目录

[排序 1](#_Toc47470411)

[字符串搜索 5](#_Toc47470412)

[树和图 6](#_Toc47470413)

[数值优化 9](#_Toc47470414)

# 排序

**快速排序**：

1. 将数组划分为left子数组 <= pivot <= right子数组
2. 对pivot左边的序列递归快速排序
3. 对pivot右边的序列递归快速排序

void q\_sort(int v[], int left, int right) {

if (left >= right) return;

int idx\_pivot = partition(v, left, right);

q\_sort(v, left, idx\_pivot - 1);

q\_sort(v, idx\_pivot + 1, right);

}

如何划分数组，使left子数组 <= pivot <= right子数组

设最左边值为pivot，

j从右到左搜索，找小于pivot的第一个值

i从左到右搜索，找大于pivot的第一个值

交换以上两个值，继续剩余搜索，直至i遇到j

此时i就是划分点，与pivot交换

int partition(int v[], int left, int right) {

int pivot = v[left];

int i = left;

int j = right;

while (i != j) {

while (v[j] >= pivot && i < j) j--;

while (v[i] <= pivot && i < j) i++;

if (i < j) swap(v[i], v[j]);

}

swap(v[left], v[i]);

return i;

}

**堆排序**

基本思想：

1. 将待排序数组构成最大堆

for (int i = n/2-1; i >= 0; --i)

max\_heap(v, i, n);

1. 将顶端数与末尾数交换，则末尾为最大值

swap(v[0], v[i]);

1. 对剩余n-1个数再建最大堆，重复2,3直至i=1

max\_heap(v, 0, i);

void heap\_sort(int v[], int n) {

// build max heap

for (int i = n/2-1; i >= 0; --i) max\_heap(v, i, n);

for (int i = n-1; i >= 1; i--) {

swap(v[0], v[i]);

max\_heap(v, 0, i);

}

}

**如何建最大堆（堆排序）？**

1. 对当前节点i, 若子节点超出边界，结束；
2. 求最大左右子节点，若大于当前节点，则互换交换；
3. 对子树建堆

void max\_heap(int v[], int curIdx, int n) {

int leftChild = 2 \* curIdx + 1;

if (leftChild >= n) return;

// max of left child and right child

if (leftChild + 1 < n && v[leftChild] < v[leftChild + 1]) ++leftChild;

// exchange parent node and max child node

if (v[curIdx] < v[leftChild]) {

swap(v[curIdx], v[leftChild]);

