CS31005 Algorithms – II, Autumn 2021–2022

Third Test

Nov 16, 2021, 2:15pm — Maximum marks: 45

Duration: 80 minutes (65 minutes for answering questions + 15 minutes for download/submission)

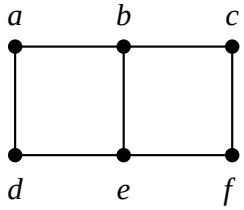
All your answers MUST BE HANDWRITTEN on paper. Scan all papers with your answers in a SINGLE pdf, and upload it as the answer to the Quiz in Moodle. The size of the final pdf must be less than 10 MB. You must upload the pdf strictly by 3:35 pm Moodle server time, the quiz submission will close after that.

Answer all questions.

1. Consider the following algorithm for computing the maximum (unweighted) cut in an undirected graph $G = (V, E)$. Here, $S - u$ means remove u from S, and $S + v$ means add v to S. Likewise, for $T + u$ and $T - v$.

> Start with an arbitrary cut (S, T) of V.
> while (1) {
> If for some $u \in S$, moving u from S to T increases the cut size, then set $(S, T) = (S - u, T + u)$,
> else if for some $v \in T$, moving v from T to S increases the cut size, then set $(S, T) = (S + v, T - v)$,
> else break;
> }
> Return (S, T).

This algorithm is run on the following graph with the initial cut $S = \{a, b, c\}$ and $T = \{d, e, f\}$.



Explain the workings of the iterations of the above algorithm in the format given below. **(8)**

	S	T	Number of neighbors in own part						Number of neighbors in other part					
			a	b	c	d	e	f	a	b	c	d	e	f
Initialization	$\{a,b,c\}$	$\{d,e,f\}$												
After Iteration 1														
After Iteration 2														
⋮														

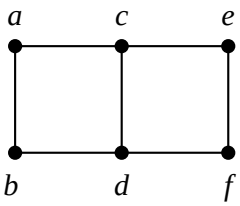
Solution

	S	T	Number of neighbors in own part						Number of neighbors in other part					
			a	b	c	d	e	f	a	b	c	d	e	f
Initialization	$\{a,b,c\}$	$\{d,e,f\}$	1	2	1	1	2	1	1	1	1	1	1	1
After Iteration 1	$\{a,c\}$	$\{b,d,e,f\}$	0	1	0	1	3	1	2	2	2	1	0	1
After Iteration 2	$\{a,c,e\}$	$\{b,d,f\}$	0	0	0	0	0	0	2	3	2	2	3	2

1. Consider the following algorithm for computing the maximum (unweighted) cut in an undirected graph $G = (V, E)$. Here, $S - u$ means remove u from S, and $S + v$ means add v to S. Likewise, for $T + u$ and $T - v$.

> Start with an arbitrary cut (S, T) of V.
> while (1) {
> If for some $u \in S$, moving u from S to T increases the cut size, then set $(S, T) = (S - u, T + u)$,
> else if for some $v \in T$, moving v from T to S increases the cut size, then set $(S, T) = (S + v, T - v)$,
> else break;
> }
> Return (S, T).

This algorithm is run on the following graph with the initial cut $S = \{a, b, c\}$ and $T = \{d, e, f\}$.



Explain the workings of the iterations of the above algorithm in the format given below. **(8)**

			Number of neighbors in own part						Number of neighbors in other part					
	S	T	a	b	c	d	e	f	a	b	c	d	e	f
Initialization	$\{a,b,c\}$	$\{d,e,f\}$												
After Iteration 1														
After Iteration 2														
$\vdots$														

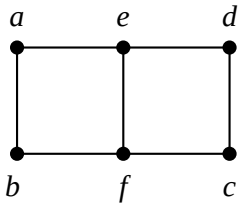
Solution

			Number of neighbors in own part						Number of neighbors in other part					
	S	T	a	b	c	d	e	f	a	b	c	d	e	f
Initialization	$\{a,b,c\}$	$\{d,e,f\}$	2	1	1	1	1	2	0	1	2	2	1	0
After Iteration 1	$\{b,c\}$	$\{a,d,e,f\}$	0	0	0	1	1	2	2	2	3	2	1	0
After Iteration 2	$\{b,c,f\}$	$\{a,d,e\}$	0	0	0	0	0	0	2	2	3	3	2	2

1. Consider the following algorithm for computing the maximum (unweighted) cut in an undirected graph $G = (V, E)$. Here, $S - u$ means remove u from S, and $S + v$ means add v to S. Likewise, for $T + u$ and $T - v$.

