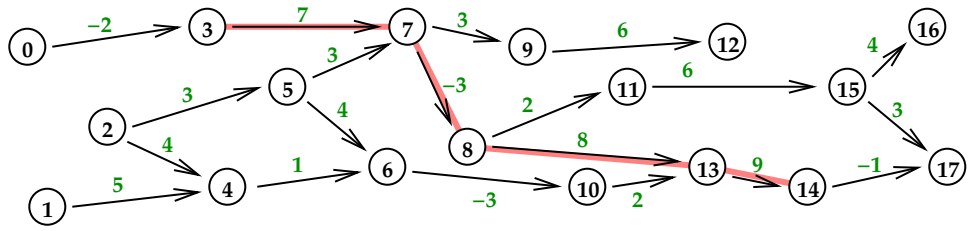
University of Nevada, Las Vegas Computer Science 477/677 Spring 2022

Assignment 7: Due Tuesday May 3, 2022, midnight.

Name:_____

1. Given an acyclic weighted directed graph G, write a dynamic program which finds a directed path through G of maximum total weight. Let the vertices of G be the integers $\{i\}_{0 \le i < n}$ and assume there is no edge from i to j if $i > j$. An example of such a graph is shown in the figure below, where the maximum weight path is indicated.



There are two ways to work the problem. You only need to do one of them.

(a) Identify the subproblems.

(b) Your code should work each subprogram in topological order.

(c) Your code should print the maximal weight path.

Use whatever pseudo-code you like, but make sure it's understandable.

Let $w(i, j)$ be the weight of the arc from i to j, if that edge exists. There are two equivalent ways to work the problem, which we call left-to-right and right-to-left.

Left-to-Right

The subproblems are $S_1, \ldots S_n$. Subproblem i is to find the maximum weight path in G which ends at i.

For each i, let $\texttt{In}_i$ be the set of in-neighbors of i, let M_i be the maximum weight of any path in G which ends at i, and let $\texttt{back}_i$ be the back pointer of i, which is either undefined, indicated by *, or is the next-to-the-last vertex on the maximum weight path to i. Here is the program.

```
writepath(k)
 {
  if(M_k > 0)
   writepath(back_k);
  write(k);
 }
```

```
maxM = 0 // the maximum weight of any path
last = 0 // the last vertex on the maximum weight path
For i from 0 to n
  {
   M_i = 0;
   back_i = *;
   For j ∈In_i
    temp = M_j + w(j, i);
    If(temp > M_i)
     {
      M_i = temp;
      back_i =j;
      If(M_i > maxM)
       {
        last = i;
        maxM = M_i;
       }
     }
  }
writepath(last)
```

Right-to-Left

The time complexity is unchanged, and the program is equally valid. Subproblem i is to find the maximum weight path through G that starts at i. The subproblems are worked in reverse order starting with n. We use forward pointers instead of back pointers, and $\texttt{Out}_i$ for the set of out-neighbors of i. Another difference is that we don't need recursion to print the path. `maxM` $= 0$ // the maximum weight of any path
`last` $= 0$ // the last vertex on the maximum weight path
For i from n downto 0
{
$M_i = 0$;
$\texttt{forw}_i = *$;
For $j \in \texttt{Out}_i$
`temp` $= w(i,j) + M_j$;
If(`temp` $> M_i$)
{
$M_i =$ `temp`;
$\texttt{forw}_i =$`j`;
If($M_i >$ `maxM`)
{
`first` $= i$;
`maxM` $= M_i$;
}
}
}
$k =$`first`
write k
While($M_k > 0$)
{
$k =\texttt{forw}_k$
write k
}

2. You need to store an array A where $A[i][j][k]$ is defined if $0 \le k \le j \le i < N$. Note that A is sparse, since the size of A is $\binom{N+2}{3}$, which is roughly $N^3/6$. To save space, you store the items of A in a 1-dimensional array $X[M]$ in row-major order, where $M = \binom{N+2}{3}$. You want to complete the following code.

```
int index(int i, int j, int k)
 {
  assert(i < N and j <= i and k <= j and k >= 0);
  return                                  ; // Insert the index in X of A[i][j][k]
 }
```

```
fetchA(int i, int j, int k)
 {
  assert(i >= 0 and i <= j and j <= k and k < N);
  return X[index(i,j,k)]
 }
```

Hint: The formula can most easily be expressed using combinatorials, that is, entries of Pascal's triangle.

Hint: Try working out the number of predecessors for a few cases, such as $A[3][2][1]$, $A[5][3][2]$, *etc.*.

If you write them all out, you will find that $A[3][2][1]$ has predecessors

$A[0][0][0] \ldots A[2][2][2], A[3][0][0] \ldots A[3][1][1], A[3][2][0]$

a total of $10 + 3 + 1$ predecessors. Similarly, $A[5][3][2]$ has predecessors

$A[0][0][0] \ldots A[4][4][4], A[5][0][0] \ldots A[5][2][2], A[5][3][0], A[5][3][1]$

a total of $35 + 6 + 2$ predecessors. Looking at Pascal's triangle, you should be able to conjecture the general formula, which is that $A[i][j][k]$ has $\binom{i+2}{3} + \binom{j+1}{2} + \binom{k}{1} = \frac{(i+2)(i+1)i}{6} + \frac{(j+1)j}{2} + k$ predecessors.

