Assignment 2
2015313754 TaehyungGil (길태형)

October 31, 2020

My code consists of three parts.

1. Get the whole input sequences from ‘hw2_input.txt’.

2. Fill the matrix using DP algorithm.

3. Tracing back with using stack, print the result.

Below are detailed explanations and performance analysis of each part.

1. Get the whole input sequences from ‘hw2_input.txt’.

```
143     for(int i=0;i<k;i++)
144     {
145         int buff_idx=0;
146         input_sequence_len[i]=0;
147         output_sequence_now_idx[i]=0;
148         star_sequence[i]=0;
149         input_state = fgets(input_buffer,BUFFER_SIZE,fd_input);
150         while(input_buffer[buff_idx]!='\n'&&input_buffer[buff_idx]!=0)
151         {
152             input_sequence[i][buff_idx]=input_buffer[buff_idx];
153             buff_idx++;
154             input_sequence_len[i]++;
155         }
156     }
```

2. Fill the matrix using DP algorithm.

```
277 void alloc2(int x, int y)
278 {
279     array2 = (short**)malloc(sizeof(short*)*(x+1));
280     for(int i=0;i<=x;i++)
281         array2[i]=(short*)malloc(sizeof(short)*(y+1));
282     for(int i=0;i<=x;i++)
283         for(int j=0;j<=y;j++)
284             array2[i][j]=0;
285 }
286 void alloc3(int x, int y, int z)
287 {
288     array3 = (short***)malloc(sizeof(short**)*(x+1));
289     for(int i=0;i<=x;i++)
290     {
291         array3[i]=(short**)malloc(sizeof(short*)*(y+1));
292         for(int j=0;j<=y;j++)
293             array3[i][j] = (short*)malloc(sizeof(short)*(z+1));
294     }
295     for(int i=0;i<=x;i++)
296         for(int j=0;j<=y;j++)
297             for(int k=0;k<=z;k++)
298                 array3[i][j][k]=0;
299 }
```

If S1, S2, … Sn sequences were given, and the lengths of each sequences are L1,L2,...Ln, allocate n dimensional matrix dynamically. In allocn(int L1, int L2, … int Ln) function. (Array[L1][L2]... [Ln]) (alloc2,alloc3,alloc4,alloc5 functions)

Fill the matrix by calling dpn_new() function. (dp2_new(),dp3_new(),dp4_new(),dp5_new())

```
375 void dp3_new()
376 {
377     for(int x=0;x<=input_sequence_len[0];x++)
378         for(int y=0;y<=input_sequence_len[1];y++)
379             for(int z=0;z<=input_sequence_len[2];z++)
380             {
381                 // printf("%d %d \n",i,j);
382                 if(x==0 && x==y && y==z)
383                     continue;
384                 int max_score = INT_MIN;
385                 for(int i=0;i<4;i++)
386                 {
387                     int x_=x+way3[i][0];
388                     int y_=y+way3[i][1];
389                     int z_=z+way3[i][2];
390                     if(x_<0||y_<0||z_<0)
391                         continue;
392                     short score = array3[x_][y_][z_];
393                     if(i==0)
394                     {
395                         if(input_sequence[0][x-1]==input_sequence[1][y-1]&&input_sequence[1][y-1]==input_sequence[2][z-1])
396                             score+=1;
397                     }
398                     if(score>max_score)
399                         max_score=score;
400                 }
401                 array3[x][y][z]=max_score;
402             }
403 }
```

```
73  const int way2[3][2]={
74      {-1,-1},{-1,0},{0,-1}
75  };
76  const int way3[4][3]={
77      {-1,-1,-1},{-1,0,0},{0,-1,0},{0,0,-1}
78  };
79  const int way4[5][4]={
80      {-1,-1,-1,-1},{-1,0,0,0},{0,-1,0,0},{0,0,-1,0},{0,0,0,-1}
81  };
82  const int way5[6][5]={
83      {-1,-1,-1,-1,-1},{-1,0,0,0,0},{0,-1,0,0,0},{0,0,-1,0,0},{0,0,0,-1,0},{0,0,0,0,-1}
84  };
```

There are 4 arrays which define the directions that should be checked. The dpn_new() function go to all the elements of the matrix [L1][L2]...[Ln]. And when it gets to element [x][y]..[z], it checks [x-1][y]..[z], [x][y-1]..[z], [x][y]...[z-1], and [x-1][y-1]...[z-1] elements. And get the scores to corresponding elements as candidate scores.
Only if the score corresponds to the element of index [x-1][y-1]...[z-1] and all the characters corresponding to index x-1, y-1, … (of S1, S2,...Sn) are same, the candidate score number gets increase by one. Or the candidate score gets same score as the previous score. It stores the maximum value among the candidate score numbers. By doing this, it can store the maximum score which is given when considering only input_sequence[0][0...x], input_sequence[1][0...y] … input_sequence[k][0..z].