> Start with an arbitrary cut (S, T) of V.
> while (1) {
> If for some $u \in S$, moving u from S to T increases the cut size, then set $(S, T) = (S - u, T + u)$,
> else if for some $v \in T$, moving v from T to S increases the cut size, then set $(S, T) = (S + v, T - v)$,
> else break;
> }
> Return (S, T).

This algorithm is run on the following graph with the initial cut $S = \{a, b, c\}$ and $T = \{d, e, f\}$.



Explain the workings of the iterations of the above algorithm in the format given below. **(8)**

			Number of neighbors in own part						Number of neighbors in other part					
	S	T	a	b	c	d	e	f	a	b	c	d	e	f
Initialization	$\{a,b,c\}$	$\{d,e,f\}$												
After Iteration 1														
After Iteration 2														
$\vdots$														

Solution

			Number of neighbors in own part						Number of neighbors in other part					
	S	T	a	b	c	d	e	f	a	b	c	d	e	f
Initialization	$\{a,b,c\}$	$\{d,e,f\}$	1	1	0	1	2	1	1	1	2	1	1	2
After Iteration 1	$\{a,b,c,e\}$	$\{d,f\}$	2	1	0	0	1	0	0	1	2	2	2	3
After Iteration 2	$\{b,c,e\}$	$\{a,d,f\}$	0	0	0	0	0	0	2	2	2	2	3	3

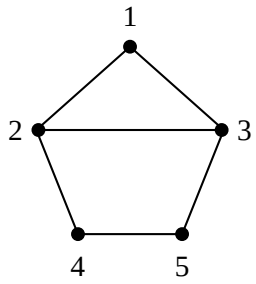
Grading Guidelines

- One mark for each non-empty cell of the table.
  - 2 in Col 2 (S and T),
  - 3 in Col 3 (neighbors in own part)
  - 3 in Col 4 (neighbors in other part)

2. Consider an undirected graph $G = (V, E)$ with n nodes numbered $1, 2, 3, \ldots, n$. We want to find a maximum independent set S of G (the maximum sized subset S of V such that there is no edge between any two nodes in S). The following Branch-and-Bound algorithm is proposed for it (notations used are from slides given).

- Structure of solution S : $\langle x_1, x_2, x_3, \ldots, x_n \rangle$ with $x_i = 1$ if node i is in S, 0 otherwise.
- $F(S) = \sum_{k=1}^{n} x_k$.
- Initial value of v_{lower} (lower bound of optimal value) = 1.
- For any partial solution $S_i = \langle x_1, x_2, x_3, \ldots, x_i \rangle$, $i < n$:
  - $F(S_i) = \sum_{k=1}^{i} x_k$.
  - S_i is feasible if for any x_j, x_k, $0 < j, k \leqslant i$, $j \neq k$, if $x_j = x_k = 1$ then $(x_j, x_k) \notin E$.
  - $v_{upper}(i+1) = (n-i)$ (that is, if all remaining nodes are included in the independent set).
- Order of traversal is DFS with $x_i = 1$ generated first (left child) for each i.
- A live node is expanded if and only if it represents a feasible partial solution and there is a chance of finding a better optimal solution starting from it.

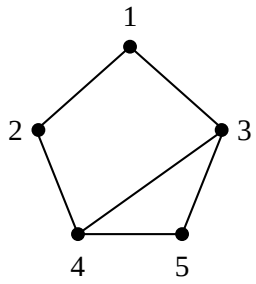
Show the complete state-space tree that will be generated if the above algorithm is applied on the following graph. **(7)**



2. Consider an undirected graph $G = (V, E)$ with n nodes numbered $1, 2, 3, \ldots, n$. We want to find a maximum independent set S of G (the maximum sized subset S of V such that there is no edge between any two nodes in S). The following Branch-and-Bound algorithm is proposed for it (notations used are from slides given).

- Structure of solution S : $\langle x_1, x_2, x_3, \ldots, x_n \rangle$ with $x_i = 1$ if node i is in S, 0 otherwise.
- $F(S) = \sum_{k=1}^{n} x_k$.
- Initial value of v_{lower} (lower bound of optimal value) = 1.
- For any partial solution $S_i = \langle x_1, x_2, x_3, \ldots, x_i \rangle$, $i < n$:
  - $F(S_i) = \sum_{k=1}^{i} x_k$.
  - S_i is feasible if for any x_j, x_k, $0 < j, k \leqslant i$, $j \neq k$, if $x_j = x_k = 1$ then $(x_j, x_k) \notin E$.
  - $v_{upper}(i+1) = (n-i)$ (that is, if all remaining nodes are included in the independent set).
- Order of traversal is DFS with $x_i = 1$ generated first (left child) for each i.
- A live node is expanded if and only if it represents a feasible partial solution and there is a chance of finding a better optimal solution starting from it.

