Memory-bounded heuristic search

Instructor

LE Thanh Sach, Ph.D.

Instructor's Information

LE Thanh Sach, Ph.D.

Office:

Department of Computer Science,
Faculty of Computer Science and Engineering,
HoChiMinh City University of Technology.

Office Address:

268 LyThuongKiet Str., Dist. 10, HoChiMinh City, Vietnam.

E-mail: LTSACH@hcmut.edu.vn

E-home: http://cse.hcmut.edu.vn/~ltsach/

Tel: (+84) 83-864-7256 (Ext: 5839)

Acknowledgment

The slides in this PPT file are composed using the materials supplied by

- **Prof. Stuart Russell and Peter Norvig**: They are currently from University of California, Berkeley. They are also the author of the book "Artificial Intelligence: A Modern Approach", which is used as the textbook for the course

- **Prof. Tom Lenaerts**, from Université Libre de Bruxelles

Recap

Strategies	Search Algorithm	Completeness	Optimality	Time Complexity	Space Complexity
Best-First Search, h(n): admissible	Tree Search	No	No	$O(b^m)$	$O(b^m)$
	Graph Search	Yes	No	$O(b^m)$	$O(b^m)$
A*, h(n): admissible, consistent	Tree Search	Yes	Yes	$O(b^m)$	$O(b^m)$
	Graph Search	Yes	Yes	$O(b^m)$	$O(b^m)$

Good heuristics can provide enormous savings compared to the use of uninformed searches; however, memory requirement is still exponential in the worst cases

=> Impractical for almost real applications with an "Out-of-Memory Error"!

Recap

Strategies	Search Algorithm	Completeness	Optimality	Time Complexity	Space Complexity
Best-First Search, h(n): admissible	Tree Search	No	No	$O(b^m)$	$O(b^m)$
	Graph Search	Yes	No	$O(b^m)$	$O(b^m)$
A*, h(n): admissible, consistent	Tree Search	Yes	Yes	$O(b^m)$	$O(b^m)$
	Graph Search	Yes	Yes	$O(b^m)$	$O(b^m)$

Need searching algorithms that are able to reduce the space complexity

Memory-bounded heuristic search

- Some solutions to A* space problems (maintain completeness and optimality)
  - Iterative-deepening A* (IDA*)
    - ✓ Here cutoff information is the *f*-cost (*g*+*h*) instead of depth
  - Recursive best-first search(RBFS)
    - ✓ Recursive algorithm that attempts to mimic standard best-first search with linear space.
  - (simple) Memory-bounded A* ((S)MA*)
    - ✓ Drop the worst-leaf node when memory is full

Iterative-Deepening A*

Iterative-Deepening A*

function ITERATIVE_DEEPENING_SEARCH(*problem*) **return** a solution or failure

inputs: *problem*

for *depth* ← 0 to ∞ **do**
    result ← DEPTH-LIMITED_SEARCH(*problem*, *depth*)
    if *result* ≠ *cuttoff* **then return** *result*

Original IDS

IDA* is similar to IDS, but it uses f-cost (i.e., g + h) instead of depth to force the backtracking.

Iterative-Deepening A*

```
function ITERATIVE_DEEPENING_ASTAR(problem) return a solution or failure
    inputs: problem

    new-limit = ∞
    limit = 0
    while depth < ∞:
        result ← COST-LIMITED-ASTAR(problem, limit, new-limit)
        depth = new-limit
        if result ≠ cuttoff then return result
```

IDA*

new-limit: passed-by-reference to update

Iterative-Deepening A*

function COST-LIMITED-ASTAR(*problem,limit, **new-limit***) **return** a solution or failure/cutoff
    return RECURSIVE-COST-ASTAR (MAKE-NODE(INITIAL-STATE[*problem*]),
                                    problem, *limit, **new-limit***)

function RECURSIVE-COST-ASTAR(*node*, *problem, limit, **new-limit***) **return** a solution or failure/cutoff
    cutoff_occurred? ← false
    if GOAL-TEST[*problem*](STATE[*node*]) **then return** SOLUTION(*node*)
    else if DEPTH[*node*] >= *limit* then:
        new-limit t = min(***new-limit***, node.f)
        return *cutoff*
    else for each *successor* **in** EXPAND(*node*, *problem*) **do**
        result ← RECURSIVE-COST-ASTAR(*successor*, *problem*, *limit, **new-limit***)
        if *result* == *cutoff* **then** *cutoff_occurred?* ← true
        else if *result* ≠ *failure* **then return** *result*
    if *cutoff_occurred?* **then return** *cutoff* **else return** failure

