Homework-7 ITCS-6114

Nisarg Antony - 800989573

Survey of 20 most popular algorithms.

(1) Merge Sort

Merge Sort is a divide and conquer algorithm that divides the input array in two halves, calls itself for the two halves and then merges the two sorted halves.

The entire process goes as follows:

1. Divide by finding the number q of the position midway between p and r. Do this step the same way we found the midpoint in binary search

2. Conquer by recursively sorting,

the subarrays is each of the two ~~sup~~ subproblems created by the divide step. That is, recursively sort the subarray array[p..q] and recursively sort the subarray array[q+1..r].

3 Combine by merging the two sorted subarrays back into the single sorted subarray array[p..r].

(2) Quick Sort:

Like merge sort, quicksort uses divide and conquer and so it's a recursive algorithm.

Quicksort is different from merge sort in some ways. All of the work happens in the divide step as opposed to the combine step in merge

soot. It is in place and its worst case running times are as bad as selection soots and insertion soots: $\Theta(n^2)$

But it's average case running time is as good as merge sorts: $\Theta(n \lg n)$

In practice, quicksort outperforms merge soot and significantly outperforms selection soot and insertion soot.

The idea behind quicksort is

1. Divide by choosing a pivot element in the subarray $A[p..r]$. Rearrange so that all elements less than or equal to the pivot lie on its left and all the elements greater than the pivot lie to its right.

2. Conquer by recursively sorting the subarrays array [p..q-1] and array [q+1..r].

3. Combine. This step doesn't do anything in terms of actual work.

(3) Dijkstra's Algorithm

Dijkstra's algorithm is used for finding the shortest path from a starting node to a target node in a weighted graph.

The algorithm creates a tree of shortest paths from the source to all other nodes in the graph.

The algorithm works as follows:

1. While the queue of all nodes in the graph, Q is not empty, pop the node v that is not already in S (S is an empty set which contains the nodes the algorithm has already visited) from Q with the smallest distance (v)

2. Add node v to S, to indicate that v has been visited.

3. Update dist values of adjacent nodes of the current node as follows:

- if dist(v) + weight(u,v) < dist(v) there is a new minimal distance found for u so update dist(u)

- otherwise, no updates are made to dist(v)

(4) A* Pathfinding Algorithm.

A* ~~is~~ a algorithm is widely used in pathfinding and graph traversal. The algorithm efficiently plots a walkable path between multiple nodes or points on the graph.

The A* algorithm introduces a heuristic into a regular graph-searching algorithm, essentially planning ahead at each step so a more optimal decision is made.

Like Dijkstra, A* works by making a lowest-cost path from the start node to the target node.

It uses a function f(n) that gives an estimate

of the total cost of a path using that node.

A* expands paths that are already less expensive by using this function

$$f(n) = g(n) + h(n)$$

where

$f(n)$ = total estimated cost of path through node n

$g(n)$ = cost so far to reach node n

$h(n)$ = estimated cost from n to goal

(5) Binary Search Algorithm

Binary search is an efficient algorithm for finding an

item from an ordered list of items.

It works by repeatedly dividing in half the portion of the list that could contain the algorithm

Procedure
Given an array A of ~~element~~ n sorted element and target value T.

1. Set L to 0 and R to n-1.
2. If L > R, terminate the search
3. Set the position of the middle element to the floor of (L+R)/2
4. If Am < T set L -> m-1 & goto step2
5. If Am > T set R -> m-1 goto step3

6. Now Am=.T, the search is done. return m.

(6) Breadth First Search (BFS) and Depth First Search (DFS)

BFS and DFS algorithms are used for traversal of trees through the construction of spanning trees.

1. BFS:

BFS can be thought of as being like Dijkstra's algorithm for shortest paths, but with every edge having the same length.

Algorithm

unmark all vertices
choose some starting vertex v
mark v
list L=v

tree T = ∅.
while nonempty
choose ~~for~~ some vertex v
from front of list
visit v
for each unmarked neighbor w
mark w
add it to end of list
add ~~dr~~ edge vw to T

2. DFS

DFS is another way of traversing graphs, which is closely related to preorder traversal of a tree.

```
dfs(vertex v)
{
    visit(v);
    for each neighbor w of v
    if w is unvisited
    {
        dfs(w);
        add edge vw to tree T
    }
}
```

7. Knapsack Problem

The knapsack problem is a problem in combinatorial optimization.

Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that their total weight is less than or equal to a given limit and the total value is as large as possible.

Generally these are three variants of this problem

1. The 0-1 knapsack problem which restricts the number of copies of each kind of item to zero or one.

2. The bounded knapsack problem which removes the restricto that there is only one of each item but restricts the number x_i of copies of each kind of item to a max. non-negative integer value.

3. The unbounded knapsack problem which places no upper bound on the no of copies of each kind of item.

(8) Knuth-Morris-Pratt Algorithm

The KMP algorithm is a string searching algorithm that searches for occurrences of a word W within a main text string S by employing the observation that when a mismatch occurs the word itself embodies ... information to

determine where the next match could begin, thus bypassing re-examination of previously matched characters.

The Algorithm

KNUTH-MORRIS-PRATT(T, P)

Input: Strings T[0..n] and P[0..m]

Output: Starting index of substring of T matching P.

f ← compute failure function of pattern P.

i ← 0

j ← 0

while i < length[T] do

if $j \leq m-1$ then

return $i - m + 1$

$i \leftarrow i + 1$

$j \leftarrow j + 1$

else if $j > 0$

$j \leftarrow f(j-1)$

else

$i \leftarrow i + 1$

Worst case time complexity
$O(m+n)$

Space complexity : $O(m)$

(9) Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm in

graph theory that finds a minimum spanning tree for a connected weighted graph.

