K^2 Tree

Compact Representation of Web Graphs with Extended Functionality

Cheng Zhao

October 25, 2017

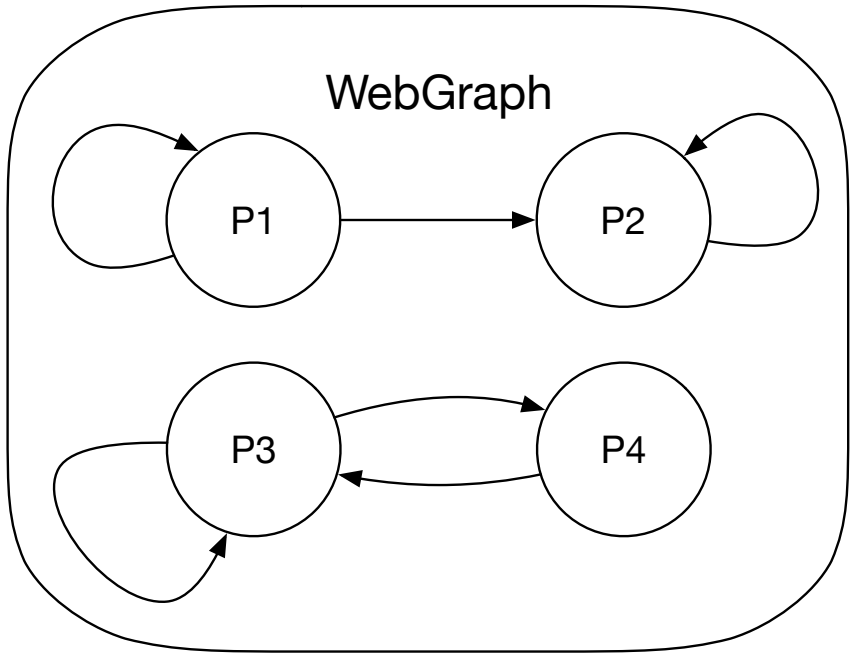
Table of contents

Overview

- k^2tree 是在图邻接矩阵上做压缩的一种树状的数据结构

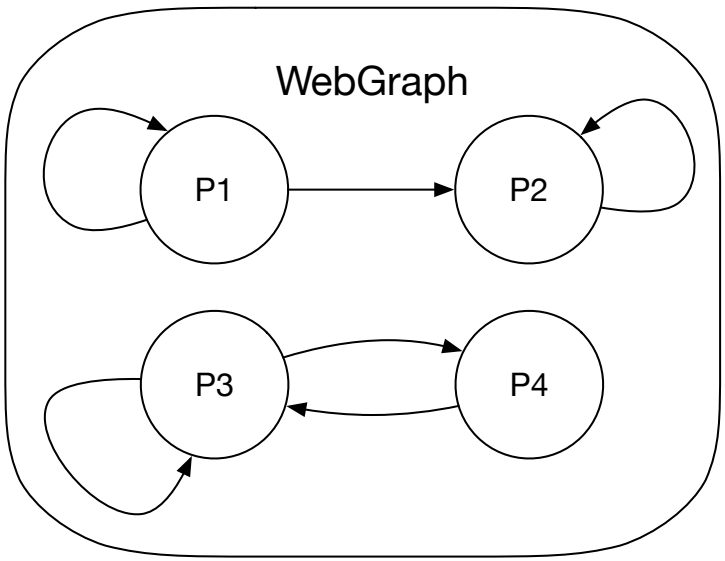
Overview

- k^2tree 是在图邻接矩阵上做压缩的一种树状的数据结构

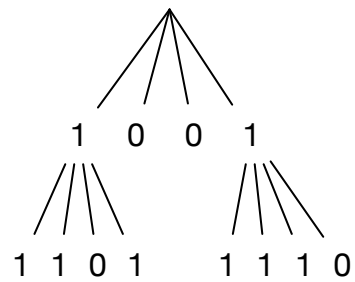


Overview

- k^2tree 是在图邻接矩阵上做压缩的一种树状的数据结构



1	1	0	0
0	1	0	0
0	0	1	1
0	0	1	0



Construction

- 构建$k^2 tree$ 其中$k = 2$

	0	1	2	3	4	5	6	7	8	9	10
0		1									
1			1	1	1						
2											
3											
4											
5											
6											
7							1				
8							1			1	
9							1		1		1
10							1			1	

Construction

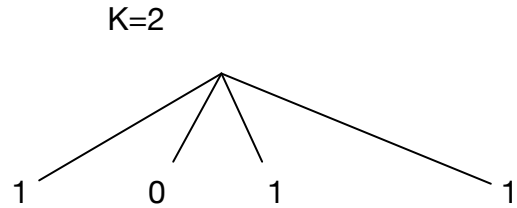
- 构建k^2tree 其中$k = 2$

	0	1	2	3	4	5	6	7	8	9	10
0		1									
1			1	1	1						
2											
3											
4											
5											
6											
7							1				
8							1			1	
9							1		1		1
10							1			1	

Construction

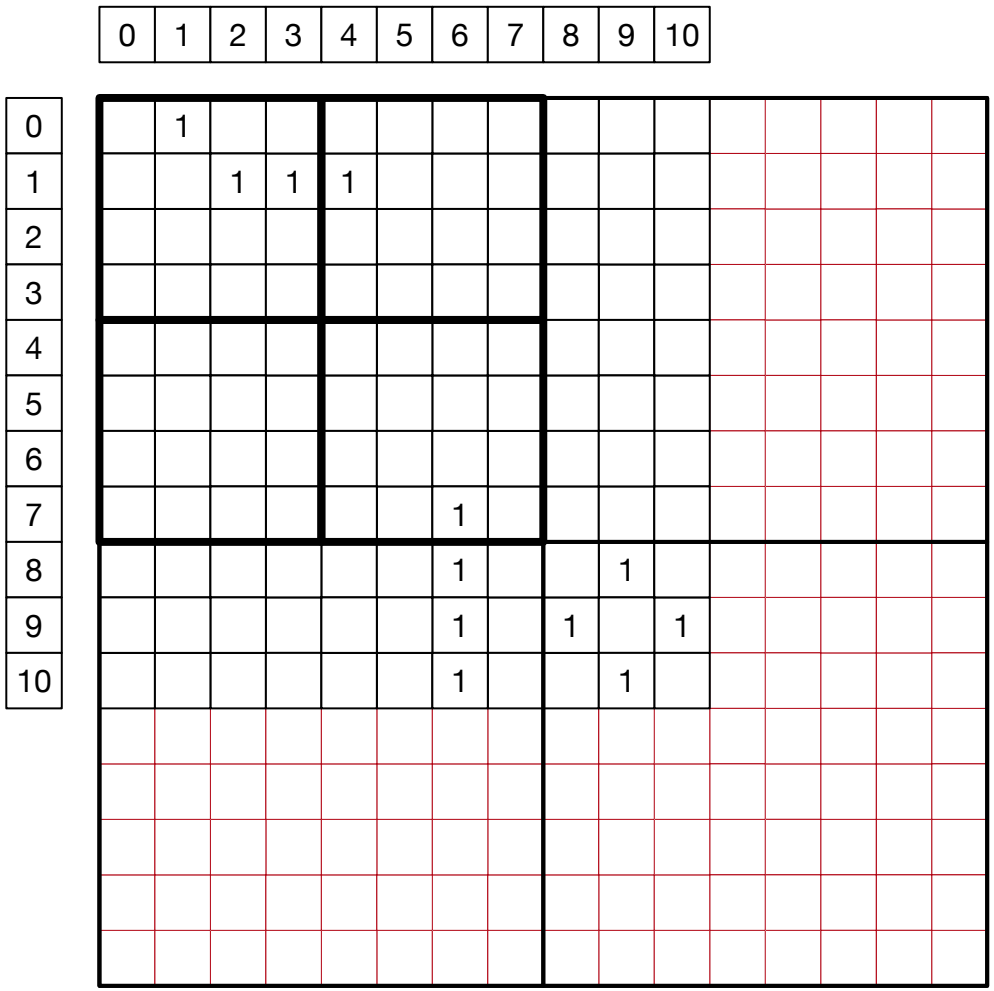
- 构建k^2tree 其中$k = 2$

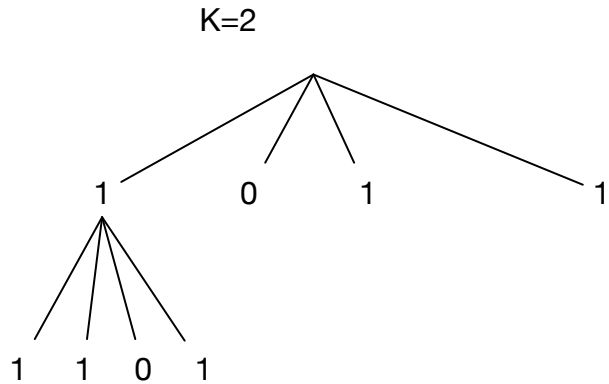
	0	1	2	3	4	5	6	7	8	9	10
0		1									
1			1	1	1						
2											
3											
4											
5											
6											
7							1				
8							1			1	
9							1		1		1
10							1			1	