Update *new-limit* with the smallest node.f

Iterative-Deepening A*

Strategies	Completeness	Optimality	Time Complexity	Space Complexity
IDA*	Yes	Yes	$O(b^d)$	$O(bd)$

Linear with the solution's depth (d)

- Constant step-cost: ε
- Cost of the optimal path: C^*

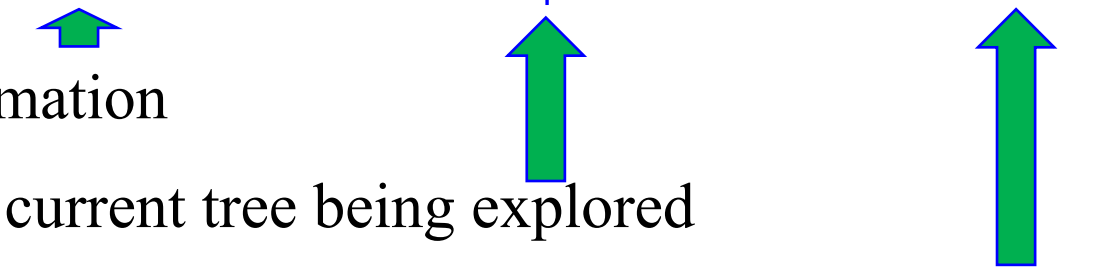
$\Rightarrow d = C^*/\varepsilon$

If ε is a tiny real-valued ➔ d is also too big!

Recursive best-first search (RBFS)

Recursive best-first search

function RECURSIVE-BEST-FIRST-SEARCH(*problem*) **return** a solution or failure
return RFBS(*problem*, MAKE-NODE(INITIAL-STATE[*problem*]), limit = ∞)



Problem's information

Root of the current tree being explored

The cost limitation for the path found by RBFS:

- RFBS will call iteself recursively

Recursive best-first search

function RFBS(*problem, node, f_limit*) **return** a solution or failure and a new *f-cost* limit

if GOAL-TEST[*problem*](STATE[*node*]) **then return** *node*

successors ← EXPAND(*node, problem*)

if *successors* is empty **then return** failure, ∞

for each *s* **in** *successors* **do**

s.f ← max(*s.g* + *s.h*, *node.f*) //update f with value from previous search, if any

repeat

best ← the lowest *f*-value node in *successors*

if *best.f* > *f_limit* **then return** failure, *best.f*

alternative ← the second lowest *f*-value among *successors*

result, *best.f* ← RBFS(*problem, best,* min(*f_limit, alternative*))

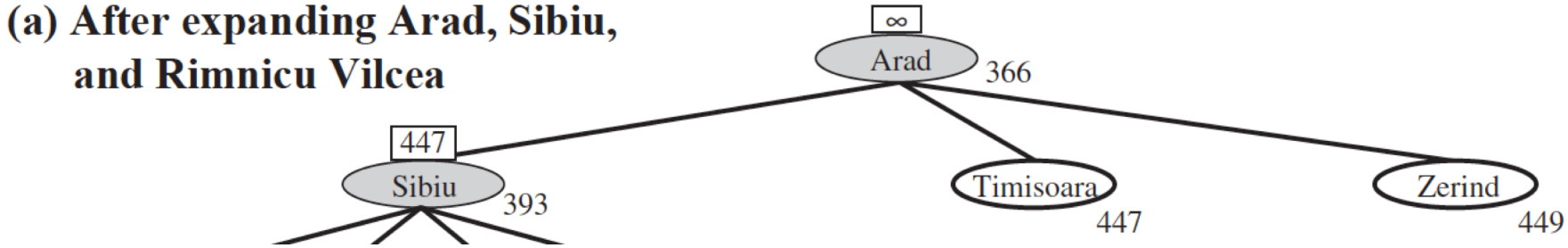
if *result* ≠ failure **then return** *result*

Update the f-cost for the best node

Recursive best-first search

- Keeps track of the f-value of the best-alternative path available.
  - If current f-values exceeds this alternative f-value than backtrack to alternative path.
  - Upon backtracking change f-value to best f-value of its children.
  - Re-expansion of this result is thus still possible.

