University of Nevada, Las Vegas Computer Science 477/677 Spring 2026

Assignment 4: Due Saturday April 4, 2026 11:59:59 PM

Name:_____

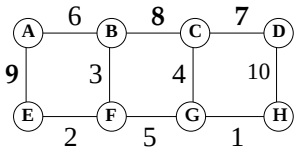
Advice of the Day: "When all else fails, follow insturctions."

1. On March 17, our graduate assistant, Rakibur Hassan, presented a complete explanation of how to use union/find to implement Kruskal's algorithm for finding a minimum weight spanning tree for a connected weighted graph.

   Use union/find to implement Kruskal's algorithm for the weighted graph below. A step consists of deciding whether a given edge should be part of the minimum spanning tree. The first step considers the edge GH. find(G) = G, find(H) = H, and union(G,H) (arbitrarily) moves G into the set led by H.

   The first array shows the initial values of *parent* and *size* of each vertex. In the second array, parent(G) has been changed to H, and size(H) to 2.

   Finish the execution of Kruskal's algorithm. At each step, show the matrix of parent and size values. Attach extra sheets as needed.



Edge processed: (none)

	A	B	C	D	E	F	G	H
parent	A	B	C	D	E	F	G	H
size	1	1	1	1	1	1	1	1

Edge processed: GH

	A	B	C	D	E	F	G	H
parent	A	B	C	D	E	F	H	H
size	1	1	1	1	1	1	1	2

Edge processed: EF

	A	B	C	D	E	F	G	H
parent	A	B	C	D	F	F	H	H
size	1	1	1	1	1	2	1	2

Edge processed: BF

	A	B	C	D	E	F	G	H
parent	A	F	C	D	F	F	H	H
size	1	1	1	1	1	3	1	2

Edge processed: CG

	A	B	C	D	E	F	G	H
parent	A	F	H	D	F	F	H	H
size	1	1	1	1	1	3	1	3

Edge processed: FG

	A	B	C	D	E	F	G	H
parent	A	F	H	D	F	H	H	H
size	1	1	1	1	1	3	1	6

Edge processed: AB

	A	B	C	D	E	F	G	H	
parent	H	H	H	D	F	H	H	H	path
size	1	1	1	1	1	3	1	7	compression

Edge processed: CD

	A	B	C	D	E	F	G	H
parent	H	H	H	H	F	H	H	H
size	1	1	1	1	1	3	1	8

2. Walk through the A^* algorithm to compute a mimimum cost path from S to T in the weighted directed graph shown below. The heurstic h is shown in red, while the weight on each arc is shown in black.

