
Contents

1	结构	1
1.1	并查集	1
2	图论	3
2.1	最短路	3
2.1.1	最短路径	3
2.2	生成树	4
2.2.1	最小生成树	4
3	应用	5
3.1	Joseph	5
3.2	位操作	6

Chapter 1

结构

1.1 并查集

```
1 // 带路径压缩的并查集，用于动态维护查询等价类
2 // 图论算法中动态判点集连通常用
3 // 维护和查询复杂度略大于  $O(1)$ 
4 // 集合元素取值  $1..MAXN-1$ (注意  $0$  不能用!), 默认不等价
5
6 const int MAXN = 100000;
7
8 #include <cstring>
9 #define _run(x) for(; p[t = x]; x = p[x], p[t] = (p[x] ? p[x] : x))
10 #define _run_both _run(i); _run(j)
11
12 class DSet {
```

```
13 public:
14     int p[MAXN],t;
15
16     void init() {
17         memset(p, 0, sizeof(p));
18     }
19
20     void setFriend(int i, int j) {
21         _run_both;
22         p[i] = (i == j ? 0 : j);
23     }
24
25     bool isFriend(int i, int j) {
26         _run_both;
27         return i == j && i;
28     }
29 };
```

Chapter 2

图论

2.1 最短路

2.1.1 最短路径

```
1 //单源最短路径, 用于路权相等的情况,dijkstra 优化为 bfs, 邻接表形式, 复杂度  $O(m)$ 
2 //求出源  $s$  到所有点的最短路径, 传入图的大小  $n$  和邻接表  $List$ , 边权值  $len$ 
3 //返回到各点最短距离  $mind[]$  和路径  $pre[],pre[i]$  记录  $s$  到  $i$  路径上  $i$  的父结点, $pre[s]=-1$ 
4 //可更改路权类型, 但必须非负且相等!
5 const int MAXN = 200;
6 const int INF = 1000000000;
7
8 struct Edge {
9     int from, to;
10    Edge *next;
11};
12
13 template <class elemType> void dijkstra(int n, const Edge *list[], elemType len, int s, \
14 elemType *mind, int *pre) {
15     Edge *t;
16     int i, que[MAXN], f = 0, r = 0, p = 1, L = 1;
17     for (i = 0; i < n; i++) {
18         mind[i] = INF;
19     }
20     mind[que[0] = s] = 0;
21     pre[s] = -1;
22     for (; r <= f; L++, r = f + 1, f = p - 1) {
23         for (i = r; i <= f; i++) {
24             for (t = list[que[i]]; t; t = t->next) {
25                 if (mind[t->to] == INF) {
26                     mind[que[p++] = t->to] = len * L;
27                     pre[t->to] = que[i];
28                 }
29             }
30         }
31     }
32 }
```

2.2 生成树

2.2.1 最小生成树

```

1 //无向图最小生成树,prim 算法, 邻接阵形式, 复杂度  $O(n^2)$ 
2 //返回最小生成树的长度, 传入图的大小  $n$  和邻接阵  $mat$ , 不相邻点边权  $INF$ 
3 //可更改边权的类型, $pre[]$  返回树的构造, 用父结点表示, 根节点 (第一个) $pre$  值为  $-1$ 
4 //必须保证图的连通的!
5 const int MAXN = 200;
6 const int INF = 1000000000;
7
8 template <class elemType>
9 elemType prim(int n, const elemType mat[][MAXN], int *pre) {
10     elemType mind[MAXN], ret = 0;
11     int v[MAXN], i, j, k;
12     for (i = 0; i < n; i++) {
13         mind[i] = INF;
14         v[i] = 0;
15         pre[i] = -1;
16     }
17     for (mind[j = 0] = 0; j < n; j++) {
18         for (k = -1, i = 0; i < n; i++) {
19             if (!v[i] && (k == -1 || mind[i] < mind[k])) {
20                 k = i;
21             }
22         }
23         v[k] = 1;
24         ret += mind[k];
25         for (i = 0; i < n; i++) {
26             if (!v[i] && mat[k][i] < mind[i]) {
27                 mind[i] = mat[pre[i] = k][i];
28             }
29         }
30     }
31     return ret;
32 }

```

Chapter 3

应用

3.1 Joseph

```
1 // Joseph's Problem
2 // input: n,m      -- the number of persons, the interval between persons
3 // output:         -- return the reference of last person
4
5 int josephus0(int n, int m) {
6     if (n == 2) {
7         return (m % 2) ? 2 : 1;
8     }
9     int v = (m + josephus0(n - 1, m)) % n;
10    if (v == 0) {
11        v = n;
12    }
13    return v;
14 }
15
16 int josephus(int n, int m) {
17     if (m == 1) {
18         return n;
19     } else if (n == 1) {
20         return 1;
21     } else if (m >= n) {
22         return josephus0(n, m);
23     }
24     int l = (n / m) * m;
25     int j = josephus(n - (n / m), m);
26     if (j <= n - l) {
27         return l + j;
28     }
29     j -= n - l;
30     int t = (j / (m - 1)) * m;
31     if ((j % (m - 1)) == 0) {
32         return t - 1;
33     }
34     return t + (j % (m - 1));
35 }
```

3.2 位操作

```
1 // 遍历一个掩码的所有子集掩码, 不包括 0 和其自身
2 // 传入表示超集的掩码
3 void iterateSubset(int mask) {
4     for(int sub = (mask - 1) & mask; sub > 0; sub = (sub - 1) & mask) {
5         int incsub = ~sub & mask; // 递增顺序的子集
6         // gogogo
7     }
8 }
9
10 // 求一个 32 位整数二进制 1 的位数
11 // 初始化函数先调用一次
12
13 int ones[256];
14
15 void initOnes() {
16     for (int i = 1; i < 256; ++i)
17         ones[i] = ones[i & (i - 1)] + 1;
18 }
19
20 int countOnes(int n) {
21     return ones[n & 255] + ones[(n >> 8) & 255] + ones[(n >> 16) & 255] + ones[(n >> 24) & 255];
22 }
23
24
25 // 求一个 32 位整数二进制 1 的位数的奇偶性
26 // 偶数返回 0, 奇数返回 1
27 int parityOnes(unsigned n) {
28     n ^= n >> 1;
29     n ^= n >> 2;
30     n ^= n >> 4;
31     n ^= n >> 8;
32     n ^= n >> 16;
33     return n & 1; // n 的第 i 位是原数第 i 位到最左侧位的奇偶性
34 }
```