# Problem 2:

## Understanding the problem

We have `n` heroes and `m` monsters. Each hero can kill certain monsters, but only one by default. With a potion, a hero can kill one extra monster, and there are `k` potions total. We want the maximum monsters killed.

This is essentially a **bipartite matching** with a global budget of `k` heroes that can have capacity `2` instead of `1`.

---

## Initial strategy

At first I thought of:

1. Build a bipartite match between heroes and monsters.
2. Afterwards, use potions to kill leftover monsters adjacent to heroes.

But this greedy idea failed — the optimal choice of potion use sometimes requires reshuffling the original matching.

---

## Final design

Model it as a **max flow**:

- `S → hero (1)` (base capacity).
- `S → P (k)` then `P → hero (1)` (potions hub lets at most `k` heroes get +1 capacity).
- `hero → monster (1)` if the hero can kill that monster.
- `monster → T (1)` ensures each monster is killed at most once.

Run Dinic's algorithm; the max flow = maximum kills.

---

## What went wrong

- **Greedy potion assignment** missed optimal rematching. → Fixed by putting potions directly into the flow network.

- **Indexing errors**: I double-subtracted monster indices and misaligned ranges. → Fixed with a clear mapping `monster = n + (id-1)`.
- **Duplicate edges**: Initially added `monster→T` multiple times (once per hero). → Moved into a single loop after input.
- **Graph size confusion**: Off-by-one in total node count. → Fixed by allocating up to `T`.

---

## Fixes

- Unified heroes in `[0..n-1]`, monsters in `[n..n+m-1]`, `S=n+m`, `P=S+1`, `T=P+1`.
- Only one `monster→T` edge per monster.
- Optional guard to prevent duplicate hero→monster edges.