Homework 1

1 Tiling Problem

将问题转化为二分图匹配问题，使用匈牙利算法，可以在$O(n^3)$的时间复杂度内得到结果。

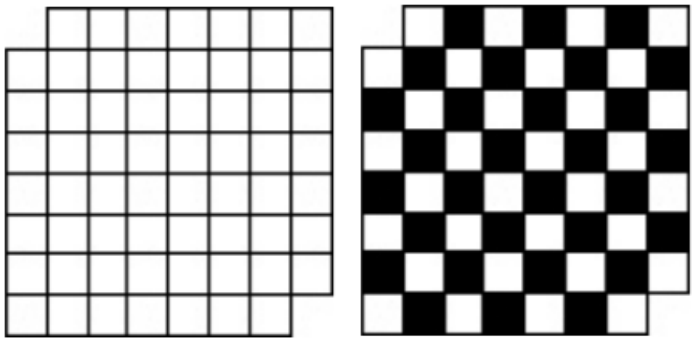
a) 伪代码

```
main()
  (L,R)= getInput() //L 和 R 是二分图的左右两部分
  check is vector //check 标记正在遍历的增广路所使用的节点
  Matching is vector//Matching[i] = j  表示 i 个节点将与 j 节点配对
  for(each node in L)
    if(node is no Matching
        reset(check, 0)
        dfs(node)
        num++
  if(num != L.size())
    return false
  else
    return true


dfs(u)
  for(i : u.neighbor)
     if(check(i) == false)
        check[i] = true
        if(matching[i] don't exist || dfs(matching[i])
              matching[i] = u
              matching[u] = i
              return true
   return false
```

b) 样例测试

i.



```
8 8 2
0 0 7 7
false
```

第一行输入：8　8　2 表示一个 8 * 8 的矩阵缺少两块。

接下来输入缺失的”块”，每两个整数为一组，表示第 i 行第 j 列的小块缺失。

输入：false 表示无法完成覆盖。

ii. 上课的例子

```
  # # #     # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
  # # #     # # #
```

```
5 10 8
0 0
0 4
0 5
0 9
4 0
4 4
4 5
4 9
false
```

