Transform to Palindrome

locked

by gravity0905

Editorial by gravity0905

An undirected graph can be built where letters are the nodes and tranformations are edges. Using DFS, we can find out the connected components in the graph. We can then find out whether 2 letters can be transformed to each other by checking whether they belong to the same connected component in constant time complexity.

Let $DP[i][j]$ denote the length of the longest Palindromic Subsequence between the indices i and j (both i and j inclusive) in the string. Clearly, $DP[i][i] = 1$ for all indices $0 \le i < L$. Consider the letters x and y present in the string at positions i and j respectively such that $i < j$.

- If x and y can be transformed to each other and $j > i + 1$, then $DP[i][j] = DP[i+1][j-1] + 2$
- If x and y can be transformed to each other and $j = i + 1$, then $DP[i][j] = 2$
- If x and y cannot be transformed to each other then $DP[i][j] = max(DP[i+1][j], DP[i][j-1])$

Statistics

Difficulty: Medium
Time Complexity: $O(N+K+L^2)$
Required Knowledge: DFS, DP
Publish Date: Apr 13 2017

Set by gravity0905

Problem Setter's code :

```
#include <bits/stdc++.h>
using namespace std;

const int maxL = 1001;
const int maxN = 100001;

int arr[maxL], LPS[maxL][maxL];
int mark[maxN];
vector<int> G[maxN];

void dfs(int s, int cc)
{
    mark[s] = cc;
    for(vector<int>::iterator it= G[s].begin(); it!=G[s].end(); ++it)
        if(!mark[*it])
            dfs(*it, cc);
}


int lps(int arr[], int L)
{
   int i, j, len;
   for(i=0; i<L; i++)
       LPS[i][i] = 1;

   for(len=2; len<=L; len++)
   {
       for(i=0; i<L+1-len; i++)
       {
           j = i + len - 1;
           if(mark[arr[i]] == mark[arr[j]])
           {
               if(len == 2)
                   LPS[i][j] = 2;
               else
                   LPS[i][j] = LPS[i+1][j-1] + 2;
           }
           else
               LPS[i][j] = max(LPS[i][j-1], LPS[i+1][j]);
       }
   }
   return LPS[0][L-1];
}

int main()
{
    int N, K, L;
    scanf("%d %d %d", &N, &K, &L);
    for(int i=0; i<K; i++)
    {
       int x, y;
       scanf("%d %d", &x, &y);
       G[x].push_back(y);
       G[y].push_back(x);
    }

    int cc = 1;
    for(int i=1; i<=N; ++i)
        if(!mark[i])
        {
            dfs(i, cc);
            cc++;
        }
    for(int i=0; i<L; i++)
        scanf("%d", &arr[i]);
    printf("%d\n", lps(arr, L));
    return 0;
}
```