// update the child tree

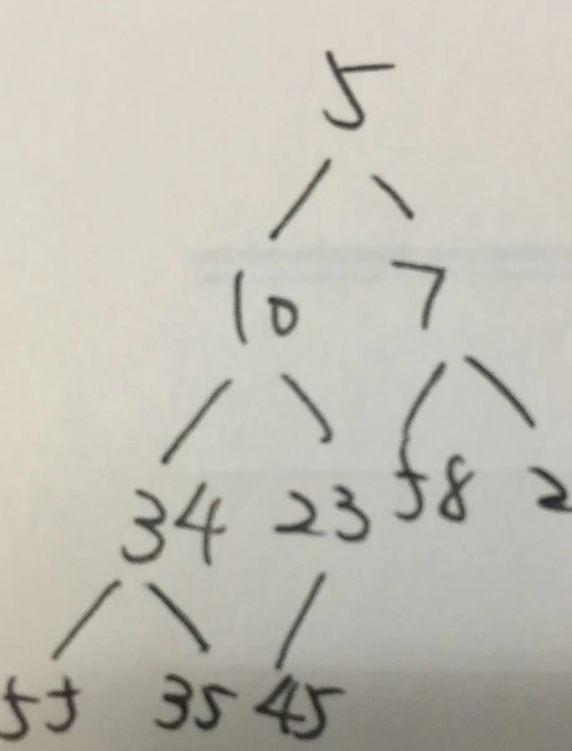
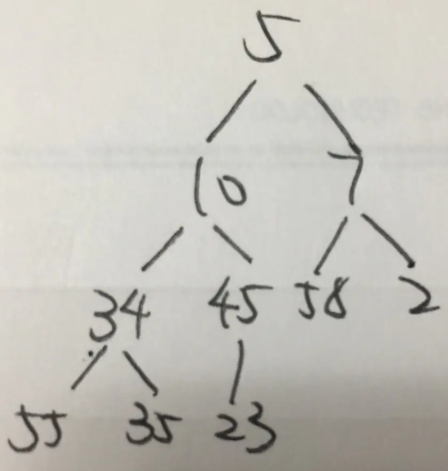
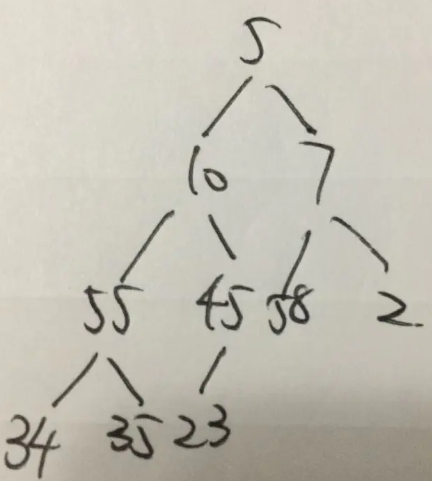
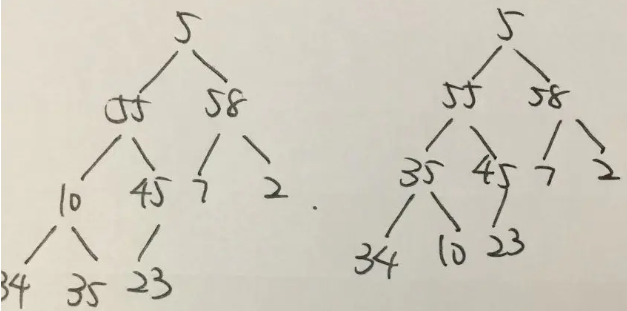
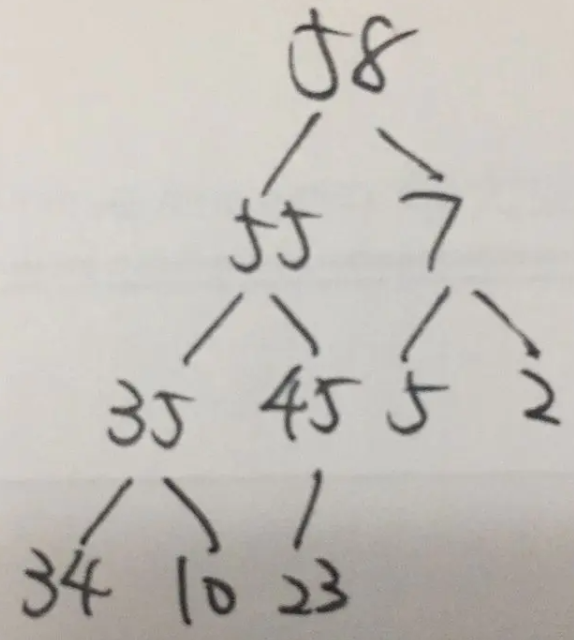
max\_heap(v, leftChild, n);

}

}

第i个序号的结点的数，那么他的父结点序号就是(i-1)/2，它的孩子结点就为2i+1与2i+2

int a[10] = {5,10,7,34,23,58,2,55,35,45}

maxheap(v, n/2-1, n) ->maxheap(v, 3, n)->maxheap(v, 2, n)-> maxheap(v, 1, n)->maxheap(v, 0,n)

**bubble sort**

对v[0 : n]依次比较相邻两个元素，若后者比前者小就交换，一轮下来最小值冒出来

对v[1 : n]重复以上过程

…

void bubble\_sort(int v[], int n) {

for (int i = 0; i < n ; ++i) {

for (int j = n-1; j >= i; --j) {

if (v[j] < v[j - 1]) {

swap(v[j], v[j + 1]);

}

}

}

}

**Selection sort**

对v[0 : n]选择最小元素放入第一个 (采用交换形式)

对v[1 : n]选择最小元素放入第二个

…

void selection\_sort(int v[], int n) {

for (int i = 0; i < n; ++i) {

求v[i : n]最小值v[min\_idx];

swap(v[i], v[min\_idx]);

