Gone Fishing (hunnu10663)

- ACM/ICPC Regional Contest East Central North America1999

瞿绍军

湖南师范大学

- 题意描述：

john现有h个小时的空闲时间，他打算去钓鱼。john钓鱼的地方共有n个湖，所有的湖沿着一条单向路顺序排列（john每在一个湖钓完鱼后，他只能走到下一个湖继续钓）， john必须从1号湖开始钓起，但是他可以在任何一个湖结束他此次钓鱼的行程。

- 题意描述：

john在每个湖中每5分钟钓的鱼数（此题中以5分钟作为单位时间），随时间的增长而线性递减。而每个湖中头5分钟可以钓到的鱼数以及每个湖中相邻5分钟钓鱼数的减少量，input中均会给出。并且John从任意一个湖走到它下一个湖的时间input中也都给出。

- 问题：

  求一种方案，使得john在有限的h小时中可以钓到尽可能多的鱼。

  output中需包括john在所有湖边所呆的时间，以及最后总的钓鱼数。

- Sample input

  2　　-湖的数量

  1　　-john有的总时间

  10　1　　-湖中头5分钟可以钓到的鱼数

  2　5　　湖中相邻5分钟钓鱼数的减少量

  2　　-从第一个湖走到第二个湖的时间

- Sample output

  45, 5　　-分别在两个湖中呆的时间

  Number of fish expected: 31

  –钓鱼的总数

- 问题分析：
  - 先考虑下面问题：

    john不能从任意湖结束他的行程，而必须一直钓到最后一个湖

那么应该怎么做呢？

- 由于每个湖都必须经过，且只经过一次，所以john花在路中的总时间是确定的。
- 在这个条件下，可以想成john学会了“瞬间移动”，即：他可以在任何时间，移动到任何他想去的湖，而移动的过程无需时间。
- 于是，john只需在每个5分钟的开始“瞬间移动”到当前5分钟中能钓到最多的鱼的湖中，且只钓5分钟的鱼。
- 这样可以保证john钓到尽可能多的鱼。

- Sample:

10(0)

15(3)

12(4)

14(3)

Sum=20 +17 +16 +……

- 回到最初的问题：

  只要枚举john的行程是从第一个湖到第k个湖（1<=k<=n），采取贪心策略，每次选一个鱼最多的湖泊钓一次鱼。对于每个湖泊来说，由于在任何时候鱼的数目只和佳佳在该湖里钓鱼的次数有关，和钓鱼总次数无关，所以这个策略是最优的。

- 枚举+贪心

  时间复杂度：0(kn^2)

- 贪心算法：

  每次选择一个局部最优策略进行实施，而不去考虑对今后的影响。

  时间复杂度非常低

  但适用范围不够广

stl如何定义排序

- 定义 < 操作符
- 使用这种方法可以使我们自定义的类能够获得与生俱来的排序能力。例如，如果有如下类：

```
struct Edge
{
    int from,to ,weight;
};
```

stl如何定义排序

- 因为要实现Kruskai算法，你希望图中的所有边依据权重按降序排列。 像这样来定义operator<

```
struct Edge
{
	int from, to ,weight;
	bool operator <(Edge other) const   //只能重载<
	{
		return weight>other.weight;
	}
};
```

stl如何定义排序

- 如果你不喜欢这种方式，比如，明明是要比较两个对象，方法却只有一个参数。你可以选择如下方式：

```
struct Edge
{
    int from,to weight;
    friend bool operator<(Edge a,Edge b)
    {
        return a.weight>b.weight;
    }
};
```

stl如何定义排序

- 堆、优先级队列中比较函数的使用
- priority_queue<int,vector<int>,less<int> > pq; //大顶堆
- priority_queue<int,vector<int>, greater<int> > pq; //小顶堆
- 自定义

stl如何定义排序

```
class elem1
{
public:
    elem1();
    elem1(int aa):a(aa){}
    friend bool operator < (const elem1 &e1,const elem1 &e2)  //只能重载<
    {
        return e1.a <e2.a;    //大顶堆是<;小顶堆是>
    }
};
……
priority_queue<elem1> q1;
```

```
class Scorer
{
public:
    bool operator ()(const elem &e1,const elem &e2)
    {
        return e1.a<e2.a;
    }
};
……
priority_queue<elem,vector<elem>,Scorer > q;
```