# CSE 40113: Algorithms

## Sample Final Exam – Spring 2025

| Name: Solutions | Email Address: |
|---|---|

---

1. This is a closed-book and closed-notes exam. You are allowed up to 4 "cheat sheets" on standard 8.5 by 11 inch paper, handwritten on the front and back.
2. Print your full name and your email address in the boxes above.
3. Print your name or initials at the top of every page.
4. Please write clearly and legibly. If I can't read your answer, I can't give you credit.
5. When asked to design an algorithm, you must also provide a runtime analysis for that algorithm. However, proofs of correctness are NOT required for algorithms on this exam. However, problems that explicitly ask you to prove something still require you to do proofs!
6. There are 7 problems on the exam. Your grade will be calculated based only on 6 out of the 7 problems. Please feel free to try all of them, or at least write "I don't know" on the one you skip; I'll take the maximum 6 for your final grade.
7. Remember, these are NOT necessarily in order of difficulty. Please read all the problems first, and don't allow yourself to get stuck on a single problem.

| # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Total |
|---|---|---|---|---|---|---|---|---|
| Max | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 120 |
| Score | | | | | | | | |

1. You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cites that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where exactly should you meet?

   You are given a weighted graph $G = (V, E)$, where the vertices $V$ represent cities and the edges $E$ represent roads that directly connect cities. Each edge $e$ has a weight $w(e)$ equal to the time required to travel between the two cities. You are also given a vertex $p$, representing your starting location, and a vertex $q$, representing your friend's starting location. Describe and analyze an algorithm to find the target vertex $t$ that allows you and your friend to meet as soon as possible, assuming both of you leave home right now.

I assume no negative weights, since w(e) = travel time.
I accepted Bellman-Ford also, if students were unsure.

Shortest paths:

Dijkstra(p)
  ↳ store lengths in p-array[1..V]

Dykstra(q)
  ↳ store lengths in q-array[1..V]

initalize dist ← ∞

for each v ∈ V
  if q-array[v] + p-array[v] < dist
    dist ← q-array[v] + p-array[v]

return dist

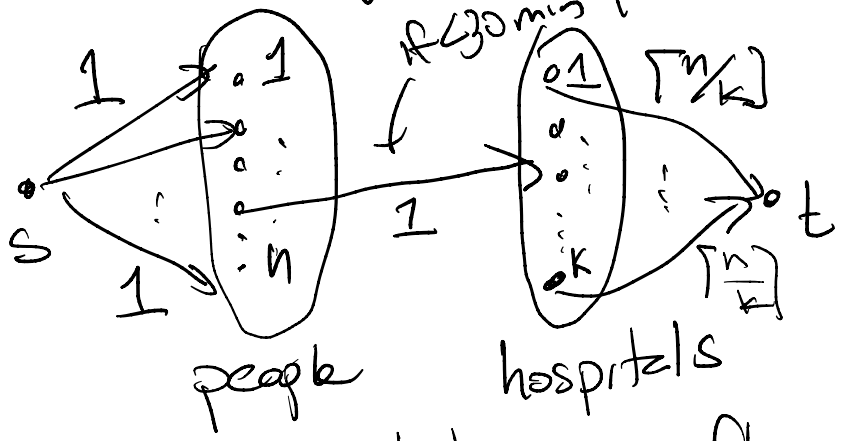
Runtime: 2 · Dykstra + O(V)
$\overset{1}{=}$ O(E log V)

2. Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of $n$ injured people distributed across the region who need to be rushed to hospitals. There are $k$ hospitals in the region, and each of the $n$ people needs to be brought to a hospital that is within a half-hour's driving time of their current location, so different people will have different options for hospitals, depending on where they are right now. (You may assume that for any person and hospital, you can can look up in $O(1)$ time if they are within a half hour.)

   At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced, so that each hospital receives at most $\lfloor n/k \rfloor$ people.

   Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible.

Model as flow: Create s & t, & one vertex per person, one vertex per hospital




Edges:

- For each person p, create s→p edge w/ cap = 1
- For each hospital, create h→t edge w/ cap = $\lceil \frac{n}{k} \rceil$
- For each person p & hospital h, create p→h edge if within a half hour, capacity = 1

Then calculate max flow. If val = n, true, else false.

