Programming Assignment № 1

Apoorva Tamaskar, Glasgow University 17/11/2017

Listing 1: Dynamic Programming with Memoisation

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <time.h>
5  #include <math.h>
6  #include <stdbool.h>
7
8  // global variables
9  enum {LCS, ED, SW, NONE} alg_type; // which algorithm to run
10 char *alg_desc; // description of which algorithm to run
11 char *result_string; // text to print along with result from algorithm
12 char *x, *y; // the two strings that the algorithm will execute on
13 char *filename; // file containing the two strings
14 int xLen, yLen, alphabetSize; // lengths of two strings and size of ↩
       alphabet
15 bool iterBool = false, recNoMemoBool = false, recMemoBool = false; // ↩
       which type of dynamic programming to run
16 bool printBool = false; // whether to print table
17 bool readFileBool = false, genStringsBool = false; // whether to read in ↩
       strings from file or generate strings randomly
18 // functions follow
19 long long int count=0;
20 // determine whether a given string consists only of numerical digits
21 bool isNum(char s[]) {
22     int i;
23     bool isDigit=true;
24     for (i=0; i<strlen(s); i++)
25         isDigit &= s[i]>='0' && s[i]<='9';
26     return isDigit;
27 }
28
29 // get arguments from command line and check for validity (return true if ↩
       and only if arguments illegal)
30 bool getArgs(int argc, char *argv[]) {
31     int i;
32     alg_type = NONE;
33     xLen = 0;
34     yLen = 0;
```

```
35      alphabetSize = 0;
36      for (i = 1; i < argc; i++) // iterate over all arguments provided (argument 0 is name of this module)
37          if (strcmp(argv[i],"-g")==0) { // generate strings randomly
38              if (argc>=i+4 && isNum(argv[i+1]) && isNum(argv[i+2]) && isNum(argv[i+3])) { // must be three numerical arguments after this
39                  xLen=atoi(argv[i+1]); // get length of x
40                  yLen=atoi(argv[i+2]); // get length of y
41                  alphabetSize = atoi(argv[i+3]); // get alphabet size
42                  genStringsBool = true; // set flag to generate strings randomly
43                  i+=3; // ready for next argument
44              }
45              else
46                  return true; // must have been an error with -g arguments
47          }
48          else if (strcmp(argv[i],"-f")==0) { // read in strings from file
49              if (argc>=i+2) { // must be one more argument (filename) after this)
50                  i++;
51                  filename = argv[i]; // get filename
52                  readFileBool = true; // set flag to read in strings from file
53              }
54              else
55                  return true; // must have been an error with -f argument
56          }
57          else if (strcmp(argv[i],"-i")==0) // iterative dynamic programming
58              iterBool = true;
59          else if (strcmp(argv[i],"-r")==0) // recursive dynamic programming without memoisation
60              recNoMemoBool = true;
61          else if (strcmp(argv[i],"-m")==0) // recursive dynamic programming with memoisation
62              recMemoBool = true;
63          else if (strcmp(argv[i],"-p")==0) // print dynamic programming table
64              printBool = true;
65          else if (strcmp(argv[i],"-t")==0) // which algorithm to run
66              if (argc>=i+2) { // must be one more argument ("LCS" or "ED" or "SW")
67                  i++;
68                  if (strcmp(argv[i],"LCS")==0) { // Longest Common Subsequence
69                      alg_type = LCS;
70                      alg_desc = "Longest Common Subsequence";
```

```
71                      result_string = "Length of a longest common ↩
                            subsequence is";
72                  }
73                  else if (strcmp(argv[i],"ED")==0) { // Edit Distance
74                      alg_type = ED;
75                      alg_desc = "Edit Distance";
76                      result_string = "Edit distance is";
77                  }
78                  else if (strcmp(argv[i],"SW")==0) { // Smith-Waterman ↩
                        Algorithm
79                      alg_type = SW;
80                      alg_desc = "Smith-Waterman algorithm";
81                      result_string = "Length of a highest scoring local ↩
                            similarity is";
82                  }
83                  else
84                      return true; // none of these; illegal choice
85              }
86              else
87                  return true; // algorithm type not given
88          else
89              return true; // argument not recognised
90          // check for legal combination of choices; return true (illegal) ↩
                if user chooses:
91          // - neither or both of generate strings and read strings from ↩
                file
92          // - generate strings with length 0 or alphabet size 0
93          // - no algorithm to run
94          // - no type of dynamic programming
95          return !(readFileBool ^ genStringsBool) || (genStringsBool && (↩
                xLen <=0 || yLen <= 0 || alphabetSize <=0)) || alg_type==NONE ↩
                || (!iterBool && !recMemoBool && !recNoMemoBool);
96  }
97
98  // read strings from file; return true if and only if file read ↩
        successfully
99  bool readStrings() {
100     // open file for read given by filename
101     FILE * file;
102     file = fopen(filename, "r");
103     // firstly we will measure the lengths of x and y before we read them ↩
            in to memory
104     if (file) { // file opened successfully
105         // first measure length of x
106         bool done = false;
107         int i;
108         do { // read from file until newline encountered
```

