代码簿

September 9, 2021

1. 杂项

- **Merge Sort**

```
1  int arr[MAXLEN]; // 数组
2  int tmp[MAXLEN]; // 暂存
3  int ans  = 0; // 逆序对
4
5  void mergeSort(int left, int right){
6    if(left >= right) return;
7    int mid = left + ((right - left) >> 1);
8    mergeSort(left, mid);
9    mergeSort(mid + 1, right);
10   int l = left;
11   int r = mid + 1;
12   int idx = 0;
13   while(l <= mid && r <= right){
14     tmp[idx++] = arr[l] <= arr[r] ? arr[l++]
15     : ((ans += mid - l + 1 /* 逆序对 */ ), arr[r++]);
16   }
17   while(l <= mid){
18     tmp[idx++] = arr[l++];
19   }
20   while(r <= right){
21     tmp[idx++] = arr[r++];
22   }
23   for(int i = 0; i < idx; ++i){
24     arr[left + i] = tmp[i];
25   }
26 }
```

- **Disjoint Set**

```
1  int djs[MAXLEN];
2  //init for(int i = 0; i < size; ++i)djs[i] = i;
3
4  int find(int id){
5    return djs[id] == id ? id : djs[id] = find(djs[id]);
6  }
7  bool uSet(int a, int b){
8    int pa = find(a), pb = find(b);
9    return pa == pb ? true : (djs[pa] = pb, false);
10 }
11
12 //with dummy head
13
14 int djs[MAXLEN];
15 //init  for(int i = 0; i < 2 * n; ++i) djs[i] = i; for(int i = 0; i < n; ++i) djs[i] = n + i;
16
17 int find(int id){
18   return djs[id] == id ? id : djs[id] = find(djs[id]);
19 }
20 bool uSet(int a, int b){
21   int pa = find(a), pb = find(b);
22   return pa == pb ? true : (djs[pa] = pb, false);
23 }
```

2. 图论

- **Eular Path**

```
1  int cd[MAXLEN], rd[MAXLEN]; //有向图
2  vector<pair<int, int>> adjs[MAXLEN]; //first for 点, second fir 边
3  bool visit[MAXLEN]; //边
```

```cpp
4
5   int cnt = 0; //判断联通(cnt == size)
6
7   void dfs(int id){
8       for(auto &adj : adjs){
9           if(!visit[adj.second]){
10              ++cnt;
11              visit[adj.second] = true;
12              dfs(adj.first);
13              //一些操作
14          }
15      }
16  }
17
18  bool judge(){
19      //无向(边为偶数除了端点)
20      int cnt = 0;
21      int qidian  = 0; // 起点
22      for(int i = 0; i < n; ++i){
23          if(adjs[i].size() & 1) ++cnt, qidian = i;
24      }
25      return (cnt == 2 || cnt == 0);
26
27      //有向((出度 - 入度) = 1 && (入度 - 出度) = 1 || 欧拉环)
28      int rdc = 0, cdc = 0;
29      int qidian = 0;
30      for(int i = 0; i < n; ++i){
31          if((cd[i] - rd[i] == 1)) ++cdc;
32          else if((rd[i] - cd[i]) == 1) ++rdc, qidian = i;
33          else if((rd[i] - cd[i]) != 0) return false;
34      }
35
36      return ((cdc == 1 && rdc == 1) || (cdc == 0 && rdc == 0));
37  }
```

- **拓扑排序**

```cpp
1   vector<int> adjs[MAXLEN];
2   int rd[MAXLEN]; //入度
3   int n; //点的数量
4   vector<int> ans; //结果
5
6   void topSort(){
7       queue<int> q;
8       int cnt= 0;
9       for(int i = 0; i < n; ++i) if(!rd[i]) q.push(i), ++cnt, ans.push_back(i);
10      while(!q.empty()){
11          int f = q.front(); q.pop();
12          for(auto adj : adjs[f]){
13              --rd[adj];
14              if(!rd[adj]) q.push(adj), ++cnt, ans.push_back(adj);
15          }
16      }
17      if(cnt == n) {}//有找到
18      else {} //没找到
19  }
```

- **多元最短路**

```cpp
1   int v[MAXLEN][MAXLEN];
2   //初始化成0x3f3f3f3f，有边的地方为权重，自己为0
3
4   for(int k = 0; k < n; ++k){
5       for(int  i = 0; i < n; ++i){
6           for(int  j = 0; j < n; ++j){
7               v[i][j] = min(v[i][k] + v[k][j], v[i][j]); //最短路径
8               v[i][j] = min(v[i][j], max(v[i][k], v[k][j])); //最大的最小
9           }
10      }
11  }
```

