二分搜索法$(binary_search)$

定义：

- 通过不断减少解的范围可能存在得范围，从而得到问题最优解的方法。

研究角度：

- 问题的背景：
  - 什么样的问题，涉及什么样资源。
- 怎么进行关键的操作？
  - 减少解存在的范围。
  - 怎么check当前询问对象在问题下的属性，从而减少范围？

二分搜索的前提，作用描述：

求解满足 $check()$ 的最优解问题，对于任意x，$check(x)$为true。有
$x' < x, check(x) = true \quad \overline{v} \quad x > x, check(x') = true$

表示不可兼得析取。
具体情况视题意而定。这样就保证了，可以根据查询结果来减少解范围。

经典问题：笔记语料，心得来自《挑战程序设计竞赛第2版》

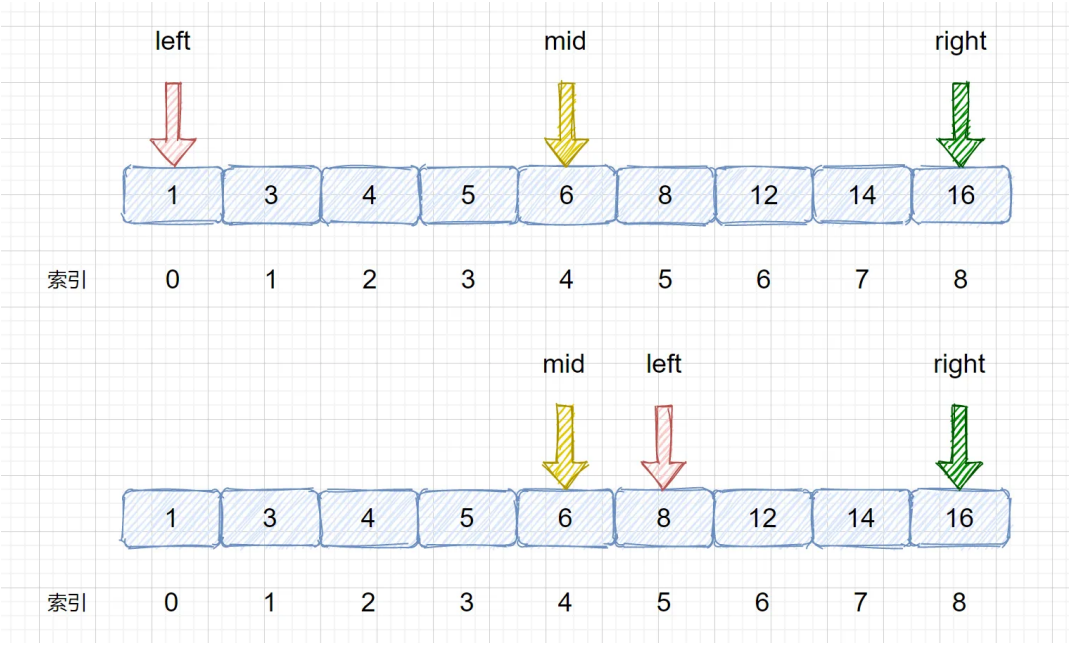
- 1.有序数组中，查找某个值。
- 2.假定一个解并且判断是否可行。
- 3.求解最大化最小化问题。（最大值，最小值，平均值等等）。

经典问题分析：

$Frst$

- **问题大意**
  - 在有序数组种查询某一个值ans的下标。
- **问题分析**
  - 我们每次查询范围的中点。为什么不先查询最大值？
    - 直接查询范围的中点，减少的范围较大，最终最坏的时间花费是$logn$。且如果有无穷多组随机合理数据测试，该方案的期望时间应该最小。

- $check(x)$定义为：x索引对应的元素小于等于ans。这样显然满足前提$check(x) = false$上部分全部排除，反之，下部分全部排除。最终我们寻找到上界即可。