Construction

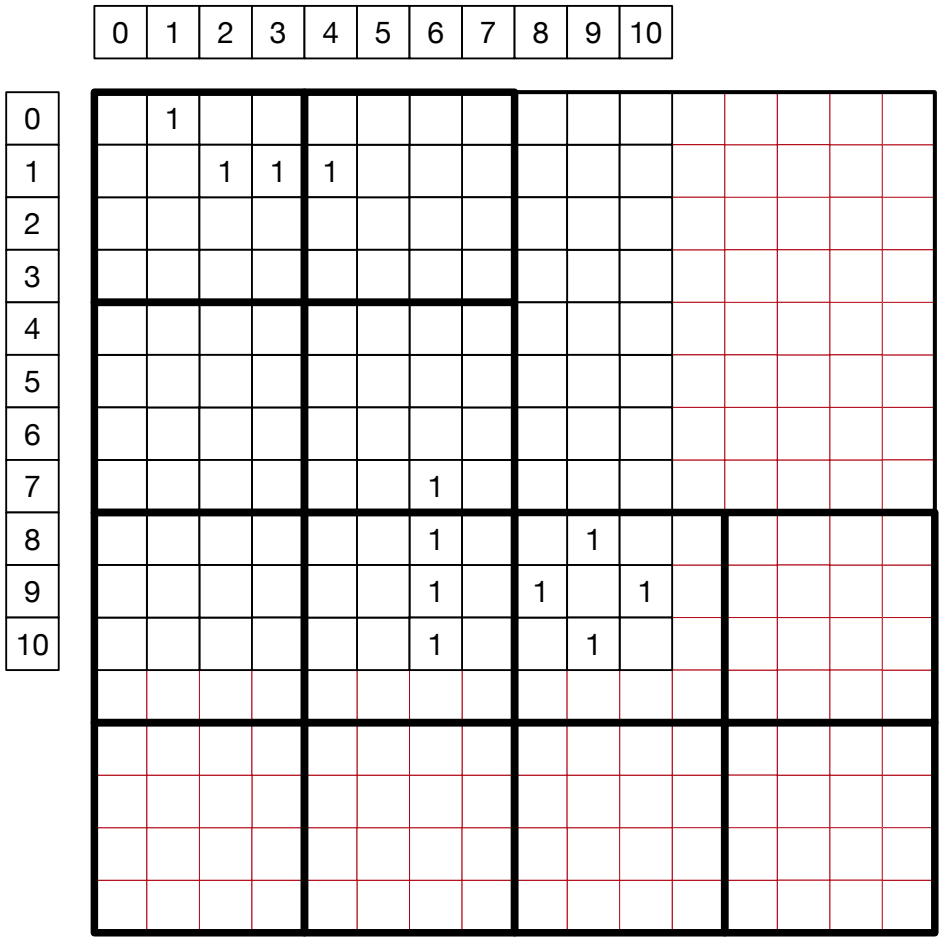
- 构建$k^2 tree$ 其中$k = 2$

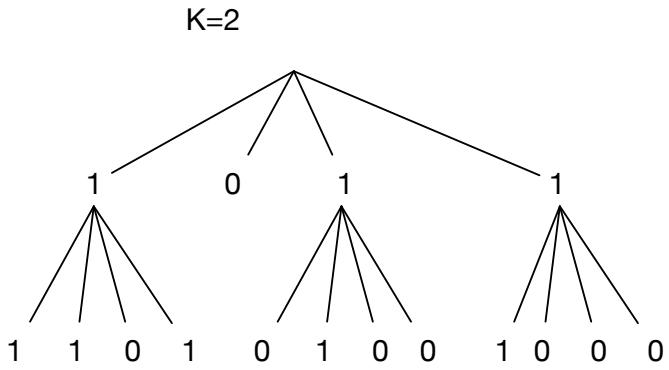




Construction

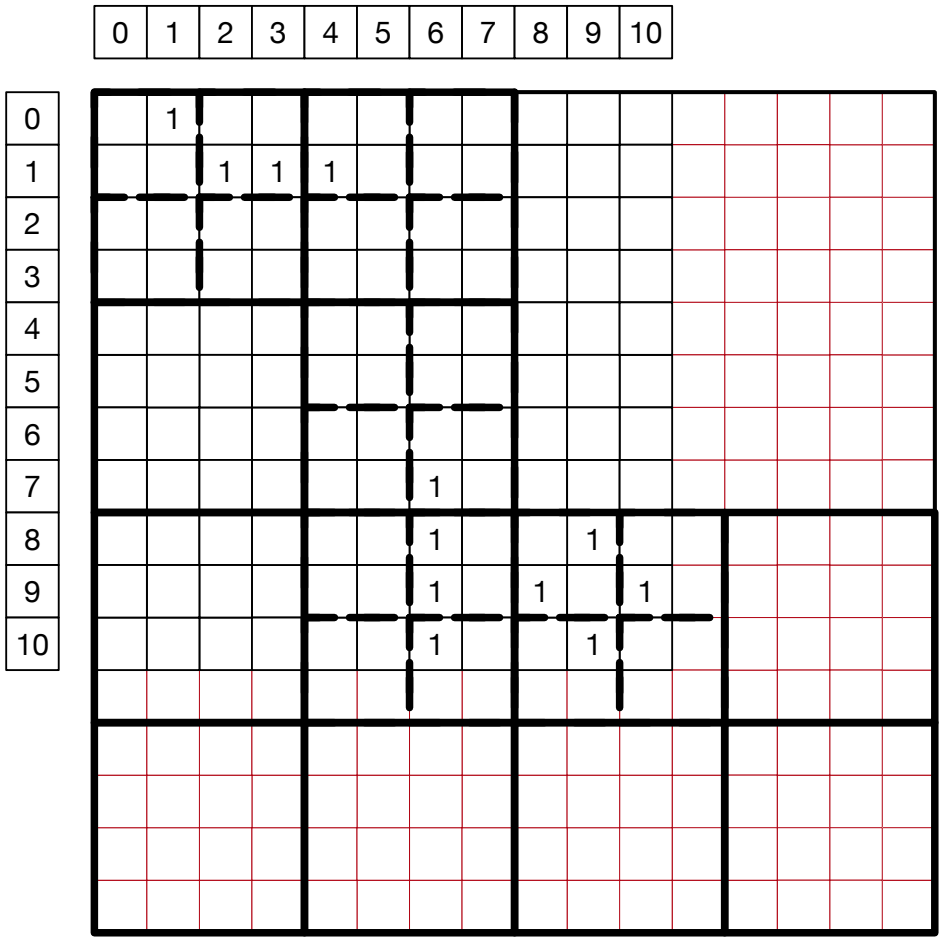
- 构建k^2tree 其中$k = 2$

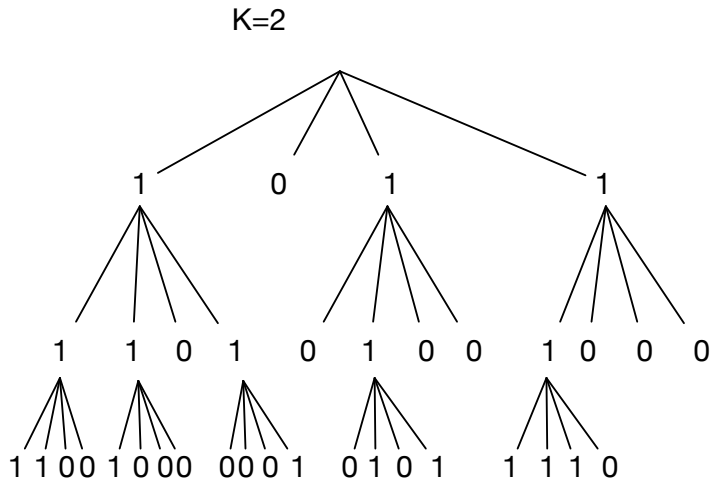




Construction

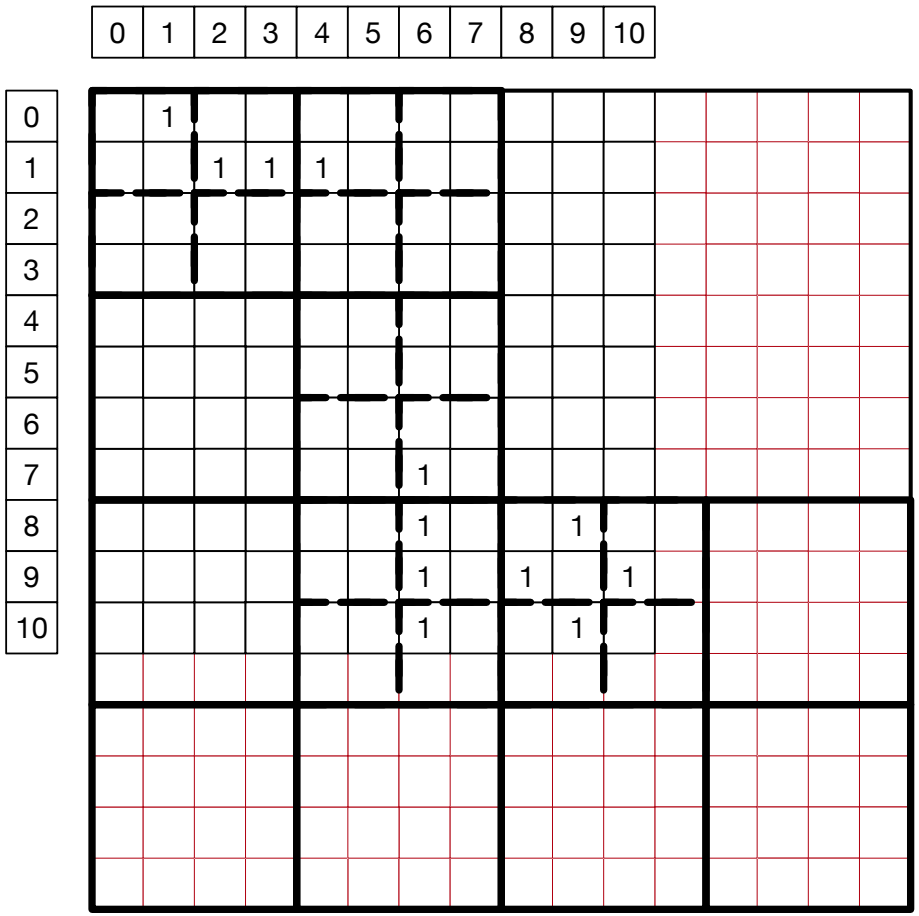
- 构建$k^2 tree$ 其中$k = 2$

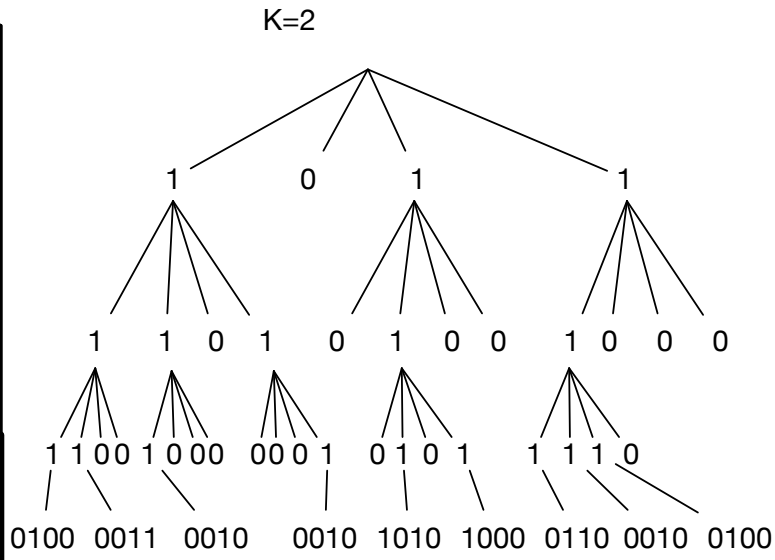




Construction

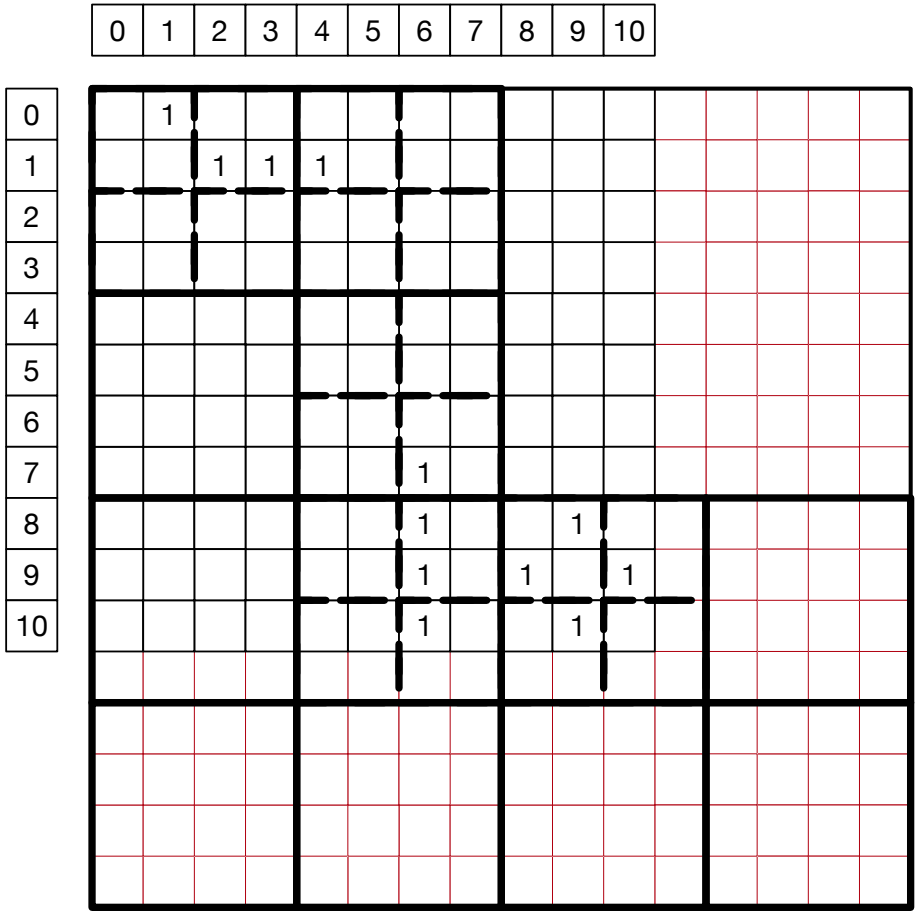
- 构建$k^2 tree$ 其中$k = 2$

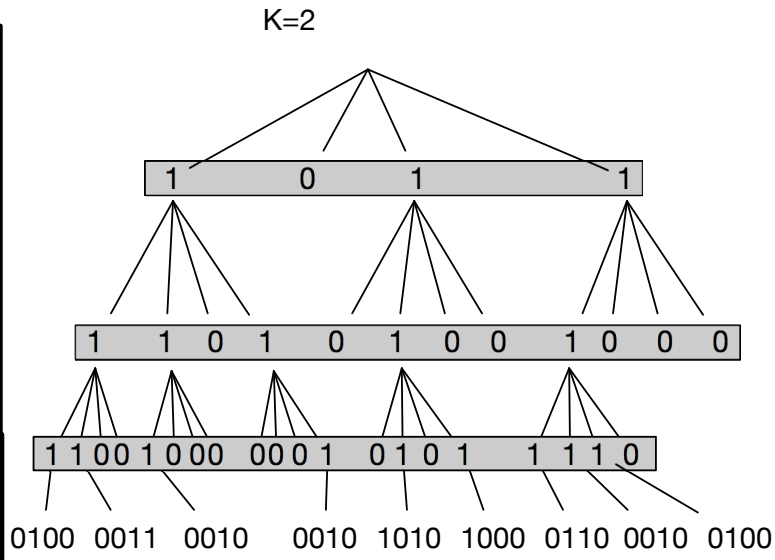




Construction

- 构建$k^2 tree$ 其中$k = 2$

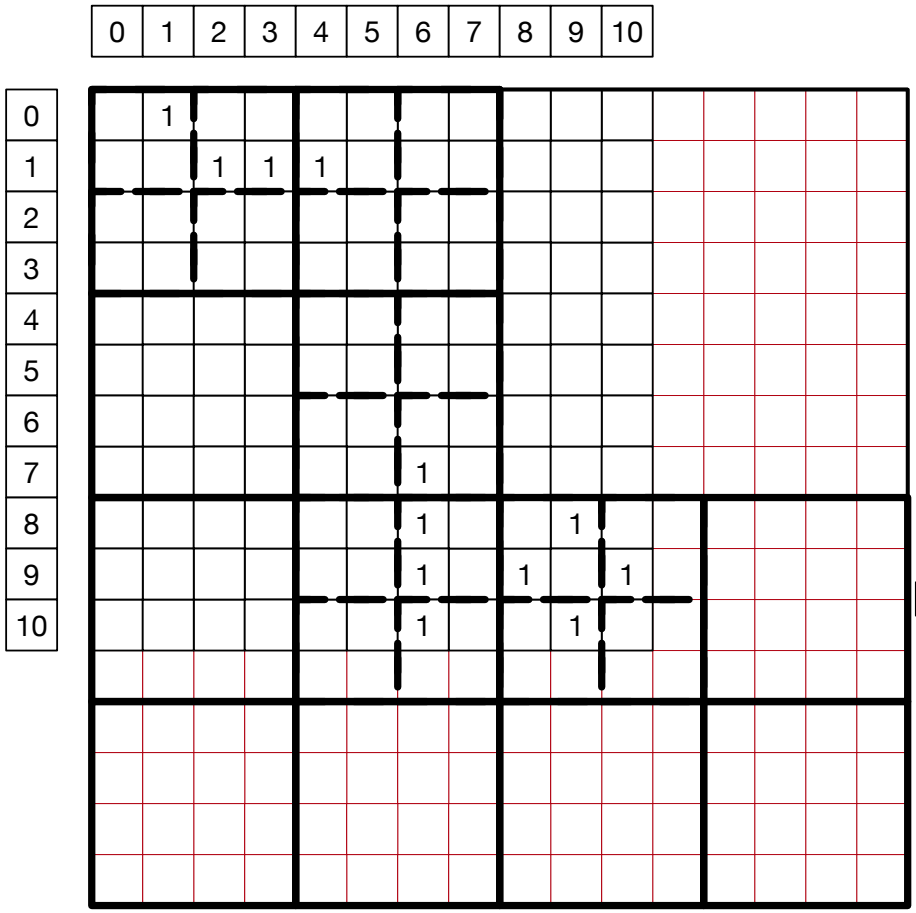


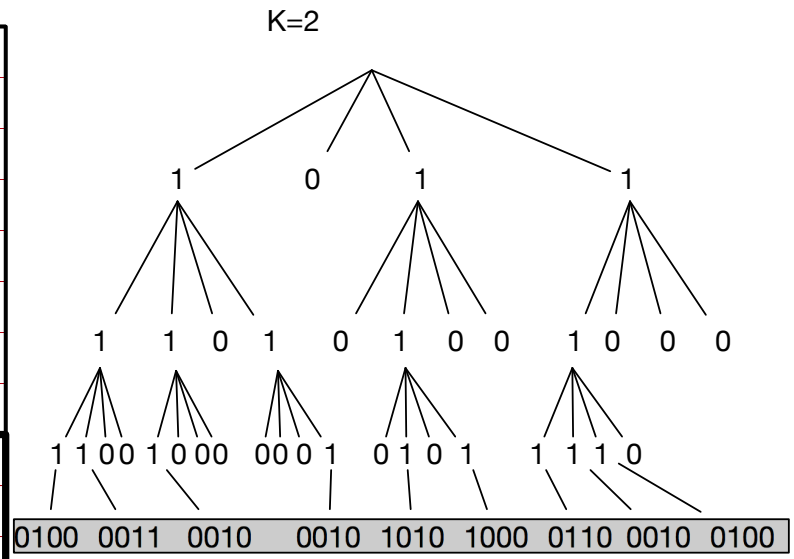


T=1011 1101 0100 1000 1100 1000 0001 0101 1110

Construction

- 构建$k^2 tree$ 其中$k = 2$



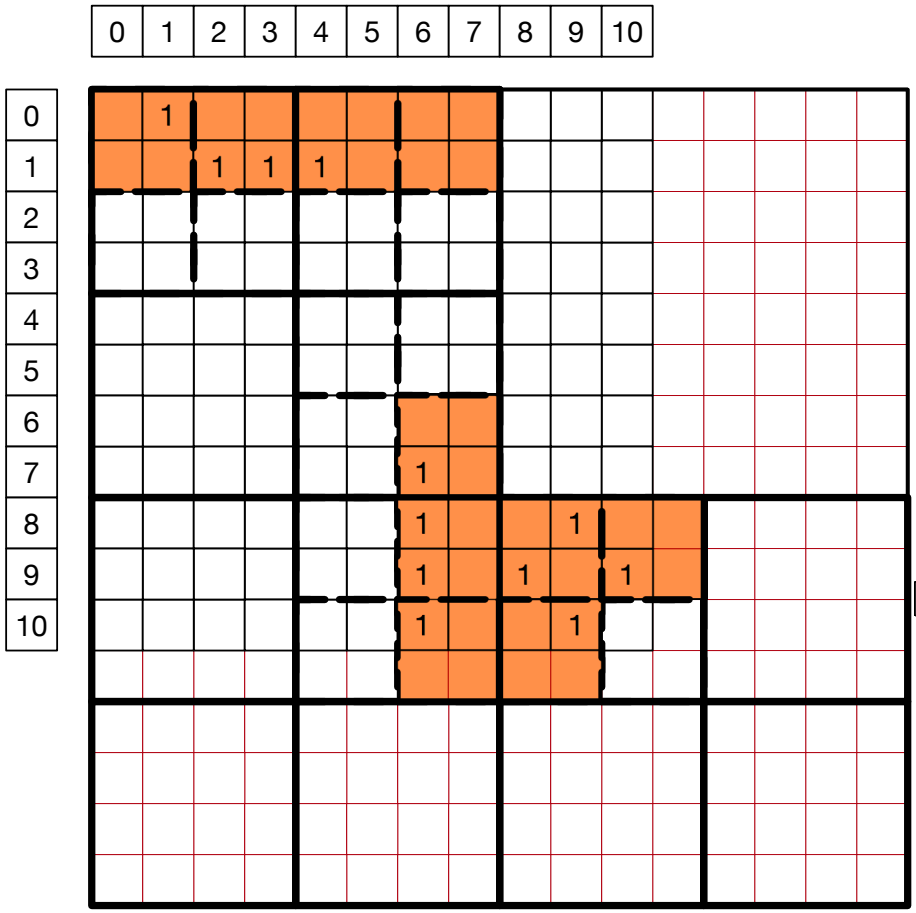


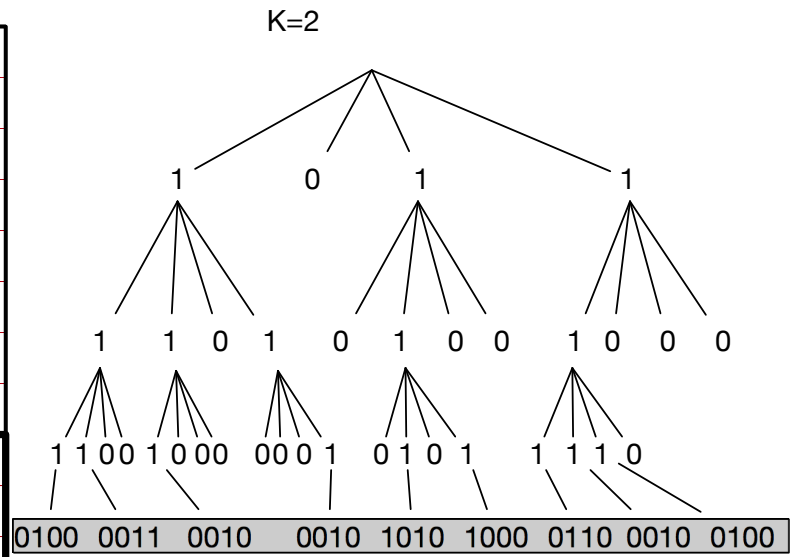
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Construction

- 构建$k^2 tree$ 其中$k = 2$



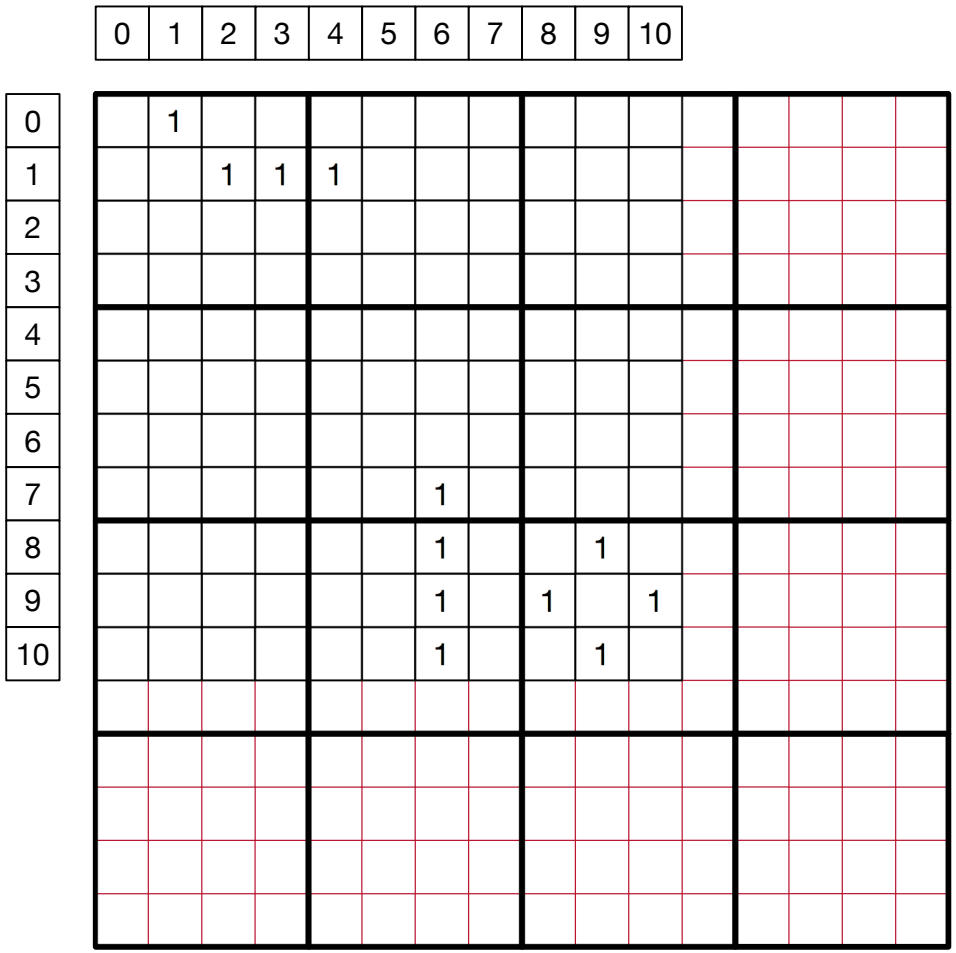


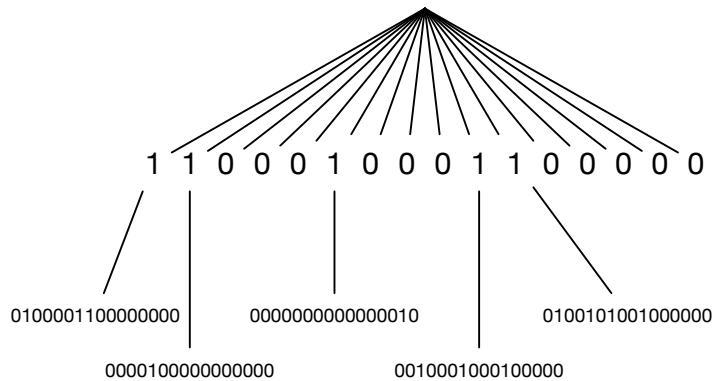
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Construction

