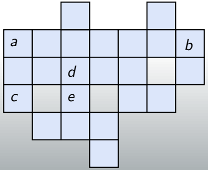
Compact Polyominoes

Shahin Kamali (University of Manitoba)

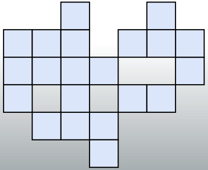
March 2021

Polyominoes

- A polyomino is the union of a set of unit cells that are adjacent edge-by-edge (each cell has up to four neighbors).
  - A polyomino is **connected** by definition (one can visit all cells moving along connected cells).
  - There can be **holes** inside a polyomino.



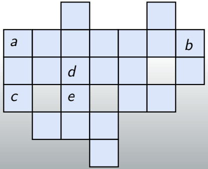
A polyomino



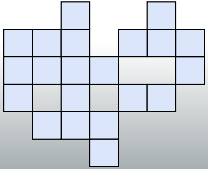
A non-polyomino

Polyominoes

- A polyomino is the union of a set of unit cells that are adjacent edge-by-edge (each cell has up to four neighbors).
  - A polyomino is **connected** by definition (one can visit all cells moving along connected cells).
  - There can be **holes** inside a polyomino.
- Motivation: algorithms for polyominoes are introduced in geometric folding and graphics (e.g., [Aichholzer et al., 2021, Biedl et al., 2012]); but how should they be stored?



A polyomino



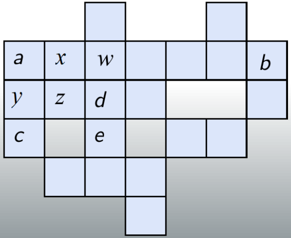
A non-polyomino

Storing Polyominoes

- Goal: store a given polyomino P with n cells **compactly** and answer **navigation** and **visibility** queries for any cell in P in constant time.

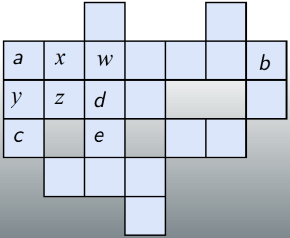
Storing Polyominoes

- Goal: store a given polyomino P with n cells **compactly** and answer **navigation** and **visibility** queries for any cell in P in constant time.
- **Navigation queries:**
  - Neighborhood: given a cell $c \in P$, report neighbors of c on its left/right/top/bottom (if they exist).
  - Degree: given a cell $c \in P$, report the number of neighbors of c.
  - Adjacency: given two cells $c_1, c_2 \in p$, indicate whether they are neighbors or not.



Storing Polyominoes

- Two cells c_1 and c_2 are **visible** iff they appear on the same cell or column of the underlying grid and the straight line segment between them is fully inside the polyomino (e.g., a, c are visible but c, e are not).
- **Visibility queries:**
  - Listing queries: given a cell $c \in P$ and a distance d, report all cells visible to c at distance d of c.
  - Examining queries: given two cells $c_1, c_2 \in p$, indicate whether they are visible or not.



Contribution

Theorem

For a polyomino P of size n, an oracle is constructed that takes $3n + o(n)$ bits and answers all visibility/navigation queries in $O(1)$.

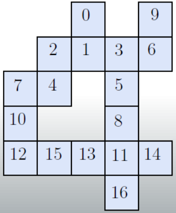
- At least $2.00091n - o(n)$ bits [Barequet et al., 2016] (likely $2.022n - o(n)$ bits [Jensen and Guttmann, 2000]) are required to distinguish polyominoes, confirming that our oracle is compact.

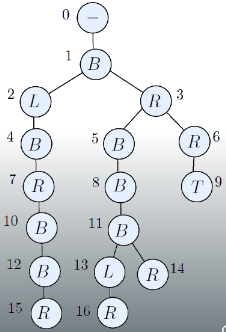
Initial Attempt (BST-tree)

- Store the input polyomino P using a labeled Breadth First Tree T.
  - Each cell in P is mapped to a node in T.
  - Each non-root node gets a label from

$$\{Left(L), Right(R), Bottom(B), Top(T)\}$$

  that indicates its position relative to its parent.



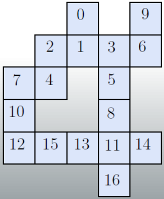


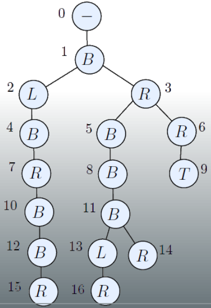
Initial Attempt (BST-tree) [cntd.]

- Using a data structure of [Geary et al., 2006] to store T, one can find the position of each cell in the underlying grid in $O(1)$.

Proposition

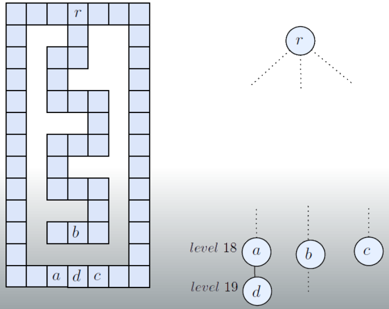
It is possible to store a polyomino of size n in $4n + o(n)$ bits, and indicate if any pair of cells are adjacent and/or visible in $O(1)$.





