数据结构 第三次作业报告

2022013014 黄泽文 未央-软件21

1. 无线广播

分析

给图上的所有节点染两种颜色之一，要求任意一条边连接的两点颜色不同。显然只有当图中存在奇环时，才可能出现无解的情况。对每个联通块搜索分析即可。

具体地说，对每个遍历过的节点标记 vis。如果搜索中遇到的节点没有访问过，则染上与其搜索树父亲相反的颜色；如果访问过，说明这是一个环，判断其是否与父亲颜色相反即可。

代码

```
bool dfs(int head, bool color) {
      for (int i = 1; i <= n; i++) {
            if (i == head)continue;
            if (map[head][i]) {
                  if (vis[i]) {
                        if (value[i] == color) {
                              return false;
                        }
                  } else {
                        vis[i] = true;
                        value[i] = !color;
                        if (!dfs(i, !color)) {
                              return false;
                        }
                  }
            }
      }
       return true;
}
```

2. 旅行商

分析

村庄 A 必定是一个入度为 0 的节点（否则我们选取其父节点为 A 能够得到更优的结果）。从每个入度为0的节点开始搜索，记录最长路径即可。为了避免大量重复搜索，可以维护一个数组，记录从每个节点出发可以得到的最长路径长度。本题的数据规模比较大，不能使用邻接矩阵存图，改用链式前向星即可。

代码

```

struct edge{
      int u, v;
      edge *next;
}*map[MAXN];
```

```
bool notHead[MAXN];
int n, m;
int ansS[MAXN];
int topo(int h) {
      if(ansS[h])return ansS[h];
      int ans = 1;
      if(map[h] == nullptr)return ans;
      for(auto i = map[h]; i; i = i->next){
            ans = max(ans, 1 + topo(i->v));
      }
      ansS[h] = ans;
      return ans;
}

int main() {
      cin >> n >> m;
      for(int i = 0; i <= n; i++)map[i] = nullptr;
      for (int i = 0; i < m; i++) {
            int a, b;
            scanf("%d%d", &a, &b);
            edge *e = new edge{a, b, map[a]};
            map[a] = e;
            notHead[b] = true;
      }
      int ans = 0;
      for (int i = 1; i <= n; i++) {
            if (!notHead[i]) {
                  ans = max(ans, topo(i));
                }
      }
      cout << ans;
}
```

3. 平均气温

分析

本题可以用 2D Tree 解决。

建树 将整个地图视为根节点，以中间节点为界左右对半切分得到两个子节点，再以各自中间节点为界上下对半切分得到各两个子节点，再各自左右对半切分，以此类推。对于每个节点，维护其区域内所有站点的上下左右边界（注意这个边界可能在对半切分操作所得的边界内部，维护前者可以在一些情况下优化查询效率）、站点数、所有站点的温度总和。在具体实现中，如果对数组进行排序，则可以直接取左右半边递归建树，但本题的数据规模下排序会超时；因而更适合只进行快速选择，选出中间元素后逐个判断区域内元素大小，构造新的数组建树，这种做法可以省下 $O(logn)$ 的时间。

查询 对于每个 `query(node *p, rect range)`，返回 p 的区域与查询区域的 *交集* 的站点个数和总温度。可以做分类讨论：

1. 当 p->range 与 range 无交集时，返回 {0, 0}；
2. 当 p->range 含于 range 时，返回 p 维护的值；
3. 当 range 含于 p->left->range 时，返回 `query(p->left, range)`；
4. 当 range 含于 p->right->range 时，返回 `query(p->right, range)`；
5. 否则返回 `query(p->left, range) + query(p->right, range)`。