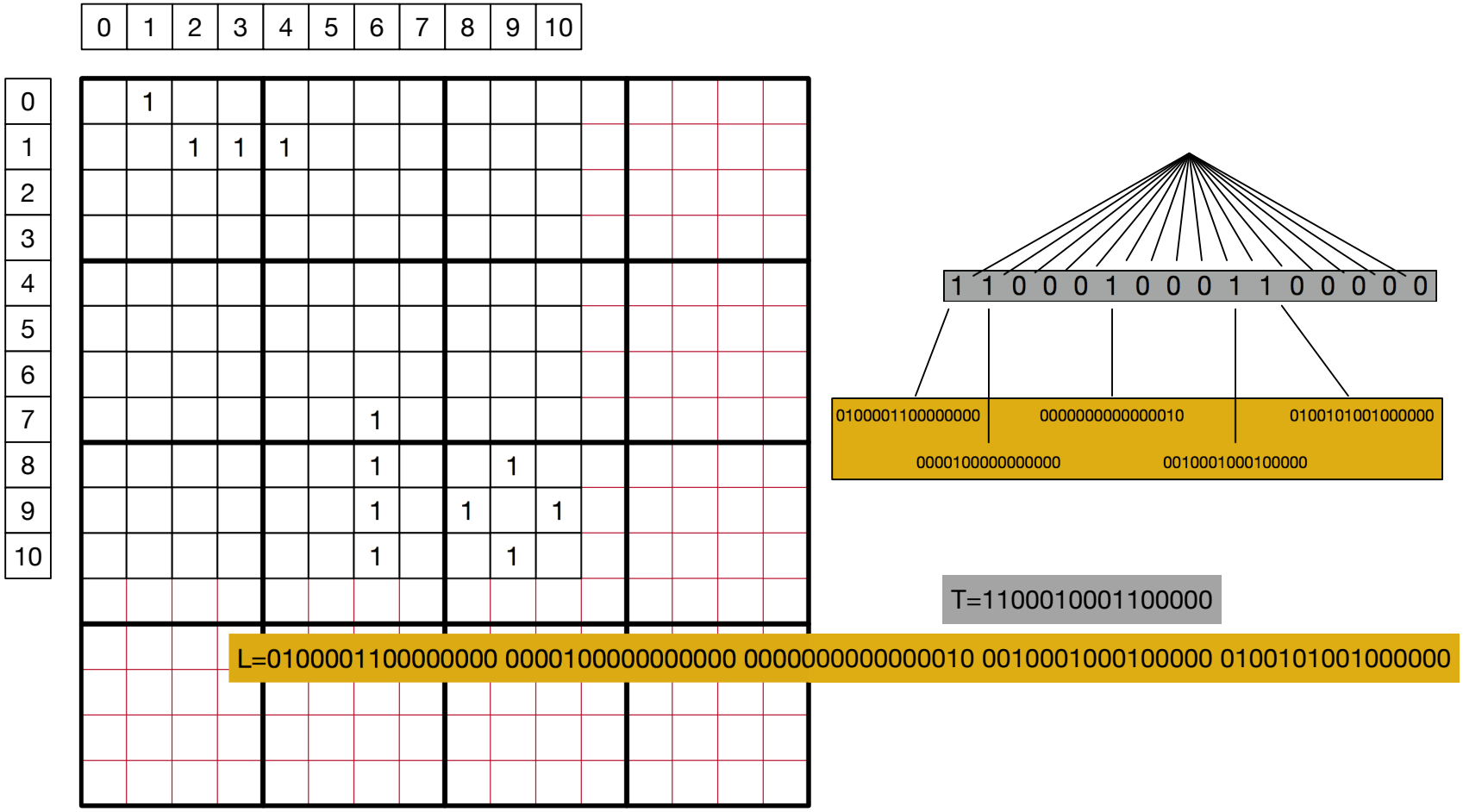
- 构建$k^2 tree$ 其中$k = 4$





Construction

- 构建$k^2 tree$ 其中$k = 4$

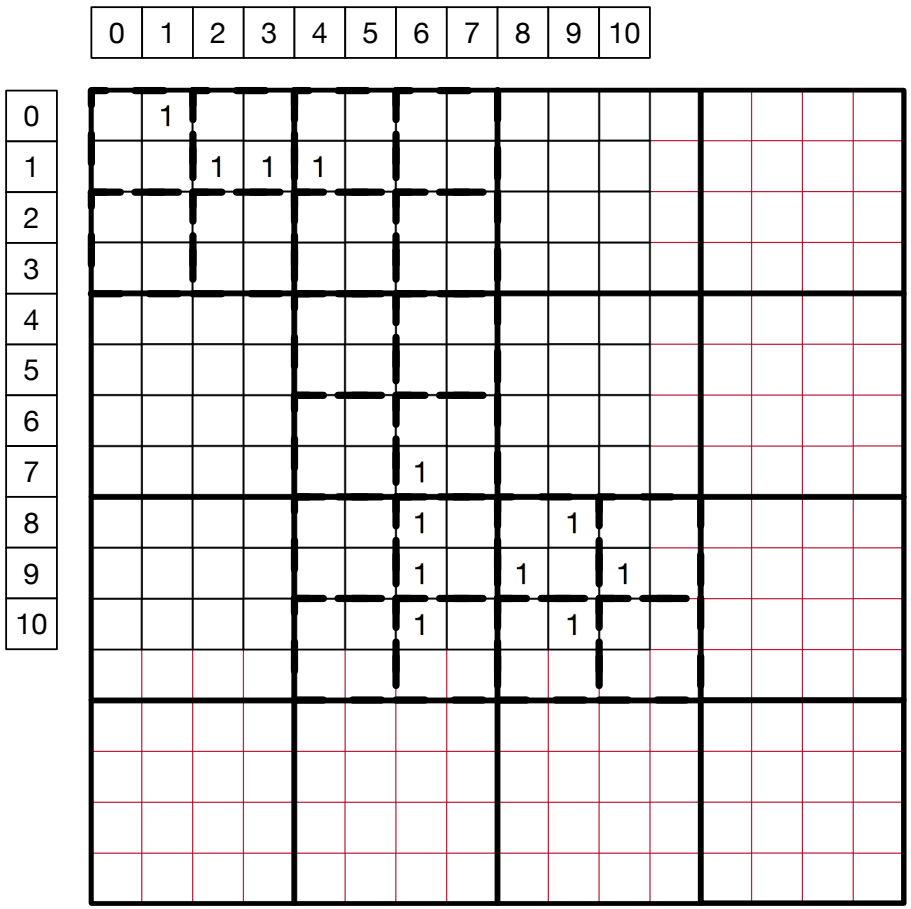


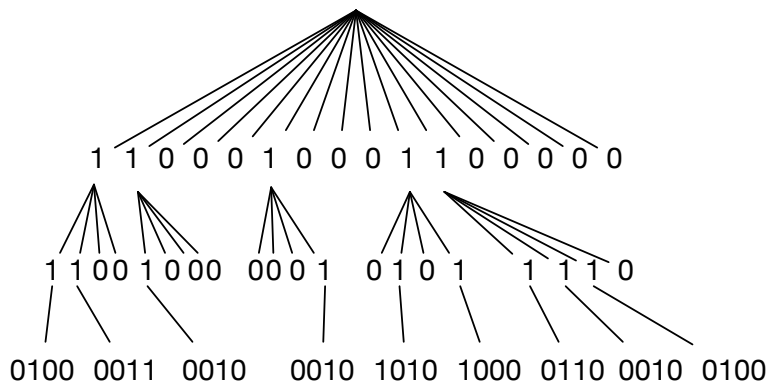
Construction

- 构建k^2tree 其中$k = 2$
- 构建k^2tree 其中$k = 4$
- k越小，树越深，占空间越小($|T \parallel L| = 72$)
- k越大，树越浅，占空间越大($|T \parallel L| = 96$)，但查询速度也越快

Hybrid K

- 构建*Hybrid* k^2*tree* 其中$k = 4$ *or* 2





Space analysis

- **Worst cast?**

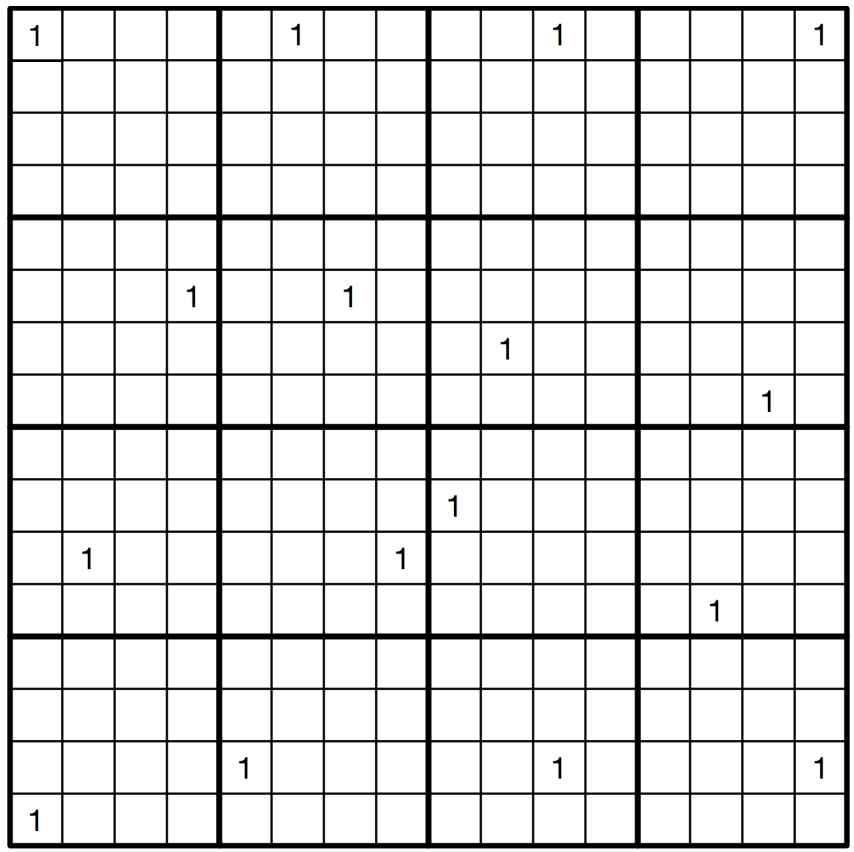
Space analysis

- **Worst cast?**

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Space analysis

- **Worst cast**
- 寻找任意两条边的路径都不相同，树的中间节点是满的，叶子层的节点每个k^2只有一个1



Space analysis

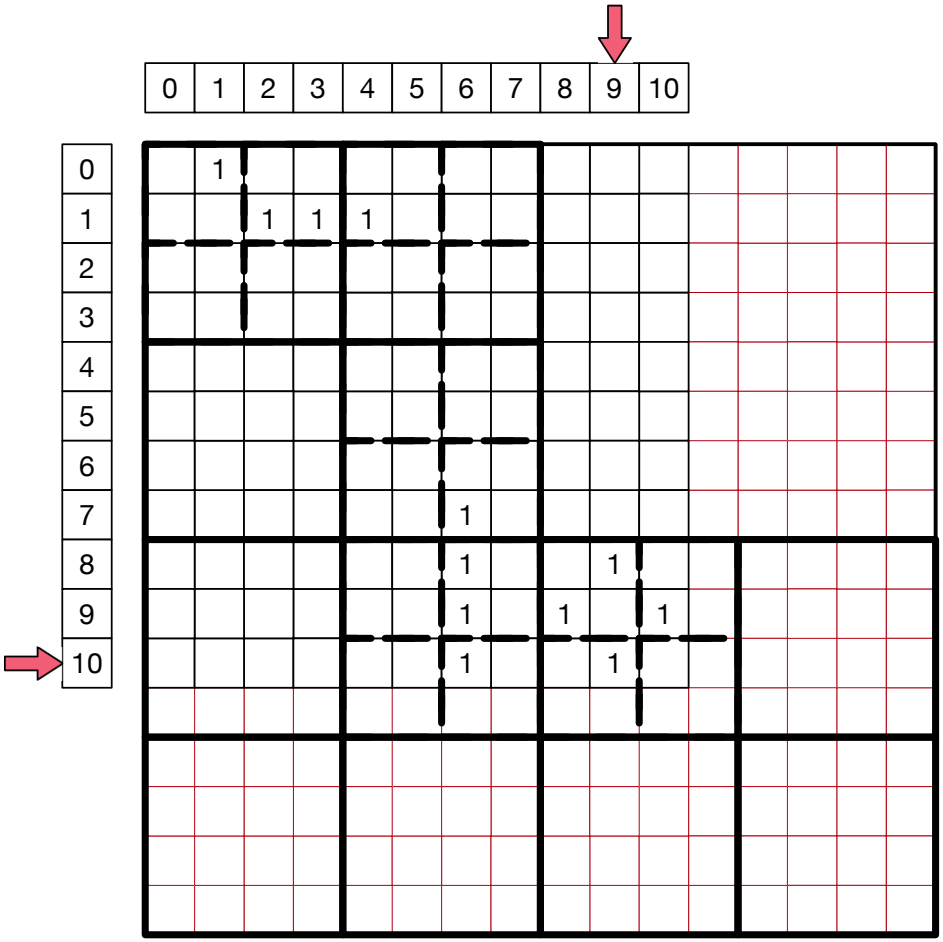
- **Worst cast**
- 记图共有n个节点，m条边
- 建树完成后高度h为$h = \lceil \log_{k^2} n^2 \rceil$
- 倒数第二层有m个节点，所以除了叶子节点共有$\lfloor \log_{k^2} m \rfloor$层
- 所以总共需要

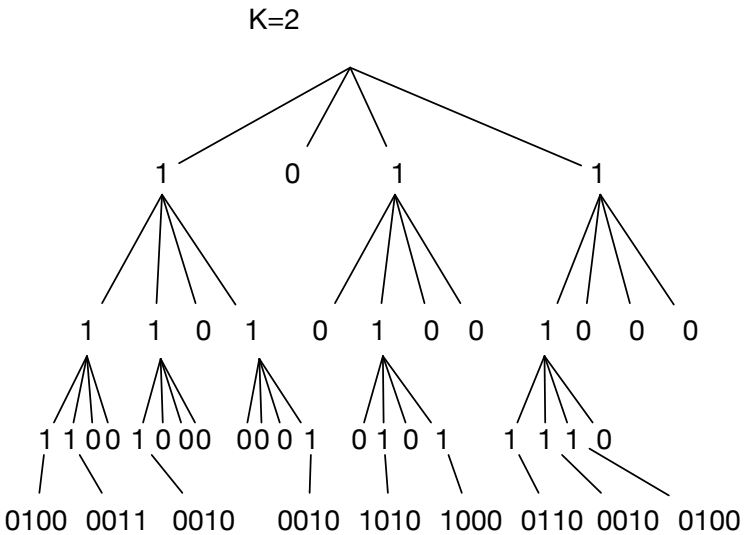
$$\left(\sum_{l=1}^{\lfloor \log_{k^2} m \rfloor} k^{2l} \right) + k^2 m (\lceil \log_{k^2} n^2 \rceil - \lfloor \log_{k^2} m \rfloor) = k^2 m \left(\log_{k^2} \frac{n^2}{m} + O(1) \right)$$

- bits. 当$k = 2$时为$2m \log_2 \frac{n^2}{m} + O(m)$，接近信息论给出的最小压缩值的两倍

Check Link

- **例**：检测有没有从10到9的边



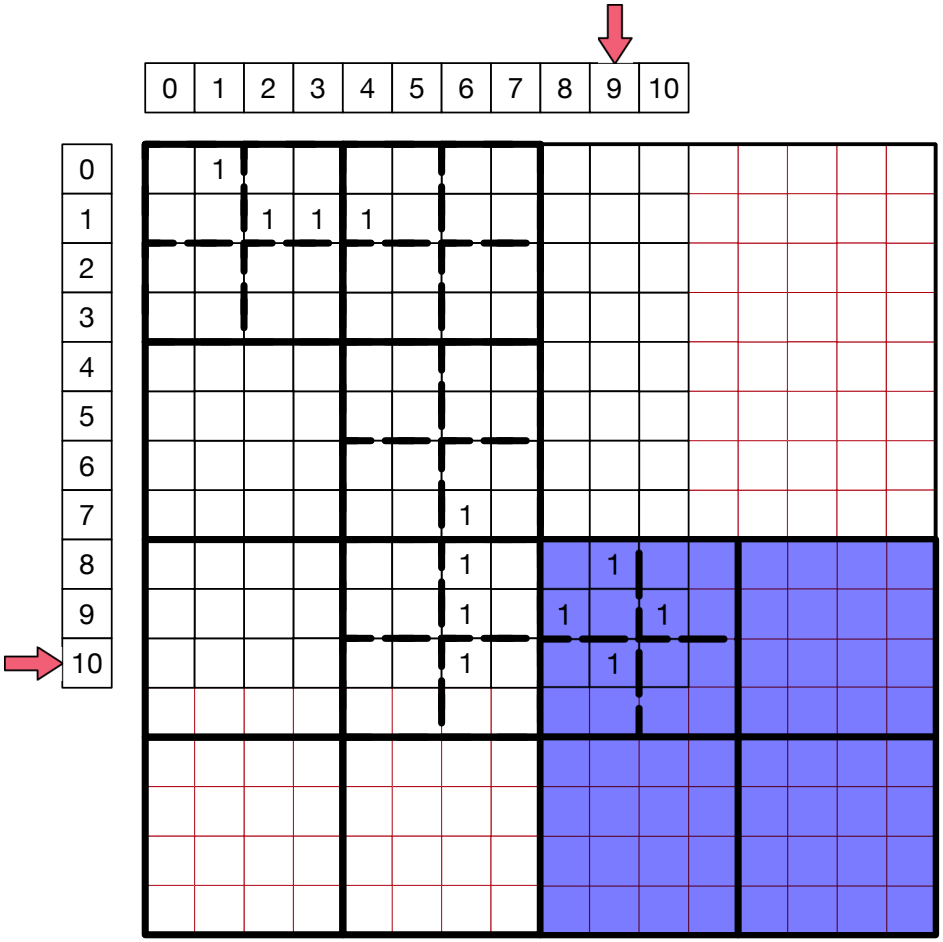


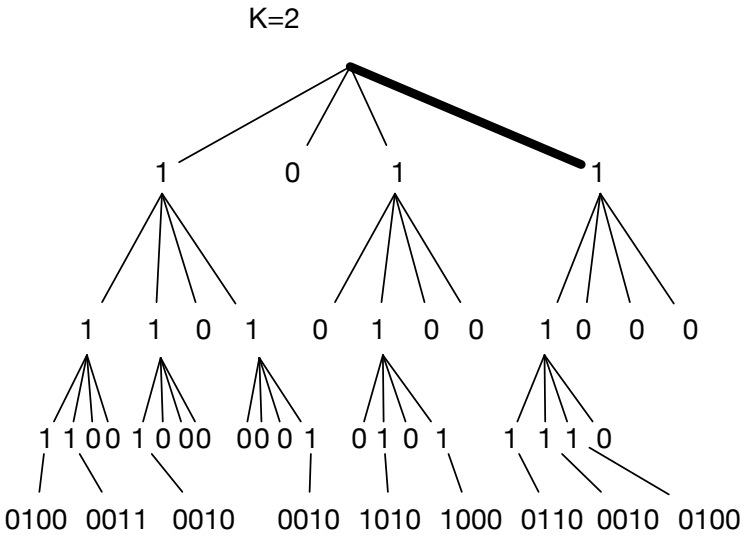
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Check Link