```
109             i = fgetc(file); // get next character
110             if (i==EOF) { // EOF encountered too early (this is first ↩
                    string)
111                 // print error message, close file and return false
112                 printf("Incorrect file syntax\n");
113                 fclose(file);
114                 return false;
115             }
116             if ((char) i=='\n' || (char) i=='\r') // newline encountered
117                 done = true; // terminate loop
118             else // one more character
119                 xLen++; // increment length of x
120         } while (!done);
121         // next measure length of y
122         if ((char) i=='\r')
123             fgetc(file); // get rid of newline character
124         done = false;
125         do { // read from file until newline or EOF encountered
126             int i = fgetc(file); // get next character
127             if (i==EOF || (char) i=='\n' || (char) i=='\r') // EOF or ↩
                    newline encountered
128                 done = true; // terminate loop
129             else // one more character
130                 yLen++; // increment length of y
131         } while (!done);
132         fclose(file);
133         // if either x or y is empty then print error message and return ↩
                false
134         if (xLen==0 || yLen==0) {
135             printf("Incorrect file syntax\n");
136             return false;
137         }
138         // now open file again for read
139         file = fopen(filename, "r");
140         // allocate memory for x and y
141         x = malloc(xLen * sizeof(char));
142         y = malloc(yLen * sizeof(char));
143         // read in x character-by-character
144         for (i=0; i<xLen; i++)
145             x[i]=fgetc(file);
146         i = fgetc(file); // read in newline between strings and discard
147         if ((char) i=='\r')
148             fgetc(file); // read \n character and discard if previous ↩
                    character was \r
149         // read in y character-by-character
150         for (i=0; i<yLen; i++)
151             y[i]=fgetc(file);
```

```
152         // close file and return boolean indicating success
153         fclose(file);
154         return true;
155     }
156     else { // notify user of I/O error and return false
157         printf("Problem opening file %s\n",filename);
158         return false;
159     }
160 }
161
162 // generate two strings x and y (of lengths xLen and yLen respectively) ↩
    uniformly at random over an alphabet of size alphabetSize
163 void generateStrings() {
164     // allocate memory for x and y
165     x = malloc(xLen * sizeof(char));
166     y = malloc(yLen * sizeof(char));
167     // instantiate the pseudo-random number generator (seeded based on ↩
        current time)
168     srand(time(NULL));
169     int i;
170     // generate x, of length xLen
171     for (i = 0; i < xLen; i++)
172         x[i] = rand()%alphabetSize +'A';
173     // generate y, of length yLen
174     for (i = 0; i < yLen; i++)
175         y[i] = rand()%alphabetSize +'A';
176 }
177
178 // free memory occupied by strings
179 void freeMemory()
180 {
181     free(x);
182     free(y);
183 }
184 //↩
```

```
185 //Returns max of 2 numbers
186 int my_max(int a, int b)
187 {
188   if(a>=b)
189   {
190     return a;
191   }
192   else
193   {
194     return b;
```

```
195   }
196 }
197 //Return min of 2 numbers.
198 int my_min(int a, int b)
199 {
200   if(a<=b)
201   {
202     return a;
203   }
204   else
205   {
206     return b;
207   }
208 }
209
210 //X,Y, xLen, yLen are global variables so not in arguement. Will carry out↩
      LCS algorithm, iterative method based.
211 void it_lcs(int* arr[])
212 {
213   int i,j,temp;
214   //everything 0
215   for (j=0; j<yLen; j++)
216   {
217     for (i=0; i<xLen; i++)
218     {
219       arr[j][i]=0;
220     }
221   }
222   //computing the values
223   for (j=1; j<=yLen; j++)
224     {
225       for (i=1; i<=xLen; i++)
226         {
227           if (x[i-1]==y[j-1])
228             {
229               arr[j][i]=arr[j-1][i-1]+1;
230             }
231           else
232             {
233               arr[j][i]=my_max(arr[j-1][i],arr[j][i-1]);
234             }
235         }
236     }
237 }
238 // number of digits in number+1. -1 case it for memoisation case.
239 int length_num(int i)
240 {
```

