Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: М. С. Гаврилов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

Задача: При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом; оценить время выполнения алгоритма и объем затрачиваемой оперативной памяти. Перед выполнением задания необходимо обосновать применимость метода динамического программирования.

Разработать программу на языке C или C++, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания:

Задан прямоугольник с высотой n и шириной m, состоящий из нулей и единиц. Найдите в нём прямоугольник наибольшой площади, состоящий из одних нулей.

Формат входных данных
В первой строке заданы 1 n 500 и 1 m 500. В следующих n строках записаны по m символов 0 или 1 — элементы прямоугольника.

Формат результата
Необходимо вывести одно число – максимальную площадь прямоугольника из одних нулей.

1 Описание и исходный код

Выполняется обход по строкам. В моем случае строка обходится с конца в начало, однако это не принципиально. Ведется подсчет высот столбцов из нулей, оканчивающихся в обозреваемой строке. Пока высота каждого следующего столбца выше чем высота предыдущего количество столбцов с той или иной высотой записывается в специальный массив высот. Как только высота падает (прекращается "поле действия"высот h $>$ h текущая), то площади прямоугольников, образованных всеми прекратившими свое действие высотами, подсчитывается. Наибольшая площадь записывается в соответствующую переменную.

Класс ячейки матрицы:

```
1
2  struct TCell {
3      int Len; //   .
4      int Vlen; // .  ,
5
6
7      TCell() {
8          Len = 0;
9          Vlen = 0;
10     }
11
12     TCell(int l, int v) {
13         Len = l;
14         Vlen = v;
15     }
16 };
```

Основная функция. В ходе обработки количество столбцов из нулей определенной высоты записывается в массив высот, если последовательность одинаковых высот (высоты n) прерывается более малой высотой, то смотрится количество столбцов, считается площадь получившегося прямоугольника и количество столбцов высоты n прибавляется к количеству следующей по размеру высоте.

```
1
2  int main() {
3      int m, n;
4      std::cin >> n >> m;
5      if(m == 0 || n == 0) {
6          printf("0\n");
7          return 0;
8      }
9      std::vector<std::vector<char>> matrix(n);
10
11     //
12     for (int i = 0; i < n; ++i) {
13         for (int j = 0; j < m; ++j) {
```

```
            char a = 0;
            std::cin >> a;
            if (a == '0') {
                a = 0;
            }
            else {
                a = 1;
            }
            matrix[i].push_back(a);
        }
    }

    //
    //

    //
    std::vector<TCell> prev(m, TCell());

    if(m > 0)prev[m - 1] = TCell();
    for (int i = m - 2; i >= 0; --i) {
        prev[i] = TCell();
    }

    int Sm = 0;
    //
    for (int k = 0; k < n; ++k) {
        std::vector<TCell> next(m, TCell());
        if(m > 0) next[m - 1] = ((matrix[k][m - 1] == 1) ? TCell() : TCell(1, 1));
        for (int i = m - 2; i >= 0; --i) { // .

            if (matrix[k][i] == 0) {
                if (next[i + 1].Len == 0) {
                    next[i] = TCell(1, 1);
                }
                else {
                    next[i] = next[i + 1];
                    next[i].Len++;
                }

            }
            else {
                next[i] = TCell();
            }

        }

        std::vector<int> heighths(n + 1, 0);
        std::list<int> hm;
        if(DEBUG) printf("next row! (Sn = %d)\n", Sm);
```

```
63          for (int i = m - 1; i >= 0; --i) {//
64              //
65
66              if (next[i].Len > 0 && prev[i].Len > 0) {
67                  next[i].Vlen = prev[i].Vlen + 1;
68              }
69
70              if (i == m - 1) {
71                  if (heighths[next[i].Vlen] == 0 && next[i].Vlen != 0) {
72                      hm.push_back(next[i].Vlen);
73                  }
74                  heighths[next[i].Vlen]++;
75                  continue;
76              }
77
78              //
79              if (next[i].Vlen >= next[i + 1].Vlen && next[i].Vlen != 0) {
80                  if (heighths[next[i].Vlen] == 0) {
81                      hm.push_back(next[i].Vlen);
82                  }
83                  heighths[next[i].Vlen]++;
84              }
85              else {
86                  if (DEBUG) printf("\t |fall of height from %d to %d\n", next[i + 1].Vlen
                        , next[i].Vlen);
87                  if (heighths[next[i + 1].Vlen] * next[i + 1].Vlen > Sm) {
88                      Sm = heighths[next[i + 1].Vlen] * next[i + 1].Vlen;
89                  }
90                  //   ,  ,    -  -  . (    )
91                  //  (  )
92
93                  if(hm.size() != 0) {
94                      hm.pop_back();
95                      if(DEBUG)if(hm.size() > 0)printf("\t|new top = %d\n", hm.back());
96                  }
97
98                  bool adedNow = false;
99                  if(hm.size() != 0 && hm.back() > next[i].Vlen) { // - .
100                     heighths[hm.back()] += heighths[next[i + 1].Vlen];
101                 }
102                 else {
103                     heighths[next[i].Vlen] += heighths[next[i + 1].Vlen];
104                     adedNow = true;
105                 }
106
107                 heighths[next[i + 1].Vlen] = 0;
108                 //  ,
109                 while (!hm.empty() && hm.back() > next[i].Vlen) {
110
```

```
111                     adcSteps++;
112                     if (heighths[hm.back()] * hm.back() > Sm) {
113                         Sm = heighths[hm.back()] * hm.back();
114                     }
115 
116                     int j = hm.back();
117                     //  ,
118                     hm.pop_back();
119                     if (!hm.empty() && hm.back() > next[i].Vlen) {
120                         //       ,   ,
121                         //      .
122                         heighths[hm.back()] += heighths[j];
123                         if(DEBUG) printf("\t |count transfered from %d to %d\n",j,hm.back
                                ());
124                     }
125                     else {
126                         //     ,      .
127                         if(DEBUG)if (!hm.empty()) printf("\t |count transfered from %d to
                                %d\n",j,next[i].Vlen);
128                         heighths[next[i].Vlen] += heighths[j];
129                         adedNow = true;
130                     }
131                     heighths[j] = 0;
132                 }
133 
134                 //
135                 if ((heighths[next[i].Vlen] == 0 || adedNow) && next[i].Vlen != 0 && hm.
                        back()!=next[i].Vlen) {
136                     hm.push_back(next[i].Vlen);
137                 }
138                 heighths[next[i].Vlen]++;
139             }
140 
141             if(DEBUG) {
142                 for(int v=0;v<m+1;++v) {
143                     printf("%d ",heighths[v]);
144                 }
145                 printf("||%d(%d)\n",(int)hm.size(),hm.size() != 0 ? hm.back() : -1);
146             }
147 
148         }
149 
150         if (DEBUG) {
151             printf("|");
152             for (int j = 0; j < n + 1; ++j) {
153                 printf("%d ", heighths[j]);
154             }
155             printf("|\n");
156             if (!hm.empty()) printf("| %d <hmm|\n", hm.back());
```

```
157             printf("row fin (Sn = %d,hm-B = %d)\n", Sm, (!hm.empty() ? hm.back() : -1))
                    ;
158         }
159
160         // ,   , ,
161         //
162         while (!hm.empty()) {
163
164             adcSteps++;
165             if (heighths[hm.back()] * hm.back() > Sm) {
166                 Sm = heighths[hm.back()] * hm.back();
167             }
168
169             int j = hm.back();
170             //  ,
171             hm.pop_back();
172             if(!hm.empty()) heighths[hm.back()] += heighths[j];
173             heighths[j] = 0;
174         }
175
176         prev = next;
177     }
178
179     printf("%d\n",Sm);
180     if(DEBUG)printf("ad: %d\n", adcSteps);
181
182 }
```

