Texture packing

组员：肖思勃、付泽阳

2024/5/21

Chapter1:Introduction

1.1 问题描述

Texture packing问题是一个二维面积最小化问题。题目给定一组矩阵和一个宽度有限高度无限的条带，将这些矩阵打包到条带中，使得高度最小。

1.2 问题思考

- 做什么：我们需要找到一种排列这些矩形的方法，使得它们在条带中的总高度尽可能低，避免浪费空间。我们组做了NFDH和skyline两种算法
- 问题意义：Texture packing问题在计算机图形学和计算几何学中有重要应用，特别是在游戏开发和图像处理等领域。例如：
  - 纹理贴图：在3D渲染中，将多个纹理合并成一个纹理贴图，以减少渲染时的切换，提高性能。
  - 内存优化：减少内存的浪费，提高缓存利用率和数据传输效率。
  - 物流和仓储管理：优化空间利用，减少存储和运输成本。

1.3 输入格式

- 第一行n：矩形数目
- 后面n行，每行包括矩阵的长和宽

1.4 输出格式

- 所需要的高度

1.5 样例输出

见后面算法分析

Chapter2:Algorithm Specification

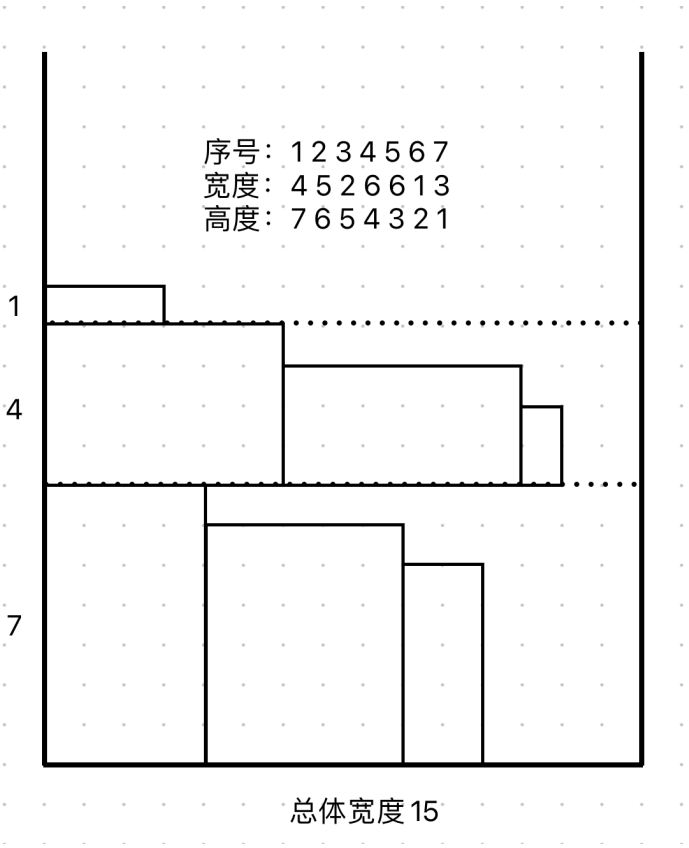
2.1 NFDH-Next Fit Decrease Height

- 数据结构
  一个结构体用来存储箱子，其中有箱子坐标、宽高，并且定义箱子的比较方法

```
struct Bin{//定义箱子的结构
    double x;//箱子放置的x坐标
    double y;//箱子放置的y坐标
    double height;//箱子的高度
    double width;//箱子的宽度

    friend bool operator < (const Bin& i1, const Bin& i2){
        return i1.height>i2.height;
    }
};
```

- 算法描述



  - 先将所有的箱子按照高度排序
  - 将容器视作一层一层的货架，每一层货架的高度由这个货架上的第一个箱子（也就是最高的箱子）决定
  - 按次序放入箱子，如果放不下就放在下一层