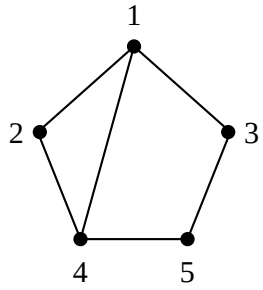
Show the complete state-space tree that will be generated if the above algorithm is applied on the following graph. **(7)**



2. Consider an undirected graph $G = (V, E)$ with n nodes numbered $1, 2, 3, \ldots, n$. We want to find a maximum independent set S of G (the maximum sized subset S of V such that there is no edge between any two nodes in S). The following Branch-and-Bound algorithm is proposed for it (notations used are from slides given).

- Structure of solution S : $\langle x_1, x_2, x_3, \ldots, x_n \rangle$ with $x_i = 1$ if node i is in S, 0 otherwise.
- $F(S) = \sum_{k=1}^{n} x_k$.
- Initial value of v_{lower} (lower bound of optimal value) = 1.
- For any partial solution $S_i = \langle x_1, x_2, x_3, \ldots, x_i \rangle$, $i < n$:
  - $F(S_i) = \sum_{k=1}^{i} x_k$.
  - S_i is feasible if for any x_j, x_k, $0 < j, k \leqslant i$, $j \neq k$, if $x_j = x_k = 1$ then $(x_j, x_k) \notin E$.
  - $v_{upper}(i+1) = (n-i)$ (that is, if all remaining nodes are included in the independent set).
- Order of traversal is DFS with $x_i = 1$ generated first (left child) for each i.
- A live node is expanded if and only if it represents a feasible partial solution and there is a chance of finding a better optimal solution starting from it.

Show the complete state-space tree that will be generated if the above algorithm is applied on the following graph. **(7)**



Full tree for all variants of the question not shown. Most students did ok on this one but with some typical problems in most cases. Here is the grading policy, please contact me if you feel there is any mistake by me after reading my feedback remarks.

1. Many of you have pruned only infeasible nodes, no pruning is done based on v_lower (current best solution known). This is just backtracking, not branch-and-bound. 2 marks deducted.
2. Many of you pruned too early, so there are many missing nodes. The question says not to expand a node if it is infeasible, but the infeasible node will still be generated (as child of a feasible node) before you can check if it is infeasible. For example, in all variants, the node for $x_1 = 1, x_2 = 1$ will be generated, and then not expanded any more as this cannot lead to a feasible solution as $(1, 2)$ is an edge in the graph. 2 marks deducted.
3. There are other cases where you have a mixture of above plus other problems. Marked accordingly.

3. Ms. Trotter wants to make a sightseeing tour to n locations. She obtains the cost $c_{i,j}$ of traveling from Location i to Location j for all i, j. She plans to visit each sightseeing location at most once, and eventually come back to the location from which she starts. She however has a limited budget B, and can afford a tour if and only if the total cost of the tour is no more than B. She needs to identify the maximum number m of locations she can visit subject to her budgetary constraint. You may assume that all $c_{i,j}$ and B are positive integers, and that $c_{i,j} = c_{j,i}$ for all i, j. The costs need **not** satisfy the triangle inequality.

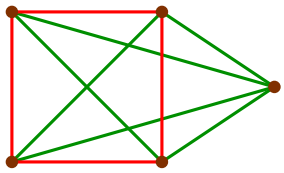
(a) Ms. Trotter's problem is an optimization (maximization) problem. Propose an equivalent decision version of the problem. You need to prove that the optimization problem can be solved in polynomial time if and only if the decision problem can be solved in polynomial time. **(5)**

Solution The decision version is: Given the inter-location costs $c_{i,j}$, the budget B, and a positive integer m, decide whether a tour of **at least** m of the given locations is possible subject to the budgetary constraint.

Equivalence: Clearly, if the optimization problem returns m^* is polynomial time, one only needs to check whether $m^* \geqslant m$. Conversely, let D be a polynomial-time algorithm for the decision version. One needs to invoke D for $m = n, n-1, n-2, \ldots$ until D returns *Yes*.

2 marks for the correct formulation, 1.5 marks for each direction of the equivalence proof.

Consider the decision version: Given the inter-location costs $c_{i,j}$, the budget B, and a positive integer m, decide whether a tour of **exactly** m of the given locations is possible subject to the budgetary constraint. This version is not equivalent to the optimization version. Of course, you can run a decider of this problem for $m = n, n-1, n-2, \ldots$ as above to find the largest value of m giving a tour meeting the budget. The converse cannot be established. Consider the case of five locations in the figure below. Let the budget be $B = 5$. The algorithm for the optimization problem outputs $m^* = 4$ (the red square).



