CS 240E S25

Tutorial 6

June 20

- Ternary Search
- Midterm Review (Partially)

Enrich Content

Ternary Search

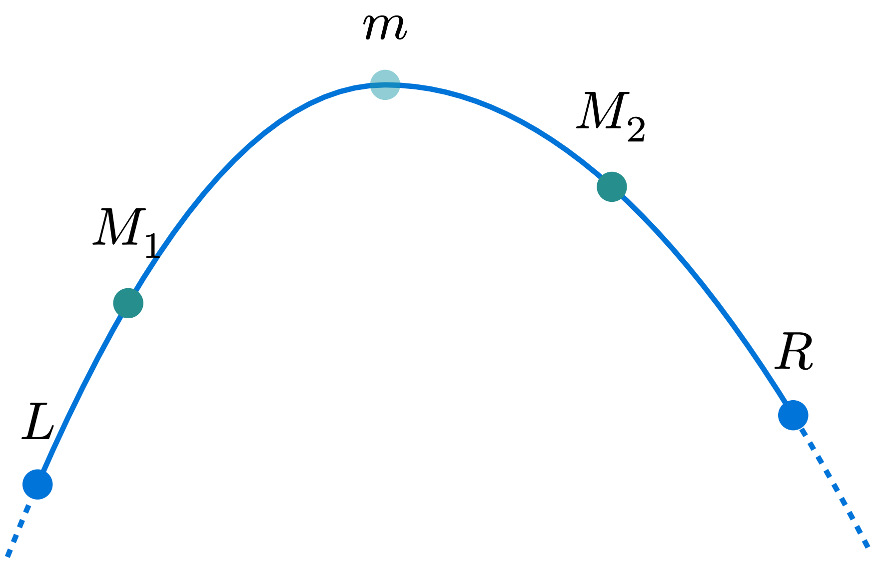
Assume you have a linear function, binary search can be used to find the target value efficiently.

However, if the function is unimodal (i.e., it has a single peak or trough), we can use **ternary search** to find the maximum or minimum value.

The formal definition of a unimodal function is: $f(x)$ is an unimodal function if there exists a point m such that for all $x \in (-\infty, m)$, $f(x)$ is increasing, and for all $x \in (m, \infty)$, $f(x)$ is decreasing, and $f(m)$ is the maximum value. (This is for the peak case, for trough case, the inequalities are reversed.)

Ternary Search: Idea

Instead of dividing the search space into two parts like binary search, we divide it into **three parts**:



We choose two points M_1, M_2 in the current search range $[L, R]$.

Assume $f(M_1) < f(M_2)$; $f(m)$ is max.

Can we conclude $m \notin [L, M_1]$? Yes!

if $m \in [L, M_1]$, then we have f decreasing in $[M_1, R]$, $f(M_1) > f(M_2)$, which is a contradiction.

But we cannot say $m \notin [M_1, R]$.

Therefore, the **new range is $[M_1, R]$.**

Ternary Search: Implementation

```
double ternarySearch(double L, double R) {
  while (R - L > eps) { // stop when the range is small enough
    doubke M1 = L + (R - L) / 3;
    doubke M2 = R - (R - L) / 3;

    if (f(M1) < f(M2)) L = M1; // move left boundary to M1
    else R = M2; // move right boundary to M2
  }
  return f((L + R) / 2); // return the maximum value
}
```

* For discrete functions, it's a bit tricky to ensure we get the max value when the algorithm stops. A simple way to address this is to stop when the range is small enough, and then evaluate all the $f(x)$ in that range to find the max value.

Ternary Search: Optimization

Each iteration only reduces the search space by either $M_1 - L$ or $R - M_2$, which is at most $\frac{1}{3}$ of the original range. Can we do better?

Notice that we don't have to choose M_1 and M_2 as equally spaced points. Instead, we can choose them as close as the midpoint $\frac{L+R}{2}$.

We can choose $M_1 = \frac{L+R}{2} - \varepsilon$ and $M_2 = \frac{L+R}{2} + \varepsilon$, where ε is a small value. Therefore, each iteraction reduces near $\frac{1}{2}$ of the search space, which is much better than $\frac{1}{3}$.

*Extension reading: If calculating $f(x)$ is expensive, We can reduce the time of evaluating $f(x)$ by choosing the golden ratio points, to utilize the property of golden ratio to reuse the previous evaluations. This is called the **golden-section search**.*