

Buying Sets tsD14729

计 72 陈嘉杰

May 11, 2018

Contents

1	题目说明	1
2	实现思路	1
3	程序编译环境	2
4	实现步骤	2
4.1	数据读入	2
4.2	完美匹配	2
4.3	图论建模	3
4.4	网络流 Dinic 算法	4
4.5	完整代码	6
5	遇到的问题和得到的收获	10

1 题目说明

输入 n 个集合，每个集合中有一定个数的元素，元素都为 1 到 n 之间的整数。每个集合都有一定的权值，现在需要选取一组集合，使得这组集合元素的并的元素个数等于选取的集合的个数，并且权值和最小。

2 实现思路

这个题目有个特殊的条件：任意 k 个集合的并的元素不少于 k 。也就是说，无论是哪些集合拿出来，元素都不少于集合的个数。于是，我们可以构造一个元素到集合的双射，跑一个完美匹配的算法即可。然后，根据这个，我们就知道，如果我们要选某一个集合，就一定要把这个集合的元素对应的集合

都选取进来，否则不可能达到要求。将这个要求进行图论建模，刚好对应与最小闭合子图的问题，而最小闭合子图类似于最大闭合子图，可以构造网络流用网络流的方法解决。

3 程序编译环境

1. 操作系统：macOS
2. 编译器：LLVM/Clang 6.0.0

4 实现步骤

4.1 数据读入

```
1 scanf("%d", &n);
2 for (int i = 1; i <= n; i++) {
3     scanf("%d", &num_edges[i]);
4     for (int j = 0; j < num_edges[i]; j++) {
5         scanf("%d", &edges[i][j]);
6     }
7 }
8 for (int i = 1; i <= n; i++) {
9     scanf("%d", &weight[i]);
10    // Find minimum instead of maximum
11    weight[i] = -weight[i];
12 }
```

首先读入每个集合的元素，和权值。因为我们后面把最小权闭合子图转化为最大权闭合子图来做，于是我们需要把权值取相反数。

4.2 完美匹配

```
1 // Perfect Matching
2 for (int i = 1; i <= n; i++) {
3     memset(visit, 0, sizeof(visit));
4     match(i);
5 }
```

主函数中对每个元素调用 matching 函数，不断查找增广路。

```

1 bool match(int cur) {
2     for (int i = 0; i < num_edges[cur]; i++) {
3         int another = edges[cur][i];
4         if (visit[another] == 0) {
5             visit[another] = 1;
6             if (matching[another] == 0 || match(matching[another]))
7                 ↪ {
8                     matching[another] = cur;
9                     return true;
10                }
11        }
12    }
13    return false;
14 }

```

这里通过一个 DFS 找增广路并且更新当前匹配。matching[num] 代表数字 num 对应的是哪个集合。

4.3 图论建模

接下来，我们构造最大闭合子图对应的网络流：正权点从源点连入，负权边向汇点连出，把前面提到的依赖关系通过一条边把点连起来。

```

1 int positive_weight = 0;
2 for (int i = 1; i <= n; i++) {
3     for (int j = 0; j < num_edges[i]; j++) {
4         // if i is covered, then
5         // all numbers in i should be covered,
6         // link those corresponding sets
7         // if (matching[edges[i][j]] != i)
8         add_edge(i, matching[edges[i][j]], INF);
9     }
10 }
11 for (int i = 1; i <= n; i++) {
12     if (weight[i] < 0) {
13         // link to sink
14         add_edge(i, n + 1, -weight[i]);
15     } else {
16         add_edge(0, i, weight[i]);

```

```

17     positive_weight += weight[i];
18 }
19 }

```

这里的 `add_edge` 采用了网络流的 `residue` 表示方法和下标实现边的链表的方法：

```

1 void add_edge(int from, int to, int cap) {
2     edges_flow[++top] = edge{to, top_edges_flow[from], cap};
3     top_edges_flow[from] = top;
4     edges_flow[++top] = edge{from, top_edges_flow[to], 0};
5     top_edges_flow[to] = top;
6 }

```

其中 `top` 表示当前的边数，`top_edges_flow` 表示该结点最后一条边的下标，正向边的余量就是 `cap`，反向边的余量就是 0。正向边和反向边可以通过改变最低位完成。

4.4 网络流 Dinic 算法

最后，在建立的图上跑 Dinic 算法。首先是对图进行 bfs：

```

1 bool bfs() {
2     for (int i = 0; i <= n + 1; i++) {
3         depth[i] = -1;
4     }
5     std::queue<int> que;
6     que.push(0);
7     depth[0] = 0;
8     while (!que.empty()) {
9         int current = que.front();
10        que.pop();
11        for (int i = top_edges_flow[current]; i != 0; i =
            ↪ edges_flow[i].next_edge) {
12            int next = edges_flow[i].to;
13            if (edges_flow[i].residue > 0 && depth[next] < 0) {
14                depth[next] = depth[current] + 1;
15                que.push(next);
16            }
17        }
18    }
19 }

```

```

19 return depth[n + 1] > 0;
20 }

```

同时检测汇点不可达的情况。然后根据得到的 depth 数组进行增广路的寻找：

```

1 int dfs(int current, int to, int current_flow) {
2     if (current == to || current_flow == 0) {
3         return current_flow;
4     }
5
6     int flow = 0;
7     for (int i = top_edges_flow[current]; i != 0; i =
        ↪ edges_flow[i].next_edge) {
8         int next = edges_flow[i].to;
9         if (edges_flow[i].residue > 0 && depth[next] ==
        ↪ depth[current] + 1) {
10             int result =
11                 dfs(next, to, min(edges_flow[i].residue, current_flow
        ↪ - flow));
12             if (result) {
13                 flow += result;
14                 edges_flow[i].residue -= result;
15                 edges_flow[i ^ 1].residue += result;
16                 if (flow == current_flow) {
17                     return flow;
18                 }
19             }
20         }
21     }
22     if (flow == 0) {
23         depth[current] = -1;
24     }
25     return flow;
26 }