Each red edge has cost 1

Each green edge has cost 10

This implies that the output of the decision problem for $m = 5$ must be *no*. But it turns out that the decision version for $m = 3$ also has output *no*. By only comparing m with the optimal m^*, you cannot always answer the decision problem. You may think of running the optimization algorithm on subgraphs with smaller numbers of nodes. But there may be exponentially many subgraphs with numbers of nodes in the range $[m, n]$, and polynomial equivalence fails.

Also note that binary search is faulty for the bad formulation. That is, if you ask the decider with $m = 3$, it says *no*. Where will you jump? $m > 3$ or $m < 3$?

(b) Prove that the decision version of Ms. Trotter's problem is NP-Complete. **(5)**

Solution The problem is certainly in NP. The tour itself provides a certificate. The total cost of the tour can be computed and compared with B in polynomial-time.

In order to prove the NP-hardness of the problem, we make a reduction from the HAM-CYCLE problem. Let $G = (V, E)$ be an instance of HAM-CYCLE with $|V| = n$. Take $c_{i,j} = 1$ if $(i, j) \in E$ and $c_{i,j} = 2$ if $(i, j) \notin E$. Finally, take $m = n$ and $B = n$. Clearly, G has a Hamiltonian cycle if and only if there is a tour of n cities of total budget n.

1 mark for considering that Ms. Trotter's problem is in NP. 4 marks for proposing a reduction from a known NP-complete problem. In the solution, a reduction from HAM-CYCLE is given. A reduction from TSP can also be easily arrived at. Whatever the source problem is, you need to clearly specify the input and the output of the reduction algorithm. An instance of Ms. Trotter's problem must consist of a *complete* graph with all edge costs $c_{i,j}$ specified, a budget B, and a target m for the number of locations.

4. Let $A = (a_1, a_2, \ldots, a_n)$ be an array of n positive integers, and t a target sum (a positive integer again). The task is to find a subset $I \subseteq \{1, 2, 3, \ldots, n\}$ for which the sum $\sum_{i \in I} a_i$ is as small as possible but at least as large as t. Assume that $t \leqslant \sum_{i=1}^{n} a_i$ (otherwise the problem has no solution). Prof. Sad proposes the following algorithm to solve this problem.

```
Initialize sum = 0, and I = ∅.
for i = 1,2,3,...,n (in that order), repeat {
    Update sum = sum + a_i, and I = I ∪ {i}.
    If sum ⩾ t, break.
}
Return I.
```

(a) Prove that the approximation ratio of Prof. Sad's algorithm cannot be restricted by any constant value. **(3)**

Solution Take any constant $\Delta \geqslant 1$ (may be very large). Consider the instance $n = 2$, $A = (\Delta + 1, 1)$, and $t = 1$. Then, Prof. Sad's algorithm returns the index set $\{1\}$ of sum $\Delta + 1$, whereas the optimal solution is $\{2\}$ of sum 1. So the approximation ratio is $\Delta + 1 > \Delta$.

Give an explicit example showing the array A and the target t. You cannot assume that the target t is achievable from every array.

(b) Prof. Atpug suggests sorting the array A in the ascending order before running the algorithm. In view of this suggestion, we now have $a_1 \leqslant a_2 \leqslant a_3 \leqslant \cdots \leqslant a_n$. Prove that Prof. Atpug's suggestion makes Prof. Sad's algorithm a 2-approximation algorithm. **(7)**

Solution We have $I = \{1, 2, 3, \ldots, k\}$ for some $k \geqslant 1$ satisfying $a_1 + a_2 + \cdots + a_{k-1} < t$ but $a_1 + a_2 + \cdots + a_{k-1} + a_k \geqslant t$. Now, consider the two cases.

Case 1: $a_k < t$. Here, $a_1 + a_2 + \cdots + a_{k-1} + a_k = (a_1 + a_2 + \cdots + a_{k-1}) + a_k < t + t = 2t$, whereas the optimal sum satisfies OPT $\geqslant t$.

Case 2: $a_k \geqslant t$. Here, the optimal sum is OPT $= a_k$ (because the preceeding numbers $a_1, a_2, \ldots, a_{k-1}$ (if any) add up to a sum smaller than t, whereas $a_{k+1}, a_{k+2}, \ldots, a_n$ are each at least as large as a_k). But then, $a_1 + a_2 + \cdots + a_{k-1} < t \leqslant a_k$, whereas the algorithm gives the sum $(a_1 + a_2 + \cdots + a_{k-1}) + a_k < a_k + a_k = 2a_k$.

