Step 3: data la pagina si ottiene l'oggetto
   page = READPAGE(p)
   for each e in page do
       if (e.mbb contains P) then result+= {e.oid}
   end for
   return result
  end
```

Esempio


00
01
10
W
11
k
i
m
j
020
n
021
p
03
12
h
13
l
g
022
023
f
2
o
300
301
e
d
302
303
31
a
b
c
P
32
33


0
1
2
3
a,b,o
02
30
33
020
021
022
023
300
301
302
303
e,f
e,f,d
e,b,c
e,c,d

Window query con quadtree lineari - 1

- Calcolare l'intervallo I di etichette che copre tutti i quadranti che si sovrappongono alla finestra W di input e formulare una range query sul B^+-albero con parametro I
- Viene eseguita in 3 passi:
  - PASSO 1. Calcolare l'etichetta l del vertice NW della finestra (come nel passo 1 dell'algoritmo per le POINT QUERY). Successivamente, calcolare MAXINF (l) (come nel passo 2 dell'algoritmo per le POINT QUERY). In tal modo si ottiene la coppia [L,p], dove L è l'estremo inferiore di I. In modo analogo, operando sul vertice SE, si ottiene la coppia [L', p'], dove L' è l'estremo superiore di I

Window query con quadtree lineari - 2

- PASSO 2. Formulare una range query sul B^+-albero con argomento l'intervallo [L,L'], che restituisce tutte le entry [l,p] con l compresa nell'intervallo

- PASSO 3. Per ogni entry del B^+-albero e = [l,p], si calcola il quadrante etichettato da e.l con la funzione QUADRANT(e.l). Se il quadrante si sovrappone alla finestra, si accede alla pagina e.p e si controlla ogni entry del quadtree [MBB,OID] per vedere se si sovrappone a W

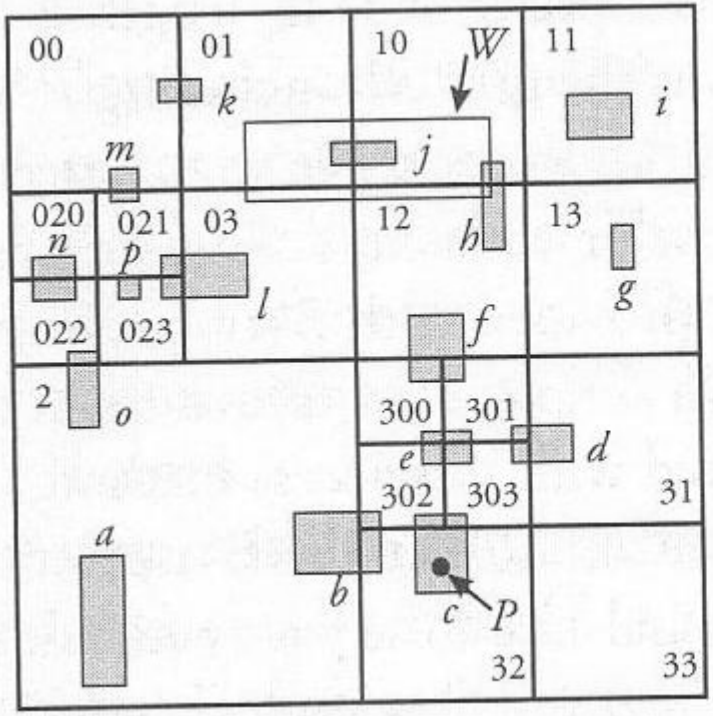
Window query: algoritmo -1

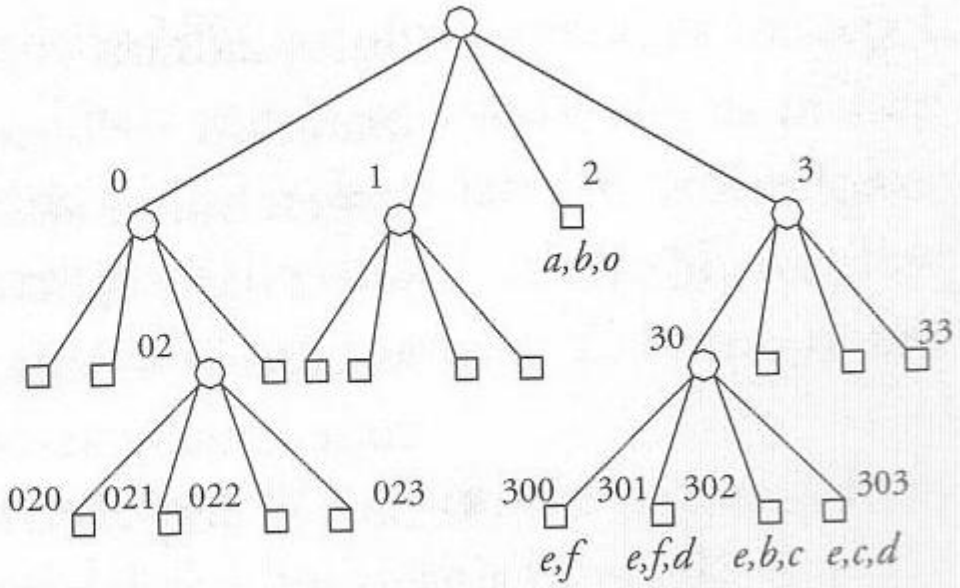
```
LQ-WINDOWQUERY (W:rectangle):set(oid)
    begin
     result = ∅
     #Step 1: dai vertici W.NW e W.SE della finestra determina
     #l'intervallo [L,L'] attraverso due ricerche sul B+-albero
     l = POINTLABEL(W.nw); [L,p] = MAXINF(I)
     l' = POINTLABEL(W.se); [L',p'] = MAXINF(I')
     #Step 2: calcola l'insieme Q di entry [l,p] con l in [L,L']
     Q = RANGEQUERY([L,L'])
     #Step 3: per ogni entry in Q il cui quadrante interseca
     #W, si accede alla pagina
```

Window query: algoritmo -2

```
        for each q in Q do
            if (QUADRANT(q.p) overlaps W) then
                page = READPAGE(q.p)
                for each e in page do
                    if (e.mbb contains W) then result+= {e.oid}
                end for
            end if
        end for
        #ordinamento risultato e rimozione duplicati
        SORT(result); REMOVEDUPL(result)
            return result
end
```

Esempio





Nota. Quale prezzo si paga nel mappare una collezione di quadranti 2D in una lista ordinata unidimensionale di celle? Molti quadranti che non si sovrappongono a W appartengono all'intervallo [L, L']!