Construction: $V = n+k+2$

$E \leq n+k+nk$

& Orlin = $O(VE)$, so

total = $O((n+k) \cdot nk)$

3. You have a collection of $n$ lockboxes and $m$ gold keys. Each key unlocks at most one box. Without a matching key, the only way to open a box is to smash it with a hammer. Your baby brother has locked all your keys inside the boxes! Luckily, you know which keys (if any) are inside each box; this is given to you in the form of $box[1 \ldots n]$, where each $box[i]$ is a list of the keys saved in that box. You may assume that each box stores at least one key, and that $m \geq n$.

   (a) Your baby brother has found the hammer and is eagerly eyeing one of the boxes. Describe and analyze an algorithm to determine if it is possible to retrieve all the keys without smashing any box except the one your brother has chosen.

Make a vertex per box, with adj. list = box[i] (directed edges)
Let s ← your brother's box
DFS(s)
If all vertices visited, true else false.

V = n
E = O(n²)
O(V+E) → total time = O(n²)

   (b) Describe and analyze an algorithm to compute the minimum number of boxes that must be smashed to retrieve all the keys.

Calculate Strongly connected components (from book)
↳ O(V+E) = O(n²)
Then, each source in resulting SCC DAG is necessary to smash, so do topological sort → O(V+E) = O(n²)
Total time: O(n²)

4. After the Revolutionary War, Alexander Hamilton's biggest rival as a lawyer was Aaron Burr. (Sir!) In fact, the two worked next door to each other. Unlike Hamilton, Burr cannot work non-stop; every case he tries exhausts him. The bigger the case, the longer he must rest before he is well enough to take the next case. If a case arrives while Burr is resting, Hamilton snatches it up instead.

   Burr has been asked to consider a sequence of n upcoming cases. He quickly computes two arrays `profit`$[1 \ldots n]$ and `skip`$[1 \ldots n]$, where for each index $i$,

   - `profit`$[i]$ is the amount of money Burr would make by taking the $i^{th}$ case, and
   - `skip`$[i]$ is the number of consecutive cases Burr must skip if he accepts the $i^{th}$ case

   That is, if Burr accepts the $i^{th}$ case, he cannot accept cases $i+1$ through $i+$`skip`$[i]$. Design and analyze an algorithm that determines the maximum total profit Burr can secure from these $n$ cases, using his two arrays as input.

Let val[i] = max amount Burr can earn for jobs i..n

```
val[n] ← profit[n]
for i ← n-1 down to 1
    without ← val[i+1]
    with ← profit[i]
    if i+skip[i] ≤ n
        with ← with + val[i+skip[i]]
    val[i] ← max{with, without}
return val[1]
```

O(1) time per iteration

Runtime: $O(n)$

5. Given two graphs $G$ and $H$, the subgraph isomorphism problems asks if $G$ has contains exact copy of $H$ somewhere inside of it.

   A bit more formally, given $G = (V, E)$ and $H = (V', E')$, we are asking if there is a function $f : V' \to V$ such that $f(V') \subseteq V$ and for every edge $uv \in E'$, $f(u)f(v)$ is also an edge in $E$.

   Prove that Subgraph Isomorphism is NP-Complete.

In NP: Certificate is vertex map from $V' \to V$.
In $O(E')$ time, check each edge is present in $G$. ✓

Reduce from Hamiltonian path

Input: graph $G=(V,E)$
Convert to subgraph isomorphism:
Let $G$ be same
Let $H$ = a path on $V$ vertices
Time spent: $O(V)$ (to create $H$)

$\Leftarrow$: If $G$ has a Ham. path, then $G$ contains a path on $V$ vertices, so $H$ is a subgraph

If $H$ is a subgraph of $G$, then

we know that G contains a path on V vertices.
Any such path is by definition a Hamiltonian path. ☑

---

Alternative reduction: from k-Clique.

Input: $G=(V,E)$ and $k$

Let G for subgraph iso. be our input $k$.

For H, create a graph with $k$ vertices and all possible $\binom{k}{2}$ edges.

