GATEFREAKS

GATE/NET/PSU

COMPUTER SCIENCE

Design and analysis of Algorithm : Shortnotes

Sachin Michu
Alumnus IIT DELHI

Neha Michu
Alumna IIT DELHI

August 6, 2018

Key concepts on Design and Analysis of Algorithms

1. Useful Formulae:
   (a) $1+2+3+...+n = \frac{n(n+1)}{2}$
   (b) $1^2+2^2+3^2+...+n^2 = \frac{n(n+1)(2n+1)}{6}$
   (c) $1^3+2^3+3^3+...+n^3 = \frac{(n(n+1))^2}{4}$
   (d) $a+(a+d)+(a+2d)...a+(n-1)d = \frac{n(2a+(n-1)d)}{2}$
   (e) $a+ar+ar^2.....ar^{n-1} = \frac{a(r^{n-1})}{(r-1)}$
   (f) $1+2+2^2+2^3...+2^n = 2^{n+1}-1$
   (g) $1+x+x^2+..... = \frac{1}{1-x}$
   (h) $1+x^n+x^{2n}+x^{3n}.... = \frac{1}{1-x^n}$
   (i) $x+2x^2+3x^3+4x^4..... = \frac{x}{(1-x)^2}$
   (j) $e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+.....$
   (k) $\sum_{k=0}^{n} {}^nC_k = 2^n$
   (l) $\log_2 2+\log_2 3+\log_2 4+.....+\log_2 n = \log_2 n! \approx \log_2 n^n = n\log_2 n$
   (m) $\log AB = \log A+\log B$
   (n) $X^{\log_2 Y} = Y^{\log_2 X}$
   (o) $\log A/B = \log A-\log B$
   (p) $\log X^Y = Y\log X$
   (q) $\log_x a = \frac{\log_2 a}{\log_2 x}$
   (r) $\lg *n$:Number of times applying log repeatedly to get value 1
   (s) $\lg *(\lg n) = \lg *n-1$
2. $O(\frac{1}{n}) < O(1) < O(\log_2\log_2 n) < O(\sqrt{\log_2 n}) < O(\log_2 n) < \sqrt{n} < O(n) < O(n\log_2 n) < O(n^2) < O(n^2\log_2 n) < O(n^3) < O(n^{\log_2 n}) < O(2^n) < O(n!) < O(n^n)$
3. Asymptotic notations
   (a) Asymptotic notations (Ω, θ, O follow symmetric , reflexive and transitive property.
   (b) If $f(n) = O(g(n))\&\&g(n) = O(h(n))$ then $f(n) = O(h(n))$,Likewise transitive property hold for Ω, θ also.
   (c) If $f(n) = O(g(n))\&\&d(n) = O(h(n))$ then $f(n)+d(n) = \max(g(n), h(n))$
   (d) If $f(n) = O(g(n))\&\&d(n) = O(h(n))$ then $f(n)*d(n) = O(g(n)*h(n))$
   (e) $\{n, 100n+\log_2 n^5+\frac{n}{10}, \frac{n^2}{n\log_2 n}+\frac{n}{200}+50, 1000\log_2 n+(\log_2 n)^{50}+20n\} = \theta(n) = O(n) = O(n^2) = \Omega(1) = \Omega(\log_2 n) = \Omega(n)$

(f) $\{n^2, 1000n + n\log_2 n^5 + \frac{1000n}{5}, \frac{100n^2}{5} + \frac{n^3}{n+50} + 150, n\log_2 n + (\log_2 n)^{50} + 20n\} = O(n^2) = \theta(n^2) = O(n^3) = O(n^4) = \Omega(1) = \Omega(n\log_2 n) = \Omega(n^2)$

4. Applications of Divide and conquer

   (a) Finding maximum and minimum
      i. Recurrence Relation:$T(n) = 2T(\frac{n}{2}) + 2$
      ii. Time complexity :$1.5n - 2 = O(n)$

   (b) Power of an element
      i. Recurrence Relation:$T(n) = T(\frac{n}{2}) + 1$
      ii. Time complexity :$O(\log_2 n)$

   (c) Binary search
      i. Used with Sorted array.
      ii. Recurrence Relation:$T(n) = T(\frac{n}{2}) + 1$
      iii. Time complexity WC:$O(\log_2 n)$, AC:$O(\log_2 n)$,BC:O(1)

   (d) Merge sort
      i. out-place Sorting algorithm.
      ii. Divide takes O(1) time , combine takes O(n) time at every step.
      iii. Merging 2-sorted array of size m and k takes $\theta(m + k)$ time
      iv. Recurrence Relation:$T(n) = 2T(\frac{n}{2}) + n$
      v. Time complexity WC:$O(n\log_2 n)$, AC:$O(n\log_2 n)$,BC:$O(n\log_2 n)$
      vi. Not suitable for smaller size array

