PRIORITY QUEUE AND APPLICATIONS

Partha P Chakrabarti
Indian Institute of Technology Kharagpur

Overview of Algorithm Design

1. Initial Solution
   a. Recursive Definition – A set of Solutions
   b. Inductive Proof of Correctness
   c. Analysis Using Recurrence Relations
2. Exploration of Possibilities
   a. Decomposition or Unfolding of the Recursion Tree
   b. Examination of Structures formed
   c. Re-composition Properties
3. Choice of Solution & Complexity Analysis
   a. Balancing the Split, Choosing Paths
   b. Identical Sub-problems
4. Data Structures & Complexity Analysis
   a. Remembering Past Computation for Future
   b. Space Complexity
5. Final Algorithm & Complexity Analysis
   a. Traversal of the Recursion Tree
   b. Pruning
6. Implementation
   a. Available Memory, Time, Quality of Solution, etc

---

1. Core Methods
   a. Divide and Conquer
   b. Greedy Algorithms
   c. Dynamic Programming
   d. Branch-and-Bound
   e. Analysis using Recurrences
   f. Advanced Data Structuring

Stacks
Queues
Lists
Arrays

Heap,
BST

2. Important Problems to be addressed
   a. Sorting and Searching
   b. Strings and Patterns
   c. Trees and Graphs
   d. Combinatorial Optimization

Priority Queue

3. Complexity & Advanced Topics
   a. Time and Space Complexity
   b. Lower Bounds
   c. Polynomial Time, NP-Hard
   d. Parallelizability, Randomization

Priority Queue: Operations & Applications

Queue : FIFO

→ Elements have key values which could have an ordering

- sorting, searching, optimization problems, max-min, max-next max
- Pole Wiring Problem
- Coins
- Activity Selection

operations

- insert
- delete Max/Min
- update
- delete element
- find ←

Heap

BST

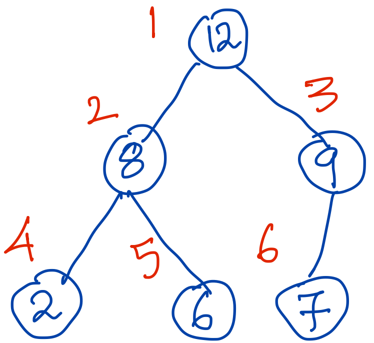
static : –

dynamic : –

Priority Queue: Heap Implementation

Build Heap
Insert-new element ←
Remove Min / Max

update element → provided
delete element } we have pointer or index of node



1. Complete Binary Tree
2. Heap property

1	2	3	4	5	6	7
12	8	9	2	6	7	

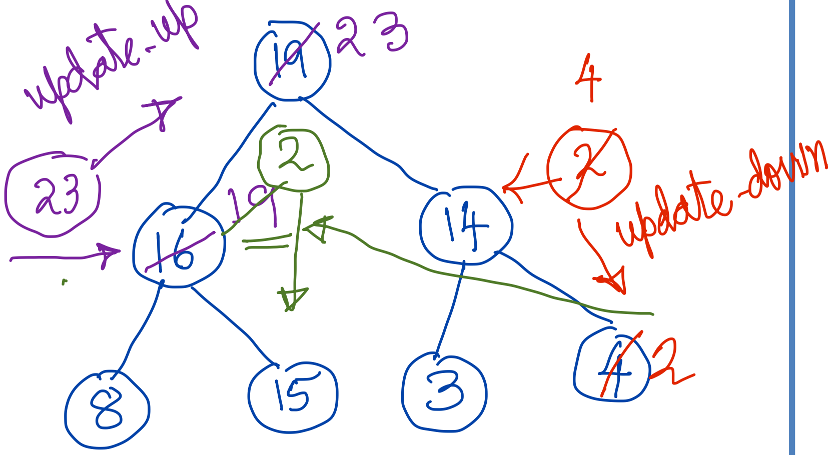
insert-new
↳ • inserts at the end
• update-up (A, node)
O(log n)

remove_max
↳ • remove to root or first element
O(log n) • replace root by last element
• update-down (A, node)