1 mark for the first line, and 3 marks each for handling each of the cases.

5. **(a)** Consider a group of 1000 people, divided into 10 groups, with each group containing exactly 125 people. Note that this means some people will appear in more than one group. It is also given that at least one person is different between any two groups. We wish to choose a group of people (from the 1000 people) as leaders so that there is at least one leader and at least one non-leader in each group. Propose a Monte Carlo algorithm for the problem with an error bound less than $1/2^{10}$. Briefly justify your answer. **(3)**

Solution Easy 2-line answer is to notice that (i) we can use the Las Vegas algorithm for the ball-coloring problem done in class (Red = Leader, Blue = non-leader), and then (ii) convert it to a Monte Carlo algorithm using the method done in class ($c = 210$). However, you can also solve it directly, but then you need to show me the probability computation. Graded accordingly depending on how clear your answer is. Note that just because you pick randomly does not make it Monte Carlo. It is not a Monte Carlo algorithm if the answer is always correct. The question asked you to pick the leader from the 1000 people. If you do either of these, you lost 2 marks: (i) pick one randomly from each group, or (ii) pick one, find all the group it covers, remove the groups and the picked one, then pick another, and continue until all groups are covered.

(b) Consider a document-storage system. Each document is identified with a set of keywords. The documents are stored in two servers X_1 and X_2, with X_1 holding all documents containing any keyword from a set of keywords K_1, and X_2 holding documents with any keyword from a set of keywords K_2. The sets K_1 and K_2 are disjoint. However, note that since a document can contain keywords from both K_1 and K_2, the set of documents stored by the servers need not be disjoint.

A user queries for documents containing a set of keywords K to a central server C. Assume that K can contain keywords from only K_1, only K_2, or from both K_1 and K_2. Server C returns all documents from X_1 and X_2 that contain ALL of the keywords in K. To do this, C sends the keywords in K to X_1 and X_2, and they communicate to send the final set of documents to C to be sent back to the user. However, you cannot send any keyword from set K_2 to X_1, or any keyword from set K_1 to X_2 at any stage. Your goal is to reduce the number of bits transferred between the servers. Suggest a Bloom filter based scheme to do this. Note that the final answer should be fully correct (no false positives). You need not try to reduce computation at a server or total time taken. For the bloom filter used, you do not have to design any hash functions exactly, just say what it will contain, and how it will be used.

List all the steps clearly, starting from C receiving the query from the user till C sending the response to the user. **(7)**

Solution No one got it at all! The thing to notice (which all of you missed, not many answered it anyway) is that optimizing computation is not important (clearly told in the question), optimizing communication is (again clearly told in the question). Most of you did Bloom filter things to reduce computation at servers, that too in many cases very vaguely. Also note that just because you used hashing does not make it a bloom filter. Also, documents are what causes most of the communication, keywords are small. I don't think I have given more than 2–3 to anyone attempting it, will not also unless you can point out a mistake by me. Most answers do not even talk about transmitting anything.

The naïve solution not using Bloom Filter is trivial. C sends $K' = K \cap K_1$ to X_1, and $K'' = K \cap K_2$ to X_2. X_1 finds the set S_1 of all documents containing all keywords in K'. Similarly X_2 finds the set S_2 of all documents containing all keywords in K''. The problem is then trying to find the intersection of the sets S_1 and S_2. So X_1 can send the set S_1 to X_2, X_2 can compute the intersection and send it to C. Or you can reverse the roles of X_1 and X_2 also. Note that since C only gets the final set, $S_1 \cap S_2$ will always have to be transmitted at least once (by X_1 or by X_2) to C whatever you do. So the question basically boils down to how do you compute the intersection of two sets S_1 and S_2 without having to send either S_1 or S_2 over a link. At least you needed to see that you need to do something with the set intersection.

Here comes Bloom filter. Instead of sending S_1 to X_2, X_1 will hash the documents in S_1 in a Bloom filter B, and send B to X_2. X_2 will now check if each element of S_2 is in B, and if yes, add it to a new set S_K. So S_K will contain $S_1 \cap S_2$, plus some elements in S_2 but not in S_1 because of the false positives from B. X_2 now sends S_K to X_1, which checks it again to filter out the false positives (remember computation minimization is not important in this question), and sends the final set to C. So rather than sending all documents in S_1 over the link, you send the intersection of S_1 and S_2 only (which is expected to be much smaller), plus the size of the bloom filter (only 1 bit per document in S_1) and some extra documents due to the false positives, which can also be made very small.