PROOF TECHNIQUES

• construction, induction, contradiction

Order of Complexities:

O (1), O (log n), O (n), O (n log n), O (n²)

asymptotic performance -- Running time -- Memory / storage requirements – Bandwidth / power requirements / logic gates / etc.

Text

Description automatically generated

Text, letter

Description automatically generated

Recurrence Relation For algorithms:

Selection: T(N) = T(N-1) + N - 1

Bubble: T(N) = T(N-1) + N-1

Insertion: T(N) = T(N-1) + N

Heap: T(N) = T(N-1) + log(N)

Quick W: T(N) = T(N-1) + N

Quick B: T(N) = 2\*T(N/2) + N

Merge: T(N) = 2\*T(N/2) + N

Divide & Conquer(D&C):

- Divide-> Conquer-> Combine

- Both paradigms (D& C and DP) divide the given problem into subproblems and solve subproblems.

- D&C should be used when the same subproblems are not evaluated many times.

Dynamic Programming (DP):

- Compute the value of an optimal solution in a bottom-up fashion

- DP uses the result of the subproblems to find the optimum solution of the main problem, sub-problems are interdependent

- *Memoization*: speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

Merge vs Quick:

- Merge: divides into subsections, conquers sections, combines back together

- Quick: Divides into subarrays, conquers them recursively, no need to combine

Randomized Quicksort:  
 No input can elicit worst case behavior  
– Worst case occurs only if we get “unlucky”   
• Worst case becomes less likely  
– Randomization does NOT eliminate the worst-case but makes it less likely!

Masters Method

A picture containing text

Description automatically generated

where, a ≥ 1, b > 1, and f(n) > 0

ϵ is a constant

Graphical user interface, text, application

Description automatically generated

Fibonacci DP to the rescue  
• Computing the nth Fibonacci number: F(n) = F(n-1) + F(n-2)  
• bottom-up iteration

Binary Tree – Binary Search Tree   
– SEARCH, MINIMUM, MAXIMUM, PREDECESSOR,   
SUCCESSOR, INSERT, DELETE  
• The expected height of the tree is lgn  
• The tree is a linear chain of n nodes

Algs/complexities: best/avg/worst

Selection: Ω (n^2) θ (n^2) O (n^2)

Bubble: Ω (n) θ (n^2) O (n^2)

Insertion: Ω (n) θ (n^2) O (n^2)

Heap: Ω (n log(n)) θ (n log(n)) O (n log(n))

Quick: Ω (n log(n)) θ (n log(n)) O (n^2)

Merge: Ω (n log(n)) θ (n log(n)) O (n log(n))

KMP: O (n)

Naïve: O ((n-m+1) m)

Huffman: O (n lg n)

Knapsack: Ω (n) θ (n log(n))

Heapsort

• Height of a node = the number of edges on the longest simple path from the node down to a leaf  
• Level of a node = the length of a path from the root to the node  
• Height of tree = height of root node  
• Full – node is leaf or of degree 2  
• Complete – interior nodes degree 2, and leaves @ same level

– There are at most 2^i nodes at level i of a binary tree

– binary tree with height d has at most 2^(d + i)-1 nodes

– A binary tree with n nodes has height at least [lgn]

A picture containing text, clock, watch, gauge

Description automatically generated

• A heap is a nearly complete binary tree with the following two properties:  
– Structural property: all levels are full, except   
possibly the last one, which is filled from left to right  
– Order (heap) property: for any node x  
Parent(x) ≥ x

Array Representation of Heaps

• A heap stored as an array A  
– Heapsize [A] ≤ length[A]  
• The elements in the subarray A  
[(⎣n/2⎦+1) ... n] are leaves

Heap Types  
• Max-heaps (largest element at root), have the   
max-heap property:   
– for all nodes i, excluding the root:   
A[PARENT(i)] ≥ A[i]  
• Min-heaps (smallest element at root), have the   
min-heap property:  
– for all nodes i, excluding the root:   
A[PARENT(i)] ≤ A[i]

Heap Sort

A picture containing table

Description automatically generated

Traversing a Binary Search Tree

• Inorder tree walk:  
– Root is printed between the values of its left and right subtrees: left, root, right  
– Keys are printed in sorted order  
• Preorder tree walk:  
– root printed first: root, left, right  
• Postorder tree walk:  
– root printed last: left, right, root

– Binary Operations: O(h)

Knuth-Morris-Pratt (KNP) String Searching

- whenever we detect a mismatch

- we already know some of the characters in the text of the next window.

