Recursive Best-First Search

Recursive Best-First Search

- Difference between A* and RBFS
  - A* keeps in memory all of the already generated nodes
  - RBFS only keeps the current search path and the sibling nodes along the path
- **RBFS is a linear-space algorithm that expands nodes in best-first order even with a non-monotonic cost function** and generates fewer nodes than iterative deepening with a monotonic cost function
- RBFS Example: ILBFS(Iterative Linear Best-First Search)

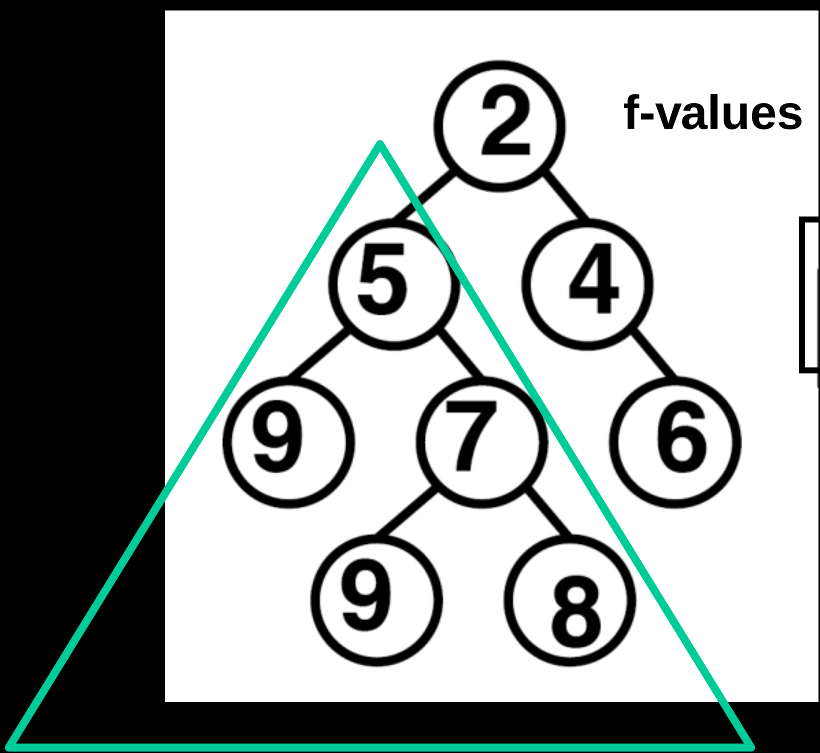
RBFS vs ID

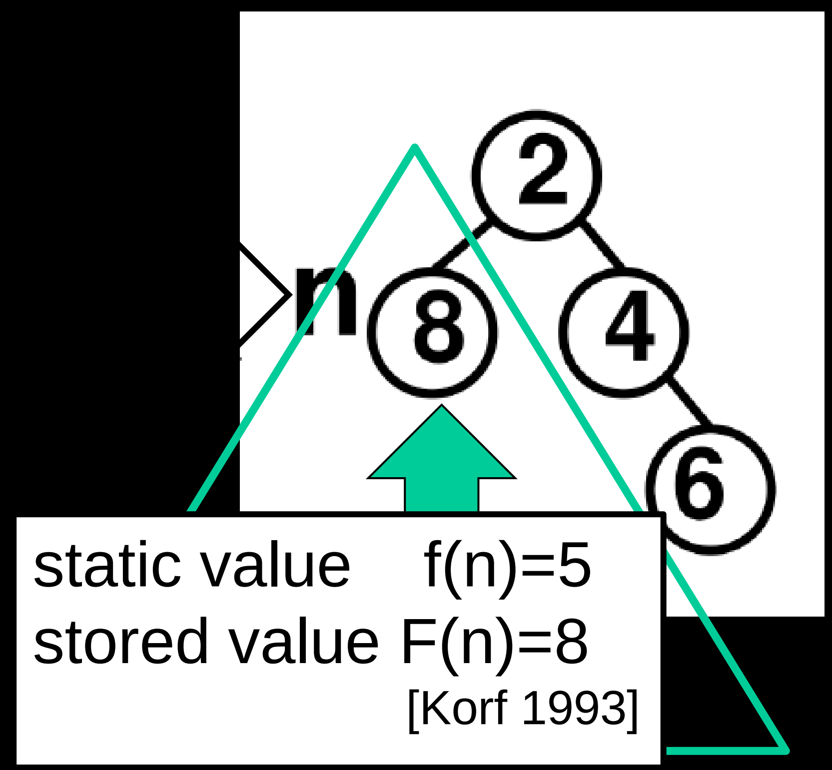
- If the cost function is non-monotonic, the two algorithms are not directly comparable.
- RBFS generate fewer nodes than ID on average. Because RBFS only backtracks to their common ancestor instead of directly to the root as IDA*.

