Experiment 5

PB17111623

范睿

EX5-1 道路规划

解题思路:

题目的本质是求得给定图的最小生成树。利用Krustal算法，每次选择不成环的权值最小的边加入生成树，直到所有结点都在生成树中。判断不成环的方法是使用不相交集合，对每个结点建立一个不相交集合，若取出的边的两个顶点是同一个不想交集合中的，说明此两边若加入，必成环。

算法实现:

```
for i = 1 to n
    Make_Set(i);
for i = 1 to m
    cin >> a >> b >>c;
    e = {a,b,c};
    queue.push(e); //queue是最小优先队列
sum = 0;
while(!queue.empty())
    e = queue.pop()
    if(Find_Set(e.u) != Find_Set(e.v))
        sum += e.w;
        Union(e.u, e.v);
cout << sum;
```

复杂度分析:

Make_Set用$\mathcal{O}(n)$，读入所有边，处理所有边共需要$\mathcal{O}(m)$；Find_Set单次操作需要$\mathcal{O}(1)$，共需2m次操作；Union时间为$\mathcal{O}(\lg n)$，所以算法的总时间为$m\lg n$

EX5-2 逃离迷宫

解题思路:

题目本质是求的s到t的最短路径。利用Dijkstra算法，找到从s所有点的最短路径，最后输出t的最短路径。利用堆来找到每次距离s最近的点。

算法:

```
G[1...n] is a new vertex set;
H is a heap;
for i = 1 to n
    G[i].d = inf;
    G[i].firstarc = NULL;
    H.array[i] = i;
for i = 1 to m
    cin >> a >> b >> c;
    addedge(&G, a, b, c);
```

```
HeapSortByCost(&H, G);
for i = 1 to n
    v = H.array[0];
    exchange(&H, 0, H.last);
    Down(&H, 0);
    p = G[v].firstarc;
    while(p){
        relax(&G[v], &G[p->v], p->cost);
        MimHeapifySum(&H, Parent(G[p->v].indexinH));
        p = p->nextarc;
    }
if(G[t].d < 1000000) cout << G[t].d;
else cout << -1;
```

复杂度分析:

初始化$\mathcal{O}(n+m)$，Heapsort$\mathcal{O}(n\lg n)$，dijkstra操作$\mathcal{O}(n\lg n+m\lg n)$，总时间为$\mathcal{O}(n\lg n+m)$

EX5-3 货物运输

解题思路:

此题本质是求s到t的最大流的问题。利用spfa算法计算s到t的最大流。每次寻找一条从s到t可到达的路径，在总流中加上此路径上的流。

算法:

```
G[1...n] is a new graph;
for i = 1 to m
    cin >> a >> b >> c;
    addedge(G, a, b, c, 0);  //flow=0
cost = 0;
flow = 0;
while(1){
    d[1...n] = {INF...INF};
    in[1...n] = {0...0};
    p[s] = -1;
    a[s] = INF;
    d[s] = 0;
    in[s] = 1;
    Q.push(s); //Q is a queue
    while(!Q.empty()){
        v = Q.pop();
        for i = 1 to v.adjnum
            e = edged[G[v][i]];
            if(e.capacity > e.flow && d[e.to] > d[v]){
                d[e.to] = d[v];
                p[e.to] = G[v][i];
                a[e.to] = min(a[v], e.capcacity-e.flow);
                if(!in[e.to]){
                    Q.push(e.to);
                    in[e.to] = 1;
                }
            }
    }
    if(d[t] == INF) break;
    flow += a[t];
```

```
    cost += d[t]*a[t];
    for(u = t;u!=s;u = edges[p[u]].from){
        edges[p[u]^1].flow -= a[t];
        edges[p[u]].flow += a[t];
    }
}
if(d[t] == INF) cout << -1;
else cout << flow;

```

复杂度分析

内层循环最多执行n次，flow增加最多执行m次，外层循环最多执行|f*|次，所以算法总时间为 $\mathcal{O}(|f^*|(n+m))$