Homework 6

Thomas Kim tsk389 51835

David Munoz dam2989 51840

CS331 Algorithms and Complexity

Problem Q1. Given integers $a_1, \ldots, a_n$, design a dynamic programming algorithm that determines whether there exists a partition of the numbers into 3 disjoint subsets P, Q, R such that the sum of the numbers in each set are equal. In other words,

$$\sum_{a_i \in P} a_i = \sum_{a_i \in Q} a_i = \sum_{a_i \in R} a_i$$

Memoization

1: Let the set of integers $a_1, \ldots, a_n$ be denoted A
2: Let the minimum element $a_i \in A$ be denoted $min(A)$
3: Let the maximum element $a_i \in A$ be denoted $max(A)$
4: Let the sum of all elements $a_1, \ldots, a_n$ be denoted $sum(A)$
5: Create a table of dimension $|n * min(A)|$ by $|n * max(A)|$
6: Each element of the table $e_{j,k}$ stores whether there exists a partition such that $\sum_{a_i \in P} a_i = j \wedge \sum_{a_i \in Q} a_i = k$

Algorithm $OPT(i, j, k)$

1: **if** $j\neg \in \mathbb{Z} \vee k\neg \in \mathbb{Z}$ **then**
2: **return** FALSE
3: **end if**
4: Let $A' := \{a_i, a_{i+1}, \ldots, a_n\}$
5: **if** $j = k = 0 \wedge i = n + 1$ **then**
6: Fill in the table entry for coordinates (j, k) to be true
7: **return** TRUE
8: **else if** $j \neq 0 \wedge k \neq 0 \wedge i = n + 1$ **then**
9: Fill in the table entry for coordinates (j, k) to be false
10: **return** FALSE
11: **end if**
12: **return** $OPT(i + 1, j - a_i, k) \vee OPT(i + 1, j, k - a_i) \vee OPT(i + 1, j, k)$

Call $OPT(1, \frac{sum(A)}{3}, \frac{sum(A)}{3})$

Correctness

Proof. The last line of the algorithm explores putting a_i in P, Q, and R in the three cases, respectively. This is repeated for every $1 \leq i \leq n$ by lines 5 and 8
By the problem definition, each a_i must go in P, Q, or R.
Furthermore, j and k cannot be any smaller than the maximum subset of A, MAX or bigger than the minimum subset of A, MIN
This follows naturally since $j - \sum a_i$ cannot be smaller than $\frac{sum(A)}{3} - sum(MAX)$ or bigger than $\frac{sum(A)}{3} + sum(MIN)$
Now we observe $sum(MAX) \leq max(A) * |MAX| \leq max(A) * n \wedge sum(MIN) \geq min(A) * |MIN| \geq min(A) * n$ Therefore, the entire space of solutions is explored and the table is big enough.

□

Problem Q2. Given positive integers n and W, design a dynamic programming algorithm that finds the number of possible n element sets $\{x_1, \ldots, x_n\}$ for which $\sum_i x_i^2 = W$ where each x_i is a nonnegative integer. **Memoization**

1: Create an n by W table

2: Each element (j, k) of the table will hold the number of unique j element sets for which $\sum_i x_i^2 = k$

Algorithm $OPT(i, j, k)$

1: **if** $j = 0 \wedge k = 0$ **then**
2: **return** 1
3: **else if** $i = 0 \vee j < 0 \vee k < 0$ **then**
4: **return** 0
5: **end if**
6: **return** $OPT(i-1, j-1, k-i^2) + OPT(i-1, j, k)$

Call $OPT(0, n, W)$

Correctness

Proof. For each value of i, a value i^2 can either be used or not used.
Both are explored by line 6 of the algorithm
Therefore the entire solution space is explored.
Solutions requiring more or fewer than n elements are discarded by line 3
Solutions which require precisely n elements are accepted by line 1

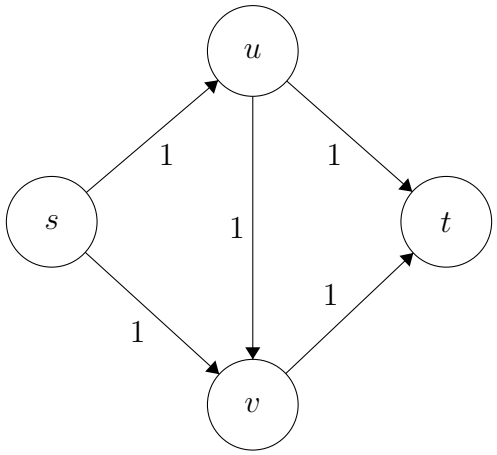
□

Runtime

Proof. At worst, the entire table is filled.
The table has nW elements
Therefore, the algorithm is pseudopolynomial in n

