Queens College, CUNY, Department of Computer Science

Numerical Methods

CSCI 361 / 761

Summer 2018

Instructor: Dr. Sateesh Mane

Project 1

due Monday July 9, 2018, 12:00 noon

- **NOTE: It is the policy of the Computer Science Department to issue a failing grade to any student who either gives or receives help on any test. A student caught cheating on any question in an exam, project or quiz will fail the entire course.**
- **Students who form teams to collaborate on projects *must inform the lecturer of the names of all team members* ahead of time, else the submissions will be classified as cheating and will receive a failing grade.**
- **Any problem to which you give two or more (different) answers receives the grade of zero automatically.**
- Please submit your solution via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
  1. The zip archive should have either of the names (CS211 or CS611):

     `StudentId_first_last_CS361_project1_July2018.zip`
     `StudentId_first_last_CS761_project1_July2018.zip`
  2. The archive should contain one "text file" named "Project1.[txt/docx/pdf]" and one cpp file per question named "Q1.cpp" and "Q2.cpp" etc.
  3. Note that not all questions may require a cpp file.
- **In all questions where you are asked to submit programming code, programs which display any of the following behaviors will receive an automatic F:**
  1. Programs which do not compile successfully (non-fatal compiler warnings are excluded).
  2. Array out of bounds, reading of uninitialized variables (including null pointers).
  3. Operations which yield NAN or infinity, e.g. divide by zero, square root of negative number, etc. *Infinite loops.*
  4. Programs which do NOT implement the public interface stated in the question.
- **In addition, note the following:**
  1. All debugging statements (for your personal testing) should be commented out.
  2. Program performance will be graded solely on the public interface stated in the questions.

You are not eligible for credit for this project if you did not pass Midterm 1.

General information

- **You are permitted to copy and use code supplied in the online lecture notes.**
- **64–bit computers**
  1. Hardware architecture will not be taken into account in the questions below.
  2. Solutions involving the writing of programs will not be judged if they work (or were written) on a 64–bit instead of a 32–bit computer.
- **The following refers to questions involving mathematical calculations.**
  1. **Value of π to machine precision on any computer.**
  2. Some compilers support the constant `M_PI` for π, in which case you can write

     ```
     const double pi = M_PI;
     ```

  3. If your compiler does not support `M_PI`, the value of π can be computed via

     ```
     const double pi = 4.0*atan2(1.0,1.0);
     ```

- **Ignore the statements above if they are not relevant for this document.**

1 Material to be used in later questions

- **This is material for use in other questions. Nothing to "calculate" here.**
- Form a set of eight digits $(d_1, \ldots, d_8)$ as follows.
- **Take the 8 digits of your student id and define $(d_1, \ldots, d_8)$ as follows:**

$$\begin{aligned} d_1 &= \text{digit 1 of student id}\,, \\ d_2 &= \text{digit 2 of student id}\,, \\ d_3 &= \text{digit 3 of student id}\,, \\ d_4 &= \text{digit 4 of student id}\,, \\ d_5 &= \text{digit 5 of student id}\,, \\ d_6 &= \text{digit 6 of student id}\,, \\ d_7 &= \text{digit 7 of student id}\,, \\ d_8 &= \text{digit 8 of student id}\,. \end{aligned} \tag{1.1}$$

- For example if your student id is 23054611, then

$$\begin{aligned} d_1 &= 2\,, \\ d_2 &= 3\,, \\ d_3 &= 0\,, \\ d_4 &= 5\,, \\ d_5 &= 4\,, \\ d_6 &= 6\,, \\ d_7 &= 1\,, \\ d_8 &= 1\,. \end{aligned} \tag{1.2}$$

- *For some student ids, it is possible that some of the digits may be zero. It is also possible that some of the digits may be equal.* Do not worry.
- For the student id 11111111, all the digits are equal.
- For the student id 33330000, four digits are zero and the other four are all equal to 3.