It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

Algorithm

```
MST-KRUSKAL(G, w)
    A ← ∅
    for each vertex v not in V[G]
    do make-set(v)
    sort the edges of E into
    non decreasing order by
    weight w.

    for each edge (u, v) not in
    E
    do if FIND-SET(u) ≠ FIND-SET
```

then A ← A not in {(u,v)}
UNION (u,v)

return A

(10) Prim's Algorithm

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph.

It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

MST- PRIM (G, w, r)

for each u not in V[G]
do key[u] ← ∞
π[u] ← NIL
key[r] ← 0
Q ← V[G]
while Q ≠ ∅

do u ← EXTRACT MIN(Q)

for each v not in Adj[u]

do if v not in Q and
w(u,v) < key[v]

then π[v] ← u

key[v] ← w[u,v]

11 RSA Algorithm:

RSA algorithm is asymetric cryptography algorithm

The idea of RSA is based on the fact that it is difficult to factorize a large integer.

The public key consists of two numbers where one number is multiplication of two large prime numbers.

And private key is also derived from the same two prime numbers.

RSA keys can be typically 1024 or 2048 bits long

(12) Rabin Karp Algorithm.

Rabin-Karp string searching algorithm calculates a numerical value for the pattern p and for each m-character substring

of text t. Then it compares the numerical values instead of comparing the actual symbols. Otherwise it shifts to ~~to~~ the next substring of t to compare with p.

Algorithm.

Compute hp (for pattern p)

Compute hi (for ~~last~~ first substring of t with m length)

For i=1 to n-m

If hp = hi

Match t[i, ..., i+m] with p
if matched return i

Else
hi = (d (ht - t[i+1] · d^(m-1)) + t[m+i+1]) mod q

End.

(13) Longest Common Subsequence.

Given two sequences $X = \langle x_1, x_2, \cdots x_m \rangle$ and $Z = \langle z_1, z_2 \cdots z_k \rangle$ we say that Z is a subsequence of X if there is a strictly increasing sequence of k indices $\langle i_1, i_2, \cdots i_k \rangle$ $(1 \le i_1 < i_2 < \cdots < i_k \le n)$ such that $Z = \langle x_{i_1}, x_{i_2}, \cdots x_{i_k} \rangle$

Given two strings X and Y, the longest common subsequence of X and Y is a longest sequence Z which is both a subsequence of X and Y.

(14) Bellman-Ford Algorithm

The Bellman-Ford algorithm is a graph search algorithm that finds the shortest path between a given

source vertex and all other vertices in the graph.

This algorithm can be used on both weighted and unweighted graphs.

Pseudo-Code

```
for v in V:
    v.distance = ∞
    v.p = Θ

    Source.distance = 0

    for i from 1 to |V| - 1:

        for (u,v) in E:

            relax(u,v).

relax(u,v):
If v.distance > u.distance + weight(u,v):
    v.distance = u.distance + weight(u,v)
    v.p = u
```

(15) K-means clustering algorithm

The K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem.

The procedure follows a simple and easy way to classify a given data set through a certain number of clusters

The main idea is to define k-centroids, one for each cluster.

The algorithm is composed of the following steps

1. Place K points into the space represented by the objects that are being clustered.

2. Assign each object to the group that has the closest centroid.

3. When all objects have been assigned, recalculate the positions of the K-centroids

4. Repeat Steps 2 and 3 until all the centroids no longer move.

(16) Euclids Algorithm

The Euclidean algorithm calculates the greatest common divisor (GCD) of the two natural numbers a and b.

The GCD is the largest number that divides both a and b without leaving a remainder

Algorithm.

```
GCD(a, b)
    while b ≠ 0
        t = b
        b = a mod b
        a = t
    return a
```

(17) Bubble Sort Algorithm.

Bubble sort is a simple and well known sorting algorithm. It is used in practice once in a blue moon and. its main application is to make an introduction to the sorting algorithm.

Algorithm.

1. Compare each pair of adjacent elements from the beginning of the array

and if they are in reversed order, swap them.

2. If at least one swap has been done, repeat step 1.

(18) Radix Sort

Radix sort is an integer sorting algorithm that sorts data with integer keys by grouping the keys by individual digits that share the same significant position and value.

Radix sort takes in a list of n integers which are in base b and so each number has at most d digits where $d \leq \lfloor \log_b(k+1) \rfloor$ and k is the largest number in the list

The algorithm runs in linear time when b and n are of the same size magnitude, so knowing n, b can be manipulated to optimize the running time of the algorithm.

1.9 Heap Sort

Heapsort is a comparison-based sorting technique based on Binary heap data structure.

It is similar to selection sort where we first find the maximum element and place the maximum element at the end.

Heap Sort Algorithm

Build a max heap from the

input data.

2. At this point the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally heapify the root of the tree.

3. Repeat above steps while size of the heap is greater than 1.

20 Insertion Sort

Insertion sort is a sorting algorithm that builds a final sorted array one element at a time.

Insertion sort has a average and worst case running time of $O(n^2)$.

The insertion sort algorithm iterates through an input array and removes one element per iteration. finds the place the element belongs in the array and then places it there.

Algorithm.

```
for i=1  to length(A)
    x= A[i]
    j = i-1

    while j >=0 and A[j]>x
        A[j+1] = A[j]
        j = j-1
    end while

    A[j+1] = x

end for
```