Algorithm Library

palayutm

June 3, 2019

Contents

1 图论

1.1 connected graph

1.1.1 割点

```
void dfs(int u, int fa)
{
    int low[u] = pre[u] = ++dfs_block;
    int child = 0;
    for(int i = 0; i < G[u].size(); i++)
    {
        int v = G[u][i];
        if(!pre[v])
        {
            child++;
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= pre[u])
                iscut[u] = true;
        }
        else if(v != fa)
            low[u] = min(low[u], pre[v]);
    }
    if(fa < 0 && child == 1) iscut[u] = false;
}
```

1.1.2 割边

```
struct node
{
    int fir, sec;
    bool operator < (node a) const {
        if(fir != a.fir) return fir < a.fir;
        return sec < a.sec;
    }
};
void dfs(int u, int fa)
{
    low[u] = pre[u] = ++dfs_block;
    for(int i = 0; i < G[u].size(); i++)
    {
        int v = G[u][i];
        if(!pre[v])
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] > pre[u])
                all[co++] = node{u, v};
        }
        else if(v != fa)
            low[u] = min(low[u], low[v]);
    }
}
```

1.1.3 强连通分量

```
void dfs(int u)
{
    low[u] = dfn[u] = ++dfs_block;
    S.push(u);
    for(int i = 0; i < G[u].size(); i++)
    {
        int v = G[u][i];
        if(!dfn[v])
        {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else if(!sccno[v])
            low[u] = min(low[u], low[v]);
    }
    if(low[u] == dfn[u])
    {
        ++scc_cnt;
        while(1)
        {
            int x = S.top(); S.pop();
            sccno[x] = scc_cnt;
            if(x == u) break;
        }
    }
}
```

1.1.4 强连通分量缩点

```
void dfs(int u, int fa)
{
    low[u] = dfn[u] = ++dfs_block;
    S.push(u);
1   bool flag = false;
    for(int i = 0; i < G[u].size(); i++)
    {
        int v = G[u][i];
        if(!dfn[v])
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] > dfn[u])
                bridge++;
        }
        else if(v != fa || flag)
            low[u] = min(low[u], low[v]);
2       if(v == fa) flag = true;
    }
    if(low[u] == dfn[u])
    {
        ++scc_cnt;
        while(1)
        {
            int x = S.top(); S.pop();
            sccno[x] = scc_cnt;
            if(x == u) break;
        }
    }
}
void find_scc()
{
    init();
    for(int i = 1; i <= n; i++)
        if(!dfn[i]) dfs(i, -1);
}
for(int i = 1; i <= n; i++)
    for(int j = 0; j < G[i].size(); j++)
    {
        int v = G[i][j];
        if(sccno[i] != sccno[v])
        {
            M[sccno[i]].push_back(sccno[v]);
            M[sccno[v]].push_back(sccno[i]);
        }
    }
void bfs(int s)
```

```
{
    queue<int> q;
    q.push(s);
    memset(vis, 0, sizeof(vis));
    vis[s] = 1; dis[s] = 0; last_node = s; ret = 0;
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        for(int i = 0; i < M[u].size(); i++)
        {
            int v = M[u][i];
            if(!vis[v])
            {
                vis[v] = 1;
                dis[v] = dis[u] + 1;
                if(dis[v] > ret)
                {
                    ret = dis[v];
                    last_node = v;
                }
                q.push(v);
            }
        }
    }
}
    bfs(1);
    bfs(last_node);
```

2 String

2.1 KMP

```
/*
 * Args:
 *   s[]: string
 * Return:
 *   fail[]: failure function
 */
int fail[N];
void getfail(char s[])
{
  fail[0] = -1;
  int p = -1;
  for (int i = 0; s[i]; i ++)
  ↪  {
    while (p!=-1 &&
    ↪  s[i]!=s[p])  p =
    ↪  fail[p];
    fail[i+1] = ++p;
  }
}
```

2.2 Suffix Automaton

```
/*
 * 1 call init()
 * 2 call add(x) to add every
↪  character in order
 *
 * Args:
 * Return:
 *   an automaton
 *   link: link path pointer
 *     len: maximum length
 */
struct node{
  node* chd[26], *link;
  int len;
}a[3*N], *head, *last;
int top;
void init()
{
  memset(a, 0, sizeof(a));
  top = 0;
  head = last = &a[0];
}
void add(int x)
{
  node *p = &a[++top], *mid;
  p->len = last->len + 1;
  mid = last, last = p;
  for (; mid && !mid->chd[x];
  ↪  mid = mid->link)
  ↪  mid->chd[x] = p;
  if (!mid)  p->link = head;
  else{
    if (mid->len + 1 ==
    ↪  mid->chd[x]->len) {
      p->link = mid->chd[x];
    } else {
      node *q = mid->chd[x],
      ↪  *r = &a[++top];
      *r = *q, q->link =
      ↪  p->link = r;
      r->len = mid->len + 1;
      for (; mid &&
      ↪  mid->chd[x] == q;
      ↪  mid = mid->link)
      ↪  mid->chd[x] = r;
    }
  }
}
```