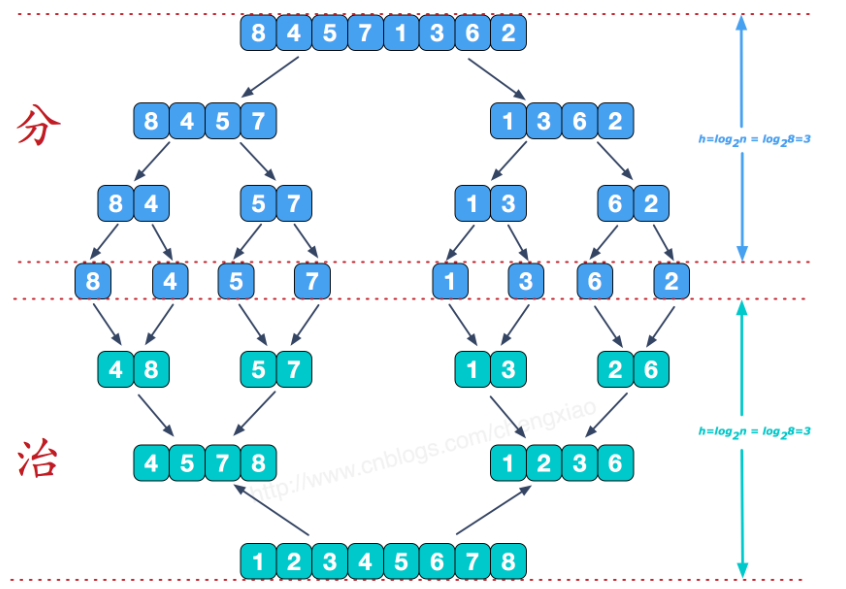
}

}

**插入排序**

**同打排一样，拿到新排插到有序的手排**

**归并排序：分而治之**



分很容易，关键是如何合并？

每次从两段中取一个数据，较小者放入temp数组

copy两段中剩余的数据于temp数组

void merge\_sort(int v[], int left, int right) {

if (left >= right) return;

int mid = left + (right - left) / 2;

merge\_sort(v, left, mid);

merge\_sort(v, mid + 1, right);

merge(v, left, mid, right);

}

void merge(int v[], int left, int mid, int right) {

int\* temp = (int\*) new int[right - left + 1];

int idx1 = left;

int idx2 = mid+1;

int idx3 = 0;

//每次从两段中取一个数据，较小者放入temp数组

while (idx1 <= mid && idx2 <= right) {

temp[idx3++] = v[idx1] < v[idx2] ? v[idx1++] : v[idx2++];

}

// copy剩余的数据

while (idx1 <= mid) temp[idx3++] = v[idx1++];

while (idx2 <= right) temp[idx3++] = v[idx2++];

for (int i = 0; i < idx3; ++i) v[left + i] = temp[i];

delete[] temp;

}

**计数排序：**

当输入的元素是整数，且序列比较集中时，用计数排序时间复杂度是O(n+k)

输入值转化为键，出现次数转化为值

void buck\_sort(int v[], int n) {

int min\_value = v[get\_min\_idx(v, n)];

int max\_value = v[get\_max\_idx(v, n)];

int bucklen = max\_value - min\_value + 1;

int\* temp = new int[bucklen];

memset(temp, 0, bucklen\*sizeof(int));

for (int i = 0; i < n; ++i) temp[v[i]-min\_value] ++;

int k = 0;

for (int i = 0; i < bucklen; ++i) {

int num = temp[i];

int value = i+min\_value;

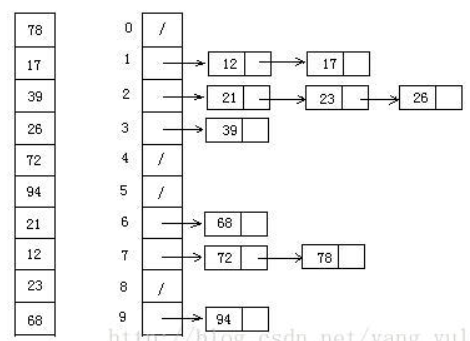
for (int j = 0; j < num; ++j) v[k++] = value;

