Data Structures

R. K. Ghosh

IIT Kanpur

Divide & Conquer Algorithm

The Paradigm

- The problem at hand is broken down into smaller sizes.
- Then smaller problems are solved possibly by brute force method.
- The solutions of the subproblems are combined into solution of the whole.
- For example, initially a problem of size n is broken into a subproblems each of size n/b, where $b > 1$.
- The splitting process is continued recursively, and subproblems becomes progressively smaller and smaller in size until each can be trivially solved.

Running Time

- Splitting process is simple.
- The smartness in solution lies in combination step when solutions of smaller problems are merged to produce solution of the problem of the original size.
- Analysis of running time is expressed as

$$T(n) = aT(n/b) + \text{ Work for merging}$$

- Recurrence is then solved to give running time.
- There is a master theorem which gives a template to solve the recurrence equation.

Revisiting Merge Sort

- Split the original input size n into two equal sized subproblems (of size $n/2$ each) .
- Each subproblem is further recursively split.
- The splitting is continued until problem size becomes 1.
- Since, one element list is trivially sorted, merging process begins from here on.
  - That is merging two sorted sequences to produce one sorted seqeunce double the size.
- After merging at the bottom level $n/2$ sorted lists of 2 elements are formed.
- These lists are further merged and we continue to build larger and larger sorted list after each process of merging.

Revisiting Quick Sort

- Quick sort is another example of DAC algorithm which you have seen already.
- The idea is to choose a pivot element which splits the input size n into two subproblems of size each equal to some fraction of n.
- The pivot is placed in the correct sorted position.
- Recursively the left part and right part is sorted using the same process of split by pivots.

Convex Hull

- A bunch of point given on a plane.
- Convex hull is used in many optimization problems.
- Our problem is to find all exterior tangents to generate a convex polygon which includes all the points either on or inside the polygon.

Definition (*Convex Hull Problem*)

Given n points on a plane $S = \{(x_i, y_i) | i = 1, 2, \ldots n\}$. Assume that no two points have same x coordinates or no two points have same y coordinates. And also no three points are colinear.

These assumptions allows us to look at pairs of points and find the segments that gives the convex hull. Convex hull is therefore smallest polygon containing all points.

Convex Hull

- CH(S) (convex hull of point set S) is represented by sequence of points that are on the boundary of the Hull in the clockwise order.
- It can be specified by a doubly linked list.
- What about a brute force algorithm without using DAC?
- Draw a line passing through a pair of input points.
- If all points are in one side of the above line then the above segment of the line between the input points, would be a segment of convex hull.
- Otherwise drop the the segment.

Running Time

- Since there could be O(n^2) segments.
- The worst case time to test for points being on one side of a single segment is O(n).
- So, overall running time is O(n^3)
- It makes sense to use DAC if we can do better than O(n^3) time.

DAC for Convex Hull

- At first break the problem into two subproblems by using a line L.
- The collection of points S_1 to left of L constitute one subproblem
- The points S_2 to right of line L defines another subproblem
- Construct CH(S_1) and CH(S_2) and merge.
- Merging process finds the upper tangent joining a pair of points.
  - One point from S_1 and
  - One point from S_2.

Upper and Lower Tangents

- The upper tangent is the line segement obtained by joining $a_L \in S_1$ and $b_R \in S_2$ which intersects L at highest y coordinate.
- Similarly, the line segement $a'_L \in S_1$ and $b'_R \in S_2$ intersecting L at lowest y defines lower tangent.
- Now the sequence of points $a_L, b_R, b_{R+1}, \cdots, b'_L, a'_L, \cdots, a_{L-1}$ defines the convex hull.
  - Consists of left chain from a'_L to a_L and right chain from b_R to b'_R.

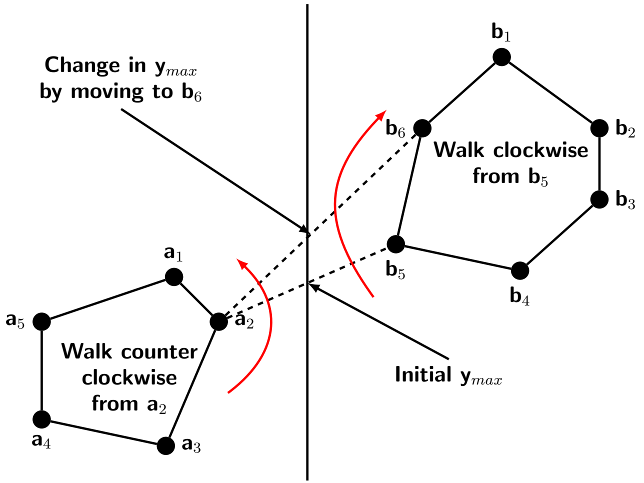
Running Time

- Since there could be n^2 segments to check the time of merging is O(n^2).
- Can we do better than that.
- For example can ge y_{max} and y_{min}.
- But how? Joining a_L (having max y coordinate) and b_R (having max y coordinate on right)?
- It is not true.
- There is a counter example to it.