   (e) Quick sort
      i. In-place , unstable sorting algorithm
      ii. Not suitable for the sorted or partially sorted array(worst case arises)
      iii. Partition procedure take O(n) time , which partition array in 2 parts.
      iv. Best case of quick sort occur if array is divided in two part k and m, such that $k \approx m$
      v. Recurrence Relation for the best case:$T(n) = 2T(\frac{n}{2}) + n$
      vi. Time complexity in best case (BC): $O(n\log_2 n)$
      vii. Recurrence Relation for the worst case:$T(n) = T(n-1) + T(1) + n$
      viii. Time complexity in worst case (WC): $O(n^2)$
      ix. Recurrence Relation for the Average case:$T(n) = T(\alpha n) + T((1-\alpha)n) + n$, $\alpha * n \neq k, k$ is constant
      x. Time complexity in average case (AC): $\Omega(n\log_{1/\alpha} n) = O(n\log_{1/(1-\alpha)} n) = \theta(n\log_2 n)\{\because \alpha > (1-\alpha)\}$

   (f) Stressan's Matrix multiplication
      i. Recurrence Relation : $T(n) = 7T\frac{n}{2} + 18(\frac{n}{2})^2$
      ii. Time Complexity :$O(n^{2.81})$

5. Insertion Sort

   (a) Time complexity , BC:$O(n)$, AC:$O(n^2)$, WC:$O(n^2)$

   (b) Suitable when array size is small or array is almost sorted.

   (c) Best case arises when array is sorted in non-decreasing order and worst case arises when array is sorted in reverse order(non-increasing) order

   (d) Number of comparisons in average case:$O(n^2)$, Number of swaps:$O(n^2)$

6. Selection Sort

   (a) Time complexity , BC:$O(n^2)$, AC:$O(n^2)$, WC:$O(n^2)$

   (b) To sort an array of size n, it requires $(n-1)$ passes and in every pass exactly one swap and at-most n comparisons.

   (c) Selection sort requires minimum number of swaps as compared to other comparison based sorting algorithms

7. Heap Sort
   (a) Time complexity , BC:$O(n \log_2 n)$, AC:$O(n \log_2 n)$, WC:$O(n \log_2 n)$
   (b) Almost complete binary tree(ACBT) and complete binary tree(CBT) are used for implementation of heap sort.
      i. Number of nodes at level i in CBT=2^{i-1}, root at level 1.
      ii. Number of nodes in CBT having level i =$2^i - 1$
      iii. Number of levels in CBT having n nodes =$\log_2(n+1)$
   (c) There are two types of heap tree : Max heap and Min heap.
   (d) Build min heap , build max heap takes $O(n)$ time.
   (e) Insertion, deletion takes $O(\log_2 n)$ time in min heap and max heap, more precisely $O(h)$, h is the height of the heap tree.
   (f) Time complexity of heap sort, BC:$O(n \log_2 n)$, AC:$O(n \log_2 n)$, WC:$O(n \log_2 n)$
   (g) Deleting n^{th} largest element from max-heap takes $O(n)$ time, likewise Min-heap takes $O(n)$ time for deleting n^{th} smallest element.
8. Radix Sort
   (a) Time complexity , $\theta((n+k)d)$, n :Count of integers, k:base, d:number of digits
9. Greedy Approach
   (a) Job sequencing with deadline
      i. sorting job decreasing order according to of deadline = $O(n \log_2 n)$
      ii. Finding slot for each job in array of size n = $O(n^2)$
      iii. Time complexity = $O(n \log_2 n) + O(n^2) = O(n^2)$
   (b) Fractional knapsack : Time complexity $O(n \log n)$
   (c) Huffman encoding
      i. generally used for data compression
      ii. Time complexity is $O(n \log n)$
   (d) Minimum cost spanning Tree
      i. Prims algorithm
         A. Time complexity using, binary heap:$O(V \log V + E \log V)$, fibonacci heap:$O(E + V \log V)$, Binomial heap:$O(V + E)$
         B. Best data structure to implement Prims algorithm is fibonacci heap, So consider time complexity $O(E + V \log V)$ (by default if implementation data structure is not specified)
         C. When graph is dense, time complexity is $O(V^2)$(fibonacci heap)
         D. For dense graph ,Prims algorithm is better than Kruskal algorithm
      ii. Kruskal's algorithm
         A. Time complexity is $O(ElogE + V)$
         B. When graph is dense, time complexity is $O(V^2 \log V)$
         C. When graph is sparse then its performance is better than Prims algorithm
   (e) Optimal Merge pattern, time complexity is $O(n \log n)$
   (f) Single source shortest path
      i. Dijkstra's algorithm
         A. Find single source shortest path for weighted graph
         B. Time complexity using, Binary min heap:$O((V+E) \log V)$, Fibonacci heap: $O(E+V \log V)$, Binomial min heap:$O(V + E)$.
         C. It can not handle negative edge weight efficiently.
      ii. Bellman Ford algorithm , works like dijkstra algorithm but it handle negative cycle efficiently , runs in $O(VE)$ time.
10. Dynamic Programming