}

}

**桶排序：**

计数排序的改进版，将数据映射到桶，各个桶有序，只要对桶分别排序。关键在于映射函数,常用哈希映射hash=e\*len/(max+1)，复杂度介于O(n)~O(nlgn)



**求top K ?**

1. 维护一个k大小的有序序列，
2. 遍历剩余数据，依次和有序序列中的最小值比较，若大则将最小值换出; 重新对k序列排序
3. 直至遍历完所有数据。

# 字符串搜索

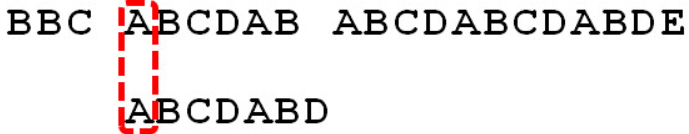
如原字符串：”BBC ABCDAB ABCDABCDABDE”

子串： ”ABCDABD”

暴力搜索：同图像处理一样，滑窗搜索

**KMP算法**：同暴力搜索一样，只是用上部分匹配的信息，部分匹配用于决定步伐

Stride = 已匹配的字符数 – 对应的部分匹配值的长度



stride = len(ABCDAB) – len(part\_match) = 6 – 2

见图b

如何计算部分匹配值？(根据子串的前缀和后缀的intersection计算出来)

“AB” 前缀：[A], 后缀为[B]，则共有元素长度为0

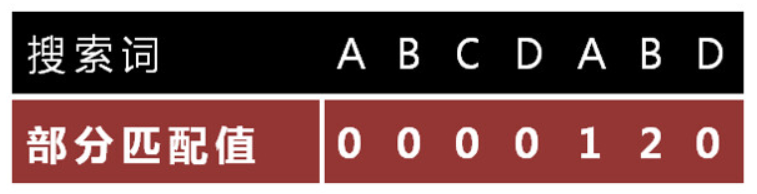
“ABC”前缀：[A, AB], 后缀为[BC, C], 则共有元素长度为0

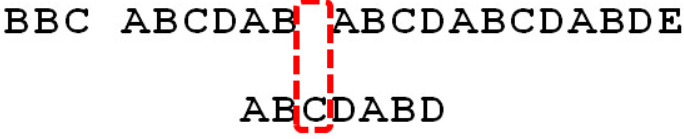
“ABCD”前缀：[A, AB, ABC], 后缀为[BCD, CD, D], 则共有元素长度为0