```
241   double x, res;
242   int temp;
243   if(i==-1)
244   {
245     return 2;
246   }
247   else
248   {
249     x=i;
250     res=log10(x);
251     temp=res+1;
252     return temp;
253   }
254 }
255
256 // Returns the number of digits+1 of the largest number in the array.
257 int max_log(int* arr[])
258 {
259   int ans=0,i,j,temp;
260   ans=length_num(my_max(xLen,yLen));
261   for (j=0;j<yLen;j++)
262   {
263     for(i=0;i<xLen;i++)
264     {
265       ans=my_max(ans,length_num(arr[j][i]));
266     }
267   }
268   return ans;
269 }
270
271 // prints a character, and gives sufficient spaces so that columns align ↩
      up
272 void print_char(int spaces, char c)
273 {
274   int temp;
275   printf("%c", c);
276   for(temp=0;temp<spaces;temp++)
277     {
278       printf(" ");
279     }
280 }
281 // prints a integer, and gives sufficients spaces so that columns align up↩
      . For -1 we will print '-' with sufficient spaces.
282 void print_int(int spaces, int i)
283 {
284   int temp, leftover;
285   if(i>0)
```

```
286     {
287       printf("%d",i);
288       leftover=spaces-length_num(i);
289       for (temp=0;temp<=leftover;temp++)
290       {
291         printf(" ");
292       }
293     }
294     else if(i==0)
295     {
296       printf("%d",0);
297       for(temp=0;temp<spaces;temp++)
298       {
299         printf(" ");
300       }
301     }
302     else if(i== -1)
303     {
304       printf("-");
305       for(temp=0;temp<spaces;temp++)
306       {
307         printf(" ");
308       }
309     }
310 }
311 // This function will print the array, 'space' tell how many spaces, so ↩
       that columns align
312 void print_array(int* arr[], int space)
313 {
314   int i,j;
315   print_char(space,' ');
316   print_char(space,' ');
317   print_char(space,' ');
318   for (i=0;i<=xLen;i++)
319   {
320     print_int(space,i%10);
321   }
322   printf("\n");
323   print_char(space,' ');
324   print_char(space,' ');
325   print_char(space,' ');
326   print_char(space,' ');
327   for(i=0;i<xLen;i++)
328   {
329     print_char(space,x[i]);
330   }
331   printf("\n");
```

```
332     print_char(space,' ');
333     print_char(space,' ');
334     for(i=0;i<=xLen+1;i++)
335     {
336       for (j=0;j<=space;j++)
337       {
338         printf("_");
339       }
340     }
341     printf("\n");
342     print_int(space,0);
343     print_char(space,' ');
344     print_char(space,'|');
345     for (i=0;i<=xLen;i++)
346     {
347       print_int(space,arr[0][i]);
348     }
349     printf("\n");
350     for(j=1;j<=yLen;j++)
351     {
352       print_int(space,j);
353       print_char(space,y[j-1]);
354       print_char(space,'|');
355       for(i=0;i<=xLen;i++)
356       {
357         print_int(space,arr[j][i]);
358       }
359       printf("\n");
360     }
361 }
362 // In case of LCS, this will print out an opitmal alignment(There are many↩
        alignments!).
363 void print_alignment(int* arr[],int l)
364 {
365   int for_x[l], for_y[l];
366   int i=xLen, j=yLen, len=l,temp;
367   //Finding the LCS
368   while(j>0 && i>0)
369   {
370     if(x[i-1]==y[j-1])
371     {
372       for_x[len-1]=i-1;
373       for_y[len-1]=j-1;
374       i--;
375       j--;
376       len--;
377     }
```

```
378       else
379       {
380         if(arr[j-1][i]==arr[j][i])
381         {
382           j--;
383         }
384         else
385         {
386           i--;
387         }
388       }
389     }
390     int to_print;
391     //Will work our way step by step, will print x then '|' then y.
392     //Line 1 print x with - at appropriate places.
393     for(i=0;i<l;i++)
394     {
395       if(i==0)
396       {
397         to_print=for_x[i]+for_y[i]+1;
398         for(j=0;j<for_x[i];j++)
399         {
400           printf("%c",x[j]);
401         }
402         for(j=for_x[i];j<to_print-1;j++)
403         {
404           printf("-");
405         }
406         printf("%c",x[for_x[i]]);
407       }
408       else
409       {
410         to_print=for_x[i]+for_y[i]-for_x[i-1]-for_y[i-1]-1;
411         for(j=for_x[i-1]+1;j<for_x[i];j++)
412         {
413           printf("%c",x[j]);
414         }
415         for(j=for_x[i]-for_x[i-1]-1;j<to_print-1;j++)
416         {
417           printf("-");
418         }
419         printf("%c",x[for_x[i]]);
420       }
421     }
422     to_print=xLen+yLen-for_x[l-1]-for_y[l-1]-2;
423     for(j=for_x[l-1]+1;j<xLen;j++)
424     {
```