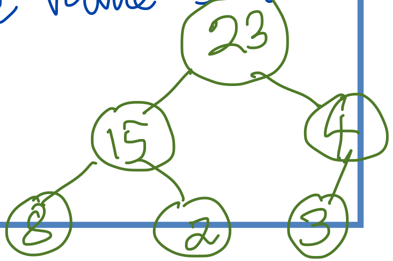
Heapify

Build-heap
Bottom up update-down iteratively
O(n)

Heap Priority Queue: Update / Revise / Delete



We need the index/pointer to the node whose value is to be updated



Find(key) → O(n)

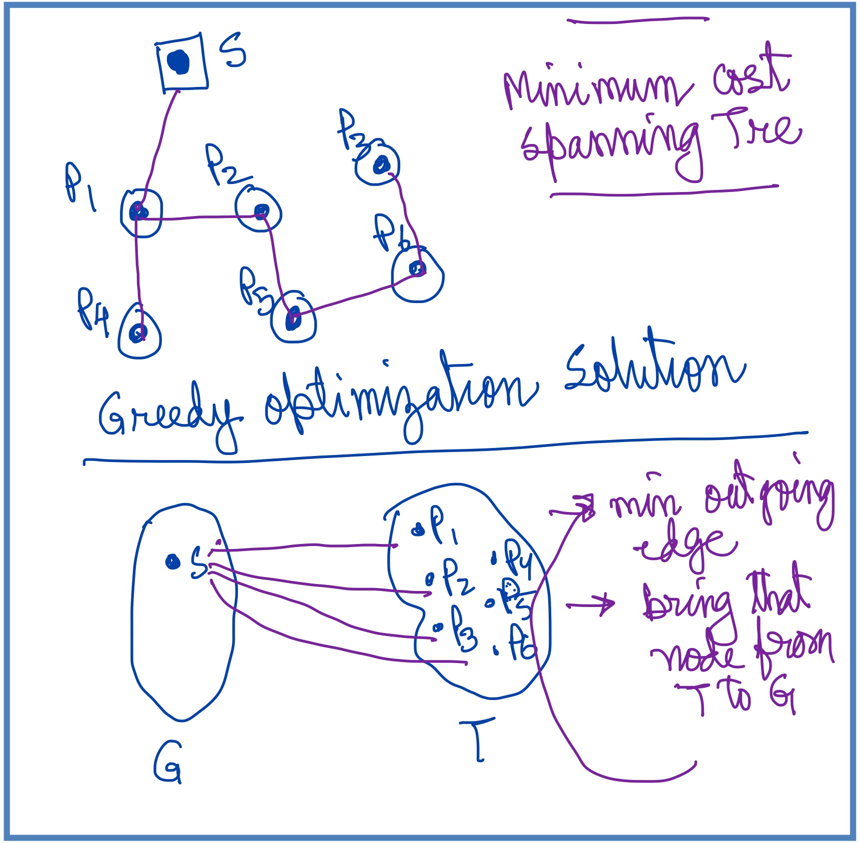
update(A, node)
{ check whether the key(node) is greater than parent
 ↳ if so → update_up(A, node)
else
 ↳ update_down(A, node)

O(logn)

delete(A, node)
{ - remove(A, node)
 - replace by the last element of heap, decrement size
 - update(A, node)
}

O(logn)

Electrical Pole Placement Problem



What data structure to use?

operations :- PRIORITY QUEUE

R: Set of edges between G & T

- insert-new
- delete
- remove-min

→ Edges (e)

→ Nodes (n)