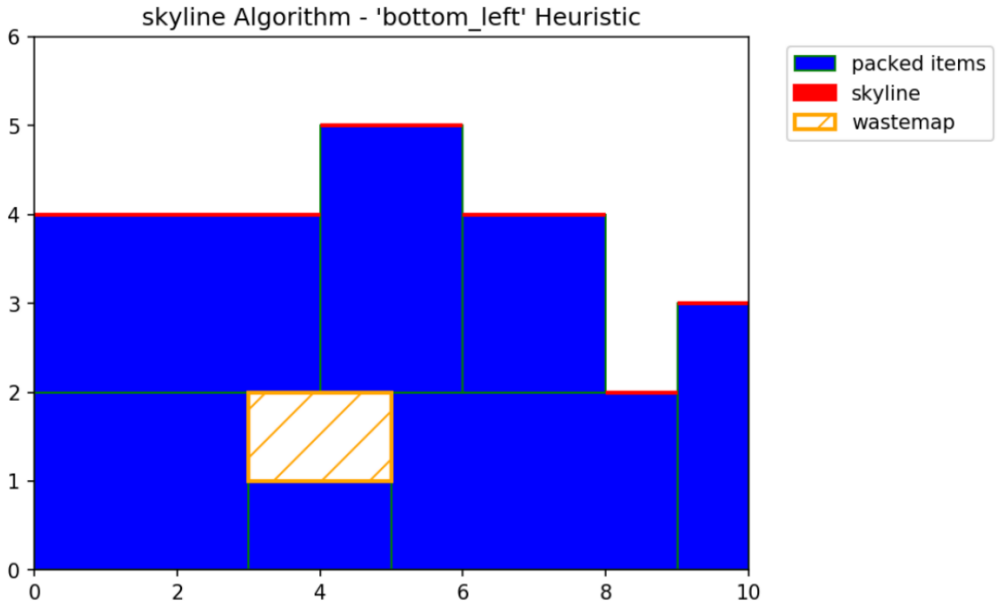
```
NFDH{
    sort by height dec;
    double allHeight = 0;
    for every bin in bins{
        if(can put the bin in this level){
            put the bin in this level;
            update something;
        }else{
            put the bin in the next level;
            update allHeight and something;
        }
    }
    cout<<allHeight;
}
```

- 时间复杂度为$O(NlogN)$，具体分析见后文算法分析
- Approximation ratio=2，有不等式$NFDH(L) <= 2OPT(L) + h_{max}$，具体分析见后文算法分析

2.2 SkyLine--online

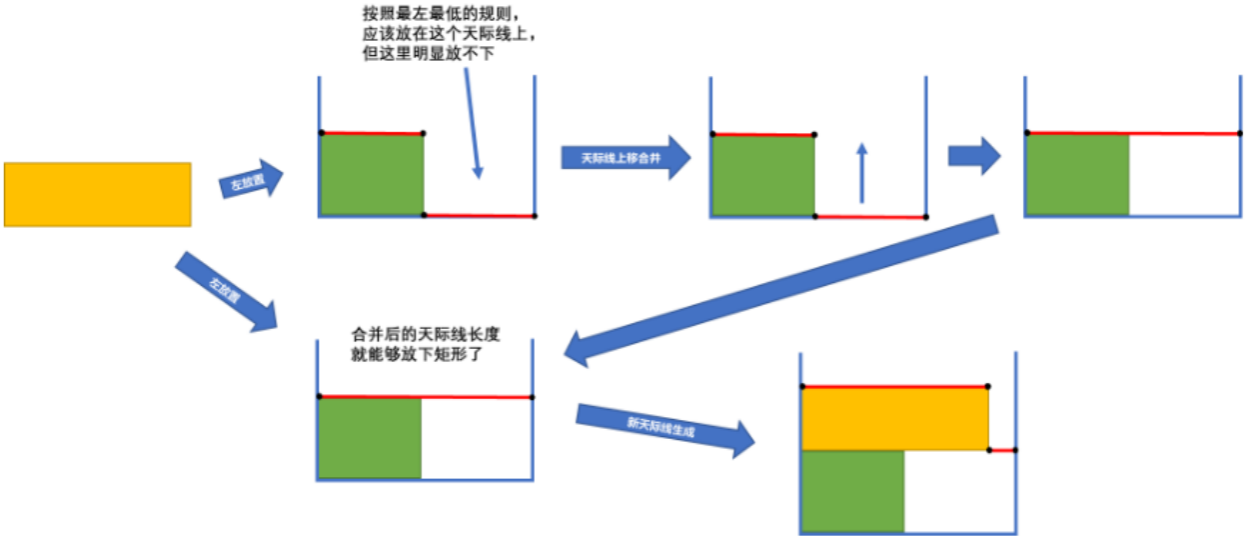
- 算法描述

- 天际线启发式是一种简单而快速的打包方法。它只维护一个水平段的列表，这些水平段也被称为天际线段，由整个打包区域的顶端的线段组成。
- 一个天际线段可以由两个对象决定：左边的端点坐标（x，y）和线段的长度w。在天际线之上的是空间，它可以被看作是未打包区域的一个子集。
- 一个空间由三个对象定义：左墙的高度hl、右墙的高度hr和其对应的天际线段。