```
425       printf("%c",x[j]);
426     }
427     for(j=for_y[l-1]+1;j<yLen;j++)
428     {
429       printf("-");
430     }
431     printf("\n");
432     //Line 2 print " " and "|"
433     for(i=0;i<l;i++)
434     {
435       if(i==0)
436       {
437         to_print=for_x[i]+for_y[i]+1;
438         for(j=0;j<to_print-1;j++)
439         {
440           printf(" ");
441         }
442         printf("|");
443       }
444       else
445       {
446         to_print=for_x[i]+for_y[i]-for_x[i-1]-for_y[i-1]-1;
447         for(j=0;j<to_print-1;j++)
448         {
449           printf(" ");
450         }
451         printf("|");
452       }
453     }
454     printf("\n");
455     //Line 3 print y with '-' at appropriate places
456     for(i=0;i<l;i++)
457     {
458       if(i==0)
459       {
460         to_print=for_x[i]+for_y[i]+1;
461         for(j=for_y[i];j<to_print-1;j++)
462         {
463           printf("-");
464         }
465         for(j=0;j<for_y[i];j++)
466         {
467           printf("%c",y[j]);
468         }
469         printf("%c",y[for_y[i]]);
470       }
471       else
```

```
472       {
473         to_print=for_x[i]+for_y[i]-for_x[i-1]-for_y[i-1]-1;
474         for(j=for_y[i]-for_y[i-1]-1;j<to_print-1;j++)
475         {
476           printf("-");
477         }
478         for(j=for_y[i-1]+1;j<for_y[i];j++)
479           {
480             printf("%c",y[j]);
481           }
482         printf("%c",y[for_y[i]]);
483       }
484     }
485     to_print=xLen+yLen-for_x[l-1]-for_y[l-1]-1;
486     for(j=for_x[l-1]+1;j<xLen;j++)
487       {
488         printf("-");
489       }
490     for(j=for_y[i-1]+1;j<yLen;j++)
491       {
492         printf("%c",y[j]);
493       }
494     printf("\n");
495   }
496
497   //x,y,xLen,yLen are global variables, so not arguements. this will carry ↩
        out Smith Watermann algorithm, iterative method.
498   void it_sw(int *arr[])
499   {
500     int i,j,temp;
501     //everything 0
502     for (j=0; j<=yLen; j++)
503       {
504         for (i=0; i<=xLen; i++)
505           {
506             arr[j][i]=0;
507           }
508       }
509    //computing the/*  values
510      for (j=1; j<=yLen; j++)
511        {
512          for (i=1; i<=xLen; i++)
513            {
514              if (x[i-1]==y[j-1])
515                {
516                  temp=arr[j-1][i-1];
517                  arr[j][i]=temp+1;
```

```
518                 }
519               else
520                 {
521                   temp=my_max(arr[j-1][i]-1,arr[j][i-1]-1);
522                   temp=my_max(arr[j-1][i-1],temp);
523                   temp=my_max(0,temp);
524                   arr[j][i]=temp;
525                 }
526             }
527       }
528  }
529 //Find the maximum elelemt of the array, used in SW algorithm.
530 int max_of_array(int* arr[])
531 {
532   int i,j,ans=0;
533   for(j=0;j<=yLen;j++)
534   {
535     for(i=0;i<=xLen;i++)
536     {
537       ans=my_max(ans,arr[j][i]);
538     }
539   }
540   return ans;
541 }
542 //x,y,xLen,yLen are global variables, so not arguements. this will carry ↩
      out Edit Distance algorithm, iterative method
543 void it_ed(int *arr[])
544 {
545   int i,j,temp;
546   //everything 0
547   for (j=0; j<=yLen; j++)
548     {
549       arr[j][0]=j;
550     }
551   for(i=0;i<=xLen;i++)
552     {
553       arr[0][i]=i;
554     }
555   //computing the/*  values
556   for (j=1; j<=yLen; j++)
557   {
558     for (i=1; i<=xLen; i++)
559     {
560       if (x[i-1]==y[j-1])
561       {
562         arr[j][i]=arr[j-1][i-1];
563       }
```

