Resumen T-K^{2}raster

Related work

Es importante que primero contextualicemos hablando acerca de las estructuras e ideas precursoras al T-k2raster.
El objetivo principal es la representación de raster time series como una estructura de datos compacta, ya que esta nos perimte trabajar de forma eficiente, permitiendo consultas sobre los datos sin la necesidad de descomprimir estos.
Más aún queremos aprovechar la localidad de los datos, es decir, aprovechar las regularidades temporales que existen entre instantes de tiempo cercanos.

Rank y select en bitmaps

- Definimos las operaciones de rank y select para bitmap, las cuales serán usadas durante todo este trabajo.
- Considere un bitmap $B[0 \dots n-1]$, que guarda una secuencia de n bits.
- $rank_a(B, i)$: cuenta las apariciones del bit a en $B[0 \dots i]$.
- $select_a(B, i)$: localiza la posición de la i-ésima ocurrencia de a en B.

Quadtree

- Árbol donde cada nodo, excepto las hojas, tienen cuatro hijos.
- Tamaño potencia de 2.
- Representada por una matriz la cual se subdivide en cuatro cuadrados (quadboxes) recursivamente hasta cumplir alguna condición de termino dada.
- A un nodo se le asigna un label de "1" si este contiene hijos con label "1". Se le asigna un 0 en caso contrario.
- Los nodos con label 0 no se siguen dividiendo, mientras los que tienen label 1 sigue hasta encontrar un quadbox vacío o las celdas individuales del raster.

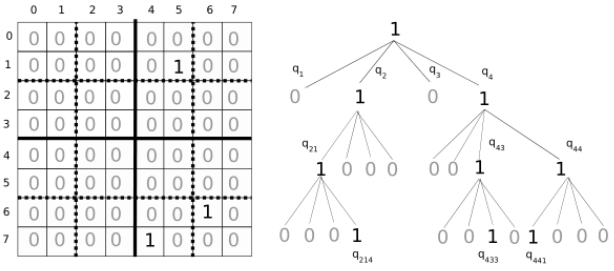

Figure 1: A binary raster and its corresponding quadtree.

K^{2}tree

- Corresponde a un region quadtree (para $k = 2$) donde los nodos del árbol se almacenan siguiendo el orden de una búsqueda en anchura.
- Por practicidad los nodos se almacenan en dos arreglos, L formado por los bits del último nivel (leafs), y T donde se almacena el resto de nodos.
- Dada una posición p en T con valor 1, podemos obtener la posición donde los k^2 hijos de p comienzan de la siguiente forma:
$$p_{hijos} = rank_1(T, p) \cdot k^2$$
- En el caso que p_{hijos} corresponda a hojas, ($p_{hijos} > |T|$) podemos obtener su posición como $L[p_{hijos} - |T|]$.
- La consulta *rank* se puede efectuar en tiempo constante, agregando una *estructura de rank* sobre T, la cual usa un espacio sublinear.

K^{3}tree

- Es la extensión del k^{2}tree añadiendo una dimensión más.
- Para $k = 2$ se representa por un octree.
- Se puede navegar eficientemente utilizando los mismos métodos del k^{2}tree, pero extendido a tres dimensiones.

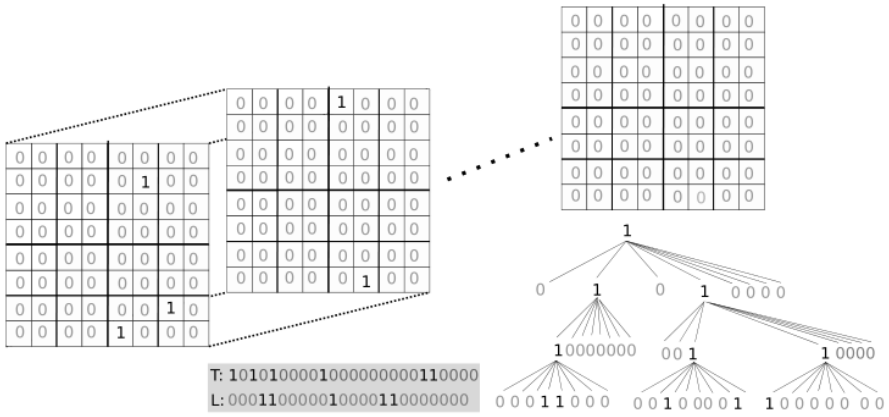

Figure 2: A sequence of binary rasters and the corresponding k^{3}tree.

Compact representation of rasters

k^{2}raster

- Además de almacenar matrices binarias como lo hace el k^{2}tree, puede almacenar matrices de enteros.
- Divide el espacio de manera similar al k^{2}tree, pero los nodos del árbol almacenan los valores máximos y mínimos de su quadtree correspondiente.
- La subdivisión termina cuando los valores max y min son iguales, o cuando se llega la las celdas del raster.

