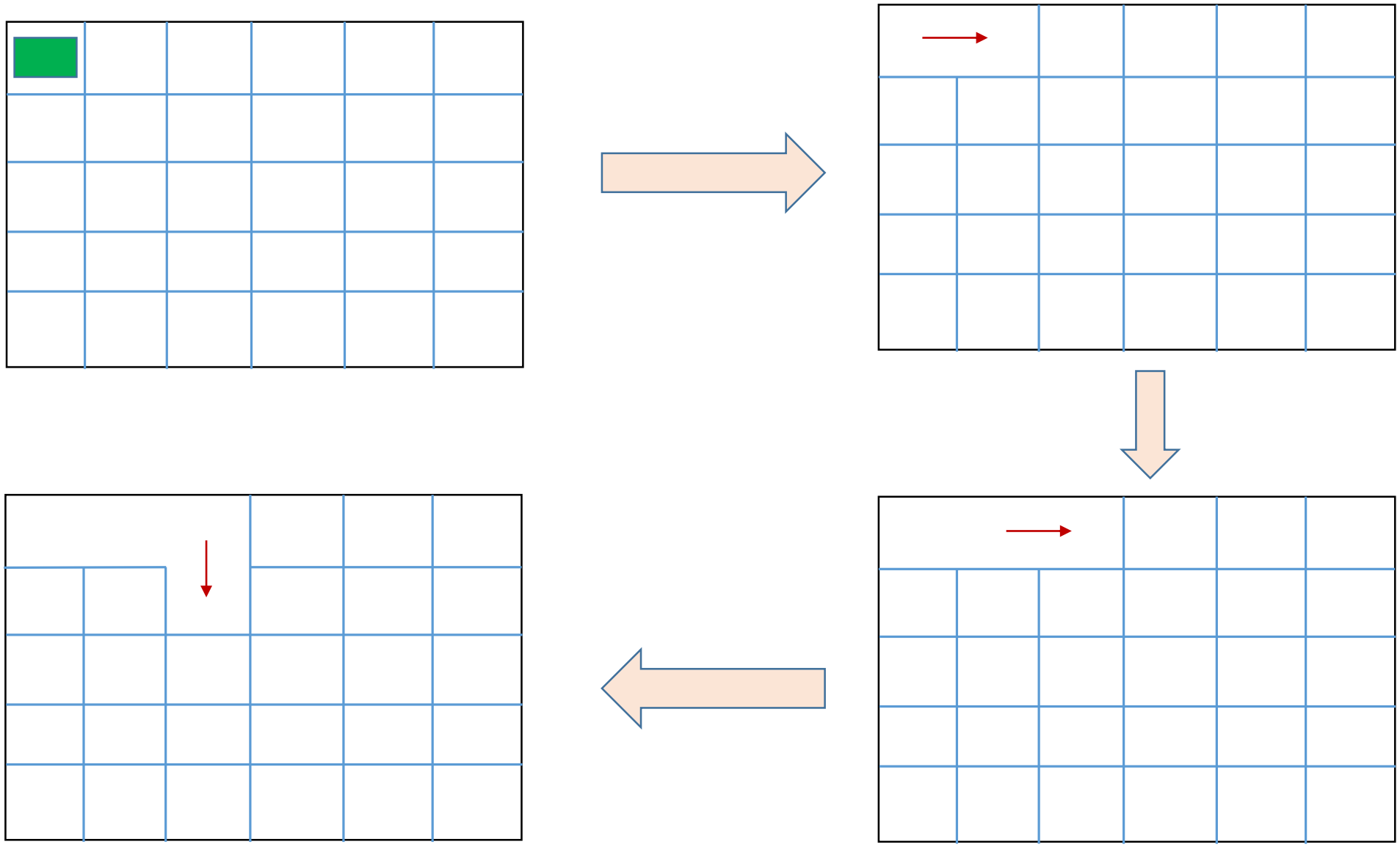
Laboratorio di Algoritmi e Strutture Dati