```
564         else
565         {
566           temp=my_min(arr[j-1][i],arr[j][i-1]);
567           temp=my_min(arr[j-1][i-1],temp);
568           arr[j][i]=temp+1;
569         }
570       }
571     }
572 }
573 //For recursive without memoisation algorithms, how will go about them is ↩
      we will use an array to store all the values, that array is initalised↩
       to 0 everywhere.
574 //Will call a void function so that entries are modified.
575
576
577 //This is implementation of Recursion without memoisation of LCS algorithm↩
      .
578 void r_lcs(int* arr[], int j, int i)
579 {
580   arr[j][i]=arr[j][i]+1;
581   if(i!=0 && j!=0)
582   {
583     if (x[i-1]==y[j-1])
584     {
585       r_lcs(arr,j-1,i-1);
586     }
587     else
588     {
589       r_lcs(arr,j,i-1);
590       r_lcs(arr,j-1,i);
591     }
592   }
593 }
594
595 //This is implementation of Recursion without memoisation of ED algorithm.
596 void r_ed(int* arr[], int j, int i)
597 {
598   arr[j][i]++;
599   if(i!=0 && j!=0)
600   {
601     if (x[i-1] == y[j-1])
602     {
603       r_ed(arr, j-1, i-1);
604     }
605     else
606     {
607       r_ed(arr, j, i-1);
```

```
608         r_ed(arr, j-1, i);
609         r_ed(arr, j-1, i-1);
610       }
611     }
612 }
613
614 //Finds the sum of all the entries in the matrix, which is used in ↩
      recursion without memoisation case.
615 int sum_of_entries(int *arr[])
616 {
617   int sum=0,j,i;
618   for(j=0;j<=yLen;j++)
619   {
620     for(i=0;i<xLen;i++)
621     {
622       sum=sum+arr[j][i];
623     }
624   }
625   return sum;
626 }
627
628 //For recursive with memoisation algorithms, how will go about them is we ↩
      will use an array to store all the values, that array is initalised to↩
       0 everywhere.
629 //Will call a void function so that entries are modified.
630
631 //Implementation of Recursion with memoisation version of LCS algorithm.
632 //This function will end with making value of m_v[j][i] genuine and the ↩
      related necessary changes.
633 void m_lcs(int* m_v[], int* m_p[], int* m_b, int j, int i)
634 {
635   if(is_valid(m_v,m_p,m_b,j,i,count)==0)
636   {
637     if(i==0 || j==0)
638     {
639       m_v[j][i]=0;
640       m_p[j][i]=count;
641       m_b[count]=i+(j*(xLen+1));
642       count++;
643     }
644     else
645     {
646       if(x[i-1]==y[j-1])
647       {
648         if(is_valid(m_v,m_p,m_b,j-1,i-1)==0)
649         {
650           m_lcs(m_v,m_p,m_b,j-1,i-1);
```

```
651             }
652             m_v[j][i]=m_v[j-1][i-1]+1;
653             m_p[j][i]=count;
654             m_b[count]=i+(j*(xLen+1));
655             count++;
656           }
657           else
658           {
659             if(is_valid(m_v,m_p,m_b,j,i-1)==0)
660             {
661               m_lcs(m_v,m_p,m_b,j,i-1);
662             }
663             if(is_valid(m_v,m_p,m_b,j-1,i)==0)
664             {
665               m_lcs(m_v,m_p,m_b,j-1,i);
666             }
667             m_v[j][i]=my_max(m_v[j-1][i],m_v[j][i-1]);
668             m_p[j][i]=count;
669             m_b[count]=i+(j*(xLen+1));
670             count++;
671           }
672       }
673     }
674 }
675
676 //This checks if given value at location [j][i] of m_v is genuine.
677 int is_valid(int* m_v[], int* m_p[], int* m_b, int j, int i)
678 {
679   int k=m_p[j][i];
680   int temp=m_b[k];
681   if(k<0||k>count)
682   {
683     return 0;
684   }
685   else
686   {
687     if(temp== i+(j*(xLen+1)))
688     {
689       return 1;
690     }
691     else
692     {
693       return 0;
694     }
695   }
696 }
697
```

```
698 //Implementation of Recursion with memoisation version of ED algorithm.
699 //This function will end with making value of m_v[j][i] genuine and the ↩
      related necessary changes.
700 void m_ed(int *m_v[], int *m_p[], int* m_b, int j, int i)
701 {
702   if(is_valid(m_v,m_p,m_b,j,i)==0)
703     {
704       if(i==0 && j!=0)
705         {
706           m_v[j][i]=j;
707           m_p[j][i]=count;
708           m_b[count]=i+(j*(xLen+1));
709           count++;
710         }
711       else if (j==0)
712         {
713           m_v[j][i]=i;
714           m_p[j][i]=count;
715           m_b[count]=i+(j*(xLen+1));
716           count++;
717         }
718       else if(i!=0 && j!=0)
719         {
720           if(x[i-1]==y[j-1])
721             {
722               if(is_valid(m_v,m_p,m_b,j-1,i-1)==0)
723                 {
724                   m_ed(m_v,m_p,m_b,j-1,i-1);
725                 }
726               m_v[j][i]=m_v[j-1][i-1];
727               m_p[j][i]=count;
728               m_b[count]=i+(j*(xLen+1));
729               count++;
730             }
731           else
732             {
733               if(is_valid(m_v,m_p,m_b,j,i-1)==0)
734                 {
735                   m_ed(m_v,m_p,m_b,j,i-1);
736                 }
737               if(is_valid(m_v,m_p,m_b,j-1,i)==0)
738                 {
739                   m_ed(m_v,m_p,m_b,j-1,i);
740                 }
741               if(is_valid(m_v,m_p,m_b,j-1,i-1)==0)
742                 {
743                   m_ed(m_v,m_p,m_b,j-1,i-1);
```