Collapse and Restore macros

The ILBFS – RBFS alogorithms

Collapse macro for best-first search

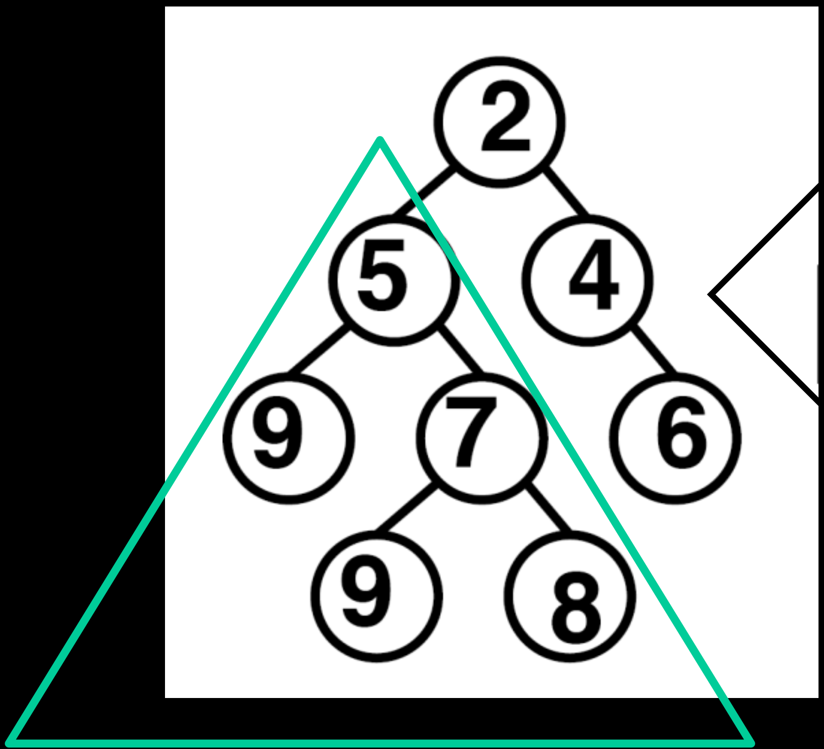


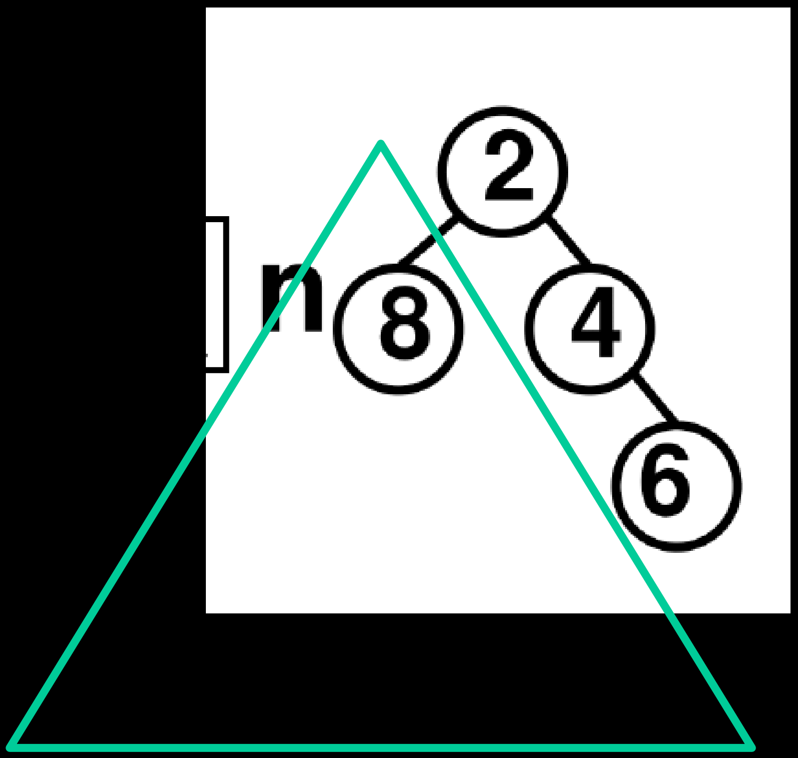


9 9 8 6

8 6

Restore macro





9 9 8 6

8 6

- Collapse is a **lossy** compression

  1) How do we know a node was collapsed?

  2) How do we restore?

<u>Restore is algorithm dependent</u>

If F(n) > f(n) and the f-value is monotonically increasing just perform a bounded DFS by F(N). [Korf 1993]

ILBFS [#1 SoCS-2015]

Iterative linear best-first search

Iterative variant of RBFS [Korf, AIJ 1993]

Algorithm 1: High-level ILBFS

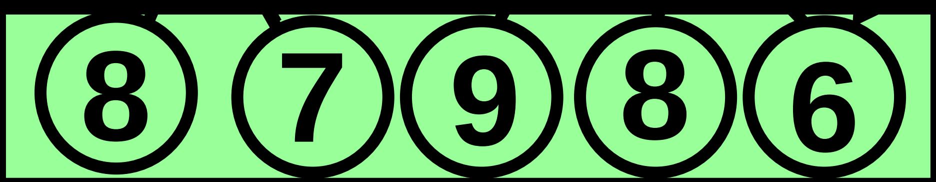
```
Input: Root R
Insert R into OPEN and TREE
oldbest=NULL
while OPEN not empty do
    best=extract_min(OPEN)
    if goal(best) then
        exit
```

Collapse

Restore

```
    foreach child C of best do
        Insert C to OPEN and TREE
    oldbest ← best
```

ILBFS



Principal branch invariant

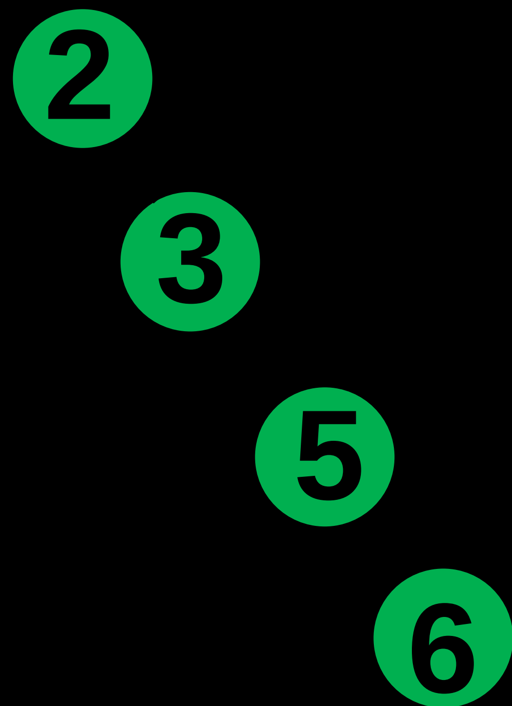
Store only the branch of the best node and its siblings

