1.

Answer:

Suppose that there are n items, and the knapsack has maximum weight capacity W.

(7%) Algorithm

Special-0-1knapsack-problem(w, v, n, W):

1. Sort the items by increasing weight.
2. Let remaining capacity $W_{current} = W$.
3. Let the total profit $P = 0$.
4. for each item i $(1 \leq i \leq n)$:
5. if $w_i \leq W_{current}$:
6. $W_{current} = W_{current} - w_i$
7. $P = P + v_i$
8. return P

(3%) Correctness

Greedy-choice property:

Suppose we choose an item j that does not have the least weight $(w_j < w_i)$ and does not have the greatest value $(v_j < v_i)$. Then, we could replace item j with item i, which has a smaller weight and a greater value. This replacement would result in a better profit or, at worst, maintain the same profit while keeping the total weight within the knapsack's capacity.

(0%) Time complexity

Sorting the items by increasing weight takes $O(n \log n)$.

Iterating through all n items in the for loop takes $O(n)$.

The overall time complexity is $O(n \log n)$.

2.

Prove that a non-full binary tree cannot correspond to an optimal prefix-free code.

A:

Let *T* be a binary tree corresponding to an optimal prefix code and suppose that *T* is not full. Let internal node n have a single child x, where x can be internal node or leaf node. Let T_0 be the tree obtained by removing n and replacing it by x. Let M be a set of leaf nodes which are descendants of x. Then we have

$$cost(T_0) \leq \left(\sum_{c \in C-M} c.freq \cdot d_T(c) \right) + \left(\sum_{m \in M} m.freq \cdot (d_T(m) - 1) \right)$$

$$= \left(\sum_{c \in C} c.freq \cdot d_T(c) \right) - \left(\sum_{m \in M} m.freq \right)$$

$$< \sum_{c \in C} c.freq \cdot d_T(c) = cost(T)$$

, where C is the set of all character leaf nodes in tree *T*, $d_T(c)$ is the depth of character *c* in tree *T* , which contradicts the fact that *T* was optimal. Therefore every binary tree corresponding to an optimal prefix code is full.

3.

$\phi(i) = 114513i - 2^{\lceil \lg(i+1) \rceil}$

amortized cost :

$\forall k \in N$

$i = 2^k$: $\hat{c}_i = c_i + \phi(i) - \phi(i-1)$

$= i + 114513i - 2^{\lceil \lg(i+1) \rceil} - 114513(i-1) + 2^{\lceil \lg i \rceil}$

$= i + 114513i - 2i - 114513i + 114513 + i$

$= 114513$

$i \neq 2^k$: $\hat{c}_i = c_i + \phi(i) - \phi(i-1)$

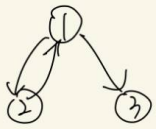
$= 1 + 114513i - 2^{\lceil \lg(i+1) \rceil} - 114513i + 114513 + 2^{\lceil \lg i \rceil}$

$= 114514$

total amortized cost $\leq 114514n$

$\Rightarrow$ total amortized cost $= O(n)$

4.



DFS : $1 \to 2 \to 1 \to 3 \to 1$

DFS finish : $2 \to 3 \to 1$

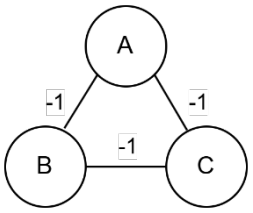
DFS from 2, 2·1·3 will be marked in a SCC

real SCC : $\{\{1, 2\}, \{3\}\}$

5. (10%) Argue that if all edge weights of a graph are positive, then the subset of edges that connects all vertices and has minimum total weight must be a tree. Give an example to show that the same conclusion does not follow if we allow some weights to be nonpositive.

A.

If we have a graph like this, then the subset of edges that connects all vertices and has minimum total weight include e_{AB}, e_{AC}, e_{BC}. Since this graph have a cycle, and this would mean that removing any of the edges in this cycle would mean that the remaining edges would still connect all the vertices, but would have a total weight that's less by the weight of the edge that was removed.



Grading: Explanation(5%) Example(5%)

6.

Answer:

Algorithm

Modified-Bellman-Ford(G, s):

1. for each v, set $d(v) = \infty$.
2. Set $d(s) = 0$.
3. for ($k = 1$ to $|V| - 1$):
4. $\quad relax\_flag = flase$ // Flag to detect if any edge is relaxed
5. $\quad$ for each edge(u, v) with weight w in edges: // Relax all edges
6. $\quad\quad$ if $d(u) + w < d(v)$:
7. $\quad\quad\quad d(v) = d(u) + w$
8. $\quad\quad\quad relax\_flag = true$
9. $\quad$ if $relax\_flag$ is $false$: // Terminate if no edges were relaxed
10. $\quad\quad$ break
11. return d

Explanation

Since m is the maximum over all vertices $v \epsilon V$ of the minimum number of edges in a shortest path from the source s to v, after m iterations, all shortest paths in the graph will have been determined. Therefore, no further edge relaxation will occur in $m + 1$-th iteration. By detecting that no edge relaxation happens, we can terminate the algorithm.

Time complexity

The time complexity of the modified algorithm is the same as the original Bellman-Ford algorithm, which is $O(VE)$.

