# Q-M算法

19063110蔡思林

19063140郑凯心

## 一、需求分析

本算法要求输入一最小项和无关项的待化简表达式，通过程序处理输出化简后的结果。

## 二、算法流程

程序首先要求用户输入待化简表达式的变量数，然后要求用户逐个输入值为1的项，以2表示输入结束，再要求用户逐个输入无关项，同样以2结束。程序将用户输入的所有数据储存在一个数据容器vector 中，并将相关项单独存在另一个vector 中。

然后程序通过循环遍历所有相关项，将相邻的项合并成蕴涵项存入新建的 vector中，无法再合并的项显然为本源蕴涵项，存入专门的容器。之后再对新建的vector中的蕴涵项重复上述操作，直到找到所有的本源蕴涵项。由于变量最多为10个，故合并最多进行10次，故所有合并可在一个10元vector数组内完成。

当之前的工作完成时,此时显然已找到所有的本源蕴涵项,此时遍历之前存储的相关项，统计被覆盖的次数并输出只被覆盖一次的项所属的本源蕴涵项(皆为本质本源蕴涵项)。之后再从vector尾部开始输出其他本源蕴涵项直到所有的相关项都被覆盖为止。

## 三、运行结果

测试数据采用F=Σm(4,5,6,8,9,10,13)+d(0,7,15);结果并非最优的，是因为最后没有寻找最小覆盖导致的。

输入

4

0100

0101

0110

1000

1001

1010

1101

2

0000

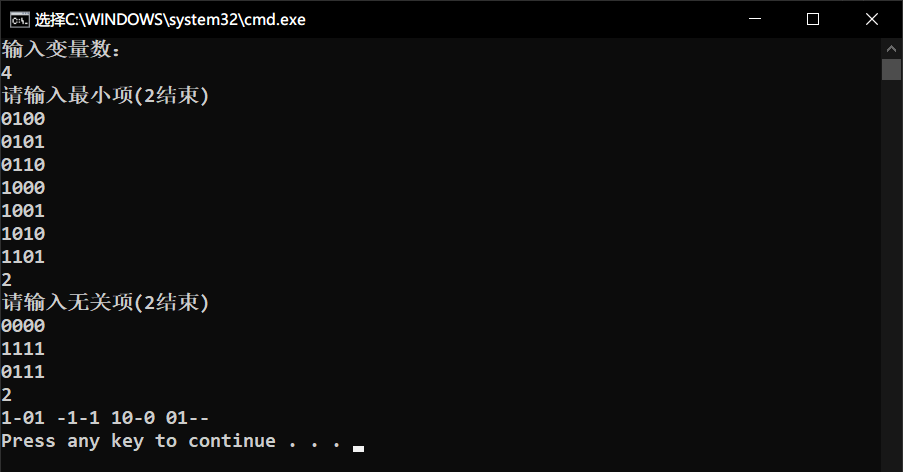
1111

0111

2

输出

1-01 -1-1 10-0 01--



## 四、C++源代码

\*/#include <bits/stdc++.h>

using namespace std;

bool comp(int n, char \*a, char \*b) {

  for (int i = 0; i < n; ++i) {

    if (a[i] != b[i] && (a[i] != '-' && b[i] != '-')) return false;

  }

  return true;

}

int implicant(int n, char \*a, char \*b) {

  int count = 0, temp;

  for (int i = 0; i < n; ++i) {

    if (a[i] != b[i]) {

      count++;

      temp = i;

    }

  }

  if (count == 1) return temp;

  else return -1;

}

bool cointain(int n, char \*a, char \*b) {

  for (int i = 0; i < n; ++i) {

    if (a[i] != b[i] && (a[i] != '-')) return false;

  }

  return true;

}

signed main() {

  int n, count, tmp;

  char \*temp;

  bool flag[2];

  vector<char\*> v[10];

  vector<char\*> relative;

  vector<char\*> prime;

  vector<char\*> result;

  cout << "输入变量数：\n";

  cin >> n;

  temp = new char[n];

  cout << "请输入最小项(2结束)\n";

  flag[0] = true;

  while (flag[0]) {

    for (int i = 0; i < n; ++i) {

      cin >> temp[i];

      if (temp[i] != '0' && temp[i] != '1') {

        flag[0] = false;

        break;

      }

    }

    if (flag[0]) {

      v[0].emplace\_back(new char[n]);

      relative.emplace\_back(new char[n]);

      strcpy(v[0].back(), temp);

      strcpy(relative.back(), temp);

    }

  }

  cout << "请输入无关项(2结束)\n";

  flag[0] = true;

  while (flag[0]) {

    for (int i = 0; i < n; ++i) {

      cin >> temp[i];

      if (temp[i] != '0' && temp[i] != '1') {

        flag[0] = false;

        break;

      }

    }

    if (flag[0]) {

      v[0].emplace\_back(new char[n]);

      strcpy(v[0].back(), temp);

    }

  }

  for (int i = 0; i < 10; ++i) {

    if (v[i].empty()) break;

    for (int j = 0; j < (int)v[i].size(); ++j) {

      flag[0] = false;

      for (int k = 0; k < (int)v[i].size(); ++k) {

        if (implicant(n ,v[i][j], v[i][k]) != -1) {

          strcpy(temp, v[i][j]);

          temp[implicant(n, v[i][j], v[i][k])] = '-';

          flag[1] = true;

          flag[0] = true;

          for (int l = 0; l < (int)v[i+1].size(); ++l) {

            if (!strcmp(v[i+1][l], temp)) flag[1] = false;

          }

          if (flag[1]) {

            v[i+1].emplace\_back(new char[n]);

            strcpy(v[i+1].back(), temp);

          }

        }

      }

      if (!flag[0]) {

        prime.emplace\_back(new char[n]);

        strcpy(prime.back(), v[i][j]);

      }

    }

  }

  for (int i = 0; i < (int)relative.size() && !relative.empty(); ++i) {

    count = 0;

    for (int j = 0; j < (int)prime.size() && !prime.empty(); ++j) {

      if (comp(n, relative[i], prime[j])) {

        count++;

        tmp = j;

      }

    }

    if (count == 1) {

      result.emplace\_back(new char[n]);

      strcpy(result.back(), prime[tmp]);

      for (int j = 0; j < (int)relative.size(); ++j) {

        if (comp(n, relative[j], prime[tmp])) {

          relative.erase(relative.begin()+j);

          --j;

        }

      }

      prime.erase(prime.begin()+tmp);

      --i;

    }

  }

  while (!relative.empty()) {

    strcpy(temp, prime.back());

    prime.pop\_back();

    count = 0;

    for (int i = 0; i < (int)relative.size(); ++i) {

      if (comp(n, relative[i], temp)) {

        relative.erase(relative.begin()+i);

        --i;

        ++count;

      }

    }

    if (count > 0) {

      result.emplace\_back(new char[n]);

      strcpy(result.back(), temp);

    }

  }

  while (result.size()) {

    for (int i = 0; i < 4; ++i) {

      cout << \*(result.back()+i);

    }

    cout << ' ';

    result.pop\_back();

  }

  cout << '\n';

  system("pause");

  return 0;

}

## 演示视频

略

## 参考文献

[1] QM算法源代码及说明,https://wenku.baidu.com/view/e4dc6dc2bb4cf7ec4afed0f3.html?qq-pf-to=pcqq.c2c