```
744                     }
745                 m_v[j][i]=my_min(m_v[j-1][i],m_v[j][i-1]);
746                 m_v[j][i]=my_min(m_v[j-1][i-1],m_v[j][i])+1;
747                 m_p[j][i]=count;
748                 m_b[count]=i+(j*(xLen+1));
749                 count++;
750               }
751           }
752       }
753
754 }
755
756 //↩
```

```
757 // main method, entry point
758 int main(int argc, char *argv[])
759 {
760     bool isIllegal = getArgs(argc, argv); // parse arguments from command ↩
        line
761   if (isIllegal) // print error and quit if illegal arguments
762         printf("Illegal arguments\n");
763     else
764   {
765     // int *it[yLen+1],*r[yLen+1],*m_v[yLen+1],*m_p[yLen+1];
766     int i,j;
767     int **it = (int **)malloc((yLen+1) * sizeof(int *));
768     int **r = (int **)malloc((yLen+1) * sizeof(int *));
769     int **m_v = (int **)malloc((yLen+1) * sizeof(int *));
770     int **m_p = (int **)malloc((yLen+1) * sizeof(int *));
771     for (i=0; i<yLen+1; i++)
772     {
773       it[i] = (int *)malloc((xLen+1) * sizeof(int));
774       r[i] = (int *)malloc((xLen+1) * sizeof(int));
775       m_v[i] = (int *)malloc((xLen+1) * sizeof(int));
776       m_p[i] = (int *)malloc((xLen+1) * sizeof(int));
777     }
778     int *m_b= (int*)malloc((xLen+1)*(yLen+1)*sizeof(int));
779     int space_it,space_r,space_m;
780     int temp;
781     long long int total;
782     double time=0,pro_1=0,pro_2=0;
783     /*
784     for (j=0; j<yLen+1; j++)
785     {
786       r[j] = (int *)malloc((xLen+1) * sizeof(int));
787       m_v[j] = (int *)malloc((xLen+1) * sizeof(int));
```

```
788         m_p[j] = (int *)malloc((xLen+1) * sizeof(int));
789         it[j] = (int *)malloc((xLen+1) * sizeof(int));
790       }
791       */
792           printf("%s\n", alg_desc); // confirm algorithm to be executed
793           bool success = true;
794       if (genStringsBool)
795               generateStrings(); // generate two random strings
796           else
797               success = readStrings(); // else read strings from file
798           if (success)
799       {
800         // do not proceed if file input was problematic
801               // confirm dynamic programming type
802               // these print commamds are just placeholders for now
803         // check if algorithm to execute if LCS
804         if(alg_type==LCS)
805         {
806           //do we have to print DP tables??
807           if(printBool)
808           {
809             //Method to execute the algorithm in is iterative
810             if(iterBool)
811             {
812               printf("Iterative version\n");;
813               clock_t start=clock();
814               it_lcs(it);
815               space_it=max_log(it);
816               printf("Length of a longest common subsequence is: %d\n",it[↩
                      yLen][xLen]);
817               printf("Dynamic programming table:\n");
818               print_array(it,space_it);
819               printf("\nOptimal alignment:\n");
820               print_alignment(it,it[yLen][xLen]);
821               clock_t stop =clock();
822               time = (stop-start)/CLOCKS_PER_SEC;
823               printf("\nTime taken: %f seconds\n\n",time);
824             }
825             //Method to execute the algorithm in is Recursion without ↩
                    memoisation
826             if(recNoMemoBool)
827             {
828               printf("Recursive version without memoisation\n");
829               clock_t start=clock();
830               for(j=0;j<=yLen;j++)
831                 {
832                   for(i=0;i<=xLen;i++)
```

