

Algorithm Library

palayutm

February 26, 2017

Contents

1	Graph Theory	3
1.1	Minimum Spanning Tree	3
1.1.1	Kruskal	3
1.2	Shortest Path	3
1.2.1	SPFA	3
2	String	5
2.1	KMP	5
2.2	Suffix Automaton	5

1 Graph Theory

1.1 Minimum Spanning Tree

1.1.1 Kruskal

```

/*
 * Args:
 *   edge: edges of graph, (u, v, w) = (edge[i].second.first,
 *   ↪ edge[i].second.first, edge[i].first)
 *   n: number of node, from 1 to n
 * Return:
 *   minimum spanning tree
 */
vector<pair<int, pair<int, int> > > edge;
int pre[N];
int find(int u)
{
    return u == pre[u] ? u : pre[u] = find(pre[u]);
}
int Union(int u, int v)
{
    pre[find(u)] = find(v);
}
int kruskal(int n)
{
    for (int i = 1; i <= n; i++) pre[i] = i;
    sort(edge.begin(), edge.end());
    int ans = 0;
    for (auto x : edge) {
        int u = x.second.first, v = x.second.second, w = x.first;
        if (find(u) != find(v)) {
            Union(u, v);
            ans += w;
        }
    }
    return ans;
}

```

1.2 Shortest Path

1.2.1 SPFA

```

/*
 * Args:

```

```

*   g[]: graph, (u, v, w) = (u, g[u][i].first,
↪   g[u][i].second)
*   st: source vertex
*   Return:
*   dis[]: distance from source vertex to each other vertex
*/
vector<pair<int, int> > g[N];
int dis[N], vis[N];
void spfa(int st)
{
    memset(dis, -1, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    q.push(st);
    dis[st] = 0;
    vis[st] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = false;
        for (auto x : g[u]) {
            int v = x.first, w = x.second;
            if (dis[v] == -1 || dis[u] + w < dis[v]) {
                dis[v] = dis[u] + w;
                if (!vis[v]) {
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

```

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;
        }
    }
}

```