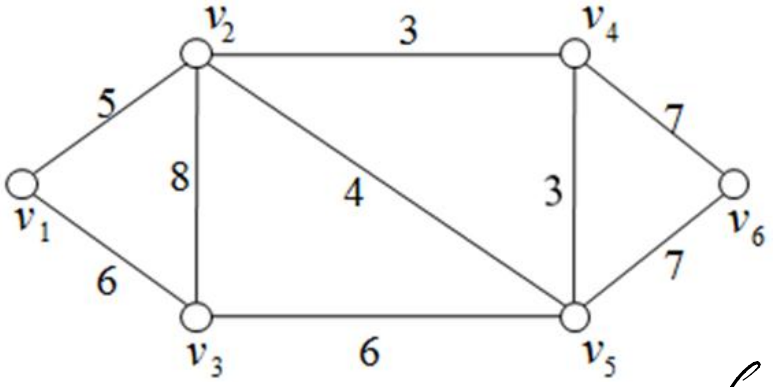
Time: $O(V+E+k^2)$

$\Leftarrow$ if G has a k-clique, then that clique gives our H.

if H is a subgraph, then some $k$ vertices have all edges

$\Rightarrow$ we have a k-clique ☑

6. Find the following objects for the weighted graph shown below:




(a) A breadth-first spanning tree rooted at $v_1$.

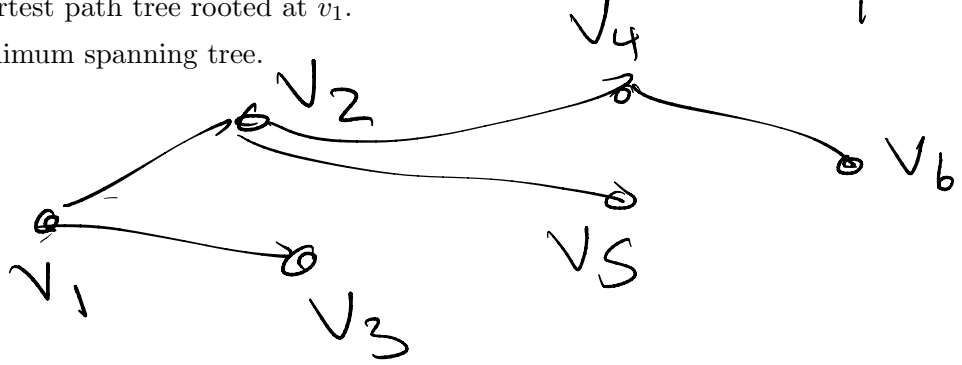
(b) A depth-first spanning tree rooted at $v_1$.

(c) A shortest path tree rooted at $v_1$.

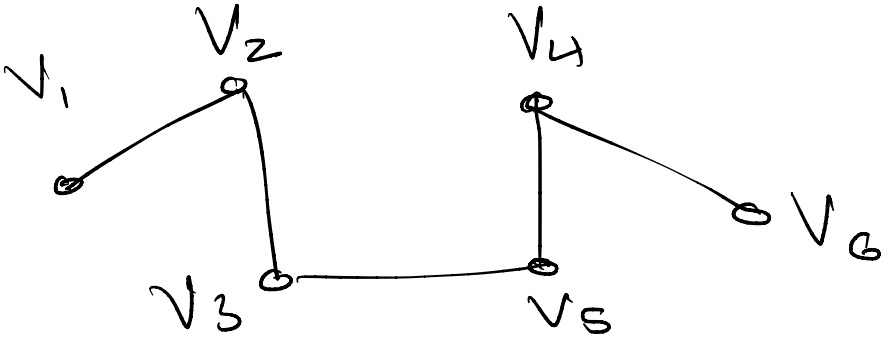
(d) A minimum spanning tree.

Note: Any BFS, DFS, + MST tree was acceptable. (I broke ties by index)

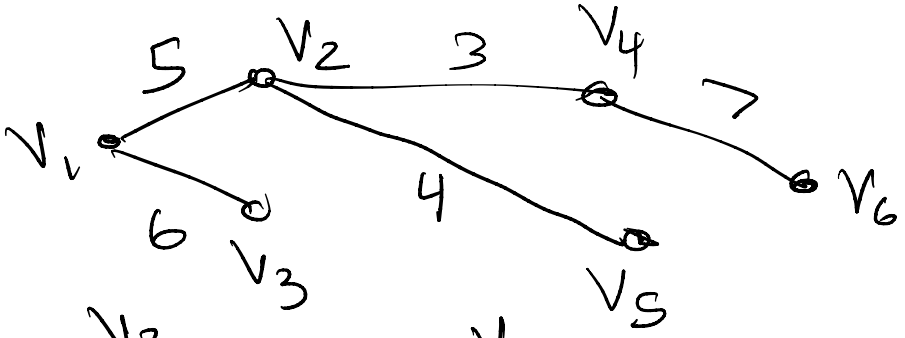
a)



