1 算法分析

- 第二题是第三题在$value = 1$下的特例.
- 假设有N个格子.
- 第三题是一个典型的动态规划问题, 其子问题是
- 在各个end_time之前, 能够取到的最大值
- 在end_time取到max_end_time时, 为所求max_value
- 递推方程为$DP[end_time] = \max_{t \le begin_time}(DP[t] + value, DP[prev(end)])$
- 递推方程求解$O(N)$次, 每一次为二分查表, 耗时$O(logN)$, 共计$O(NlogN)$

2 伪代码

1. if Problem.No is 2, assign 1.0 to all values
2. sort $node(beg, end, value)$ by end, increasingly
3. let $max = 0$
4. for each $node$
5. if $DP[beg] + value > max$
6. update $max \leftarrow DP[beg] + value$
7. insert $DP[end] \leftarrow max$
8. return max

3 代码

见 interval_cover 文件夹内容, 推荐使用cmake编译

```
double interval_cover(vector<Data> courses) {
  std::sort(courses.begin(), courses.end(),
            [](Data a, Data b) { return get<1>(a) < get<1>(b); });
  //
  // endtime ==> value
  std::map<double, double> table;
  table[std::numeric_limits<double>::min()] = 0;
  table[std::numeric_limits<double>::max()] =
      std::numeric_limits<double>::max();
  //
  double max = 0;
  for (auto t : courses) {
    auto beg_time = get<0>(t);
    auto end_time = get<1>(t);
    auto value = get<2>(t);
    auto iter = table.upper_bound(beg_time);
    --iter;
    if (iter->second + value > max) {
      max = iter->second + value;
      table[end_time] = max;
    }
  }
  return max;
```

```
}
```

4 测试样例

使用Google Test进行单元测试, 请查看test.cpp
    包含一个二进制文件, 有大量测试数据, 结果为李真同学计算的数据

```
TEST(interval_cover, test0) {
  vector<Data> data{
      {1, 1, 2.0}, //
      {2, 3, 2.0}, //
      {3, 4, 2.0}, //
      {1, 4, 5.0}, //
      {0, 1, 1.0}, //
  };
  int sum = interval_cover(std::move(data));
  EXPECT_EQ(sum, 7);
}


TEST(interval_cover, binary_test) {
  // std::unique_ptr<FILE, decltype(&fclose)> fp{fopen("p3-in-big.dat", "rb"), &fclose};
  auto fp = fopen("p3-in-big.dat", "rb");
  ASSERT_TRUE(!!fp) <<  "File Open failed";
  uint32_t n;
  fread(&n, 4, 1, fp);
  vector<Data> data(n);
  fread(data.data(), 8, 3 * n, fp);
  fclose(fp);
  for(int i = 0; i < n; ++i){
    std::swap(std::get<0>(data[i]), std::get<2>(data[i]));
  }
  // auto t1 = chrono::high_resolution_clock::now();
  double sum = interval_cover(data);
  // auto t2 = chrono::high_resolution_clock::now();
  // auto time = chrono::duration_cast<chrono::milliseconds>(t2 - t1).count();
  EXPECT_FLOAT_EQ(sum, 72.648146) << "wrong answer";
}
```