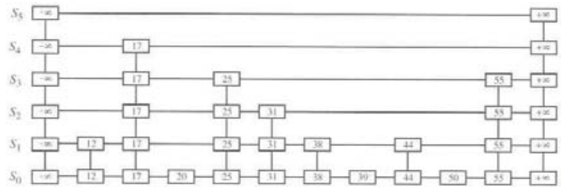
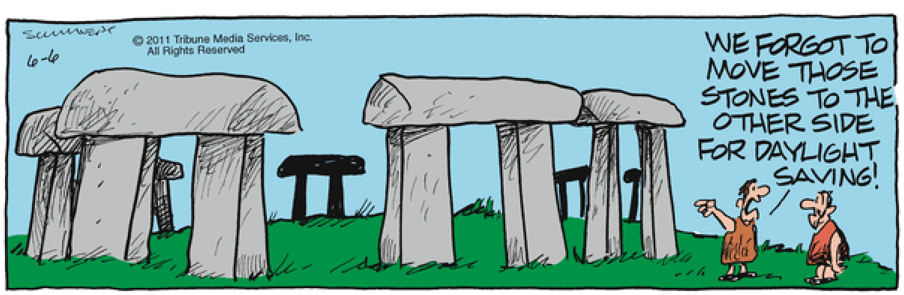


Figure 8.9: Example of a skip list.

CSSE 230 Day 25

Skip Lists

Reminders/Announcements



Skip Lists

An alternative to balanced trees
Sorted data.
Random.
Expected times are O(log n).

An alternative to balanced trees

- Indexed lists
  - One-level index.
  - 2nd-level index.
  - 3rd-level index
  - log-n-level index.
- Problem: insertion and deletion.

Remember the problem with keeping trees *completely* balanced"?

- Solution: Randomized node height: Skip lists.
  - Pugh, 1990 CACM.
- https://people.ok.ubc.ca/ylucet/DS/SkipList.html

Note that we can iterate through the list easily and in increasing order, like a threaded BST"

A slightly different skip list representation

- Uses a bit more space, makes the code simpler.
- Michael Goodrich and Roberto Tamassia.

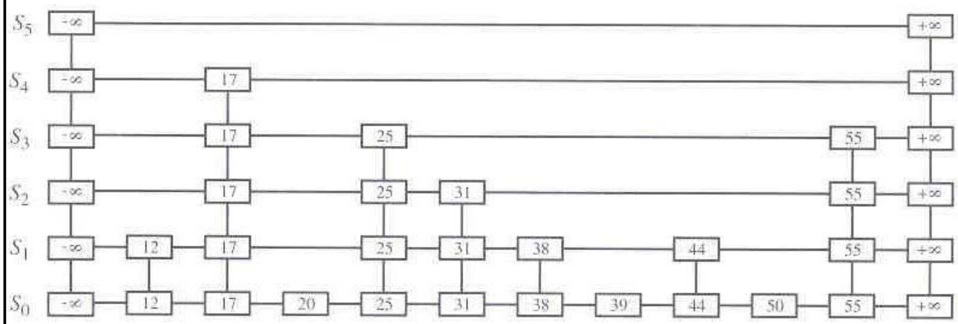


Figure 8.9: Example of a skip list.

Methods in SkipListNode class

after(p): Return the position following p on the same level.

before(p): Return the position preceding p on the same level.

below(p): Return the position below p in the same tower.

above(p): Return the position above p in the same tower.

Search algorithm

1. If S.below(p) is null, then the search terminates—we are ***at the bottom*** and have located the largest item in S with key less than or equal to the search key k. Otherwise, we ***drop down*** to the next lower level in the present tower by setting $p \leftarrow S$.below(p).
2. Starting at position p, we move p forward until it is at the right-most position on the present level such that key(p) $\leq k$. We call this the ***scan forward*** step. Note that such a position always exists, since each level contains the special keys $+\infty$ and $-\infty$. In fact, after we perform the scan forward for this level, p may remain where it started. In any case, we then repeat the previous step.