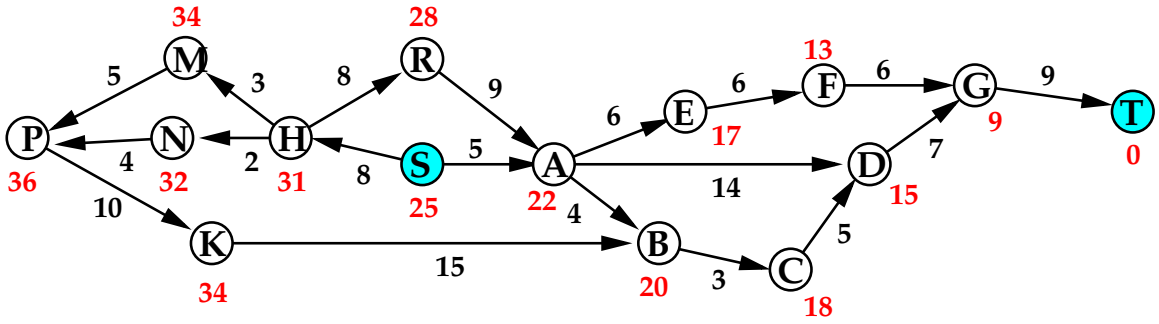
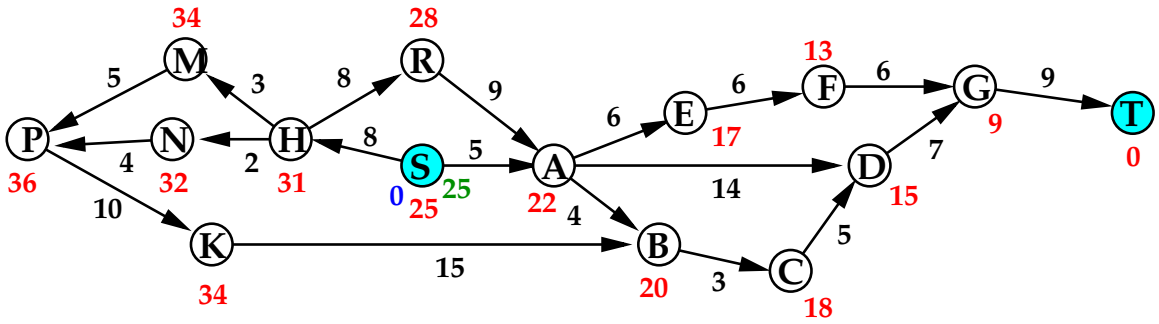
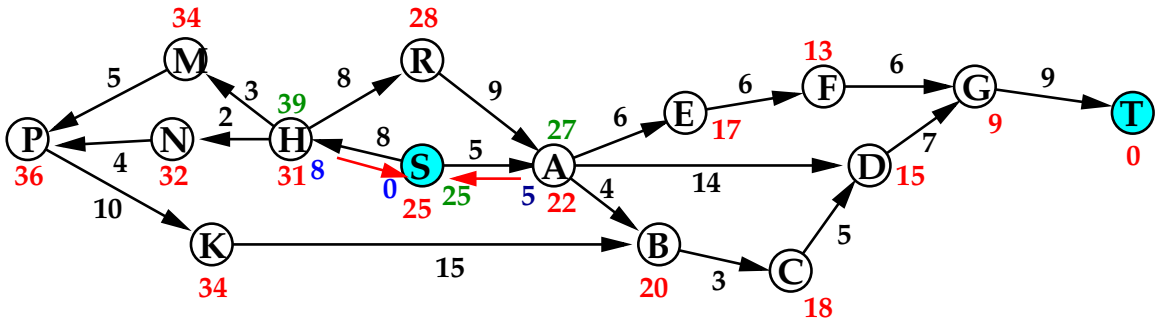
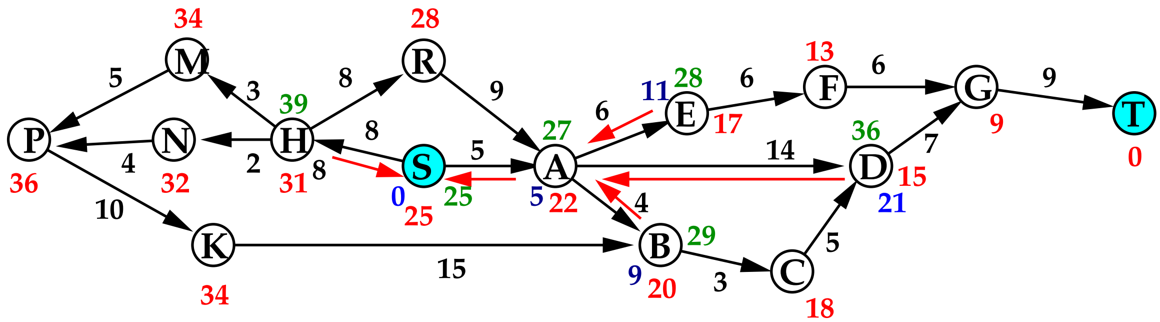


