电类工程导论（C 类）大作业报告

课程代码 EE208

组号：11 组

姓名 郭嘉宋 学院 电院 学号 517030910374 班级 F1703015

姓名 姜泽坤 学院 电院 学号 517030910375 班级 F1703015

姓名 马嘉琪 学院 电院 学号 517030910379 班级 F1703015

姓名 刘若峰 学院 电院 学号 517030910377 班级 F1703015

组内分工：

所有前端 & 前后端连接、最后整合 — 郭嘉宋；

爬虫部分 — 刘若峰；

文字数据处理与整合 — 马嘉琪；

图像识别 — 姜泽坤；

目录

好大学在线搜索（*Online Search For University*）

郭嘉宋[1]，姜泽坤[2]，马嘉琪[3]，刘若峰[4]

（[1]上海交通大学 电子信息与电气工程学院，上海 200240，中国；[2]上海交通大学 电子信息与电气工程学院，上海 200240，中国；[3]上海交通大学 电子信息与电气工程学院，上海 200240，中国；[4]上海交通大学 电子信息与电气工程学院，上海 200240，中国）

摘要：在中国，有大量的学生需要在网上寻找关于大学的信息；还有许多人对生活中看到的未知校徽感兴趣。为了实现对中国大学信息的整合收集，以及提供对中国大学信息搜索服务，实现同学们对未知校徽进行搜索的功能，我们利用课程所学习过的相关知识设计了好大学在线搜索引擎，满足了大家对大学信息的搜索、以及对校徽图片的识别。

关键词：好大学在线搜索；爬虫；信息整合；图像搜索；前端设计

Online Search For University

Jiasong Guo[1]，Zekun Jiang[2]，Jiaqi Ma[3], Ruofeng Liu[4]

（[1]**Shanghai Jiao Tong University School of Electronic Information And Electronic Engineering，Shanghai 200240，China；** [2] **Shanghai Jiao Tong University School of Electronic Information And Electronic Engineering，Shanghai 200240，China；** [3] **Shanghai Jiao Tong University School of Electronic Information And Electronic Engineering，Shanghai 200240，China；** [4] **Shanghai Jiao Tong University School of Electronic Information And Electronic Engineering，Shanghai 200240，China**）

Abstract: In China, a large number of students need to search for information about Universities on the Internet, and many people are interested in the unknown school emblem they see in their lives. In order to realize the integration and collection of university information in China, as well as provide information search service for Chinese universities, and realize the function of searching unknown emblems for students, we have designed a good university online search engine based on the knowledge we have learned in the course, which meets the needs of searching university information and recognizing University Emblem pictures.

Key words: good university online search; crawler; information integration; image search; front-end design

一. 爬虫部分

1. 文字爬取部分

1.1：为实现对中国大学信息的收集，我们要运用到爬虫技术。首先我们寻找到一个与大学信息有关的网站。然后以这个网站为起点开始我们的爬取。

1.2：HTTP 协议—GET 方法。为了请求网页，我们调用 urllib2 的库，模拟 GET 方法来让我们能够收到返回的网页。

1.3：页面剖析—BeautifulSoup。在得到网页后，我们要对其进行页面的 HTML 元素剖析，提取我们想要的内容。其中最实用的方法是使用 BeautifulSoup 库对页面进行剖析.

1.4：多线程并发抓取。为了加快我们的爬取速度，我们使用多线程并发抓取的方法。同时这种方法的搜索规则类似于广度优先搜索（BFS），满足我们想要爬取的策略。

1.5:保存。最后我们将每个爬过的网页的 url 保存在一个 txt 文件中，将爬过的网页内容保存在文件夹 html 中。

2. 图片爬取部分

2.1：静态网页图片爬取。对于无变化的静态网页内容，我们只需要在页面剖析后，寻找 img 部分即可，然后提取保存。

2.2：翻页网页图片爬取。对于存在翻页情况的网页，分析其 url 即可知，每次翻页变化的是 url 中某些部分的数字。我们在每次循环中变化这些数字即可，然后再进行页面剖析，寻找 img 部分。

2.3：动态网页图片爬取。对于百度图片等含有 javascript 部分的动态网页，我们要分析其 Json 数据，然后找寻规律，发现图片的 url 包裹在哪一部分，然后提取。

2.4：将大学校徽图片与大学名匹配，保存在两个文件中，根据序号一一对应。

二． 文字数据处理与整合

1.给所有网页的 URL 制作索引

```
    global cont
        path = os.path.join(root, "html/index.txt")
        file = open(path)
        line = file.readline()
        while line:
            row = line.split("\t")
            while len(row)!=2:
                line = file.readline()
                if len(line) > 0:
                    row.append(line)
            row[0] = row[0].strip()
            row[1] = row[1].strip()
            menu[row[1]]=row[0]
            line = file.readline()
        file.close()

        for root, dirnames, filenames in os.walk(os.path.join(root, "html")):
            for filename in filenames:
                if not (filename.endswith('.html') or filename.endswith('.cn') or filename.endswith('.com')):
                    continue
                print "adding", filename
                try:
                    path = os.path.join(root, filename)
                    file = open(path)
                    contents = file.read()
                    soup = BeautifulSoup(contents)
                    title = soup.title.string
            print title
                    file.close()
            doc = Document()
            doc.add(Field("Contents", contents, t2))
                    doc.add(Field("Title", title, t1))
                    if filename in menu.keys():
                        doc.add(Field("URL", menu[filename], t1))
                    writer.addDocument(doc)
                except Exception, e:
                    print "Failed in indexDocs:", e
```

具体与之前的 lab 做法相同，将 URL 与文件名先进行预处理的匹配，之后遍历所有的网页文件，读入到索引之中。

2.给所有的学校名制作索引

同 1

3.制作爬取学校基本介绍的爬虫

研究网页的架构，发现基本介绍都被写在 Class=lemma-summary 的 div 中，但问题在于，这些介绍之中穿插着各种带有超链接的词条，于是，我们的首要任务就是将这些词条的超链接去掉，即去 tag 操作

```
intro = soup.find('div',{'class': 'lemma-summary'}).get_text(strip=True)
```

利用 beautifulsoup 中的 get_text 函数，即可得到我们需要的内容

同时，在爬取网页的同时，需要注意，由于百度百科具有一定的防爬虫机制（具体体现于爬虫工作一段时间后会莫名停止工作，分析在于服务器拒绝了爬虫的访问），于是，我们需要给爬虫增加一个伪装的过程，即添加一些 header 说明来方便爬虫自由访问网站

```
headersParameters = {
        'Connection': 'Keep-Alive',
        'Accept': 'text/html, application/xhtml+xml, */*',
        'Accept-Language': 'en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3',
        'Accept-Encoding': 'gzip, deflate',
        'User-Agent': 'Mozilla/6.1 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko'
    }
```

上述说明，使得将爬虫伪装成浏览器再进行服务器的访问，同时 Connection 的说明维护了爬虫与服务器的连接，防止访问过程出现中断。

4.搜索索引，得到学校名的相关信息

4.1：首先根据关键词得到有关学校。

```
def run(searcher, analyzer):
    while True:
        print
        print "Hit enter with no input to quit."
        command = raw_input("Query:")
        if command == '':
            return

        print
        print "Searching for:", command
    boolquery = BooleanQuery.Builder()
        query = QueryParser("name", analyzer).parse(command)
        boolquery.add(query, BooleanClause.Occur.MUST)
        scoreDocs = searcher.search(boolquery.build(), 50).scoreDocs
        print "%s total matching documents." % len(scoreDocs)

    S = list()
        for scoreDoc in scoreDocs:
            doc = searcher.doc(scoreDoc.doc)
        if doc.get("School") in S:
        pass
        else:
                print doc.get("School")+'\t'+doc.get("path")
        S.append(doc.get("School"))
```

4.2：然后根据学校名可以得到相关网页

```
def run(searcher, analyzer):
    res = dict()
    Query = open("logo.txt","r")
    out = open("web.txt", "w")
    command = Query.readline().strip()
    while command:
        print "Searching for:", command
    boolquery = BooleanQuery.Builder()
        query = QueryParser("Contents", analyzer).parse(command)
        boolquery.add(query, BooleanClause.Occur.MUST)
        scoreDocs = searcher.search(boolquery.build(), 3).scoreDocs

        for scoreDoc in scoreDocs:
            doc = searcher.doc(scoreDoc.doc)
        if not command in res.keys():
        res[command]=list()
            res[command].append(doc.get('URL')+'$'+doc.get('Title'))

        #print doc.get('Title')
    out.write(command+'\t'+"\t".join(res[command])+'\n')
    command = Query.readline().strip()
    Query.close()
    out.close()
```