Generazione casuale di labirinti

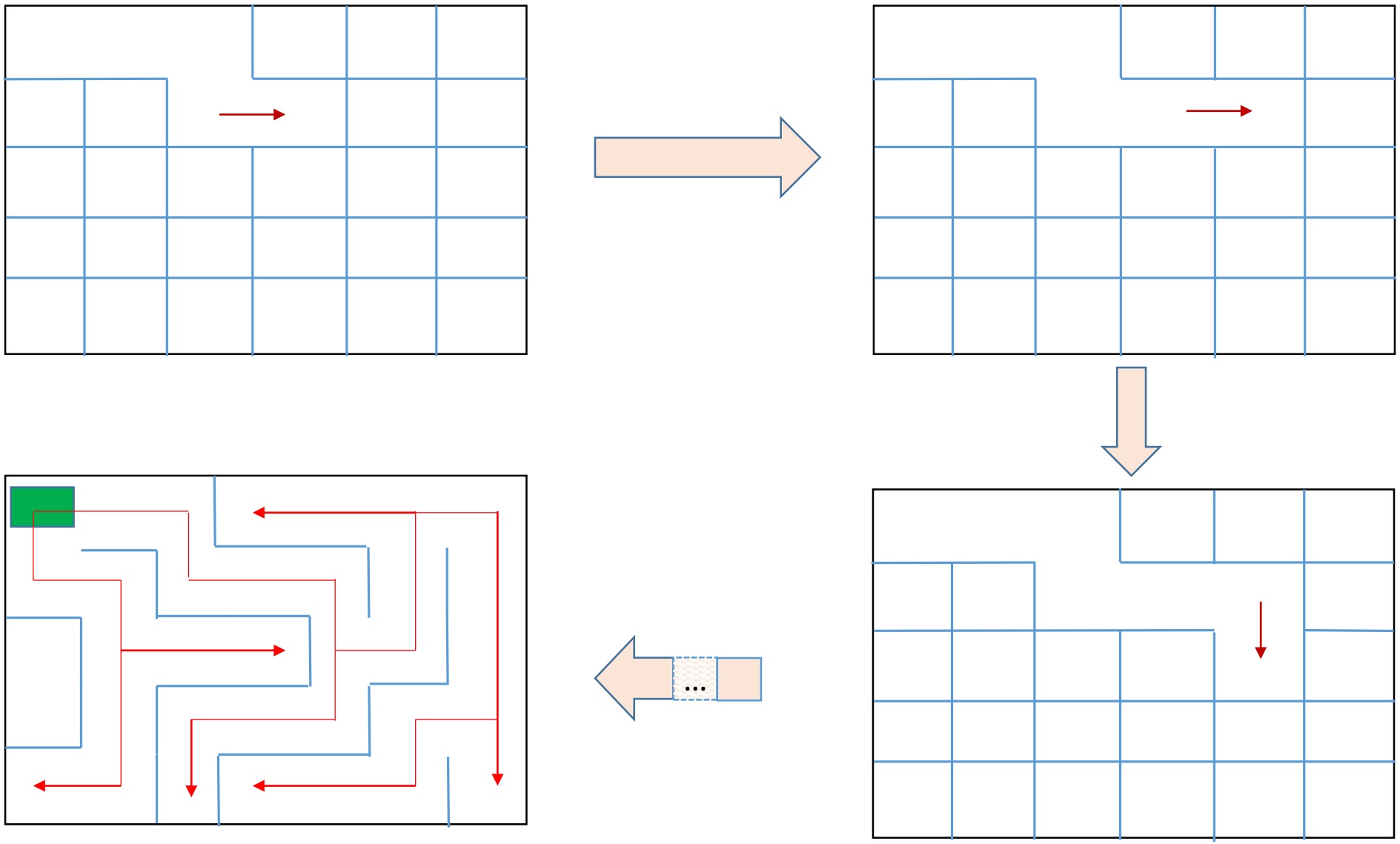
Generazione casuale di labirinti

- Esistono molti algoritmi per la generazione casuale di labirinti
- Ne vediamo brevemente 3 esempi:
  - I primi due “scavano” dei collegamenti tra celle inizialmente tutte scollegate tra di loro seguendo dei cammini casuali nel labirinto (***random walk***);
  - Il terzo decompone ***ricorsivamente*** una “stanza” inizialmente libeara, cioè contenente celle “completamente” tra loro collegate.

