計算機輔助設計特論 HW1

311510207 江尹凡

讀檔

```
221  int main(int argc, char* argv[])
222  {
223     ifstream inFile;
224      inFile.open(argv[1]);
225      ofstream fout(argv[2]);
226      if (!inFile) {
227          return 1;
228      }
229      string line;
230      int int_temp;
231      while (getline(inFile, line)) {
232          stringstream input;
233          if(line == ".i"){
234              getline(inFile, line);
235              input << line;
236              input >> num_var;
237          }
238          if(line == ".m"){
239              getline(inFile, line);
240              input << line;
241
242              while(input >> int_temp){
243
244                  ON_set.push_back(int_temp);
245              }
246          }
247          if(line == ".d"){
248              getline(inFile, line);
249              input << line;
250              while(input >> int_temp){
251                  DC_set.push_back(int_temp);
252              }
253          }
254      }
255      inFile.close();
```

計算 Part 1 (重續迭代直到沒有需要合併項)

```
257     for(int i=0 ; i<ON_set.size() ; i++){
258         Implicant.push_back(dec2bin(ON_set[i]));
259     }
260     for(int i=0 ; i<DC_set.size() ; i++){
261         Implicant.push_back(dec2bin(DC_set[i]));
262     }
263 
264      ON_set_binary.resize(ON_set.size() );
265      ON_set_binary.assign  (Implicant.begin(),Implicant.begin()+ON_set.size());
266 
267 
268     bool Implicant_done = false;
269 
270     while(!Implicant_done){
271         used.resize(Implicant.size() );
272         Implicant_done = Implicant_caculation(Implicant ,Second_Implicant,used);
273          for(int i=0 ; i<used.size() ; i++){
274              if(!used[i] && !Implicant_done){
275                 Second_Implicant.push_back(Implicant[i]);
276              }
277              used[i]=0;
278           }
279          if(!Implicant_done){
280              Implicant.resize(Second_Implicant.size() );
281              Implicant = Second_Implicant;
282              Second_Implicant.clear();
283          }
284     }
285 
286     sort(Implicant.begin(), Implicant.end());
287     reverse(Implicant.begin(), Implicant.end());
288     part1_answer.resize(Implicant.size());
289     part1_answer = Implicant;
```

Decimal 轉 binary

```
32    string dec2bin(int n){
33          string binary_str;
34          for(int i=0 ; i < num_var ; i++){
35              binary_str += n%2 +'0';
36              n=n/2;
37          }
38          reverse(binary_str.begin(), binary_str.end());
39          return binary_str;
40    }
```