- take advantage of this information to avoid matching the characters that will match anyway

- backtracking never occurs

Naïve String Matching

Finds all valid shifts using a loop that checks the condition P [1 ... m] = T [s + 1 ... s + m] for each of the n – m + 1 possible values of s.

Graphs: a finite set of dots called vertices (or *nodes*) connected by links called edges (or *arcs*).

- directed: strongly connected, every two vertices are reachable from each other

- undirected: connected, every pair of vertices is connected by a path

- Tree: connected acyclic undirected graph

Minimum Spanning Tree (MST): connected, undirected graph, has all the vertices

-spanning tree with minimum sum of weights

- spanning forest: a graph in not connected, a spanning tree for each connected component of the graph

Dijkstra:

The code calculates the shortest distance but doesn’t calculate the path information. We can create a parent array, update the parent array when distance is updated

Bellman-Ford:

calculates shortest paths in a bottom-up manner. It first calculates the shortest distances which have at-most one edge in the path. Then, it calculates the shortest paths with at-most 2 edges, and so on. After the i-th iteration of the outer loop, the shortest paths with at most i edges are calculated

Compare & Contrast

Dijkstra – each edge is only reflected once, cannot handle negative numbers. Complexity is better

Bellman-Ford – negative numbers are aloud.

Intractable:

Problems not in P are *intractable* or unsolvable

– Can be solved in reasonable time only for small inputs Or, cannot be solved at all

NP Theory: consists of problems that could be

solved by NP algorithms

– i.e., verifiable in polynomial time

• If we were given a “certificate” of a solution, we

could verify that the certificate is correct in time

polynomial to the size of the input

Class P consists of (decision) problems that are

solvable in polynomial time

• Examples of polynomial time:

– O(n2), O(n3), O(1), O(n lg n)

• Examples of non-polynomial time:

– O(2n), O(nn), O(n!)

Polynomial Reductions:

• Given two problems A, B, we say that A is

polynomially reducible to B (A p B) if:

1. There exists a function f that converts the input of A

to inputs of B in polynomial time

2. A(i) = YES  B(f(i)) = YES

NP-Completeness:

• A problem B is NP-complete if:

(1) B ∈ NP

(2) A ≤ p B for all A ∈ NP

• If B satisfies only property (2) B is NP-hard

• No polynomial time algorithm has been discovered for an NP-Complete problem

• That said, no proof (at least yet,) that no polynomial time algorithm exists for any NP-Complete problem

Is P=NP? Theorem: If any NP-Complete problem

can be solved in polynomial time  then P = NP.

Greedy Approach:

- Like DP, but simple, used for optimization problems

- Idea: When we have a choice to make, make the one that looks best right now to get optimal solution.

- Make a locally optimal choice in hope of getting a globally optimal solution

- Greedy algorithms don’t always yield an optimal solution

Knapsack:

- Memorization is another way to deal with overlapping subproblems in dynamic programming

- implement the algorithm recursively:

» If we encounter a new subproblem, we compute and store the solution.

» If we encounter a subproblem we have seen, we look up the answer

- Useful when the algorithm is easier to implement recursively, especially if we do not need solutions to all subproblems.

Huffman: achieves data compression by finding the best variable length binary encoding scheme for the symbols that occur in the file to be compressed

Text

Description automatically generated

Text, application

Description automatically generated

Alg: TREE-DELETE(T<z) Alg: TREE-INSERT(T,z)

A picture containing diagram

Description automatically generatedText

Description automatically generated

|  |  |
| --- | --- |
| displaymath428 | L’Hopital Rule  Lim f(n) if -> ∞  n->∞ g(n) ∞  then take the derivative of f and g independently  n! is n(n+1)/2  also equal to |