上述得到的所有信息，以字段的形式存于数据库之中，方便前端进行数据的提取与操作

（在本次作业中，由于数据规模太小，并不是特别需要数据库的介入，于是在这里用本地 txt 文件代替数据库）

5.编码问题

无论在爬取的过程中还是数据整合中，由于 python2.7 的编码体系并不是那么完善，我们需要在操作过程中不断注意自己字符串的编码类型，以防止数据交流过程中出现的由于编码问题而出现的各种错误。

Python2.7 默认的编码类型为 Unicode，然而，由于中文字符的存在，我们需要将环境的默认编码先调成 utf-8

```
reload(sys)
sys.setdefaultencoding('utf8')
```

这一步操作，不仅避免了程序本身的编码问题，又解决了在调用 BeautifulSoup 时出现的编码不统一现象（BeautifulSoup 内部默认也是采用 Unicode 编码）

三． 图像识别

1.实验原理

1.1：orb

1.1.1 综述

ORB 特征是对 FAST 特征点与 BREIF 特征描述子的一种结合与改进，这个算法是由 Ethan Rublee,Vincent Rabaud,Kurt Konolige 以及 Gary R.Bradski 在 2011 年一篇名为“ORB：An Efficient Alternative to SIFT or SURF”的文章中提出。首先，它利用 FAST 特征点检测的方法来检测特征点，Harris 角点的响应函数定义为

$R=detM-\alpha(traceM)2$R=detM−α(traceM)2

1.1.2 旋转不变性

我们知道 FAST 特征点是没有尺度不变性的，所以我们可以通过构建高斯金字塔，然后在每一层金字塔图像上检测角点，来实现尺度不变性。那么，对于局部不变性，我们还差一个问题没有解决，就是 FAST 特征点不具有方向，ORB 的论文中提出了一种利用灰度质心法来解决这个问题，灰度质心法假设角点的灰度与质心之间存在一个偏移，这个向量可以用于表示一个方向。对于任意一个特征点 p 来说，我们定义 p 的邻域像素的矩为：

$$mpq=\sum x^{p}y^{q}*I(x,y)$$

其中 $I(x,y)$为点(x,y)处的灰度值。那么我们可以得到图像的质心为：

$$C=(m10/m00,m01/m00)$$

那么特征点与质心的夹角定义为 FAST 特征点的方向：

$$\theta=arctan(m01,m10)$$

为了提高方法的旋转不变性，需要确保 x 和 y 在半径为 r 的圆形区域内，即 $x,y \in [-r,r]$，r 等于邻域半径。

1.1.3 特征点的描述

ORB 选择了 BRIEF 作为特征描述方法，但是我们知道 BRIEF 是没有旋转不变性的，所以我们需要给 BRIEF 加上旋转不变性，把这种方法称为“Steer BREIF”。对于任何一个特征点来说，它的 BRIEF 描述子是一个长度为 n 的二值码串，这个二值串是由特征点周围 n 个点对（2n 个点）生成的，现在我们将这 2n 个点(xi,yi), $i=1,2,\cdots,2n$(xi,yi), i=1,2,…,2n 组成一个矩阵 S

$S=(x1\ y1\ x2\ y2\cdots x2n\ y2n)$（y 在第二行）

Calonder 建议为每个块的旋转和投影集合分别计算 BRIEF 描述子，但代价昂贵。ORB 中采用了一个更有效的方法：使用邻域方向 θ 和对应的旋转矩阵 $R\theta$，构建 S 的一个校正版本 Sθ　Sθ=RθS，其中

$$R\theta=[\cos\theta\ -\sin\theta\sin\theta\cos\theta]$$

而 θ 即我们在 1.2 中为特征点求得的主方向。

实际上，我们可以把角度离散化，即把 360 度分为 12 份，每一份是 30 度，然后我们对这个 12 个角度分别求得一个 $S\theta$，这样我们就创建了一个查找表，对于每一个 $\theta\theta$，我们只需查表即可快速得到它的点对的集合 $S\theta$。

1.1.4 解决描述子的区分性

BRIEF 令人惊喜的特性之一是：对于 n 维的二值串的每个比特征位，所有特征点在该位上的值都满足一个均值接近于 0.5，而方差很大的高斯分布。方差越大，说明区分性越强，那么不同特征点的描述子就表现出来越大差异性，对匹配来说不容易误配。但是当我们把 BRIEF 沿着特征点的方向调整为 Steered BRIEF 时，均值就漂移到一个更加分散式的模式。可以理解为有方向性的角点关键点对二值串则展现了一个更加均衡的表现。而且论文中提到经过 PCA 对各个特征向量进行分析，得知 Steered BRIEF 的方差很小，判别性小，各个成分之间相关性较大。

为了减少 Steered BRIEF 方差的亏损，并减少二进制码串之间的相关性，ORB 使用了一种学习的方法来选择一个较小的点对集合。方法如下：

首先建立一个大约 300k 关键点的测试集，这些关键点来自于 PASCAL2006 集中的图像。

对于这 300k 个关键点中的每一个特征点，考虑它的 31×31 的邻域，我们将在这个邻域内找一些点对。不同于 BRIEF 中要先对这个 Patch 内的点做平滑，再用以 Patch 中心为原点的高斯分布选择点对的方法。ORB 为了去除某些噪声点的干扰，选择了一个 5×5 大小的区域的平均灰度来代替原来一个单点的灰度，这里 5×5 区域内图像平均灰度的计算可以用积分图的方法。我们知道 31×31 的 Patch 里共有 N=(31−5+1)×(31−5+1) 个这种子窗口，那么我们要 MN 个子窗口中选择 2 个子窗口的话，共有 CN2 种方法。所以，对于 300k 中的每一个特征点，我们都可以从它的 31×31 大小的邻域中提取出一个很长的二进制串，长度为 $M=C2NM=CN2$，表示为

$$binArray=[p1,p2,\cdots,pM],pi\in\{0,1\}$$

那么当 300k 个关键点全部进行上面的提取之后，我们就得到了一个 $300k\times M$ 的矩阵，矩阵中的每个元素值为 0 或 1。

对该矩阵的每个列向量，也就是每个点对在 300k 个特征点上的测试结果，计算其均值。把所有的列向量按均值进行重新排序。排好后，组成了一个向量 T，T 的每一个元素都是一个列向量。

进行贪婪搜索：从 T 中把排在第一的那个列放到 R 中，T 中就没有这个点对了测试结果了。然后把 T 中的排下一个的列与 R 中的所有元素比较，计算它们的相关性，旭果相关超过了某一事先设定好的阈值，就扔了它，否则就把它放到 R 里面。重复上面的步骤，只

RR 中有 256 个列向量为止。如果把 T全找完也，也没有找到 256 个，那么，可以把相关的阈值调高一些，再重试一遍。

这样，我们就得到了 256 个点对。上面这个过程我们称它为 rBRIEF。

1.1.5 python opencv 使用

ORB 中有很多参数可以设置，在 OpenCV 中它可以通过 ORB 来创建一个 ORB 检测器。

```
ORB::ORB(int nfeatures=500, float scaleFactor=1.2f, int nlevels=8, int
edgeThreshold=31, int firstLevel=0, int WTA_K=2, int
scoreType=ORB::HARRIS_SCORE, int patchSize=31)
```

下面介绍一下各个参数的含义：

nfeatures - 最多提取的特征点的数量；

scaleFactor - 金字塔图像之间的尺度参数，类似于 SIFT 中的 kk；

nlevels - 高斯金字塔的层数；

edgeThreshold - 边缘阈值，这个值主要是根据后面的 patchSize 来定的，靠近边缘 edgeThreshold 以内的像素是不检测特征点的。

firstLevel - 看过 SIFT 都知道，我们可以指定第一层的索引值，这里默认为 0。

WET_K - 用于产生 BIREF 描述子的 点对的个数，一般为 2 个，也可以设置为 3 个或 4 个，那么这时候描述子之间的距离计算就不能用汉明距离了，而是应该用一个变种。OpenCV 中，如果设置 WET_K = 2，则选用点对就只有 2 个点，匹配的时候距离参数选择 NORM_HAMMING，如果 WET_K 设置为 3 或 4，则 BIREF 描述子会选择 3 个或 4 个点，那么后面匹配的时候应该选择的距离参数为 NORM_HAMMING2。

scoreType - 用于对特征点进行排序的算法，你可以选择 HARRIS_SCORE，也可以选择 FAST_SCORE，但是它也只是比前者快一点点而已。

patchSize - 用于计算 BIREF 描述子的特征点邻域大小。

1.2：flann 匹配：

在计算机视觉和机器学习中，对于一个高维特征，找到训练数据中的最近邻计算代价是昂贵的。对于高维特征，目前来说最有效的方法是 the randomized k-d forest 和 the priority search k-means tree，而对于二值特征的匹配 multiple hierarchical clusteringtrees 则比 LSH 方法更加有效。目前来说，fast library for approximate nearest neighbors (FLANN)库可以较好地解决这些问题。