2 個 binary 字串判斷是否 merge

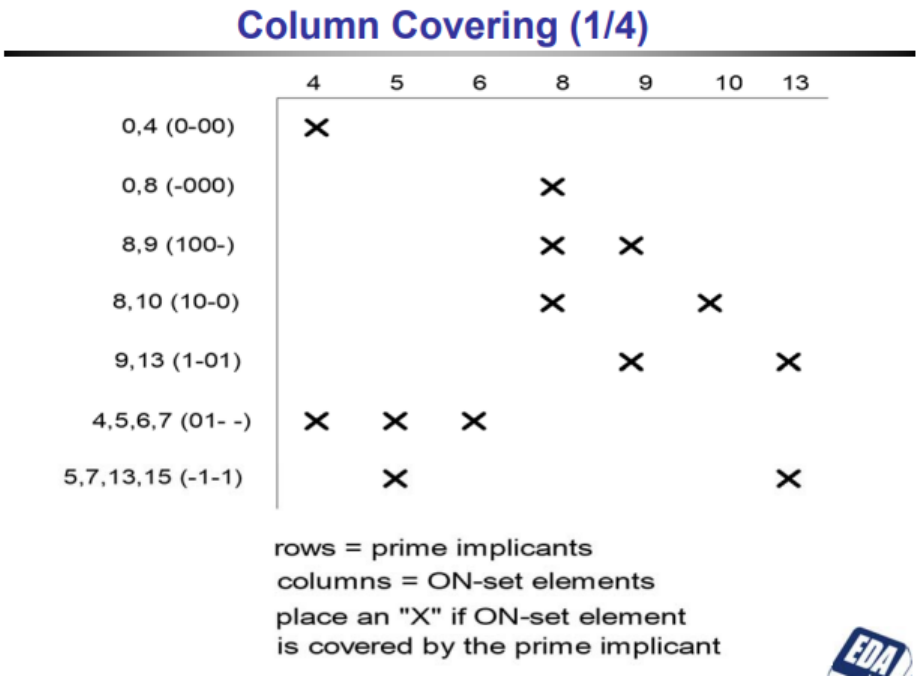
```cpp
bool merge(const string a,const string b,vector<string>& Second_Implicant  ){
    string str;
    string c=a;
    int count1=0;
    int position_;
    bool merged =  false;
    bool same =0 ;
    for(int i=0 ; i < a.length() ; i++){
        if((a[i]!=b[i])){
            count1 = count1+1;
            position_ = i;
        }
    }
    if(count1==1){
        c[position_] = '-';
        for(int i=0 ; i<Second_Implicant.size() ; i++){
            if(c == Second_Implicant[i])
                same = 1;
        }
        if(!same){
            Second_Implicant.push_back(c);
            merged = true;
        }
    }
    return merged;
}
```

```cpp
bool Implicant_caculation (vector<string>& Implicant,vector<string>& Second_Implicant,vector<bool>& used){
    bool done =1;
    for(int i =0; i<Implicant.size() ; i++){
            for(int  j=i+1; j<Implicant.size();j++){
                merged = merge(Implicant[i],Implicant[j],Second_Implicant );
                if(merged){
                    used[i] = 1;
                    used[j] = 1;
                    done = 0;
                }
            }
        }
    return done;
}
```

計算 part2 (mc)

先算 column 在 row 重複迭代直到 cover 到所有 ON.set

```
288 ////////////////////////////////////////////////////////////////////
289 //                            part 2                              //
290 ////////////////////////////////////////////////////////////////////
291
292     cal_column_sum(Implicant , ON_set_binary  , answer,column_sum );
293     reduced_table(answer , Implicant , ON_set_binary);
294     string row_max;
295     vector<string> remove;
296     int i=0;
297     while(ON_set_binary.size()!=0){
298         row_max =row_sum(Implicant , ON_set_binary  , answer);
299         answer.push_back(row_max);
300         remove.push_back(row_max);
301         reduced_table(remove , Implicant , ON_set_binary);
302         remove.clear();
303         i= i+1;
304     }
305     sort(answer.begin(), answer.end());
306     reverse(answer.begin(), answer.end());
```



先算 table column 的合為 1 的提出來 (必選的 Implicant)

```
101  void cal_column_sum(vector<string>&Implicant ,vector<string>& ON_set_binary
102      bool same;
103      string temp_imp;
104      int sum = 0 ;
105      bool answer_ext = false;
106      for(int i=0 ; i<ON_set_binary.size() ; i++){
107          sum = 0;
108          for(int j=0 ; j<Implicant.size() ; j++){
109              same = same_funtion( Implicant[j] , ON_set_binary[i]);
110              if(same==1){
111                  sum = sum+1;
112                  temp_imp = Implicant[j];
113              }
114          }
115          answer_ext = false;
116          column_sum.push_back(sum);
117          if(sum==1){
118              for(int i=0 ; i<answer.size();i++){
119                  if(temp_imp == answer[i])
120                      answer_ext=1;
121              }
122              if(!answer_ext){
123                  answer.push_back(temp_imp);
124              }
125          }
126      }
127  }
```

判斷 Impliacant 是否 ON.set 的 same_function

```
84  bool same_funtion(const string a ,const string b){
85      bool same = true;
86      for(int i=0 ; i<a.length() ; i++){
87          if((a[i]!=b[i])&(a[i]!='-')&(b[i]!='-'))
88              same = false;
89      }
90      return same;
91  }
92
```