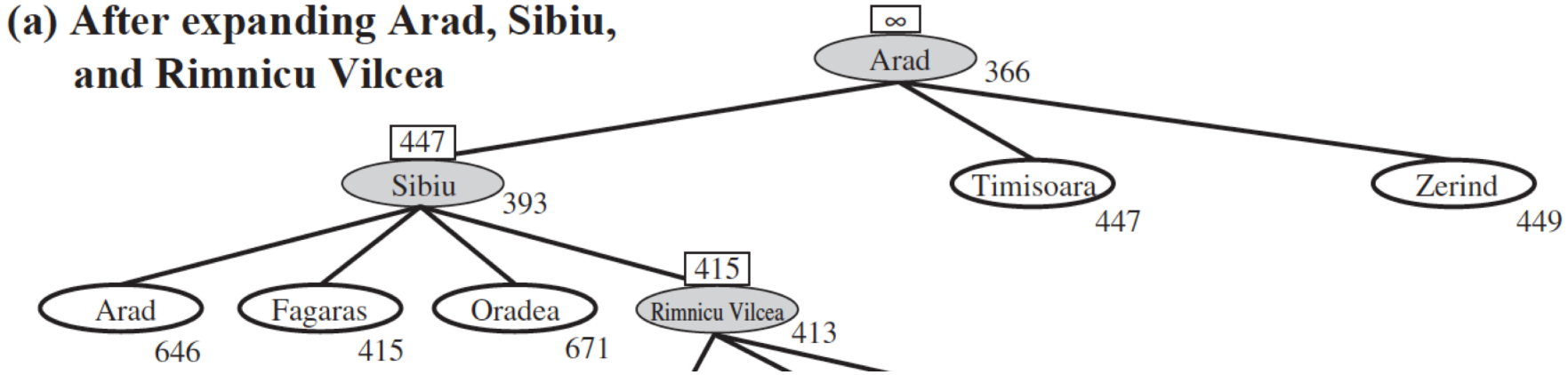
Recursive best-first search, ex.



RFBS(problem, node[**Arad**], f_limit= ∞)

- Arad has three children: Sibiu, Timisoroa, and Zerind
- Sibiu will be explored next, because its f-cost, 393 < ∞
  - New f_limit = min(447, ∞) = 447
  - f_limit = 447 means "*do not go deeper on the tree rooted at **Sibiu** if the best cost is larger than 447, try **Timisoara** instead!*"

Recursive best-first search, ex.