- **问题代码：**

  $first$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 10;
bool check(ll x) { return x <= 234; }; //检查。
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    ll low = 1, high = 1e9, ans = -1;
    while (low <= high)
    {
        ll mid = (low + high) / 2;
        if (check(mid)) //之前的
            ans = mid, low = mid + 1;
        else
            high = mid - 1;
    }
    cout << ans << '\n';//另外定义一个ans来记录。
}
```

$second$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 10;
bool check(ll x) { return x <= 234; }; //check，最终确定答案在上方还是在下方。
int main()
{
    ios::sync_with_stdio(false);
```

```
    cin.tie(0), cout.tie(0);
    ll low = 1, high = 1e9, ans = -1;
    while (low < high)
    {
        ll mid = (low + high + 1) >> 1;//这里加一个1，可以向下取整数。最终，即使出现相邻的情况时，也可以使两个指针相遇。
        if (check(mid)) //
            low = mid;
        else
            high = mid - 1;
    }
    cout << low << '\n';//low就是true--ans.
}
```

Second

- 问题大意 -POJ N0.1064
  - 将N条绳子(长度分别为L_i)，分成k条长度相同的绳子。这k条绳子最长能有多大。答案保留小数点后两位。
- 问题分析：
  - 对于一个$x, check(x) = false$,若$x' > x$则，$check(x') = false$
  - 对于一个$x, check(x) = true$,若$x' < x$则，$check(x') = true$
  - 综上可以使用二分搜索。
- 算法设计
  - 如何假定一个解？
  - 就是取范围中点即可。初步范围应该是$0.00 - -maxl.$
  - 设计$check()$
  - 可裁剪长度为x的绳子数的总和是否大于等于k即可。

自己的代码。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e4 + 10;
double a[maxn];
int n, k;
bool check(double x)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
        sum += (int)(a[i] / x);
    return sum >= k;
}

void bin_search()
{
    double low = 0, high = 10000;
    double ans = 0;
    while (low <= high)
```

```
    {
        double mid = (low + high) / 2;
        if (check(mid))
            ans = mid, low = mid + 0.00001;
        else
            high = mid - 0.00001;
    }
    cout << fixed << setprecision(2) << ans << '\n';
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    cin >> n >> k;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    bin_search();
}
```

书本的参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int N, k;
const int MAX_N =10000;
double L[MAX_N];
bool c(double x){
    int num = 0;
    for (int i = 0; i < N;i++)
        num += (int)(L[i] / x);
    return num >= k;
}
void solve()
{
    double lb = 0, ub = 1e9;
    for (int i = 0;i<100;i++)
    {
        double mid = (lb + ub) / 2;
    if(c(mid))
        lb = mid;
    else
        ub = mid;
    }
    printf(".%.2f\n", floor(ub * 100) / 100);
}
int main(){
    solve();
}
```

生长思考：

- 遇到了新的问题。查询的域是小数，不忽略无穷的小数位。如下情况：

    - ans是一个无理数，
    - 每次check一个值之后，无法定位向下的一个空间。因为域是稠密的，不存在一个有限非0位数，使得它和check值之间有不存在数。
- 解决：
    - 对于输出小数的问题中，一般会指定允许的误差出现范围或者是指定输出中小数后出现的位数。所以我们只要追求求出精确的前几位即可。
    - $first$像上面自己的方法，check一个值之后加减一个比较小的值$1e^{-6}$。在上述问题中，最终的结果和答案，的误差小于$2e^{-6}$。这样，最终得到的ans和实际上的答案，前面的几位相同，为正确解答。
        - 缺点，可能会由于浮点数存储的精度问题，出现死循环。
    - $Second$像参考书上的方法。控制好循环的次数，100次就可以可以达到10^-30的精度范围。基本上是没有问题的。

精度的理解体会：

- 精度问题为什么会导致死循环，这个过程分析是怎么样的？
- 参见语言基础
- 分析如下
    - 当每次变化的值比较小，且原有的操作数上已经有很多为有效数字，这个变化值加上去对mid大小没有影响。即
      $mid + d = mid$
      这样就相当于low，,一直取mid值，最终肯定会相等的。所以该情况下也没有问题。第一个代码while判定也会跳出循环，没有问题。与答案也非常接近了。
    - while判定条件变为mid-low>EPS.当EPS取得非常小时。由于精度问题，连个操作数都比较大，可保存的最小一位都是整数位。，操作后的数是整数位。由于舍弃问题，当它们的最终只在最后一位有效位上相差1时，check判定使最终处理较小值，纸样导致，它们不可能相等，差值不可能为0.从此陷入死循环。
    - 小数不能取出范围比如$1e^{-10000}$最小的应该是$1e^{-16}$不然最终结果会是0，两者相减始终是0，或者正数，将进入死循环。