3. Tracing back with using stack, print the result.

```
void print_result2_new(FILE * fd_output)
{
    Node2 * top =NULL;
    int now_x=input_sequence_len[0];
    int now_y=input_sequence_len[1];
    int ret_x=0;
    int ret_y=0;
    int ret_x_len=0;
    int ret_y_len=0;
    int output_len=0;
    push2(&top,now_x,now_y);
    while(now_x!=0||now_y!=0)
    {
        if(input_sequence[0][now_x-1]==input_sequence[1][now_y-1])
            if(array2[now_x][now_y]==(array2[now_x-1][now_y-1]+1))
            {
                now_x--;
                now_y--;
                push2(&top,now_x,now_y);
                continue;
            }
        int max_idx=-1;
        int max_score = INT_MIN;
        int x_,y_;
        for(int i=0;i<3;i++)
        {
            x_ = now_x + bway2[i][0];
            y_ = now_y + bway2[i][1];
            if(x_<0||y_<0)
                continue;
            int score = array2[x_][y_];
            if(score>max_score)
            {
                max_idx=i;
                max_score=score;
            }
        }
        now_x = now_x + bway2[max_idx][0];
        now_y = now_y + bway2[max_idx][1];
        push2(&top,now_x,now_y);
    }
    // top = (Node*)malloc(sizeof(Node));
    output_sequence_now_idx[0]=0;
    output_sequence_now_idx[1]=0;
    while(top!=NULL)
    {
        Pos2* top_pos = pop2(&top);

        if(ret_x==top_pos->x&&ret_y==top_pos->y)
        {
            output_len--;
        }
        else if(ret_x==top_pos->x&&ret_y!=top_pos->y) //move down
        {
            output_sequence[0][output_len]='-';
            output_sequence[1][output_len]=input_sequence[1][top_pos->y-1];
            star_sequence[output_len]=' ';
        }
        else if(ret_x!=top_pos->x&&ret_y==top_pos->y) //move right
        {
            output_sequence[0][output_len]=input_sequence[0][top_pos->x-1];
            output_sequence[1][output_len]='-';
            star_sequence[output_len]=' ';
        }
        else if(ret_x!=top_pos->x&&ret_y!=top_pos->y) //Move cross
        {
            output_sequence[0][output_len]=input_sequence[0][top_pos->x-1];
            output_sequence[1][output_len]=input_sequence[1][top_pos->y-1];
            if(output_sequence[0][output_len]==output_sequence[1][output_len])
            {
                star_sequence[output_len]='*';
                start_num++;
            }
            else
                star_sequence[output_len]=' ';
        }
        ret_x=top_pos->x;
        ret_y=top_pos->y;
        output_len++;
        free(top_pos);
    }
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<output_len;j++)
            fputc(output_sequence[i][j],fd_output);
        fputc('\n',fd_output);
    }
    for(int j=0;j<output_len;j++)
        fputc(star_sequence[j],fd_output);
}
```

Trace back from the Array[L1-1][L2-1]...[Ln-1]. It chooses next position by selecting the position with best scores among the candidate positions. When it is at [x][y]..[z], the candidate positions are [x-1][y]...[z], [x][y-1]...[z], [x][y]...[z-1],and [x-1][y-1]...[z-1]. Push the next position to a stack. When it gets to Array[0][0]...[0], it pops one by one and write aligned sequences. Only when all the points get different, there is no dash. Or, only one sequence writes its character, while the others write gap('-'). The sequence which write its character is one that has different position from the previous popped one.

For example, when it goes [x+1][y]...[z] from [x][y]...[z], it pushes input_sequence[0][x+1] to S1'(Aligned S1) and dashes to all the other aligned sequences. When it goes to [x+1][y+1]...[z+1] from [x][y]...[z], it pushes input_sequence[0][x+1], input_sequence[1][y+1]…,input_sequence[k][z+1] to corresponding aligned sequences.

The reason why doing this is, only when all the points get different, it got one more score than previous score meaning all the characters from all the input sequences were same. Otherwise it pushes only one character only one input sequence and pushes dashes to the others.

This algorithm takes O(L1*L2*...Ln) for filling the matrix and take O(max(L1,L2,…)) to back trace.
The reason why it takes O(L1*L2*...Ln) for filling the matrix is that it should fill the all elements of the matrix. The matrix size is L1*L2*...Ln. So, it takes O(L1*L2*...Ln) for filling the matrix.
The reason why it takes O(max(L1,L2,...Ln)) for back trace is that it takes L1+L2+L3+...Ln for maximum to arrive Array[0][0]..[0] from Array[L1-1][L2-1]...[Ln-1].
L1+L2+...Ln<=5*max(L1,L2,...Ln) = O(max(L1,L2,..Ln)).
So. the total time complexity is O(L1*L2*...*Ln)

Thank you for reading.