Условие

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 5: Обход матрицы
Задана матрица натуральных чисел A размерности n x m. 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 $2 \leq n \leq 1000$ и $2 \leq m \leq 1000$, 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.

Описание алгоритма:

Очевидно, что минимальный штраф мы заплатим если перейдем в требуемую ячеку из ячейки в предыдущей строке с минимальным штрафом (если мы пойдем не из минимальной то путь не будет являться оптимальным так как можно выбрать штраф меньше). Таким образом, переходя из ячеек с минимальным штрафом в следующую, мы получим в последней строке значения минимального штрафа который необходимо заплатить для того чтобы туда добраться. Выбрав минимальный из них мы и получим ответ.

Путь можно восстановить если для каждой ячейки записывать откуда мы пришли, а после получения ответа идти в обратную сторону по получившимся ссылкам. Однако путь получится инвертированным и его придется либо выводить на экран рекурсивно, либо инвертировать еще раз, например записав в массив в обратном порядке. Но если решать задачу наоборот т. е. проходить снизу вверх, то решение останется прежним, а инвертированный путь к этой задаче будет нормальным ответом для исходной задачи.

Описание реализации:

Считываются размеры матрицы, сама матрица, и начинается поиск оптимального пути.

Для того чтобы сделать вывод ответа максимально простым будем идти не сверху вниз, а снизу вверх т. е. решается эквивалентная задача, но для "перевернутой" матрицы. Так же для получения оптимального пути заведем двумерный массив (назовем его массив переходов) той же размерности что и сама матрица. Каждый элемент массива может принимать значения: 1, 0 или -1, что соответствует смещению вправо на следующей строке, сохранению позиции и смещению влево.

Итак начинаем идти с предпоследней строчки: для каждого элемента строки выбираем один из 3-х (по условию) эл-тов находящейся ниже строки значение которого минимально, прибавляем это минимальное значение к значению текущего элемента, и записываем в массив переходов число соответствующее положению минимального элемента относительно текущего. Элементы находящиеся в начале и в конце строки рассматриваем отдельно, так как для них есть только два возможных перехода. Продолжаем для каждой строки.

После завершения данного процесса проходим по первой строке и находим минимальное значение. Это и будет та клетка откуда начинается оптимальный путь. Выводим значение этого элемента и его индекс в заданном формате, далее проходим по массиву переходов, спускаясь по строчкам вниз, сдвигая при этом индекс в строке на значение хранящееся в текущей ячейке матрицы переходов и также выводим индекс этой ячейки.

Затраты памяти: $O(nm)$ так как используется два массива заданной размерности.
Сложность: $O(nm)$ так как в процесе выполнения алгоритма мы проходимся по каждому элементу каждой строки за $O(nm)$, а затем проходим по оптимальному пути за $O(n)$.

Дневник отладки

При отправке решения на чекер система выдавала $WA\#1$. Оказалось, что ответ выдавался верный но из-за особенностей реализации тестирующей системы пробел который выводился в конце строки приводил к вердикту что файл с правильным ответом не совпадает с тем что выдала программа. После того как последний пробел в строке был убран, программа первый тест прошла.
Далее чекер выдал $WA\#11$. Оказалось что тип выбранный для элементов массива не вмещал такие большие тесты (так как в условии про ограничения на эл-ты матрицы ничего не сказано пришлось действовать вслепую). После заменты типа на более вместительный, 11 тест также был пройден.