ILBFS

Principal branch invariant

Store only the branch of the best node and its siblings

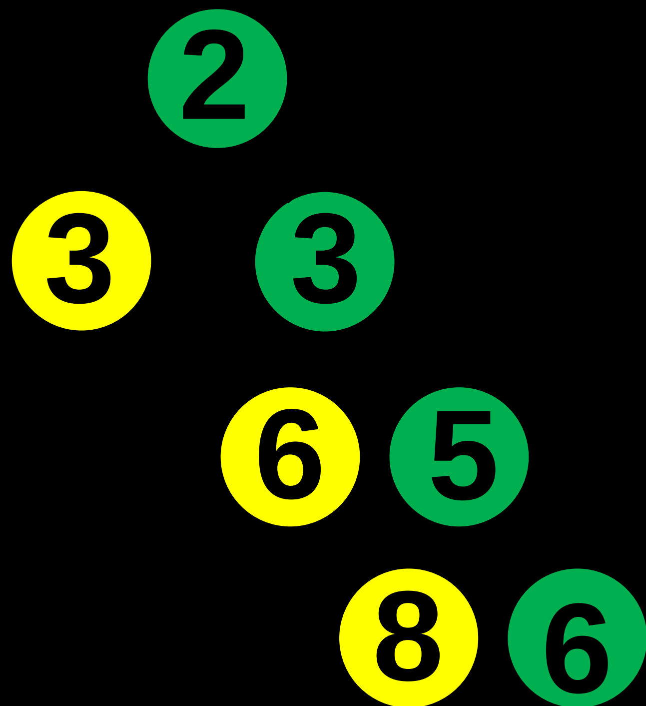
ILBFS



Principal branch invariant

Store only the branch of the best node and its siblings

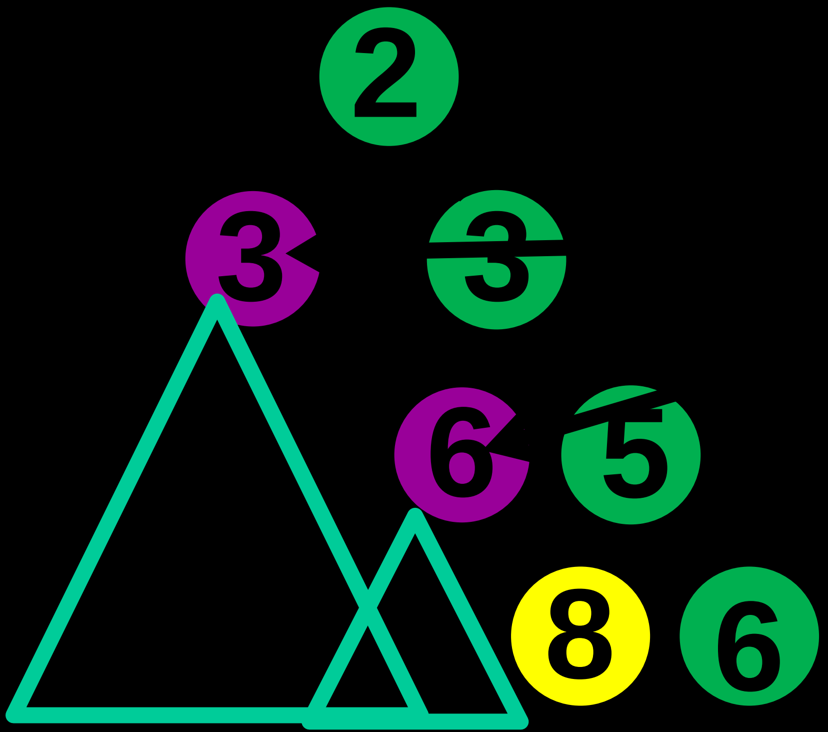
ILBFS



Principal branch invariant

Store only the branch of the best node and its siblings

ILBFS

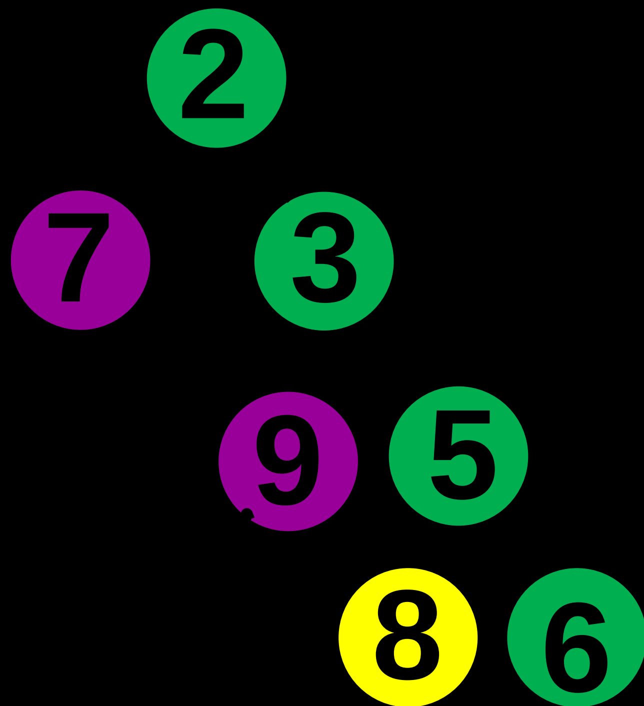


Collapse

Principal branch invariant

Store only the branch of the best node and its siblings