```
833                     {
834                       r[j][i]=0;
835                     }
836                   }
837                 r_lcs(r,yLen,xLen);
838                 space_r=max_log(r);
839                 total=sum_of_entries(r);
840                 print_array(r,space_r);
841                 printf("Total number of times a table entry computed: %lld\n",↩
                        total);
842                 clock_t stop =clock();
843                 time = (stop-start)/CLOCKS_PER_SEC;
844                 printf("Time taken: %f seconds\n\n", time);
845               }
846               //Method to execute the algorithm in is recursion with ↩
                    memoisation
847               if(recMemoBool)
848               {
849                 count=0;
850                 printf("Recursive version with memoisation\n");
851                 clock_t start=clock();
852                 m_lcs(m_v,m_p,m_b,yLen,xLen);
853                 for(j=0;j<=yLen;j++)
854                 {
855                   for(i=0;i<=xLen;i++)
856                   {
857                     if(is_valid(m_v,m_p,m_b,j,i)==0)
858                     {
859                       m_v[j][i]=-1;
860                     }
861                   }
862                 }
863                 space_m=max_log(m_v);
864                 printf("\nLength of longest common subsequence is: %d\n",m_v[↩
                        yLen][xLen]);
865                 printf("Dynamic programming table:\n");
866                 print_array(m_v,space_m);
867                 print_alignment(m_v,m_v[yLen][xLen]);
868                 printf("Number of table entries computed: %lld\n", count);
869                 pro_1=(xLen+1)*(yLen+1);
870                 pro_2=count*100;
871                 printf("Proportion of table computed: %f\% \n", pro_2/pro_1);
872                 clock_t stop =clock();
873                 time = (stop-start)/CLOCKS_PER_SEC;
874                 printf("Time taken: %f seconds\n\n", time);
875               }
876             }
```

```
877         //do we have to print DP tables??
878         else
879         {
880           //Method to execute the algorithm in is Iterative
881           if(iterBool)
882           {
883             printf("Iterative version\n");
884             clock_t start=clock();
885             it_lcs(it);
886             printf("Length of a longest common subsequence is: %d\n",it[↩
                  yLen][xLen]);
887             clock_t stop =clock();
888             time = (stop-start)/CLOCKS_PER_SEC;
889             printf("\nTime taken: %f seconds\n\n",time);
890           }
891           //Method to execute the algorithm in is recursion without ↩
                memoisation
892           if(recNoMemoBool)
893           {
894             printf("Recursive version without memoisation\n");
895             clock_t start=clock();
896             for(j=0;j<=yLen;j++)
897             {
898               for(i=0;i<=xLen;i++)
899               {
900                 r[j][i]=0;
901               }
902             }
903             r_lcs(r,yLen,xLen);
904             total=sum_of_entries(r);
905             printf("Total number of times a table entry computed: %lld\n",↩
                    total);
906             clock_t stop =clock();
907             time = (stop-start)/CLOCKS_PER_SEC;
908             printf("\nTime taken: %f seconds\n\n",time);
909           }
910           //Method to execute the algorithm in is recursion with ↩
                memoisation
911           if(recMemoBool)
912           {
913             count=0;
914             printf("Recursive version with memoisation\n");
915             clock_t start=clock();
916             m_lcs(m_v,m_p,m_b,yLen,xLen);
917             clock_t stop =clock();
918             printf("Length of longest common subsequence is: %d\n",m_v[↩
                  yLen][xLen]);
```

```
919               printf("Number of table entries computed: %lld\n", count);
920               pro_1=(xLen+1)*(yLen+1);
921               pro_2=count*100;
922               printf("Proportion of table computed: %f\%\n",  pro_2/pro_1);
923               time = (stop-start)/CLOCKS_PER_SEC;
924               printf("Time taken: %f seconds\n\n", time);
925             }
926           }
927         }
928         //Check if the algorithm to execute is ED.
929         else if(alg_type==ED)
930         {
931           //do we have to print DP tables??
932           if(printBool)
933           {
934             //Method to execute the algorithm in is Iterative
935             if(iterBool)
936             {
937               printf("Iterative version\n");
938               clock_t start=clock();
939               it_ed(it);
940               space_it=max_log(it);
941               printf("Edit distance  is: %d\n",it[yLen][xLen]);
942               printf("Dynamic programming table:\n");
943               print_array(it,space_it);
944               clock_t stop =clock();
945               time = (stop-start)/CLOCKS_PER_SEC;
946               printf("\nTime taken: %f seconds\n\n",time);
947             }
948             //Method to execute the algorithm in is recursion without ↩
                  memoisation
949             if(recNoMemoBool)
950             {
951               printf("Recursive version without memoisation\n");
952               clock_t start=clock();
953               r_ed(r,yLen,xLen);
954               space_r=max_log(r);
955               total=sum_of_entries(r);
956               printf("Dynamic programming table:\n");
957               print_array(r,space_r);
958               printf("\nTotal number of times a table entry computed: %lld\n↩
                     ", total);
959               clock_t stop =clock();
960               time = (stop-start)/CLOCKS_PER_SEC;
961               printf("\nTime taken: %f seconds\n\n", time);
962             }
963             //Method to execute the algorithm in is recursion with ↩
```