快速近似 NN 匹配（FAST APPROXIMATE NN MATCHING）

1.2.1 随机 k-d 树算法（The Randomized k-d TreeAlgorithm）

a. Classick-d tree

找出数据集中方差最高的维度，利用这个维度的数值将数据划分为两个部分，对每个子集重复相同的过程。

参考 http://www.cnblogs.com/eyeszjwang/articles/2429382.html。

b. Randomizedk-d tree

建立多棵随机 k-d 树，从具有最高方差的 N_d 维中随机选取若干维度，用来做划分。在对随机 k-d 森林进行搜索时候，所有的随机 k-d 树将共享一个优先队列。

增加树的数量能加快搜索速度，但由于内存负载的问题，树的数量只能控制在一定范围内，比如 20，如果超过一定范围，那么搜索速度不会增加甚至会减慢。

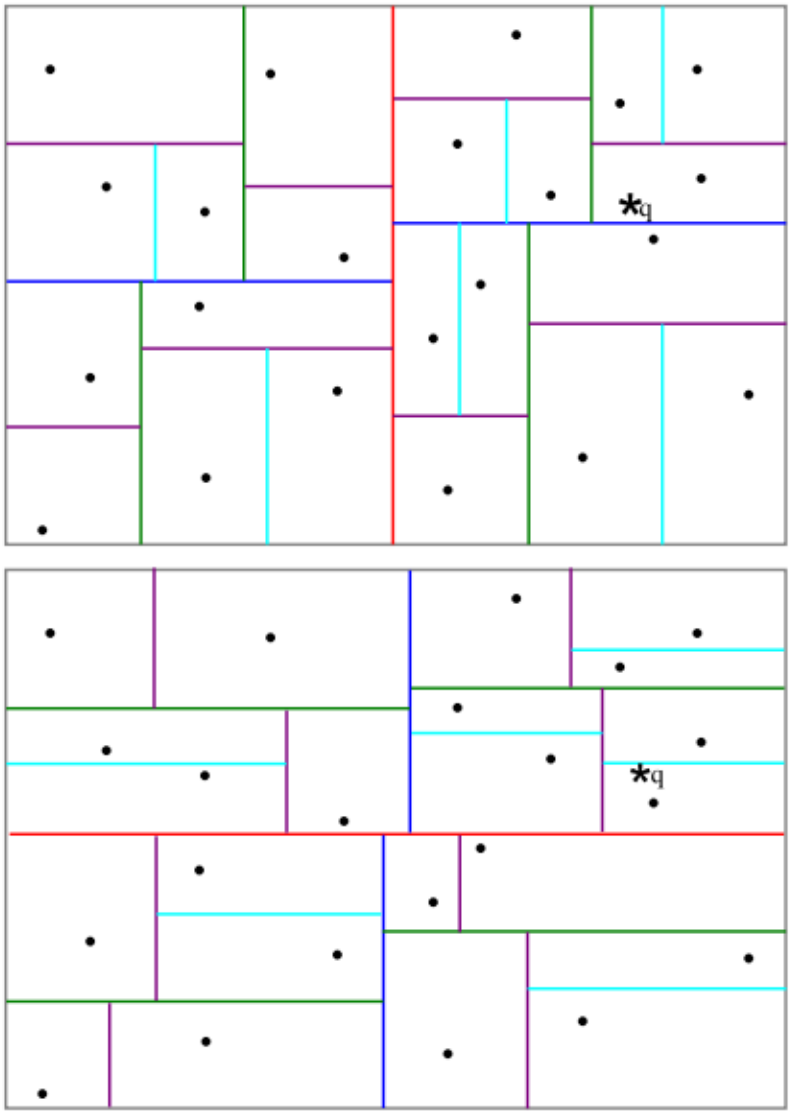


Fig. 2. Example of randomized kd-trees. The nearest neighbor is across a decision boundary from the query point in the first decomposition, however is in the same cell in the second decomposition.

1.2.2 优先搜索 k-means 树算法（The Priority Search K-MeansTree Algorithm）

随机 k-d 森林在许多情形下都很有效，但是对于需要高精度的情形，优先搜索 k-means 树更加有效。 K-means tree 利用了数据固有的结构信息，它根据数据的所有维度进行聚类，而随机 k-d tree 一次只利用了一个维度进行划分。

a 算法描述

算法 1 建立优先搜索 k-means tree:

(1) 建立一个层次化的 k-means 树；

(2) 每个层次的聚类中心，作为树的节点；

(3) 当某个 cluster 内的点数量小于 K 时，那么这些数据节点将做为叶子节点。

Algorithm 1 Building the priority search k-means tree

Input: features dataset D, branching factor K, maximum iterations I_{max}, centre selection algorithm to use C_{alg}

Output: k-means tree

```
1: if |D| < K then
2:    create leaf node with the points in D
3: else
4:    P ← select K points from D using the C_alg algorithm
5:    converged ← false
6:    iterations ← 0
7:    while not converged and iterations < I_max do
8:       C ← cluster the points in D around nearest centres P
9:       P_new ← means of clusters in C
10:      if P=P_new then
11:         converged ← true
12:      end if
13:      P ← P_new
14:      iterations ← iterations + 1
15:   end while
16:   for each cluster C_i ∈ C do
17:      create non-leaf node with center P_i
18:      recursively apply the algorithm to the points in C_i
19:   end for
20: end if
```

算法 2 在优先搜索 k-means tree 中进行搜索：

(1) 从根节点 N 开始检索；

(2) 如果是 N 叶子节点则将同层次的叶子节点都加入到搜索结果中，count += |N|；

(3) 如果 N 不是叶子节点，则将它的子节点与 query Q 比较，找出最近的那个节点 Cq，同层次的其他节点加入到优先队列中；

(4) 对 Cq 节点进行递归搜索；

(5) 如果优先队列不为空且 count<L，那么从取优先队列的第一个元素赋值给 N，然后重复步骤(1)

Algorithm 2 Searching the priority search k-means tree

Input: k-means tree T, query point Q, maximum number of points to examine L, number of neighbors K

Output: K nearest approximate neighbors of query point

procedure SEARCHKMEANSTREE(T,Q,L,K)

```
1: count ← 0
2: PQ ← empty priority queue
3: R ← empty priority queue
4: call TRAVERSEKMEANSTREE(T,PQ,R,count,Q)
5: while PQ not empty and count < L do
6:     N ← top of PQ
7:     call TRAVERSEKMEANSTREE(N,PQ,R,count,Q)
8: end while
9: return K top points from R
```

procedure TRAVERSEKMEANSTREE(N,PQ,R,$count$,Q)

```
1: if node N is a leaf node then
2:     search all the points in N and add them to R
3:     count ← count + |N|
4: else
5:     C ← child nodes of N
6:     C_q ← closest node of C to query Q
7:     C_p ← C\C_q
8:     add all nodes in C_p to PQ
9:     call TRAVERSETREE(C_q,PQ,R)
10: end if
```

聚类的个数 K，也称为 branching factor 是个非常主要的参数。

建树的时间复杂度 = O(ndKI (log(n)/log(K))) n 为数据点的总个数，I 为 K-means 的迭代次数。搜索的时间复杂度 = O(L/K * Kd * (log(n)/(log(K))) = O(Ld (log(n)/(log(K)))。

1.2.3 层次聚类树 (The Hierarchical ClusteringTree)

层次聚类树采用 k-medoids 的聚类方法，而不是 k-means。即它的聚类中心总是输入数据的某个点，但是在本算法中，并没有像 k-medoids 聚类算法那样去最小化方差求聚类中心，而是直接从输入数据中随机选取聚类中心点，这样的方法在建立树时更加简单有效，同时又保持多棵树之间的独立性。

同时建立多棵树，在搜索阶段并行地搜索它们能大大提高搜索性能（归功于随机地选择聚类中心，而不需要多次迭代去获得更好的聚类中心）。建立多棵随机树的方法对 k-d tree 也十分有效，但对于 k-means tree 却不适用。