□

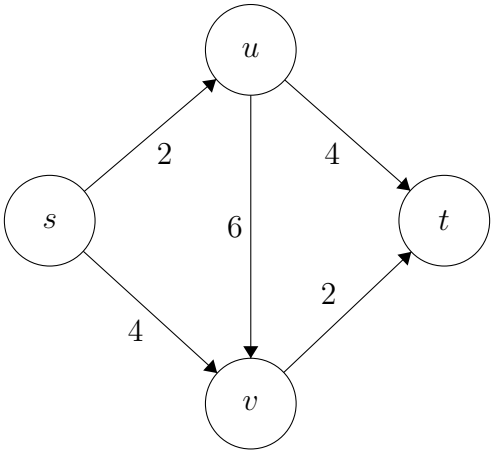
Problem Q3(a). List all the minimum $s-t$ cuts in the flow network in figure 7.24 (reproduced below).



Minimum cuts:

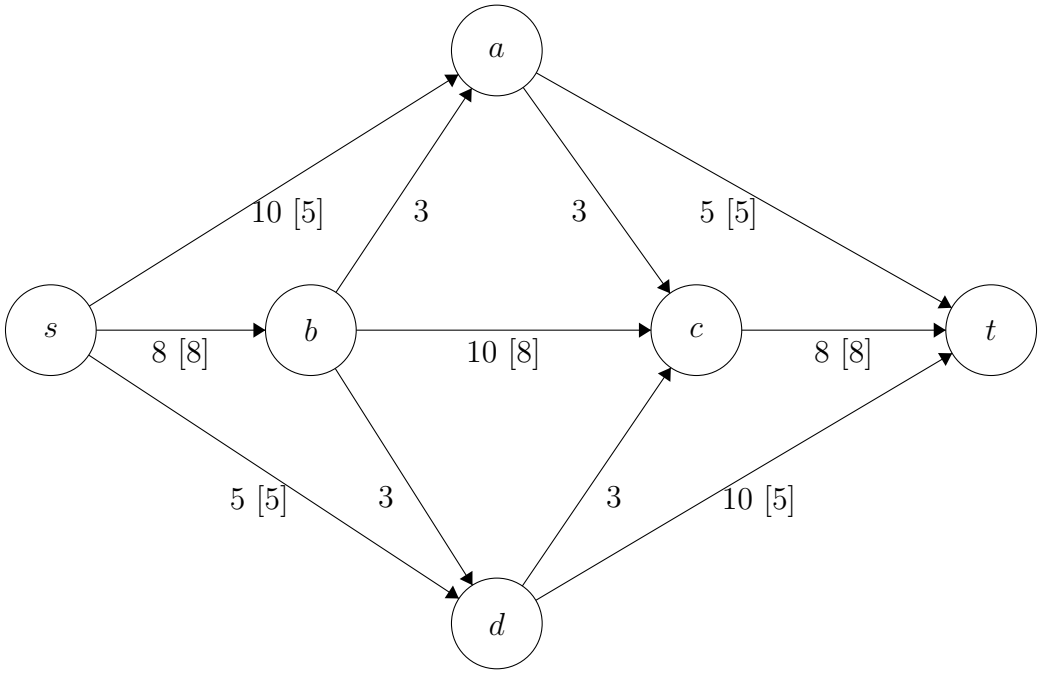
1. $\{s\}, \{u, v, t\}$
2. $\{s, v\}, \{u, t\}$
3. $\{s, v, u\}, \{t\}$

Problem Q3(b). What is the minimum capacity of an $s - t$ cut in the flow network in Figure 7.25 (reproduced below)?