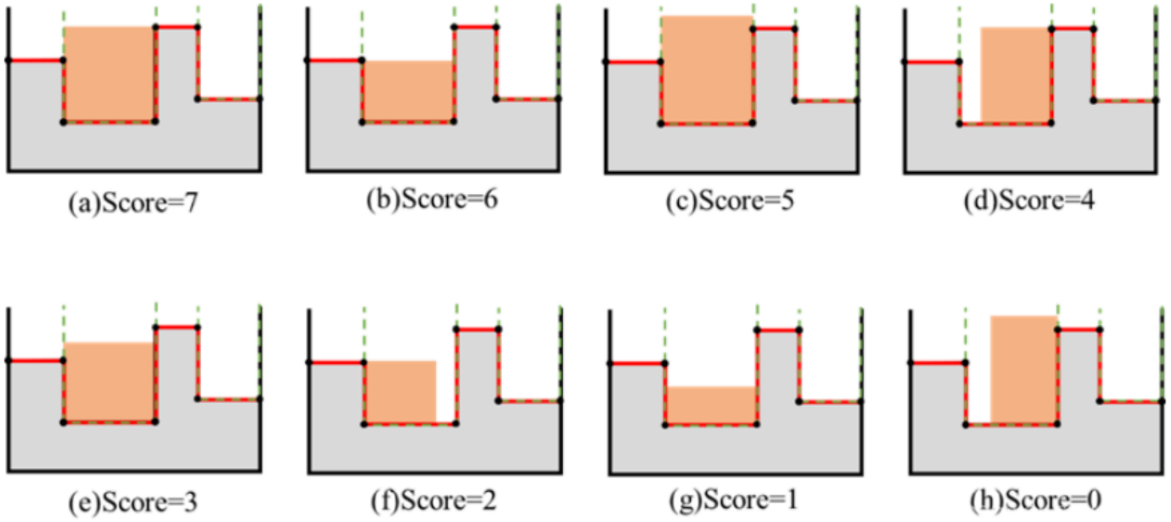
- 新天际线的生成
  矩形只能放置在天际线的最左端或者最右端，一旦放置在天际线上，就会将被放置的天际线“切割”，即根据矩形的放置方式缩短为一条新的天际线。然后再将矩形的顶部的边作为一条新的天际线加入天际线队列中。也就是说，一般情况下，放置一个矩形会多生成一条天际线。