- **单元最短路**

```cpp
1   vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 边权
2   int dis[MAXLEN];
3   bool visit[MAXLEN];
4
5   void dij(int id){
6       memset(dis, 0x3f, sizeof(dis));
7       memset(visit, 0, sizeof(visit));
```

```
8        dis[id] = 0;
9        priority_queue<pii, vector<pii>, greater<pii>> q;
10
11       //first for 目前距离 second for 点
12       q.push({0, id});
13       while(!q.empty()){
14           pair<int, int> front = q.top(); q.pop();
15           if(visit[front.second]) continue;
16           visit[front.second] = true;
17           for(auto adj : adjs[front.second]){
18               if((adj.second + dis[front.second]) <  dis[adj.first]){
19                   dis[adj.first] = adj.second + dis[front.second];
20                   if(visit[adj.first]) continue;
21                   q.push({dis[adj.first], adj.first});
22               }
23           }
24       }
25   }
```

- **找负环**

```
1  vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 权重
2  int dis[MAXLEN]; //最短路径
3  int m, n; //n 点数 m 边数
4
5  bool bellman(){
6      memset(dis, 0x3f, 4 * n);
7      for(int c = 0; c < n - 1; ++c){
8        for(int i = 0; i < n; ++i){
9          for(auto adj : adjs[i]){
10           dis[adj.first] = min(dis[adj.first], dis[i] + adj.second);
11          }
12        }
13      }
14      bool hasAnswer = false;
15    for(int i = 0; i < n; ++i){
16      for(auto adj : adjs[i]){
17        if(dis[adj.first] > dis[i] + adj.second){
18          hasAnswer = true;
19          goto ans;
20        }
21      }
22    }
23      ans:
24      return hasAnswer;
25  }
```

- **MST**

```
1  // kruskal
2
3  pair<int, pii> adjs[MAXLEN]; //first for 权重 //second.first for 起点 //second.second for 终点
4  int n; //边的数量
5  int m; //点的数量
6  int djs[MAXLEN];
7
8
9  int parent(int i){
10     return (djs[i] == i) ? i : djs[i] = parent(djs[i]);
11 }
12
13 void kruskal(){
14     for(int i = 0; i <= m; ++i){
15         djs[i] = i;
16     }
17
18     sort(adjs, adjs + n);
19     int cnt = m; //计算边的数量
20     for(int i = 0; i < n; ++i){
21         int p1 = parent(adjs[i].second.first), p2 = parent(adj[i].second.second);
22         if(p1 == p2) continue;
23         parent[p1] = p2;
24         //一些操作
25         if((--cnt) == 1) break;
26     }
27 }
28
29 //Prim 普利姆算法（待续）
30
31 void prim(){
```

```
32
33  }
```

- **最大流 (Dinic)**

```
1   struct edge
2   {
3       int to;
4       long long val;
5       int rev;
6       edge(int _to, long long _val, int _rev):to(_to), val(_val), rev(_rev){}
7   };
8
9   int n, m, s, t;
10
11
12  vector<edge> adjs[MAXLEN];
13
14  int depth[MAXLEN];
15
16  bool bfs(){ //分层 + 确定没有回去找
17      memset(depth, 0, sizeof(depth));
18      depth[s] = 1;
19      queue<int> q;
20      q.push(s);
21      while(!q.empty()){
22          int f = q.front(); q.pop();
23          for(edge &adj : adjs[f]){
24              if(!depth[adj.to] && adj.val){
25                  depth[adj.to] = depth[f] + 1;
26                  q.push(adj.to);
27              }
28          }
29      }
30      return depth[t];
31  }
32
33  long long dfs(int u = s, long long in = 0x3f3f3f3f3f3f3f3f){ //多路增广
34      if(u == t) return in;
35      if(in == 0) return 0;
36      long long out = 0;
37      for(edge &adj : adjs[u]){
38          if((depth[adj.to] == (depth[u] + 1)) && adj.val){
39              long long dist = dfs(adj.to, min(in, adj.val));
40              adj.val -= dist;
41              adjs[adj.to][adj.rev].val += dist;
42              in -= dist;
43              out += dist;
44          }
45      }
46      if(!out) depth[u] = 0;
47      return out;
48  }
49
50  long long ans = 0;
51
52  void dinic(){
53      while(bfs()){
54          while(long long d = dfs()){
55              ans += d;
56          }
57      }
58  }
```