Generazine con cammini casuali (random walk)



Generazine con cammini casuali (random walk)



Generatore ispirato a PRIM

```
Prim_maze(MAZE maze)
 initcell = Random_cell(ROWS,COLS)
 Enqueue(Q,initcell)
 WHILE (Q) DO
      cell = Get_random_element(Q)
      IF (Has_unvisited_adjacent(cell)) THEN
           adjs = Get_unvisited_adjacents(maze,cell)
           newcell = Get_random_element(adjs)
           Dig_to(maze,cell,newcell)
           queue = Enqueue(Q,newcell)
      ELSE
           queue = Dequeue(Q, cell)
```

Una cella è "unvisited" se non è stata inserita nel labirinto, cioè se è circondata da 4 muri

Has_unvisited_adjacent(*cell*) restituisce true se *cell* ha almeno un adiacente "unvisited"

Dig_to(*maze*,*cell*,*newcell*) collega *cell* a *newcell* in *maze* (quindi entrambe diverranno "visited")

Generatore ispirato a DFS

```
DFS_maze(MAZE maze)
 initcell = Random_cell(ROWS,COLS)
 S = Push(S,initcell)
 WHILE (S) DO
      cell = Top(S)
      IF (Has_unvisited_adjacent(cell)) THEN
           adjs = Get_unvisited_adjacents(maze,cell)
           newcell = Get_random_cell(adjs)
           Dig_to(maze,cell,newcell)
           S = Push(S,newcell)
      ELSE
           S = Pop(S)
```

Generatore ricorsivo (ispirato a DFS)

```
Recursive_maze(MAZE maze, int r, int c)
   randDirs  = Get_Random_Directions()  // sequenza random di direzioni
   FOR i = 0 TO 3
     SWITCH(randDirs[i])
      CASE UP:
         IF (Dig_up(maze,r, c)) THEN Recursive_maze(maze, r - 2, c)
         BREAK
      CASE DOWN:
         IF (Dig_down(maze,r, c)) THEN Recursive_maze(maze, r + 2, c)
         BREAK
     CASE RIGHT:
         IF (Dig_right(maze,r, c)) THEN Recursive_maze(maze, r, c + 2)
         BREAK
     CASE LEFT:
         IF (Dig_left(maze,r, c)) THEN Recursive_maze(maze, r, c - 2)
         BREAK
```

Generatore ricorsivo (ispirato a DFS)

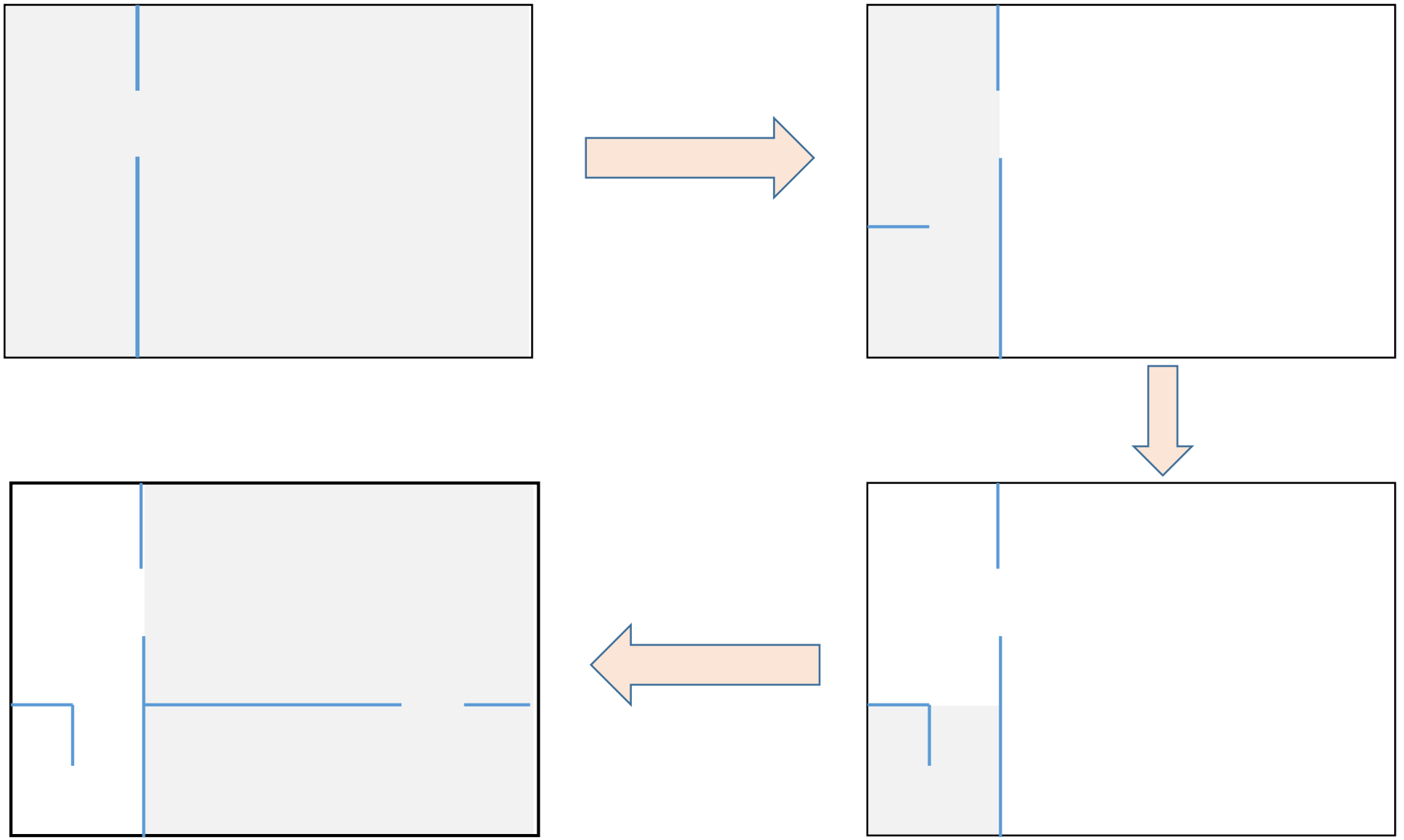
```
Dig_up(MAZE maze, int r, int c)
 IF r-2 > 0 THEN
      maze[r-2][c] = CORR
      maze[r-1][c] = CORR
      return 1
 ELSE
     return 0
```

```
Dig_down(MAZE maze, int r, int c)
 IF r+2 < (ROWS - 1) THEN
      maze[r+2][c] = CORR
      maze[r+1][c] = CORR
      return 1
 ELSE
     return 0
```

```
Dig_left(MAZE maze, int r, int c)
 IF c-2 > 0 THEN
      maze[r][c-2] = CORR
      maze[r][c-1] = CORR
      return 1
 ELSE
     return 0
```

```
Dig_right(MAZE maze, int r, int c)
 IF c+2 < (COLS - 1) THEN
      maze[r][c+2] = CORR
      maze[r][c+1] = CORR
      return 1
 ELSE
     return 0
```