“ABCDA”前缀：[A, AB, ABC, ABCD], 后缀为[BCDA, CDA, DA, A], 则共有元素长度为1

…

得部分匹配表：





stride = len(AB) – len(part\_match) = 2 – 0

Repeat above process

# 树和图

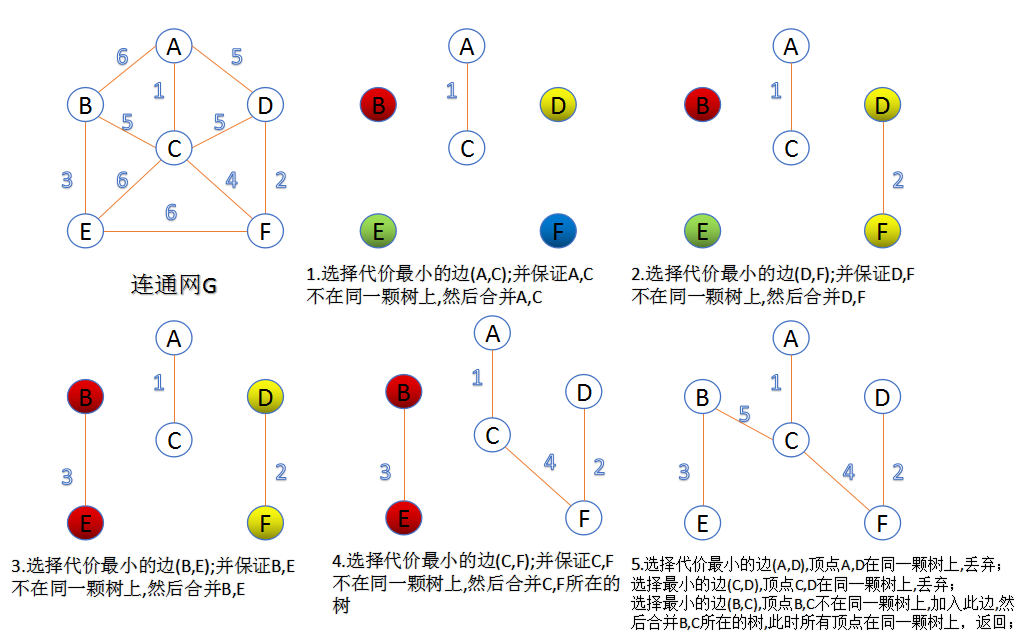
最小生成树

1. Kruskal算法 （加边法）

1. 把图中的所有边按代价从小到大排序；n个顶点看成n棵树；

2. 按权值从小到大选择边，所选边顶点应属于不同的树，加入到最小生成树

3. 重复以上,直到所有顶点都在树内

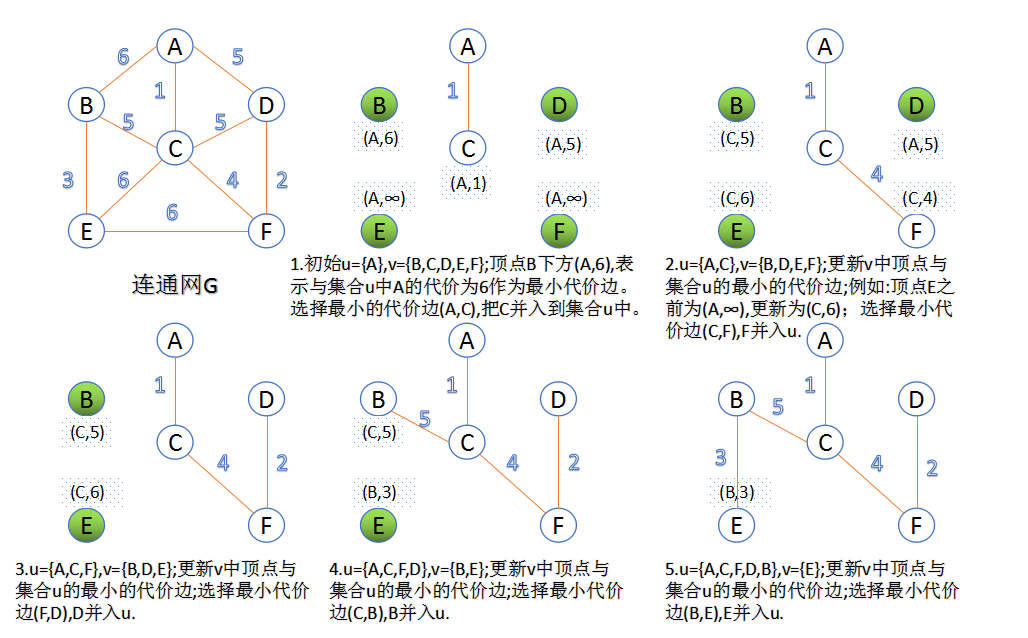


2.Prim算法 （加点法）

1）以某点开始作为初始点集合

2）求点集合相连的最小边，最小边中的新点加入点集合

3）重复以上过程，直到点集合含n个顶点



**KD-Tree**

优缺点: Kd树在维度较小时（比如20、30），算法的查找效率很高，然而当数据维度增大（例如：K≥100），查找效率会随着维度的增加而迅速下降。假设数据集的维数为D，一般来说要求数据的规模N满足N>>2的D次方，才能达到高效的搜索。

1. Kd-Tree的构建

1. 在K维数据集合中选择具有最大方差的维度k， (Note: 数据方差大说明沿该坐标轴方向上数据点分散的比较开)

然后在该维度上选择中值m为pivot对该数据集合进行划分，得到两个子集合 (Note:

确保建立的树尽量地平衡)