2 Question 2

- You are given input arrays a and b, both of length n and type `double`.
- Hence the array elements are `a[i], b[i]` for $i = 0, \ldots, n-1$.
- **Write functions to efficiently calculate the following sums of products:**

$$T_1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a[i]\, b[j]\,,$$

$$T_2 = \sum_{i=0}^{n-1} \sum_{j=0}^{i} a[i]\, b[j]\,.$$

- ***Note: return*** 0 ***if*** $n \leq 0$***.***
- The function signatures are

```
double sum_prod_ab1(int n, const double a[], const double b[];
double sum_prod_ab2(int n, const double a[], const double b[];
```

- **Efficiency: calculate the computational complexity of your algorithms.**
- In this context, the computational complexity is defined as follows.
  1. Write down the number of **floating-point additions** for `a[i]`, `b[j]`, etc.
  2. Write down the number of **floating-point multiplications.**
  3. End of loop tests and increments of loop variables i and j do not count.
- **Algorithms with a lower computational complexity will score a higher grade.**
- *Hint: If you do not immediately see the answer, it may help to work out a few simple cases "by hand" for example $n = 1$, $n = 2$, etc., to see the general pattern.*

3 Question 3

- You are given input arrays a, b and c, all of length n and all of type `double`.
- Hence the array elements are `a[i], b[i], c[i]` for $i = 0, \ldots, n-1$.
- **Write functions to efficiently calculate the following sums of products:**

$$S_1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} (-1)^{i+2j+3k} a[i]\, b[j]\, c[k]\,,$$

$$S_2 = \sum_{i=0}^{n-1} \sum_{j=0}^{i} \sum_{k=0}^{j} (-1)^{i+2j+3k} a[i]\, b[j]\, c[k]\,.$$

- ***Note: return*** 0 ***if*** $n \leq 0$***.***
- The function signatures are

```
double sum_prod1(int n, const double a[], const double b[], const double c[]);
double sum_prod2(int n, const double a[], const double b[], const double c[]);
```

- **Efficiency: calculate the computational complexity of your algorithms.**
- In this context, the computational complexity is defined as follows.
  1. Write down the number of **floating-point additions** for `a[i]`, `b[j]`, etc.
  2. Write down the number of **floating-point multiplications.**
  3. End of loop tests and increments of loop variables i, j and k do not count.
- **Algorithms with a lower computational complexity will score a higher grade.**
- *Hint: If you do not immediately see the answer, it may help to work out a few simple cases "by hand" for example $n = 1$, $n = 2$, etc., to see the general pattern.*
- *It may especially help to review the answer to Question 3.*

4 Question 4

- A **Dyck path** is defined as follows.
- First we define a two dimensional integer lattice with points (i, j) where $i \geq 0$ and $j \geq 0$.
- Also define integers $k \geq 1$ and $n \geq 1$.
- A **Dyck path** is a path on that integer lattice, which begins at the origin $(0, 0)$ and ends at the point with coordinates (n, kn).
  1. **A Dyck path has only horizontal steps (to the right) and vertical steps (upwards).**
  2. Therefore, at any node (i, j), a Dyck path can take a step to the right $i \to i + 1$ or a step up $j \to j + 1$.
  3. *Diagonal steps are not allowed.*
  4. The Dyck path **can touch but cannot pass above the line $y = kx$.**
  5. In other words, we must have $0 \leq i \leq n$, $0 \leq j \leq kn$ and $j \leq ki$.
- A graph of sample Dyck paths is displayed in Fig. 1 for $n = 4$ and $k = 1$.
  1. The paths begin at $(0, 0)$ and terminate at $(4, 4)$.
  2. The paths can touch but cannot cross above the line $y = x$, therefore always $0 \leq i \leq 4$, $0 \leq j \leq 4$ and $j \leq i$.
- A graph of Dyck paths is displayed in Fig. 2 for $n = 2$ and $k = 2$.
- Fig. 2 displays all the Dyck paths for $n = 2$ and $k = 2$ (totally three paths).
  1. The paths begin at $(0, 0)$ and terminate at $(2, 4)$.
  2. The paths can touch but cannot cross above the line $y = 2x$, therefore always $0 \leq i \leq 2$, $0 \leq j \leq 4$ and $j \leq 2i$.
- **Write a function to print Dyck paths to the console.**

```
int Dyck_path(int n, int k);
```

