Exercises 5

12 February

Questions 3,4,7 are assessed. Hand in by Monday 19 February.

1. Suppose that we have a weighted graph where the weights are integers, and may be negative as well as positive. The weight of a path is the sum of the weights of the arcs in the path as before.
Give an example to show that Dijkstra's algorithm does not necessarily give the shortest path.

2. Given a weighted directed graph (G,W), a *routing table* is a square array next$[i, j]$ which gives the first step of a shortest path from i to j (all nodes i, j). Thus if next$[i, j] = k$ then there is a shortest path from i to j of the form $i,k,\ldots, j$. So the next node after k along a shortest path would be next$[k, j]$.
Adapt the code for Floyd's all-pairs shortest path algorithm (slide 155) to additionally produce a routing table for a weighted directed graph (G,W).

3. [3 marks] Given a weighted directed graph, we say that the *bandwidth* of a path in the graph is the weight of the lowest weight arc in the path. For instance, if a path has arc weights $3,4,2,6$ then the bandwidth is 2. The *widest path problem* is as follows: given a weighted directed graph (G,W) for each pair of nodes i, j find the bandwidth of the path from i to j with greatest bandwidth. If there is no path from i to j return 0 for this pair of nodes.
Give an $O(n^3)$ algorithm based on Floyd's algorithm to solve the widest path problem given a weighted directed graph (G,W) with n nodes. You may use pseudocode as in the lectures or give a Java program.

4. [4 marks] By adapting the Bellman-Held-Karp algorithm for the Travelling Salesman Problem (slides 162-3), give pseudocode (or a Java program) to solve the Hamiltonian circuit problem in time $O(n^2 2^n)$ on an undirected graph with n nodes.
[Hint: Calculate and store $H[S,x]$, which holds iff there is a path from Start to x with intermediate nodes precisely S and used exactly once.]
Note that the weighted graph for the TSP is complete, which is not the case for the HCP, of course.

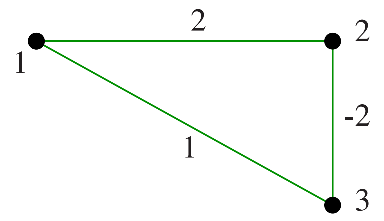
5. How many comparisons does it take to find the smallest element of an unordered list of length n? Why is your answer best possible?

6. (Baase) Write an algorithm to find both the smallest and largest elements in a list of n entries. Try to find a method that does at most roughly $1.5n$ comparisons of list entries. [Hint: adapt the Wimbledon-style knockout tournament algorithm for finding the largest element.]

7. [adapted from 2015 exam] (a) [1 mark] Give a decision tree for the Binary Search algorithm applied to ordered lists of length six, with elements indexed from 0 to 5. Assume that the algorithm chooses the element with lower index at any point where there is a choice.
(b) [1 mark] State the worst-case number of comparisons.
(c) [1 mark] Calculate the average number of comparisons, assuming that the element being searched for is in the list, and all positions in the list are equally likely.

Answers to Exercises 5

1.



For instance, let the nodes be $\{1,2,3\}$ with arcs $(1,2),(1,3),2,3)$ with $W(1,2)=2$, $W(2,3)=-2$, $W(1,3)=1$. If we use Dijkstra's algorithm to find the shortest path from 1 to 3, it will immediately find the direct path $1,3$ of weight 1, whereas the shortest path is in fact $1,2,3$ of weight 0.
Remark: Floyd's algorithm can be used on graphs with negative weights, as long as we do not have a negative weight cycle within the graph.

