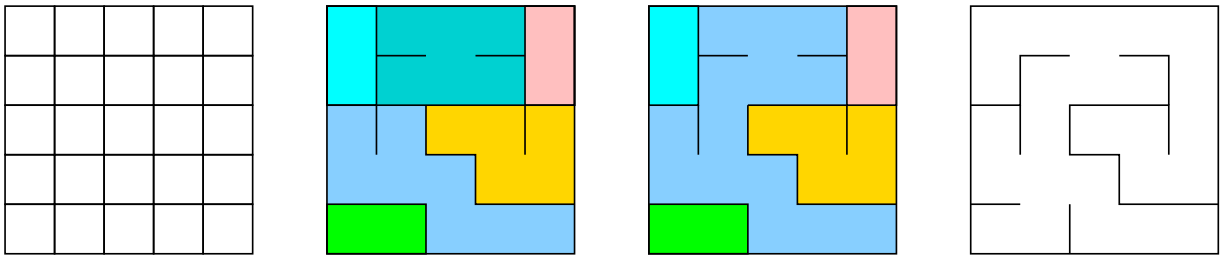
CS29003 Algorithms Laboratory

Assignment No: 7

Last date of submission: 07–September–2017

The King of Computica wants to build a labyrinth. He initially prepares an $m \times n$ mesh of rooms with walls standing between every two adjacent rooms. He then randomly chooses walls to remove. Each wall removal connects two regions. After $mn - 1$ walls are removed, there is only one region, that is, each room can be reached from each other room in a unique way. The following figure illustrates this concept.

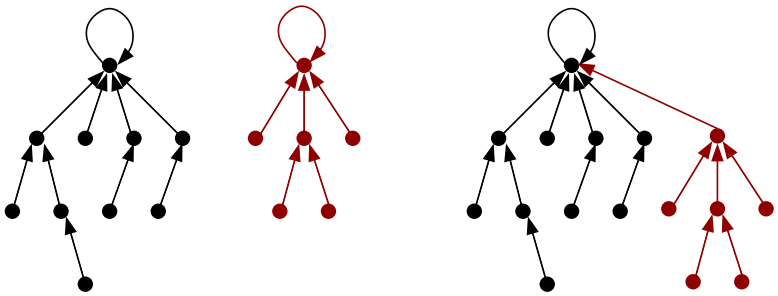


(a) Initial configuration (b) Intermediate regions (c) One iteration (d) Final configuration

Connected regions in the partially constructed labyrinth are treated as sets (of rooms). Let R_1 and R_2 be the regions to which two adjacent rooms belong. Assume that the wall between these rooms is not removed yet. We have either $R_1 = R_2$ or $R_1 \cap R_2 = \emptyset$. If $R_1 = R_2$, the removal of the wall does not connect two separate regions, and should be avoided. Otherwise, the removal of the wall merges two disjoint regions R_1 and R_2 to a single connected region $R_1 \cup R_2$. Regions other than R_1 and R_2 are not affected by this wall removal.

Part 1: Implementation of Disjoint Sets

Let X be a set of size s. Let $X_1, X_2, \ldots, X_t$ be pairwise disjoint subsets of X such that $X = \bigcup_{k=1}^{t} X_k$. For our application, X is the set of mn rooms, and X_k are the connected regions. We represent each X_k as a rooted tree connected only by parent pointers. We assume that the parent of a root is the root itself. We identify the tree with the root, that is, in order to find the subset X_k to which an individual element $x \in X$ belongs, we start at the node storing x, move up the tree following parent pointers, and return the root. Each root also maintains the size of the tree rooted at it. When we want to merge two disjoint subsets X_k and X_l, we make the root of the smaller tree a child of the root of the larger tree.



(a) Two disjoint sets (b) After merging

Write the following three functions for implementing the disjoint-set data structure.

initrooms Initially, with all the walls in place, the rooms are isolated from one another, that is, each room belongs to a singleton set. Create an $m \times n$ array R of nodes. Let the parent pointer of each node point to itself. Also set the size field of each node to one.

findroot Given i, j (a pair of indices), locate and return the root (a pointer) of the tree (connected region) to which the (i, j)-th room belongs. Go to $R[i, j]$, and follow parent pointers.

mergeregions Given two distinct root pointers x and y, make the root of the smaller tree a child of the root of the larger tree. Update the size field of the root node of the merged tree.

Part 2: Wall Management

You need to keep track of which walls are removed, and which are still in place. Walls are of two types: horizontal and vertical. Create an $(m-1) \times n$ array H for the horizontal walls, and an $m \times (n-1)$ array V for the vertical walls. Initially, all walls should be marked as NOT_REMOVED. Write a function *printlabyrinth* that, upon the input of the arrays H and V, prints the rooms and the existing walls in a format illustrated in the sample output below.

Part 3: Building the Labyrinth

Write a function *buildlabyrinth* that repeats the following steps until only one connected region is left. Choose a random wall (horizontal or vertical) to remove. The wall must not have been removed so far. Moreover, it must separate two rooms belonging to different connected regions. Record the removal of the wall, and merge the regions to which the two rooms belong.

The *main*() function

- Read m and n from the user.
- Initialize the arrays R, H, and V. Print the mesh.
- Call *buildlabyrinth* to remove walls leading to a connected labyrinth. Print the labyrinth.

Sample output

```
    m = 6
    n = 10

+++ Initial labyrinth
    +---+---+---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+

+++ Labyrinth created after 59 wall removals

+++ Final labyrinth
    +---+---+---+---+---+---+---+---+---+---+
    |       |       |               |   |   |
    +   +---+   +   +   +---+---+   +   +   +
    |           |   |       |   |           |
    +   +---+   +---+   +   +   +   +---+   +
    |   |               |   |   |   |       |
    +   +---+   +---+---+---+   +   +   +---+
    |       |       |               |   |   |
    +---+   +---+   +---+---+   +---+   +   +
    |       |       |       |       |       |
    +   +   +---+   +   +---+---+---+   +---+
    |   |   |                       |       |
    +---+---+---+---+---+---+---+---+---+---+
```

Submit a single C/C++ source file. Do not use global/static variables.