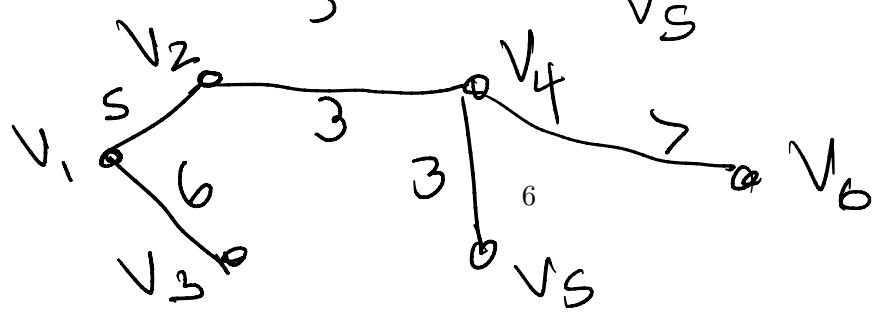

b)




c)




d)



7. Suppose you are given a directed graph $G = (V, E)$, each of whose edges are colored red, green, or blue. Edges in $G$ do not have weights, and $G$ is not necessarily a DAG (directed acyclic graph). A rainbow walk is a walk in $G$ that does not contain two consecutive edges with the same color. Describe and analyze an algorithm to find all vertices in $G$ that are reachable from a given vertex $s$ through a rainbow walk.

Modify WFS:

Initalize 3 "visited by color" arrays:

red[1..V]
blue[1..V]
green[1..V]
} all false initally

red[s] ← true
blue[s] ← true
green[s] ← true
(s is visited by empty path)

for each edge s → v:
    add (s, v) to queue Q

next page for code

Runtime: BFS, still O(1) per vertex, so O(V+E)

```
while Q is not empty:
    remove (p,v) from Q
    if (p,v) is red:
        if red[v] is false:
            red[v] ← true
            for each edge (v,w)
                if (v,w) is blue or green
                    add (v,w) to Q

    if (p,v) is blue:
        if blue[v] is false
            blue[v] ← true
            for each edge (v,w)
                if (v,w) is green or red
                    add (v,w) to Q

    if (p,v) is green:
        if green[v] is false
            green[v] ← true
            for each edge (v,w)
                if (v,w) is red or blue
                    add (v,w) to Q

for each vertex v
    if (red[v]=true) or (blue[v]=true) or
                (green[v]=true)
        reachable[v] ← true
    else
        reachable[v] ← false
```

**You may assume the following problems are NP-Hard:**
(Note that I'm giving the decision versions here, which all have a yes/no answer.)

- CIRCUITSAT: Given a boolean circuit, are there any input values that make the circuit output True?
- 3SAT: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?
- INDEPENDENTSET: Given an undirected graph $G$ and a positive integer $k$, is there a subset of vertices in $G$ of size $k$ that have no edges among them?
- KCLIQUE: Given an undirected graph $G$ and a positive integer $k$, is there a clique in $G$ of size $k$?
- VERTEXCOVER: Given an undirected graph $G$ and a positive integer $k$, is there a subset of vertices of size $k$ that touch every edge in G?
- HITTINGSET: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set $S$, is there a subset of $S$ of size $k$ that intersects every subset $S_i$?
- HAMILTONIANCYCLE: Given a graph $G$, is there a cycle in G that visits every vertex exactly once?
- HAMILTONIANPATH: Given a graph $G$, is there a path in G that visits every vertex exactly once?
- TRAVELINGSALESMAN: Given a graph $G$ with weighted edges and a value $k$, is there a Hamiltonian path/cycle in $G$ of total weight $\leq k$?
- SUBSETSUM: Given a set $X$ of positive integers and an integer $k$, does X have a subset whose elements sum to k?
- PARTITION: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?
- 3COLORABLE: Given a graph, does it have a 3-coloring?

(scratch paper)

(scratch paper)