University of Nevada, Las Vegas Computer Science 477/677 Fall 2024

Answers to Second Examination October 23, 2024

The entire examination is 225 points.

1. [15 points] Name three kinds of priority queue. **stack queue heap**

2. [10 points] If a binary tree has 9 nodes, its height is at least **3** and no more than **8**.

3. Solve these recurrences.

   (i) [10 points] $F(n) = F(n - \sqrt{n}) + 1$

   $$\frac{F(n) - F(n - \sqrt{n})}{\sqrt{n}} = \frac{1}{\sqrt{n}} = n^{-\frac{1}{2}}$$

   $F'(n) = n^{-\frac{1}{2}}$

   $F(n) = \Theta(n^{\frac{1}{2}})$

   (ii) [10 points] $F(n) = 4F(n/2) + n^2$

   A = 4, B = 2, C = 2, $\log_B A = 2$

   $F(n) = \Theta(n^2 \log n)$

   (iii) [10 points] $F(n) = 2F(n/2) + \sqrt{n}$

   A = 2, B = 2, C = $\frac{1}{2}$, $\log_A B = 1$

   $F(n) = \Theta(n)$

   (iv) [10 points] $F(n) = 2F(n/4) + \sqrt{n}$

   A = 2, B = 4, C = $\frac{1}{2}$, $\log_4 2 = \frac{1}{2}$

   $F(n) = \Theta(\sqrt{n} \log n)$

4. Find the time complexity of each code fragment in terms of n.

   (i) [10 points]

```
for(int i = 1; i < n; i++)
 for(int j = i; j < n; j = 2*j)
```

   $\Theta(n)$

   (ii) [10 points]

```
for(int i = 1; i < n; i++)
 for(int j = 1; j < i; j = 2*j)
```

   $\Theta(n \log n)$

5. [30 points] Evaluate each expression.

(i) $\dfrac{\log_2 9}{\log_2 3} = 2$

(ii) $\log_2 8 = 3$

(iii) $\log_4 2 = \dfrac{1}{2}$

(iv) $\log_{10} \dfrac{1}{1000000} = -6$

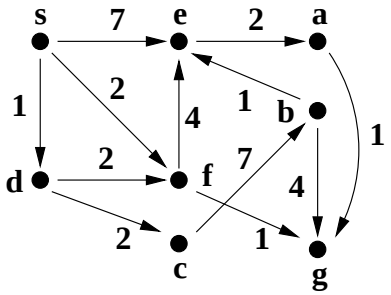
(v) $\log_2(\frac{1}{2})^5 = -5$

(vi) $(\log_7 5) \cdot (\log_5 7) = 1$

6. [20 points] Write pseudocode for the Bellman Ford algorithm. Assume that there are n vertices and m arcs that the source vertex is 0, and that the j^{th} arc is $(s[j], t[j])$ and has weight $w[j]$. Be sure to include the shortcut.

```
for all i from 1 to n
    V[i] = ∞
V[0] = 0
bool finished = false
while(not finished)
    finished = true;
    for all j from 1 to m
        temp = V[s[j]] + w[j]
        if(temp < V[t[j]])
            V[t[j]] = temp
            back(t[j] = s[i])
            finished = false
```

7. [20 points] Execute Dijkstra's algorithm on the weighted directed graph shown below. Be sure to show the array of back pointers. Cross out, but do not erase, intermediate values.



I will walk through the complete analysis of this problem using Dijkstra's algorithm.

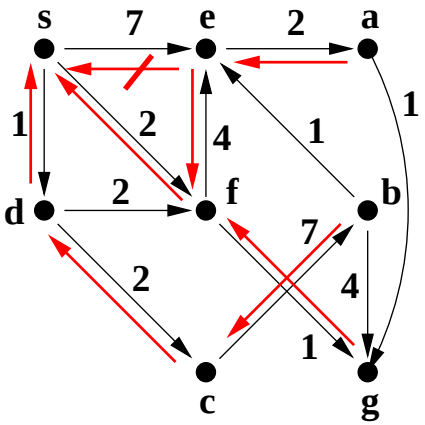
For any vertex v, we let $V[v]$ be the minimum length of any path from s to v found so far. After each step, each vertex is fully processed, partially processed, or unprocessed. Initially, the source vertex s is partially processed, and all other vertices are unprocessed.