- 新天际线的合并



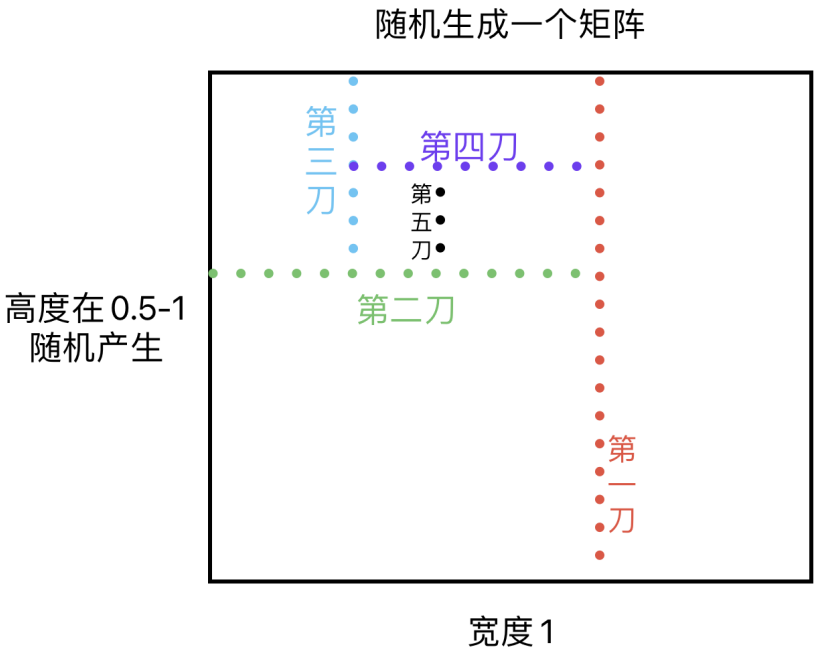
- 启发式策略

- 采取BL(bottom-left)启发式方法，即尽可能选择y轴最小，x轴也最小（左下角为O点）的空白空间来堆放下一个。
- 下图展示了hl小于hr的八种情况（类似的hl大于等于hr的情况只需要将hr和hl互换即可）。对于hl大于等于hr，类似地，也存在八种情况。我们会选择得分最高的放置方式，如果几个矩形有相同的得分，则选择索引较小的矩形的放置方式。
  - a) 矩形的宽等于天际线长度，且矩形的高等于hr
  - b) 矩形的宽等于天际线长度，且矩形的高等于hl
  - c) 矩形的宽等于天际线长度，且矩形的高大于hr
  - d) 矩形的宽小于天际线长度，且矩形的高等于hr，此时靠右放置
  - e) 矩形的宽等于天际线长度，且矩形的高大于hl小于hr
  - f) 矩形的宽小于天际线长度，且矩形的高等于hl
  - g) 矩形的宽等于天际线长度，且矩形的高小于hl
  - h) 矩形的宽小于天际线长度，且矩形的高不等于hr，此时靠右放置



Chapter3: Testing Results

3.1 测试程序设计



- 首先生成一个宽度固定为1，高度在0.5-1之间的矩形
- 切割的总刀数为cuts（在测试程序中宏定义出来，可以更改），以cuts=5为例
- 第一步先纵向切一刀，再在最半边和右半边随机选择一个再横行切一刀。依次纵向横向交替进行切割
- 当一个矩形被切割五次得到六个小矩形之后，回到第一步新建立一个随机矩形进行切割（因此需要保证输入的需要产生矩形数目为cuts+1的倍数）
- optimal solution的最小总高度为第一步产生所有没有被切割的矩形的高度之和

产生的输入用例

- n=6
  6
  0.872328 0.692874
  0.127672 0.144429
  0.0743036 0.548445
  0.0377267 0.0533682
  0.0306627 0.510718
  0.510718 0.0227055
- n=12
  12
  0.0815374 0.692623
  0.110749 0.918463
  0.13013 0.581874
  0.788333 0.118282
  0.536742 0.463592
  0.463592 0.251591
  0.603164 0.623869
  0.376131 0.462885
  0.0886549 0.140279

0.287476 0.0287224
0.111556 0.0139264
0.111556 0.27355

- 18
  18
  0.693195 0.479279
  0.520721 0.0359811
  0.21633 0.657213
  0.520759 0.304391
  0.136455 0.0478179
  0.136455 0.256574
  0.641719 0.518572
  0.358281 0.2838
  0.265488 0.234772
  0.186167 0.0927933
  0.0481447 0.0486054
  0.0486054 0.0446486
  0.554901 0.77308
  0.22692 0.250463
  0.0312468 0.304438
  0.0755015 0.195674
  0.1305 0.228936
  0.228936 0.0651733

3.2 算法正确性测试

- 输入
  18
  0.992676 0.67925
  0.32075 0.00793728
  0.275819 0.984738
  0.0449306 0.503263
  0.481475 0.0219627
  0.119932 0.0229678
  0.0149224 0.361543
  0.095089 0.00804544
  0.266454 0.00804544
  0.897275 0.772332
  0.227668 0.783003
  0.0586488 0.114271
  0.0748952 0.169019
  0.0393761 0.0643075
  0.104712 0.0185374
  0.0208387 0.0518717