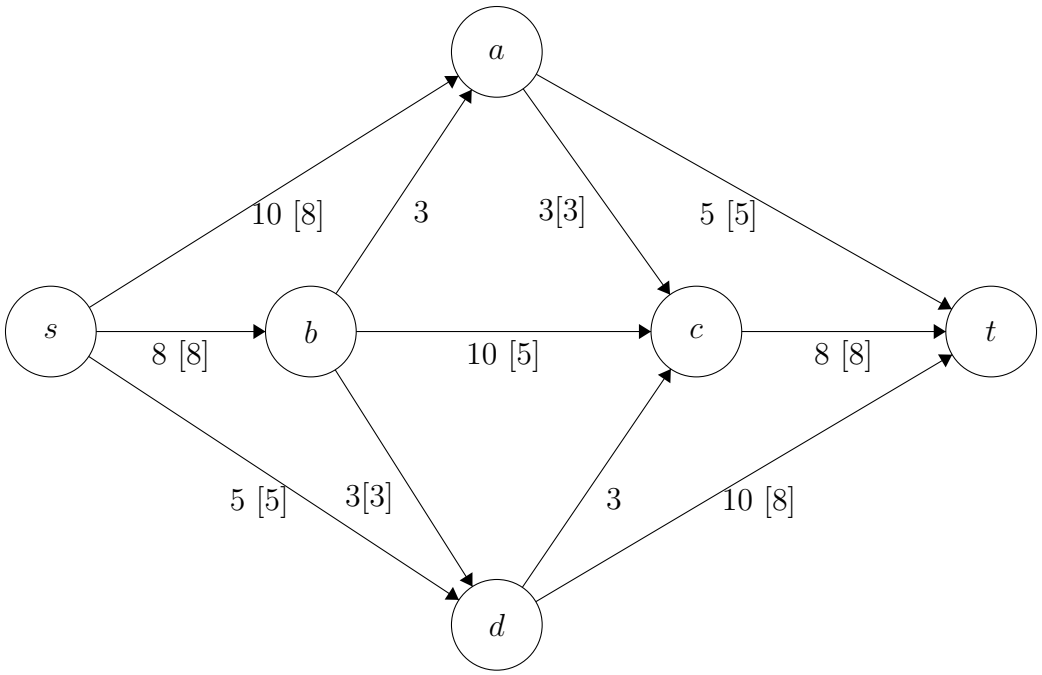


Minimum capacity: 4 (The cutset for $\{s, v\}$ contains edges (s, u) and (v, t) each of weight 2)

Problem Q4(a). What is the value of the $s - t$ flow depicted in figure 7.26? Is this a maximum (s, t) flow in this graph? The figure is reproduced below with additional node labels.



$v(f) = 18$, this is not a maximum flow.
There exists, trivially, a flow of higher value shown below.



Problem Q4(b). Find a minimum $s-t$ cut in the flow network pictured in figure 7.26, and also say what its capacity is.
A minimum $s-t$ cut is $\{s, a, b, c\}, \{d, t\}$ with a capacity of 21.

Problem Q5. Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible *base stations*. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations; the positions of each of these is specified by (x, y) coordinates as well. For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways.

- There is a *range parameter* r - a client can only be connected to a base station that is within distance r.
- There is also a *load parameter* L - no more than L clients can be connected to any single base station.

Your goal is to design a polynomial-time algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station, subject to the range and load conditions in the previous paragraph.

1: Create a graph $G(V, E)$
2: Let some edge $e(a, b)_w$ be a directed edge from a to b with weight w
3: Let the set of clients be denoted $C = \{c_1, c_2, \ldots, c_n\}$
4: Let the set of base stations be denoted $B = \{b_1, b_2, \ldots, b_n\}$
5: Define the function $valid(c_i)$ to return some set $B' = \{b'_1, b'_2, \ldots, b'_n\} \subset B$ such that c_i is within range $\forall b_i \in B'$
6: $V = \{s, t, c_1, c_2, \ldots, c_n, b_1, b_2, \ldots, b_n\}$
7: Initially $E = \{\}$

8: **for** $\forall v_i \in V$ **do**
9: **if** $v_i \in C$ **then**
10: $B' = valid(v_i)$
11: $E = E \cup \{(s, v_i)_1, (v_i, b'_1)_1, \ldots, (v_i, b'_n)_1\}$
12: **end if**
13: **if** $v_i \in B$ **then**
14: $E = E \cup \{(b_i, t)_{L_{b_i}}\}$
15: **end if**
16: **end for**
17: Run Ford-Fulkerson on G to find maximum flow f
18: **if** $f = n$ **then**
19: **return** TRUE
20: **else**
21: **return** FALSE
22: **end if**

Correctness

Proof. The sum of all the edge capacities leaving s is n
If some client c_i cannot be connected to a base station, it is either out of range or overloaded for all base stations.
If it is out of range, there exists no edge between c_i and b_i, so there can be no flow between them.
If it is overloaded, no flow can go from c_i to any b_i
Therefore the maximum flow will be less than n, as there is no other path from c_i to t by construction.
Therefore, the algorithm will never return false positives.
Suppose Ford-Fulkerson returns $f = n$ when there exists no matching.
This means that some client c_i cannot connect to any base station b_i.
However, there exists some b_i that c_i connected to the Ford-Fulkerson algorithm.
By line 14, b_i is connected to at most L_{b_i} clients.
This means b_i has enough capacity to accept c_i.
By line 11, b_i is in range.
Contradiction.
Therefore, the algorithm returns no false positives and no false negatives.

□

Runtime

Proof. The graph construction takes $n + k$ (polynomial) time, and Ford-Fulkerson runs in polynomial time.
Therefore, the total runtime is polynomial.

□