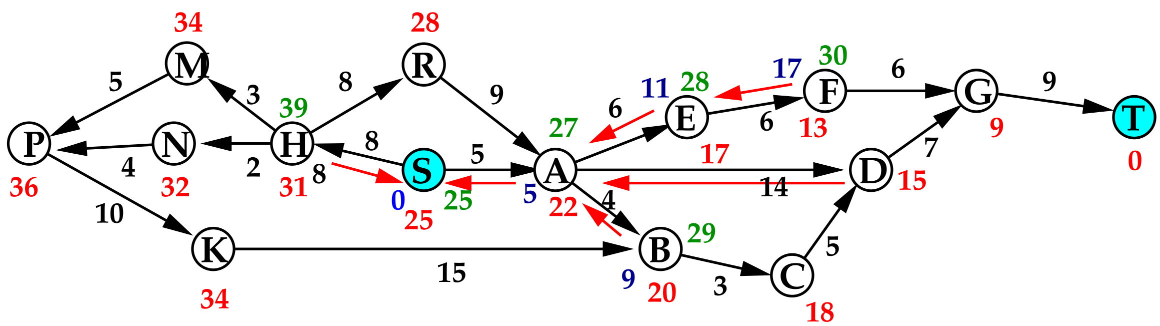
Figure 1: Example single pair minpath problem.

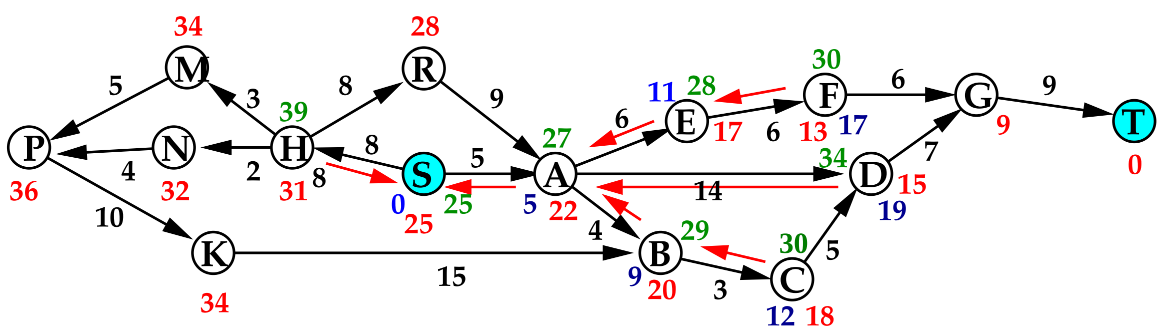
To avoid making you repeatedly redraw the figure, I have included several coies of Figure 1 on subsequent pages. If you need more of these figures, simply print another copy of one of those pages.

