数据结构与算法 II 上机实验 (11.10)

中国人民大学　信息学院　崔冠宇　2018202147

注：请使用 C++17 或以上编译器！

上机题 T16.2-4　Gekko 教授一直梦想用直排轮滑的方式横穿北达科他州。他计划沿 U.S.2 号高速公路横穿，这条高速公路从明尼苏达州东部边境的大福克斯市到靠近蒙大拿州西部边境的威利斯顿市。教授计划带两公升水，在喝光水之前能滑行 m 英里（由于北达科他州地势相对平坦，教授无需担心在上坡路段喝水速度比平地或下坡路段快）。教授从大福克斯市出发时带整整两公升水。他携带的北达科他州官方地图显示了 U.S.2 号公路上所有可以补充水的地点，以及这些地点间的距离。

教授的目标是最小化横穿途中补充水的次数。设计一个高效的方法，以帮助教授确定应该在哪些地点补充水。证明你的策略会生成最优解，分析其运行时间。

一、问题描述

最小化旅程中的补水次数。（具体描述见题目要求）

1. 输入：一次补水的“续航距离”m，路程总长度 L，各补水点距离起点的距离 $d =< d_0, d_1, d_2, \cdots, d_n, d_{n+1} >$（其中 $d_0 = 0$，表示出发点；$d_{n+1} = L$ 表示终点与起点的距离；保证按非减排列）。
2. 输出：最小补水次数 $count$，选择的补水点的下标 $D =< D_0, D_1, D_2, \cdots, D_k >$（其中 $D_0 = 0$，表示出发点；保证非减排列）。
3. 附加约束条件：任意两相邻补水点之间的距离小于等于 m，保证一次补水至少能到达下一补水点。

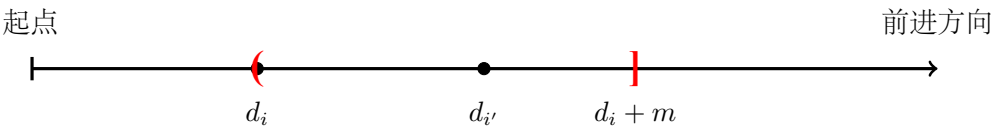
二、算法基本思路

直觉上看，每次选择应该让教授尽可能走的远一些，水接近耗尽（即前进里程接近 m）时再次补水。下面先分析问题是否具有最优子结构和贪心选择性质：

- **最优子结构**：

  记 $S(i)(i = 0, 1, \cdots, n)$ 为从补水点 i 出发后最优补水点的下标 i'，$C(i)(i = 0, 1, \cdots, n)$ 为从 i 出发的最小补水次数。

  显然当 $d_i + m \geq L$ 时，可以直接到达终点，从这些点出发不需要补水；否则从某 i 出发的最优方案必然是先到达 $(d_i, d_i + m]$ 范围内的某补水点 i'，然后加上从 i' 出发的最优方案。选择下一次补水点 i' 应该使从 i' 出发补水次数最少（若最优方案不是如此的话，“剪贴”后续最少补水点的方案至此点之后，将使补水次数减少）；同时从 i 出发所需的最小补水次数为下一次补水点补水次数加一（起点不加）。



根据上述讨论，可以写出递归式：

$$S(i)=\begin{cases}\arg\min\limits_{i',d_i<d_{i'}\le d_i+m} C(i'), & d_i+m<L\\ n+1, & d_i+m\ge L\end{cases}$$

$$C(i)=\begin{cases}\min\limits_{i',d_i<d_{i'}\le d_i+m} C(i'), & i=0\text{且}d_i+m<L\\ \min\limits_{i',d_i<d_{i'}\le d_i+m} C(i')+1, & i\ge 1\text{且}d_i+m<L\\ 1, & d_i+m\ge L\end{cases}$$

题目要求解的是最小补水次数 $C(0)$ 以及补水点序列 $0\to S(0)\to S(S(0))\to\cdots n+1$。至此，若从后至前填表 S 与 C，就已经可以写出动态规划的解法了。但是此题还具有贪心选择性质，使得可以被进一步改进。

- **贪心选择性质：**

  每次尽量让教授在满足约束条件 $d_i<d_{i'}\le d_{i+1}$ 下走的远一些，即

$$S_g(i)=\begin{cases}\max\limits_{i,d_i<d_{i'}\le d_i+m}\{i'\}, & d_i+m<L\\ n+1, & d_i+m\ge L\end{cases}$$