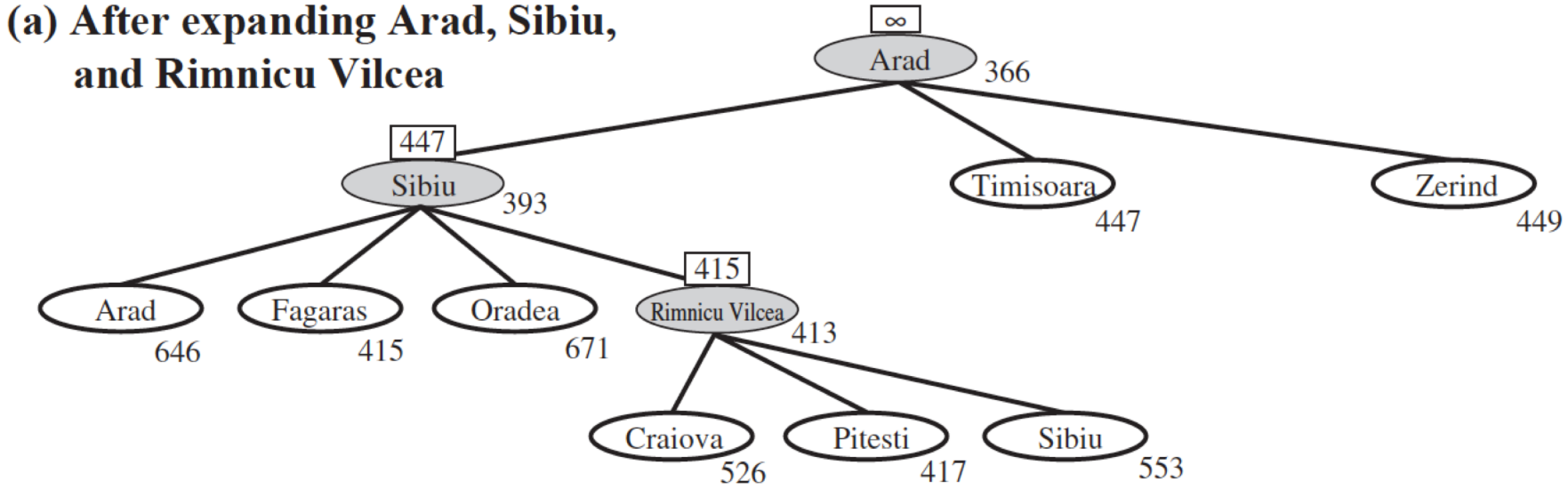


RFBS(problem, node[**Arad**], f_limit= ∞)

→ RFBS(problem, node[**Sibiu**], f_limit= 447)

- Sibiu has four children, including Arad (repeated)
- Rimnicu Vilcea will be explored next, because its f-cost, 413 < 447
  - New f_limit = min(415, 447) = 415: means "*do not go deeper on tree rooted at* **Rimnicu Vilcea** *if the best cost is larger than 415, try* ***Fagaras*** *instead!*"

Recursive best-first search, ex.



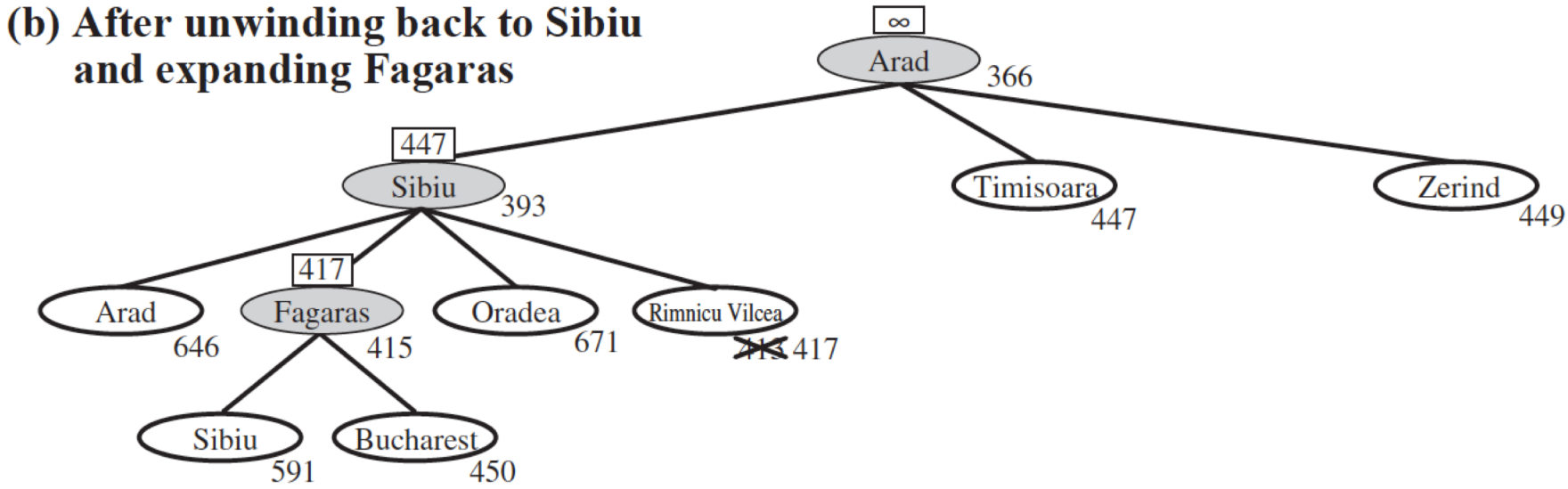
RFBS(problem, node[**Arad**], f_limit= ∞)

→ RFBS(problem, node[**Sibiu**], f_limit= 447)

→ RFBS(problem, node[**Rimnicu Vilcea**], f_limit= 415)

- **Rimnicu Vilcea** has three children, including Sibiu (repeated)
- No child has f-cost smaller than f_limit (415)
  - => Do backtracking, try **Fagaras (415)** instead
  - => Allow **Rimnicu Vilcea** **being re-explored** if all other paths larger than its best-child, i.e., Pitesti (417), by updating **Rimnicu Vilcea's f-cost = 417**

Recursive best-first search, ex.