0.05284 0.00223987

0.0185988 0.05284

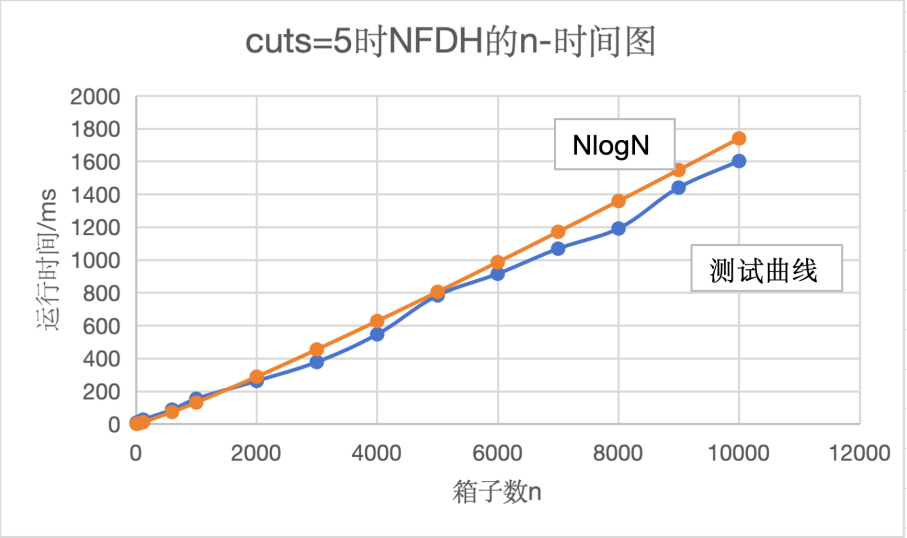
- 输出

```
<bound method Skyline.bin_stats of Skyline([Item(width=9927, height=6793, x=0, y=0), Item(width=3208, height=80, x=0, y=6793), Item(width=9848
, height=2759, x=0, y=6873), Item(width=5033, height=450, x=0, y=9632), Item(width=4815, height=220, x=5033, y=9632), Item(width=1200, height=
230, x=5033, y=9852), Item(width=3616, height=150, x=6233, y=9852), Item(width=81, height=951, x=9849, y=6793), Item(width=2665, height=81, x=
6233, y=10002), Item(width=8973, height=7724, x=0, y=10083), Item(width=7831, height=2277, x=0, y=17807), Item(width=587, height=1143, x=8973,
 y=10002), Item(width=749, height=1691, x=8973, y=11145), Item(width=644, height=394, x=8973, y=12836), Item(width=186, height=1048, x=9722, y
=10002), Item(width=209, height=519, x=9722, y=11050), Item(width=23, height=529, x=9849, y=7744), Item(width=186, height=529, x=9722, y=11569
)])>
```

3.3 运行时间测试

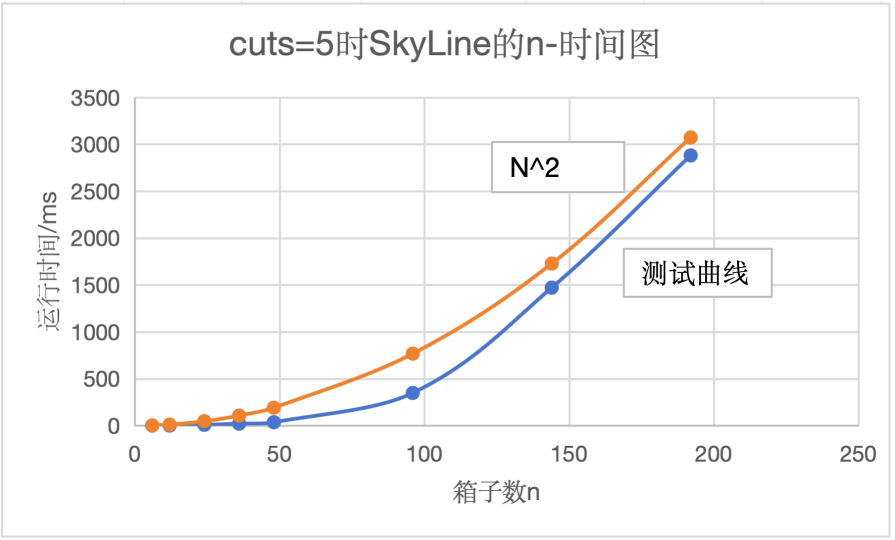
- NFDH

n	时间/ms	nlogn/23
12	8	0.563051085
60	15	4.638655436
120	26	10.84790215
600	87	72.47351088
1002	153	130.7334547
2004	262	287.6957837
3000	377	453.537555
4002	546	626.7962127
5004	783	804.8427728
6000	915	985.604674
7002	1068	1170.619356
8004	1191	1358.350864
9000	1440	1547.312286
10002	1602	1739.516029



