代码簿

August 31, 2021

1. 杂项

- **Merge Sort**

```
int arr[MAXLEN]; // 数组
int tmp[MAXLEN]; // 暂存
int ans  = 0; // 逆序对

void mergeSort(int left, int right){
  if(left >= right) return;
  int mid = left + ((right - left) >> 1);
  mergeSort(left, mid);
  mergeSort(mid + 1, right);
  int l = left;
  int r = mid + 1;
  int idx = 0;
  while(l <= mid && r <= right){
    tmp[idx++] = arr[l] <= arr[r] ? arr[l++]
    : ((ans += mid - l + 1 /* 逆序对 */ ), arr[r++]);
  }
  while(l <= mid){
    tmp[idx++] = arr[l++];
  }
  while(r <= right){
    tmp[idx++] = arr[r++];
  }
  for(int i = 0; i < idx; ++i){
    arr[left + i] = tmp[i];
  }
}
```

- **Disjoint Set**

```
int djs[MAXLEN];
//init for(int i = 0; i < size; ++i)djs[i] = i;

int find(int id){
  return djs[id] == id ? id : djs[id] = find(djs[id]);
}
bool uSet(int a, int b){
  int pa = find(a), pb = find(b);
  return pa == pb ? true : (djs[pa] = pb, false);
}
```

2. 图论

- **Eular Path**

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

int cnt = 0; //判断联通(cnt == size)

void dfs(int id){
    for(auto &adj : adjs){
        if(!visit[adj.second]){
            ++cnt;
            visit[adj.second] = true;
            dfs(adj.first);
            //一些操作
        }
    }
}
```

```

bool judge(){
    //无向(边为偶数除了端点)
    int cnt = 0;
    int qidian  = 0; // 起点
    for(int i = 0; i < n; ++i){
        if(adjs[i].size() & 1) ++cnt, qidian = i;
    }
    return (cnt == 2 || cnt == 0);

    //有向((出度 - 入度) = 1 && (入度 - 出度) = 1 || 欧拉环)
    int rdc = 0, cdc = 0;
    int qidian = 0;
    for(int i = 0; i < n; ++i){
        if((cd[i] - rd[i] == 1)) ++cdc;
        else if((rd[i] - cd[i]) == 1) ++rdc, qidian = i;
        else if((rd[i] - cd[i]) != 0) return false;
    }

    return ((cdc == 1 && rdc == 1) || (cdc == 0 && rdc == 0));
}
```

- **拓扑排序**

```
vector<int> adjs[MAXLEN];
int rd[MAXLEN]; //入度
int n; //点的数量
vector<int> ans; //结果

void topSort(){
    queue<int> q;
    int cnt= 0;
    for(int i = 0; i < n; ++i) if(!rd[i]) q.push(i), ++cnt, ans.push_back(i);
    while(!q.empty()){
        int f = q.front(); q.pop();
        for(auto adj : adjs[f]){
            --rd[adj];
            if(!rd[adj]) q.push(adj), ++cnt, ans.push_back(adj);
        }
    }
    if(cnt == n) {}//有找到
    else {} //没找到
}
```

- **多元最短路**

```
int v[MAXLEN][MAXLEN];
//初始化成0x3f3f3f3f，有边的地方为权重，自己为0

for(int k = 0; k < n; ++k){
    for(int  i = 0; i < n; ++i){
        for(int  j = 0; j < n; ++j){
            v[i][j] = min(v[i][k] + v[k][j], v[i][j]); //最短路径
            v[i][j] = min(v[i][j], max(v[i][k], v[k][j])); //最大的最小
        }
    }
}
```

- **单元最短路**

```
vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 边权
int dis[MAXLEN];
bool visit[MAXLEN];

void dij(int id){
    memset(dis, 0x3f, sizeof(dis));
    memset(visit, 0, sizeof(visit));
    dis[id] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> q;

    //first for 目前距离 second for 点
    q.push({0, id});
    while(!q.empty()){
        pair<int, int> front = q.top(); q.pop();
        if(visit[front.second]) continue;
        visit[front.second] = true;
        for(auto adj : adjs[front.second]){
            if((adj.second + dis[front.second]) <  dis[adj.first]){
                dis[adj.first] = adj.second + dis[front.second];
                if(visit[adj.first]) continue;
                q.push({dis[adj.first], adj.first});
            }
        }
    }
}
```

- **找负环**

```
vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 权重
int dis[MAXLEN]; //最短路径
int m, n; //n 点数 m 边数

bool bellman(){
    memset(dis, 0x3f, 4 * n);
    for(int c = 0; c < n - 1; ++c){
      for(int i = 0; i < n; ++i){
        for(auto adj : adjs[i]){
          dis[adj.first] = min(dis[adj.first], dis[i] + adj.second);
        }
      }
    }
    bool hasAnswer = false;
  for(int i = 0; i < n; ++i){
    for(auto adj : adjs[i]){
      if(dis[adj.first] > dis[i] + adj.second){
        hasAnswer = true;
        goto ans;
      }
    }
  }
    ans:
    return hasAnswer;
}
```

- **MST**