each operation will be of $O(e)$

$$O(n \log e + e \log e)$$

$$= O(e \log e)$$

$$= O(n^2 \log n)$$

$$e = O(n^2)$$

Evaluation of Recursion Tree

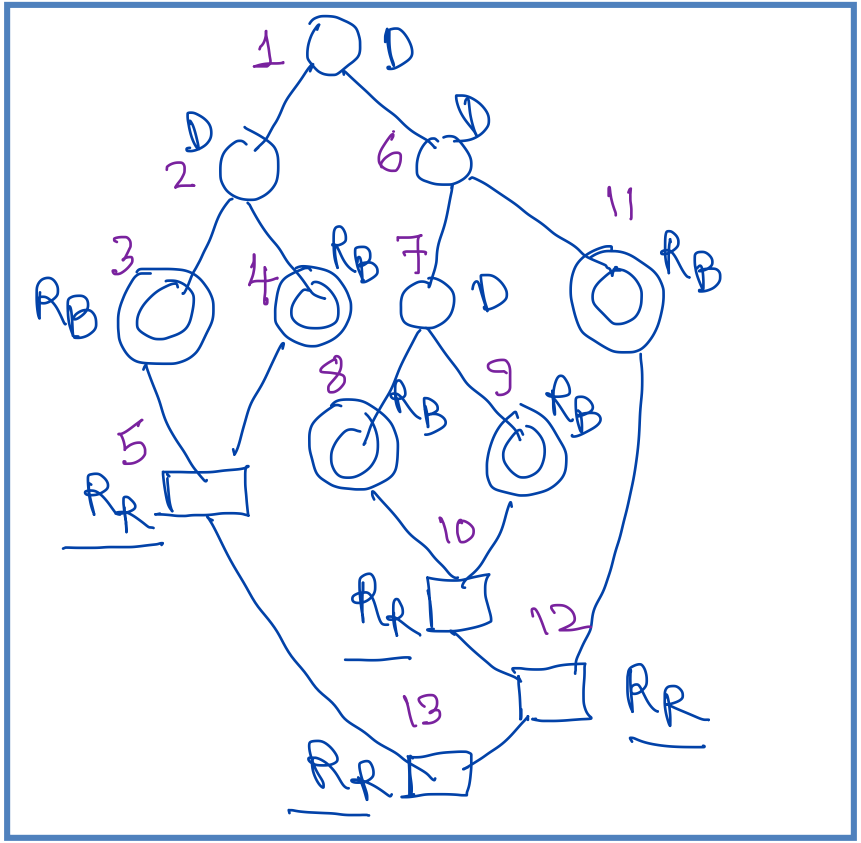
$$f(x) = R_B(x) \text{ if } B(x)$$

$$= \{ \; 1.\ (y_1, y_2, \ldots, y_k) = D(x)$$

$$2.\ R_R\big(f(y_1), f(y_2), \ldots, f(y_k)\big) \; \}$$

Evaluation of the Recursion Tree

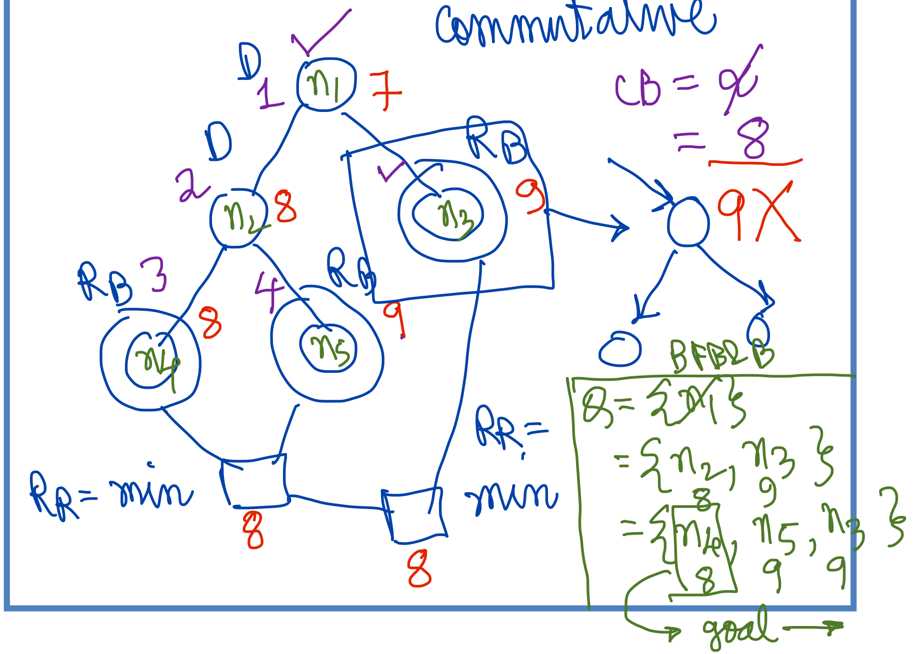
- Depth-first → Stack
- Breadth-first → Queue
- Iterative Deepening