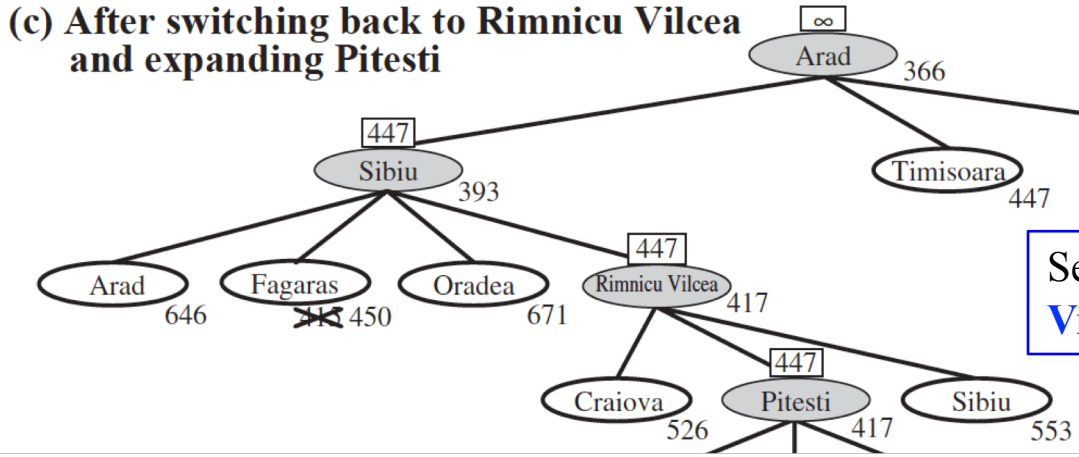
RFBS(problem, node[**Arad**], f_limit= ∞)

→ RFBS(problem, node[**Sibiu**], f_limit= 447)

→ RFBS(problem, node[**Fagaras**], f_limit= 417)

- Rimnicu Vilcea's f-cost = 417 (updated)
- Best node = Fagaras, its f-cost = 415 => explore Fagaras with f_limit = min(417, 447) = 417
- Fagaras has two children, including Sibiu (repeated), all children's f-cost > 417
  - => Do backtracking, try **Rimnicu Vilcea (417) instead**
  - => Allow **Fagaras** being re-explored if all other paths larger than its best-child, i.e., Bucharest, by updating **Fagaras's f-cost = 450**

Recursive best-first search, ex.



Second time exploring **Rimnicu Vilcea,** f_limit = min(447, 450)

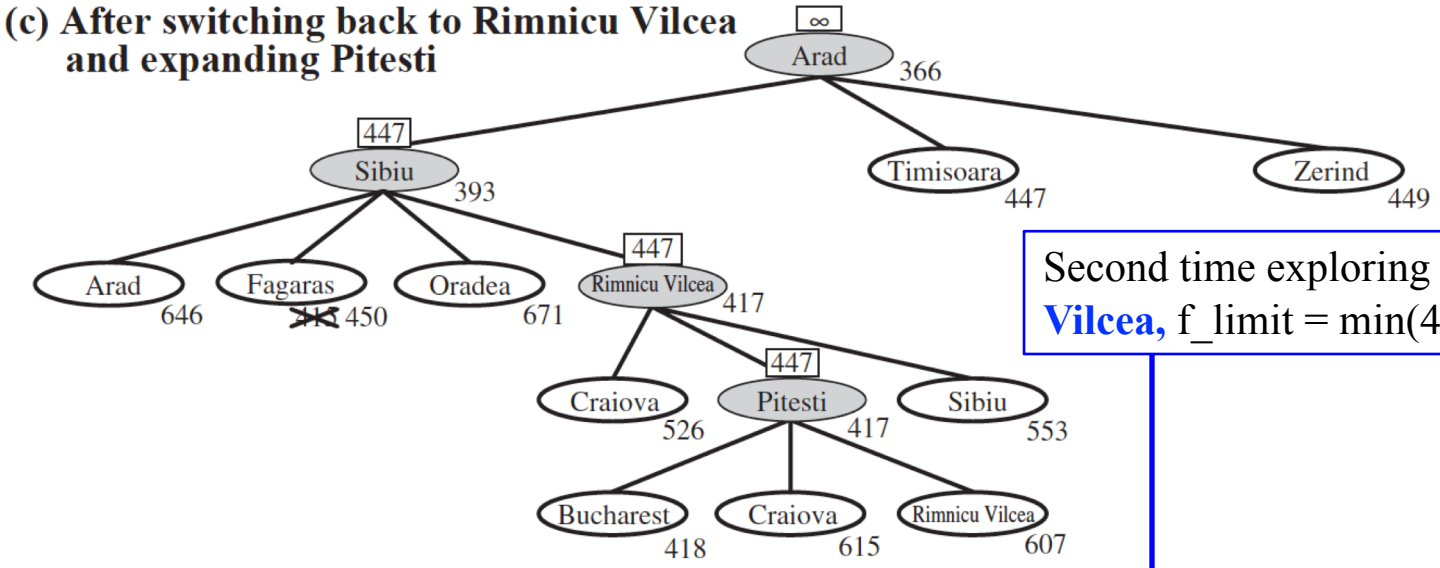
RFBS(problem, node[**Arad**], f_limit= ∞)