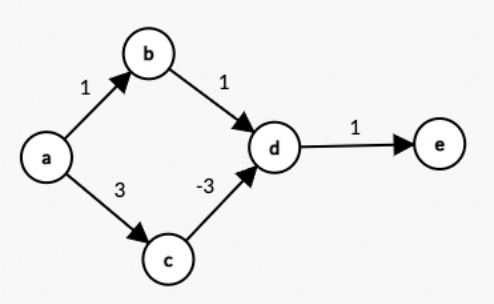
Grading Policy

(3%) Bellman-Ford algorithm

(7%) Modified the algorithm that allows it to terminate in m+1 iterations.

7.
Give a simple example of a directed acyclic graph with negative-weight edges for which Dijkstra's algorithm produces an incorrect answer.

A:



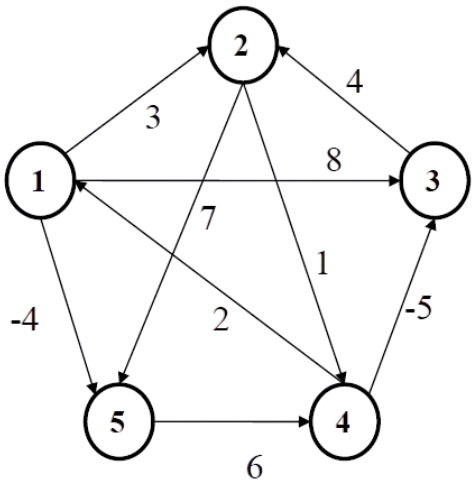
If source is node a.
Dijkstra's algorithm:
Step 1:
Visit a:
Shortest_path(a, b) = 1
Shortest_path(a, c) = 3
Step 2:
Visit b:
Shortest_path(a, d) = 2
Step 3:
Visit d:
Shortest_path(a, e) = 3
Step 4:
Visit c:
Shortest_path(a, d) = 0

The correct shortest path is a -> c -> d -> e, which weight is 1, not 3.
The reason is that Dijkstra's algorithm only visit each node one time.

8. (10%) Run the Floyd-Warshall algorithm on the weighted, directed graph of Figure 1. Show the matrix $D^{(k)}$, for k=0,1,2,3.

A.

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



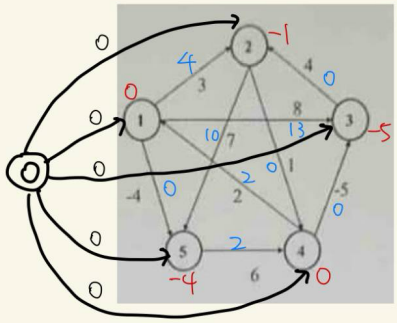
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Grading: $D^{(0)}$(4%), each of remaining step(2%)(2%)(2%)

9.



10.

```
Minimum_vertex_cover(int V, vector<pair<int,int>> E){
    bool cover[V]={0}; ← O(V)
    for(auto [u,v] : E){ ← O(E)
        if(!cover[u] && !cover[v]){ ← O(1)
            cover[u]=cover[v]=1;
        }
    }
    return cover;
}
```

Time complexity: $O(V)+O(E)\cdot O(1)=O(V+E)$

Approximation ratio:

min cover ≥ each edge we choose only one of the endpoints

⇒ min cover ≥ $\frac{1}{2}$ my cover

⇒ my cover ≤ 2 min cover

11. (12%) For each statement, determine whether it is true (T) or false (F).(是非題答錯倒扣一分)

(1) The factoring problem is an NP-complete problem.

(2) If we prove that the satisfiability problem can polynomial-time reduce to problem A in NP, then problem A is NP-complete.

(3) A 3-SAT (satisfiability) problem is polynomial-time-reducible to a 2-SAT problem.

(4) If G is an undirected bipartite graph with an odd number of vertices, then G is non hamiltonian.

(5) If a problem is NP-complete, we have not yet found a polynomial-time solution in the worst cases.

(6) We have a dynamic programming (DP) algorithm to solve the 0/1 Knapsack problem that runs in O(nW) time, where n is the number of items and W is the maximum weight of items that the thief can put in his knapsack. Therefore, the DP algorithm is a polynomial-time algorithm.

A.

(1) (F) The factoring problem is in NP but no in NP-complete.

(2) (T) The satisfiability problem (SAT) is NP-complete, and if we can polynomially reduce SAT to A and A is in NP, this proves that A satisfies both conditions for being NP-complete. Therefore, problem A is NP-complete.

(3) (F) A 3-SAT problem cannot be reduced to a 2-SAT problem. Consider a 3-CNF clause, such as ($x_1 \vee x_2 \vee x_3$), cannot be directly expressed as a combination of 2-CNF clauses without losing equivalence.

(4) (T) A Hamiltonian cycle in a graph must visit every vertex exactly once and return to the starting vertex. If $G$is Hamiltonian, it must contain odd cycle. However, bipartite graphs cannot have odd cycles due to their structure, where vertices are divided into two disjoint sets, and all edges connect vertices from different sets. This contradiction implies that if G is a bipartite graph with an odd number of vertices, it cannot be Hamiltonian.

(5) (T) NP-complete only ensure polynomial time of verification, not ensure we can get a solution in exponential time, unless P = NP.

(6) (F) It's a pseudo-polynomial time problem, which depends on the scale of n and W.