ILBFS



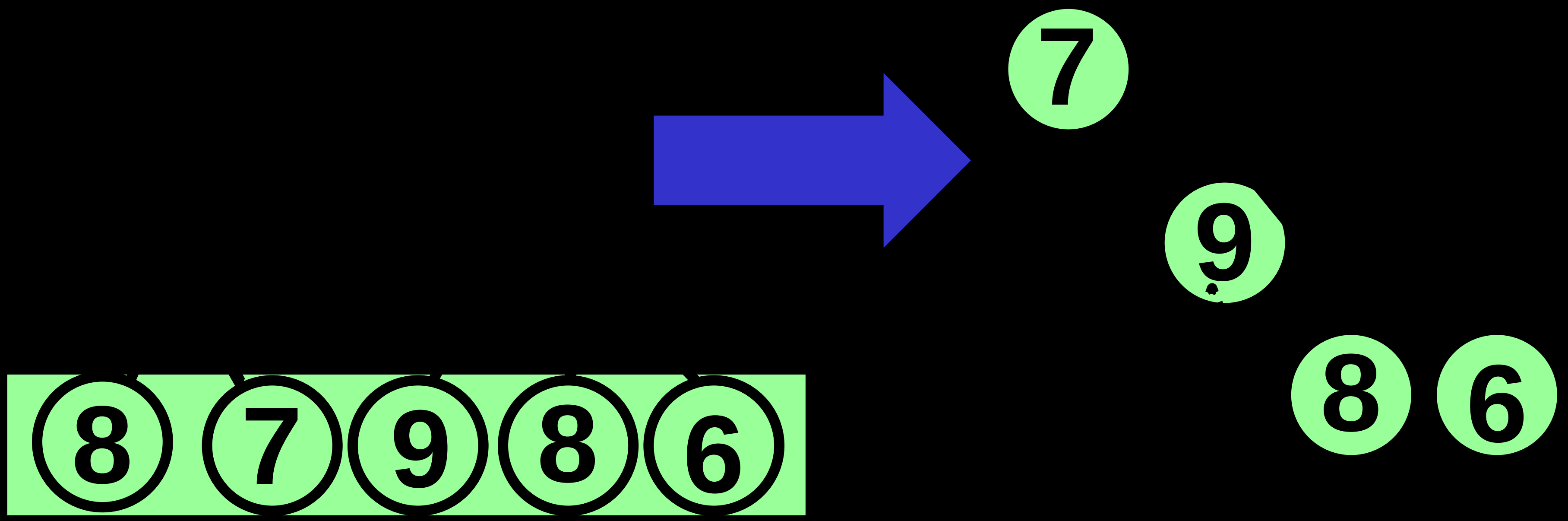
ILBFS

7

9

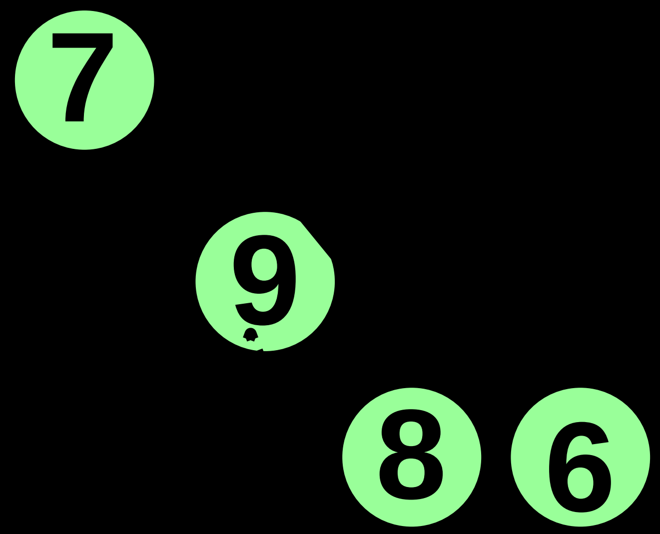
8 6

ILBFS



Principal branch invariant

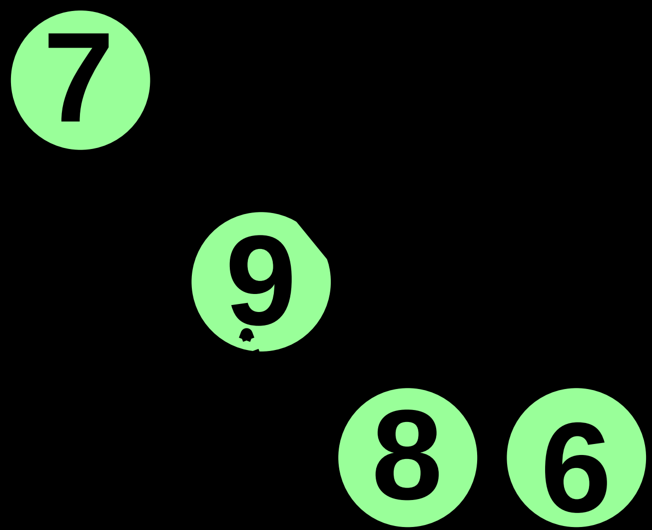
ILBFS



Case 1:

Best is a child of oldbest

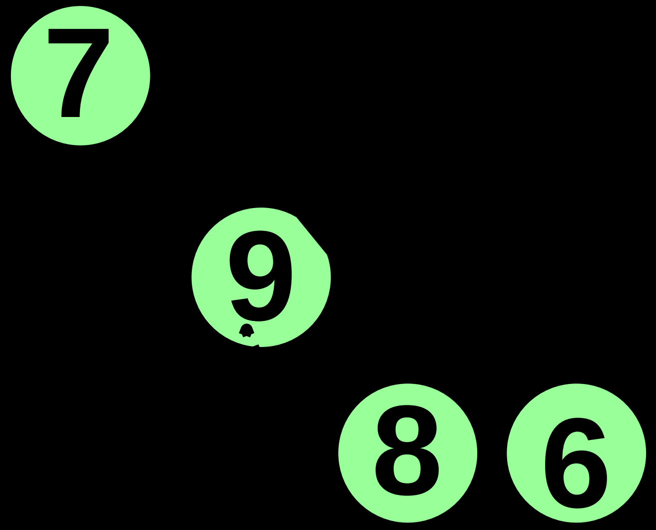
ILBFS



Case 1:

Best is a child of oldbest

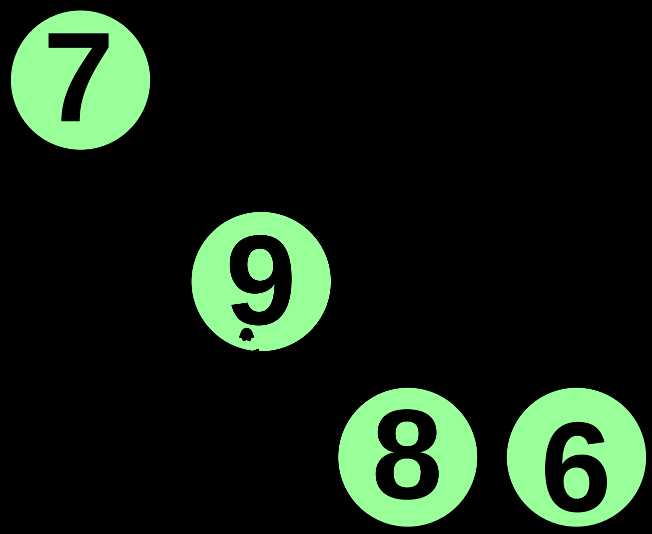
ILBFS



Case 1:

Best is a child of oldbest

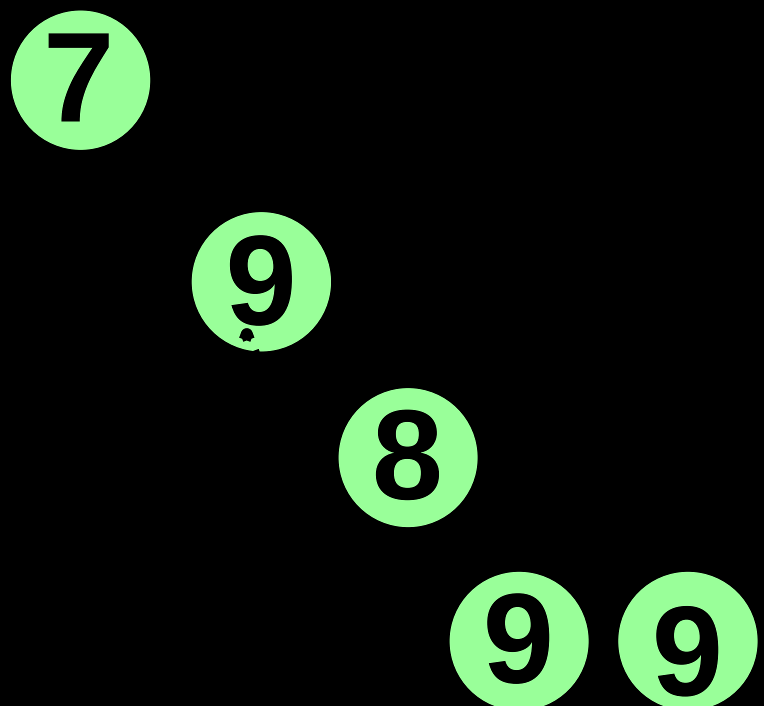
ILBFS



Case 1:

Best is a child of oldbest

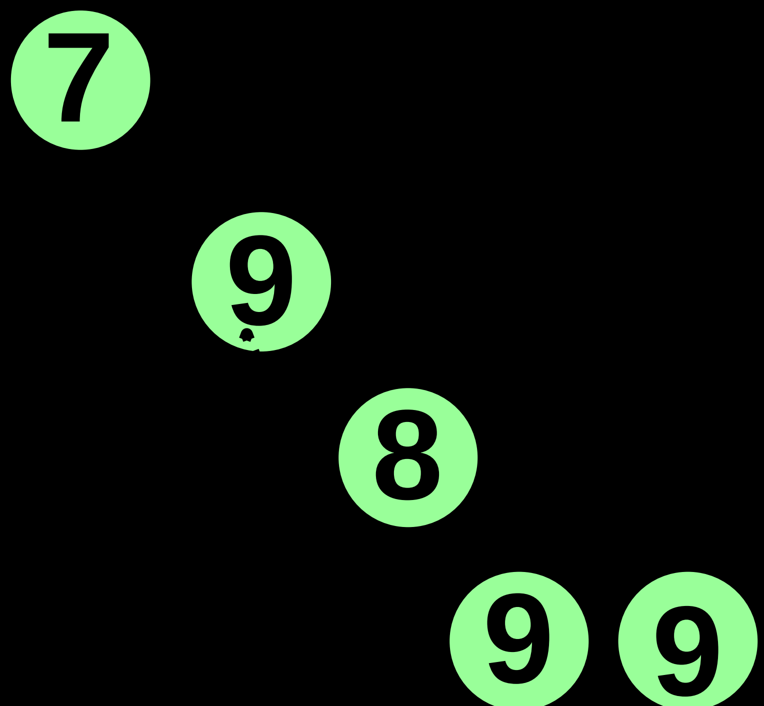
ILBFS



Case 1:

Best is a child of oldbest

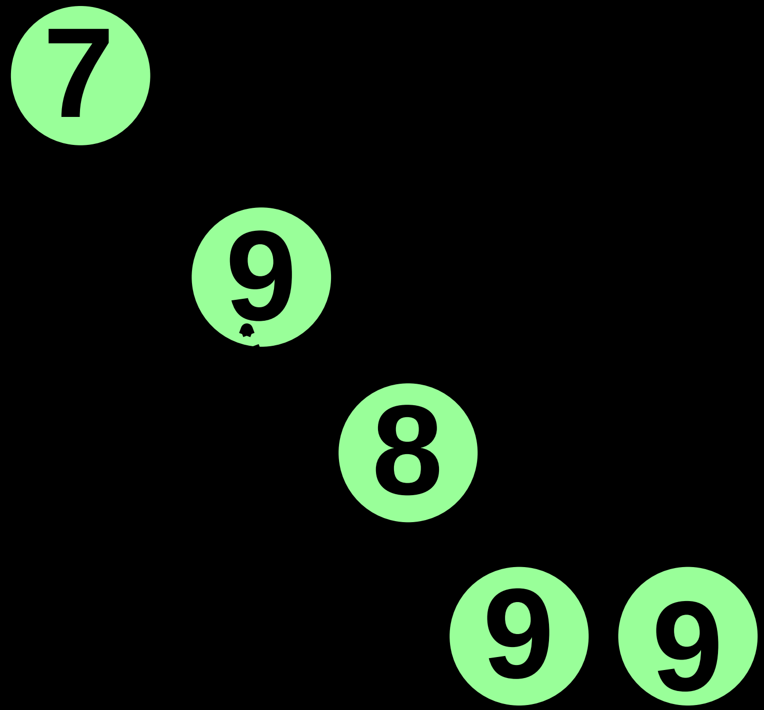
ILBFS



Case 2:

Best is not a child of oldbest

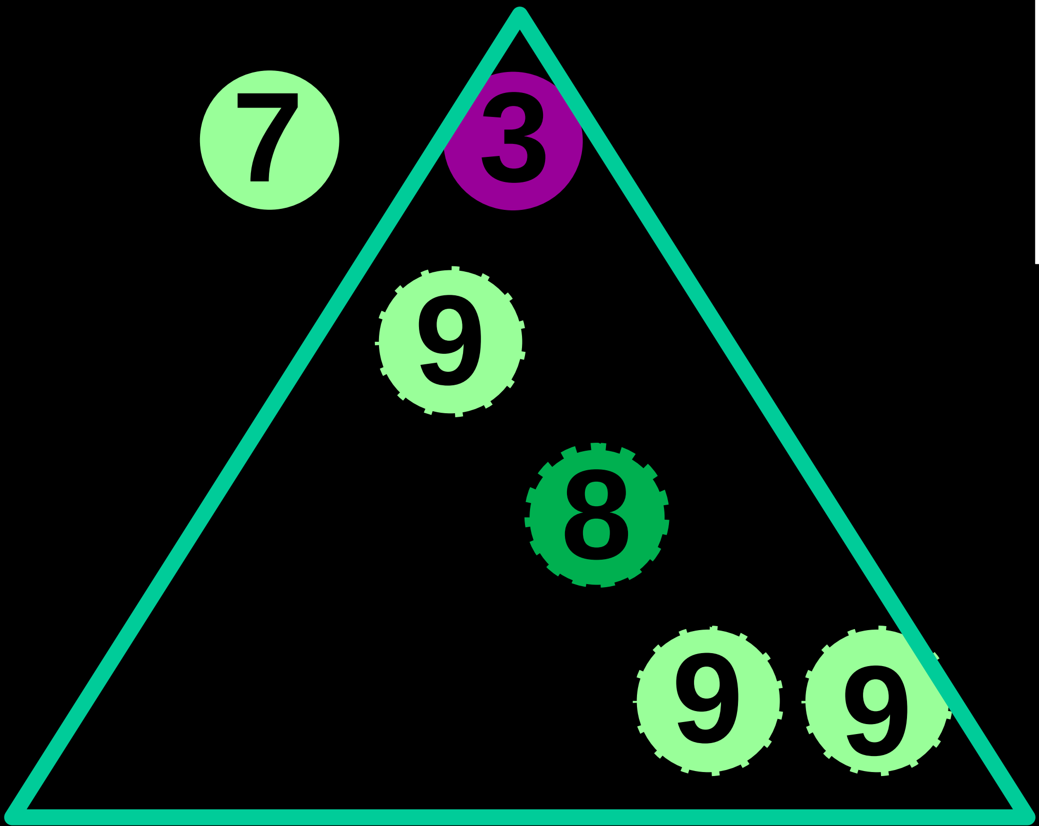
ILBFS



Case 2:

Best is not a child of oldbest

ILBFS

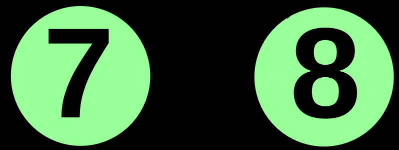


collapse

Case 2:

Best is not a child of oldbest

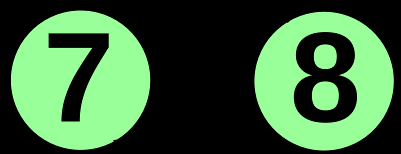
ILBFS



Case 2:

Best is not a child of oldbest

ILBFS



Case 2:

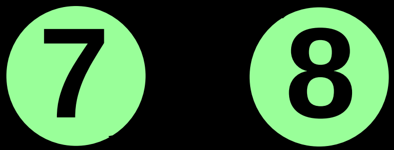
Best is a collapsed node

Restore

f=3, F=7

DFS(7)

ILBFS



Case 2:

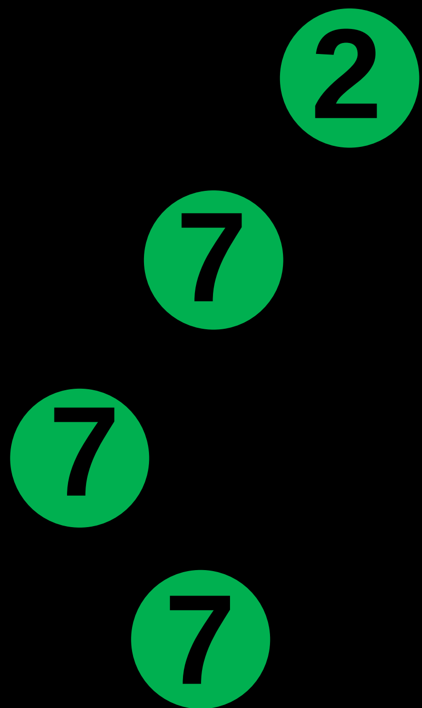
Best is a collapsed node

Restore

f=3, F=7

DFS(7)

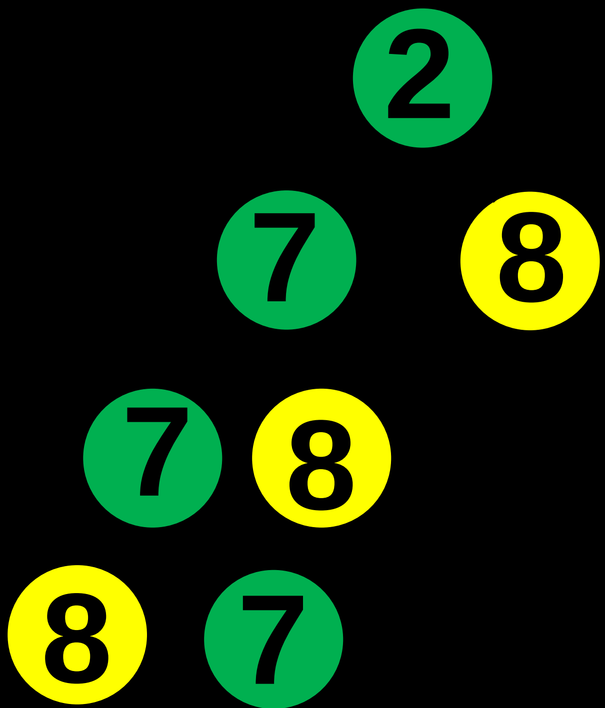
ILBFS



Case 2:

Best is a collapsed node

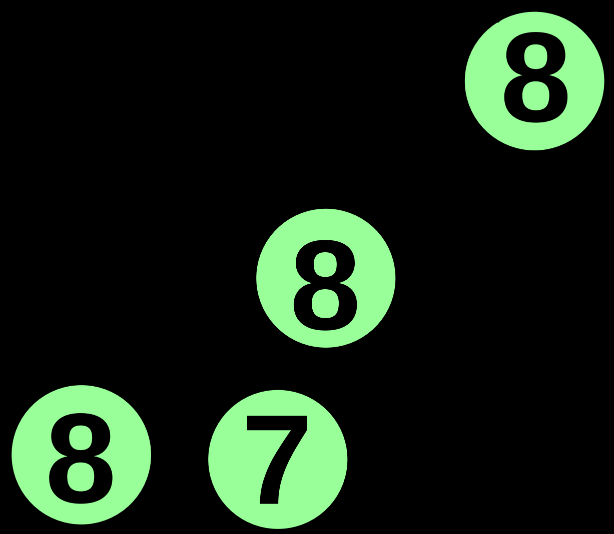
ILBFS



Case 2:

Best is a collapsed node

ILBFS



Case 2:

Best is a collapsed node

Linear-space best-first search

Iterative variant - ILBFS

Algorithm 1: High-level ILBFS

Input: Root R

Insert R into OPEN and TREE
oldbest=NULL
while *OPEN not empty* **do**
  best=extract_min(OPEN)
  if *goal(best)* **then**
    exit
  if *oldbest* $\neq$ *best.parent* **then**
    $B \leftarrow$ sibling of *best* that is ancestor of *oldb*
    collapse(B)
  if *best.C*=*True* **then**
    best $\leftarrow$ restore(*best*)
  foreach *child C of best* **do**
    Insert C to OPEN and TREE
  oldbest $\leftarrow$ *best*

Recursive variant - RBFS

RBFS(n, B)
1. if n is a goal
2. $\quad solution \leftarrow n$; exit()
3. $C \leftarrow expand(n)$
4. if C is empty, return ∞
5. for each child n_i in C
6. $\quad$ if $f(n) < F(n)$ then $F(n_i) \leftarrow max(F(n), f(n_i))$
7. $\quad$ else $F(n_i) \leftarrow f(n_i)$
8. $(n_1, n_2) \leftarrow best_F(C)$
9. while $(F(n_1) \leq B$ and $F(n_1) < \infty)$
10. $\quad F(n_1) \leftarrow$ RBFS$(n_1, min(B, F(n_2)))$
11. $\quad (n_1, n_2) \leftarrow best_F(C)$
12. return $F(n_1)$

Restoring?