A picture containing diagram

Description automatically generated

|  |  |
| --- | --- |
| ***MergeSort* (*A*, *p*, *r*)**  **//** sort *A*[*p...r*] by d & c  **if***p* < *r*  **then** *q* ← ⎣(*p* + *r*)/2⎦  *MergeSort* (*A*, *p*, *q*)  *MergeSort* (*A*, *q*+1, *r*)  *Merge* (*A*, *p*, *q*, *r*) // merges *A*[*p...q*] with *A*[*q+1..r*] | **Merge (*A*, *p*, *q*, *r*)**  1 *n*1 ← *q* – *p* + 1  2 *n*2 ← *r* – *q*  **for** *i* ← 1 **to** *n*1  **do** *L*[*i*] ← *A*[*p* + *i* – 1]  **for** *j* ← 1 **to** *n*2  **do** *R*[*j*] ← *A*[*q* + *j*]  *L*[*n1*+1] ← ∞  *R*[*n2*+1] ← ∞  *i* ← 1  *j* ← 1  **for** *k* ←*p* **to** *r*  **do if** *L*[*i*] ≤ *R*[*j*]  **then** *A*[*k*] ← *L*[*i*]  *i* ← *i* + 1  **else** *A*[*k*] ← *R*[*j*]  *j* ← *j* + 1 |

Alg.: QUICKSORT (A, p, r)  
if p < r  
 then q ← PARTITION (A, p, r)  
 QUICKSORT (A, p, q)  
 QUICKSORT (A, q+1, r)

Alg. PARTITION (A, p, r)  
1. x ← A[p]  
2. i ← p – 1  
3. j ← r + 1  
4. while TRUE  
5. do repeat j ← j – 1  
6. until A[j] ≤ x  
7. do repeat i ← i + 1  
8. until A[i] ≥ x  
9. if i < j  
10. then exchange A[i] ↔ A[j]  
11. else return j

Alg.: RANDOMIZED-PARTITION (A, p, r)  
i ← RANDOM(p, r)  
exchange A[p] ↔ A[i]  
return PARTITION(A, p, r)

Alg . : RANDOMIZED-QUICKSORT(A, p, r)  
if p < r  
then q ← RANDOMIZED-PARTITION(A, p, r)  
RANDOMIZED-QUICKSORT(A, p, q)  
RANDOMIZED-QUICKSORT(A, q + 1, r)

Text

Description automatically generatedText

Description automatically generated

Alg: Generic MST

1. A ← 

2. while A is not a spanning tree

3. do find an edge (u, v) that is safe for A

4. A ← A  {(u, v)}

5. return A

Alg: PRIM (V, E, w, r)

1. Q ← ∅

2. for each u ∈ V

3. do key[u] ← ∞

4. π[u] ← NIL

5. INSERT(Q, u)

6. DECREASE-KEY(Q, r, 0) ► key[r] ← 0

7. while Q != ∅

8. do u ← EXTRACT-MIN(Q)

9. for each v ∈ Adj[u]

10. do if v  Q and w(u, v) < key[v]

11. then π[v] ← u

12. DECREASE-KEY(Q, v, w(u, v))

Alg: NAÏVE-STRING-MATCHER (T, P)

1. n ← *length*[*T*]

2. m ← *length*[*P*]

3. for s ← 0 to *n – m*

4. do if *P*[1…m] = *T*[s + 1 .. s + m]

5. then print “Pattern occurs with shift” s

Alg: FIB-DYN-PROG

1. fib(n):

2. if n == 0:

3. return 0

4. if n == 1:

5. return 1

6. if memo[n] != -1:

7. return memo[n]

8. memo[n] = fib(n-1) + fib(n-2)

9. return memo[n]

Alg: Memoized DP Rod cutting Top-Down:

MemoizedCutRod (p, n)

r: array(0..n) := (0 => 0, others =>MinInt)

return MemoizedCutRodAux (p, n, r)

MemoizedCutRodAux (p, n, r)

if r(n) = MinInt then -- calculate a new solution?

q: int := MinInt

for i in 1 .. n loop

q := max(q, p(i) + MemoizedCutRodAux(p, n-i, r))

end loop

end if

r(n) := q

end if

return r(n)

Algo: Huffman

Procedure Huffman(C): // C is the set of n characters and related information

n = C.size

Q = priority\_queue()

for i = 1 to n

n = node(C[i])

Q.push(n)

end for

while Q.size() is not equal to 1

Z = new node()

Z.left = x = Q.pop

Z.right = y = Q.pop

Z.frequency = x.frequency + y.frequency

Q.push(Z)

end while

Return Q

Alg.: Fractional-Knapsack (W, v[n], w[n])

1. While w > 0 and as long as there are items remaining

2. pick item with maximum vi/wi

3. xi  min (1, w/wi)

4. remove item i from list

5. w  w – xiw