1. 未排序数组和为指定值的最大长度。

```
#include <map>
#include <vector>

class Solution {
public:
  int maxLength(std::vector<int> &nums, int target) {
    int len = nums.size();
    if (len == 0) {
      return 0;
    }

    std::map<int, int> sumMap;
    int                res = INT_MIN;
    int                sum = 0;
    sumMap.insert(0, -1);

    for (int i = 0; i < len; i++) {
      sum += nums[i];

      if (sumMap.count(sum - target)) {
        res = std::max(res, i - sumMap[sum - target]);
      }

      if (!sumMap.count(sum)) {
        sumMap[sum] = i;
      }
    }

    return res;
  }
};
```

2. 二叉树中和为指定值的最大长度。 分析：
3. 主要记录当前遍历的路径，采用回溯算法计算。

```
#include <cstddef>
#include <map>

struct TreeNode {
  int       val;
  TreeNode *left;
  TreeNode *right;
  TreeNode() : val(0), left(nullptr), right(nullptr) {
  }
  TreeNode(int x) : val(x), left(nullptr), right(nullptr) {
```

```
  }
  TreeNode(int x, TreeNode *left, TreeNode *right)
      : val(x), left(left), right(right) {
  }
};

class Solution {
public:
  int getMaxLengthPartSum(TreeNode *root, int sum) {
    std::map<int, int> sumMap;  // 存储当前和与level
    sumMap.insert(0, 0);

    return getMaxLengthPartSum(root, sum, 0, 1, 0, sumMap);
  }

private:
  int getMaxLengthPartSum(TreeNode           *root,
                          int                 target,
                          int                 presum,
                          int                 level,
                          int                 maxLen,
                          std::map<int, int> &sumMap) {
    if (root == nullptr) {
      return 0;
    }

    // 做选择
    int cursum = presum + root->val;

    // 判断是否存在
    if (sumMap.count(cursum) == 0) {
      sumMap[cursum] = level;
    }

    // 最值判断
    if (sumMap.count(cursum - target)) {
      maxLen = std::max(maxLen, level - sumMap[cursum - target]);
    }

    // 递归
    maxLen = getMaxLengthPartSum(root->left,
                                 target,
                                 cursum,
                                 level + 1,
                                 maxLen,
                                 sumMap);
    maxLen = getMaxLengthPartSum(root->right,
                                 target,
                                 cursum,
                                 level + 1,
                                 maxLen,
                                 sumMap);

    // 回溯，撤销选择，需要重新判断当前cursum是否为本次设置
```

```
      if (level == sumMap[cursum]) {
        sumMap.erase(level);
      }

      return maxLen;
    }
  };
```