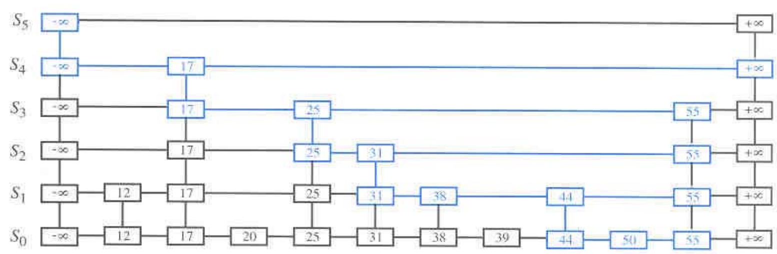
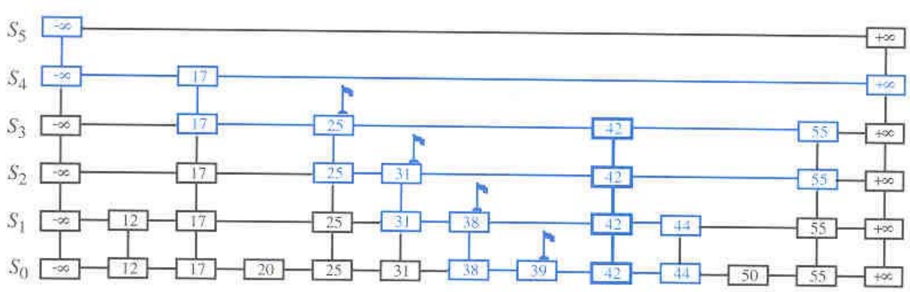
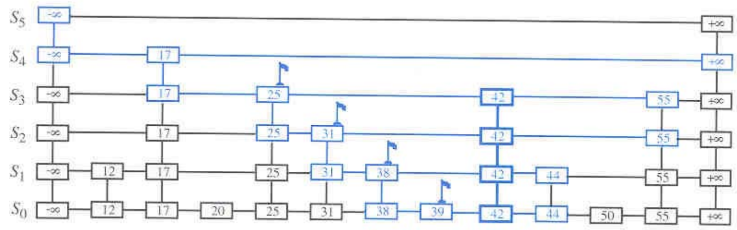


Figure 8.10: Example of a search in a skip list. The positions visited when searching for key 50 are highlighted in blue.

Insertion diagram



Insertion algorithm



Algorithm SkipInsert(k, e):

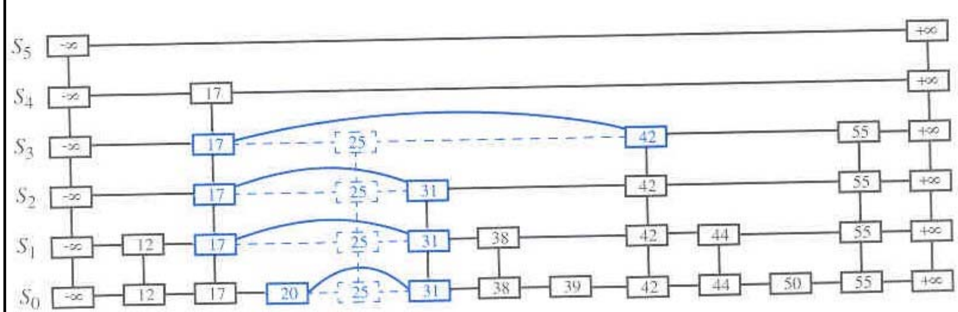
Input: Item (k, e)

Output: None

```
p ← SkipSearch(k)
q ← insertAfterAbove(p, null, (k,e))          {we are at the bottom level}
while random() < 1/2 do
   while above(p) = null do
      p ← before(p)          {scan backward}
   p ← above(p)          {jump up to higher level}
   q ← insertAfterAbove(p, q, (k,e))          {insert new item}
```

Code Fragment 8.5: Insertion in a skip list, assuming random() returns a random number between 0 and 1, and we never insert past the top level.

Remove algorithm



(sort of) Analysis of Skip Lists

- No guarantees that we won't get O(N) behavior.
  - The interaction of the random number generator and the order in which things are inserted/deleted *could* lead to a long chain of nodes with the same height.
  - But this is **very** unlikely.
  - ***Expected*** time for search, insert, and remove are O(log n).

Questions

Exhaustive Search and Backtracking

» A taste of artificial intelligence

Exhaustive search

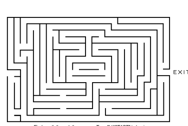
- Given: a large set of possible solutions to a problem

The "search space"

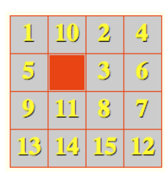
- Goal: Find all solutions (or an optimal solution) from that set

- Questions we ask:
  - How do we represent the possible solutions?
  - How do we organize the search?
  - Can we avoid checking some obvious non-solutions?