- Utiliza un arreglo de bits para representar la topología del árbol, y codificación diferencial para los enteros.

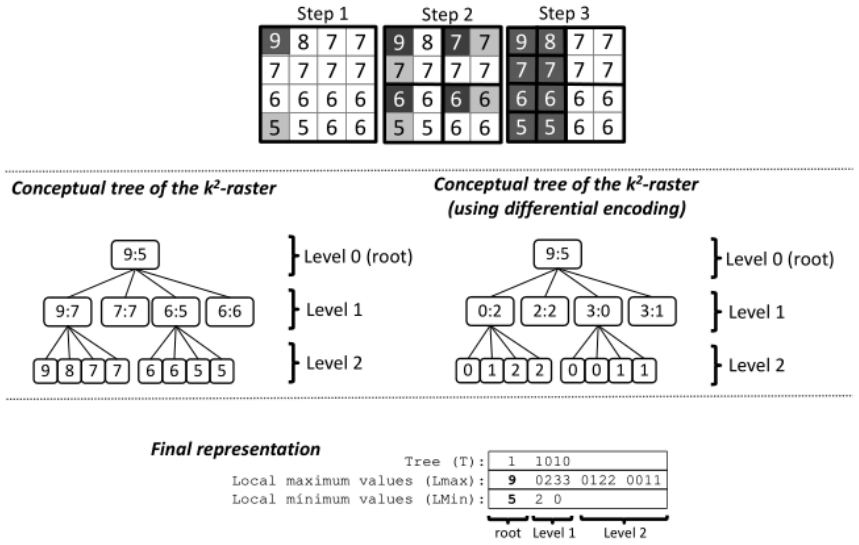

Figure 3: Example (using $k = 2$) of integer raster matrix (top), conceptual tree of the k^{2}raster, conceptual tree with differential encoding, and final representation of the raster matrix.

- Los valores max y min de cada nodo son codificados respecto a la diferencia que existe entre el max y min del nodo padre. Las hojas por defecto se comparan con el máximo.
- Estas diferencias son almacenadas en dos secuencias, *Lmax* y *Lmin*, siguiendo una búsqueda en anchura a través del árbol conceptual.

3D2D-mapping

- Método para tranformar una matriz que representa un raster, en una matriz binaria la cual se almacena en un k^{2}tree para obtener compresión y consultas rápidas directamente en el.
- La transformación se basa en el *Morton order*, también conocido como *Z-order*, el cual cuenta con una buena preservación de la localidad espacial.
- Mapea desde un espacio multidimensional a un espacio unidimensional.

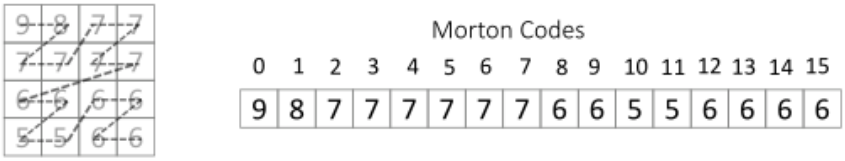

Figure 4: Z-order curve (left) and the corresponding Morton codes (right).

- Sea el vector creado V, se construye la matriz binaria BM de la siguiente forma:

La celda $BM[x][y]$ se setea en 1 si $V[x] = y$, de otra forma, se almacena un 0.

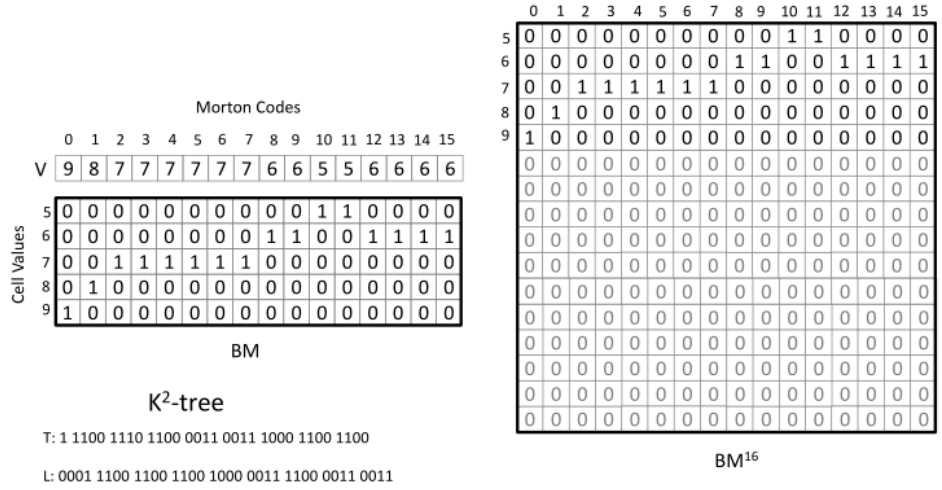

Figure 5: 3D2D-mapping of the matrix of Figure 3.