- **例**：检测有没有从10到9的边



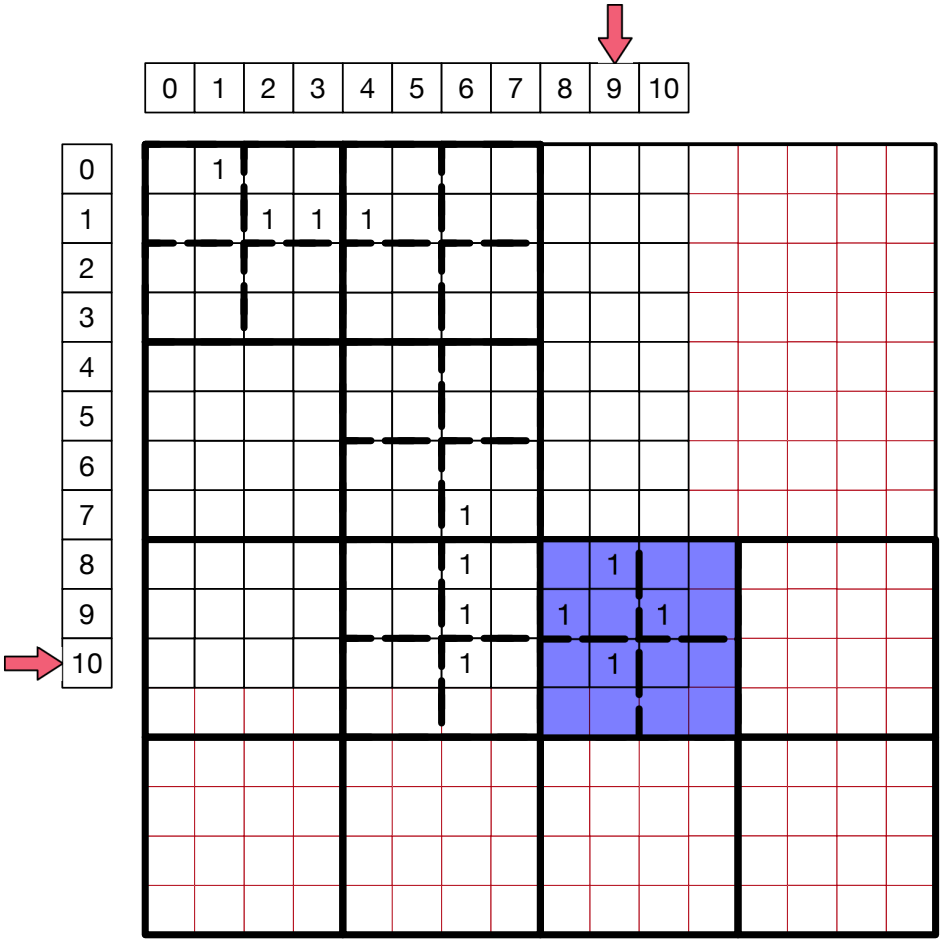


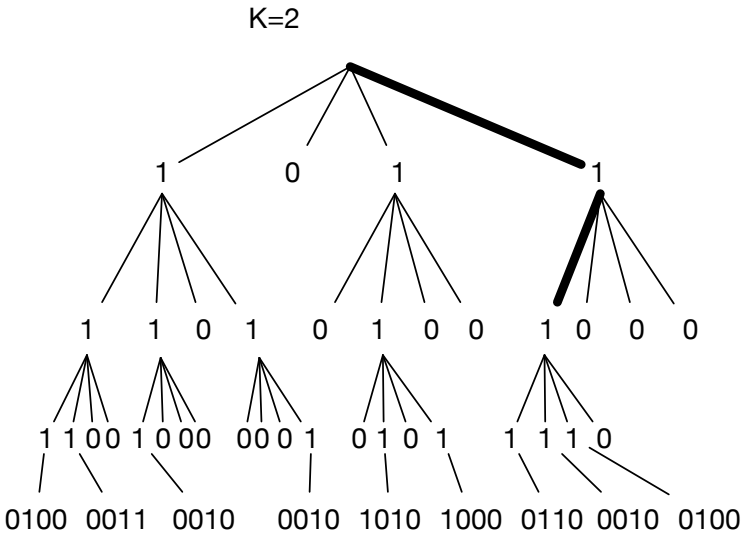
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Check Link

- **例**：检测有没有从10到9的边



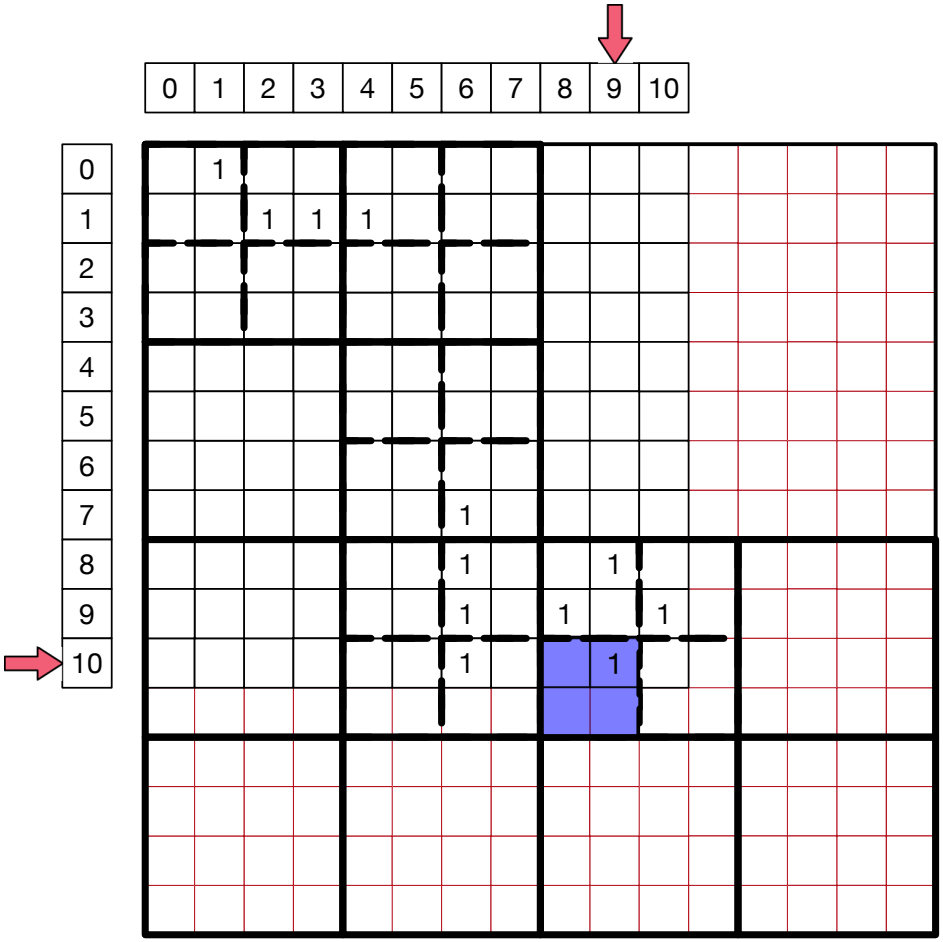


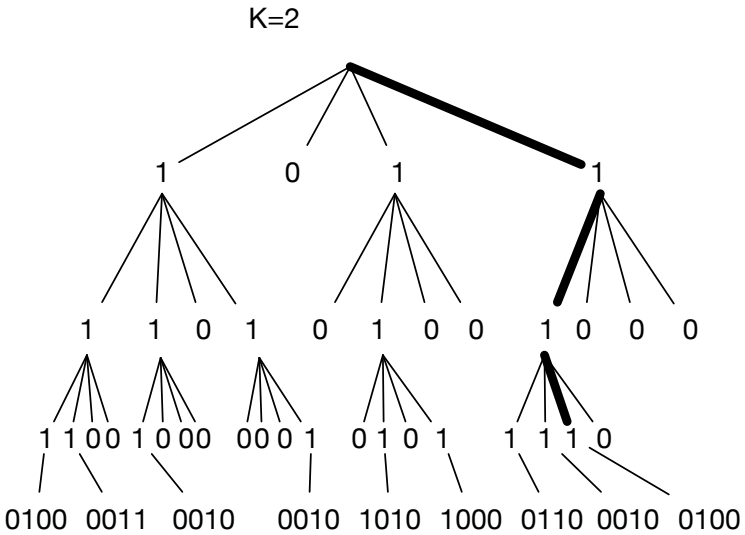
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Check Link

- **例**：检测有没有从10到9的边



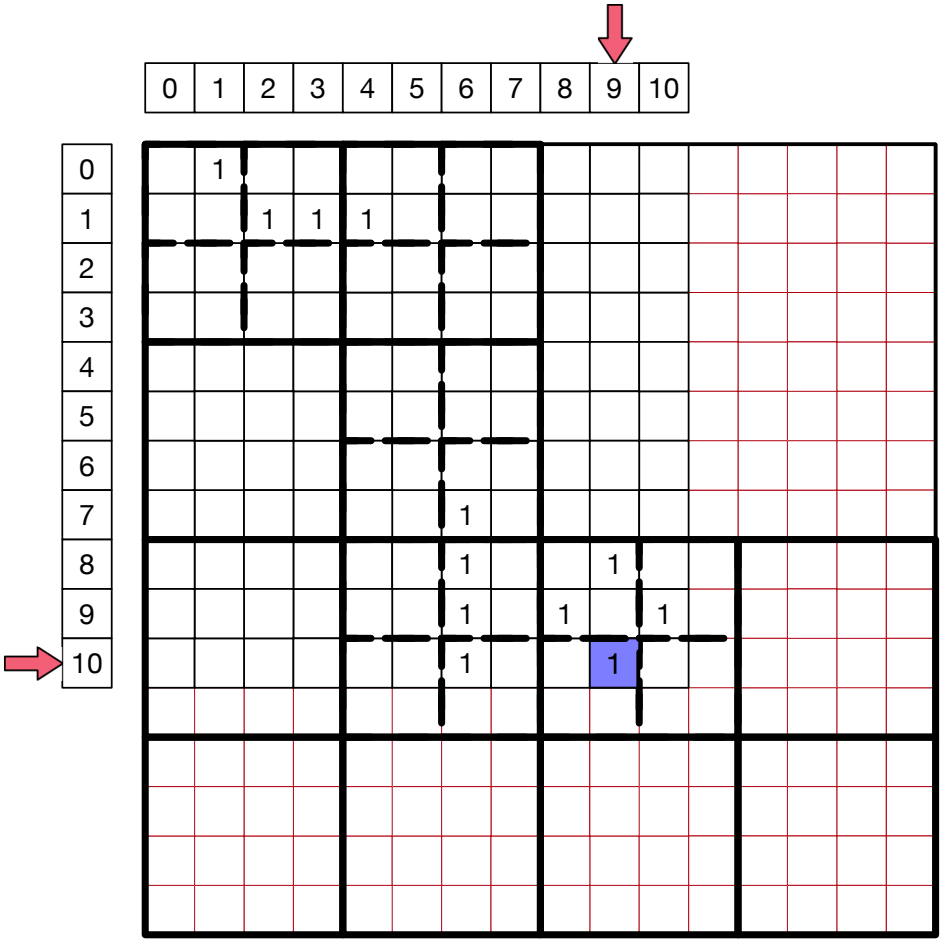


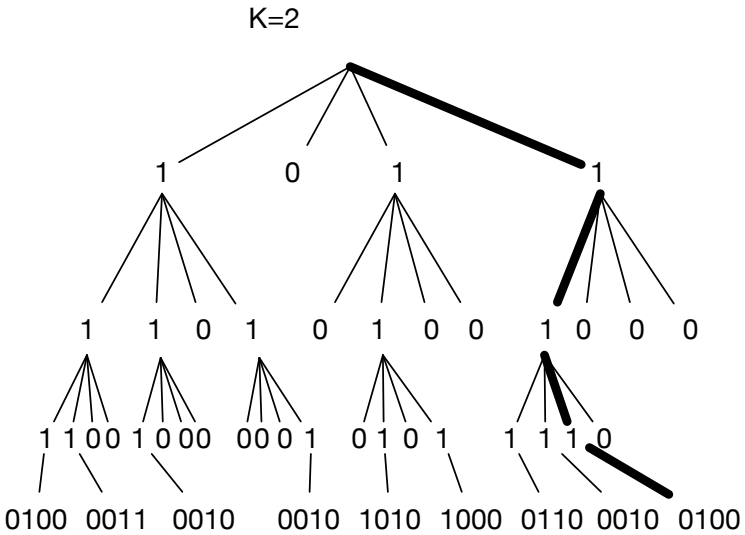
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Check Link

- **例**：检测有没有从10到9的边



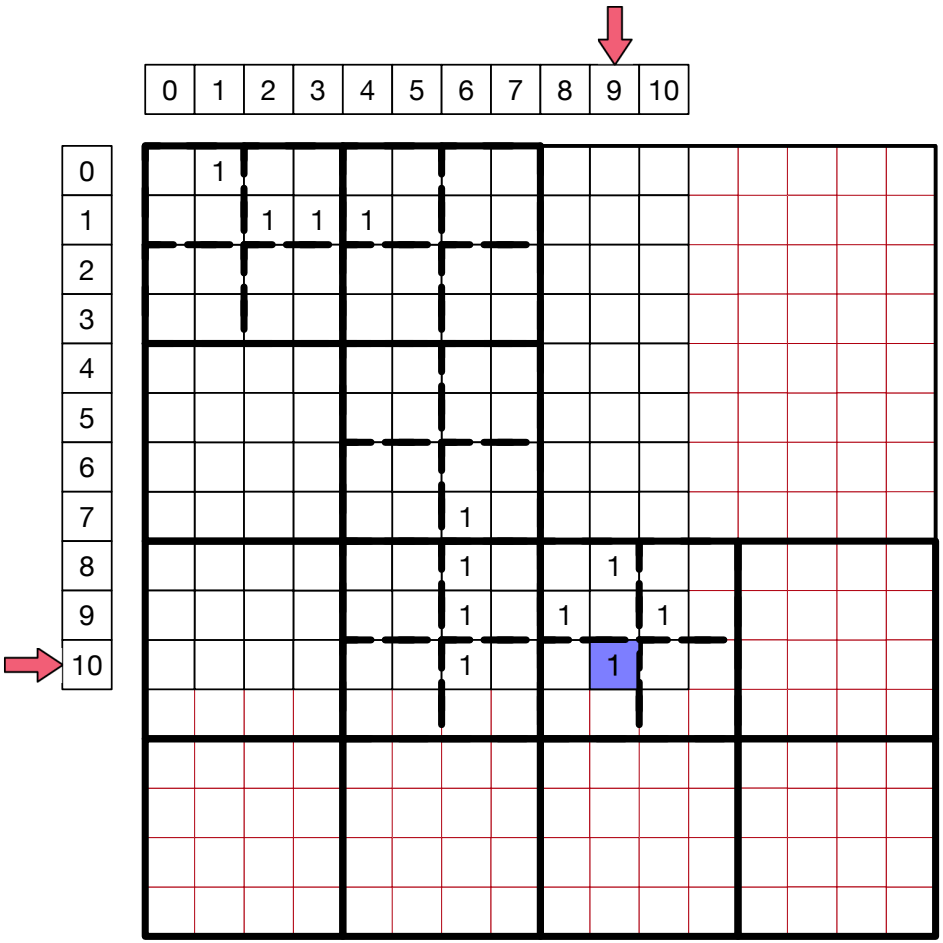


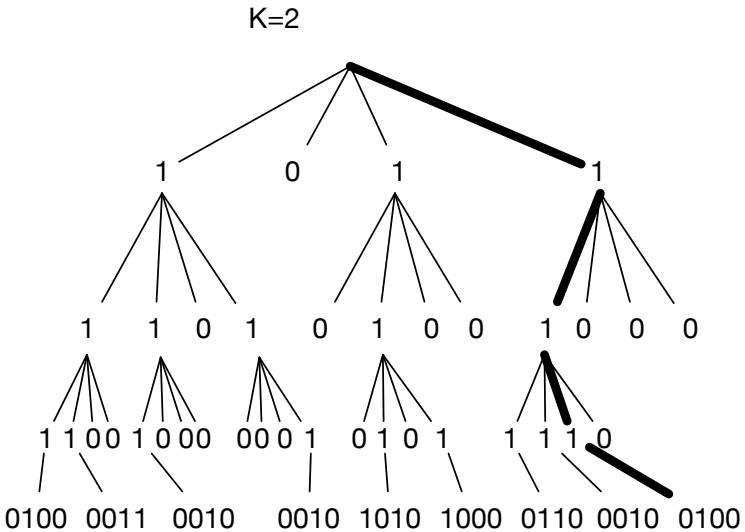
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Check Link

- 例：检测有没有从10到9的边





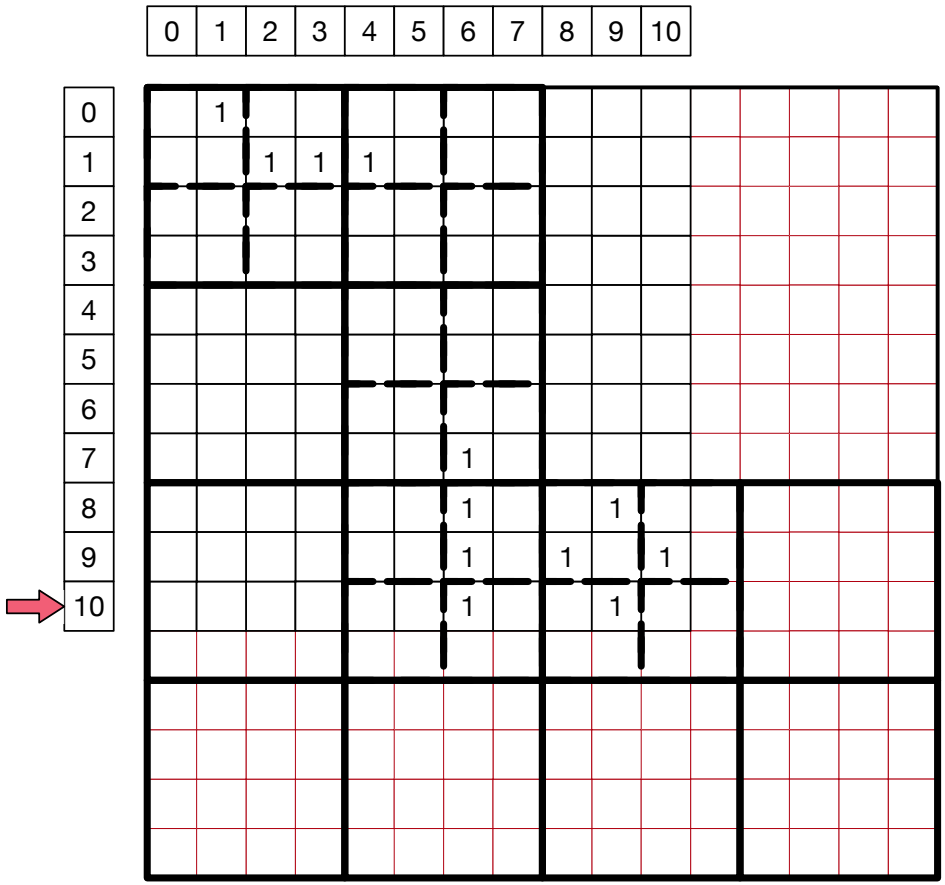
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

