

CS 170 Homework 8

Due **Saturday 10/24/2026, at 10:00 pm** (grace period until 11:59pm)

1 Max Independent Set: Part Two (Solo Question)

You are given a binary tree T with n nodes and root r , where every node v has a weight $W[v]$. A set of nodes S is a k -independent set of T if $|S| = k$ and no two nodes in S share an edge in T . The weight of such a set is $W(S) = \sum_{v \in S} W[v]$. Given an integer $k \leq n$, find the maximum weight of any k -independent set of T .

Describe an $O(nk^2)$ algorithm and analyze its runtime and space complexity. Recall that in a binary tree, every node has at most 2 children.

Solution:

Subproblems. For each node v and $0 \leq i \leq k$, let

- $f_0[v][i]$ be the max weight of an independent set of size i in the subtree rooted at v that does not include v , and
- $f_1[v][i]$ be the max weight of any independent set of size i in the subtree rooted at v .

Set an entry to $-\infty$ if no such set exists.

Recurrence. Let L and R be the left and right children of v . Then

$$f_0[v][i] = \max_{0 \leq j \leq i} (f_1[L][j] + f_1[R][i - j]),$$

$$f_1[v][i] = \max \left(f_0[v][i], W[v] + \max_{0 \leq j \leq i-1} (f_0[L][j] + f_0[R][i - 1 - j]) \right),$$

where the second term inside f_1 is considered only when $i \geq 1$. We split the i chosen nodes between the two child subtrees. If v is included, it uses one of the i slots, and neither child may be chosen.

Base case. We treat a missing child as an empty subtree $\emptyset$, with $f_b[\emptyset][0] = 0$ and $f_b[\emptyset][i] = -\infty$ for $i \geq 1$, for $b \in \{0, 1\}$. This handles leaves automatically.

Answer. The answer is $f_1[r][k]$.

Pseudocode.

```

1: for  $v$  in post-order (leaves to root) do
2:   for  $i = 0$  to  $k$  do
3:     compute  $f_0[v][i]$ , then  $f_1[v][i]$ , using the recurrence
4: return  $f_1[r][k]$ 

```

Runtime. There are $2n(k+1) = O(nk)$ subproblems, and each takes $O(k)$ time to maximize over the split j . The total runtime is $O(nk^2)$.

Space. The two tables have $O(nk)$ entries in total, so the algorithm uses $O(nk)$ space.

2 Understanding Convex Polytopes

So far in this class we have seen linear programming defined as

$$(\mathcal{P}) = \begin{cases} \max & c^T x \\ \text{s.t.} & Ax \leq b. \end{cases}$$

Today, we explore the different properties of the region $\Omega = \{x : Ax \leq b\}$ – i.e. the region that our linear program maximizes over.

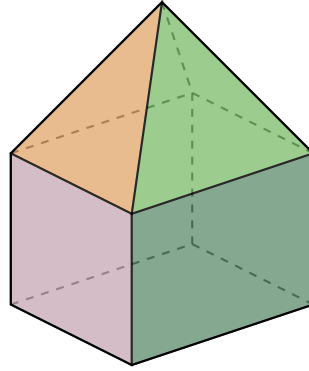


Figure 1: An example of a convex polytope. We can consider each face of the polytope as an affine inequality and then the polytope is all the points that satisfy each inequality. Notice that a polytope defines a half-plane and therefore is also the intersection of the half-planes.

- (a) The first property that we will be interested in is *convexity*. We say that a space X is convex if for any $x, y \in X$ and $\lambda \in [0, 1]$,

$$\lambda x + (1 - \lambda)y \in X.$$

That is, the entire line segment $\overline{xy}$ is contained in X . Prove that Ω is indeed convex.

Solution: Let $x, y \in \Omega$. We need to show that

$$A(\lambda x + (1 - \lambda)y) \leq b.$$

We apply the only facts we know, namely $Ax \leq b$ and $Ay \leq b$.

$$\begin{aligned} A(\lambda x + (1 - \lambda)y) &= \lambda Ax + (1 - \lambda)Ay \\ &\leq \lambda b + (1 - \lambda)b \\ &= b. \end{aligned}$$

- (b) The second property that we will be interested in is showing that linear objective functions over convex polytopes achieve their maxima at the vertices. A vertex is any point $v \in \Omega$ such that v **cannot** be expressed as a point on the line $\overline{yz}$ for $v \neq y, v \neq z$, and $y, z \in \Omega$.

Prove the following statement: Let Ω be a convex space and f a linear function $f(x) = c^T x$. Show that for a line $\overline{yz}$ for $y, z \in \Omega$, that $f(x)$ is maximized on the line at either y or z . I.e. show that

$$\max_{\lambda \in [0,1]} f(\lambda y + (1 - \lambda)z)$$

achieves the maximum at either $\lambda = 0$ or $\lambda = 1$. *Hint: Assume without loss of generality that $f(y) \geq f(z)$.*

Solution: Assume without loss of generality that $f(y) \geq f(z)$. (Otherwise, swap their names). Then $c^T y \geq c^T z$. We now aim to show the maximum is achieved at $\lambda = 1$. Then,

$$\begin{aligned} f(\lambda y + (1 - \lambda)z) &= c^T(\lambda y + (1 - \lambda)z) \\ &= \lambda c^T y + (1 - \lambda)c^T z \\ &\leq \lambda c^T y + (1 - \lambda)c^T y \\ &= f(y). \end{aligned}$$

- (c) Now, prove that global maxima will be achieved at vertices. For simplicity, you can assume there is a unique global maximum.

