Team Notebook

July 21, 2020

Contents

1 Algorithms . . . 2
- 1.1 DFS . . . 2
  - 1.1.1 DFS on graph . . . 2
  - 1.1.2 DFS on tree . . . 2
  - 1.1.3 Flatten tree . . . 2
- 1.2 String . . . 2
  - 1.2.1 kmp . . . 2

2 Data Structure . . . 2
- 2.1 segment tree l to r-1 . . . 2

1 Algorithms

1.1 DFS

1.1.1 DFS on graph

```
#include <bits/stdc++.h>
using namespace std;
vector<int> adj[10001];
bool vis[10001]={0};
int ii;
void dfs(int v)
{  vis[v]=true; ii++;
   cout<<v;
   for(auto u: adj[v])
   if(!vis[u])
   dfs(u);
}
int main() {
 int n,m,u,v;
 cin>>n>>m;
 for(int i=0;i<m;i++)
 { cin>>u>>v;
   adj[u].push_back(v);
   adj[v].push_back(u);
 }
 dfs(1);
 return 0;
}
```

1.1.2 DFS on tree

```
#include <bits/stdc++.h>
using namespace std;
vector<int> adj[10001];
int ii;
void dfs(int v, int par){
 cout<<v<<" ";
 for(auto u: adj[v])
  {  if (u == par) continue;
   dfs(u, v);
  }
}
int main() {
 int n,u,v;
 cin>>n;
 for(int i=0;i<n-1;i++)
 { cin>>u>>v;
   adj[u].push_back(v);
   adj[v].push_back(u);
 }
 dfs(1,-1);
 return 0;
}
```

1.1.3 Flatten tree

```
   int timer = 0;
 void dfs(int v, int par){
   entr[v] = timer++;
  for(auto u: adj[v])
  { if (u == par) continue;
  dfs(u, v);
  }
   ext[v] = timer++;
   }

 vector<LL> flattenedTree(2*n);
       for(int u = 0; u < n; u++)
        {
           flattenedTree[entr[u]] = s[u];
           flattenedTree[ext[u]] = -s[u];
        }
```

1.2 String

1.2.1 kmp

```
#include<iostream>
using namespace std;
#define MAXN 100000
int b[MAXN]={0};

void preprocess(string s){
    int i = 0, j =-1;
    b[0] = -1;
    while(i<s.size()){
        while(j>=0 && s[i]!=s[j])
            j = b[j];
        j++;i++;
        b[i] = j;
    }
}

void kmpsearch(string t, string s){
    int i=0,j=0;
    int n = t.size(), m = s.size();
    while(i<n){
        while(j>=0 && t[i] != s[j])
            j = b[j];
        j++;i++;
        if(j == m){
            printf("Pattern found at position %d\n",i-j);
            j = b[j];
        }
    }
}
int main(){
    string t, s;
    getline(cin,t);
    getline(cin,s);
    preprocess(s);
    kmpsearch(t,s);
    return 0;
}
```

2 Data Structure

2.1 segment tree l to r-1

```
struct segtree
{ int size;
  vector<ll> sums;
  void init(int n)
  {   size = 1;
      while(size<n) size*=2;
      sums.assign(2 * size, 0LL);
  }
  void pull(int x)
  {
      sums[x] = sums[2*x+1] + sums[2*x+2];
  }
  void build(vector<int> &a, int x, int lx, int rx)
  {
      if(rx-lx==1)
      {
          if(lx<(int)a.size())
          {
              sums[x]=a[lx];
          }
         return;
      }
      int m = (lx+rx)/2;
```

```
    build(a,2*x+1,lx,m);
    build(a,2*x+2,m,rx);
    pull(x);
}
void build(vector<int> &a)
{
    build(a,0,0,size);
}
void update(int idx, int val, int x, int lx, int rx)
{
   if(rx-lx==1)
   {
      sums[x] = val;
      return;
   }
   int m = (lx+rx)/2;
   if(idx<m)
   update(idx,val,2*x+1,lx,m);
   else
   update(idx,val,2*x+2,m,rx);
   pull(x);
}
void update(int idx,int val)
{
    update(idx,val,0,0,size);
}
ll query(int l,int r, int x, int lx, int rx)
{
   if(r<=lx || l>=rx) return 0;
   if( l<=lx && rx<=r ) return sums[x];
   int m = (lx+rx)/2;
   ll a = query(l,r,2*x+1,lx,m);
   ll b = query(l,r,2*x+2,m,rx);
   return a+b;
}
ll query(int l,int r)
{
   return query(l,r,0,0,size);
}
};
```