3. The number of proper divisors of a positive integer n can be computed by the following C++ code.

```
int numdiv(int n)
 {
  assert(n > 0);
  int numd = 1;
  int d = 2;
  while(d*d < n)
   {
    if(n % d == 0) numd = numd+2;
    d++;
   }
  if(d*d == n) numd++;
  return numd;
 }
```

For example, numdiv(1) = 1, numdiv(2) = 1, numdiv(3) = 1, numdiv(4) = 2, numdiv(5) = 1, and numdiv(6) = 3. Note that numdiv(p) = 1 if p is prime, and that numdiv(60) = 11.

You wish to store a 2-dimensional ragged array D, where D[i][j-1] is the j^{th} proper divisor of i, for all integers i from 2 up to some constant N, in a 1-dimensional array X, such that D[i][j] = X[index(i,j)].

The first nine rows (for $2 \leq i \leq 10$) of D look like this:

```
1
1
1 2
1
1 2 3
1
```

```
1 2 4
1 3
1 2 5
```

which means that the first 17 entries of X are: 1 1 1 2 1 1 2 3 1 1 2 4 1 3 1 2 5

How would you implement this project?

You need to compute an array B, where $B[i]$ is the number of predecessors of $D[i][0]$, which is the number of proper divisors of all positive integers less than i. Then $D[i][j] = X[B[i] + j]$. We compute B with the following code:

```
void initB()
 {
  B[2] = 0;
  for(int i = 3; i <= N; i++)
    B[i] = B[i-1] + numdiv(i-1);
 }
```

4. For each of the following C++ code fragments: run it on your computer, observe the output, then give the asymptotic time complexity in terms of n. Don't hand in the output of your program.

   I don't want you to hand in the program. It's only purpose is to guide your choice. If you are sure of the answer already, you don't have to run the program. Each answer is either $\Theta(n)$ or $\Theta(n \log n)$.

   (a)
   ```
   int main()
    {
     int n;
     cout << "Enter n: ";
     cin >> n;
     for(int i = 1; i < n; i++)
      for(int j = 1; j < i; j = 2*j)
       cout << i << " " << j << endl;
    }
   ```
   $\Theta(n \log n)$

   (b)
   ```
   int main()
    {
     int n;
     cout << "Enter n: ";
     cin >> n;
     for(int i = 1; i < n; i++)
      for(int j = i; j < n; j = 2*j)
       cout << i << " " << j << endl;
    }
   ```
   $\Theta(n)$

(c)
```
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  for(int i = n; i > 0; i = i/2);
   for(int j = 1; j < i; j++)
    cout << i << " " << j << endl;
 }
```

(d) $\Theta(n)$

```
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  for(int i = n; i > 0; i = i/2);
   for(int j = i; j < n; j++)
    cout << i << " " << j << endl;
 }
```

(e) $\Theta(n \log n)$

5. Run each of the following recursive C++ code fragments on your computer and observe the ouput. Then give an asymptotic solution to the recurrence. in terms of n. Don't hand in the output of your program.

(a)
```
int F(int n)
 {
  if(n <= 1) return 1;
  else return 4*F(n/2)+n*n; // This is the right side of the recurrence
 }
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  cout << "F(" << n << ") = " << F(n) << endl;
 }
```

$\Theta(n^2 \log n)$ by the master theorem.

(b)
```
int F(int n)
 {
  if(n <= 1) return 1;
  else return F(3*n/5)+F(4*n/5)+1; // This is the right side of the recurrence
 }
```

```
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  cout << "F(" << n << ") = " << F(n) << endl;
 }
```

$\Theta(n^2)$ by the generalized master theorem, since $(3/5)^2 + (4/5)^2 = 1$.

(c)
```
int F(int n)
 {
  if(n <= 1) return 1;
  else return F(sqrt(n))+1; // This is the right side of the recurrence
 }
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  cout << "F(" << n << ") = " << F(n) << endl;
 }
```

The recurrence is $F(n) = F(\sqrt{n}) + 1$. Use substitution. Let $m = \log_2 n$, and $G(m) = F(n)$ Thus $F(n) = G(\log_2 n)$. Note that $F(\sqrt{n}) = G(\log_2(\sqrt{n}) = G(\frac{1}{2}\log_2 n) = G(m/2)$. We thus have the recurrence $G(m) = G(m/2) + 1$. By the master theorem, $G(m) = \Theta(\log m)$. Hence $F(n) = G(m) = \Theta(\log m) = \Theta(\log \log n)$

6. Levenshtein edit distance is used for approximate string matching. The levenshtein distance between two words w_1 and w_2 is the number of edits needed to change one to the other. Three kinds of edits are permitted.

   (a) Insert a symbol.

   (b) Delete a symbol.

   (c) Replace a symbol with another symbol.

   Find the Levenshtein distance between "abbabacaa" and "babacbacab" Show the matrix.

		b	a	b	a	c	b	a	c	d	b
	0	1	2	3	4	5	6	7	8	9	10
a	1	1	1	2	3	4	5	6	7	8	9
b	2	1	2	1	2	3	4	5	6	7	8
b	3	2	2	2	2	3	3	4	5	6	7
a	4	3	2	3	2	3	4	3	4	5	6
b	5	4	3	2	3	3	3	4	4	5	5
a	6	5	4	3	2	3	4	3	4	5	6
c	7	6	5	4	3	2	3	4	3	4	5
a	8	7	6	5	4	3	3	3	4	4	5
a	9	8	7	6	5	4	4	3	4	5	5

The Levenstein distance is 5.