- The return value is the total number of Dyck paths. (For $n = 2$ and $k = 2$ the number is 3.)
- Print each path with the format below. We employ $n = 2$ and $k = 2$ as an example.

```
(0,0) (1,0) (2,0) (2,1) (2,2) (2,3) (2,4)
(0,0) (1,0) (1,1) (2,1) (2,2) (2,3) (2,4)
(0,0) (1,0) (1,1) (1,2) (2,2) (2,3) (2,4)
```

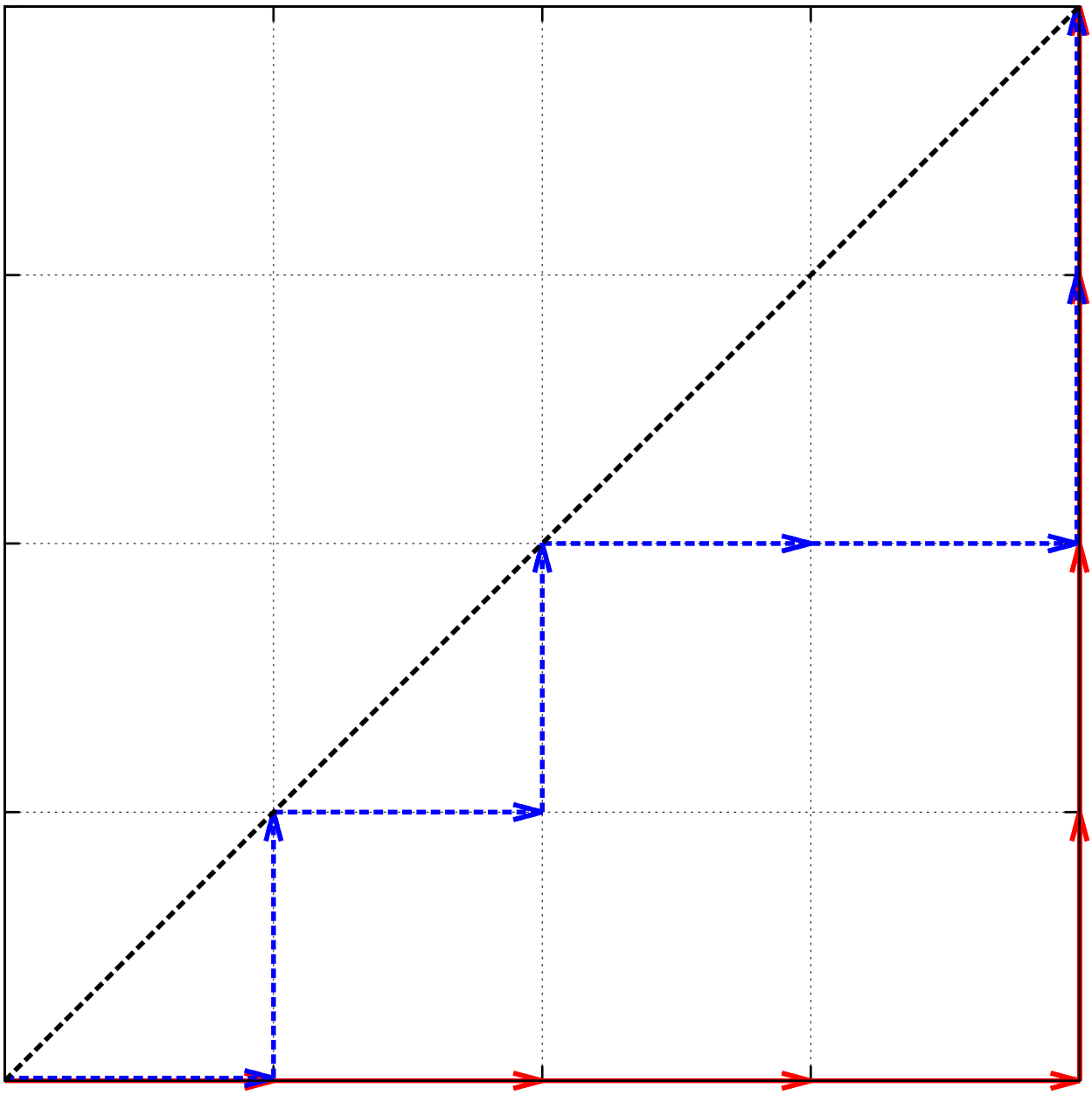

Figure 1: Graph of sample Dyck paths for $n = 4$ and $k = 1$.