- El k^{2}tree requiere almacenar matrices de tamaño $n \times n$ donde n es una potencia de 2.
- Si BM es de tamaño $r \times c$, se extiende a tamaño $n \times n$ con bits 0 en las celdas creadas, siendo n la potencia de 2 más pequeña que sea mayor o igual a r y s. Esta matriz es llamada BM^n.
- Finalmente es representado el k^{2}tree el cual es capaz de almacenar grandes áreas llenas de 0-bits, y como se puede apreciar, utiliza mucho menos espacio que BM y BM^2.
- Ahora, se pueden obtener los valores de una celda o ventana de la matriz utilizando las consultas para k^{2}tree.

Compact representation of raster times series

4D3D-mapping

- Basado en 3D2D-mapping.
- Se utiliza el mapping para obtener una matriz binaria por cada raster representando un instante de tiempo.
- Las matrices binarias se extienden con bits 0 hasta obtener un cubo 3D perfecto de tamaño $m \times m \times m$, siendo m la menor potencia de 2 mayor que $r \times s \times \tau$, donde τ es la cantidad de instantes de tiempo.
- El cubo binario resultante es almacenado utilizando un k^3tree. El cual provee almacenamiento comprimido y consultas rápidas directamente en el.
- Dado los raster originales $M = (M_1, M_2, \ldots, M_\tau)$, se les aplica el 3D2D mapping a cada M_i, obteniendo así $(BM_1, BM_2, \ldots, BM_\tau)$, matrices binarias. Recordar que todas las matrices tienen tamaño $r \times c$.
- Definimos a BM_i^m como:
  - Para $1 \leq i \leq \tau$, BM_i se extiende con bits 0 hasta obtener una matriz de $m \times m$.
  - Para $\tau + 1 \leq i \leq m$, se agrega una matriz de tamaño $m \times m$ llena de bits 0.

- Finalmente el 4D3D-mapping se obtiene al guardar $(BM_1^m, BM_2^m, \ldots, BM_m^m)$ en un k^3tree de tamaño $m \times m \times m$, llamado BC (binary cube).

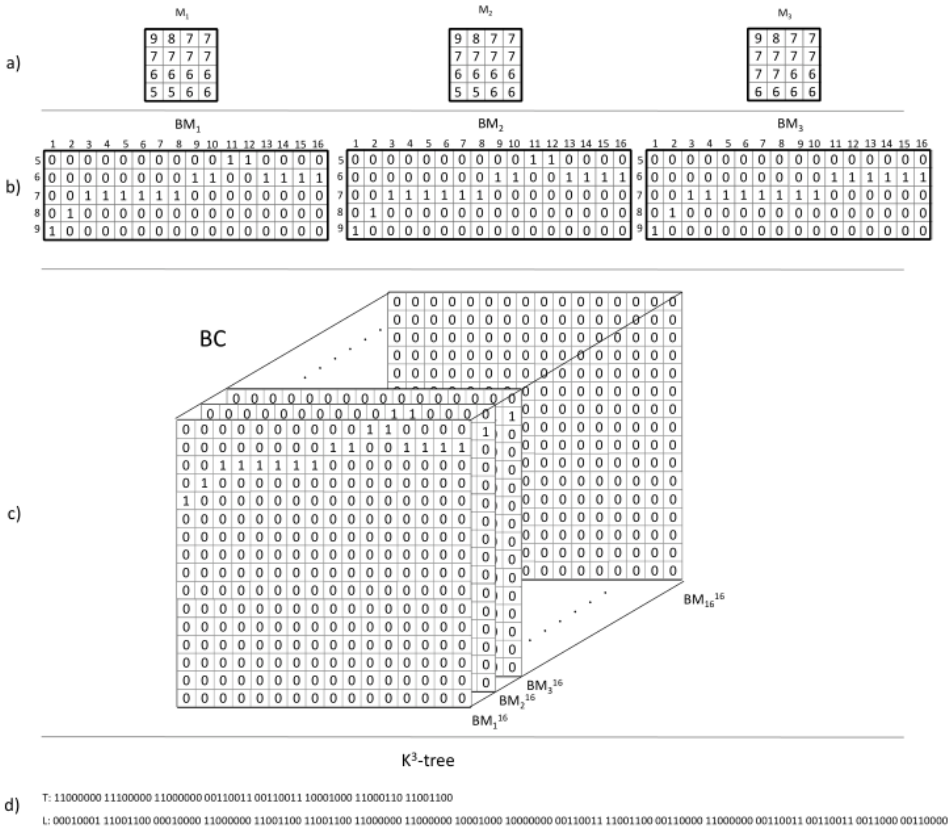

Figure 6: 4D3D-mapping for the top left submatrices in Figure 8.