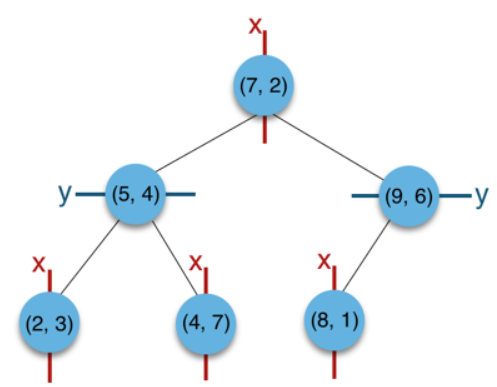
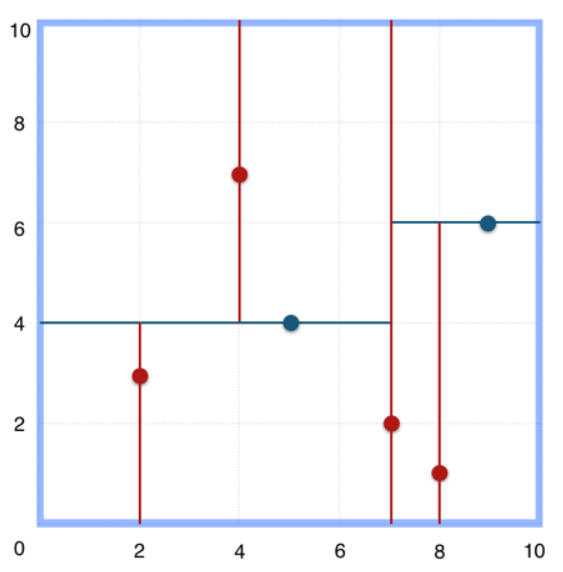
2. 对两个子集合重复（1）步骤的过程，直至所有子集合都不能再划分为止；

假定二维平面上，点的集合{(2,3)，(5,4)，(9,6)，(4,7)，(8,1)，(7,2)}

寻找根节点：计算数据在x和y轴的各自方差, 求出x轴方差大，作为切分维度x; 将数据在x轴排序，得中值(7, 2)作为根节点

划分为左子集{(2,3), (4,7), (5,4)}, 右子集{(8,1), (9,6)}

对左子集，右子集重复以上过程

1. Kd-Tree的最近邻查找
2. 将查询点Q从根结点开始，按切分维比较，获取从根结点->叶子结点的路径
3. 进行回溯操作，该操作是为了找到离Q更近的“最近邻点”

例1：查找点Q(2.1,3.1)

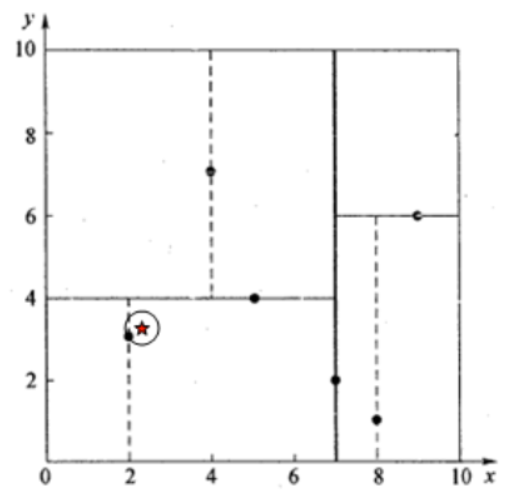
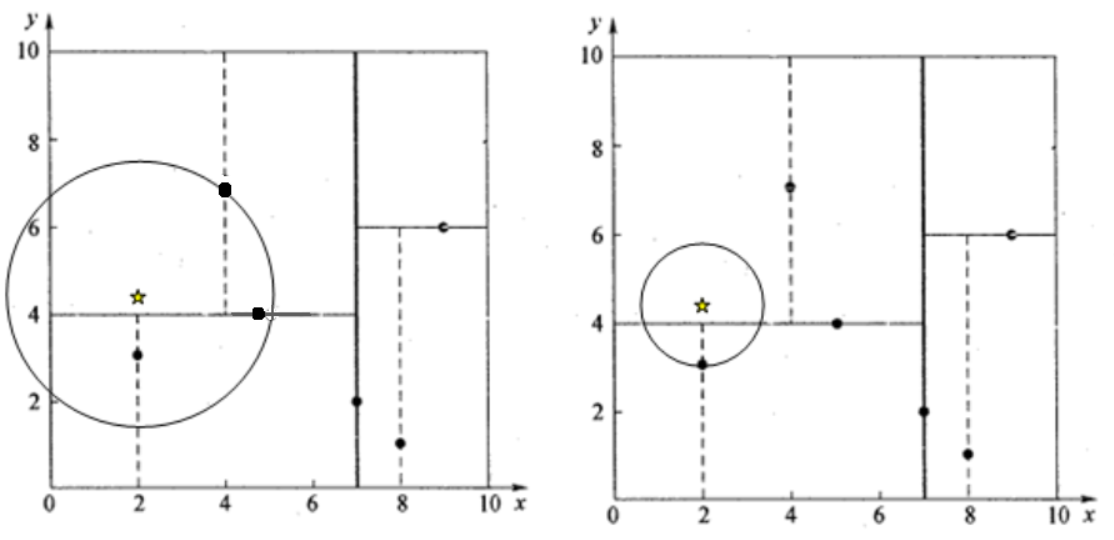
1. 获取搜索路径(7,2, x) -> (5,4, y) -> (2,3, x)
2. '回溯'操作