- Sky

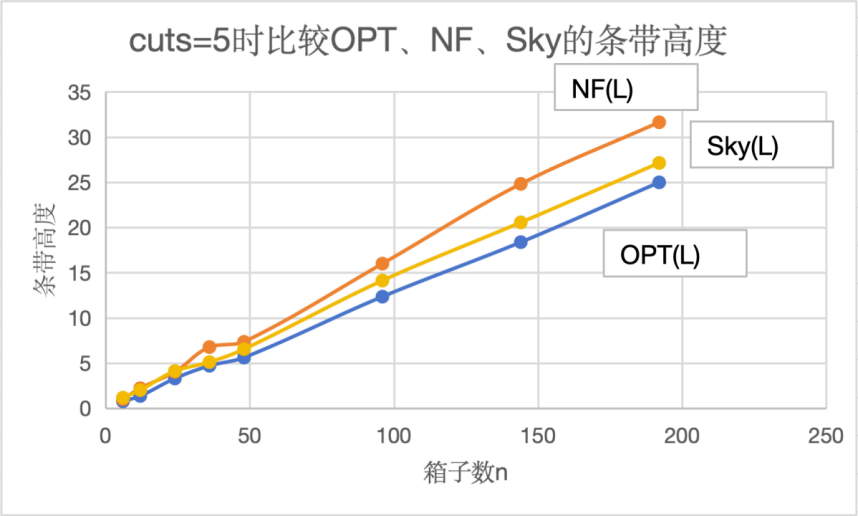
n	时间/ms	n^2/12
6	1.8	3
12	2	12
24	10.1	48
36	22.1	108
48	36.9	192
96	349.3	768
144	1471.5	1728
192	2880.9	3072



3.4 近似率测试

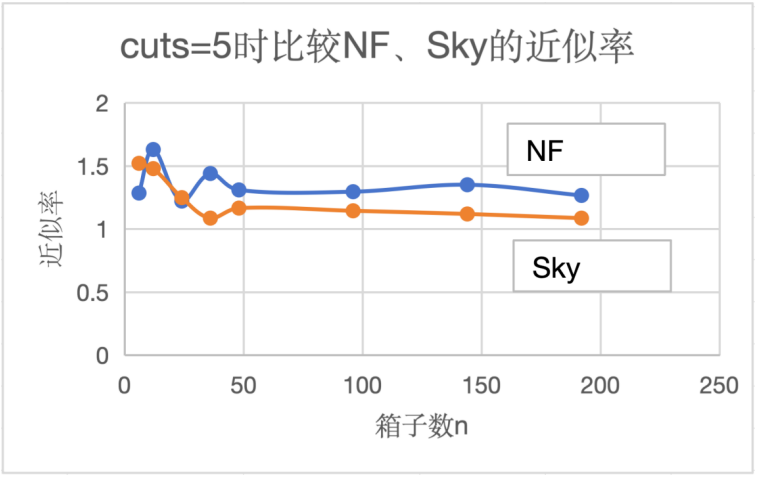
cuts=5时N变化比较OPT、NF、Sky的条带高度

n	opt	NF	Sky
6	0.772	0.992	1.174
12	1.36	2.217	2.011
24	3.294	4.023	4.113
36	4.703	6.772	5.108
48	5.609	7.347	6.538
96	12.346	15.997	14.127
144	18.3731	24.818	20.56
192	24.989	31.64	27.136