Бенчмарк

```
#include <cstdio>

int n,m;
long long int** matrix;

long long int findAns(int begI, int begJ) {
```

```
  if (begI == n - 1) {
    return matrix[begI][begJ];
  }
  long long int min = findAns(begI + 1,begJ);
  if (begJ > 0) {
    long long int temp = findAns(begI + 1,begJ - 1);
    if (temp < min) {
      min = temp;
    }
  }
  if (begJ < m - 1) {
    long long int temp = findAns(begI + 1,begJ + 1);
    if (temp < min) {
      min = temp;
    }
  }
  return min + matrix[begI][begJ];
}

void printAns(int begI, int begJ) {
  if (begI == n - 1) {
    printf("(%d,%d)\n",begI + 1,begJ + 1);
    return;
  }
  long long int min = matrix[begI][begI];
  int best = begJ;
  if (begJ > 0) {
    if (matrix[begI + 1][begJ - 1] < min) {
      best = begJ - 1;
      min = matrix[begI + 1][begJ - 1];
    }
  }
  if (begJ < m - 1) {
    if (matrix[begI + 1][begJ + 1] < min) {
      best = begJ + 1;
    }
  }
  printf("(%d,%d) ",begI + 1, begJ + 1);
  printAns(begI + 1,best);
}

int main() {
    scanf("%d %d",&n,&m);
    matrix = new long long int*[n];
    for (int i = 0; i < n;++i) {
        matrix[i] = new long long int[m];
        for (int j = 0;j < m;++j) {
            scanf("%lld",matrix[i] + j);
        }
    }
    int best = 0;
```

```
    long long int ans = findAns(0,0);
    for (int i = 1; i < m; ++i) {
//      printf("%d\n",i);
      long long int temp = findAns(0,i);
      if (temp < ans) {
        ans = temp;
        best = i;
      }
    }
    printf("%lld\n",ans);
    printAns(0,best);
    for (int i = 0; i < n;++i) {
      delete [] matrix[i];
    }
    delete [] matrix;
    return 0;
}
```

```
sergey@Work-computer:~/Desktop/4_sem/lab7$ g++ extra.cpp
sergey@Work-computer:~/Desktop/4_sem/lab7$ ./a.out > orderedTest
sergey@Work-computer:~/Desktop/4_sem/lab7$ cat orderedTest
17 17
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238
239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289
sergey@Work-computer:~/Desktop/4_sem/lab7$ g++ -o slow.out slowMatrix.cpp
sergey@Work-computer:~/Desktop/4_sem/lab7$ g++ -o dynamic.out lab7.cpp
sergey@Work-computer:~/Desktop/4_sem/lab7$ time ./slow.out < orderedTest >
    slowRes

real  0m23.936s
user  0m23.908s
sys 0m0.004s
sergey@Work-computer:~/Desktop/4_sem/lab7$ time ./dynamic.out < orderedTest >
    dynamicRes

real  0m0.005s
user  0m0.004s
```

```
sys 0m0.004s
sergey@Work-computer:~/Desktop/4_sem/lab7$ diff -s slowRes dynamicRes
Files slowRes and dynamicRes are identical
sergey@Work-computer:~/Desktop/4_sem/lab7$ g++ testGen.cpp
sergey@Work-computer:~/Desktop/4_sem/lab7$ ./a.out > randomTest
sergey@Work-computer:~/Desktop/4_sem/lab7$ time ./slow.out < randomTest >
    slowRes

real  0m2.503s
user  0m2.496s
sys 0m0.000s
sergey@Work-computer:~/Desktop/4_sem/lab7$ time ./dynamic.out < randomTest >
    dynamicRes

real  0m0.006s
user  0m0.004s
sys 0m0.000s
sergey@Work-computer:~/Desktop/4_sem/lab7$ diff -s slowRes dynamicRes
2c2
< (1,10) (2,11) (3,11) (4,10) (5,10) (6,10) (7,9) (8,8) (9,7) (10,8) (11,9)
    (12,10) (13,9) (14,9) (15,8)
---
> (1,10) (2,10) (3,9) (4,8) (5,7) (6,8) (7,9) (8,8) (9,7) (10,8) (11,8) (12,7)
    (13,6) (14,6) (15,7)
sergey@Work-computer:~/Desktop/4_sem/lab7$ cat slowRes
5469
(1,10) (2,11) (3,11) (4,10) (5,10) (6,10) (7,9) (8,8) (9,7) (10,8) (11,9)
    (12,10) (13,9) (14,9) (15,8)
sergey@Work-computer:~/Desktop/4_sem/lab7$ cat dynamicRes
5469
(1,10) (2,10) (3,9) (4,8) (5,7) (6,8) (7,9) (8,8) (9,7) (10,8) (11,8) (12,7)
    (13,6) (14,6) (15,7)
sergey@Work-computer:~/Desktop/4_sem/lab7$ cat randomTest
15 15
1605 1437 1215 1914 1772 974 550 465 1177 308 1064 1758 1845 1623 22
904 43 1183 1103 111 1187 1436 1649 1473 639 43 947 763 1564 128
1036 1169 1917 252 1435 1690 1578 1985 155 755 645 1219 513 490 1195
888 1747 1238 71 1202 1349 1610 639 998 1083 1630 1393 30 393 957
510 1781 478 428 385 1913 118 1963 251 625 719 896 1845 1584 1739
1040 472 1486 630 544 688 332 154 1679 1682 1590 1309 1076 1620 54
385 131 188 1216 911 573 1481 1029 537 1732 1654 1608 981 1851 1192
720 1243 1665 558 226 561 1598 558 1067 1278 240 657 939 1668 630
1346 406 761 1534 1974 1672 107 1455 1053 644 1540 1059 252 521 911
1797 1593 506 1814 151 732 375 101 1642 1442 1731 235 452 671 1903
1082 17 661 1843 1903 635 1867 10 443 1272 1007 335 331 1259 856
1594 1408 801 101 1222 1304 1185 1597 1405 828 1392 1137 1415 1844 160
1670 926 529 332 1121 432 1319 1340 442 1762 612 1449 97 943 1061
1305 890 821 106 991 44 1762 528 1641 1520 1708 1385 657 1123 1581
1169 794 859 1698 1126 1980 482 445 1672 924 560 636 726 1009 1932
```

Во втором тесте ответы не совпали так как возможны случаи когда существует несколько

оптимальных траекторий и в силу разного построения алгоритмов они находят разные варианты оптимального пути.

Выводы

Решение задач методом динамического программирования не составляет особой сложности. Смысл этого подхода в том что задача сводится к такой же задаче (одной или нескольким), но меньшей размерности, и постепенно решая подзадачи мы находим требуемый ответ. Однако этот метод метод не всегда применим. Чаще всего он применяется в задачах связанных с оптимизацией или с такими задачи где подзадачи перекрываются и нет смысла решать их несколько раз.