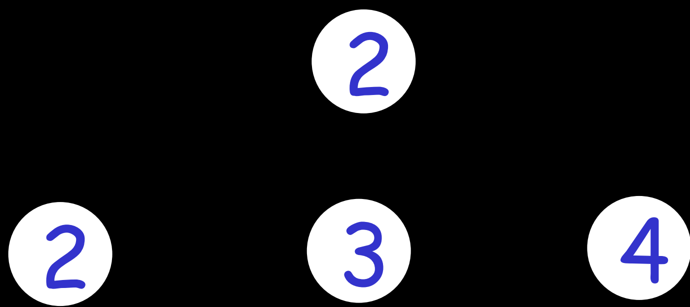
Ordering child nodes

Ordering child nodes
with new F-values

ILBFS/RBFS Summary

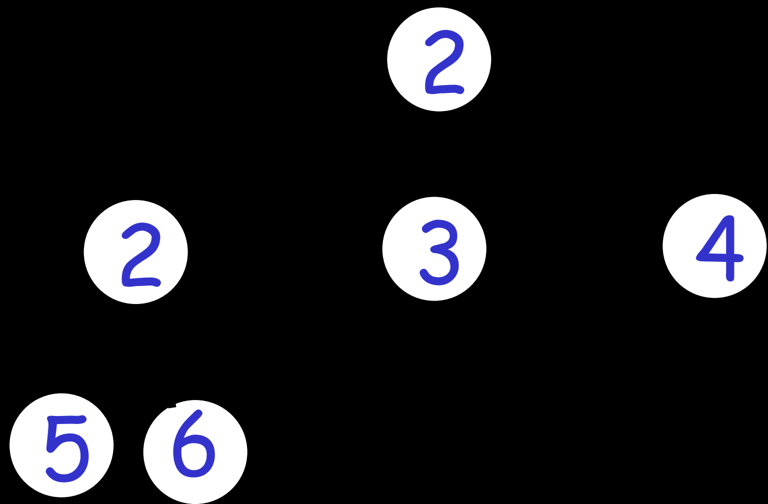
- More efficient than IDA* and still optimal
  - Best-first Search based on next best f-value-threshold (countour); fewer regeneration of nodes
  - Exploit results of search at a specific f-value-threshold by saving next f-value-threshold associated with a node who successors have been explored.
- Like IDA* still suffers from excessive node regeneration
- IDA* and RBFS not good for graphs
- Can’t check for duplicates other than those on current path
- Both are hard to characterize w.r.t. expected time complexity

SMA* (Simplified Memory Bounded A*) [#2:Russell 1992]



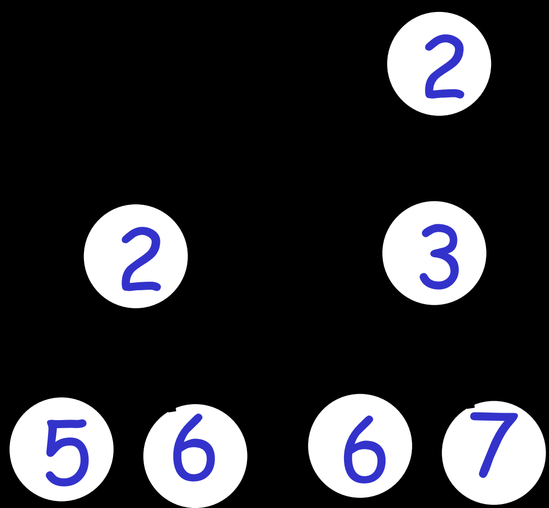
2 3 4

SMA* [Russell 1992]



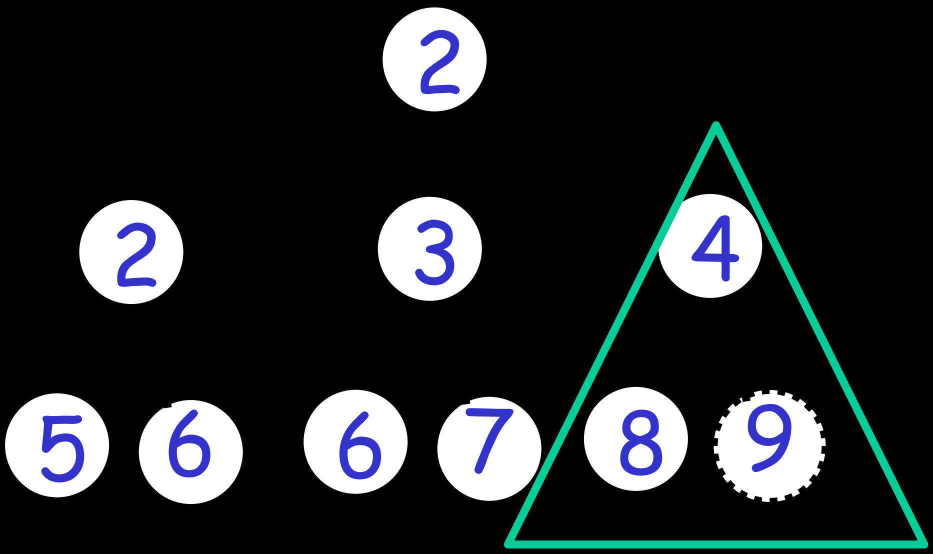
3 4 5 6

SMA* [Russell 1992]



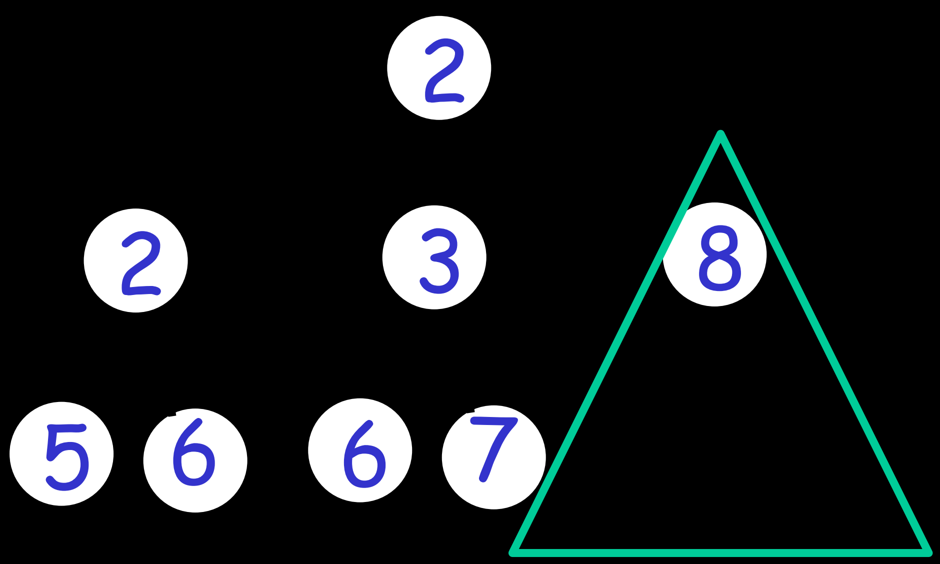
4 5 6 6 7

SMA* [Russell 1992]



