Algorithm Design Manual Notes

Vikas Goudar

3 October, 2024

Contents

1 Introduction to Algorithm Design

1.1 Insertion Sort

Insertion sort incrementally adds an element to a sorted list and processes it

```
insertion_sort(item s[], int n){
  int i,j; // counters

  for (i = 1; i < n; ++i){
    j = i;
    while ((j > 0) && (s[j] < s[j-1])){
      swap(&s[j], &s[j-1]);
        --j;
    }
  }
}
```

1.2 Robot Tour Optimization (TSP)

Problem Statement There exists a robot which is used to solder points on a chip, each chip has a set of points that need to be soldered, specifically the robot is to start at a point and visit all other points and return. Time taken is proportional to the distance travelled, and a path is to be found which minimizes time taken.

Essentially this problem, can be summarized as

Input A set S of n points in the plane.

Output Shortest cycle that visits each point in set S

One possible solution that comes to mind (which is wrong) is to visit the node which is unvisited and closest. However it has cases where it fails, for example refer to Figure 1

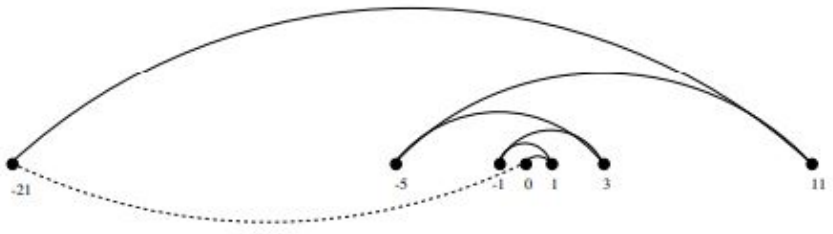

Figure 1: Case where nearest-neighbor heurestic fails

Where this algorithm fails is it doesn't know what to do in case of a tie where two nodes are equally close, and results in paths as shown in Figure 1.
We try a different heurestic, one that connects closes pair of endpoints while ensuring that their connection does prematurely end our cycle.

Initially, each vertex is treated is treated as its own seperate set, we progressively merge the sets until one set remains.

```
let n be the number of points in set S
for (int i = 1; i<n; ++i){
  d = inf
  for each pair of endpoints (s,t) from distinct vertex sets{
    if (dist(s, t) <= d){
      sm = s
      tm = t
      d = dist(s, t)
    }
  }
  connect(sm,st) -> add to same set
}
connect the last two end points remaining
```

However this algorithm also fails in certain scenarios. Refer to Figure 2

OptimalTSP(P)
$d = \infty$
For each of the $n!$ permutations P_i of point set P
If $(cost(P_i) \leq d)$ then $d = cost(P_i)$ and $P_{min} = P_i$
Return P_{min}

Figure 2: Case where nearest-endpoints heurestic fail

A possible fix is to try out all possible permutations and choose the optimal solution out of them. But this is too slow.
A quest to solve this problem in an efficient way is called the Travelling Salesman Problem(TSP)

1.3 Job Scheduling

Problem can be summarized as

Input A Set I of n intervals

Output Largest subset of mutually non-overlapping intervals from set I

Two possible solutions are Shortest-Job-First and Earliest-Job-First, however there exists cases where they fail. Refer to Figure 3

Figure 3: (l) earliest job first fails and (r) shortest job first fails

A brute-force method is to generate all 2^n subsets.
Test all 2^n subsets of intervals and return the largest subset that -¿ consist of mutually non-overlapping intervals

However thankfully :D a better solution exists
Choose the job which has earliest ending time

```
sort jobs in ascending_order w.r.t finish_time
for (job in jobs){
  if (job.start_time >= last_end_time){
    # Process job
    last_end_time = job.finish_time
  }
}
```

1.4 Proof by induction/recursion

Most of the algorithms in computer science invlove recursion/induction.
There are two main steps to check if an induction/recursion algorithm is correct

Base Case Prove that the statement $P(n)$ holds for the smallest value of n, (*usually n = 0/1*)

Inductive Step Assume that $P(n)$ holds for some arbitrary number n, or every number uptil n. Then, prove that $P(n+1)$ is also true
If both case hold true, the algorithm is correct for all $n \geq 0$ or any other base case.

Example 1 Prove the correctness of the following recursive algorithm foir incrementing natural numbers, $y- > y+1$

```
Increment(y){
  if (y == 0){
    return 1
  }
  else{
    if (y mod 2){
      return 2*Increment([y/2])
    }
    else {
      return y + 1
    }
  }
}
```

Base case: (y==0) is handled correctly.
Assume the function works correctly for the general case of $F(n-1)$, now proving for general case of $F(n)$.
Even numbers, $y+1$ is retured, so its correct
Odd numbers, $2*Increment([y/2])$ is returned, since we our assumption holds for all cases $y \leq n-1$.

$$
\begin{aligned}
& & y &= 2m+1 \quad \text{for some integer m} \\
2 \cdot \text{Increment}\left(\frac{2m+1}{2}\right) &= 2 \cdot \text{Increment}\left(m+\frac{1}{2}\right) & & \\
&= 2 \cdot \text{Increment}(m) & & \\
&= 2(m+1) & & \\
&= 2m+2 & & \\
&= y+1 & &
\end{aligned}
$$