检查点(2,3):以(2,3)作为当前最近邻点，计算到查询点距离作为当前最近距离0.1414

检查点(5,4): 以查询点为圆心，以当前最近距离为半径画圆，如图所示,发现该圆并不和划分轴y = 4相交，即|Q[y] - 4|=|3.1 - 4|=0.9 > 0.1414，因此不进入该检查点右子空间中去搜索。

同理回溯检查点(7,2)

得最近点(2,3), 最近距离0.1414

例2：查找点Q(2,4.5)

1. **获取搜索路径**(7,2, x, 根) -> (5,4, y, 根) -> (4,7, x, 叶)
2. **'回溯'操作**

检查点(4,7,x,叶)：直接以叶节点(4,7)作为当前最近邻点，计算到查询点距离作为当前最近距离3.202

检查点(5,4,y,根): 以查询点为圆心，以当前最近距离为半径画圆，如图所示,发现该圆和划分轴y = 4相交，即|Q[y] - 4|=|4.5 - 4|=0.5 < 3.202，因此需要进入该检查点另一子空间中去搜索,

更新搜索路径为(7,2, x,根) -> (2,3, x,叶), 更新当前最近邻点(5,4), 当前最近距离3.04

检查点(2,3,x,叶): 计算查询点与叶节点距离，更新当前最近邻点(2,3),当前最近距离1.5

同理回溯剩余检查点得最近点(2,3), 最近距离1.5

Sklearn 代码：

from sklearn.neighbors import KDTree

points = np.random.random((100, 2))

tree = KDTree(points)

query = points[0]

# kNN

dists, indices = tree.query([query], k=3)

# query radius

indices = tree.query\_radius([point], r=0.2)

# 数值优化

牛顿法

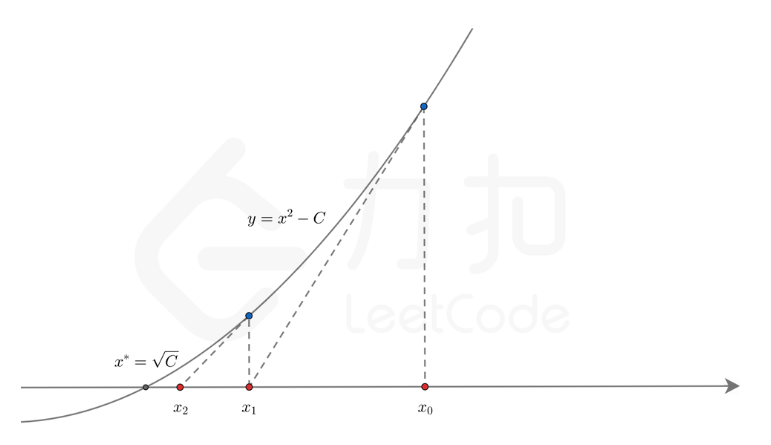
基本思想： 求 等效于用切线近似曲线, 逐步逼近

步骤：

1. 假定初始值

求平方根？ （牛顿法O(lgn)）

即 =0，牛顿法



Style transfer

Stylized image = image – its texture + style of painting

AdaIN style transfer, Kaggle’s Painter by Numbers dataset

<https://github.com/rgeirhos/Stylized-ImageNet>