(a) Fibonacci series, time complexity is $O(n)$

(b) Longest common subsequence

i. Recurrence Relation $T(n,m) = \begin{cases} T(n-1,m-1)+O(1), & \text{if } (X[m-1]=Y[n-1]) \\ T(n-1,m)+T(n,m-1)+O(1) & \text{otherwise} \end{cases}$

ii. Time Complexity $T(m,n) = 2^{\max(m,n)}$ with-out using dynamic programming and $O(n*m)$ using dynamic programming

(c) Matrix chain multiplication

i. Recurrence Relation $T(n,m) = \begin{cases} 0, & \text{if } (n=m) \\ min(T(n,k)+T(k+1,j)+p_{i-1}p_kp_j) & \text{if } i<j \end{cases}$

ii. Time Complexity $T(n) = n^3$, n denotes the sequence matrices of length n

(d) Multi-stage Graph

(e) Traveling Salesman problem,time complexity $(n^2 * 2^n)$

(f) All pair shortest Path

i. Floyd-Warshall algorithm is used

ii. Time complexity is $O(V^3)$

iii. Using Warshall algorithm, Transitive closure of a graph can be calculated in $O(V^3)$ time.

(g) 0/1 knapsack, time complexity be $O(nW)$ where n is the number of items and W is the restriction on weight.

(h) Sum of the subsets problem,time complexity $O(nS)$, n is the sequence of integers in array and S is the Sum.

(i) Optimal Binary search tree , runs in $O(n^3)$ time.

11. Breadth First Search(BFS)

(a) Time complexity : $O(|V|+|E|)$.

(b) Uses Queue data structure

(c) Result in level order traversal for a tree if starting node is root.

(d) Used to find if there is a path between two vertices.

(e) Used to find single source shortest path and spanning tree.

(f) Used for Cycle detection in undirected graph.

(g) Used for finding all nodes within one connected component.

(h) FordFulkerson algorithm which runs in $O(VE^2)$, uses Breadth First search to find the maximum flow.

(i) Crawlers build index using BFS in Search Engines.

(j) Used to check if graph is bipartite.

12. Depth First Search(DFS)

(a) Time complexity : $O(|V|+|E|)$.

(b) Uses Stack data structure.

(c) For an unweighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.

(d) Used for Cycle detection in undirected graph.

(e) Used to find if there is a path between two vertices.

(f) Used to find Topological Sorting.

(g) Used to check if graph is bipartite.

(h) Finding Strongly Connected Components of a directed graph.

(i) Solving puzzles with only one solution, such as mazes.

(j) Finding bridge in the graph.

(k) Used for Garbage Collection.

13. Hashing

(a) Used for efficient searching

(b) Searching ,insertion , deletion takes O(1) time in best case and $O(n)$ time in worst case.

(c) A good hash function must be chosen to avoid collision.Commonly used hash function is modulo function , for example $h(k) = k \bmod m$, k is the key value and m is table size.

i. m should not be power of 2,because if $m = 2^p$, then $h(k)$ is just p lowest order bits of k.

ii. m should be a prime number but it should not be very close to the power of 2.

(d) Let m be the size of hash table and n be number of keys so load factor (α)is defined by n/m

(e) Collision resolution techniques

i. Chaining(open hashing)

A. Keys are not stored in the hash table , only pointer to the linked list is stored in hash table.

B. Insertion takes O(1) time

C. In successful search , on average we have to traverse the half length of linked list so time complexity is O(1+α/2) which is equivalent to O(1+α).

D. .In case of unsuccessful search , we have to traverse the complete linked list at a given slot so total time is O(1+n/m) or O(1+α)

E. Time for deletion is same as of insertion.

ii. Open addressing(close hashing)

A. Linear probing

- All keys are stored in the hash table.
- Hash function: $h(k) = (k + i) \bmod m, i$ is probe number.
- After every deletion a Marker sign(X) (which signify the key value has been deleted and slot is empty) is inserted.
- Main problem is primary clustering

B. Quadratic probing

- All keys are stored in the hash table.
- Hash function: $h(k) = (k + i^2) \bmod m, i$ is probe number.
- Main problem is Secondary clustering

C. Double hashing

- All keys are stored in the hash table.
- Hash function: $h(k) = (k + i * h'(k)) \bmod m, i$ is probe number, $h'(k)$ is another hash function.

Gatefreaks believe in providing quality. Qualifying any exam is not difficult if you have proper guidance and quality material. Our focus is to provide you the best error free content. We provide complete material including short notes, detailed notes, previous year solved papers and practice tests as a part of our intensive classroom program.For further details visit www.gatefreaks.com