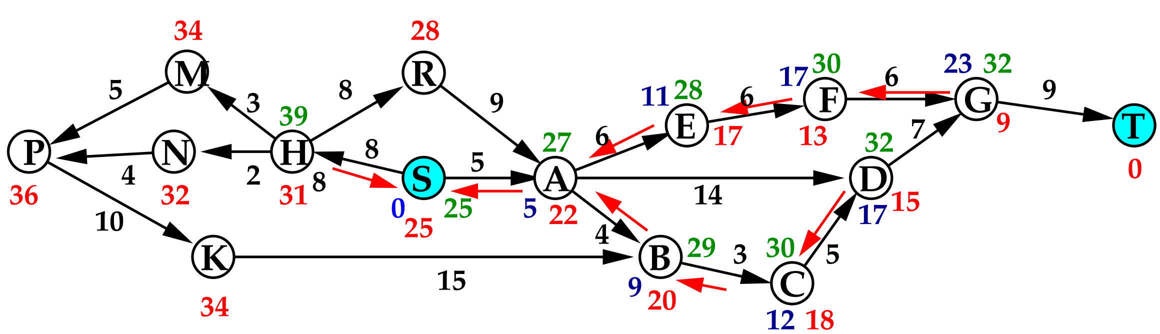


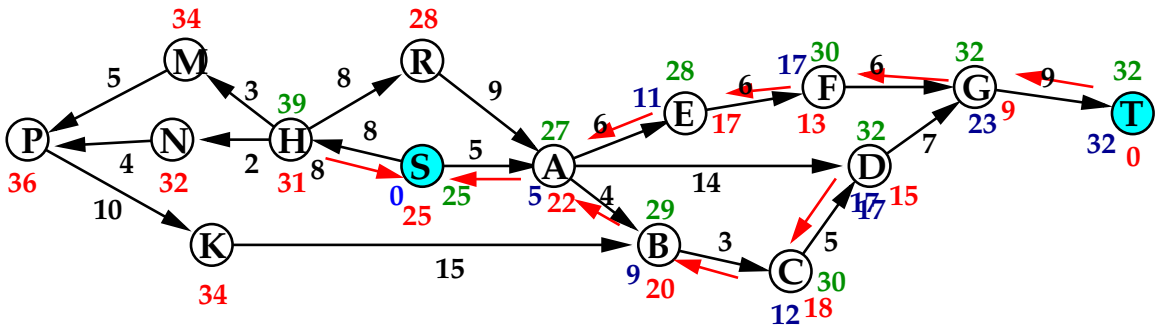



M
R
P
N
H
S
A
E
F
G
T
D
K
B
C


M
R
P
N
H
S
A
E
F
G
T
D
K
B
C


M
R
P
N
H
S
A
E
F
G
T
D
K
B
C


M
R
P
N
H
S
A
E
F
G
T
D
K
B
C



T is partially processed. At the next step, not shown becausse the figure is identical, T is fully processed and we are done.

3. Consider the following definition of a function g.

As I announced in class, I have changed the recurrence for g so as to make the answers to the problem easier to calculate. In fact, without that change, there is no solution to part (c).

$g(n) = 1$ if $n \leq 4$
$g(n) = g((n-1)/2) + g(n/2) + g((n+1)/2) + g((n+2)/2) + n$ for $n > 4$.

(a) The following recursive function computes $g(n)$.

```
int g(int n)
 {
  if(n <= 4) return 1;
  else return g((n-1)/2)+g(n/2)+g((n+1)/2)+g((n+2)/2)+n;
 }
```

What is the asymptotic time complexity, in terms of n, of that code?

Let $T(n)$ be the time to execute $G(n)$. The asymptotic recurrence is $T(n) = 4T(n/2) + 1$. By the master theorem, $T(n) = \Theta(n^2)$.

(b) Here is a dynamic program which computes values of $g(n)$ for $0 \leq n \leq N$. Let G be an array.

```
for(int n = 0; n <= 4; n++)
 G[n] = 1;
for(int n = 5; n <= N; n++)
 G[n] = G[(n-1)/2] + G[n/2] + G[(n+1)/2] + G[(n+2)/2] + n;
```

What is the asymptotic time complexity, in terms of N, of that code?

$G[i]$ is computed in constant time for each i from 0 to N. The time complexity is $\Theta(N)$.

(c) Use memoization to design an algorithm which computes $g(N)$ is $O(\log N)$ time, not counting the time needed for fetching memos from and storing memos into the search structure.

Memos are of the form $(i, g(i))$. When we say "store $g(i)$" we are storing that memo. When we say "If $g(i)$ is stored" we mean that the that memo has been stored. When we say "fetch $g(i)$" we are

querying the data structure for a memo whose first term is i, and we get back the value of $g(i)$, if the memo is in the structure.

Here is the algorithm.

```
int g(int i)
  {
    if (g(i) is stored)
      {
        fetch g(i);
        return g(i);
      }
    else if(i ≤ 4) return 1;
    else
      {
        g(i) = g((i − 1)/2) + g(i/2) + g((i + 1)/2) + g((i + 2)/2);
        store g(i);
        return g(i);
      }
  }
```

Computation of $g(N)$ for a specific N requires $O(\log N)$ memos to be created and stored. The diagram in the handout fibon.pdf shows the set of values of n for which F_n must be stored when computring F_N. The set of memos needed for computationof $g(n)$ is illustrated by a simlar diagram, and hence there are only $O(\log N)$ such memos needed.