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. А. Бронников
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2019

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 зададния.

Вариант задания: Задана матрица натуральных чисел A размерности nm. Из текущей клетки можно перейти в любую из 3-х соседних, стоящих в строке с номером на единицу больше, при этом за каждый проход через клетку (i, j) взымается штраф $A_{i,j}$. Необходимо пройти из какой-нибудь клетки верхней строки до любой клетки нижней, набрав при проходе по клеткам минимальный штраф.

Входные данные: Первая строка входного файла содержит в себе пару чисел $2n1000$ и $2m1000$, затем следует n строк из m целых чисел.

Выходные данные: Необходимо вывести в выходной файл на первой строке минимальный штраф, а на второй - последовательность координат из n ячеек, через которые пролегает маршрут с минимальным штрафом.

1 Описание

Как сказано в [2]: «Оптимальная подструктура в динамическом программировании означает, что оптимальное решение подзадач меньшего размера может быть использовано для решения исходной задачи.».

«В общем случае мы можем решить задачу, в которой присутствует оптимальная подструктура, проделывая следующие три шага.

1. Разбиение задачи на подзадачи меньшего размера.
2. Нахождение оптимального решения подзадач рекурсивно, проделывая такой же трехшаговый алгоритм.
3. Использование полученного решения подзадач для конструирования решения исходной задачи.

Подзадачи решаются делением их на подзадачи ещё меньшего размера и т. д., пока не приходят к тривиальному случаю задачи, решаемой за константное время (ответ можно сказать сразу). »[2]

Для решения задачи будем использовать динамическое программирование, так как в этой задаче можно послеовательно посчитать ответ для всех ячеек массива, на основании результатов предыдущих ячеек. В противном случае, мы бы получили перебор значений с огромной сложностью, не меньше чем $O(n^3)$, когда как динамическое программирование решает эту проблему за $O(n^2)$

В данной задаче мы будем делать проход снизу вверх(так как сумма коммутативна) последовательно считая наименне возможный суммарный штраф попадания в клетку массива для каждой из клеток на основе суммы штрафа предыдущих позиций. Для каждой клетки рассматриваается три смежных из позиций снизу и выбирается наименьший штраф в сумме с фтрафом посещения текущей клетки. Далее выбирается наиболее оптимальный штраф и алгоритм идет сверху вниз, делая обратный проход, выбирая наименьшие суммы штрафов и выводя их позиции на экран.

2 Исходный код

В массиве A будем хранить массив, для которого вычисляем ответ, а в массиве B будем хранить результаты предыдущих вычислений для последующего использования динамически.

```
#include <iostream>

using namespace std;

long long min_of_three(long long a1, long long a2, long long a3){
  if(a1 < a2){
    if(a1 < a3){
      return a1;
    }else{
      return a3;
    }
  }else{
    if(a2 < a3){
      return a2;
    }else{
      return a3;
    }
  }
}

int main(){
  short unsigned n, m;
  cin >> n >> m;
  if(n > 1000 || n < 2 || m > 1000 || m < 2){
    return 0;
  }
  long int* A = (long int*)malloc(sizeof(long int) * n * m); // n*m*4
  long long int* B = (long long int*)malloc(sizeof(long long int) * n * m);
  short unsigned k = 0, j = 0, i = 0;
  for(; i < n; ++i){
    for(j = 0; j < m; ++j){
      cin >> A[i*m + j];
    }
  }

  for(j = 0; j < m; ++j){
    B[(n-1)*m + j] = A[(n - 1)*m + j];
  }
  for(i = n - 2; ; --i){
    B[i*m] = min_of_three(B[(i+1)*m], B[(i+1)*m], B[(i+1)*m + 1]) + A[i*m];
    for(j = 1; j < m - 1; ++j){
```

```
42      B[i*m + j] = min_of_three(B[(i+1)*m + j-1], B[(i+1)*m + j], B[(i+1)*m + j+1]) + A
            [i*m + j];
43    }
44    B[i*m + m - 1] = min_of_three(B[(i+1)*m + m-2], B[(i+1)*m + m-1], B[(i+1)*m + m-1])
          + A[i*m + m-1];
45    if(!i){
46      break;
47    }
48  }
49  for(j = 1; j < m; ++j){
50    if(B[j] < B[k]){
51      k = j;
52    }
53  }
54  cout << B[k] << '\n' << "(1," << k + 1 << ')';
55  for(i = 1; i < n; ++i){
56    if(k && B[i*m + k-1] < B[i*m + k]){
57      if(k < m - 1 && B[i*m + k-1] > B[i*m + k+1]){
58        ++k;
59      }else{
60        --k;
61      }
62    }else{
63      if(k < m - 1 && B[i*m + k+1] < B[i*m + k]){
64        ++k;
65      }
66    }
67    cout << ' ' << '(' << i+1 << ',' << k+1 << ')';
68  }
69  cout << endl;
70  free(B);
71  free(A);
72  return 0;
73 }
```

$min_o f_t ree - , 3.$

3 Консоль

В test.cpp находится генератор тестов для программы main.cpp.

```
(base) max@max-X550CC:~/DA/lab7$ g++ -std=c++17 -Wall -pedantic -o test test.cpp
(base) max@max-X550CC:~/DA/lab7$ ls
da_lab7.pdf  file  main.cpp  test  test.cpp
(base) max@max-X550CC:~/DA/lab7$ cat file
8 6
-788183269 151914327 440813985 -673082500 -442254918 172775084
-663563946 -697133726 135503011 330119723 -244357832 527698909
-176719602 -485110005 567456246 -432603585 -927451663 270715652
617179657 966422798 187581368 -298028983 -895958129 -821279507
-584686695 -290408410 -957696749 -385465226 -959419273 999978574
865921644 879510457 -699186263 974986161 -474026183 732772598
-438227855 -368772301 894281532 -990333248 -914681831 -930588832
862197611 299579672 -662730845 666896859 -202159017 -863756730
(base) max@max-X550CC:~/DA/lab7$ g++ -std=c++17 -Wall -pedantic -o main main.cpp
(base) max@max-X550CC:~/DA/lab7$ ./main <file
-5968641142
(1,4) (2,5) (3,5) (4,5) (5,5) (6,5) (7,6) (8,6)
(base) max@max-X550CC:~/DA/lab7$ exit
```

4 Выводы

Благодаря седьмой лабораторной работе по курсу «Дискретный анализ», я, наконец, узнвл что такое динамическое программирование и где оно применяется. Этот метод по своей структуре оказался проще, чем я подозревал, однако, несмотря на это, он применяется во многих областях.

Эта лабораторная работа оказалась одрной из самых простых за курс. Наибольшие неудобсва мне в доставила лишь моя невнимательность, из-за которой у меня происходило переполнение в числовых переменных. Тем не менее мне было очень интересно придумать алгоритм для решения задачи, а не пользоватся уже готовым, как мы это делали в предыдущих работах. В процессе решения я понял, что придумать рекурсивную формулу не такая уж и тривиальная задача.

Я рад, что теперь в моем арсенале появился такой элегантный и мощный метод, как динамическое программирование, ведь без знаний о нем не может обойтись любой хороший программист. Я уверен, что этот метод мне не раз пригодится в будущем, однако задачи, которые мне придется с ним решать, будут намного труднее.

Список литературы

[1] Динамическое программирование. Классические задачи - Хабр
URL: https://habr.com/ru/post/113108/ (дата обращения: 16.05.2019).

[2] Динамическое программирование — Википедия.
URL: https://ru.wikipedia.org/wiki/Динамическое(: 16.05.2019).