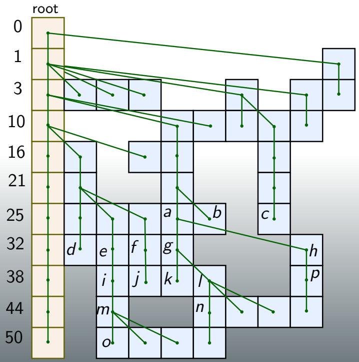
Initial Attempt (BST-tree) [cntd.]

- Unfortunately, using a BST approach, it is not possible to report neighbors/visible cells of a given vertex.
- E.g., it is not clear how c should be located when reporting neighbors of a.



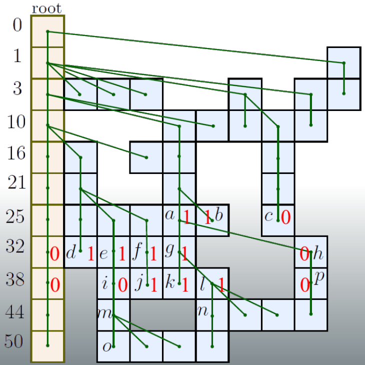
Compact Oracle for Polyominoes

- **Covering Tree T^*:**
  - Nodes at each level of the tree correspond to one row of the polyomino in the underlying grid.
  - The parent of cell c is the rightmost cell at previous level that appears on the same column or on the left of c.
    - Add a column of **dummy cells** on the left to ensure such a parent exists.



Compact Oracle for Polyominoes [cntd.]

- **Left bitstring L:**
  - For each cell c, store one bit that indicates whether c is adjacent to its sibling on its left in T^*.
  - E.g., the bits of L starting at cell a and ending at cell p (in the level-order traversal) are "110011110001110".



Compact Oracle for Polyominoes [cntd.]

- Store the covering tree T^* using a recent succinct oracle of [He et al., 2020] that enables a mapping between the pre-order and level-order traversal for ordinal trees.
- Store the left bitstreing L using a common rank/select data structure for bitstrings (e.g., the structure of [Barbay et al., 2010]).

Compact Oracle for Polyominoes [cntd.]

- Store the covering tree T^* using a recent succinct oracle of [He et al., 2020] that enables a mapping between the pre-order and level-order traversal for ordinal trees.
- Store the left bitstreing L using a common rank/select data structure for bitstrings (e.g., the structure of [Barbay et al., 2010]).
- Storing T^* takes $2n + o(n)$ and storing L takes $n + o(n)$; in total, we use $3n + o(n)$ bits.

Answering queries

- Given any cell c, we can find its levels in T^* and its level-order index in T^* in $O(1)$.
- Two cells at the same level are:
  - **adjacent** iff they appear consequently in the level-order traversal, and the bit stored for the second one in L is 1.
  - **visible** iff the substring of L between them is formed by all 1's.
- Two cells at different levels are:
  - **adjacent** iff one is the leftmost child of the other.
  - **visible** iff one is the leftmost descendant of the other among cells of the same level.
- All queries can be answered in $O(1)$ given the support provided by the data structures used for storing T^* and L.

Summary

- Here is the summary of the results:

Theorem

For a polyomino P of size n, an oracle is constructed that takes $3n + o(n)$ bits and answers the following queries in $O(1)$.

- *Given a cell $c \in P$, report the vertices that are visible to c and located at distance $d \geq 1$ of c on its left/right/top/bottom.*
- *Given two cells $c_1, c_2 \in P$, indicate whether c_1 and c_2 are visible.*

References

Aichholzer, O.; Akitaya, H. A.; Cheung, K. C.; Demaine, E. D.; Demaine, M. L.; Fekete, S. P.; Kleist, L.; Kostitsyna, I.; Löffler, M.; Masárová, Z.; Mundilova, K.; and Schmidt, C. (2021).
"Folding polyominoes with holes into a cube".
Comput. Geom., 93, pp. 101700.

Barbay, J.; Gagie, T.; Navarro, G.; and Nekrich, Y. (2010).
"Alphabet Partitioning for Compressed Rank/Select and Applications".
pages 315–326.

Barequet, G.; Rote, G.; and Shalah, M. (2016).
"$\lambda > 4$: an improved lower bound on the growth constant of polyominoes".
Commun. ACM, 59(7), pp. 88–95.

Biedl, T. C.; Irfan, M. T.; Iwerks, J.; Kim, J.; and Mitchell, J. S. B. (2012).
"The Art Gallery Theorem for Polyominoes".
Discret. Comput. Geom., 48(3), pp. 711–720.

Geary, R. F.; Raman, R.; and Raman, V. (2006).
"Succinct ordinal trees with level-ancestor queries".
ACM Trans. Algorithms, 2(4), pp. 510–534.

He, M.; Munro, J. I.; Nekrich, Y.; Wild, S.; and Wu, K. (2020).
"Distance Oracles for Interval Graphs via Breadth-First Rank/Select in Succinct Trees".
In *proc. 31st International Symposium on Algorithms and Computation (ISAAC)*, page to appear.

Jensen, I. and Guttmann, A. J. (2000).
" "Statistics of lattice animals (polyominoes) and polygons"".
J. Phys. A: Math. Gen., 33, pp. 257–263.