这个分类讨论包括了比较充分的剪枝，可以通过题目的数据规模。

代码

```
#define ll long long

struct station {
	ll x, y, temp;
} ma[MAXN];

struct node {
	ll x1, y1, x2, y2;
	ll temp;
	int num;
	node *left, *right;
};

struct pairt {
	ll temptot;
	int num;
};

bool rectcontain(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {//rect 34 in 12
	return x1 <= x3 && x2 >= x4 && y1 <= y3 && y2 >= y4;
}

bool rectcross(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {//rect 12 cross 34
	return x1 <= x4 && x2 >= x3 && y1 <= y4 && y2 >= y3;
}

int qsx(station array[], int low, int high, int k) {
	int pivot = array[high].x;
	int i = low - 1;
	for (int j = low; j <= high - 1; j++) {
		if (array[j].x <= pivot) {
			i++;
			std::swap(array[i], array[j]);
		}
	}
	swap(array[i + 1], array[high]);
	int partitionIndex = i + 1;
	if (low <= high) {
		if (partitionIndex == k) {
			return partitionIndex;
		} else if (partitionIndex < k) {
			return qsx(array, partitionIndex + 1, high, k);
		} else {
			return qsx(array, low, partitionIndex - 1, k);
		}
	}

}

int qsy(station array[], int low, int high, int k) {
	int pivot = array[high].y;
```

```
        int i = low - 1;
        for (int j = low; j <= high - 1; j++) {
            if (array[j].y <= pivot) {
                i++;
                std::swap(array[i], array[j]);
            }
        }
        swap(array[i + 1], array[high]);
        int partitionIndex = i + 1;
        if (low <= high) {
            if (partitionIndex == k) {
                return partitionIndex;
            } else if (partitionIndex < k) {
                return qsy(array, partitionIndex + 1, high, k);
            } else {
                return qsy(array, low, partitionIndex - 1, k);
            }
        }
    }


    void buildTree(node *p, bool dir, station array[], int size) {
        //build a kd-tree
        if (size == 0)return;
        if (size == 1) {
            p->x1 = p->x2 = array->x;
            p->y1 = p->y2 = array->y;
            p->temp = array->temp;
            p->num = 1;
            return;
        }
        ll temp = 0;
        p->x1 = p->x2 = array->x;
        p->y1 = p->y2 = array->y;
        for (station *i = array; i < array + size; i++) {
            temp += i->temp;
            p->x1 = min(p->x1, i->x);
            p->x2 = max(p->x2, i->x);
            p->y1 = min(p->y1, i->y);
            p->y2 = max(p->y2, i->y);
        }
        p->num = size;
        p->temp = temp;
        if (dir) {
            //x
            int mid = qsx(array, 0, size - 1, size / 2 - 1), cntl = 0, cntr = 0;
            station ls[size], rs[size];
            for (int i = 0; i < size; i++) {
                if (array[i].x <= array[mid].x)ls[cntl++] = array[i];
                else rs[cntr++] = array[i];
            }
            p->left = new node;
            buildTree(p->left, !dir, ls, cntl);
            p->right = new node;
            buildTree(p->right, !dir, rs, cntr);
        } else {
```

```
108             //y
109             int mid = qsy(array, 0, size - 1, size / 2 - 1), cntl = 0, cntr = 0;
110             station ls[size], rs[size];
111             for (int i = 0; i < size; i++) {
112                   if (array[i].y <= array[mid].y)ls[cntl++] = array[i];
113                   else rs[cntr++] = array[i];
114             }
115             p->left = new node;
116             buildTree(p->left, !dir, ls, cntl);
117             p->right = new node;
118             buildTree(p->right, !dir, rs, cntr);
119       }
120 }
121
122 pair query(node *p, int x1, int y1, int x2, int y2) {
123       if (p->num == 0)return {0, 0};
124       if (rectcontain(x1, y1, x2, y2, p->x1, p->y1, p->x2, p->y2)) {
125             return {p->temp, p->num};
126       }
127       if (!rectcross(x1, y1, x2, y2, p->x1, p->y1, p->x2, p->y2))return {0, 0};
128       if (rectcontain(p->left->x1, p->left->y1, p->left->x2, p->left->y2, x1, y1, x2, y2)) {
129             return query(p->left, x1, y1, x2, y2);
130       }
131       if (rectcontain(p->right->x1, p->right->y1, p->right->x2, p->right->y2, x1, y1, x2, y2)) {
132             return query(p->right, x1, y1, x2, y2);
133       }
134       auto q1 = query(p->left, x1, y1, x2, y2);
135       auto q2 = query(p->right, x1, y1, x2, y2);
136       return {q1.temptot + q2.temptot, q1.num + q2.num};
137 }
138
139 int main() {
140       n = GetNumOfStation();
141       for (int i = 0; i < n; i++) {
142             int x, y, temp;
143             GetStationInfo(i, &x, &y, &temp);
144             ma[i] = station{x, y, temp};
145       }
146       node root;
147       buildTree(&root, true, ma, n);
148       int x1, y1, x2, y2;
149       while (GetQuery(&x1, &y1, &x2, &y2)) {
150             auto q = query(&root, x1, y1, x2, y2);
151             if (q.num)Response(q.temptot / q.num);
152             else Response(0);
153       }
154 }
```