Algorithm 1 Building one hierarchical clustering tree

Input: features dataset D
Output: hierarchical clustering tree
Parameters: branching factor K, maximum leaf size S_L

if *size of* $D < S_L$ **then**
  create leaf node with the points in D
else
  $P \leftarrow$ *select* K *points at random from* D
  $C \leftarrow$ *cluster the points in* D *around nearest centers* P
  for *each cluster* $C_i \in C$ **do**
    create non-leaf node with center P_i
    recursively apply the algorithm to the points in C_i
  end for
end if

Algorithm 2 Searching parallel hierarchical clustering trees

Input: hierarchical clustering trees T_i, query point Q
Output: K nearest approximate neighbors of query point
Parameters: max number of points to examine L_{max}

1: $L \leftarrow 0$ $\{L$ = *number of points searched*$\}$
2: $PQ \leftarrow$ *empty priority queue*
3: $R \leftarrow$ *empty priority queue*
4: **for** each tree T_i **do**
5: *call* TRAVERSETREE(T_i, PQ, R)
6: **end for**
7: **while** PQ *not empty and* $L < L_{max}$ **do**
8: $N \leftarrow$ *top of* PQ
9: *call* TRAVERSETREE(N, PQ, R)
10: **end while**
11: **return** K *top points from* R

procedure TRAVERSETREE(N, PQ, R)

1: **if** *node* N *is a leaf node* **then**
2: *search all the points in* N *and add them to* R
3: $L \leftarrow L + |N|$
4: **else**
5: $C \leftarrow$ *child nodes of* N
6: $C_q \leftarrow$ *closest node of* C *to query* Q
7: $C_p \leftarrow C \backslash C_q$
8: *add all nodes in* C_p *to* PQ
9: *call* TRAVERSETREE(C_q, PQ, R)
10: **end if**

2.实验过程

2.1 数据的表示

2.1.1 ORB 特征提取

在 orb 特征提取的过程中，我开始尝试了自己编写的 sift 特征提取，发现在图片集合比较大的时候认错率相当的高，因为毕竟很多学校的校徽本身就具有一定的相似度，而且单个校徽的特征点角点也不是特别明显，因此斟酌之后我还是用了 opencv 自带的 orb 特征提取，

我们知道 orb 特征提取具备一定的参数，最有用的是特征数量 nfeature，默认值为 500，它表示了要保留特征的最大数目。scoreType 设置使用 Harris 打分还是使用 FAST 打分对特征进行排序（默认使用 Harris 打分）。参数 WTA_K 决定了产生每个 oriented_BRIEF 描述符要使用的像素点的数目，默认值是 2，也就是一次选择两个点。在这种情况下进行匹配，要使用 NORM_HAMMING 距离。如果 WTA_K 被设置成 3 或 4，那匹配距离就要设置为 NORM_HAMMING2。在实际操作过程中我使用了默认的参数提取 ORB 特征。

```
query = cv2.imread('/home/hduser/17.png')
gray = cv2.cvtColor(query, cv2.COLOR_BGR2GRAY)
folder = '/home/hduser/logo'
orb=cv2.ORB_create()
kp= orb.detect(gray, None)
query_kp, query_ds= orb.compute(gray, kp)
```

效果如下

2.1.2 flannmatch 我们知道 flannmatch 是一种自适应性质的匹配，它可以根据输入的数据改变不同的匹配方式，其中包括 **LinearIndexParams**，该结构对应的索引进行线性的、暴力(brute-force)的搜索；**KDTreeIndexParams**，该方法对应的索引结构由一组随机 kd 树构成(randomized kd-trees)，它可以平行地进行搜索；**KMeansIndexParams**，该方法对应的索引结构是一个层次 k 均值树(a hierarchical k-means tree)，

CompositeIndexParams，该结构结合随机 kd 树和层次 k 均值树来构建索引等等搜索。

本次实验中为了结合我们上课所学内容，我使用了 LSH 式的索引建立方式，

```
FLANN_INDEX_LSH=6
indexParams=dict(algorithm=FLANN_INDEX_LSH,
                 table_number = 6, #12/home/hduser/jiaoda.jpg
                 key_size = 12,      #20
                 multi_probe_level = 2)#2
searchParams=dict(checks=500)
flann = cv2.FlannBasedMatcher(indexParams, searchParams)
```

如上所示我把匹配的级数参数定为 6，把 keysize 定为 16，LSH 索引建立的 multi-probe 定为 2，最后把迭代的次数定为 500，就完成了 flann 匹配器的搭建。

2.1.3 提前处理

为了提高我们图片匹配的实时性，我们要把我们图片库里的所有图片都先进行一次预处理，提取它们的 orb 特征并保存在 npy 文件中

```
import cv2
import numpy as np
from os import walk
from os.path import join
detector=cv2.ORB_create()
def create_descriptors(folder):
    files = []
    for (dirpath, dirnames, filenames) in walk(folder):
        files.extend(filenames)
    for f in files:
        if '.png in f:
            save_descriptor(folder, f, cv2.ORB_create())

def save_descriptor(folder, image_path, feature_detector):
    if image_path.endswith("npy"):
```

```
        return
    img = cv2.imread(join(folder,image_path),0)
    keypoints, descriptors = detector.detectAndCompute(img, None)
    descriptor_file = image_path.replace("png", "npy")
    np.save(join(folder, descriptor_file), descriptors)


if __name__=='__main__':
    path = '/home/hduser/logo'
    create_descriptors(path)
```

这个操作可以通过读取每一个文件的后缀名，如果后缀名是 png 那么就打开这个文件并且读取并保存这个文件的 orb 特征，在保存 orb 特征的同时也产生一个同样名字的但是把后缀名改成 npy 的文件。

2.1.4 实时匹配中使用 flann 匹配进行匹配

通过 flann.knnmatch 进行匹配

```
potential_culprits = {}
gg=[]
for d in descriptors:
    try:
        matches = flann.knnMatch(query_ds, np.load(join(folder, d)), k=2)
        #good = []
        p=0
        try:
            for i, (m, n) in enumerate(matches):
                if m.distance < 0.7* n.distance:
                    p+=1
            print p
        except Exception, e:
            print e
        print("img is %s ! matching rate is (%d)" % (d, p))
```

```
            potential_culprits[d]+= p*2
        except Exception, e:
            print e
```

我们用 potential_culprits 这个字典将第 n 张图片和第 n 张图片的匹配结果对应起来，这样就可以用某个匹配数值代替一张图片，之后对文件夹内的每一个 npy 文件都进行 match，matches = flann.knnMatch(query_ds, np.load(join(folder, d)), k=2)，因为我们之前设置的 k 是 2 维的，所以我们在 matches 的结果里会看到一个点（m，n）m，n 都是一个 D_match 常量，当 m.distance < 0.7* n.distance:，我们认为这个特征点匹配成功

2.1.5

之后为了提高准确度，用粗暴匹配 bfmatch 进行二次匹配，

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING)
for d in descriptors:
    try:
        matches = bf.match(query_ds, np.load(join(folder, d)))
        matches = sorted(matches, key=lambda x: x.distance)
        #good = []
        p=0
        try:
            for i, m, in enumerate(matches):
                if m.distance < 55:
                    p+=1
            print p
        except Exception, e:
            print e
        print("img is %s ! matching rate is (%d)" % (d, p))
        potential_culprits[d]+=0.25*p
        gg.append((d,p))
    except Exception, e:
```

```
            print e
```

bf 匹配的原理比较简单，就是直接进行逐一匹配，然后设立匹配阈值，低于阈值的证明匹配度足够高，因此该特征点匹配成功。

2.1.6 寻找匹配程度最好的图片

想要寻找匹配程度最好的图片，那其实就是比较之前设立的字典的最大的 value 所对应的 key

```
max_matches = None
potential_suspect = None
for culprit, matches in potential_culprits.items():
    if max_matches == None or matches > max_matches:
        max_matches = matches
        potential_suspect = culprit

print("potential suspect is %s     , points %d" % (potential_suspect.replace(".npy", "").upper(),max_matches))
tjpp=potential_suspect.replace(".npy", "").upper()
tjpp=int(tjpp)
file = open("/home/hduser/logo.txt","r")
contents = file.readlines()
print contents[tjpp-1]
```

由上面可见，我们可以通过立一个最大值的 flag，当目前的匹配 value 大于 flag 时候，更新 flag，同时保存当前的 key，这样遍历字典之后就找到了目标图片。

最后一步是打开之前建立的一个序号对应学校的文件，通过最后得到的最佳匹配的序号输出最佳匹配学校名称，就大功告成啦！

3．成果展示

四．前端、页面连接与前后端数据整合

注：所有内容除一个JavaScript外联库以外都是纯手写内容，没有采用任何现有的模板。

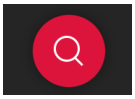
1．前端

所有前端都采用 HTML 为主语言进行编写，如下图所示，完整的搜索页与结果页代码长达约 1500 行，在此不做展示，详见源码。