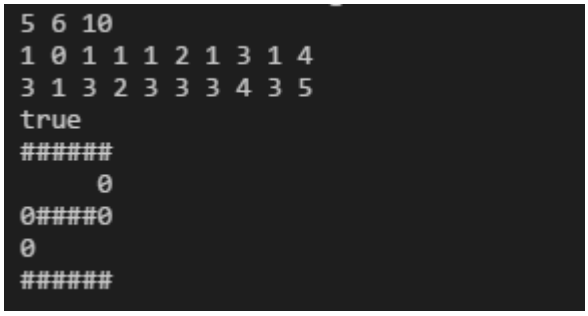
第一行输入：5　10　8 表示一个 5 * 10 的矩阵缺少 8 块。

接下来输入缺失的”块”，每两个整数为一组，表示第 i 行第 j 列的小块缺失，共八行。

输入：false 表示无法完成覆盖。

iii.

```
# # # # # #
          #
# # # # # #
#
# # # # # #
```



输入 5 * 6 缺少十个位置的棋盘，输出了可行解。

“#”表示横着放，”0”表示竖着放。

c) 代码

```
1.  #include <iostream>
2.  #include <cstring>
3.  #include <string>
4.  #include <vector>
5.  //二分图匹配问题，匈牙利算法，增广路定理
6.  using namespace std;
7.  int* Matching;
8.  int* check;
9.  int **G;
10.
11. vector<int> L;
12. vector<int> R;
13.
14. bool findMaxMatching();
15. bool dfs(int u);
16.
17. int m, n, N, a, b;
18. int main() {
19.  cin >> m >> n >> N;
20.  int totalNode = 0;
21.  totalNode = m * n - N;
22.  int *noNode;
23.  noNode = new int[m * n];
24.  memset(noNode, 0, m * n * sizeof(int));
25.  vector<string> res(m, string(n, ' '));
26.
27.  while (N > 0) {
28.          cin >> a >> b;
29.          noNode[a * n + b] = 1;
30.          N--;
31.  }
32.  G = new int*[m * n];
```

```
33.
34.  for (int i = 0; i < m * n; ++i) {
35.          G[i] = new int[4];
36.          memset(G[i], -1, 4 * sizeof(int));
37.  }
38.
39.  Matching = new int[m * n];
40.  memset(Matching, -1, m * n * sizeof(int));
41.
42.  for (int i = 0; i < m; ++i) {
43.          for (int j = 0; j < n; ++j) {
44.                  if (noNode[i * n + j] == 0) {
45.                          if ((i + j) % 2 == 0) {
46.                                  L.push_back(i * n + j);
47.                          }
48.                          else {
49.                                  R.push_back(i * n + j);
50.                          }
51.                          // top
52.                          if (i != 0 && noNode[i * n + j - n] == 0) {
53.                                  G[i * n + j][0] = i * n + j - n;
54.                          }
55.                          // down
56.                          if (i != n - 1 && noNode[i * n + j + n] == 0) {
57.                                  G[i * n + j][1] = i * n + j + n;
58.                          }
59.                          // left
60.                          if (j != 0 && noNode[i * n + j - 1] == 0) {
61.                                  G[i * n + j][2] = i * n + j - 1;
62.                          }
63.                          // right
64.                          if (j != n - 1 && noNode[i * n + j + 1] == 0) {
65.                                  G[i * n + j][3] = i * n + j + 1;
66.                          }
67.                  }
68.          }
69.  }
70.
71.  if (findMaxMatching()) {
72.          cout << "true" << endl;
73.          for (auto i : L) {
74.                  int j = Matching[i];
75.                  if (j + 1 == i || j - 1 == i) {
76.                          res[i / n][i % n] = '#';
```

```
77.                         res[j / n][j % n] = '#';
78.                 }
79.                 else {
80.                         res[i / n][i % n] = '0';
81.                         res[j / n][j % n] = '0';
82.                 }
83.         }
84.         for (int i = 0; i < m; ++i) {
85.                 cout << res[i] << "\n";
86.         }
87.  }
88.  else {
89.         cout << "false" << endl;
90.  }
91.  return 0;
92. }
93.
94.  bool dfs(int u) {
95.   for (int i = 0; i < 4; ++i) {
96.         if (G[u][i] == -1)
97.                 continue;
98.         int v = G[u][i];
99.         if (!check[v]) {
100.                check[v] = true;
101.                if (Matching[v] == -1 || dfs(Matching[v])) {
102.                        Matching[v] = u;
103.                        Matching[u] = v;
104.                        return true;
105.                }
106.        }
107. }
108. return false;
109. }
110.
111. bool findMaxMatching() {
112.  int ans = 0;
113.  if (L.size() != R.size()) {
114.         return false;
115.  }
116.  check = new int[m * n];
117.  for (int i = 0; i < L.size(); ++i) {
118.         int node = L[i];
119.         if (Matching[node] == -1) {
120.                 memset(check, 0, m * n * sizeof(int));
```

```
121.                if (dfs(node)) {
122.                        ans++;
123.                }
124.        }
125. }
126. return ans == L.size();
127.        }
```

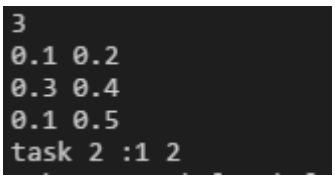
2. Interval scheduling problem 1

一道典型的贪心问题，上课的时候得到了贪心结构并且证明了可以得到最优解。使用优先队列的数据结构方便编程。时间复杂度为 O(nlgn)

a) 伪代码

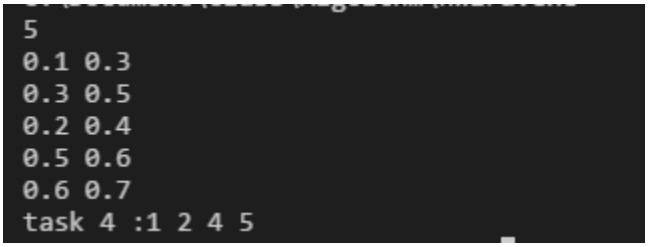
```
vector R //保存结果
Heap = getInput() //按照结束时间构建小端在上的堆
while(Heap isn't empty)
   interval = Heap.pop()
   if(R is empty)
      R.push(interval)
   else if(interval.s >= R.lastElm.f)//保证没有 overlap
      R.push(interval)
output R
```