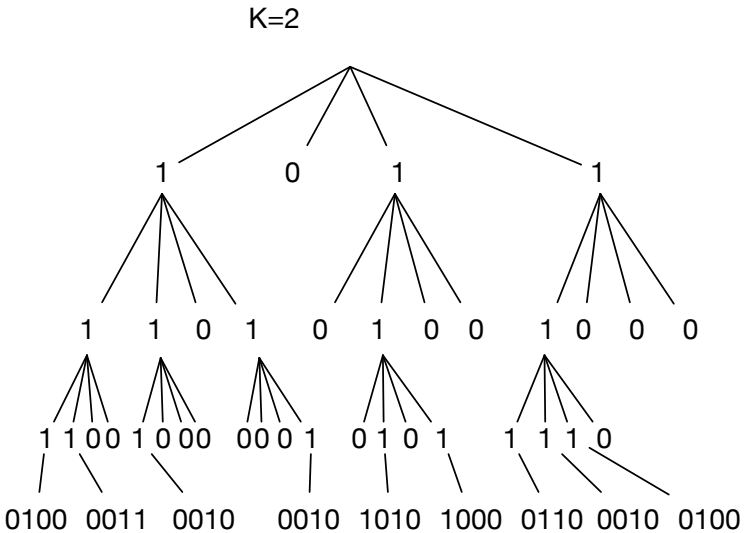
L=0100 0011 0010 0010 1010 1000 0110 0010 0100

- 时间复杂度：$O(\log_k n)$

Successor

- 例：寻找节点10的出节点（孩子结点）





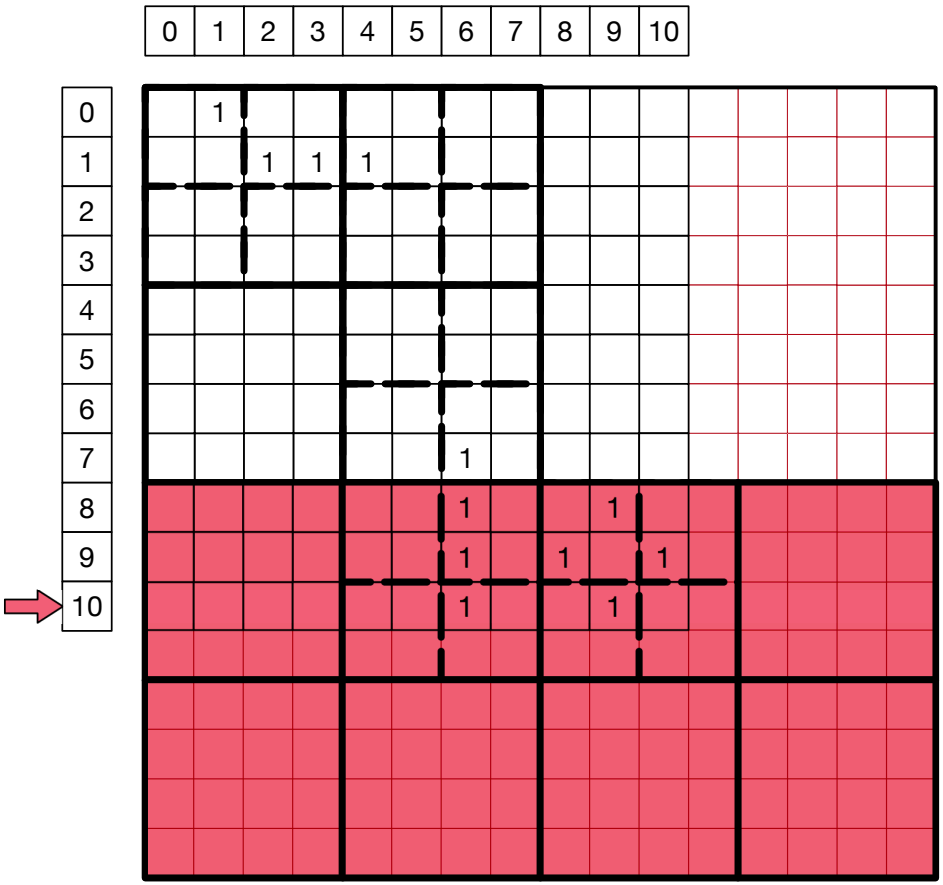
0 4 8 12 16 20 24 28 32

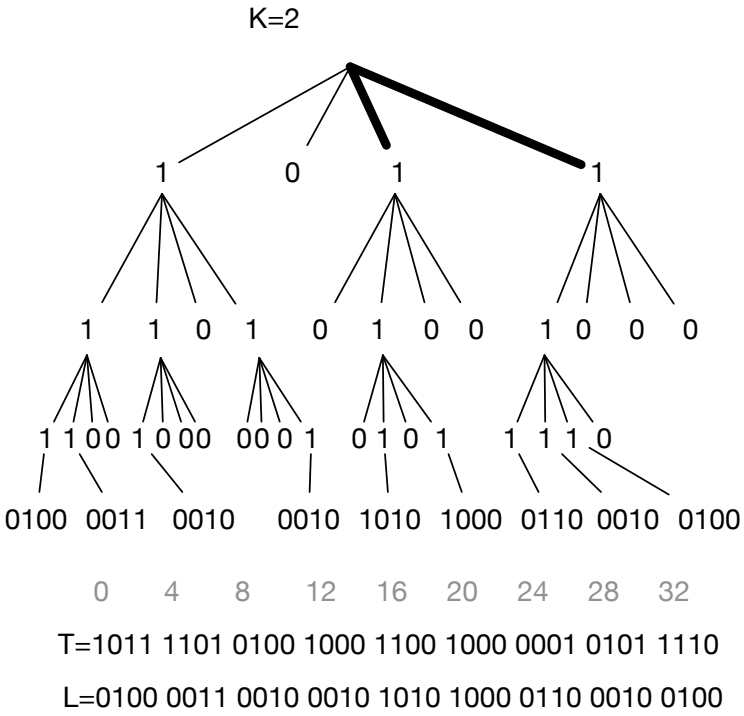
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Successor