Generazine per suddivisione ricorsiva



Generatore a suddivisione ricorsiva

```
Divide(MAZE maze, int r, int, c, int h, int w)
  IF (h > 1 AND w > 1) THEN
     IF (h > w) THEN Divide_horizontally(maze,r,c,h,w)
     ELSE            Divide_vertically(maze,r,c,h,w)
```

(**r**,**c**) indica la cella iniziale della stanza da decomporre

(**h**,**w**) indica le dimensioni della stanza (altezza,larghezza)

```
Divide_horizontally(MAZE maze, int r, int, c, int h, int w)
   h_wall = Random_even(h)  // chose where to put the horizontal wall
   door = Random_odd(w)     // chose where to put the door
   FOR i = 0 TO w-1         // build the separating wall
     IF (i != door) THEN
        maze[r+h_wall][c+i] = WALL
   Divide(maze, r, c, h_wall, w)                  // recursively divide
   Divide(maze, r+h_wall+1, c, h-h_wall-1, w)  // the two new rooms
```

Generatore a suddivisione ricorsiva

```
Divide_vertically(MAZE maze, int r, int, c, int h, int w)
   v_wall = Random_even(w) // chose where to put the vertical wall
   door = Random_odd(h)    // chose where to put the door
   FOR i = 0 TO h-1        // build the separating wall
     IF (i != door) THEN
        maze[r+i][c+w_wall] = WALL
   Divide(maze, r, c, h, v_wall)                  // recursively divide
   Divide(maze, r, c+v_wall+1, h, w-v_wall-1)  // the two new rooms
```

```
Random_odd(int bound)
   val = rand()%bound
   IF (val%2 = 0) THEN
      IF (val > 0) THEN
         val = val-1
      ELSE
          val = val+1
  return val
```