Problem definition

- Tenemos una secuencia de matrices raster para distintos instantes de tiempo τ. Asumimos que cada celda de los raster almacena un entero.

Querys

Vamos a lidiar con los siguientes tipos de querys:

- $access(r, c, t)$: entrega el valor de la celda (r, c) en el instante t.
- $windowQuery(r_1, r_2, c_1, c_2, t_1, t_2)$: Entrega todos los valores en el cuboide rectangular de esquinas (r_1, c_1) y (r_2, c_2) en un intervalo de tiempo $[t_1, t_2]$.
- $rangeQuery(r_1, r_2, c_1, c_2, t_1, t_2, rMin, rMax)$: Entrega todos los valores en el cuboide rectangular de esquinas (r_1, c_1) y (r_2, c_2) en un intervalo de tiempo $[t_1, t_2]$ y entre un rango de valores $[rMin, rMax]$. Esta consulta es la más general y engloba al resto, las cuales corresponden a casos particulares de esta.

Q-cols & Q-rows

- Dada una matriz de $n \times n$, la dividimos en k grupos.

- Habrán $\frac{n}{k}$ $q-cols$ y $q-rows$.
- Dada una q-row, q-row$_i$ y la fila original $r \mid r \in \{0,\ldots,n-1\}$, definimos $relative\_row$(q-row$_i, r$) de la siguiente forma:
  - Si $r \in q-row_i$, retorna la posición relativa de r dentro de q-row$_i$.
  - Si r está en una q-row anterior a q-row$_i$, retorna 0.
  - Si r está en una q-row posterior a q-row$_i$, retorna $\frac{n}{k}-1$.
- La función $relative\_col(q-col_i, r)$ funciona de manera análoga.

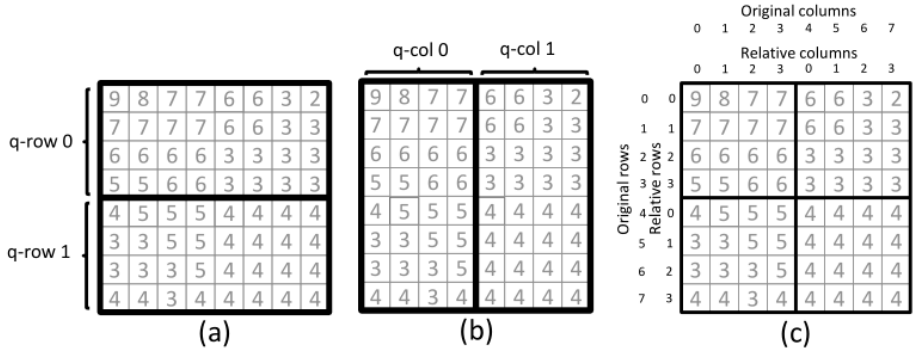

Figure 7: Definitions.

T-k^{2}raster

- Basado en el $k^2raster$ y una combinación de *snapshots* y *logs*.
- La idea es usar muestreo a intervalos regulares de tamaño t_δ.
- Snapshots: Representación de matrices raster en ciertos instantes de tiempo.
- Logs: Instantes de tiempo restantes con codificación diferencial.
- Más formalmente, matrices raster correspondientes a instantes de tiempo muestreados, M_s, $s = i+1\cdot t_\delta$, $i \in [0,(\tau-1)/t_\delta]$, son completamente representados con un $k^2raster$, y nos referimos a ellos como *snapshots*.
- Las $t_\delta - 1$ matrices raster M_t, $t \in [s+1, s+t_\delta-1]$ que siguen un snapshot M_s, son codificados usando a M_s como referencia. Para hacer esto creamos un $k^2raster'$ modifcado para representar M_t.

k^2raster

- El $k^2raster'$ es similar al $k^2raster$, pero en cada paso del proceso de construcción, los valores de los quadboxes son codificados como diferencias con respecto a los quadboxes correspondientes en M_s, en vez de diferencias con respecto al nodo padre, como se hacía en el $k^2raster$ regular.
- En el $k^2raster'$ también codificamos los valores máximos y mínimos de los nodos del árbol conceptual de M_t como diferencias a los valores correspondientes a los nodos del *snapshots*.