Hint: Use the definition of a vertex presented above. (Side note: This argument is the basis of the Simplex algorithm by Dantzig to solve linear programs.)

Solution: Assume, for contradiction, that the maximum was not achieved at a vertex and was instead achieved at a point x that was *not* a vertex. Then, there exists a line y, z containing x such that $x \neq y$ and $x \neq z$. But by the previous argument, the

function achieves a maximum at either y or z . then, the maximum isn't unique. A contradiction.

3 A Cohort of Secret Agents

A cohort of k secret agents residing in a certain country needs escape routes in case of an emergency. They will be traveling using the railway system, which we can think of as a directed graph $G = (V, E)$, with V being the cities and E being the railways. Each secret agent i has a starting city $s_i \in V$, and all s_i 's are distinct. Every secret agent needs to reach the consulate of a friendly nation; these consulates are in a known set of cities $T \subseteq V$. In order to move undetected, the secret agents agree that at most c of them should ever pass through any one city, including the cities where they start and end. Our goal is to find a set of paths, one for each secret agent, that satisfies these requirements, or to determine that no such set exists.

Model this problem as a flow network. Specify the vertices, edges, capacities, source, and sink, and describe how to transform a maximum flow in your network into a valid set of paths for the agents (or how to determine that none exists). You do not need to explain how to compute the maximum flow itself.

Solution: We can think of each secret agent i as a unit of flow that we want to move from s_i to any vertex in T . To do so, we can model the graph as a flow network by setting the capacity of each edge to ∞ , adding a new vertex t , and adding an edge (t', t) of capacity ∞ for each $t' \in T$. We can add a source s and edges of capacity 1 from s to s_i . By doing so, we restrict the maximum flow to be at most k .

Lastly, we need to ensure that no more than c secret agents are in a city. We want to add vertex capacities of c to each vertex. To implement it, we do the following: for every vertex $v \in V$, we create 2 vertices v_{in} and v_{out} . v_{in} has all the incoming edges of v (including the edge from s if $v = s_i$) and v_{out} has all the outgoing edges (including the edge to t if $v \in T$). We also put a directed edge from v_{in} to v_{out} with capacity c .

To see why the max flow tells us whether a solution exists, note that any valid set of k routes gives a flow of value k : send one unit along each route; each edge (v_{in}, v_{out}) then carries the number of agents passing through v , which is at most c . Hence if the max flow is less than k , we report that no solution exists.

If the max flow is indeed k , then as every capacity is an integer, the Ford-Fulkerson algorithm will output an integral flow. For each secret agent i , we find a path from s_i to t using only edges with positive flow (e.g., by DFS). Such a path exists by flow conservation: one unit of flow enters s_i , and it must keep moving forward until it reaches t . We then subtract one unit of flow along this path, so that no two agents use the same unit of flow, and repeat for the next agent. Since the paths together use each edge (v_{in}, v_{out}) at most c times, at most c agents pass through any city.

4 Mowing the Lawn

Alice is mowing her lawn. Her lawn is an $n \times m$ grid, and each of the nm squares has one of two types of terrain: *Cut Grass* or *Tall Grass*. To mow her lawn, she has set up $n + m$ tractors, one for each row and one for each column. Each tractor mows its entire row or column and costs 1 gallon of gas to run. When a tractor mows a square, it converts Tall Grass to Cut Grass (or does nothing if the square already has Cut Grass). Alice can run the tractors in any order she chooses. Our goal is to find a set of tractors for Alice to run that uses the minimum amount of gas, such that afterward the entire lawn consists of Cut Grass.

Model this problem as a flow network. Specify the vertices, edges, capacities, source, and sink, and describe how to transform a maximum flow in your network into an optimal set of tractors for Alice to run. You do not need to explain how to compute the maximum flow itself.

Solution: The order of the tractors does not matter: the lawn is fully mowed if and only if, for every Tall Grass square (i, j) , Alice runs the tractor for row i or the tractor for column j . So we want the smallest set C of rows and columns such that every Tall Grass square lies in a row or column of C ; call such a set *valid*.

We create a vertex r_i for each row, a vertex c_j for each column, a source s , and a sink t . We add edges (s, r_i) with capacity 1, edges (c_j, t) with capacity 1, and, for each Tall Grass square (i, j) , an edge (r_i, c_j) with capacity ∞ . By the max-flow min-cut theorem, it suffices to show that the minimum cut capacity equals the size of a smallest valid set.

Given a valid set C , let $S = \{s\} \cup \{r_i : i \notin C\} \cup \{c_j : j \in C\}$. No ∞ -edge leaves S : if $r_i \in S$ and (i, j) is Tall Grass, then row $i \notin C$, so column $j \in C$ and $c_j \in S$. The only edges leaving S are (s, r_i) for rows in C and (c_j, t) for columns in C , so this cut has capacity $|C|$.

Conversely, given a cut $(S, \bar{S})$ of finite capacity, let C consist of the rows i with $r_i \notin S$ and the columns j with $c_j \in S$; the cut capacity is exactly $|C|$. For every Tall Grass square (i, j) with $r_i \in S$, the ∞ -edge (r_i, c_j) cannot cross the cut, so $c_j \in S$. Hence every Tall Grass square is covered by its row or column, and C is valid.

Therefore the max flow equals the minimum amount of gas. To recover the tractors, let S be the set of vertices reachable from s in the residual graph of a maximum flow; this is a minimum cut, and Alice runs the tractors for the rows i with $r_i \notin S$ and the columns j with $c_j \in S$.