```

最后，在 main 中多次循环，并且最后输出最大闭合子图的结果：

```

1 int max_flow = 0;
2 while (bfs()) {
3     max_flow += dfs(0, n + 1, INF);

```

```

4 }
5 printf("%d\n", max_flow - positive_weight);

```

4.5 完整代码

```

1 #include <memory.h>
2 #include <queue>
3 #include <stdio.h>
4 #include <string.h>
5
6 const static int INF = 1 << 30;
7
8 int n;
9 // 1~n: set
10 int num_edges[700] = {0};
11 int edges[700][700] = {{0}};
12 int matching[700] = {0};
13 int visit[700] = {0};
14
15 // 0: source
16 // n+1: sink
17 struct edge {
18     int to;
19     int next_edge;
20     int residue;
21 } edges_flow[500 * 500 * 2];
22 int top = 1;
23 int top_edges_flow[700] = {0};
24
25 int weight[500] = {0};
26 int depth[500] = {0};
27 int map_set[500] = {0};
28
29 inline int min(int a, int b) { return a > b ? b : a; }
30
31 void add_edge(int from, int to, int cap) {
32     edges_flow[++top] = edge{to, top_edges_flow[from], cap};
33     top_edges_flow[from] = top;
34     edges_flow[++top] = edge{from, top_edges_flow[to], 0};
35     top_edges_flow[to] = top;

```

```

36 }
37
38 bool match(int cur) {
39     for (int i = 0; i < num_edges[cur]; i++) {
40         int another = edges[cur][i];
41         if (visit[another] == 0) {
42             visit[another] = 1;
43             if (matching[another] == 0 || match(matching[another]))
44                 ↪ {
45                     matching[another] = cur;
46                     return true;
47                 }
48         }
49     }
50     return false;
51 }
52
53 bool bfs() {
54     for (int i = 0; i <= n + 1; i++) {
55         depth[i] = -1;
56     }
57     std::queue<int> que;
58     que.push(0);
59     depth[0] = 0;
60     while (!que.empty()) {
61         int current = que.front();
62         que.pop();
63         for (int i = top_edges_flow[current]; i != 0; i =
64             ↪ edges_flow[i].next_edge) {
65             int next = edges_flow[i].to;
66             if (edges_flow[i].residue > 0 && depth[next] < 0) {
67                 depth[next] = depth[current] + 1;
68                 que.push(next);
69             }
70         }
71     }
72     return depth[n + 1] > 0;
73 }

```

```

74 int dfs(int current, int to, int current_flow) {
75     if (current == to || current_flow == 0) {
76         return current_flow;
77     }
78
79     int flow = 0;
80     for (int i = top_edges_flow[current]; i != 0; i =
        ↪ edges_flow[i].next_edge) {
81         int next = edges_flow[i].to;
82         if (edges_flow[i].residue > 0 && depth[next] ==
            ↪ depth[current] + 1) {
83             int result =
84                 dfs(next, to, min(edges_flow[i].residue, current_flow
                    ↪ - flow));
85             if (result) {
86                 flow += result;
87                 edges_flow[i].residue -= result;
88                 edges_flow[i ^ 1].residue += result;
89                 if (flow == current_flow) {
90                     return flow;
91                 }
92             }
93         }
94     }
95     if (flow == 0) {
96         depth[current] = -1;
97     }
98     return flow;
99 }
100
101 int main() {
102     scanf("%d", &n);
103     for (int i = 1; i <= n; i++) {
104         scanf("%d", &num_edges[i]);
105         for (int j = 0; j < num_edges[i]; j++) {
106             scanf("%d", &edges[i][j]);
107         }
108     }
109     for (int i = 1; i <= n; i++) {
110         scanf("%d", &weight[i]);

```



```

111     // Find minimum instead of maximum
112     weight[i] = -weight[i];
113 }
114 // Perfect Matching
115 for (int i = 1; i <= n; i++) {
116     memset(visit, 0, sizeof(visit));
117     match(i);
118 }
119 // Maximum flow
120 int positive_weight = 0;
121 for (int i = 1; i <= n; i++) {
122     for (int j = 0; j < num_edges[i]; j++) {
123         // if i is covered, then
124         // all numbers in i should be covered,
125         // link those corresponding sets
126         // if (matching[edges[i][j]] != i)
127         add_edge(i, matching[edges[i][j]], INF);
128     }
129 }
130 for (int i = 1; i <= n; i++) {
131     if (weight[i] < 0) {
132         // link to sink
133         add_edge(i, n + 1, -weight[i]);
134     } else {
135         add_edge(0, i, weight[i]);
136         positive_weight += weight[i];
137     }
138 }
139 int max_flow = 0;
140 while (bfs()) {
141     max_flow += dfs(0, n + 1, INF);
142 }
143 printf("%d\n", max_flow - positive_weight);
144 return 0;
145 }

```

5 遇到的问题和得到的收获

遇到的问题是，首先在编写完美匹配的代码的时候，记错了这个算法的一个细节，导致调试了很久。第二个就是，如果要通过异或来得到反向边的下标，需要注意加入边时下标是否满足这个。写代码就是这样，总会在意想不到的地方出现自己的失误。