- En adición, si un quadbox en M_t es idéntico, o todos sus valores se diferencian en un único valor α al mismo quadbox en M_s, dejamos de dividir recursivamente y guardamos una referencia a ese quadbox de M_s y a la diferencia α.
- Guardar esa referencia es barato, ya que solo tenemos que marcar en el árbol conceptual de M_t, que para el sub-árbol con raíz en un nodo dado p , tiene la misma estructura que el del árbol de M_s.
- Para este propósito, el k^2tree' incluye además otro bitmap eqB, alineados a los ceros en T.
- Si $T[i] = 0$ (nodo sin hijos), entonces $eqB[rank_0(T, i)] \leftarrow 1$ y $Lmax[i] \leftarrow \alpha$.
- Además, si $T[i] = 0$, seteamos $eqB[rank_0(T, i)] \leftarrow 0$ y $Lmax[i] \leftarrow \beta$ (donde β es la diferencia entre los valores máximos de ambos quadboxes, como los valores del instante de tiempo es almacenado como esa diferencia) para manejar el caso donde el quadbox correspondiente de M_t tiene todas las celdas con el mismo valor (como en un $k^2raster$ regular).
- La construcción del $k^2raster'$ para la matriz M_t relacionada al snapshot M_s se puede resumir como, Sea T_t un bitmap, $Lmax_t$ y $Lmin_t$ arreglos, todos ellos cumpliendo la misma función que en un $k^2raster$ normal.
- Partiendo de la matriz completa M_t seguimos un proceso recursivo. Consideramos q_{tj} el quadbox de M_t que está siendo procesado, y q_{sj} el quadbox relacionado en M_s.
- $maxval_t$ y $minval_t$; valores max y min de q_{tj}.
- $maxval_s$ y $minval_s$; valores max y min de q_{sj}.
- z_{tj} ; posición en el bitmap T_t correspondiente a q_{tj}.
- Si q_{tj} es un quadbox de 1×1, $Lmax_t[z_{tj}] \leftarrow (maxval_t - maxval_s)$ y la recursión se detiene.
- Como el $k^2raster$ lidia con valores positivos y negativos, se aplica *folklore zigzag* para mapear negativos a positivos.
- Si $maxval_t == minval_s$; el proceso recursivo para. Seteamos $T_t[z_{tj}] \leftarrow 0$, $eqB[rank_0(T_t, z_{tj})] \leftarrow 0$ y $Lmax_t[z_{tj}] \leftarrow (maxval_t - maxval_s)$.
  - Si q_{tj} y q_{ts} difieren por completo en α, $T_t[z_{tj}] \leftarrow 0$, $eqB[rank_0(T_t, z_{tj})] \leftarrow 1$ y $Lmax[z_{tj}] \leftarrow \alpha$.

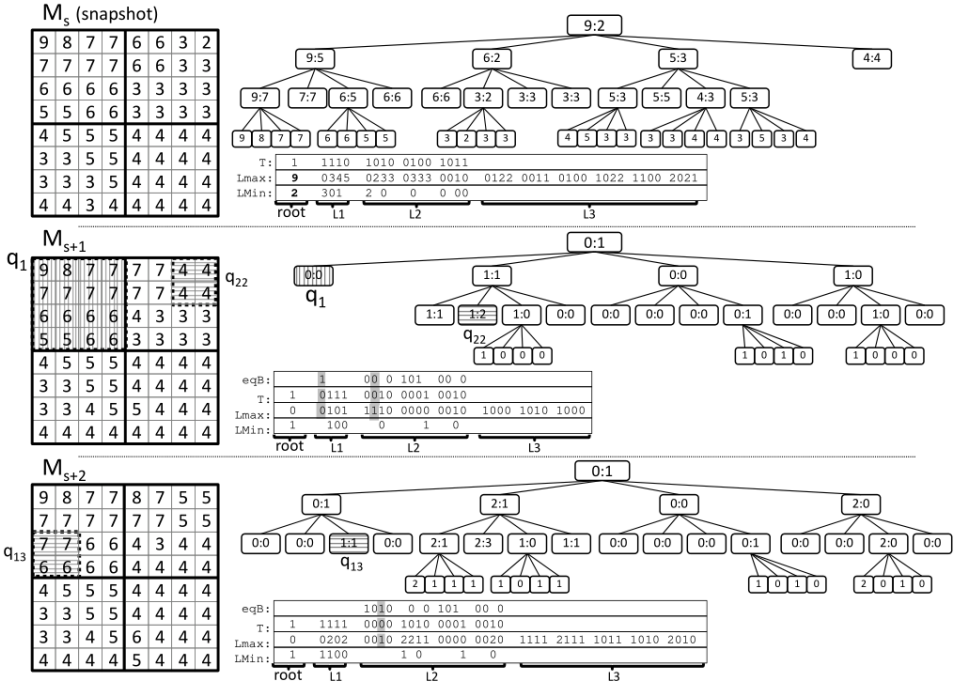

Figure 8: Structures involved in the creation of a $\mathsf{T}-\mathsf{k}^2\mathsf{raster}$ considering $\tau = 3$.