A fully processed vertex v has its final value $V[v]$ and its final backpointer back$[v]$, unless $v = s$
A partially processed vertex v has temporary values of V and back. Every vertex that has been visited is fully or partially processed, and the partially processed vertices form an updatable minqueue.
Vertices that have not been visited are unprocessed. We could default $V[v]$ to ∞ if v is unprocessed.
Initially, s is partially processed and all over vertices are unprocessed. back$[s]$ is undefined and remains undefined throughout.
Let H be the minheap which hold the partially processed vertices. During a step, the value of $V[v]$ might be decreased for some v in H, in which case bubbleup(v) is executed.

Dijkstra's algorithm consists of n *steps*. During each step, exactly one partially processed vertex becomes fully processed and any number of unprocessed vertices become partially processed. Execution ends when all vertices are fully processed, that is, H is empty.

Here is the code for one step.

```
v = deletemin(H)
    //v is fully processed
for all u ∈ Outnbr(v)
    if(u is unprocessed)
        V[u] = V[v] + W[v, u]
        back[u] = v
        insert(H, u)
    else
        temp = V[v] + W[v, u]
        if(temp < V[u])
            V[u] = temp
            back[u] = v
            bubbleup(u)
```



s	a	b	c	d	e	f	g
0	8	10	3	1	~~7~~ 6	2	3
*	e	c	d	s	~~s~~ f	s	f

Here are the steps of Dijkstra's algorithm for the directed graph the shown in the figure. Initially, $H = \{s\}$.

(a) deletemin$(H) = s$. Outnbrs$(s) = \{d, e, f\}$.

i. $V[d] = 1$, back$[d] = s$, insert(H, d).

ii. $V[e] = 7$, back$[e] = s$, insert(H, e).

iii. $V[f] = 2$, back$[2] = s$, insert(H, f).

(b) deletemin$(H) = d$. Outnbrs$(d) = \{c, f\}$.

i. $V[c] = 3$, back$[c] = d$, insert(H, c).

ii. Do nothing, since $V[f]$ remains 2.

(c) deletemin$(H) = f$. Outnbrs$(f) = \{e, g\}$.

i. $V[e]$ changes to 6 and back$[e]$ changes to f. Bubbleup(e).

ii. $V[g] = 3$ and back$[g] = f$. Insert(H, g).

(d) deletemin$(H) = g$. Outnbrs$(g) = \emptyset$.

(e) deletemin(H) = c. Outnbrs(c) = $\{b\}$.

i. $V[b] = 10$ and back$[b] = c$. Insert(H, b).

(f) deletemin(H) = e. Outnbrs(e) = $\{a\}$.

i. $V[a] =$ back$[a] = e$. Insert(H, a).

(g) deletemin(H) = a. Outnbrs(a) = $\{g\}$.

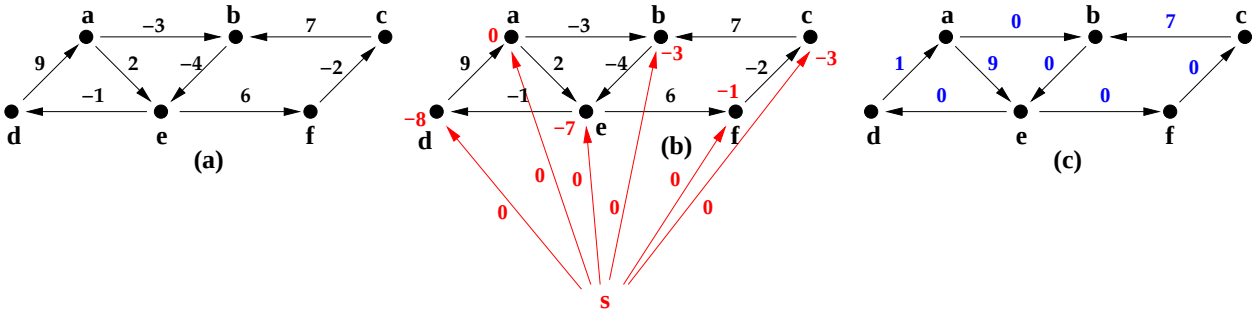
i. Do nothing, since $V[g]$ remains 3.