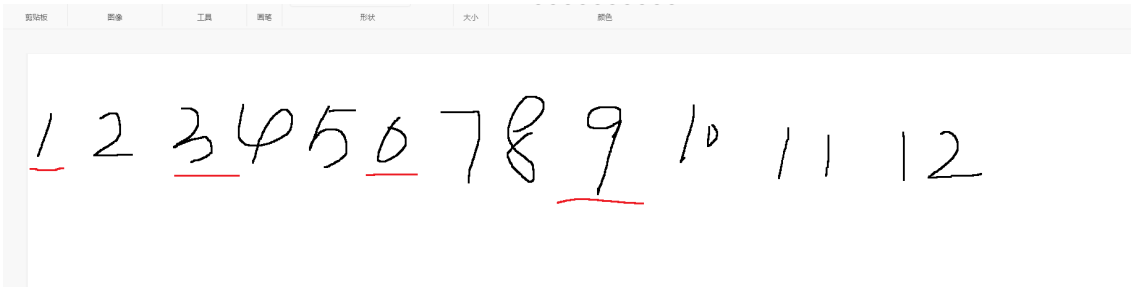
$third$

最大化最小值

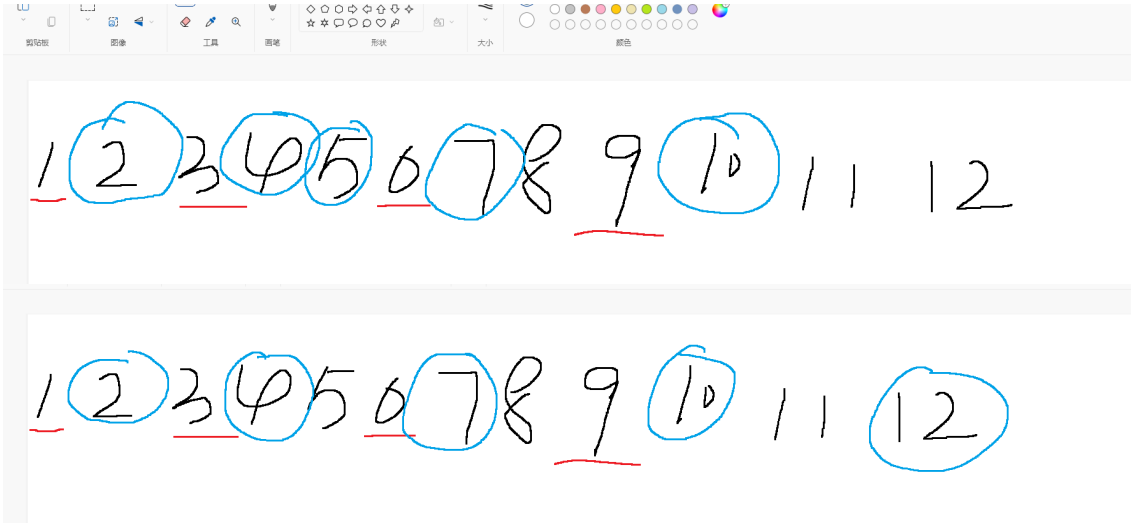
- **题目大意：一维坐标轴上，有N个牛棚，要放K头牛。每个牛棚放一头。问，两牛之间的最短距离的上界是什么？**
- **二分前提分析：**
    - 随便代一个情况进去，如果$check$为$true$。按照相同的方式选择，前面的情况也为$true$。
- **check设计：**
    - 贪心 从第一个牛棚开始，放牛，向下查询，如果和上一个已经选择的牛棚距离相差大于等于x,那么就放。如果牛可以放完，说明为true，最优解在上，像上搜。
    - 该贪心的证明：

这里只是感性的证明：

$tips-1$:



$tips-2$:多种不合法选择如下几图；



$tips-3$:正确的选法以及走向

综上。对于其它的方案，最多只能放和贪心方案一样的牛数。正常情况下，两个红点之间只能选择一个。出现交叉前放了相等的，或者更少的牛。交叉之后。1.走和之前一样的路。2.类比初始第一个点的选择继续向后选。最终结果必然会小于或等于算法方案的结果。（放了多少牛）所以只要贪心的检查就能确定，$check(x)$是否为$true$。若为$false$不可能有其他方案使当前最小距离可以实现。