```
// kruskal

pair<int, pii> adjs[MAXLEN]; //first for 权重 //second.first for 起点 //second.second for 终点
int n; //边的数量
int m; //点的数量
int djs[MAXLEN];


int parent(int i){
    return (djs[i] == i) ? i : djs[i] = parent(djs[i]);
}

void kruskal(){
    for(int i = 0; i <= m; ++i){
        djs[i] = i;
    }

    sort(adjs, adjs + n);
    int cnt = m; //计算边的数量
    for(int i = 0; i < n; ++i){
        int p1 = parent(adjs[i].second.first), p2 = parent(adj[i].second.second);
        if(p1 == p2) continue;
        parent[p1] = p2;
        //一些操作
        if((--cnt) == 1) break;
    }
}

//Prim 普利姆算法（待续）

void prim(){

}
```

- **最大流 (Dinic)**

```
struct edge
{
    int to;
    long long val;
    int rev;
    edge(int _to, long long _val, int _rev):to(_to), val(_val), rev(_rev){}
};

```

```cpp
int n, m, s, t;


vector<edge> adjs[MAXLEN];

int depth[MAXLEN];

bool bfs(){ //分层 + 确定没有回去找
    memset(depth, 0, sizeof(depth));
    depth[s] = 1;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int f = q.front(); q.pop();
        for(edge &adj : adjs[f]){
            if(!depth[adj.to] && adj.val){
                depth[adj.to] = depth[f] + 1;
                q.push(adj.to);
            }
        }
    }
    return depth[t];
}

long long dfs(int u = s, long long in = 0x3f3f3f3f3f3f3f3f){ //多路增广
    if(u == t) return in;
    if(in == 0) return 0;
    long long out = 0;
    for(edge &adj : adjs[u]){
        if((depth[adj.to] == (depth[u] + 1)) && adj.val){
            long long dist = dfs(adj.to, min(in, adj.val));
            adj.val -= dist;
            adjs[adj.to][adj.rev].val += dist;
            in -= dist;
            out += dist;
        }
    }
    if(!out) depth[u] = 0;
    return out;
}

long long ans = 0;

void dinic(){
    while(bfs()){
        while(long long d = dfs()){
            ans += d;
        }
    }
}
```

- **费用流 (Dinic)**

```cpp
struct edge
{
    int to;
    int val;
    int rev;
    int cost;
    edge(int _to, int _val, int _rev, int _cost):to(_to), val(_val), rev(_rev), cost(_cost){}
};

int n, m, s, t;

vector<edge> adjs[MAXLEN];

int dis[MAXLEN];
bool visit[MAXLEN];



long long zuidaliu = 0, zuixiaofeiyong = 0;

bool spfa(int id = t){
  memset(visit, 0, sizeof(visit));
  memset(dis, 0x3f, sizeof(dis));
  visit[id]  = true;
  dis[id] = 0;
  deque<int> q;
  q.push_front(id);

  while(!q.empty()){
    int front = q.front(); q.pop_front();

    for(edge &adj : adjs[front]){
      if(adjs[adj.to][adj.rev].val > 0 && dis[adj.to]  > dis[front] - adj.cost ){
        dis[adj.to] = dis[front] - adj.cost;
        if(!visit[adj.to]){
          visit[adj.to] = true;
          if(!q.empty() && dis[adj.to] < dis[q.front()]) q.push_front(adj.to);
          else q.push_back(adj.to);
        }
      }
    }
    visit[front] = false;
  }
  return dis[s] != 0x3f3f3f3f;
}

int dfs(int u = s, int flow =  0x3f3f3f3f){
  if(u == t){
    visit[u] = true;
    return flow;
  }
  int in = flow;
  int out = 0;
  visit[u] = true;
  for(edge &adj : adjs[u]){

    if(adj.val >0 && !visit[adj.to] && dis[adj.to] == (dis[u] - adj.cost)){
      int dis = dfs(adj.to, min(in, adj.val));
      if(dis > 0){
        in -= dis;
        adj.val -= dis;
        out += dis;
        zuixiaofeiyong += dis * adj.cost;
        adjs[adj.to][adj.rev].val += dis;
      }
      if(!in) break;

    }
  }

  return out;
}



void dinic(){
  while(spfa()){
    while(int d = dfs()){
      memset(visit, 0, sizeof(visit));
      zuidaliu += d;
    }
  }
}
```

3. 动态规划

- **区间覆盖**

```
pii intv[MAXLEN];
int n; //区间数量
int end; //终点


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**

```
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;
    }
}
```