- Con todo lo visto anteriormente ya podemos comprender el algoritmo de contrucción del k^2raster'.

Algorithm 1: build(n, l, r, c) builds $\mathsf{k}^2\mathsf{raster}'$

```
1  if n=1 then
2  |  Lmax_t[l].push(M_t[r,c] − M_s[r,c])
3  |  return (M_s[r,c], M_t[r,c], M_s[r,c], M_t[r,c], M_t[r,c] − M_s[r,c])
4  else
5  |  maxval_t ← −∞; maxval_s ← −∞; minval_t ← ∞; minval_s ← ∞; diff ← −∞;
6  |  for i ← k − 1 ... 0 do /* Iterates over its q-rows */
7  |  |  for j ← k − 1 ... 0 do /* Iterates over its q-cols */
8  |  |  |  (maxval_sij, maxval_tij, minval_sij, minval_tij, diff_ij) ←build(n/k, l + 1, r + i · (n/k), c +
   |  |  |    j · (n/k))
9  |  |  |  if diff = −∞ then diff = diff_ij;
10 |  |  |  else if diff ≠ diff_ij then diff ← ∞;
11 |  |  |  if maxval_sij > maxval_s then maxval_s ← maxval_sij;
12 |  |  |  if minval_sij < minval_s then minval_s ← minval_sij;
13 |  |  |  if maxval_tij > maxval_t then maxval_t ← maxval_tij;
14 |  |  |  if minval_tij < minval_t then minval_t ← minval_tij;
15 |  if maxval_t = minval_t then /* All values are equal */
16 |  |  for i ← 0 ... k^2 − 1 do /* Remove the children, now this node is a leaf */
17 |  |  |  Lmax[l + 1].pop
18 |  |  |  if n > k then
19 |  |  |  |  T_t[l + 1].pop
20 |  |  |  |  eqB[l + 1].pop
21 |  |  T_t[l].push(0)
22 |  |  eqB[l].push(0)
23 |  |  Lmax_t[l].push(maxval_t − maxval_s)
24 |  else if diff ≠ ∞ then /* Quadbox with the same structure as that of the snapshot */
25 |  |  for i ← 0 ... k^2 − 1 do /* Remove the children, now this node is a leaf */
26 |  |  |  Lmax[l + 1].pop
27 |  |  |  if n > k then
28 |  |  |  |  T_t[l + 1].pop
29 |  |  |  |  eqB[l + 1].pop
30 |  |  T_t[l].push(0)
31 |  |  eqB[l].push(1)
32 |  |  Lmax_t[l].push(maxval_t − maxval_s)
33 |  else /* The node is not a leaf */
34 |  |  T_t[l].push(1)
35 |  |  Lmax_t[l].push(maxval_t − maxval_s)
36 |  |  Lmin_t[l].push(minval_t − minval_s)
37 |  return (maxval_s, maxval_t, minval_s, minval_t, diff)
```

Querying

Obtener el valor de una celda en un instante de tiempo

- Dos Casos:
- Si estamos haciendo la consulta sobre un snapshot, usamos el algoritmo para obtener el valor de una celda de un k^2 raster normal.
- En caso contrario, utilizamos un recorrido top-down sincronizado de los árboles representando a M_t y al snapshot previo más cercano.

Algorithm 2: access(r, c, t) returns the value of the cell (r, c) of the raster t

```
if t (mod t_δ) = 0 then /* snapshot */
    return M_t.getCell_Normal_k^2-raster(r, c) // Algorithm 2 in [24]
else
    return M_t.getCell(n, r, c, 0, 0, Lmax_s[0], Lmax_t[0]) // Algorithm 3
```