- **费用流 (Dinic)**

```
1   struct edge
2   {
3       int to;
4       int val;
5       int rev;
6       int cost;
7       edge(int _to, int _val, int _rev, int _cost):to(_to), val(_val), rev(_rev), cost(_cost){}
8   };
9
10  int n, m, s, t;
11
12  vector<edge> adjs[MAXLEN];
13
14  int dis[MAXLEN];
```

```
15  bool visit[MAXLEN];
16
17
18
19  long long zuidaliu = 0, zuixiaofeiyong = 0;
20
21  bool spfa(int id = t){
22    memset(visit, 0, sizeof(visit));
23    memset(dis, 0x3f, sizeof(dis));
24    visit[id]  = true;
25    dis[id] = 0;
26    deque<int> q;
27    q.push_front(id);
28
29    while(!q.empty()){
30      int front = q.front(); q.pop_front();
31
32      for(edge &adj : adjs[front]){
33        if(adjs[adj.to][adj.rev].val > 0 && dis[adj.to]  > dis[front] - adj.cost ){
34          dis[adj.to] = dis[front] - adj.cost;
35          if(!visit[adj.to]){
36            visit[adj.to] = true;
37            if(!q.empty() && dis[adj.to] < dis[q.front()]) q.push_front(adj.to);
38            else q.push_back(adj.to);
39          }
40        }
41      }
42      visit[front] = false;
43    }
44    return dis[s] != 0x3f3f3f3f;
45  }
46
47  int dfs(int u = s, int flow =  0x3f3f3f3f){
48    if(u == t){
49      visit[u] = true;
50      return flow;
51    }
52    int in = flow;
53    int out = 0;
54    visit[u] = true;
55    for(edge &adj : adjs[u]){
56
57      if(adj.val >0 && !visit[adj.to] && dis[adj.to] == (dis[u] - adj.cost)){
58        int dis = dfs(adj.to, min(in, adj.val));
59        if(dis > 0){
60          in -= dis;
61          adj.val -= dis;
62          out += dis;
63          zuixiaofeiyong += dis * adj.cost;
64          adjs[adj.to][adj.rev].val += dis;
65        }
66        if(!in) break;
67
68      }
69    }
70
71    return out;
72  }
73
74
75
76  void dinic(){
77    while(spfa()){
78      while(int d = dfs()){
79        memset(visit, 0, sizeof(visit));
80        zuidaliu += d;
81      }
82    }
83  }
```

3. 动态规划

- **区间覆盖**

```
1  pii intv[MAXLEN];
2  int n; //区间数量
3  int end; //终点
4
5
```

```cpp
int solve(){
    sort(intv, intv + n);
    int tmp = 0 /* 暂存 */ , r = 0 /* 右边更新 */;
    int sum = 0; //答案
    int i = 0;
    while(i < n && r < end){
        if(intv[i].first > r) break;
        while(intv[i].first <= r  && tmp < end){
            tmp = max(intv[i].second, tmp);
            ++i;
        }
        r = tmp;
        ++sum;
    }
    return r >= end : sum ? 0; // 没有则回传0
}
```

- **LIS**

```cpp
int tmp[MAXLEN]; //暂存
int arr[MAXLEN]; //待处理数组
int dp[MAXLEN]; //各个结尾的值

void LIS(){
    for(int i = 0; i <= n; ++i) tmp[i] = INT_MAX;
    tmp[0] = 0;
    for(int i = 0; i < n; ++i){
        int pos = lower_bound(tmp, tmp + n, arr[i]) - tmp; //lower_bound找< // upper_bound找<=
        tmp[pos] = arr[i], dp[i] = pos;
    }
}
```

4. 字串

- **KMP**

```cpp
int f[MAXLEN]; // failure Function 1-index

void kmp(char s[], char p[]){
  int i = 0, j = -1;
  int n = strlen(s), m = strlen(p);f[0]=-1;
  while(i < m){
    if(j == -1 || p[i] == p[j]){
      f[++i] = ++j;
    }else{
      j = f[j];
    }
  }
  i = 0; j = 0;
  while(i <= n){
    if(j == -1 || s[i] == p[j]){
      ++j, ++i;
      if(j == m){
        //一些操作
      }

    }else{
      j = f[j];
    }
  }
}
```

- **马拉车**

```cpp
char s[MAXLEN]; //字串 记得填充$#^(aaa-> $#a#a#a#^)
int p[MAXLEN]; //回文长度

int len = 0 //s的长度

void manache(){
  int i_r = 0, c = 0, r = 0; //i_r 反射点，c 中心点 r 右端点
  for(int i = 1; i < len; ++i){
    i_r = 2 * c - i;
    if(r > i) p[i] = min(r - i, p[i_r]);
    else p[i] = 0;
    while(s[i + 1 + p[i]] == s[i - 1 - p[i]]) ++p[i];
    if(i + p[i] > r) c = i, r = i + p[i];
  }
  int m = 0; // 长度
```

```c
   c = 0; //中央
   for(int i = 0; i < len - 1; ++i){
      if(p[i] > m) c = i, m = p[i];
   }
   printf("%d\n", m);
}
```