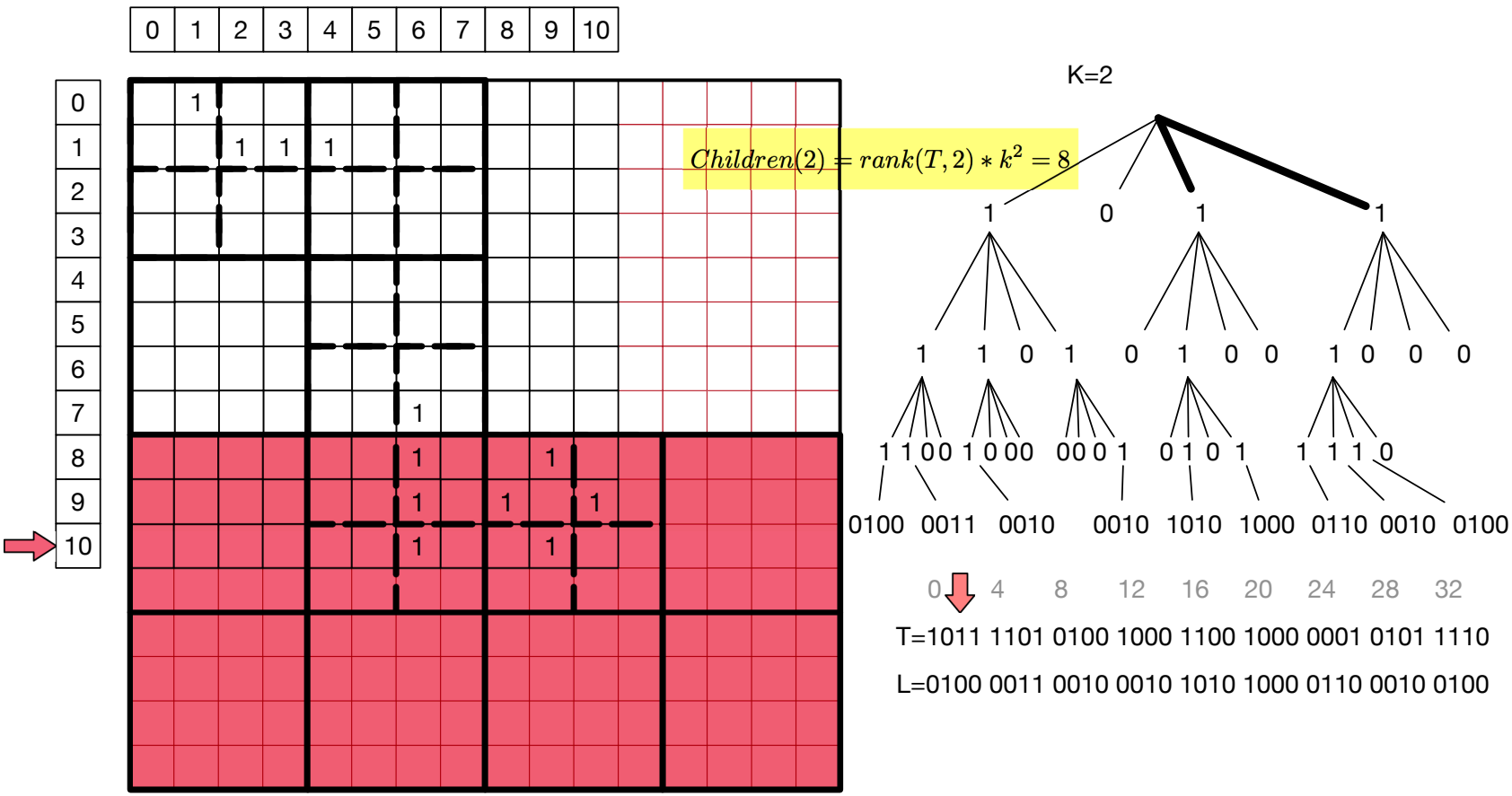
- 例：寻找节点10的出节点（孩子结点）





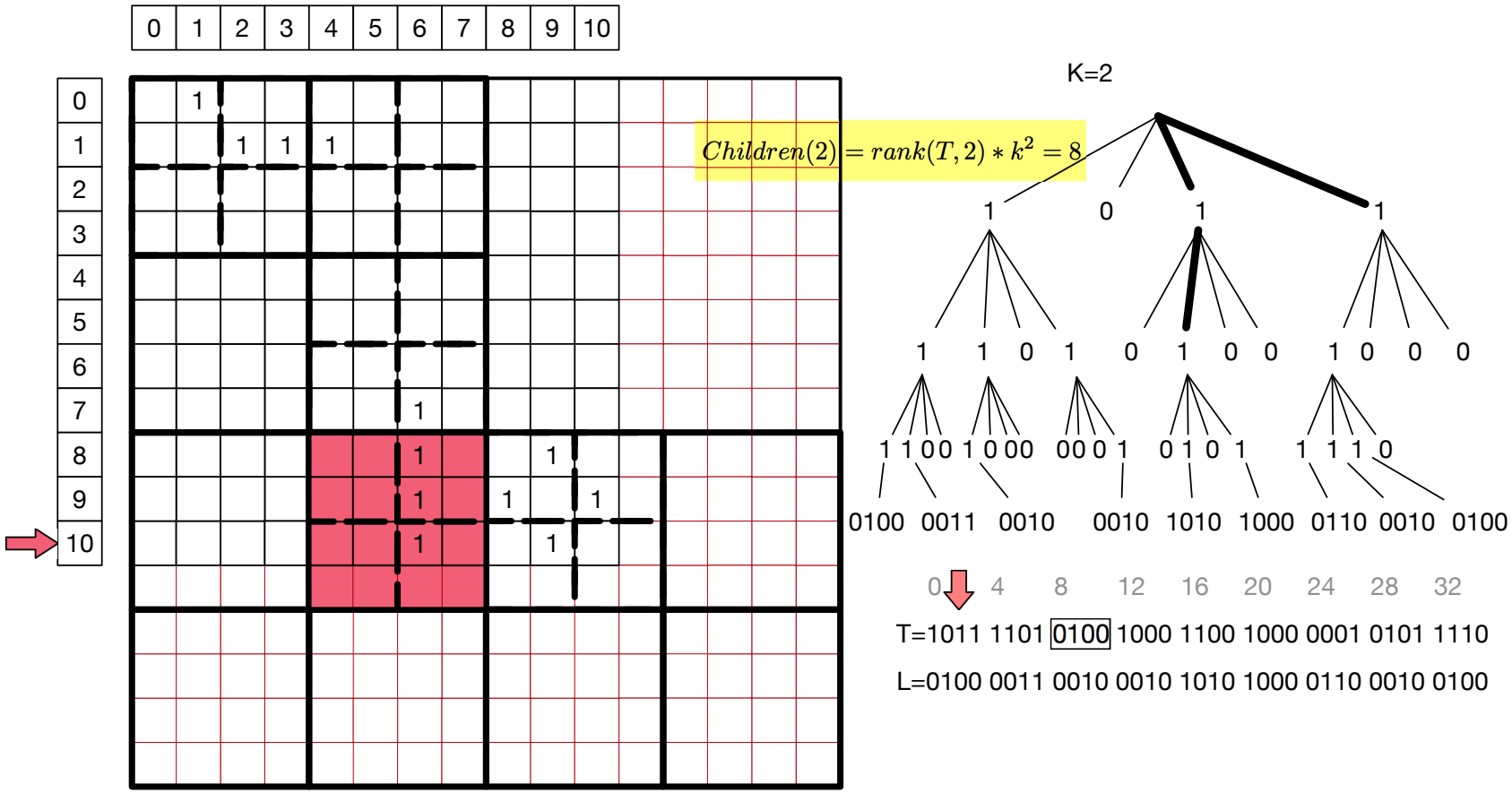
Successor

- 例：寻找节点10的出节点（孩子结点）



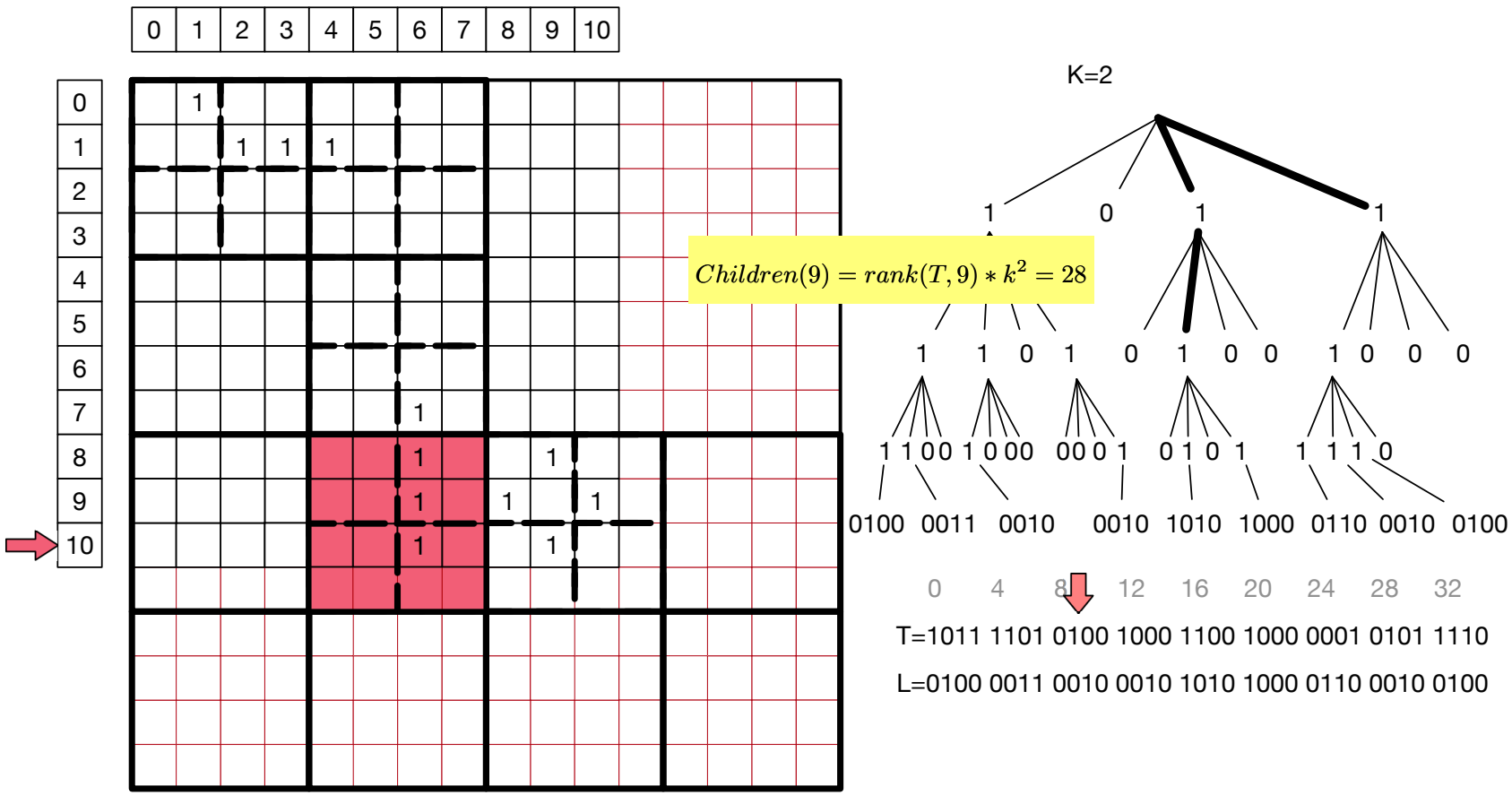
Successor

- 例：寻找节点10的出节点（孩子结点）



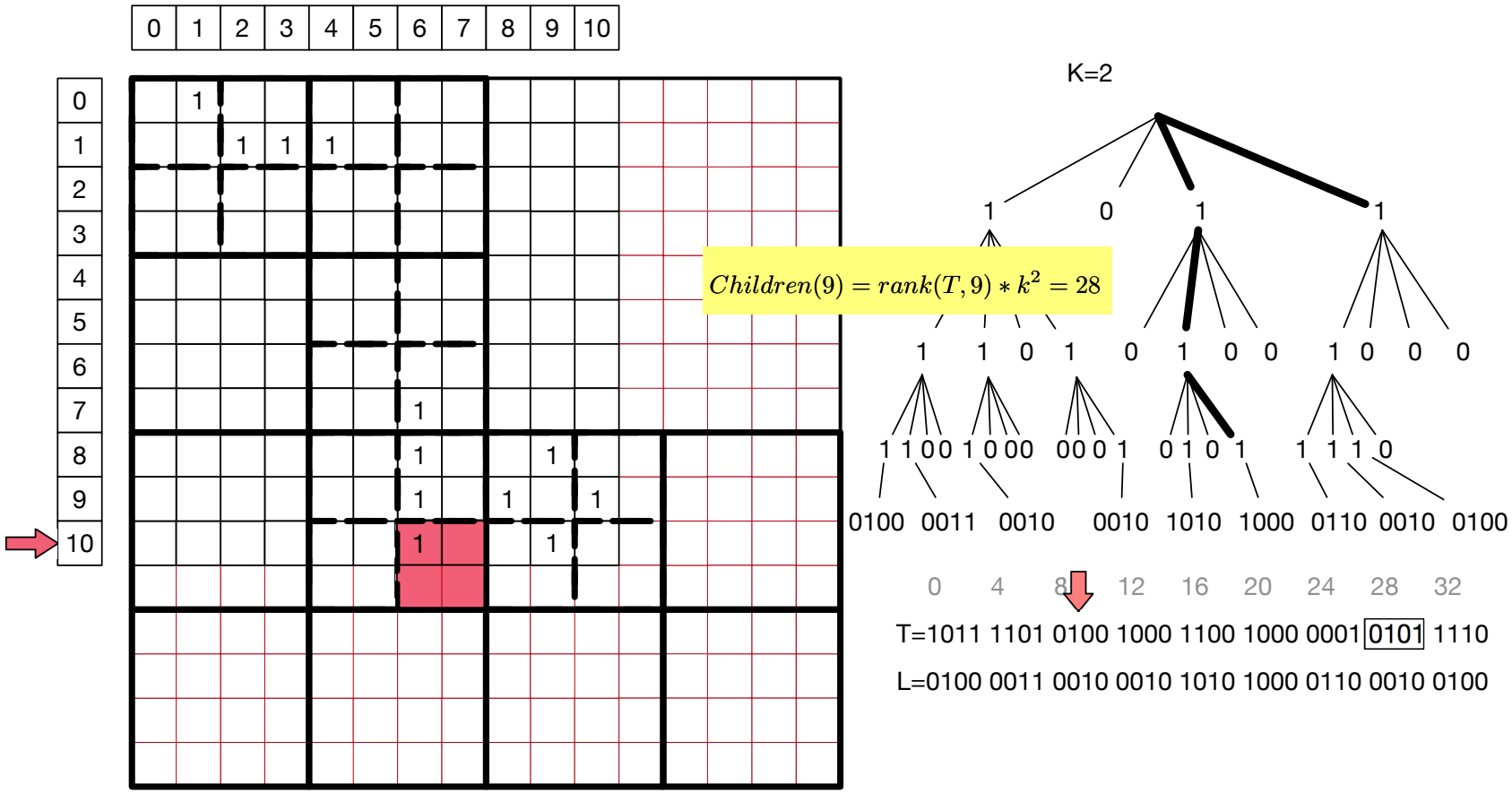
Successor

- 例：寻找节点10的出节点（孩子结点）



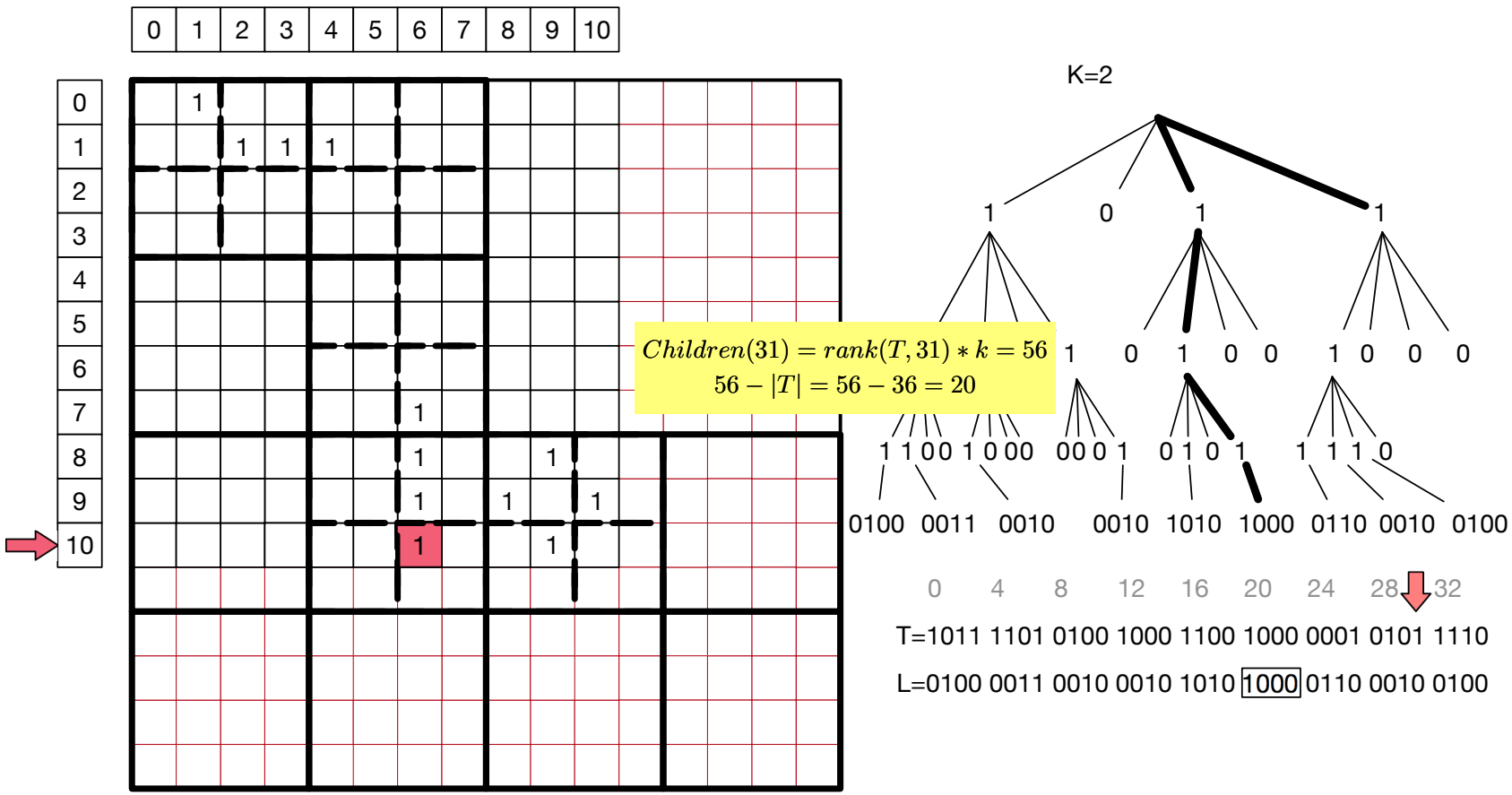
Successor

- 例：寻找节点10的出节点（孩子结点）



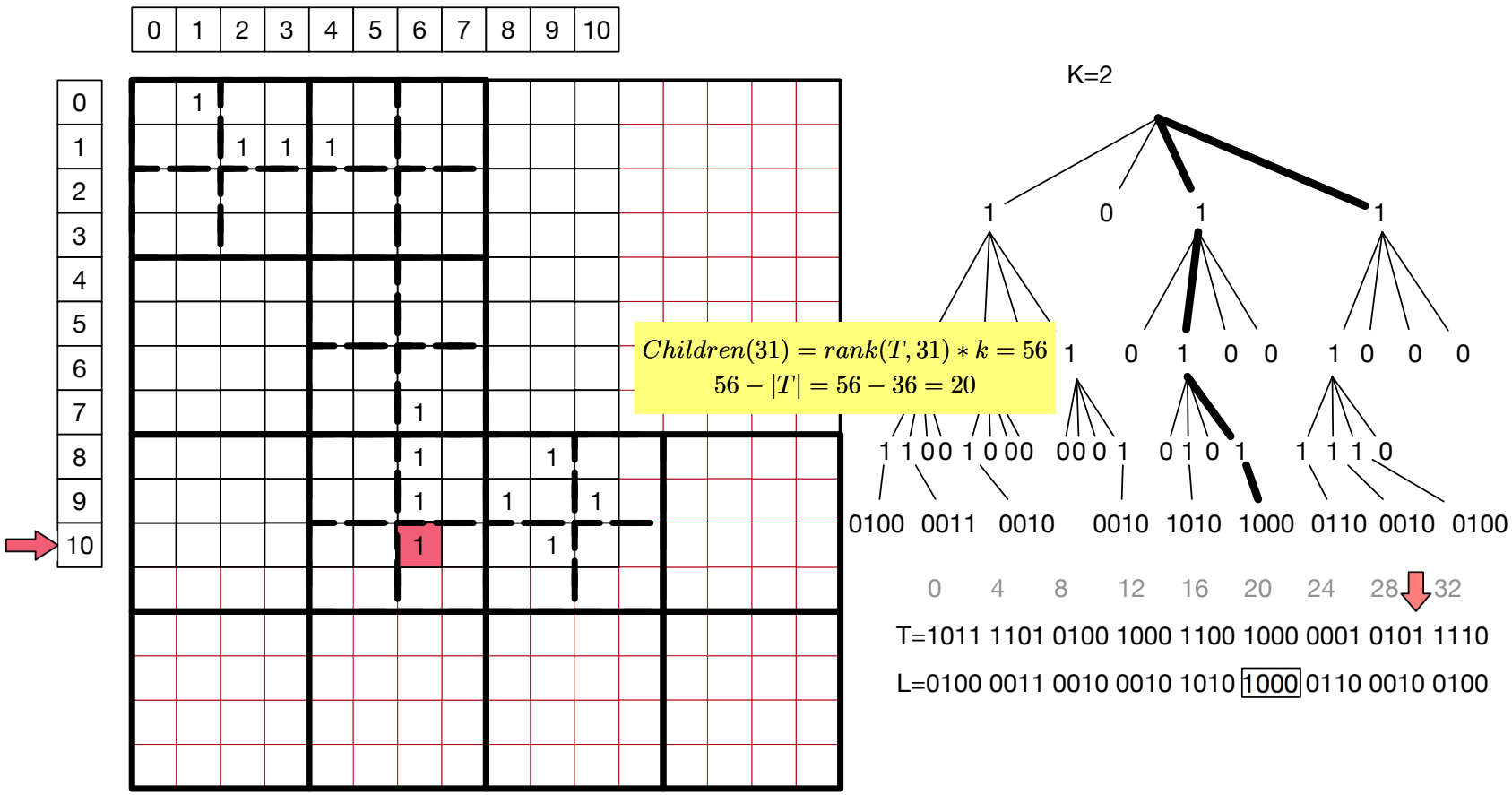
Successor

- 例：寻找节点10的出节点（孩子结点）



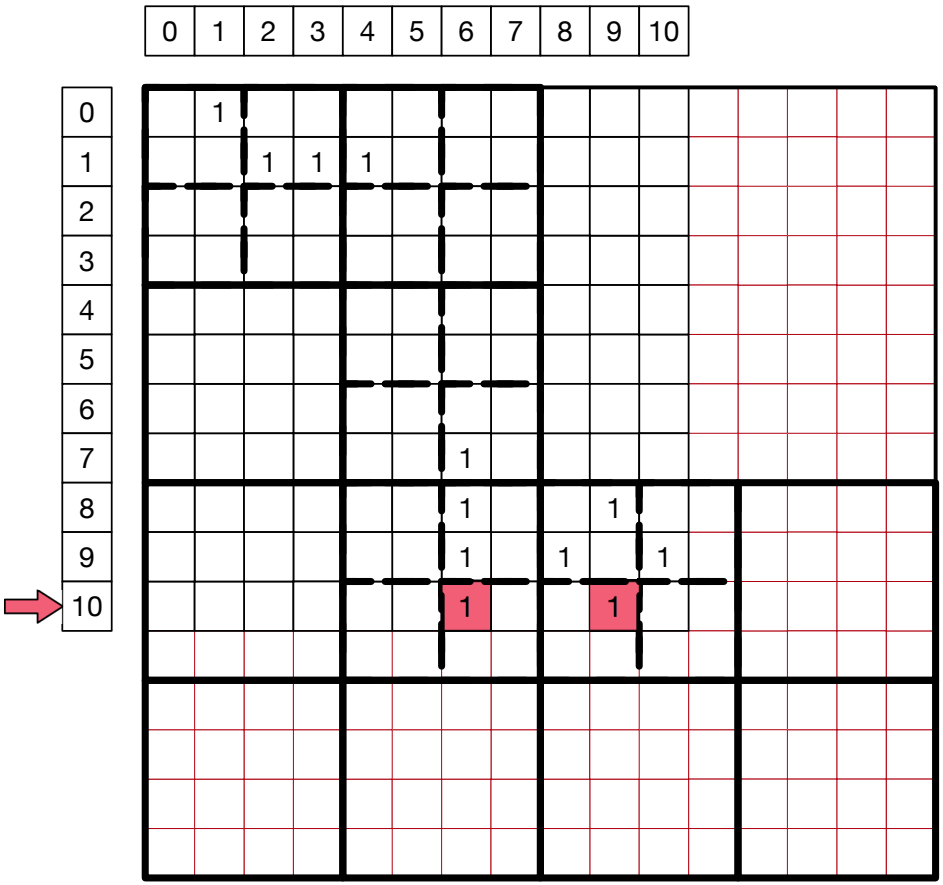
Successor

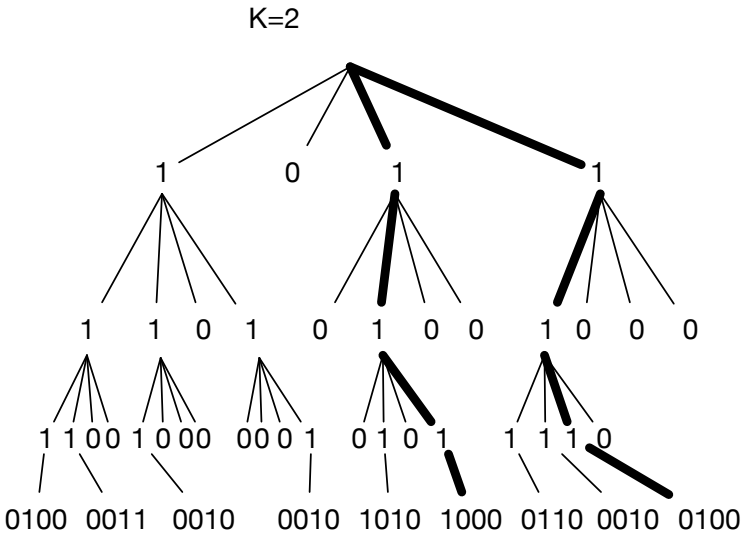
- 例：寻找节点10的出节点（孩子结点）



Successor

- 例：寻找节点10的出节点（孩子结点）





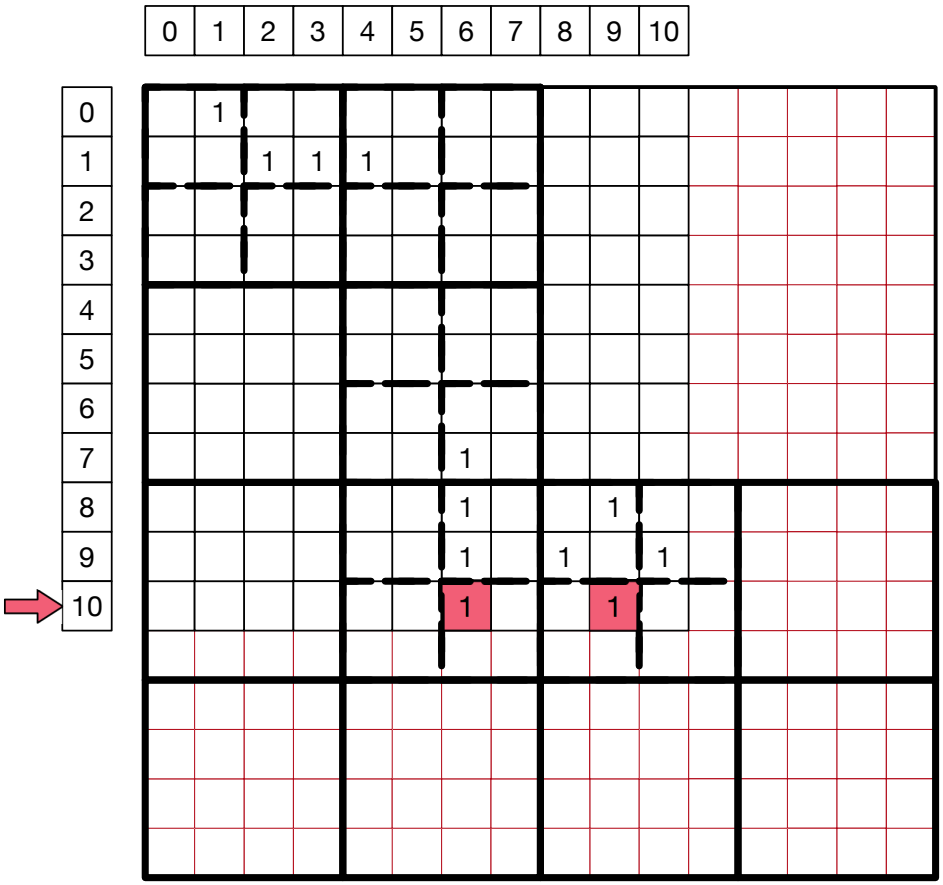
0 4 8 12 16 20 24 28 32

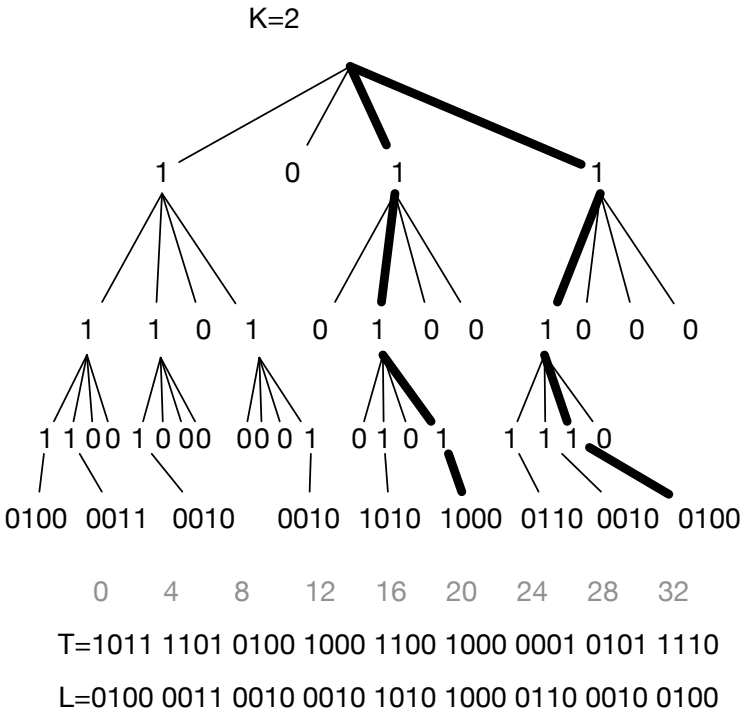
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