→ RFBS(problem, node[**Sibiu**], f_limit= 447)

→ RFBS(problem, node[**Rimnicu Vilcea**], f_limit= 447)

- **Rimnicu Vilcea** has three children, Pitesti's f-cost = 417 < 447
- **Pitesti** will be explored next, with f_limit =min(447, 526) = 447

Recursive best-first search, ex.



Second time exploring **Rimnicu Vilcea,** f_limit = min(447, 450)

RFBS(problem, node[**Arad**], f_limit= ∞)

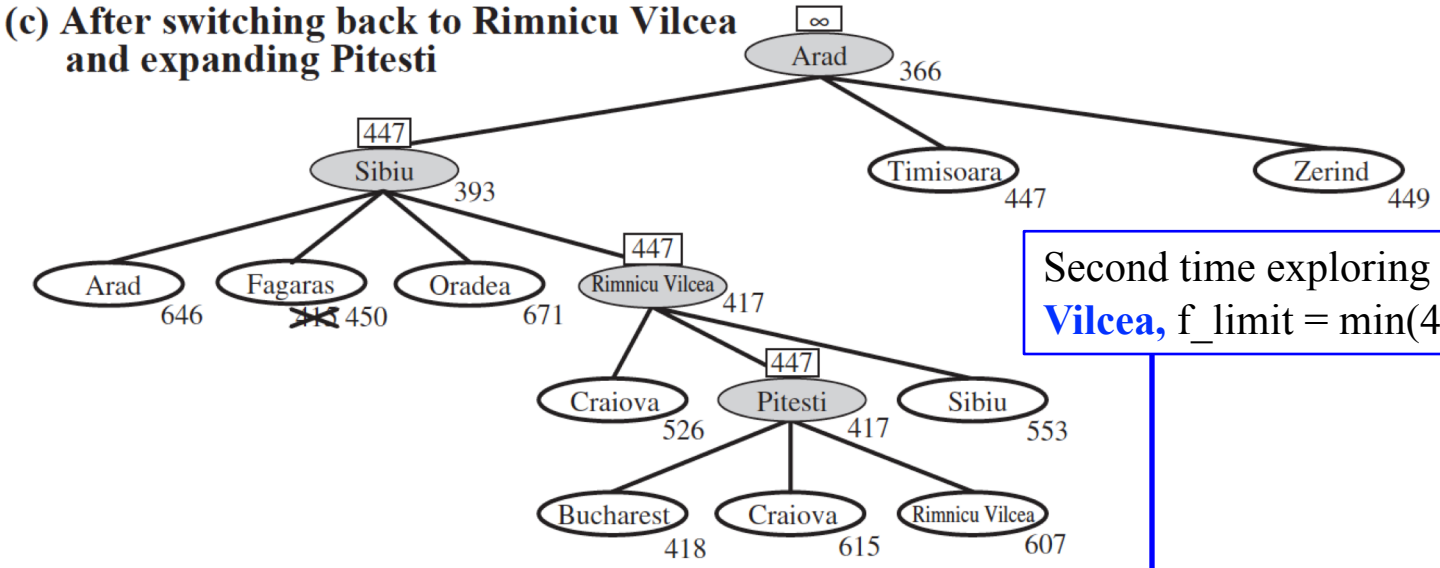
→ RFBS(problem, node[**Sibiu**], f_limit= 447)