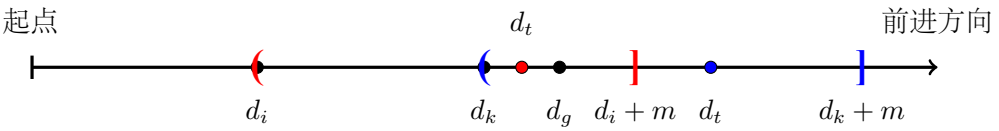
  下面证明贪心选择性质可以生成最优解之一：

  如下图所示，假若教授在补水点 i 不贪心选择下一补水点 g（g 是 d_i+m 前最远的点），而是选择较近的补水点 k 就能生成最优解。设选择 k 的最优解下一次选择要经过补水点 $t(d_k<d_t\le d_k+m)$，则贪心法生成的解是 $i\to g\to\cdots$，而最优解是 $i\to k\to t\to\cdots$，此时一定有 $d_i<d_k<d_g\le d_k+m<d_g+m$。

  若教授从最优解 k 改选 g，则

  1. 要么教授可以从 g 直接到达 t（因为 $d_k<d_t\le d_k+m$，蓝色点情况），从而 $C(k)=C(g)$，即选择 g 也能构成最优解；
  2. 要么贪心点 g 已经远于 t（红色点情况），此时 $i\to k\to t\to\cdots$ 比 $i\to g\to\cdots$ 补水次数还要多。

  综上，贪心选择总可以保证生成最优解，所以此题目有贪心选择性质。



可以写出贪心算法实现的补水问题的伪代码：

```
// 最优补水问题
// 参数：
//      m：续航距离
```

```
// 	d: 补水点与起点的距离
// 返回值:
// 	D: 最优补水点下标序列
GET-WATER(m, d):
    let D[] be a new array
    D[0] = 0
    j = 0
    // 遍历数组 d
    for i = 0 to d.size() - 1:
        // 与上一补水点距离刚大于 m
        if d[i] - d[D[j]] > m:
            // 数量增加
            j++
            // 上一点的坐标放入补水点数组
            D[j] = i - 1
    return D
```

同时重构解也很简单:

```
// 打印解
// 参数:
// 	D: 最优补水点下标序列
// 	d: 补水点与起点的距离
PrintSolution(D, d):
    print("在起点出发。")
    // 打印各个补水点信息
    for(i = 1 to i < D.size() - 1:
        print("在第 ", D[i], " 个补水点补水，此时距离起点 ", d[D[i]], "。")
    print("到达终点，此时距离起点 ", d[d.size() - 1], "。")
    print("共补水 ", D.size() - 1, " 次。")
```

三、算法复杂性分析

先分析 GET-WATER 的复杂度:

设问题规模 n 为补水点的总数。

1. 时间复杂度:
   算法从前到后扫描数组 d，所以时间复杂度为 $T(n) = \Theta(n)$。

2. 空间复杂度：

   算法返回的各补水点的下标肯定不超过 n 个，所以空间复杂度为 $S(n) = O(n)$。

再分析 PrintSolution 的复杂度：

1. 时间复杂度：

   算法从前到后扫描数组 D，而 D 中元素个数不超过 n 个，所以时间复杂度为 $T(n) = O(n)$。

2. 空间复杂度：

   算法内部不需要辅助空间，所以空间复杂度为 $T(n) = \Theta(1)$。

四、程序源代码

最优补水问题: **getwater.cpp**:

```cpp
#include <iostream>
#include <vector>

// 最优补水问题
// 参数：
//      m: 续航距离
//      d: 补水点与起点的距离
// 返回值：
//      D: 最优补水点下标序列
// 时间复杂度：
//      T(n) = O(n)
// 空间复杂度：
//      S(n) = O(n)
auto GetWater(int m, const std::vector<int> & d)
{
    // 新 vector 用于存储补水点下标
    std::vector<size_t> D;
    // 最多与 d 一样多
    D.resize(d.size());
    D[0] = 0;
    // 补水点个数
    size_t count = 0;
    // 遍历数组 d
    for(size_t i = 0; i < d.size(); i++)
    {
```

```
26          // 与上一补水点距离刚大于 m
27          if(d[i] - d[D[count]] > m)
28          {
29              count++;
30              D[count] = i - 1;
31          }
32      }
33      // 去掉多余空间
34      D.resize(count + 1);
35      return D;
36  }
37
38  // 打印解
39  // 参数:
40  //      D: 最优补水点下标序列
41  //      d: 补水点与起点的距离
42  // 时间复杂度:
43  //      T(n) = O(n)
44  // 空间复杂度:
45  //      S(n) = O(1)
46  void PrintSolution(const std::vector<size_t> & D, const std::vector<int> & d)
47  {
48      std::cout << "在起点出发。" << std::endl;
49      // 打印各个补水点信息
50      for(size_t i = 1; i < D.size(); i++)
51      {
52          std::cout << "在第 " << D[i] << " 个补水点补水，此时距离起点 " << d[D[i]]
        << "。" << std::endl;
53      }
54      std::cout << "到达终点，此时距离起点 " << d[d.size() - 1] << "。" << std::endl;
55      std::cout << "共补水 " << D.size() - 1 << " 次。" << std::endl;
56  }
57
58  int main()
59  {
60      int m = 5;
61      int L = 26;
```

```
62      std::vector<int> d = {0, 2, 5, 7, 9, 13, 14, 15, 18, 21, 25, L};
63      auto D = GetWater(m, d);
64      PrintSolution(D, d);
65      return 0;
66  }
```