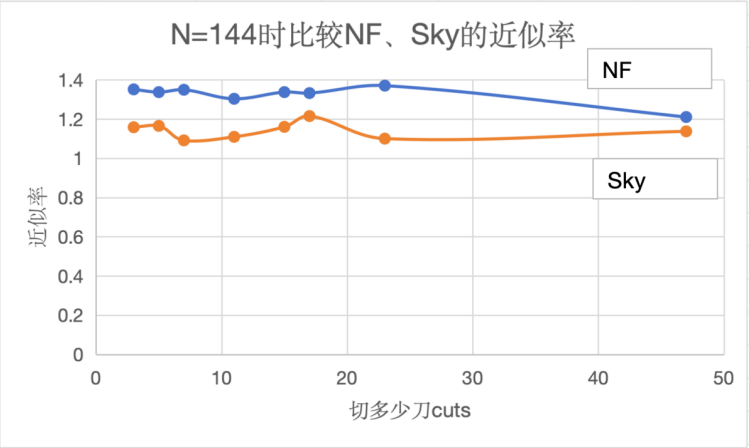
cuts=5时N变化比较NF和Sky的近似率

n	appro-NF	appro-Sky
6	1.284974093	1.520725389
12	1.630147059	1.478676471
24	1.221311475	1.24863388
36	1.439931958	1.086115246
48	1.309859155	1.165626671
96	1.295723311	1.144257249
144	1.350779128	1.119027274
192	1.266157109	1.085917804



N=144时cuts变化观察NF和Sky的近似率

cuts	appro-NF	appro-Sky
3	1.351	1.158
5	1.337	1.165
7	1.349	1.091
11	1.303	1.109
15	1.337	1.16
17	1.332	1.215
23	1.37	1.1
47	1.21	1.137



分析

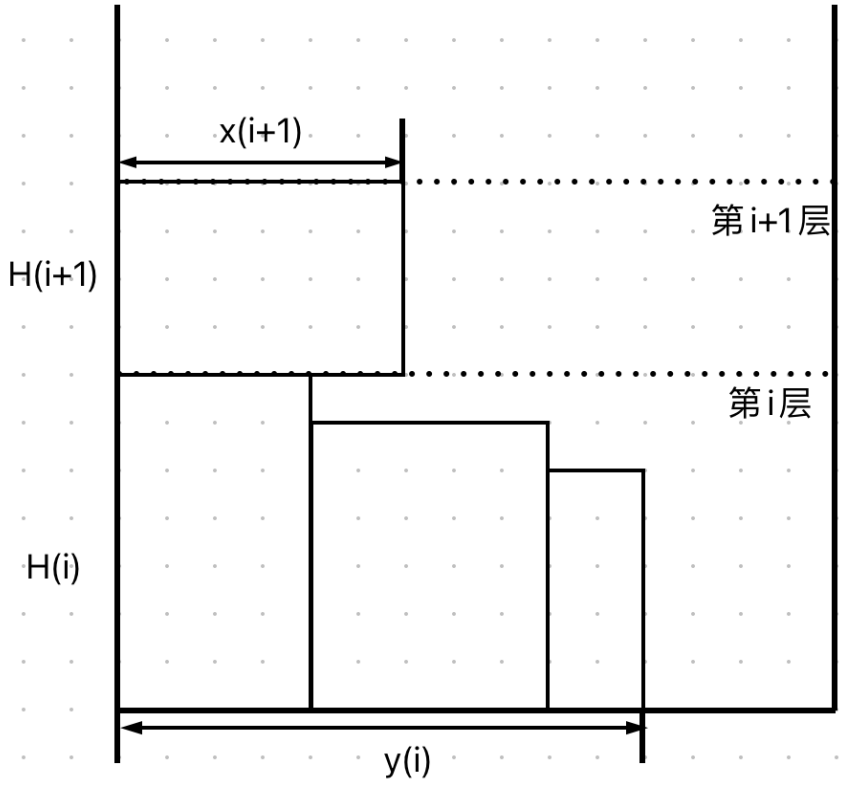
- NF的近似度虽然是2，测试中大多数情况小于2，在1.3左右
  Sky相较于NF总体上近似率要低，但在N较小的时候可能会出现没有NF好，分析可能是因为Sky为online算法，而且按照我们的方式产生的小矩形在初始几刀切出来的长宽的期望比较大，在N较小的时候有更大可能产生了更多的废图
- 另外，当cuts一定的时候，也就是产生的矩形的矩形长宽下限一定的时候，在N较大的时候NF和Sky的近似率基本没有太大变化，大概率是因为多个个大矩形切割产生小矩形在cuts数一定时，概率分布是一定的，N的增大并没有改变这一分布情况
- 当N一定的时候，随着cuts的增大，也就是小矩形的最小长宽期望下限不断减小（小矩形的细碎程度不断增大），NF和Skyline的近似率基本没有太大变化的原因可能是因为这种数据的不断减小的最小长宽期望下限对高度的提升影响程度在不断减小，cuts越多，多产生的小矩形的期望长宽实际上是越小的，对高度提升的影响力也越小