```
                <p><a href="https://baijiahao.baidu.com/s?id=1602773187785033252&wfr=spider&for=
        </div>

    </div>
    <div class="content has-main-image">
        <div class="content-inner" data-content-field="main-content">
            <div class="title-desc-wrapper">
                <div class="title-desc-inner" data-collection-id="5727a163d210b841fd838f42">
                    <div class="page-title-wrapper"><h1 class="page-title">UNDER THE BIG BLACK S
                    <br> <!-- this keeps the inline-blocks from collapsing -->
                    <div class="page-desc" data-content-field="description"><p>OUT NOW</p><p><a
                </div>
            </div>
```

我的前端部分写了很长时间，差不多写了三周每天写五六个小时，是一块一块元素结构拼出来的。首先是局部的动态搜索框，我采用 jQuery 做了动态动画，点击按钮可以打开对应的搜索框，如下图所示：

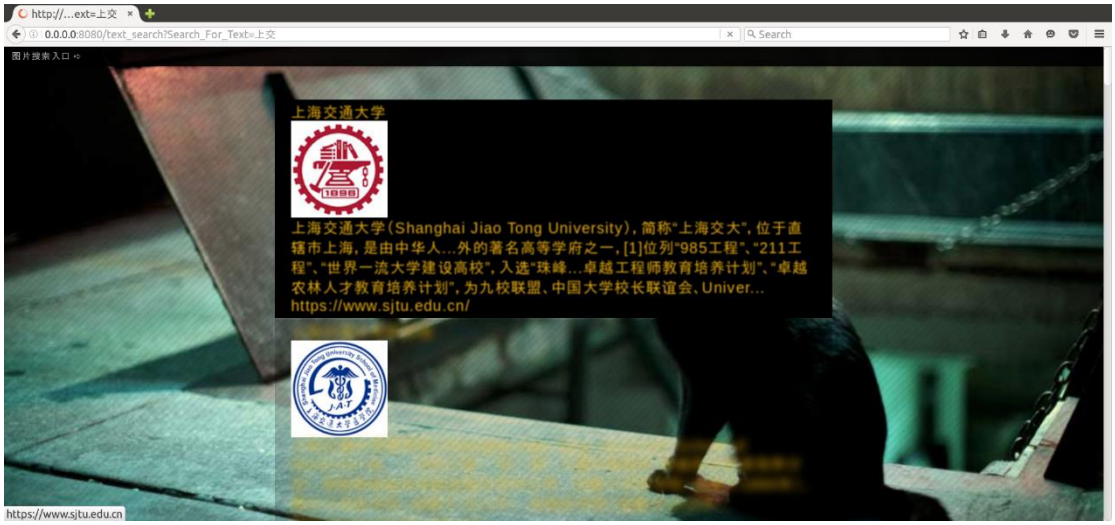


之后我又制作了动态按钮，在最终网页中进行页面转换，采用 JavaScript 制作脚本。

整体制作 CSS 框架，一定要注意 CSS 的兼容性，如小 CSS 嵌套在大的 CSS 中的时候一定要注意格式和信息重载。这个如果不注意 debug 的时候真的伤头皮、掉头发，因为根本无法看出问题，并且修改了参数也没有结果呈现。我的搜索页 CSS 框架储存在了 index.css 文件中。

结果页面的制作采用另一种风格，我采用分框效果，每一个结果框我都使用 jQuery 制

作了动态模糊效果，如下图所示，鼠标聚焦的地方进行高亮，其余部分模糊。



此时鼠标聚焦在“上海交通大学”，其余部分全部模糊。

鼠标移开全部高亮。

结果页的 css 存在了 style2.css、style3.css、style4.css 中。

```
.bmenu{
    padding: 0px;
    margin: 0 0 10px 0;
    position: relative;
}
.bmenu li{
    font-size: 50px;
    display: block;
}
.bmenu li a{
    display: block;
    text-transform: uppercase;
    text-shadow: 1px 1px 2px rgba(89,22,20,0.3);
    color: #581514;
    padding: 5px 20px;
    margin: 2px;
    background: rgba(255,255,255,0.2);
    letter-spacing: 1px;
    -webkit-transform: skew(-12deg);
```

此处仅对一个结果页和部分模块进行了具体展示，其余部分详见最后完整结果展示。

在图片上传框的部分，我采用 JavaScript 进行传参，如下图所示：

```
  <div class="title-desc-inner" data-collection-id="5727a163d210b841fd838f42" data-edit-main-image=
      <div class="page-title-wrapper"><h1 class="page-title" data-content-field="title">

              <meta charset="UTF-8">
              PHOTO UPLOAD <br><br>
          <style>
              .center{
                  display: flex;
                  justify-content: space-between;
                  flex-direction: column;
                  align-items: center
              }
          </style>
          <div class="center">
```

```
<div class="center">
<script src="/static/js/jquery.js"></script>
<script src="/static/js/upload.js"></script>
    <br><br>
<div id="drop_area"></div>
<form method="POST"  action="/photo" >
<input type="file" name="myfile" >

<script type="text/javascript">
    var dragImgUpload = new DragImgUpload("#drop_area",{
        callback:function (files) {
            //回调函数，可以传递给后台等等
            var file = files[0];
            console.log(file.name);
```

操作后的前端效果是：

上传图片后可以看到图片的信息和图片名。

同时我网页中所有的前端按钮都用 jquery 制作了选中特效，如下图所示：

未选中：

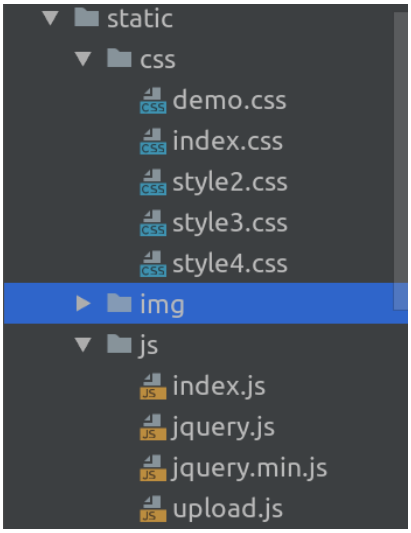
选中：

2. 页面连接

页面连接采用 web.py 的服务器 action、get 进行连接，需要注意的是，web.py 进行页面连接时一定要传递参数，即使参数为空，否则无法连接，这个最开始我 debug 了很久无法解决，传一个无用的参数即可，如下图所示。