Alternative Approach

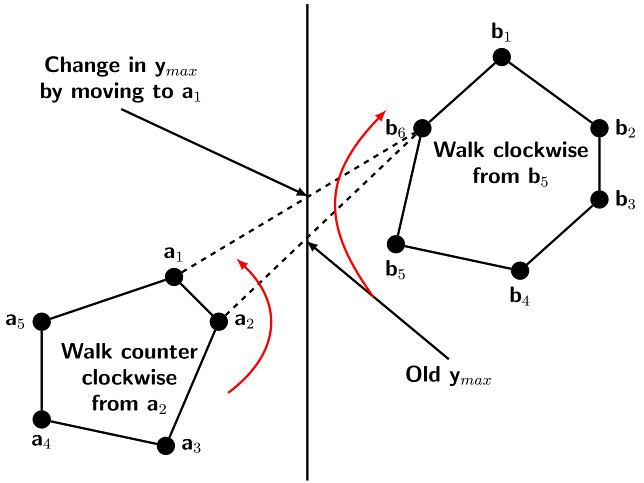
- Called two finger algorithm.
- Start from closest points to line L.
  - a_L having max x coordinate, and
  - b_R having min x coordinate.
- Mark intercept on L, let it be denoted by y_{max}.
- Then keep a_L fixed, move clockwise next point (of b_R) on the right hull.
- If y coordinate of the intercept increases, then repeat the same action by moving next point counter clock on left hull.

Illustration



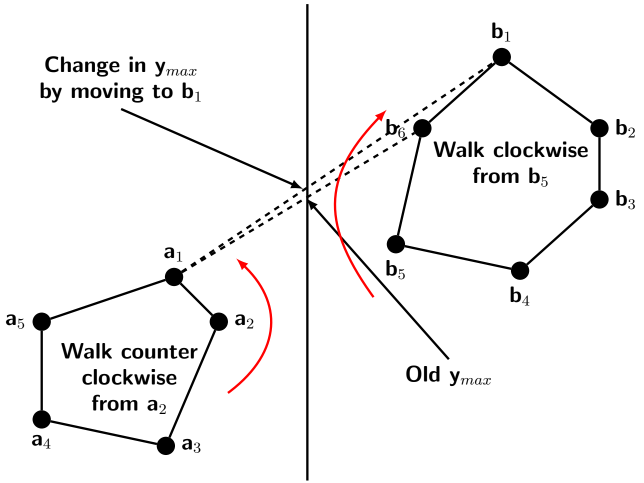
Change in $\mathbf{y}_{max}$ by moving to $\mathbf{b}_6$
$\mathbf{b}_1$
$\mathbf{b}_6$
$\mathbf{b}_2$
Walk clockwise from $\mathbf{b}_5$
$\mathbf{b}_3$
$\mathbf{a}_1$
$\mathbf{b}_5$
$\mathbf{b}_4$
$\mathbf{a}_5$
$\mathbf{a}_2$
Walk counter clockwise from $\mathbf{a}_2$
Initial $\mathbf{y}_{max}$
$\mathbf{a}_4$
$\mathbf{a}_3$

Illustration



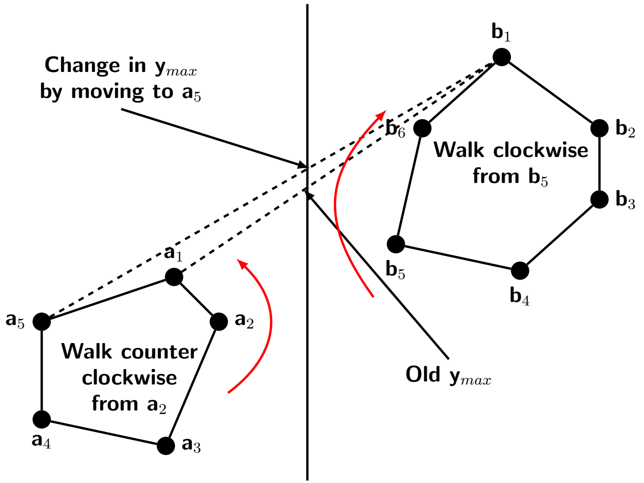
Change in y_{max} by moving to a_1
b_1
b_6
b_2
Walk clockwise from b_5
b_3
a_1
b_5
b_4
a_5
a_2
Walk counter clockwise from a_2
Old y_{max}
a_4
a_3