Pruning & Branch & Bound

optimization problems : min/max

R_R → min/max
↳ associative & commutative



pruning

DFB&B : — Maintain a current Best (initialized to ∞)

- whenever we find a solution we update current best to the better solution
- Any node which has got cost ≥ current best is pruned

Best first B&B

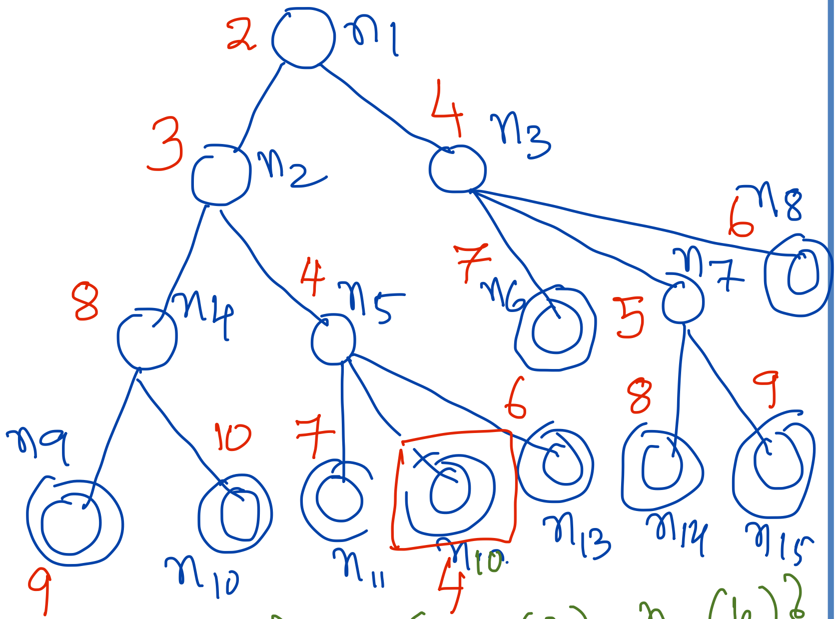
- Ordered search using a priority queue of costs :
  - Remove-Min , insert-children , stop when goal is reached .

Best First Search using Priority Queues

MINIMIZATION

1. Initialize : Q = { start }
2. Remove from Q to node n with lowest cost. If Q is empty → fail
3. If n is goal → stop with solution
4. Generate successors of node n using the Decomposition Rule. Let them be $n_1, n_2, \ldots, n_k$
5. If $n_i \notin Q$ insert n_i in Q
6. If $n_i \in Q$ update n_i in Q
7. Goto step 2.

Q ← Priority Queue
1. Insert
2. Delete Min
3. Update



$Q = \{ n_1(2) \} \rightarrow \{ n_2(3), n_3(4) \}$

$\rightarrow \{ n_3(4), n_4(8), n_5(4) \}$

$\rightarrow \{ n_4(8), \cancel{n_5(4)}, n_6(7), n_7(5), n_8(6) \} \rightarrow \{ \ldots n_{10}(4) \ldots \}$

