Buying Sets tsD14729

计 72 陈嘉杰

May 11, 2018

Contents

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的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

```
scanf("%d", &n);
for (int i = 1; i <= n; i++) {
  scanf("%d", &num_edges[i]);
  for (int j = 0; j < num_edges[i]; j++) {
    scanf("%d", &edges[i][j]);
  }
 }
for (int i = 1; i <= n; i++) {
  scanf("%d", &weight[i]);
  // Find minimum instead of maximum
  weight[i] = -weight[i];
 }
```

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

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

主函数中对每个元素调用 `matching` 函数，不断查找增广路。

```cpp
bool match(int cur) {
  for (int i = 0; i < num_edges[cur]; i++) {
    int another = edges[cur][i];
    if (visit[another] == 0) {
      visit[another] = 1;
      if (matching[another] == 0 || match(matching[another]))
      ↪ {
        matching[another] = cur;
        return true;
      }
    }
  }

  return false;
}
```

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。

```cpp
int positive_weight = 0;
for (int i = 1; i <= n; i++) {
  for (int j = 0; j < num_edges[i]; j++) {
    // if i is covered, then
    // all numbers in i should be covered,
    // link those corresponding sets
    // if (matching[edges[i][j]] != i)
    add_edge(i, matching[edges[i][j]], INF);
  }
 }
for (int i = 1; i <= n; i++) {
  if (weight[i] < 0) {
    // link to sink
    add_edge(i, n + 1, -weight[i]);
  } else {
    add_edge(0, i, weight[i]);
```

```
    positive_weight += weight[i];
  }
 }
```

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：

```
void add_edge(int from, int to, int cap) {
  edges_flow[++top] = edge{to, top_edges_flow[from], cap};
  top_edges_flow[from] = top;
  edges_flow[++top] = edge{from, top_edges_flow[to], 0};
  top_edges_flow[to] = top;
}
```

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 ：

```
bool bfs() {
  for (int i = 0; i <= n + 1; i++) {
    depth[i] = -1;
  }
  std::queue<int> que;
  que.push(0);
  depth[0] = 0;
  while (!que.empty()) {
    int current = que.front();
    que.pop();
    for (int i = top_edges_flow[current]; i != 0; i =
    ↪  edges_flow[i].next_edge) {
      int next = edges_flow[i].to;
      if (edges_flow[i].residue > 0 && depth[next] < 0) {
        depth[next] = depth[current] + 1;
        que.push(next);
      }
    }
  }
```

```
  return depth[n + 1] > 0;
}
```

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：

```
int dfs(int current, int to, int current_flow) {
  if (current == to || current_flow == 0) {
    return current_flow;
  }

  int flow = 0;
  for (int i = top_edges_flow[current]; i != 0; i =
  ↪  edges_flow[i].next_edge) {
    int next = edges_flow[i].to;
    if (edges_flow[i].residue > 0 && depth[next] ==
    ↪  depth[current] + 1) {
      int result =
        dfs(next, to, min(edges_flow[i].residue, current_flow
        ↪  - flow));
      if (result) {
        flow += result;
        edges_flow[i].residue -= result;
        edges_flow[i ^ 1].residue += result;
        if (flow == current_flow) {
          return flow;
        }
      }
    }
  }
  if (flow == 0) {
    depth[current] = -1;
  }
  return flow;
}
```

最后，在 main 中多次循环，并且最后输出最大闭合子图的结果：

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

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

4.5 完整代码

```
#include <memory.h>
#include <queue>
#include <stdio.h>
#include <string.h>

const static int INF = 1 << 30;

int n;
// 1~n: set
int num_edges[700] = {0};
int edges[700][700] = {{0}};
int matching[700] = {0};
int visit[700] = {0};

// 0: source
// n+1: sink
struct edge {
  int to;
  int next_edge;
  int residue;
} edges_flow[500 * 500 * 2];
int top = 1;
int top_edges_flow[700] = {0};

int weight[500] = {0};
int depth[500] = {0};
int map_set[500] = {0};

inline int min(int a, int b) { return a > b ? b : a; }

void add_edge(int from, int to, int cap) {
  edges_flow[++top] = edge{to, top_edges_flow[from], cap};
  top_edges_flow[from] = top;
  edges_flow[++top] = edge{from, top_edges_flow[to], 0};
  top_edges_flow[to] = top;
```

```
}

bool match(int cur) {
  for (int i = 0; i < num_edges[cur]; i++) {
    int another = edges[cur][i];
    if (visit[another] == 0) {
      visit[another] = 1;
      if (matching[another] == 0 || match(matching[another]))
      ↪  {
        matching[another] = cur;
        return true;
      }
    }
  }

  return false;
}

bool bfs() {
  for (int i = 0; i <= n + 1; i++) {
    depth[i] = -1;
  }
  std::queue<int> que;
  que.push(0);
  depth[0] = 0;
  while (!que.empty()) {
    int current = que.front();
    que.pop();
    for (int i = top_edges_flow[current]; i != 0; i =
    ↪  edges_flow[i].next_edge) {
      int next = edges_flow[i].to;
      if (edges_flow[i].residue > 0 && depth[next] < 0) {
        depth[next] = depth[current] + 1;
        que.push(next);
      }
    }
  }
  return depth[n + 1] > 0;
}

```

```
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]);
```

```
    // Find minimum instead of maximum
    weight[i] = -weight[i];
  }
  // Perfect Matching
  for (int i = 1; i <= n; i++) {
    memset(visit, 0, sizeof(visit));
    match(i);
  }
  // Maximum flow
  int positive_weight = 0;
  for (int i = 1; i <= n; i++) {
    for (int j = 0; j < num_edges[i]; j++) {
      // if i is covered, then
      // all numbers in i should be covered,
      // link those corresponding sets
      // if (matching[edges[i][j]] != i)
      add_edge(i, matching[edges[i][j]], INF);
    }
  }
  for (int i = 1; i <= n; i++) {
    if (weight[i] < 0) {
      // link to sink
      add_edge(i, n + 1, -weight[i]);
    } else {
      add_edge(0, i, weight[i]);
      positive_weight += weight[i];
    }
  }
  int max_flow = 0;
  while (bfs()) {
    max_flow += dfs(0, n + 1, INF);
  }
  printf("%d\n", max_flow - positive_weight);
  return 0;
}
```

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。