選出第一輪 Implicant 簡化 table

```cpp
void remove_vector( vector<string>& ON_set_binary , string target) {
    ON_set_binary.erase(remove(ON_set_binary.begin(), ON_set_binary.end(), target), ON_set_binary.end());

}
void reduced_table(vector<string>& target , vector<string>&Implicant , vector<string>& ON_set_binary  ){
    bool same = 0;
    vector<string> remove_string;
    for(int i=0 ; i<target.size() ; i++){
        for(int j=0 ; j<ON_set_binary.size() ; j++){
            same = same_funtion( target[i] , ON_set_binary[j]);
            if(same)
                remove_string.push_back(ON_set_binary[j]);
        }
        for(int i=0 ; i<remove_string.size() ; i++){
            remove_vector(ON_set_binary,remove_string[i] );
        }
        remove_string.clear();
         remove_vector(Implicant,target[i] );
    }
}
```

計算列的和 (選涵蓋最多 ON.set 如果相同就優先選 don’care 多的)

```cpp
string row_sum(vector<string>&Implicant ,vector<string>& ON_set_binary  ,vector<string>& answer
    bool same;
    string temp_imp;
    int sum = 0 ;
    int temp_row = 0;
    int temp_dash_num = 0;
    int new_dash_num =0;
    for(int i=0 ; i<Implicant.size() ; i++){
        sum = 0;
        new_dash_num=0;
        for(int j=0 ; j<ON_set_binary.size() ; j++){
           same = same_funtion( Implicant[i] , ON_set_binary[j]);
            if(same==1){
              sum = sum+1;
            }
        }
         if(sum >temp_row){
            temp_imp =  Implicant[i];
            temp_row = sum;
            temp_dash_num = num_dash(Implicant[i]);
        }
        else if(sum == temp_row){
            new_dash_num = num_dash(Implicant[i]);
            if(new_dash_num > temp_dash_num){
                temp_imp =  Implicant[i];
                temp_dash_num = new_dash_num;
            }
        }
    }
    return temp_imp;
}
```

計算 “–“ 的數量 (don’t care)

```
93    int num_dash(const string a){
94        int num = 0;
95        for(int i=0 ; i<a.length() ; i++){
96            if(a[i]=='-')
97                num = num+1;
98        }
99        return num;
100   }
```

把答案 1-01 轉成 AC’D 、算 literal 的數量

```
184   string binary_2_ABC(string answer ) {
185       string answer_ABC;
186       for(int i=0; i<answer.length();i++){
187           if(answer[i]=='1'){
188               answer_ABC += 'A'+i;
189           }
190           else if(answer[i]=='0'){
191               answer_ABC += 'A'+i;
192               answer_ABC += 39;
193           }
194       }
195       return answer_ABC;
196   }
197
198   int count_literal(vector<string>& answer ) {
199       int literal = 0;
200        string temp ;
201        for(int i=0; i<answer.size();i++){
202            temp = answer[i];
203            for(int j=0; j<temp.length();j++){
204                if((temp[j] =='1')||(temp[j] =='0')){
205                    literal ++;
206                }
207            }
208        }
209       return literal;
210   }
```

Output 寫檔

```
317     // output
318     int out_num = 0;
319     fout << ".p " << answer_ABC.size() << endl;
320     for(int i=0 ; i<answer_ABC.size() ; i++){
321         out_num++;
322         fout << answer_ABC[i] ;
323         fout   << endl;
324         if(out_num>=15){
325             break;
326         }
327      }
328     fout   << endl;
329 
330     fout << ".mc " << final_answer_ABC.size() << endl;
331     for(int i=0 ; i<final_answer_ABC.size() ; i++){
332         fout << final_answer_ABC[i] ;
333         fout  << endl ;
334      }
335      fout  << "literal=" <<literal  ;
336 
337     fout.close();
338 
339       return 0;
```