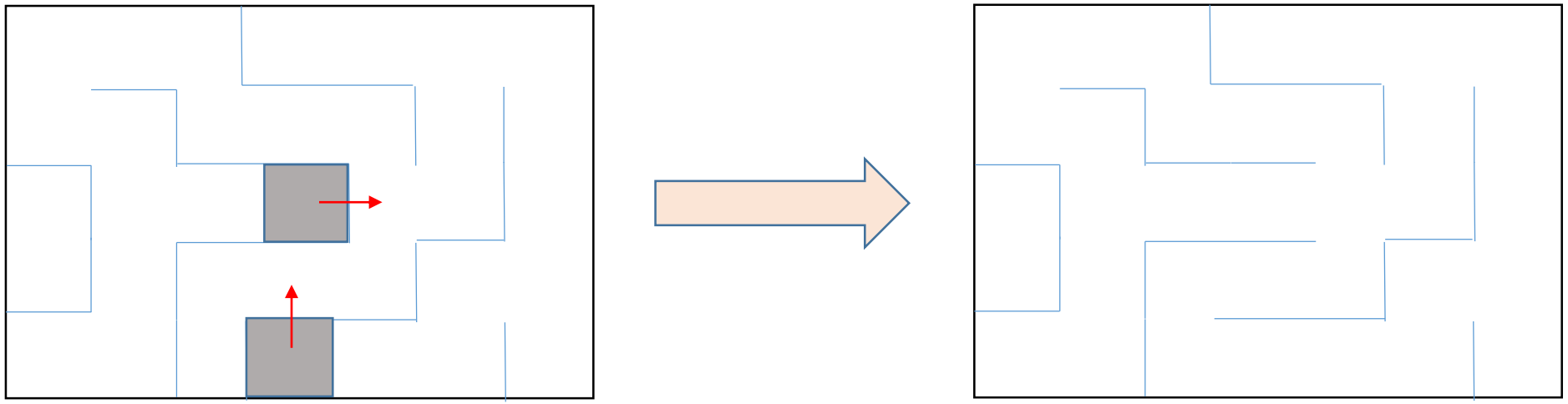
```
Random_even(int bound)
   val = rand()%bound
   IF (val%2 = 1) THEN
      IF (val > 0) THEN
         val = val-1
      ELSE
          val = val+1
  return val
```

Vicoli ciechi

• Gli algoritmi di generazione casuale visti, chi più chi meno, costruiscono labirinti senza cicli. Di conseguenza, sono presenti *vicoli ciechi*, cioè celle una volta giunti quali si può solo tornare indietro.

• Un ***vicolo cieco*** è quindi una cella con una sola uscita (circondata da 3 muri)

• È spesso utile eliminare tutti o una parte (ad. es. in modo casuale) di questi vicoli ciechi, in modo da rendere più agevole muoversi nel labirinto.



Progetto Finale: descrizione e funzionalità

- Implementare in C tutti i generatori casuali di labirinti descritti qui.
- Eventualmente, implementarne altri, basandosi sulla documentazione disponibile ad esempio in rete.
- Prevedere tecniche di eliminazione totale e/o parziale dei vicoli ciechi
- Realizzare un'applicazione, utilizzando la libreria ***ncurses*** per:
  - Genere casualmente labirinti, tramite selezione da menu del metodo da usare
  - Visualizzare il labirinto
  - Modificare il labirinto in maniera interattiva (ad esempio, spostandosi sul video tramite tastiera o mouse per selezionare muri da eliminare o corridoi da chiudere)
  - Ricercare il percorso minimo da una sorgente a una destinazione, tramite uno degli algoritmi studiati (sempre selezionabile da menu)
    - Permettere la selezione interattiva di sorgente e destinazione
    - Visualizzazione del percorso minimo a video
    - Animazione a video di un personaggio che segue il percorso minimo
    - Possibilità di visualizzare ‘‘graficamente” (ad es. con colori differenti) le celle effettivamente esplorate dall'algoritmo di ricerca del percorso minimo selezionato
  - Proporre e realizzare eventuali sviluppi personale (da discutere prima col docente)

Progetto Finale: criteri di valutazione

Il progetto verrà valutato secondo i seguenti criteri:

1. Flessibilità e generalità della progettazione dell'applicazione, con particolare attenzione all'applicazione delle tecniche affrontate durante il corso;
2. Organizzazione, leggibilità e strutturazione del codice sorgente;
3. Originalità della proposta e della sua realizzazione, con riferimento anche a sviluppi personali del candidato;
4. Documentazione tecnica dell'applicazione, che deve coprire sia le indicazioni d'uso e le funzionalità che gli aspetti salienti di progettazione e implementazione.