```
$def with (t,answer)
<head>
    <title></title>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="Blur Menu with CSS3 Transitions" />
    <meta name="keywords" content="css3, transitions, menu, blur, navigation, typography, font,
    <meta name="author" content="Codrops" />
    <link rel="shortcut icon" href="../favicon.ico">
    <link rel="stylesheet" type="text/css" href="/static/css/demo.css" />
    <link rel="stylesheet" type="text/css" href="/static/css/style4.css" />
    </head>
    <body style="background-image: url(/static/img/pattern.png), url(/static/img/4.jpg);">
```

同时使用 webpy 时一定要迎合 webpy 对静态文件的需求，否则 JavaScript 和 css 等静态文件都是连接不上的，一定要放在“static”文件夹下，并且这个文件夹不可更改名字。直接使用 url 连接静态文件比较方便，但最后还是采用了本地连接，如下图所示，全部采用静态库进行连接。



3. 前后端数据整合

使用 webpy 的 post、get 等进行页面连接，同时衔接之前其他组员的后端程序，建议提前确定好版本和接口，否则衔接时会出现比较多的问题，这个不伤头皮，直接掉头，我们组最开始没有讨论好接口连接，所以我整合的时候遇到了很多麻烦，接口问题、版本问题等等。

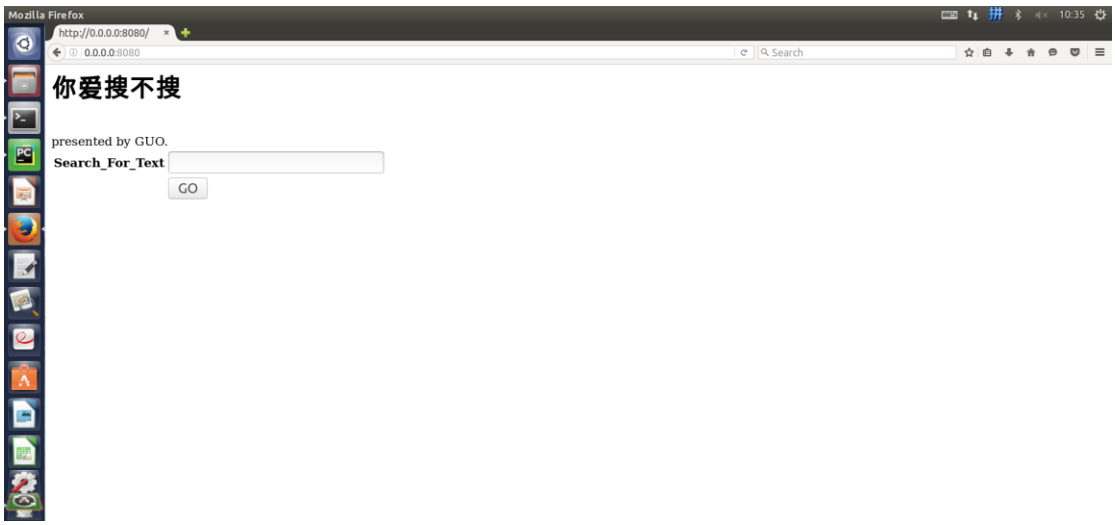
因此面向对象是必要的，同时衔接时图片的 post 采用地址传递，确保信息对应。

Html 和 python 一定注意 python 的缩进格式，采用 $ 引用 python 脚本。