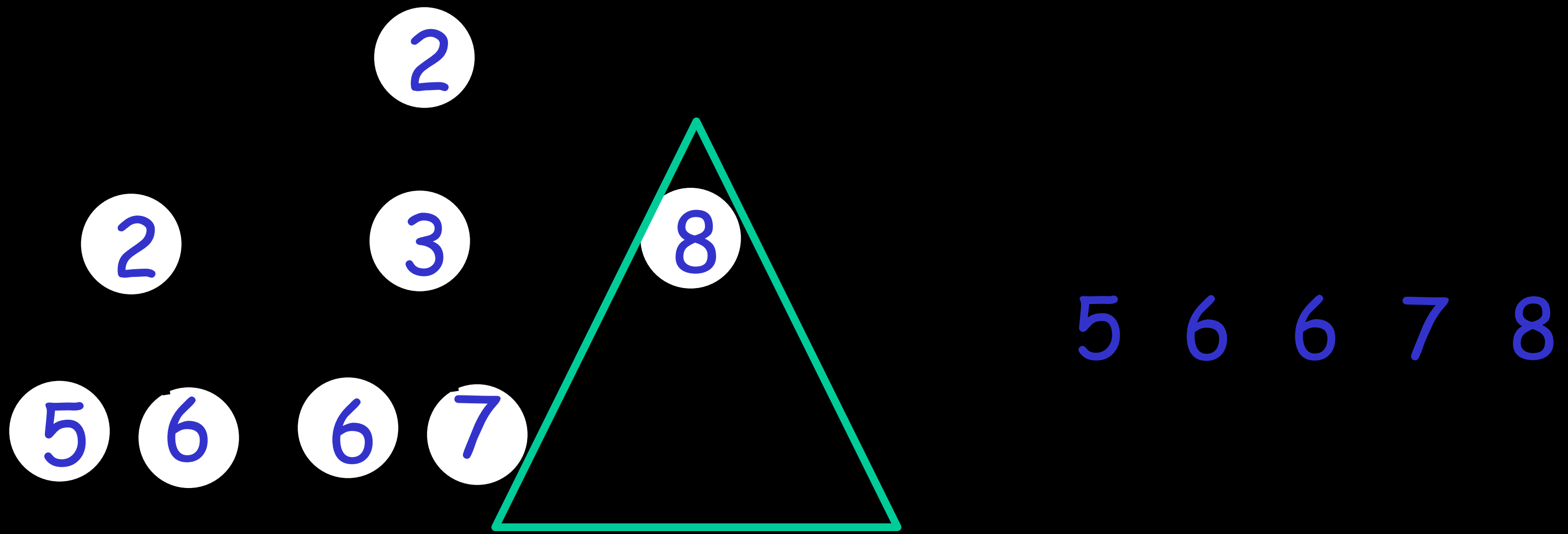
5 6 6 7 8 9

SMA* [Russell 1992]



5 6 6 7 8

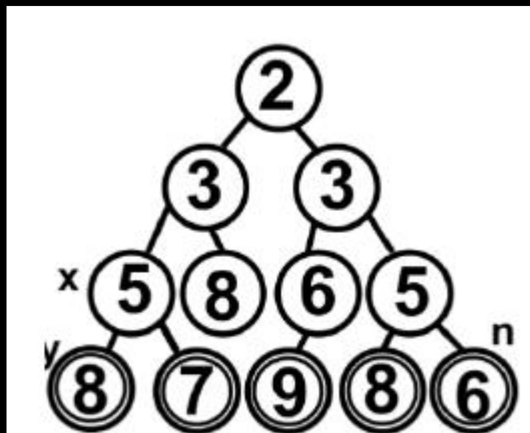
SMA* [Russell 1992]

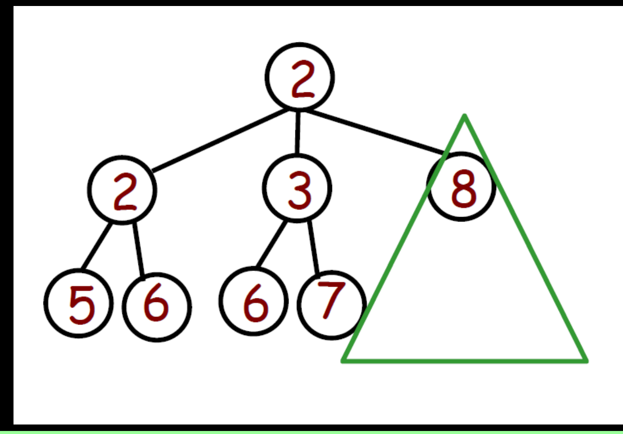


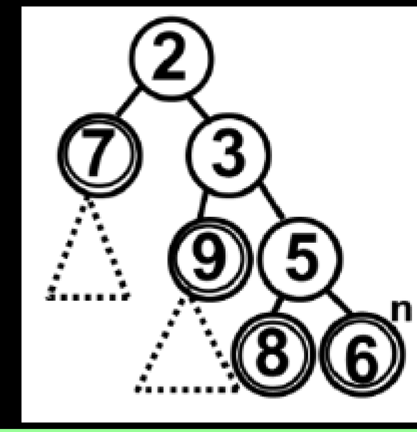
SMA* uses a variant of pathmax for its restore macro

SMA* Summary

- It is complete, provided the available memory is sufficient to store the shallowest solution path.
- It is optimal, if enough memory is available to store the shallowest optimal solution path. Otherwise, it returns the best solution (if any) that can be reached with the available memory.
- Can keep switching back and forth between a set of candidate solution paths, only a few of which can fit in memory (thrashing)
  - **Memory limitations can make a problem intractable wrt time**
- With enough memory for the entire tree, same as A*







memory	b^d	M	$O(d)$
collapse	never	lazily	eagerly

continuum