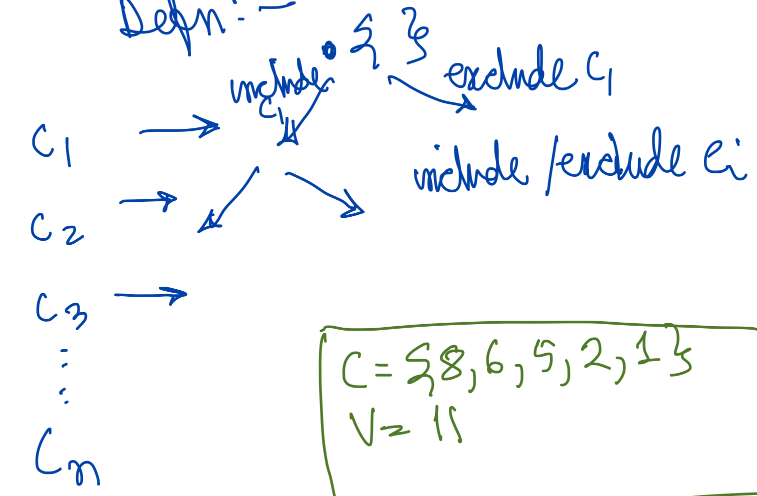
SOLUTION

Coin Selection Problem

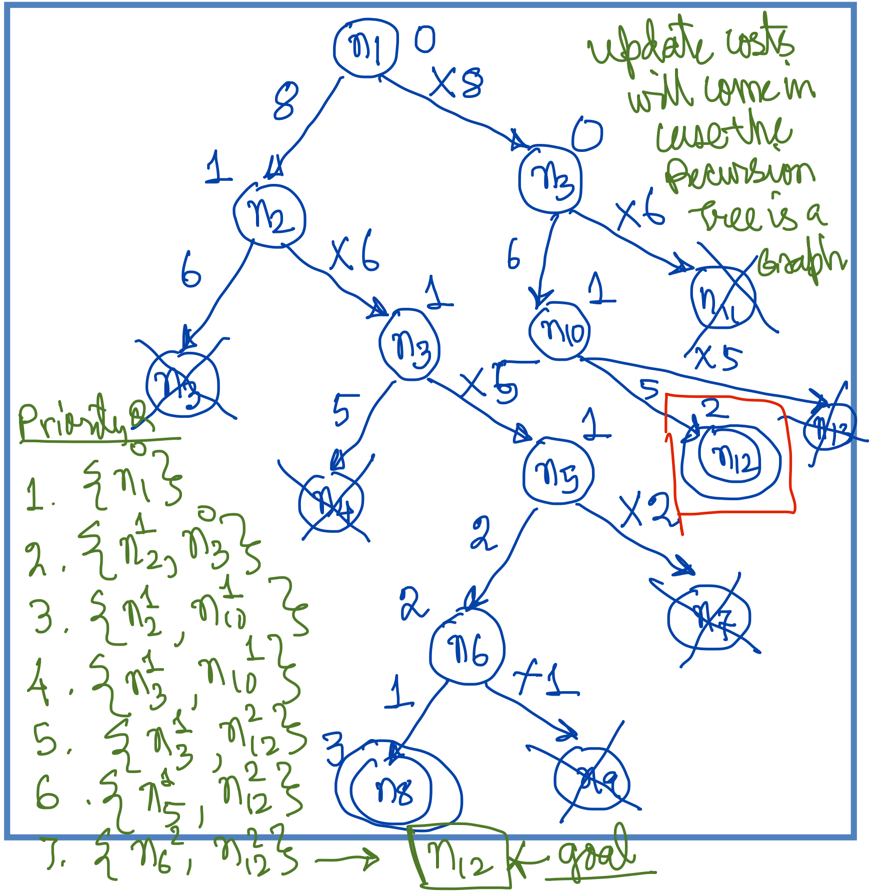
Coins = $\{c_1, c_2, \dots, c_n\}$, V

↳ Find the minimum no. of coins to get the exact value of V

→ Inclusion - Exclusion Recursive Defn:-



C = {8, 6, 5, 2, 1}
V = 11



Related Data Structures

- Binary Search Trees (BSTs)
  - ↳ Balanced BSTs
    - ↳ AVL Trees
    - B-Trees, etc
- Advanced Heaps
  - ↳ Binomial Heaps
  - Fibonacci Heaps
  - etc

- Weighted BSTs
  - ↳ frequency of access (DYNAMIC PROGRAMMING METHOD)
- Data structures using Advanced operations
  - Union, Intersection, Set Difference, etc.

Summary

- insert
- remove-min/max
- update/delete

↳ Priority Queue operations

↳ Implemented by a Heap Data Structure

In case we have

Find ← BST

Union/Intersection, etc ← ??

Operations could be

- provided before the algo. (static)
- online

Large Class of optimization problems → Greedy, B&B

Thank you

Any Questions?