> 现在有n个城市，分别用0,1…,n - 1这些序号表示，城市之间的航线用三元组[from, to, price]来表示，比如说三元组[0,1,100]就表示，从城市0到城市1之间的机票价格是 100 元。
> 题目会给你输入若干参数：
> 正整数n代表城市个数，数组flights装着若干三元组代表城市间的航线及价格，城市编号src代表你所在的城市，城市编号dst代表你要去的目标城市，整数K代表你最多经过的中转站个数。

分析： 从题目理解，可以看出本题问加权有向图求最小路径问题。 即

> 一幅加权有向图，让你求src到dst权重最小的一条路径，同时要满足，这条路径最多不能超过K + 1条边（经过K个节点相当于经过K + 1条边。

由此首先要构造图数据结构，因为为有向图可以采用邻接表的形式。 建立函数dp,表示经过k次中转到达节点s的最小花费dp(s,k)。 递归算法：

```
class Solution {
public:
  int findCheapestPrice(int                                n,
                        std::vector<std::vector<int>>& flights,
                        int                                src,
                        int                                dst,
                        int                                k) {
    src_ = src;
    dst_ = dst;

    // 构建图
    for (auto item : flights) {
      int to    = item[1];
      int from  = item[0];
      int price = item[2];
      graph_[to].push_back(std::vector<int>{from, price});
    }

    return dp(dst, k);
  }

private:
  int dp(int s, int k) {
    if (s == src_) {
      return 0;
    }

    if (k == -1) {
      return -1;
    }

    int res = INT_MAX;
```

```
    if (graph_.count(s)) {
      // 当s有入度节点时分解为子问题
      for (auto item : graph_[s]) {
        int from  = item[0];
        int price = item[1];
        // 从src 到达临边所需的最小代价
        int subProblem = dp(from, k - 1);
        if (subProblem == -1) {
          continue;
        } else {
          res = std::min(res, price + subProblem);
        }
      }
    }

    return res == INT_MAX ? -1 : res;
  }

  std::map<int, std::vector<std::vector<int>>> graph_;  // to-->from--
>price
  int                                          src_;
  int                                          dst_;
};
```

备忘录优化：

```
class Solution {
public:
  int findCheapestPrice(int                              n,
                        std::vector<std::vector<int>>& flights,
                        int                              src,
                        int                              dst,
                        int                              k) {
    src_ = src;
    dst_ = dst;

    memo =
        std::vector<std::vector<int>>(n + 1, std::vector<int>(k + 1,
INT_MIN));

    // 构建图
    for (auto item : flights) {
      int to    = item[1];
      int from  = item[0];
      int price = item[2];
      graph_[to].push_back(std::vector<int>{from, price});
    }

    return dp(dst, k, memo);
  }

private:
```

```
    int dp(int s, int k, std::vector<std::vector<int>>& memo) {
      if (s == src_) {
        return 0;
      }

      if (k == -1) {
        return -1;
      }

      if (memo[s][k] != INT_MIN) {
        return memo[s][k];
      }

      int res = INT_MAX;

      if (graph_.count(s)) {
        // 当s有入度节点时分解为子问题
        for (auto item : graph_[s]) {
          int from  = item[0];
          int price = item[1];
          // 从src 到达临边所需的最小代价
          int subProblem = dp(from, k - 1, memo);
          if (subProblem == -1) {
            continue;
          } else {
            res = std::min(res, price + subProblem);
          }
        }
      }

      return memo[s][k] = res == INT_MAX ? -1 : res;
    }

    std::map<int, std::vector<std::vector<int>>> graph_;  // to-->from--
>price
    int                                          src_;
    int                                          dst_;
    std::vector<std::vector<int>>                memo;
};
```