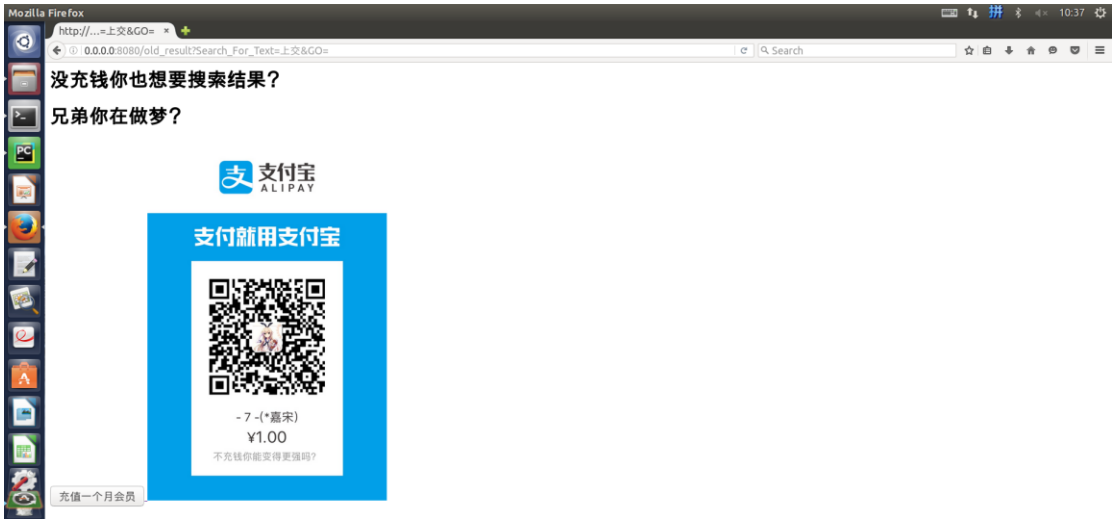
多语言衔接非常恶心，多输出可以更高效的 debug，语言的连接一定要注意格式与形式，否则基本看不出问题，因为任何一个语言都不会再衔接处报错。

4. 最终前端&整合结果

打开首页：



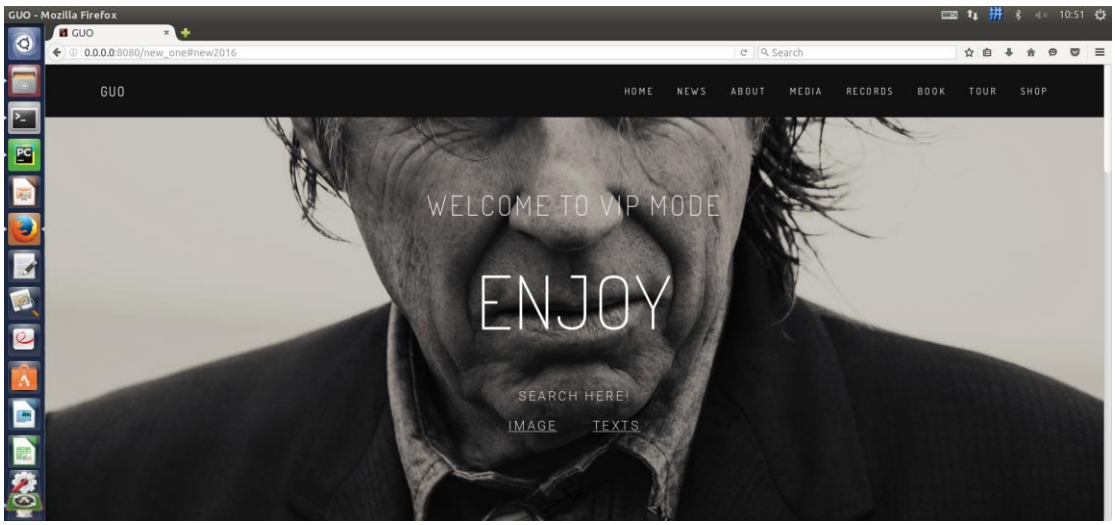
大家不禁感到困惑，为何制作了这么久的前端就是上图这个样子，我们不妨搜索“上交”试试。



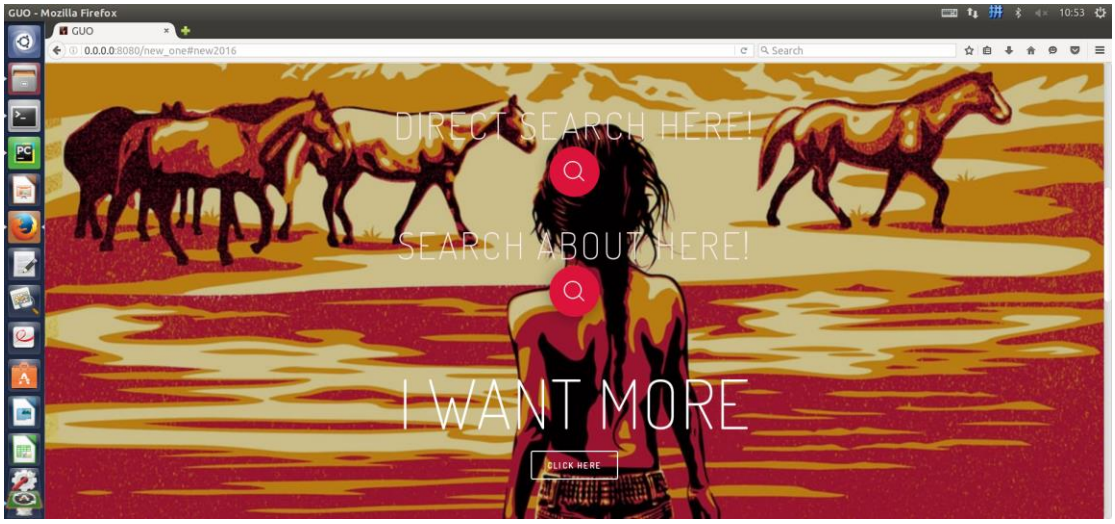
果然，当你发现你的网页索然无味的时候，就是应该充钱的时候了，我们充值一下会员，点击充值一个月会员。（试图商业化 hhh(*≧▽≦*)）

注：所有支付宝的效果采用 Alipay 的接口，可以完成线上缴费（只是为了娱乐 hhh，不是真的要钱 hhh）。

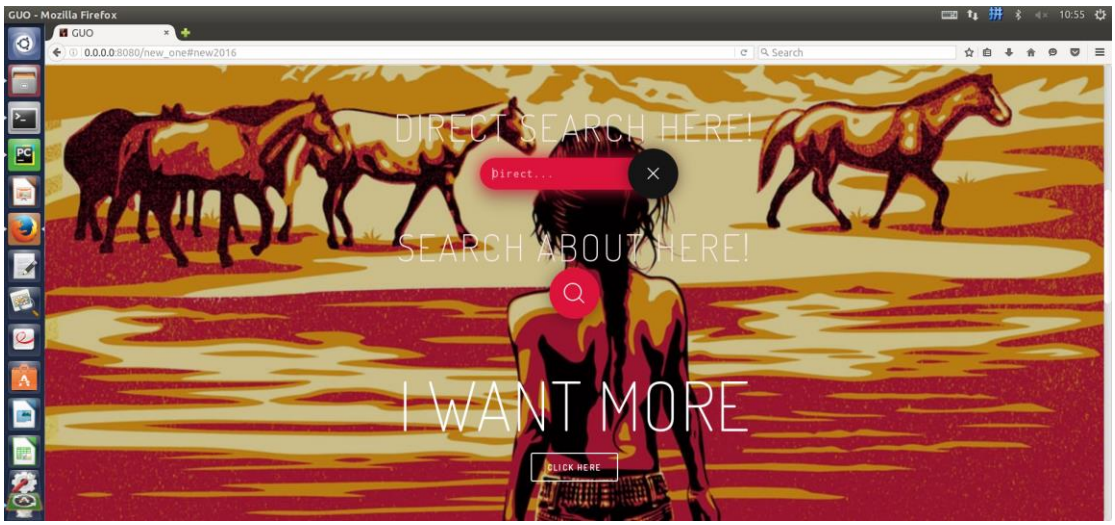
充值后的首页如图所示：

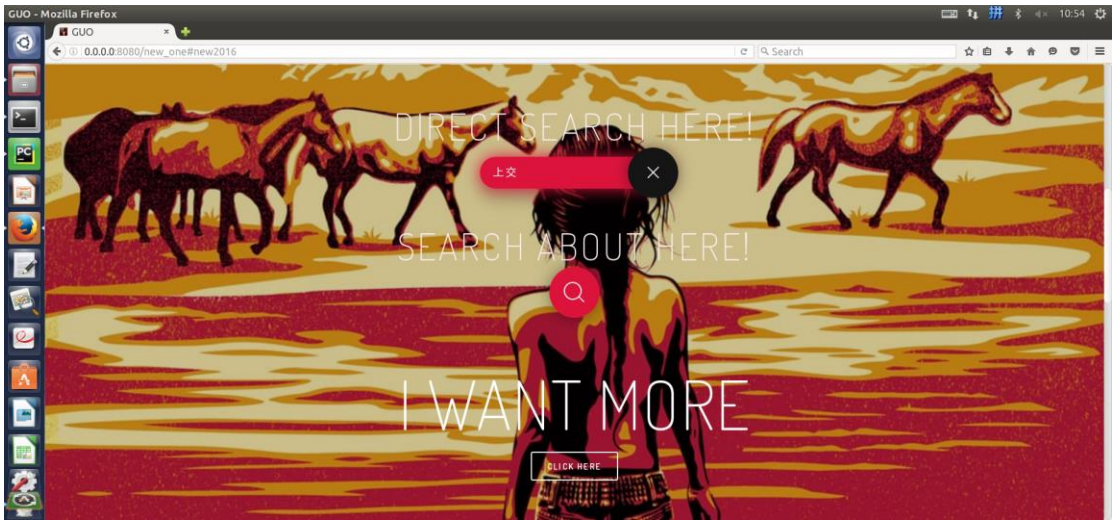


你会发现还是没有搜索框，在页面最下方位置我嵌入了 JavaScript 的按钮，点击即可，会跳转到下一个页面：



Direct Search 就是寻找和学校有关的直接信息，我们搜索“上交”为例。


GUO - Mozilla Firefox
GUO
0.0.0.0:8080/new_one#new2016
Search
DIRECT SEARCH HERE!
Direct...
SEARCH ABOUT HERE!
I WANT MORE
CLICK HERE


GUO - Mozilla Firefox
GUO
0.0.0.0:8080/new_one#new2016
Search
DIRECT SEARCH HERE!
上交
SEARCH ABOUT HERE!
I WANT MORE
CLICK HERE

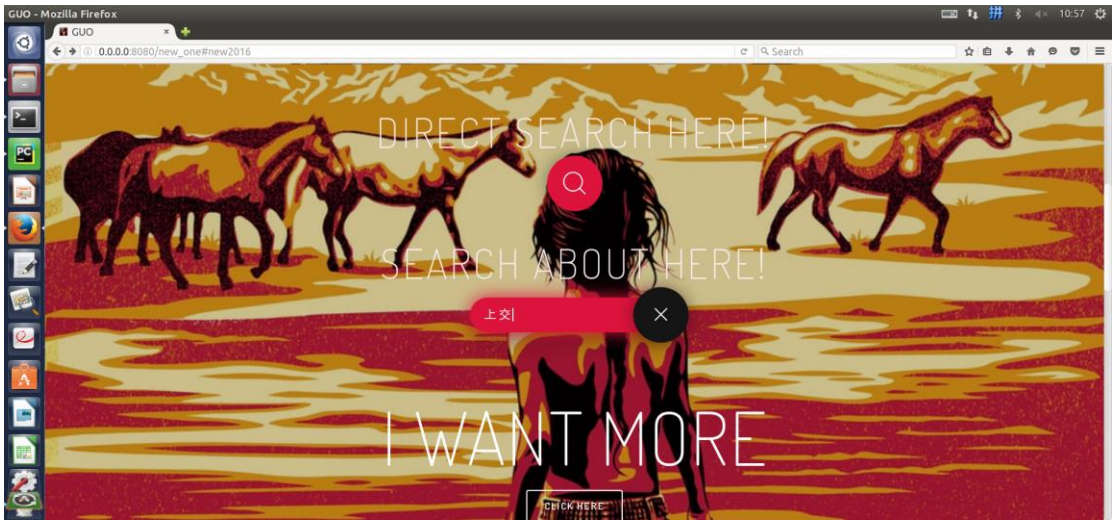
之后就给出了搜索结果：


http://...ext=上交
0.0.0.0:8080/text_search?Search_For_Text=上交
Search