- Examples:



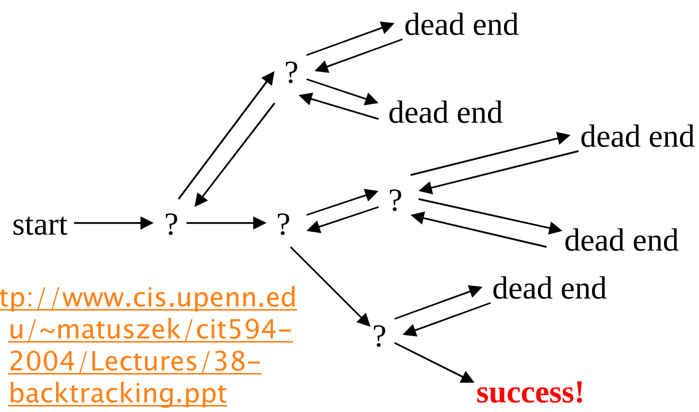
Mazes



The "15" puzzle

http://www.numeracysoftware.com/m[illegible].bmp

In backtracking, we always try to extend a partial solution



http://www.cis.upenn.edu/~matuszek/cit594-2004/Lectures/38-backtracking.ppt

Note: the search is a tree and we explore it using a pre-order traversal

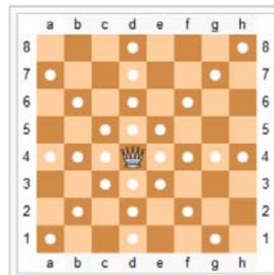
In backtracking, we always try to extend a partial solution

The non-attacking chess queens problem is a famous example

- In how many ways can N chess queens be placed on an NxN grid, so that none of the queens can attack any other queen?
- I.e. there are not two queens on the same row, same column, or same diagonal.

- There is no "formula" for generating a solution.

- The famous computer scientist Niklaus Wirth described his approach to the problem in 1971: **Program Development by Stepwise Refinement** http://sunnyday.mit.edu/16.355/wirth-refinement.html#3



Possible moves of the queen are shown

http://en.wikipedia.org/wiki/Queen_(chess)

With a partner, discuss "possible solution" search strategies

- In how many ways can N chess queens be placed on an NxN grid, so that none of the queens can attack any other queen?
  - I.e. no two queens on the same row, same column, or same diagonal.

Two minutes
No Peeking!

Search Space Possibilities 1/5

1

- Very naive approach. Perhaps stupid is a better word!
  There are N queens, N^2 squares.
- For each queen, try every possible square, allowing the possibility of multiple queens in the same square.
  - Represent each potential solution as an N-item array of pairs of integers (a row and a column for each queen).
  - Generate all such arrays (you should be able to write code that would do this) and check to see which ones are solutions.
  - Number of possibilities to try in the NxN case:
  - Specific number for N=8:

281,474,976,710,656

Search Space Possibilities 2/5

Slight improvement. There are N queens, N^2 squares. For each queen, try every possible square, notice that we can't have multiple queens on the same square.

- Represent each potential solution as an N-item array of pairs of integers (a row and a column for each queen).
- Generate all such arrays and check to see which ones are solutions.
- Number of possibilities to try in NxN case:
- Specific number for N=8:

178,462,987,637,760

(vs. 281,474,976,710,656)

Search Space Possibilities 3/5

- Slightly better approach. There are N queens, N columns. If two queens are in the same column, they will attack each other. Thus there must be exactly one queen per column.
- Represent a potential solution as an N-item array of integers.
  - Each array position represents the queen in one column.
  - The number stored in an array position represents the row of that column's queen.
  - Show array for 4x4 solution.
    - Generate all such arrays and check to see which ones are solutions.
    - Number of possibilities to try in NxN case:
    - Specific number for N=8:

16,777,216

Search Space Possibilities 4/5

- Still better approach There must also be exactly one queen per row.
- Represent the data just as before, but notice that the data in the array is a set!
  - Generate each of these and check to see which ones are solutions.
  - How to generate? A good thing to think about.
  - Number of possibilities to try in NxN case:
  - Specific number for N=8:

40,320

Search Space Possibilities 5/5

- Backtracking solution
- Instead of generating all permutations of N queens and checking to see if each is a solution, we generate "partial placements" by placing one queen at a time on the board
- Once we have successfully placed $k<N$ queens, we try to *extend* the partial solution by placing a queen in the next column.
- When we extend to N queens, we have a solution.

Experimenting with 8 x 8 Case

- Play the game:
  - http://homepage.tinet.ie/~pdpals/8queens.htm
- See the solutions:
  - http://www.dcs.ed.ac.uk/home/mlj/demos/queens (if you can figure out how to enable Java in your browser)

Program output:

```
>java RealQueen 5
SOLUTION:  1 3 5 2 4
SOLUTION:  1 4 2 5 3
SOLUTION:  2 4 1 3 5
SOLUTION:  2 5 3 1 4
SOLUTION:  3 1 4 2 5
SOLUTION:  3 5 2 4 1
SOLUTION:  4 1 3 5 2
SOLUTION:  4 2 5 3 1
SOLUTION:  5 2 4 1 3
SOLUTION:  5 3 1 4 2
```

Tommorrow:

We'll look at details of the algorithm.

Bring your computer, capabl;e of compiling and running Java programs.