Inserimento di rettangoli

- 2 possibili casi:

  1. No split dell'embedded quadtree (la pagina del quadrante è acceduta e aggiornata)

  2. Split dell'embedded quadtree (una entry del B^+-albero deve essere rimossa e sostituita da 4 nuove entry

Complessità

- Numero di operazioni di I/O per POINT QUERY

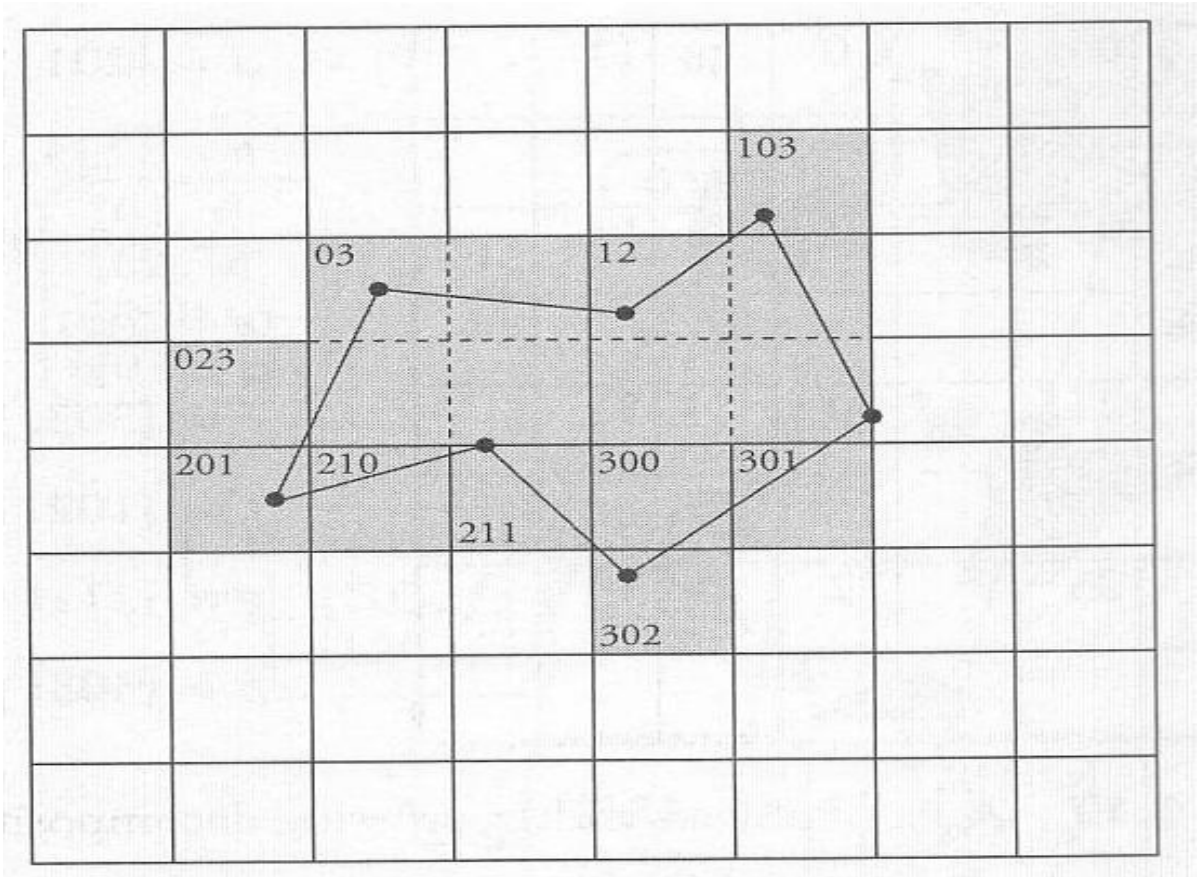
  d operazioni di I/O per raggiungere una foglia del B^+-albero (PASSO 2) e 1 operazione per l'accesso alla pagina del quadrante (PASSO 3). Totale: d + 1 operazioni di I/O

- Numero di operazioni di I/O per WINDOW QUERY (più complesso)

  PASSO 1: $2 \cdot d$ operazioni di accessi (per i 2 accessi al B^+-albero)

  PASSO 2: d operazioni di I/O per raggiungere la foglia e, successivamente, tante operazioni di I/O quante sono le foglie concatenate del B^+-albero da visitare (dipende dalla dimensione dell'intervallo [L,L']

Z-ordering tree: idea



Z-ordering e decomposizione di un oggetto