→ RFBS(problem, node[**Rimnicu Vilcea**], f_limit= 447)

→ RFBS(problem, node[**Pitesti**], f_limit= 447)

- **Pitesti** has three children, Bucharest's f-cost = 418 < 447
- **Bucharest** will be explored next, with f_limit =min(447, 607) = 447

Recursive best-first search, ex.



Second time exploring **Rimnicu Vilcea,** f_limit = min(447, 450)

RFBS(problem, node[**Arad**], f_limit= ∞)

→ RFBS(problem, node[**Sibiu**], f_limit= 447)

→ RFBS(problem, node[**Rimnicu Vilcea**], f_limit= 447)

→ RFBS(problem, node[**Pitesti**], f_limit= 447)

→ RFBS(problem, node[**Buchrest**], f_limit= 447)

- Bucharest is a goal => STOP
  - Path = [Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest],
  - Path's cost = 418

RBFS evaluation

- RBFS is a bit more efficient than IDA*
  - Still excessive node generation (mind changes)
  - For example, re-generate "Rimnicu Vilcea" in previous slides
- Like A*, optimal if *h(n)* is admissible
- Time complexity difficult to characterize
  - Depends on accuracy if h(n) and how often best path changes.

RBFS evaluation

- RBFS is a bit more efficient than IDA*
  - Still excessive node generation (mind changes)
  - For example, re-generate "Rimnicu Vilcea" in previous slides
- Like A*, optimal if *h(n)* is admissible
- Time complexity difficult to characterize
  - Depends on accuracy if h(n) and how often best path changes.
- Space complexity is *O(bd)*.
  - IDA* retains only one single number (the current f-cost limit)

RBFS evaluation

Calling stack:

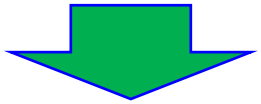
RFBS(problem, node[**Arad**], f_limit= ∞) → Store b children

→ RFBS(problem, node[**Sibiu**], f_limit= 447) → Store b children

→ RFBS(problem, node[**Rimnicu Vilcea**], f_limit= 447) → Store b children

→ RFBS(problem, node[**Pitesti**], f_limit= 447) → Store b children

→ RFBS(problem, node[**Buchrest**], f_limit= 447) → Store b children



$O(bd)$

RBFS evaluation

- RBFS is a bit more efficient than IDA*
- Like A*, optimal if *h(n)* is admissible
- Time complexity difficult to characterize
  - Depends on accuracy if h(n) and how often best path changes.
- Space complexity is *O(bd)*.
  - IDA* retains only one single number (the current f-cost limit)
- IDA* en RBFS suffer from ***too little*** memory.
  - If there is more memory => IDA* en RBFS cannot utilize
  - IDA* en RBFS forget everything between iterations/recursive-calls => re-explore a node many times

RBFS's Exercises

- Run RBFS step-by-step to find a solution for going from "Zerind" to "Bucharest"
  - How many times RBFS does explore "Sibiu" and what is f_limit for each of exploration?
- 8-puzzle game
  - Give a start and a goal state, maybe randomly generated
  - Run RBFS step-by-step to find the solution
    - ✓ Try RBFS with the following heuristics: "num of misplaced tiles" and "manhattan distance". Which heuristic is more efficient?

Simplified memory-bounded A* (SMA*)

(simplified) memory-bounded A*

- SMA*
  - Work similar to A*,
    - ✓ but it allocates "frontier" and "explored-list" as big as it can (depending on the system's memory)
  - Expand best leafs until available memory is full
  - When full,
    - ✓ SMA* drops worst leaf node (highest *f*-value)
  - Like RFBS,
    - ✓ it backups the f-cost of the forgotten node to its parent

(simplified) memory-bounded A*

- What if all leafs have the same *f*-value?
  - Same node could be selected for expansion and deletion.
  - SMA* solves this by expanding *newest* best leaf and deleting *oldest* worst leaf.
- SMA* is complete if solution is reachable, optimal if optimal solution is reachable.