19CSE302 – Design and Analysis of algorithms
Assignment – 10.11.2021

1. Stella is hosting a Party. Stella invited few guests for the dinner and the party is arranged in the garden with few dining tables. Suppose that the host wants her friends to know each other. Use Branch and Bound method to solve the problem, also implement the solution.

15/11/21

Assignment

R. Abhinav.
CB.EN.U4CSE19453

1) Given scenario:,

Stella – hosting

Case

(i) stella wants her friends to know each other.

(ii) Stella's friends may not be known each other.

let us solve this using branch & bound

Approach:-

1. for every node of a state-space tree, a bound on the best value of the objective function, on any solution.
2. The value of node's bound is not better than a value of the best solution seen so far.
3. The node represents no feasible solutions because the constraints of the problem already violated.
4. The subset of feasible solutions represented by node consists of single point.

In our case;

- we compare the value of the objective function for this feasible solution.

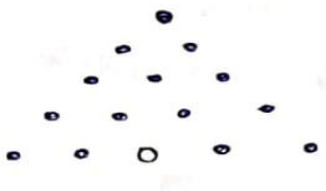
2. Puzzle Pegs :

2) Back-tracing approach :-

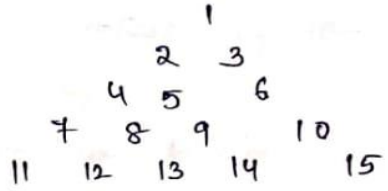
- Here, consists of building a set of all solutions incrementally.
- Since, problem have constraints;
  the solutions that fail to satisfy them will remove.

Working :-

- When starts exploring the solutions,
  (i) a bounding function is applied so that algorithm can check if the so-far built solution satisfies constraints.
  (ii) If it does, it continues searching.
  else; the branch would be eliminated.
  and algo. goes back to before level.



→



Algorithm :-

- for move in range [list]

```
if PHP
  return
else move +1

if true:
   function call
```

- if recursion true solution found.

True here is last peg left on board in specific position.

False, no solution.

↓

back trace & iterate to next move of list.

3. **S**ay that a pattern P of length m is a circular substring of a text T of length n if there is an index $0 \leq i < m$, such that $P = T[n - m + i..n - 1] + T[0..i - 1]$, that is, if P is a substring of T or P is equal to the concatenation of a suffix of T and a prefix of T. Give an O(n +m)-time algorithm for determining whether P is a circular substring of T.

3) Given,

Pattern-P of length m

↓

is a circular substring of T-text of length n

here; n > m

- P → is a substring of text T

To provide; O(n+m) for determining whether P is circular substring of T.

- also given;

$P = T[n-m+k..n-1] + T[0.... k-1]$.

to prove O(n+m);

we will apply KMP algorithm on text & pattern:

now new text $T = T[n-m.....n] + T[0...m]$.

Two components of algorithm :-

- Prefix function: how the pattern is searched & matched
- KMP matcher: we use to get final exact pattern matching.

Pseudo code :-

```
on T & P :-
let, Constraints 1 ≤ m ≤ n

q ← 0
i ← 0
while (i<n)
        P[1...q] == T[i-q+1...i]
{
 if P[q+1] == T[i+1]
 {
  q ← q+1
  i ← i+1
 if (q==m)
  {
  Out (i-q);
 q ← π(q);
 }
}
```

```
else
{
 if (q == 0)
  {i ← i+1
  }
 else
  {
   q ← Π(q)
  }
}
```

Walk through:.

1. First initialsed q, i
2. check for any matching pattern 'P' in text 'T'.
3. Check for next pattern to match with text 'T'.
4. next character matches
5. Check for new matches.
   - Shift the given pattern and check for next matches.
   - Otherwise check if there is any mismatch occured.

Finally;

- generate a new text $T' = T[n-m \ldots n] + T[0 \ldots m]$
- Next using KMP on T & P.
- After T' defined; Pattern is matched with Text.
- Get the final list of all numbers in such a way that pattern occurs with shift in T.
- Time for determing whether 'P' is a circular substring of T is

$$\boxed{O(n+m)}$$

4. A crime has been committed in the city and the forensics person is able to identify some unknown DNA sample ‘UD’ on the crime scene. On the other hand, police has identified ‘N=5’ suspects who have the highest probability to be the murderer. The probability of a suspect to be a murderer is obtained by matching his DNA with unknown DNA ‘UD’. Arrange the suspects in the order of their decreasing probability to be a murderer.

4) To find:-

- minimum window in string "s" having all characters as in substring "T"

let

main string = "s"

required string = "T"

Algorithm:-

1. Traverse through the string "T" and find occurence of all characters into array.
2. Traverse through the string "s" and find occurence letter by letter.
3. If letter found is in required string "T" increment count

   else:

   don't increment

   else:

   if occurence of letter > required string letter don't increment
4. check if count > len (required string)

   ⟶ equal then print

   else:

   minimize window by removing letters.

   that are not in away so if occurence of char

   is greater than requried occurence.