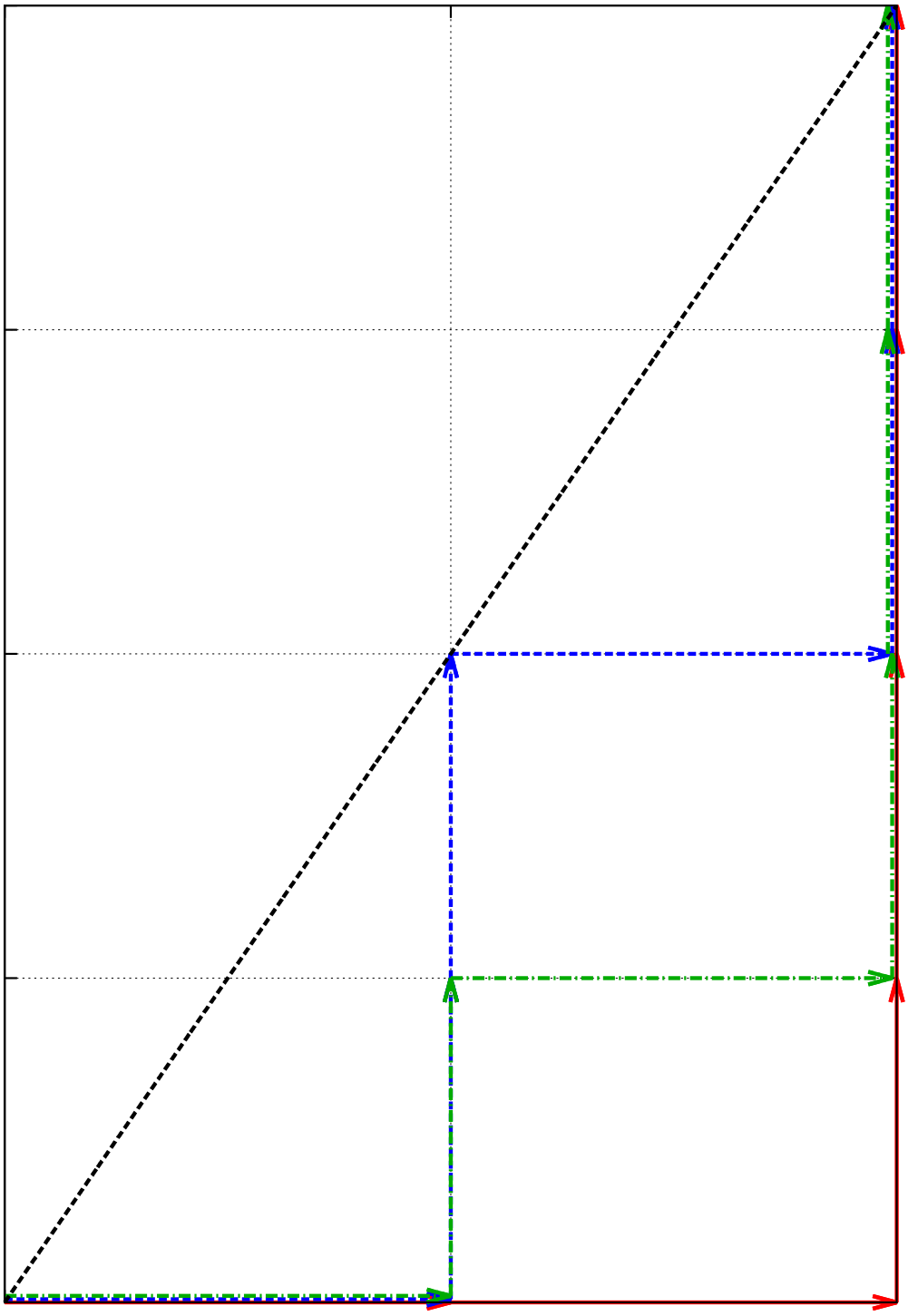

Figure 2: Graph of all Dyck paths for $n = 2$ and $k = 2$.