Successor

- 例：寻找节点10的出节点（孩子结点）





0 4 8 12 16 20 24 28 32

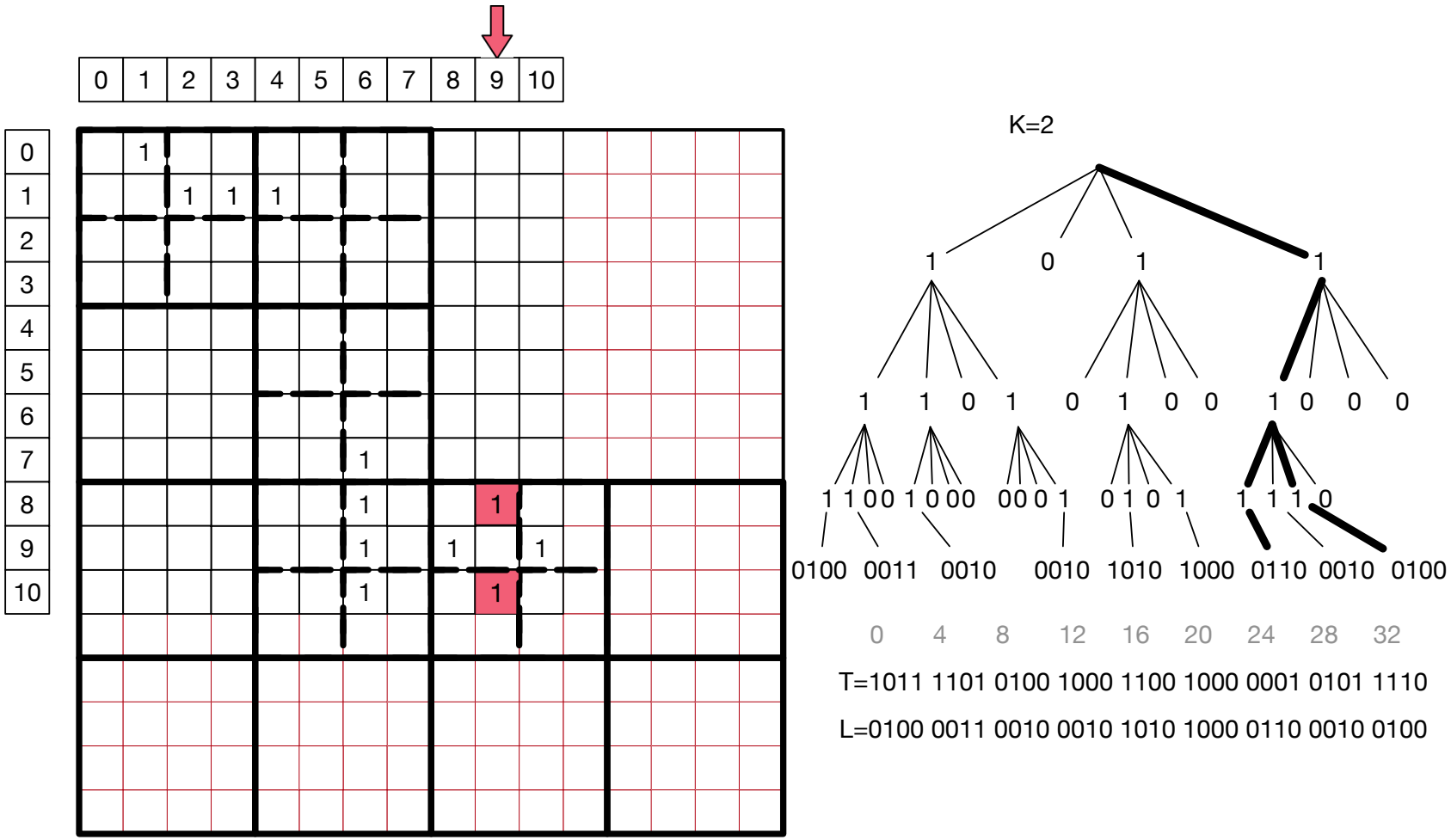
T=1011 1101 0100 1000 1100 1000 0001 0101 1110

L=0100 0011 0010 0010 1010 1000 0110 0010 0100

- 时间复杂度：$O(\sqrt{m}), m = |\{E\}|$

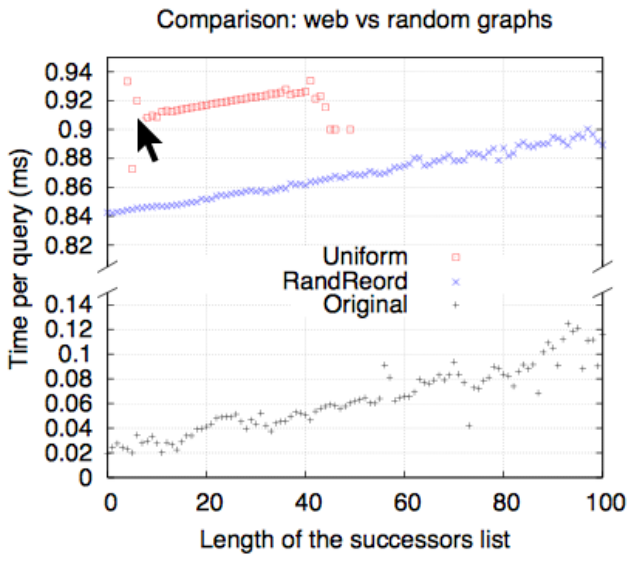
Presuccessor

- 例：寻找节点9的入节点（父结点）

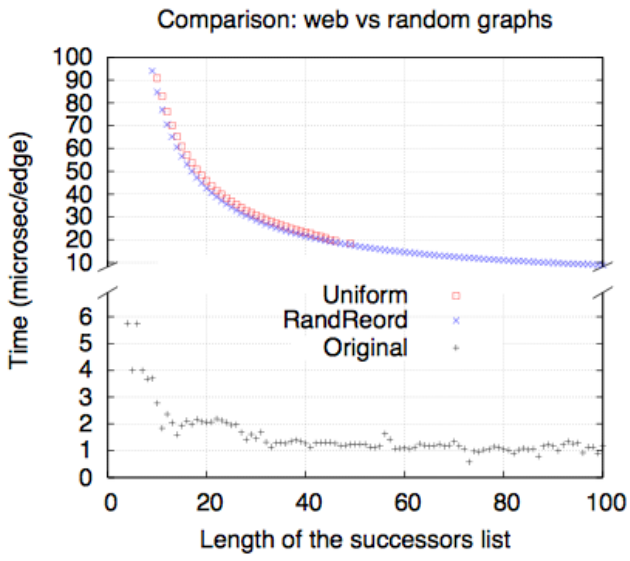


- 时间复杂度：$O(\sqrt{m}), m = |\{E\}|$

Evaluation of successor



(a) Adjacency list retrieval time (in ms) for Web graphs and random graphs.



(b) Successor retrieval time (in μs/e) for Web graphs and random graphs.

Figure 5: Time performance for Web versus random graphs, all `Indochina` variants.

Evaluation of successor

每次`successor()`的时间复杂度是$O(\sqrt{m})$，所以每个边的平均时间复杂度是$O(\sqrt{m}/(m/n)) = O(n/\sqrt{m})$

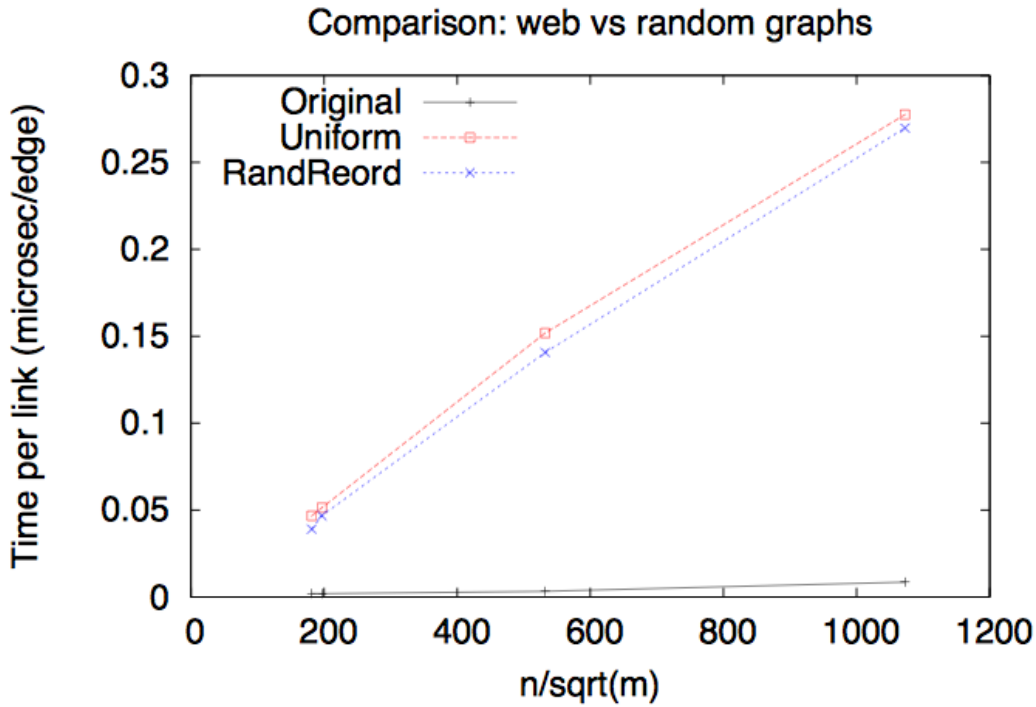
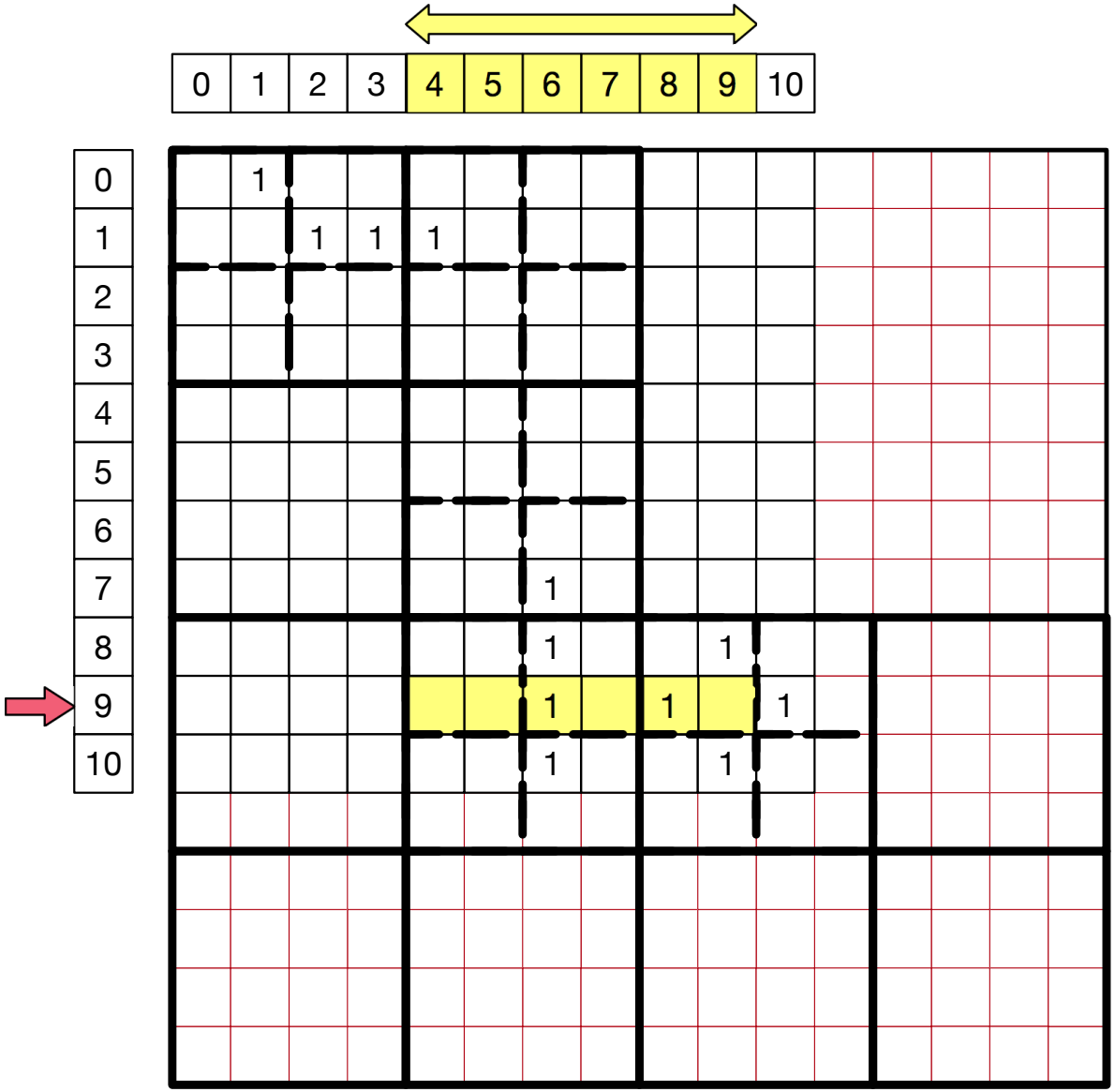


Figure 6: Successors retrieval time (in μs/e) for Web graphs and random graphs compared to the corresponding constant $n/\sqrt{m}$ for different Web graphs.

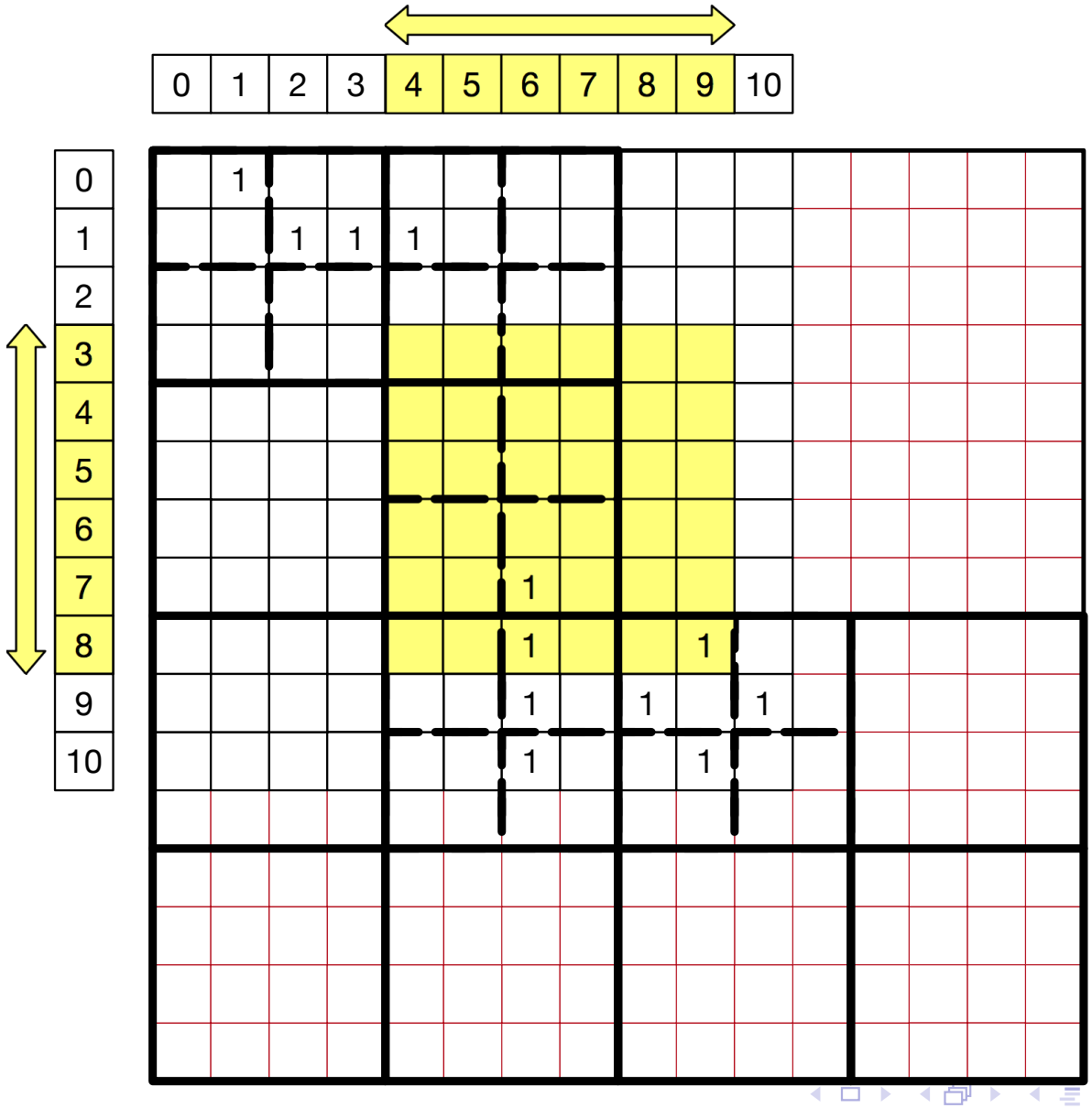
Range Query

- 例：寻找从点p到范围$[q_1, q_2]$的所有边



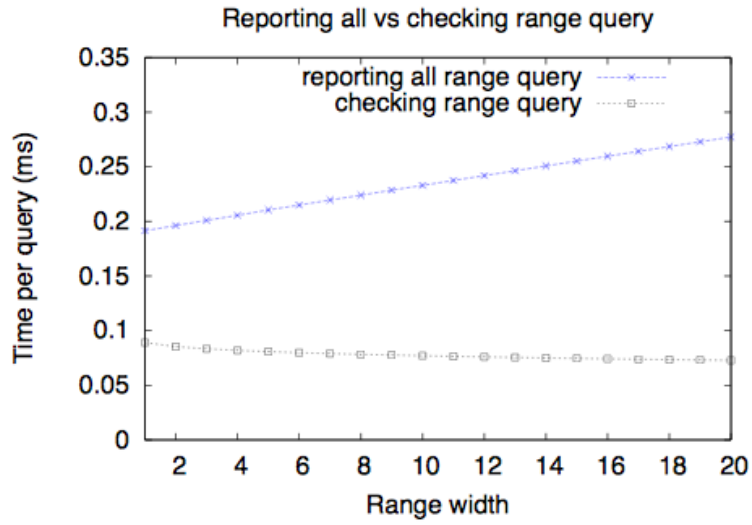
Range Query

- **例**：在给定范围从$[p_1, p_2]$到$[q_1, q_2]$内寻找所有边



Evaluation of range query

(a) Range query performance compared to simple list retrieval queries, for different range widths.