b) 样例测试



输入：3 个区间，（0.1，0.2）（0.3，0.4）（0.1，0.5）

输出：最多包含两个任务，为第一个和第二个



输入输出格式同上，结果正确

c) 代码

```
#include <iostream>
#include <algorithm>
#include <queue>
#include <vector>
using namespace std;

class interval {
public:
 double s;
 double f;
 int th;
};
struct cmp {
 bool operator()(interval a, interval b) {
      return a.f > b.f;
 }
};

int main() {
 int n = 0;
 cin >> n;
 interval val;
 vector<interval> R;
 priority_queue<interval, vector<interval>, cmp> H;
 for (int i = 0; i < n; ++i) {
      cin >> val.s >> val.f;
      val.th = i + 1;
      H.push(val);
 }
 for (int i = 0; i < n; ++i) {
      interval t = H.top();
      H.pop();
      if (R.empty()) {
           R.push_back(t);
      }
      if (t.s >= R[R.size() - 1].f) {
           R.push_back(t);
      }
 }
 cout << "task " << R.size() << " :";
 for (auto i : R) {
      cout << i.th << " ";
 }
 return 0;
```

```
}
```

3 Interval scheduling problem 2

动态规划的经典问题，将任务按照结束时间排序后，只观察前 i 个任务，则所能获得的最大收益等于在

1. 只观察前 i-1 个任务所能获得的最大收益（即放弃 i）
2. i 任务的收益和能与 i 任务并存的任务中所能取得的最大收益（即保留 i）

这两项中取较大值

而求能与 i 任务并存的任务的最大收益就成为了另一个问题。在任务序号 i 的递增过程中，最大收益肯定递增，所以只需要知道能与 i 任务并存的最大的序号是多少就好。

按照这个思路即可写出代码。

a) 伪代码

```
T = sort(Input);
for(i to n)
    p[i] =  可以与 i 任务并存的最靠后的任务//二分查找降低复杂度
set(opt[i] = w[i])
for(i to n)
    opt[i] = max(opt[i -1], wi + opt[ p[i] )
output
```

b) 样例

```
3
0.1 0.2 100
0.2 0.3 500
0.1 0.3 400
600:2 1
```

输入 3 个区间，输出结果正确

```
5
0.1 0.2 50
0.2 0.4 50
0.1 0.3 50
0.4 0.5 20
0.3 0.5 30
120:4 2 1
```

输入五个区间，结果正确

```
4
0.01 0.02 100
0.01 0.03 100
0.02 0.04 1
0.03 0.05 1
101:3 1
```

输入四个区间，结果正确

c) 代码

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <cstring>
using namespace std;
struct interval {
    double s;
    double f;
    int w;
    int th;
};
bool cmp(interval a, interval b) {
    return a.f < b.f;
}
int binary_find(vector<interval> &v, int b, int e, double s);
int main() {
    int n;
    cin >> n;

    interval temp;
    int* p = new int[n];
    int* dp = new int[n];
    memset(p, -1, n * sizeof(int));
    memset(dp, 0, n * sizeof(int));
    vector<interval> list;
    vector<int> ll;
    for (int i = 0; i < n; ++i) {
        cin >> temp.s >> temp.f >> temp.w;
        temp.th = i + 1;
        dp[i] = temp.w;
        list.push_back(temp);
    }

    sort(list.begin(), list.end(), cmp);

    for (int i = 0; i < n; ++i) {
        double s = list[i].s;
        p[i] = binary_find(list, 0, i, s);
    }

    for (int i = 1; i < n; ++i) {
        dp[i] = dp[i] > dp[i - 1] ? dp[i] : dp[i - 1];
```

```
170.          if (p[i] != -1) {
171.              dp[i] = dp[i] > dp[p[i]] + list[i].w ? dp[i] : dp[p[i]] +
    +list[i].w;
172.          }
173.      }
174.      int res = dp[n - 1];
175.      int index = n - 1;
176.
177.      while (res != 0) {
178.          if (res == dp[index - 1]) {
179.              index = index - 1;
180.          }
181.          else {
182.              res -= list[index].w;
183.              ll.push_back(list[index].th);
184.              index -= 1;
185.          }
186.      }
187.
188.      cout << dp[n - 1] << ":";
189.      for (auto i : ll) {
190.          cout << i << " ";
191.      }
192.      return 0;
193. }
194.
195. int binary_find(vector<interval> &v, int b, int e, double s) {
196.      if (b >= e) {
197.          return -1;
198.      }
199.      int mid = (b + e) / 2;
200.      if (v[mid].f > s) {
201.          e = mid;
202.      }
203.      else if (v[mid].f == s || b + 1 == e) {
204.          return mid;
205.      }
206.      else {
207.          b = mid;
208.      }
209.      return binary_find(v, b, e, s);
210.}
```