```
                memoisation
964             if(recMemoBool)
965             {
966               printf("Recursive version with memoisation\n");
967               count=0;
968               clock_t start=clock();
969               m_ed(m_v,m_p,m_b,yLen,xLen);
970               for(j=0;j<=yLen;j++)
971               {
972                 for(i=0;i<=xLen;i++)
973                 {
974                   if(is_valid(m_v,m_p,m_b,j,i)==0)
975                   {
976                     m_v[j][i]=-1;
977                   }
978                 }
979               }
980               space_m=max_log(m_v);
981               printf("\nEdit distance is: %d\n",m_v[yLen][xLen]);
982               printf("Dynamic programming table:\n");
983               print_array(m_v,space_m);
984               printf("Number of table entries computed: %lld\n", count);
985               pro_1=(xLen+1)*(yLen+1);
986               pro_2=count*100;
987               printf("Proportion of table computed: %f\% \n", pro_2/pro_1);
988               clock_t stop =clock();
989               time = (stop-start)/CLOCKS_PER_SEC;
990               printf("Time taken: %f seconds\n\n", time);
991             }
992           }
993           //do we have to print DP tables??
994           else
995           {
996             //Method to execute the algorithm in is Iterative
997             if(iterBool)
998             {
999               printf("Iterative version\n");
1000              clock_t start=clock();
1001              it_ed(it);
1002              printf("Edit distance  is: %d\n",it[yLen][xLen]);
1003              clock_t stop =clock();
1004              time = (stop-start)*1000.0/CLOCKS_PER_SEC;
1005              printf("\n Time taken: %f\n\n",time);
1006            }
1007            //Method to execute the algorithm in is recursion without ↩
                  memoisation
1008            if(recNoMemoBool)
```

```
1009            {
1010              printf("Recursive version without memoisation\n");
1011              clock_t start=clock();
1012              r_ed(r,yLen,xLen);
1013              total=sum_of_entries(r);
1014              printf("\nTotal number of times a table entry computed: %lld",↩
                      total);
1015              clock_t stop =clock();
1016              time = (stop-start)/CLOCKS_PER_SEC;
1017              printf("\nTime taken: %f seconds\n\n", time);
1018            }
1019            //Method to execute the algorithm in is recursion without m
1020            if(recMemoBool)
1021            {
1022              printf("Recursive version with memoisation\n");
1023              clock_t start=clock();
1024              m_ed(m_v,m_p,m_b,yLen,xLen);
1025              clock_t stop =clock();
1026              space_m=max_log(m_v);
1027              printf("\nEdit distance is: %d\n",m_v[yLen][xLen]);
1028              printf("Number of table entries computed: %lld\n", count);
1029              pro_1=(xLen+1)*(yLen+1);
1030              pro_2=count*100;
1031              printf("Proportion of table computed: %f\% \n", pro_2/pro_1);
1032              time = (stop-start)/CLOCKS_PER_SEC;
1033              printf("\nTime taken: %f seconds\n\n",time);
1034            }
1035          }
1036        }
1037        //Last choice is that algorithm to execute if SW algorithm
1038        else
1039        {
1040          //do we have to print DP tables??
1041          if(printBool)
1042          {
1043            if(iterBool)
1044            {
1045              printf("Iterative version\n");
1046              clock_t start=clock();
1047              it_sw(it);
1048              space_it=max_log(it);
1049              printf("Length of highest scoring local similarity is: %d\n",↩
                      max_of_array(it));
1050              printf("Dynamic programming table:\n");
1051              print_array(it,space_it);
1052              clock_t stop =clock();
1053              time = (stop-start)/CLOCKS_PER_SEC;
```

```
1054                printf("\nTime taken: %f seconds\n\n",time);
1055              }
1056            }
1057            //do we have to print DP tables??
1058            else
1059            {
1060              if(iterBool)
1061              {
1062                printf("Iterative version\n");
1063                clock_t start=clock();
1064                it_sw(it);
1065                printf("Length of highest scoring local similarity is: %d\n",↩
                       max_of_array(it));
1066                clock_t stop =clock();
1067                time = (stop-start)/CLOCKS_PER_SEC;
1068                printf("\nTime taken: %f seconds\n\n",time);
1069              }
1070            }
1071          }
1072
1073              freeMemory(); // free memory occupied by strings
1074            }
1075        }
1076        return 0;
1077    }
```