(h) deletemin(H) = b. Outnbrs(b) = $\{g\}$.

i. Do nothing, since $V[g]$ remains 3.

H is now empty.

8. [20 points] The first step of Johnson's algorithm is to adjust the edge weights. Given the weighted digraph shown in figure (a), show the adjusted edge weights in figure (c). Use (a) and (b) for your work. Do not finish Johnson's algorithm.



9. [30 points] Find the maximum flow from S to T for the weighted digraph shown in (a). I have chosen the initial step to be the flow of 3 shown in (b).

1. Sketch the residual graph in (c).

2. Finish the problem. You must choose an augmenting flow, then show the combined flow, then show the mincut on the original graph to prove that your flow is maximal.

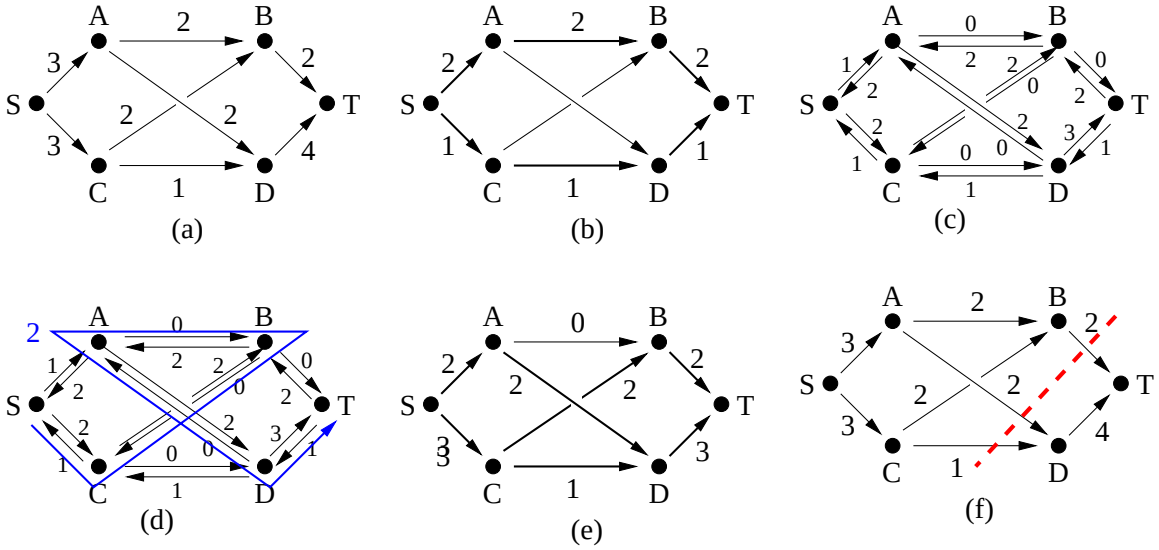


Figure (c) shows the residual graph. (d) shows an augmenting flow of 2, and (e) shows the combined flow, which is 5. (f) shows a mincut of 5, proving that our flow of 5 is maximal.

10. [20 points] The following code is for treesort. The code has been tested. I deleted the header to save space, as well as the code for `insert` and `traverse`.

My output was:

```
Enter 20 integers:
 70 29 18 45 91 58 64 97 95 67 92 19 19 63 27 19 84 73 67 60
 18 19 19 19 27 29 45 58 60 63 64 67 67 70 73 84 91 92 95 97
```

Fill in the missing code for the functions `insert` and `traverse`.

```
int const N = 20;
struct bstnode;
typedef bstnode*tree;
tree null = NULL;
struct bstnode
 {
  int item;
  tree lft;
  tree rght;
 };

void insert(tree&root,int newitem)
 {
 }

void traverse(tree root)
 {
 }

int main()
 {
  tree allitems = null;
  cout << "Enter " << N << " integers:" << endl;
  for(int i = 0; i < N; i++)
   {
    cin >> n;
    cout << " " << n;
    insert(allitems,n);
   }
  cout << endl;
  traverse(allitems);
  cout << endl;
  return 1;
 }
```