2. In order to calculate next, we also need to calculate the distance of the shortest path as usual using B. Assume that the nodes are numbered from i to n.

```
**Algorithm** Routing table:
input $A$
set $B[i,j] = \begin{cases} 0 & \text{if } i=j \\ A[i,j] & \text{if } i \neq j \text{ and there is an arc } (i,j) \\ \infty & \text{otherwise} \end{cases}$
# $B = B_0$
set $\text{next}[i,j] = \begin{cases} i & \text{if } i=j \\ j & \text{if } i \neq j \text{ and there is an arc } (i,j) \\ 0 & \text{otherwise (fictitious value)} \end{cases}$
for $k$ = 1 to $n$:
    for $i$ = 1 to $n$:
        for $j$ = 1 to $n$:
            if $b_{ik} + b_{kj} < b_{ij}$:
                $b_{ij} = b_{ik} + b_{kj}$
                $\text{next}_{ij} = \text{next}_{ik}$
return $B$, next
```

3.

```
**Algorithm** Widest path:
input $A$
set $\text{bw}[i,j] = \begin{cases} 0 & \text{if } i=j \\ A[i,j] & \text{if } i \neq j \text{ and there is an arc } (i,j) \\ 0 & \text{otherwise} \end{cases}$
```

```
for $k$ = 1 to $n$:
    for $i$ = 1 to $n$:
        for $j$ = 1 to $n$:
            if $\min(\mathrm{bw}_{ik}, \mathrm{bw}_{kj}) > \mathrm{bw}_{ij}$:
                $\mathrm{bw}_{ij} = \min(\mathrm{bw}_{ik}, \mathrm{bw}_{kj})$
return bw
```

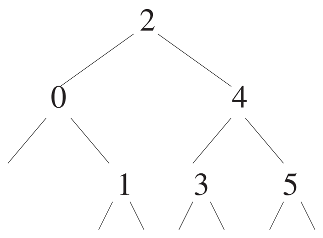
4.

```
**Algorithm** HCP:
Input $G$ with adjacency matrix $A$
Choose Start $\in$ nodes$(G)$
for $x \in$ Nodes $\setminus \{$Start$\}$:
    $H[\emptyset, x] = A[\text{Start}, x]$
# Process sets $S$ in increasing order of size.
# for size = 1 to $n-2$:
    # for $S \subseteq$ Nodes $\setminus \{$Start$\}$ with $|S| ==$ size:
for $S \subseteq$ Nodes $\setminus \{$Start$\}$ with $S \neq \emptyset$:
    for $x \in$ Nodes $\setminus (S \cup \{$Start$\})$:
        # Find $H[S,x]$
        $H[S,x] = 0$
        for $y \in S$:
            if $H[S \setminus \{y\}, y]$ and $A[y,x]$:
                $H[S,x] = 1$
                break
# Now have calculated and stored all values of $H[S,x]$
for $x \in$ Nodes $\setminus \{$Start$\}$:
    if $H[\text{Nodes} \setminus \{\text{Start}, x\}, x]$ and $A[x, \text{Start}]$:
        return true
return false
```

5. $n-1$. Move along the list, carrying the lowest element found so far. An alternative, the so-called tournament algorithm, consists of comparing $L[0]$ with $L[1]$, $L[2]$ with $L[3], \ldots$ in the first round, and then the "winners" meet in the second round, until the overall winner is chosen in the "final". The number of comparisons is still $n-1$, but there are only $\lceil \log n \rceil$ rounds, and there is scope to speed up the process by performing the comparisons of each round in parallel.
$n-1$ comparisons are truly necessary, since for one to be the winner the $n-1$ other elements must have lost at least once, and there is only one loser for each comparison.

6. Adapt the tournament idea (for finding the greatest element). After the first round (taking $n/2$ comparisons) we let the winners carry on as before (a further $n/2-1$ comparisons), while the losers compete for the booby prize (again a further $n/2-1$ comparisons). After the first round we can of course use a linear method instead on the two halves. So we can find both the greatest and the least in $1.5n-2$ comparisons.

7. (a)



(b) Worst-case number of comparisons is 3.
(c) Average case number of comparisons is $(1.1+2.2+3.3)/6 = 14/6 = 2\frac{1}{3}$.