Illustration



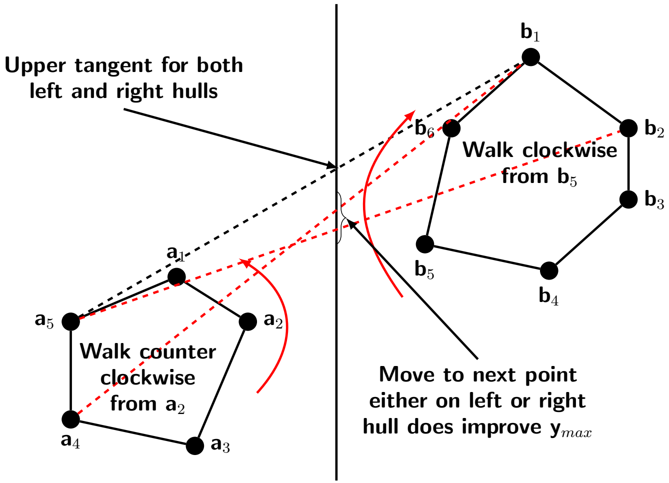
Change in y_{max} by moving to b_1
b_1
b_2
b_3
b_4
b_5
b_6
Walk clockwise from b_5
a_1
a_2
a_3
a_4
a_5
Walk counter clockwise from a_2
Old y_{max}

Illustration



Change in y_{max} by moving to a_5
b_1
b_2
b_3
b_4
b_5
b_6
Walk clockwise from b_5
a_1
a_2
a_3
a_4
a_5
Walk counter clockwise from a_2
Old y_{max}

Illustration



Upper tangent for both left and right hulls
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{b}_6$
Walk clockwise from $\mathbf{b}_5$
$\mathbf{b}_3$
$\mathbf{b}_5$
$\mathbf{b}_4$
$\mathbf{a}_1$
$\mathbf{a}_5$
$\mathbf{a}_2$
Walk counter clockwise from $\mathbf{a}_2$
$\mathbf{a}_4$
$\mathbf{a}_3$
Move to next point either on left or right hull does improve $\mathbf{y}_{max}$

Algorithm

```
// index pointer
int i = 1;
int j = 1;

// y is the method that returns the y intersection
    that takes
// two index pointers as arguments
while( y(i, j + 1) > y(i, j) || y(i - 1, j) > y(i,
    j)) {
      if( y(i, j+ 1) > y(i, j) )
            j = (j + 1) % p;
      if( y(i - 1, j) > y(i, j) )
            i = (i - 1) % q;
}
return (i,j);
```

Running Time

- The algorithm checks each point on the left hull and each point the right hull exactly once.
- Only O(1) time is spent for checking whether y_{max} improves or not.
- The checking is performed alternatively. Either moving sequentially to the next point on the left (counter clockwise walk) or to the next point on the right hull (clockwise walk).
- This implies that the running time for finding upper tangent is O(n).
- Similarly, the lower tangent will require O(n) time.
- Hence, the merge process takes O(n) time.

Running Time

- Overall running time is given by the following recurrence equation:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + O(n) \\ &= O(n \log n) \end{aligned}$$

Summary

- We looked at the general template for developing Divide and Conquer Algorithms.
- Revisited two sorting algoirthms, namely merge sort and quick sort which employ DAC strategy.
- Convex hull problem was then introduced and a simple brute force algorithm was explained in details.
- Running time of the above algorithm was O(n^3).
- We then examined how a DAC strategy can be developed for construction convex hull of a set of points in 2D.
- Finally, an improved merging technique was developed for merging two convex hulls which leads to an efficient DAC algorithm for Convex Hull.

Summary

- Although we did not separately examine the paradigm of Greedy Algorithm, we have seen a number of examples of algorithms using greedy strategy.
- Characteristics of greedy algorithms were introduced in course of discussion on these algorithms.
- It was explained that every greedy algorithms has two major characteristics, namely, *Greedy Choice Property* and *Principle of Optimality*.
- The examples greedy algorithms we spent some time are: shortest path, minimum spanning tree, Huffman Coding.