Time Complexity:-

- O(len(string))

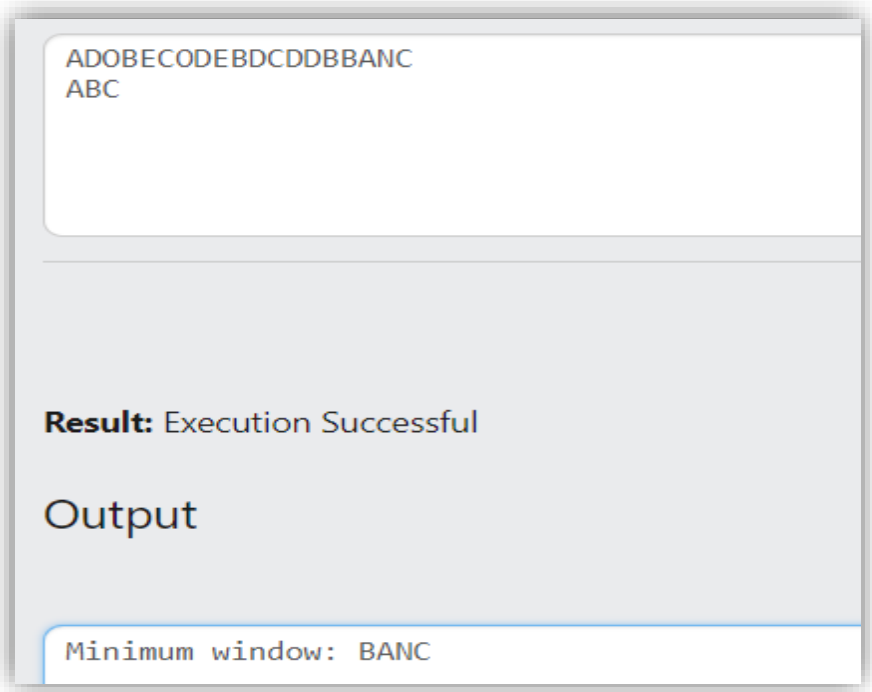
  ↓ when no string found;

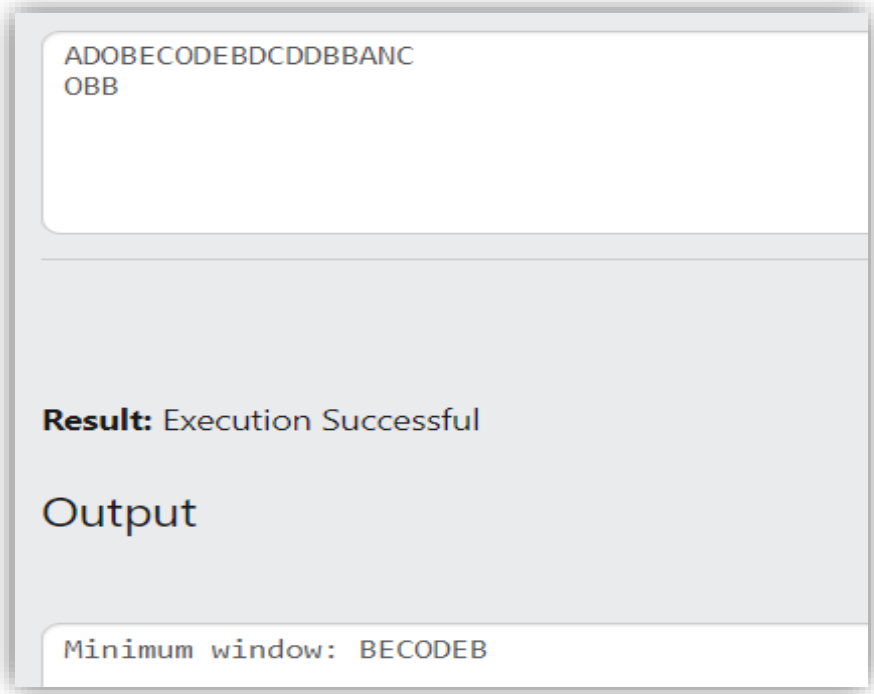
  So, iterate through the complete string.

Code:

```
def SlidingWindow(main_text, sub_text):

    main_len = len(main_text)
    if main_len < len(sub_text):
        return -1
    main_occ = [0] * 256

    start = 0
    result = main_len + 1
    count = 0

    for i in sub_text:
        main_occ[ord(i)] += 1
        if main_occ[ord(i)] == 1:
            count += 1
    j = 0
    i = 0

    while(j < main_len):
        main_occ[ord(main_text[j])] -= 1
        if main_occ[ord(main_text[j])] == 0:
            count -= 1

            while count == 0:
                if result > j - i + 1:
                    result = j - i + 1
                    start = i
                main_occ[ord(main_text[i])] += 1
                if main_occ[ord(main_text[i])] > 0:
                    count += 1
                i += 1
        j += 1
    if result > main_len:
        return "-1"
    return main_text[start:start+result]

if __name__ == "__main__":
    main_text = input()
    sub_text = input()
    result = SlidingWindow(main_text, sub_text)
    print("Minimum window:", result)
```

Output:


ADOBECODEBDCDDBBANC
ABC
Result: Execution Successful
Output
Minimum window: BANC


ADOBECODEBDCDDBBANC
OBB
Result: Execution Successful
Output
Minimum window: BECODEB