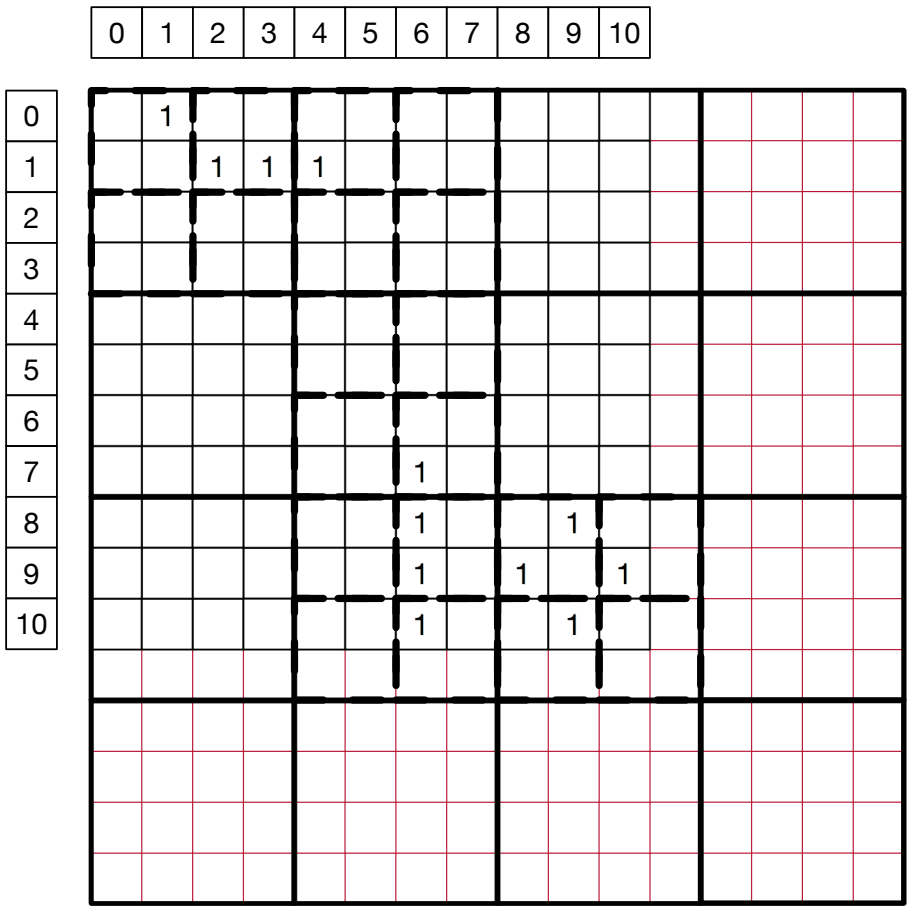


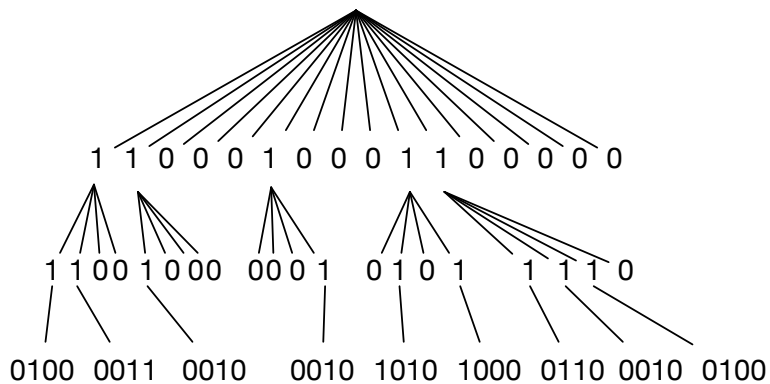
(b) Checking the existence of a link in a range compared to finding all the links in that range.

Figure 11: Performance of range queries.

Hybrid K

- 构建*Hybrid* k^2*tree* 其中$k = 4$ *or* 2





Partitioning the first level

- 使用一个较大的k_0划分第一层，一般取$k_0^2 = m$

Compressing the last level

- 最后一层使用一个较小的k_L划分
- 对在L上出现的每个$k_L \times k_L$的矩阵根据出现频率对其进行排序
- 使用第三方的**Directly Addressable Codes (DACs)**进行压缩和在压缩后的数据上直接存取
- 注：k_L只能从4取到16

Evaluation of improvement

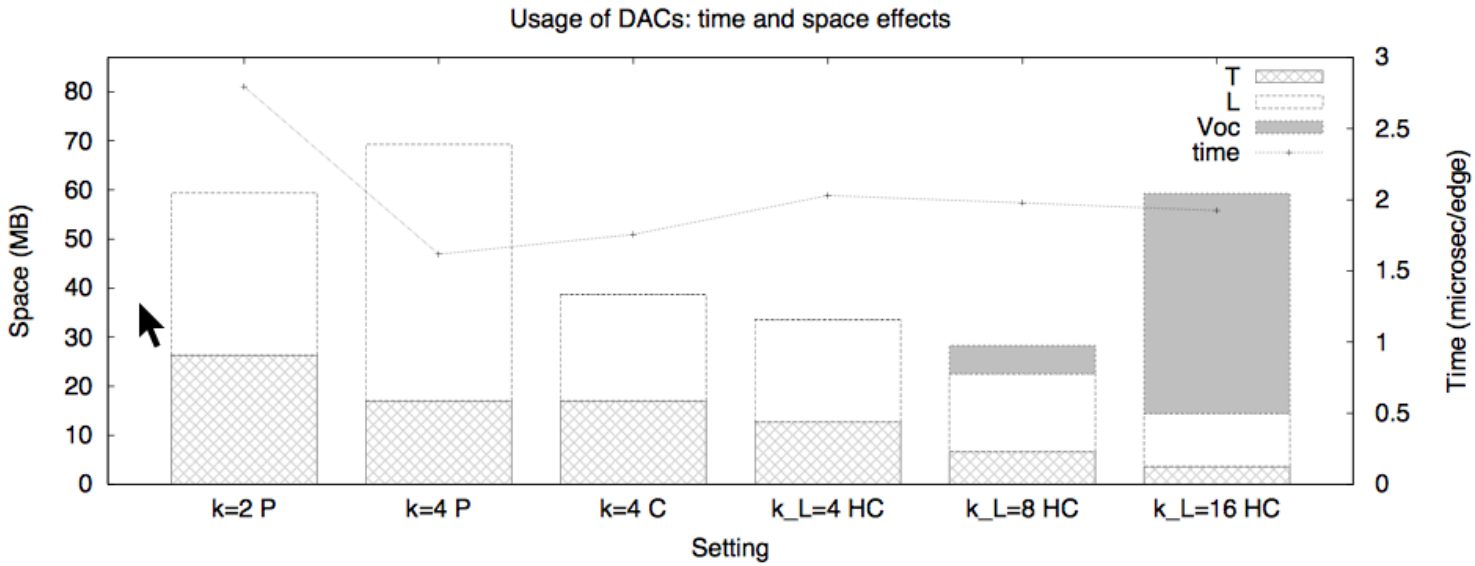


Figure 7: Time and space results for different k^2-trees over graph `Indochina` using BFS ordering. We include three settings where no compression of the leaf levels is used (`k=2 P` and `k=4 P`) and four settings using DACs (`k=4 C`, `k_L=4 HC`, `k_L=8 HC`, and `k_L=16 HC`). `P` stands for plain, `C` for compressed with DACs, and `H` for hybrid. We represent the successor retrieval time (in μs/edge) as a line and the space (in MB) as bars. We separate the space requirements among the different parts of the structure (tree bitmap, leaves and vocabulary).

Evaluation of improvement

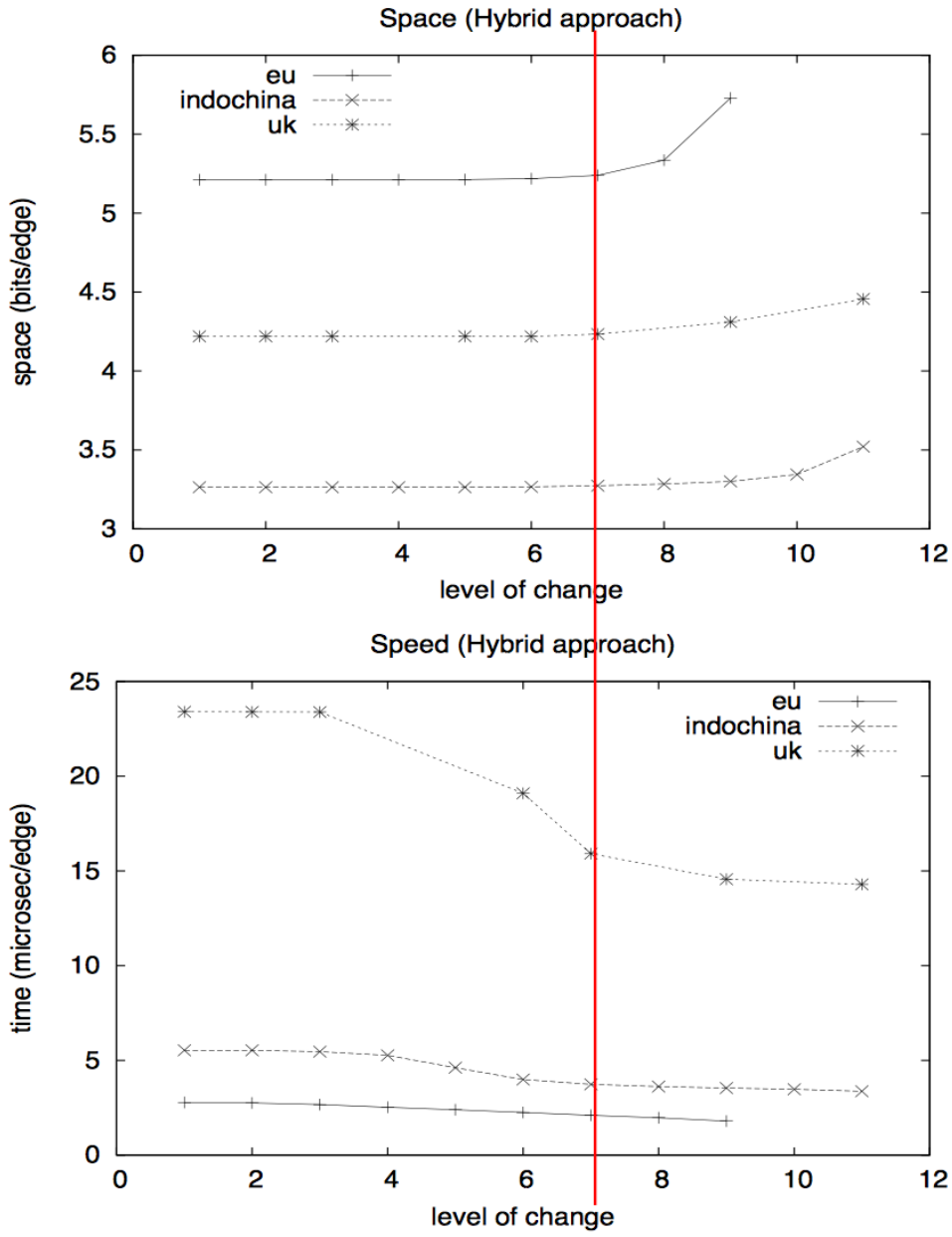
Table 1: Description of the graphs used.

File	Pages	Links	Size (MB)
CNR (2000)	325,577	3,216,152	14
EU (2005)	862,664	19,235,140	77
Indochina (2002)	7,414,866	194,109,311	769
UK (2002)	18,520,486	298,113,762	1,208

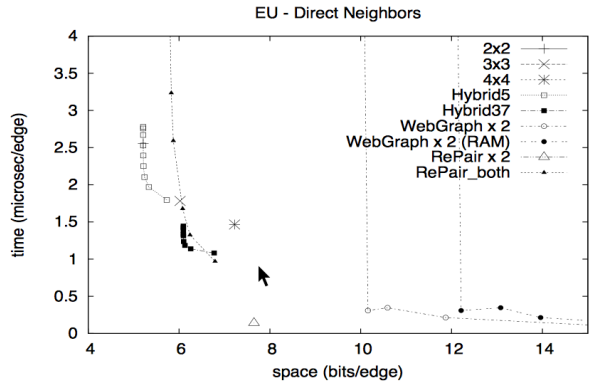
$k_1 = 4, k_2 = 2$

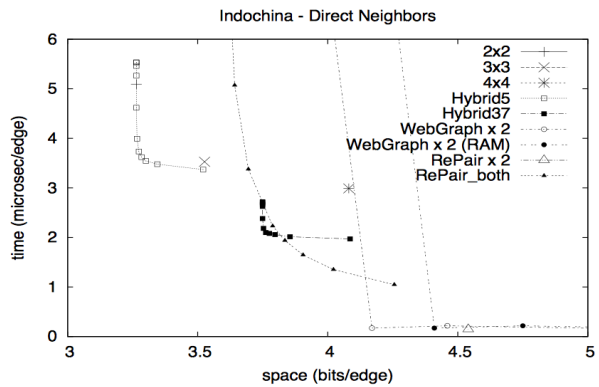
Variant	Tree (bytes)	Leaves (bytes)	Space (bpe)	Direct (μs/e)	Reverse (μs/e)
2×2	6,860,436	5,583,076	5.21076	2.56	2.47
3×3	5,368,744	9,032,928	6.02309	1.78	1.71
4×4	4,813,692	12,546,092	7.22260	1.47	1.42
$H-1$	6,860,432	5,583,100	5.21077	2.78	2.62
$H-2$	6,860,436	5,583,100	5.21077	2.76	2.59
$H-3$	6,860,412	5,583,100	5.21076	2.67	2.49
$H-4$	6,861,004	5,583,100	5.21100	2.53	2.39
$H-5$	6,864,404	5,583,100	5.21242	2.39	2.25
$H-6$	6,876,860	5,583,100	5.21760	2.25	2.11
$H-7$	6,927,924	5,583,100	5.23884	2.10	1.96
$H-8$	7,159,112	5,583,100	5.33499	1.97	1.81
$H-9$	8,107,036	5,583,100	5.72924	1.79	1.67

Evaluation of improvement



Evaluation of improvement





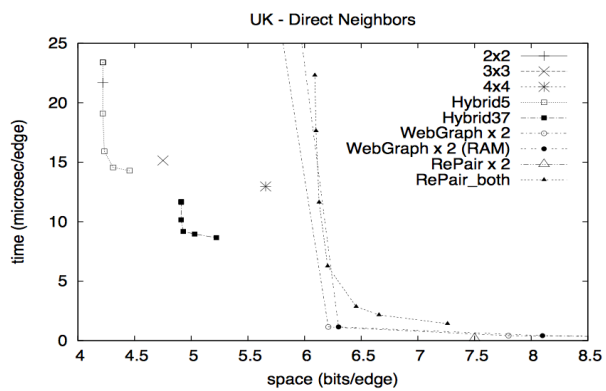
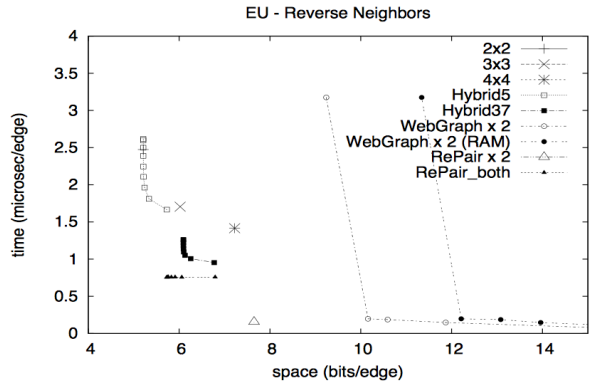
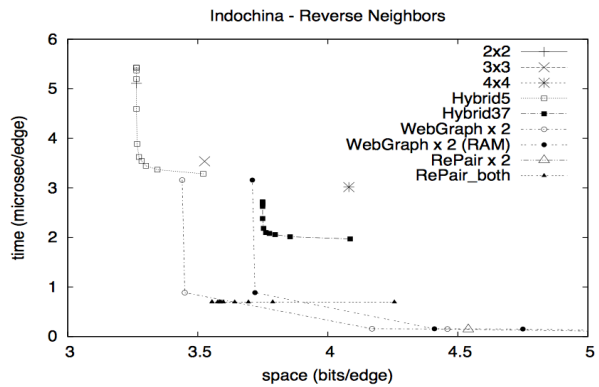


Figure 7: Space/time tradeoff to retrieve direct neighbors for different representations over graphs EU (top), Indochina (center) and UK (bottom).





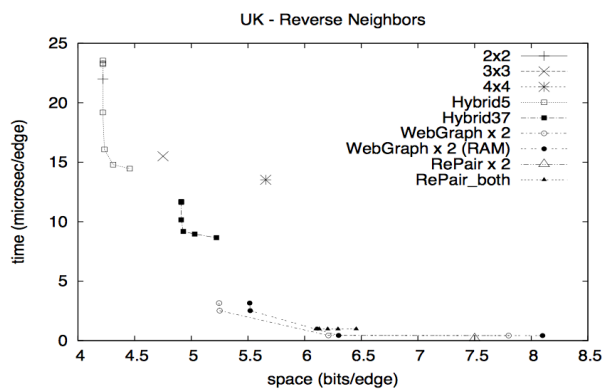


Figure 8: Space/time tradeoff to retrieve reverse neighbors for different representations over graphs EU (top), Indochina (center) and UK (bottom).