2 Доказательство корректности и тест производительности

Динамическое программирование используется в решении на этапе определения высот столбцов из нулей для каждой из строк. При рассмотрении очередной строки для каждой ячейки матрицы значение высоты определяется как увеличенное на единицу значение высоты в соответствующей ячейки с предыдущей строки если в текущей ячейки 0 или равно нулю, если в ячейке 1. Без динамического программирования значения высот надо было бы вычислять заново каждый раз, что повысило бы сложность отыскания высот для каждой клетки матрицы с O(m*n) до O(m*n*h) где h - среднее значение высоты столбцов из нулей.

Сложность отыскания высот O(m*n), цикл, который выполняется в случае если высота следующего столбца меньше предыдущего не может пройти больше шагов, чем было прервано высот, а значит в худшем случае выполняет m шагов (если все m ячеек текущего ряда это нули и высота каждой следующей больше высоты предыдущей. Однако всякий раз, когда высота столбцов возрастает цикл не вызывается, а значит суммарное число шагов, которые цикл пройдет за время рассмотрения ряда не превышает длину ряда (m) и выполнение цикла асимптотику не ухудшает.

```
max@max-Swift:~/Рабочий стол/ДА/lab7/DA_lab_7-main/deb$ ./gen.exe 1000 >testx
max@max-Swift:~/Рабочий стол/ДА/lab7/DA_lab_7-main/deb$ ./lab7.exe <testx
test complete| 221ms
21
max@max-Swift:~/Рабочий стол/ДА/lab7/DA_lab_7-main/deb$ ./gen.exe 2000 >testx
max@max-Swift:~/Рабочий стол/ДА/lab7/DA_lab_7-main/deb$ ./lab7.exe <testx
test complete| 850ms
24
max@max-Swift:~/Рабочий стол/ДА/lab7/DA_lab_7-main/deb$ ./gen.exe 4000 >testx
max@max-Swift:~/Рабочий стол/ДА/lab7/DA_lab_7-main/deb$ ./lab7.exe <testx
test complete| 3602ms
28
max@max-Swift:~/Рабочий стол/ДА/lab7/DA_lab_7-main/deb$
```

Программа gen.exe получает на вход параметр n и генерирует матрицу из нулей и единиц с размерами сторон от n до n + n/10. Видно, что при увеличении размеров матрицы (m и n) вдвое время работы программы увеличивается в 4 раза, что соответствует асимптотике O(m*n).

3 Выводы

В ходе работы над этой лабораторной работой я получил опыт использования на практике динамического программирования. С помощью динамического программирования я написал программу, которая решает задачу поиска наибольшего по размеру прямоугольника из нулей куда быстрее чем классическим перебором всех возможных вариантов, который, проверяя все возможные прямоугольники, начинающиеся из каждого нуля работал бы за O(m*n*h*l), где h и l – это средняя высота и средняя длина последовательностей нулей в матрице.

Список литературы

[1] *Динамическое программирование — Википедия*
URL: `https://ru.wikipedia.org/wiki/Динамическое_программирование` (дата обращения: 11.03.2021)

[2] *Динамическое программирование — Викиконспекты.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Динамическое_программирование` (дата обращения: 14.03.2021)