五、运行结果截图

编译运行 getwater.cpp，程序以源代码中的例子做测试：

（$m = 5$，$L = 26$，$d =< 0, 2, 5, 7, 9, 13, 14, 15, 18, 21, 25, L >$）

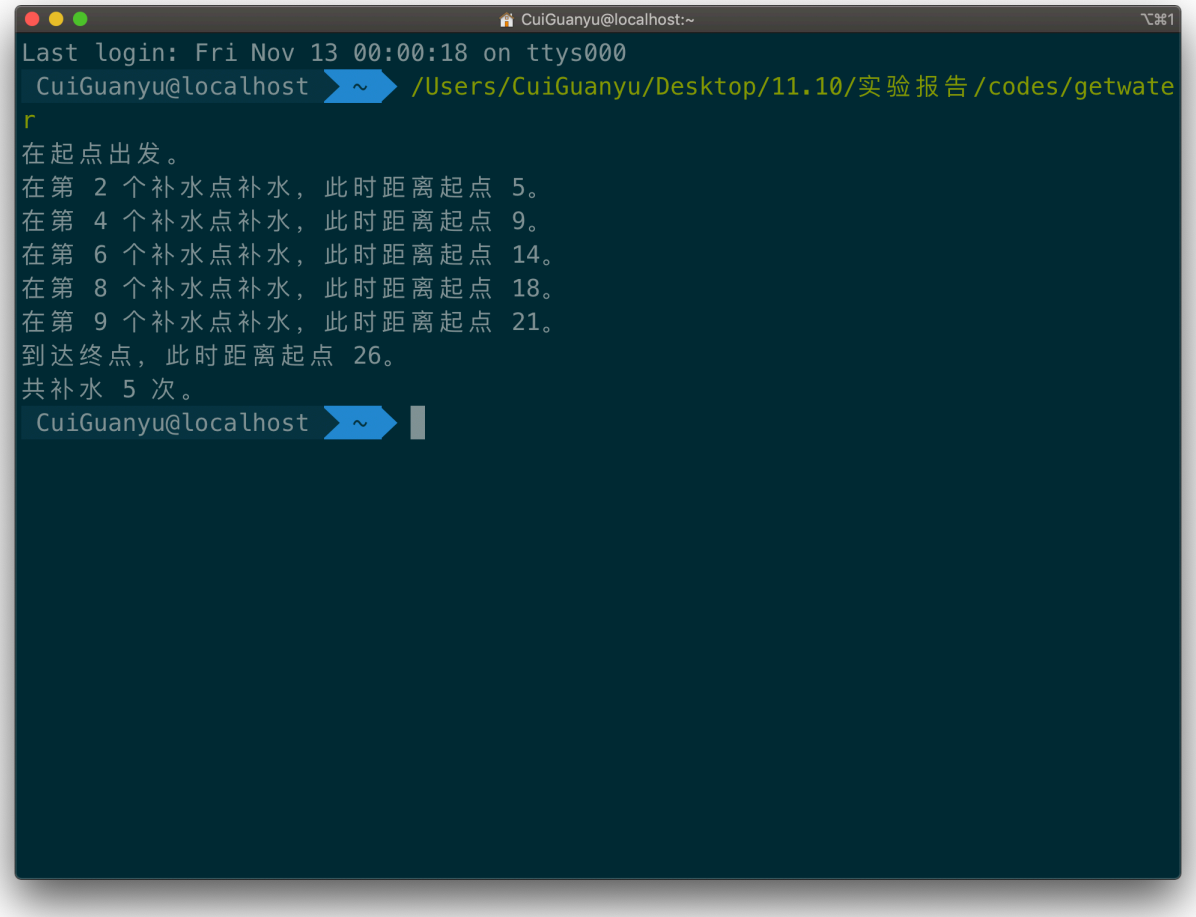


图 1: 最优补水问题测试

观察可见程序运行正确（尤其是到达距离为 21 的点时，可以直接到达终点）。