Algorithm 3: getCell$(n, r, c, z_s, z_t, maxval_s, maxval_t)$ returns the value at cell (r, c)

```
1  if z_s ≠ −1 then
2  |   z_s ← (rank_1(T_s, z_s) − 1) · k^2 + 1
3  |   z_s ← z_s + ⌊r/(n/k)⌋ · k + ⌊c/(n/k)⌋
4  |   maxval_s ← maxval_s − Lmax_s[z_s]
5  if z_t ≠ −1 then
6  |   z_t ← (rank_1(T_t, z_t) − 1) · k^2 + 1
7  |   z_t ← z_t + ⌊r/(n/k)⌋ · k + ⌊c/(n/k)⌋
8  |   maxval_t ← Lmax_t[z_t])
9  if (z_s > |T_s| or z_s = −1 or T_s[z_s] = 0) and (z_t > |T_t| or z_t = −1 or T_t[z_t] = 0) then /* Both leaves
     */
10 |   return maxval_s + ZigZag_Decoded(maxval_t)
11 else if z_s > |T_s| or z_s = −1 or T_s[z_s] = 0 then  /* Leaf in Snapshot */
12 |   z_s ← −1
13 |   return getCell(n/k, r mod (n/k), c mod (n/k), z_s, z_t, maxval_s, maxval_t)
14 else if z_t > |T_t| or z_t = −1 or T_t[z_t] = 0 then  /* Leaf in time instant */
15 |   if z_t ≠ −1 and T_t[z_t] = 0 then
16 |   |   eq ← eqB[rank_0(T_t, z_t) − 1]
17 |   |   if eq = 1 then z_t ← −1 ;
18 |   |   else return maxval_s + ZigZag_Decoded(maxval_t) ;
19 |   return getCell(n/k, r mod (n/k), c mod (n/k), z_s, z_t, maxval_s, maxval_t)
20 else  /* Both internal nodes */
21 |   return getCell(n/k, r mod (n/k), c mod (n/k), z_s, z_t, maxval_s, maxval_t)
```

Obtener un rango espacial

- Al igual que la query anterior tenemos 2 casos:
- Si estamos haciendo la consulta en un snapshots, utilizamos el algoritmo de un k^2raster para resolver esta operación.
- En caso contrario, debemos hacer un recorrido top-down sincronizado de los árboles representando a M_t y al snapshot previo más cercano. La diferencia con la query anterior es que el recorrido puede requerir varias ramas de ambos árboles, ya que la región consultada se puede superponer a los quadboxes correspondientes a varios nodos del árbol.

Algorithm 4: windowQuery$(r_1, r_1, c_2, c_2, t_1, t_2)$ returns the value of the cuboid defined by the spatial corners $[r_1, c_1]$, $[r_2, c_2]$ and time interval $[t_1, t_2]$

```
for t_i ∈ [t_1 ... t_2] do
    if t_i (mod t_δ) = 0 then /* snapshot */
        R[t_i] ← M_{t_i}.getWindow_Normal_k^2-raster(r_1, r_2, c_1, c_2)// Algorithm 3 in [24]
    else
        R[t_i] ← M_{t_i}.getWindow(n, r_1, r_2, c_1, c_2, 0, 0, Lmax_s[0], Lmax_t[0])// Algorithm 5
return R
```

Algorithm 5: getWindow$(n, r_1, r_2, c_1, c_2, z_s, z_t, maxval_s, maxval_t)$ returns all cells from region $[r_1, c_1]$ to $[r_2, c_2]$

```
1  if z_s ≠ −1 then z_s ← (rank_1(T_s, z_s) − 1) · k^2 + 1;
2  if z_t ≠ −1 then z_t ← (rank_1(T_t, z_t) − 1) · k^2+1;
3  for each q-row i overlapping [r_1, r_2] do
4      r'_1 ← relative_row(i, r_1); r'_2 ← relative_row(i, r_2);
5      for each q-col j overlapping [c_1, c_2] do
6          c'_1 ← relative_col(j, c_1); c'_2 ← relative_col(j, c_2);
7          z'_s = z'_t = −1;
8          if z_s ≠ −1 then
9              z'_s ← z_s + k · i + j;
10             maxval_s ← maxval_s − Lmax_s[z_s];
11         if z_t ≠ −1 then
12             z'_t ← z_t + k · i + j;
13             maxval_t ← Lmax_t[z_t];
14         if (z'_s > |T_s| or z'_s = −1 or T_s[z'_s] = 0) and (z'_t > |T_t| or z'_t = −1 or T_t[z'_t] = 0) then
             /* Both leaves */
15             return maxval_s + ZigZag_Decoded(maxval_t) ((r'_2 − r'_1) + 1) · ((c'_2 − c'_1) + 1) times
16         else if z'_s > |T_s| or z'_s = −1 or T_s[z'_s] = 0 then /* Leaf in Snapshot */
17             z'_s ← −1;
18             return getWindow(n/k, r'_1, r'_2, c'_1, c'_2, z'_s, z'_t, maxval_s, maxval_t);
19         else if z'_t > |T_t| or z'_t = −1 or T_t[z'_t] = 0 then /* Leaf in time instant */
20             if z_t ≠ −1 and T_t[z'_t] = 0 then
21                 eq ← eqB[rank_0(T_t, z'_t) − 1];
22                 if eq = 1 then z'_t ← −1 ;
23                 else return maxval_s + ZigZag_Decoded(maxval_t) ((r'_2 − r'_1) + 1) · ((c'_2 − c'_1) + 1)
                     times ;
24             return getWindow(n/k, r'_1, r'_2, c'_1, c'_2, z'_s, z'_t, maxval_s, maxval_t);
25         else /* Both internal nodes */
26             return getWindow(n/k, r'_1, r'_2, c'_1, c'_2, z'_s, z'_t, maxval_s, maxval_t);
```