代码如下：

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 10;
int M, N;
```

```cpp
int a[maxn];
bool check(int x)
{
    int last = -1e9 - 100000; //记录上一个选择的牛棚的位置
    int now = 0;              //记录当前检查牛棚的位置做
    for (int i = 0; i < M; i++)
    {
        while (now < N && a[now] - last < x)
            now++;
        if (now == N)
            return false;
        last = a[now];
        now++;
    }
    return true;
}
void solve()
{
    cin >> N >> M;
    for (int i = 0; i < N; i++)
        cin >> a[i];
    sort(a, a + N);
    int low = 0, high = 1e9;
    while (low < high)
    {
        int mid = (low + high + 1) >> 1;
        if (check(mid))
            low = mid;
        else
            high = mid - 1;
    }
    cout << low << '\n';
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    solve();
}
```

Five

- 最大化平均值问题。（转化问题，难题不可能使是第一步就告诉你要二分答案，怎么去设计check函数。典型$empty-grapth$就是典型，经过基础得推演转化，才得到一个check得角度。）
- **题目大意：**

  有n个物品得重量个价值分别是w_i和v_i从中选择k见物品使得单位体积得总价值最大。

- **问题分析**
  - 直接贪心，有限地选择k个单位价值最高的产品。想法挺好，但是过不了样例。
  - 二分答案，考虑前提是否满足。

下面判断前提是否满足。

  - 情况一：对于一个查询值x'。通过它，推测后面可能有最优解解（或者就是这个查询值）。即
    对于某一组选择。
    $\frac{\sum_{i\in S} v_i}{\sum_{i\in S} w_i} >= x' \quad can-founded$
    - 变化得到如下不等式：
      $\sum_{i\in S} v_i - x \times \sum_{i\in S} w_i >= 0$
      该式子成立。对于同一组这样得选择，当x变小，$check(x) = true$
      x显然成立。这样我们可以筛掉前面得范围。
  - 情况二，通过一个查询值，推测它后面没有可行值，即
    对任意一组选择。
    $\frac{\sum_{i\in S} v_i}{\sum_{i\in S} w_i} >= x' \quad not-founded$
    也就是对于任何一组选择有:
    $\frac{\sum_{i\in S} v_i}{\sum_{i\in S} w_i} < x' \quad not-founded$
    - 变化得到如下不等式：
      $\sum_{i\in S} v_i - x \times \sum_{i\in S} w_i < 0$
      上式成立，这样当x变大，对于任何一组选择，不等式左边得指标值都会变小。此时，对于所有比x大的x''，$chek（x''）= false$.可以减少范围。

综上，二分搜索可行。

---

check函数设计；因为要找一组选择满足下不等式：
$\sum_{i\in S} v_i - x \times \sum_{i\in S} w_i >= 0$
找到最大的一组判定才可以判定$check(x) = false.$
直接寻找最该指标最大的一组即可。由于每一个物品都有一个相关的明确的贡献值，直接整合成一个数组，将它们降序排序。优先选择前k个，就是指标最大的组合。

```
 #include <bits/stdc++.h>
using namespace std;
const int maxn = 1e4 + 10;
int w[maxn], v[maxn];
int n, k;
double y[maxn]; //存放每个元素各种情下的贡献。
bool check(double x)
{
    for (int i = 1; i <= n; i++)
        y[i] = v[i] - x * w[i];
    double sum = 0;
    sort(y + 1, y + 1 + n);
    for (int i = 1; i <= k; i++)
        sum += y[n - i + 1];
    return sum >= 0;
```

```
}
void solve()
{
    cin >> n >> k;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
    double low = 0, high = 1e6;
    for (int i = 0; i < 100; i++)
    {
        double mid = (low + high) / 2;
        if (check(mid))
            low = mid;
        else
            high = mid;
    }
    cout << fixed << setprecision(2) << low << '\n';
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    solve();
}
```

最后的总结。

- 问题背景：掌握已有的资源，可以对问题有非常多的解决方案。寻找可行方案的最优边界。
- 问题前提，查询范围，利用查询结果，可以确定查询值上方，或者查询值下方的check()情况。这样每次chek之后就可以大幅度的减少范围。