图片搜索入口
上海交通大学
上海交通大学(Shanghai Jiao Tong University)，简称"上海交大"，位于直辖市上海，是由中华人...外的著名高等学府之一，[1]位列"985工程"、"211工程"、"世界一流大学建设高校"，入选"珠峰...卓越工程师教育培养计划"、"卓越农林人才教育培养计划"，为九校联盟、中国大学校长联谊会、Univer...
https://www.sjtu.edu.cn/
https://www.sjtu.edu.cn

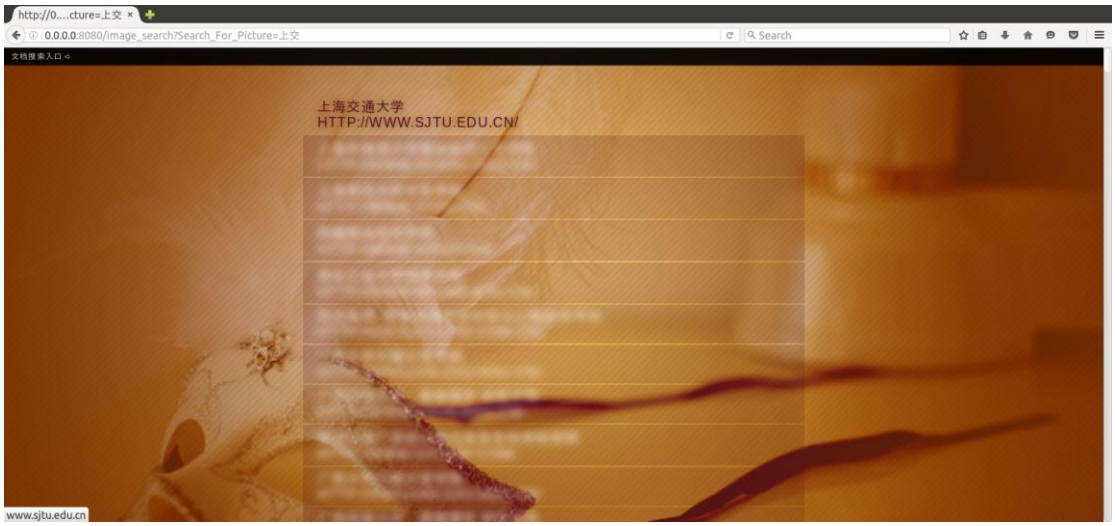
鼠标聚焦的地方为高亮，其他地方 Jquery 制作了动态模糊。

鼠标移开全是高亮，点击每一个搜索框都可以打开对应的网页。

再使用首页上的 About 搜索，就是搜索与学校有关的相关网页。



结果页为每一个学校相关的学校网址和标题，依旧使用 jquery 做了倾斜动画，鼠标聚焦的地方为高亮正体。



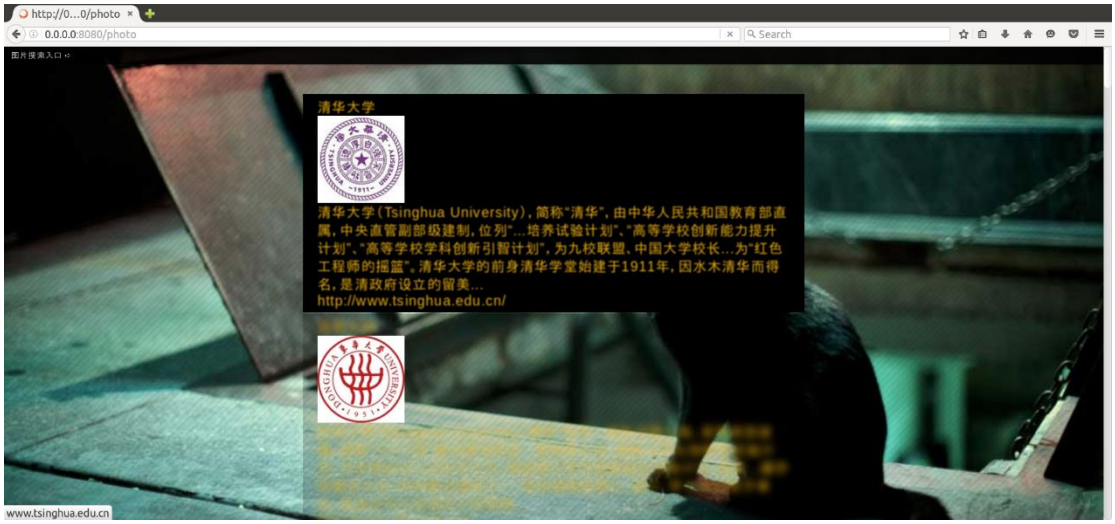
将鼠标移开就是完整的结果页，如下图所示：

点击首页的 I WANT MORE 的 CLICK HERE 是续费会员的按钮 hhh，这里就不展示了，可以自己尝试 hhh。

我们将首页往下拉，就得到了图片搜索入口：

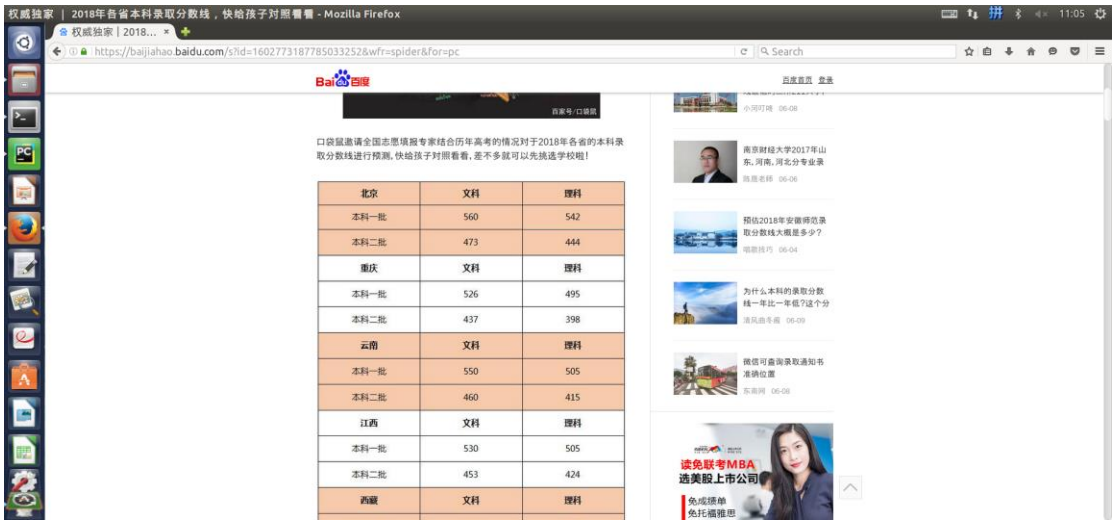
我们搜索“清华”为例：

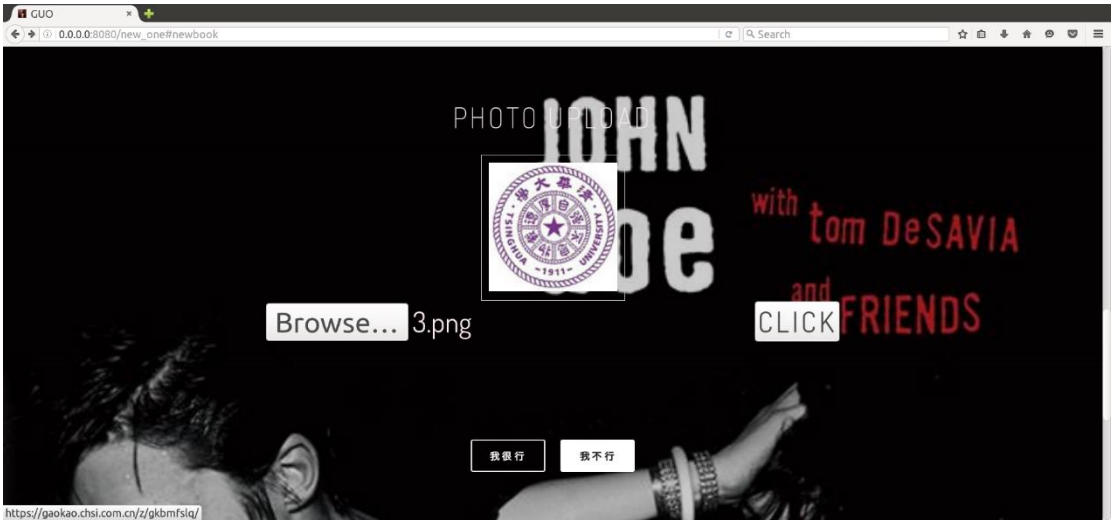
点击 CLICK，就得到了最后的搜索结果。



我们看到还有两个按钮，“我很行” 和 “我不行”。

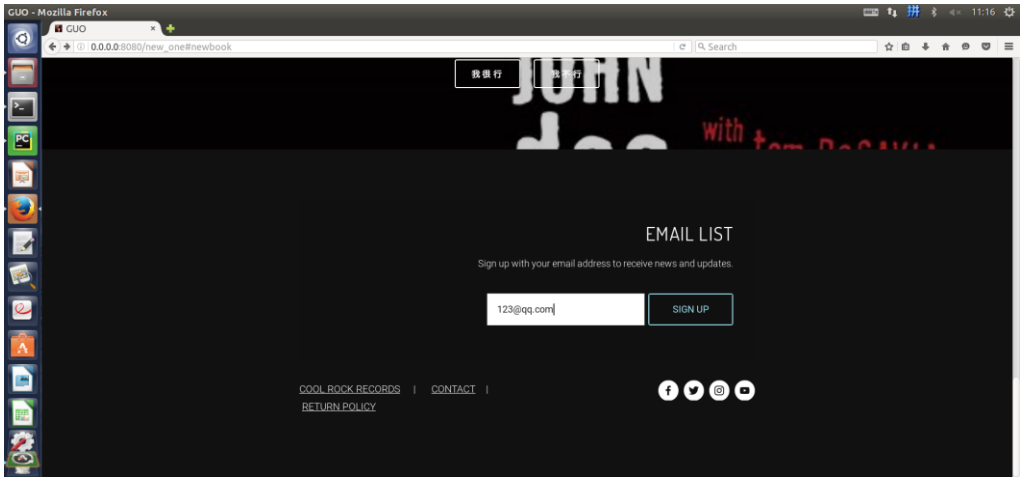
我们点击 “我很行”，我面向用户贴心的连接到了对应学校的提档线与录取分数线：





我们再点击“我不行”，我更贴心的链接到了对应的第二年高考报名的按钮，来年再战！

最后，我们来到网页的最后，就是我们的注册界面，你可以输入邮箱进行注册，这样就会注册我们的网页，可能续费会员有优惠 hh，我们会邮箱联系您：



附：组内分工：

所有前端 & 前后端连接、最后整合 — 郭嘉宋；

爬虫部分 — 刘若峰；

文字数据处理与整合 — 马嘉琪；

图像识别 — 姜泽坤；