Obtener celdas dentro de un rango de valores

Algorithm 6: rangeQuery$r_1, r_2, c_1, c_2, t_1, t_2, rMin, rMax$) returns the value of the cuboid defined by the spatial corners $[r_1, c_1]$, $[r_2, c_2]$ and time interval $[t_1, t_2]$ whose values are within the range $[rMin, rMax]$

```
for t_i ∈ [t_1 ... t_2] do
    if t_i (mod t_δ) = 0 then /* snapshot */
        R[t_i] ← M_{t_i}.searchValuesInWindow_Normal_k^2-raster(r_1, r_2, c_1, c_2)// Algorithm 4 in [24]
    else
        R[t_i] ←
            M_{t_i}.sValWin(n, r_1, r_2, c_1, c_2, 0, 0, Lmax_s[0], Lmax_t[0], Lmin_s[0], Lmin_t[0], rMin, rMax)
            // Algorithm 7
return M
```

Algorithm 7: sValWin$(n, r_1, r_2, c_1, c_2, z_s, z_t, maxval_s, maxval_t, minval_s, minval_t, rMin, rMax)$ returns all cells from region $[r_1, r_2]$, $[c_1, c_2]$ whose values are within the range $[rMin, rMax]$

```
1  maxval = maxval_s + ZigZag_Decoded(maxval_t)
2  minval = minval_s + ZigZag_Decoded(minval_t)
3  if minval ≥ rMin and maxval ≤ rMax then /* all cells meet the condition in this branch */
4  |  Output corresponding region of cells
5  |  return
6  else if minval > rMax or maxval < rMin then /* no cells meet the condition in this branch */
7  |  return
8  else if z_s ≠ −1 or z_t ≠ −1 then
9  |  if z_s ≠ −1 then z_s ← (rank_1(T_s, z_s) − 1) · k^2 + 1;
10 |  if z_t ≠ −1 then z_t ← (rank_1(T_t, z_t) − 1) · k^2+1;
11 |  for each q-row i overlapping [r_1, r_2] do
12 |  |  r'_1 ← relative_row(i, r_1);  r'_2 ← relative_row(i, r_2);
13 |  |  for each q-col j overlapping [c_1, c_2] do
14 |  |  |  c'_1 ← relative_col(j, c_1);  c'_2 ← relative_col(j, c_2);
15 |  |  |  z'_s = z'_t = −1;
16 |  |  |  if z_s ≠ −1 then
17 |  |  |  |  z'_s ← z_s + k · i + j;
18 |  |  |  |  maxval_s ← maxval_s − Lmax_s[z'_s];
19 |  |  |  |  if z'_s < |T_s| and T_s[z'_s] = 1 then minval_s ← minval_s + Lmin_s[rank_1(T_s, z'_s) − 1];
20 |  |  |  if z_t ≠ −1 then
21 |  |  |  |  z'_t ← z_t + k · i + j;
22 |  |  |  |  maxval_t ← Lmax_t[z'_t];
23 |  |  |  |  if z'_t < |T_t| and T_t[z'_t] = 1 then minval_t = Lmin_t[rank_1(T_t, z'_t) − 1];
24 |  |  |  if z'_s > |T_s| or z'_s = −1 or T_s[z'_s] = 0; then /* Leaf in Snapshot */
25 |  |  |  |  minval_s ← maxval_s
26 |  |  |  |  z'_s ← −1;
27 |  |  |  if z'_t > |T_t| or z'_t = −1 or T_t[z'_t] = 0 then /* Leaf in time instant */
28 |  |  |  |  if z'_t ≠ −1 and T_t[z'_t] = 0 and eqB[rank_0(T_t, z'_t) − 1] = 0 then
29 |  |  |  |  |  minval_t ← (maxval_s + maxval_t) − minval_s;
30 |  |  |  |  else if z'_t ≠ −1 and T_t[z'_t] = 0 and eqB[rank_0(T_t, z'_t) − 1] = 1 then
31 |  |  |  |  |  minval_t ← maxval_t;
32 |  |  |  |  z'_t ← −1;
33 |  |  |  return
   |  |  |  sValWin(n/k, r'_1, r'_2, c'_1, c'_2, z'_s, z'_t, maxval_s, maxval_t, minval_s, minval_t, rMin, rMax)
```