Chapter4:Algorithm analysis

4.1 NFDH-Next Fit Decrease Height

- 时间复杂度
  - 利用stl中sort函数将箱子从高到低排序$O(NlogN)$
    - sort源码实现的原理是混合排序：先利用快速排序，当排序数目少于一定量的时候，再利用插入排序（当排序数目少到一定数目，插入排序的效率比较高），因此stl中sort的时间复杂度为$O(NlogN)$
  - 将箱子一个一个插入容器$O(N)$
- 空间复杂度
  - 保存所有箱子$O(N)$

- Approximation ratio=2



  - H_i为第i层的高度
  - y_i为第i层所放置物体的总宽度
  - x_i为第i层第一个物体的宽度
  - A_i为第i层所有物体的总面积和
  - A为所有物体的总面积和
  - $y_i + x_{i+1} > 1$推出$H_{i+1}(y_i + x_{i+1}) > H_{i+1}$
  - $H_{i+1}y_i \leq A_i$和$H_{i+1}x_{i+1} \leq A_{i+1}$推出$H_{i+1}(x_{i+1} + y_i) \leq A_{i+1} + A_i$
  - 由上面两式得到$H_{i+1} \leq A_{i+1} + A_i$
  - 因此总共的高度$< H_1 + 2A$
  - 而$OPT(L) \geq A$
  - 所以$NFDH(L) <= 2 * OPT(L) + h_{max}$

4.2 SkyLine

- 时间复杂度
  - 对于n中每一个online输入的小矩形，比较评分、旋转、合并等的操作最多进行$O(N)$次，因此时间复杂度为$O(N^2)$
- 空间复杂度$O(N)$

    - 只需要存储n个矩形的相关信息（几何位置、宽高）和不超过n条的天际线的相关信息（几何位置，宽度），因此空间复杂度为$O(N)$

- Approximation ratio
    - 从测试结果来看，此算法大致的近似率为1.2左右，但是实际的近似率应该会比这个高

Chapter5：Declaration

“PROJECT6” 中的所有工作均由本组组员独立完成。

附录：code

- NFDH

```
#include <algorithm>
#include <iostream>
#include <fstream>
#include<ctime>
#include <chrono>
using namespace std;

#define NoLimit 100000//定义一开始的货架的上限为无穷

struct Bin{//定义箱子的结构
    double x;//箱子放置的x坐标
    double y;//箱子放置的y坐标
    double height;//箱子的高度
    double width;//箱子的宽度

    friend bool operator < (const Bin& i1, const Bin& i2){
        return i1.height>i2.height;
    }
};

int main(){
    //打开文件
    ifstream file;
    file.open("test.txt", ios::in);
    if(file.is_open()) cout << "文件打开成功" << endl;
    else cout << "文件打开失败" << endl;

    //将文件中的内容读取进来
    int i,n; file>>n;//读取箱子总数
    double optH;
    struct Bin bin[n];
```

```
    for(i=0;i<n;i++){//读取每个箱子的宽度和高度
        file>>bin[i].height>>bin[i].width;
    }
    file>>optH;
    file.close();//关闭文件

    clock_t startTime,endTime;
    startTime = clock();//计时开始

    sort(bin,bin+n);//离线，将所有的箱子的高度按照从高到低排列

    double this_width,this_height;//当前插入的这个箱子的高度和宽度
    double allWidth = 1;//当前货架的宽度，定为1
    double allHeight = 0;//货架的总高度，初始化为0
    double currentX = 0,currentY = 0;;//当前这一层货架右部分所剩空间的左下角坐标，初始为（0，0）
    double currentShelfHeight=NoLimit;//当前货架的限高：如果当前货架上没有箱子，那么限高为NoLimit
                                                //如果有箱子，那么限高为第一个箱子也就是最高的箱子的高度

    for(i=0;i<n;i++){
        this_width=bin[i].width;//赋值
        this_height=bin[i].height;//赋值
        if(this_width>allWidth-currentX||this_height>currentShelfHeight){
            //当这一行货架的剩余宽度小于插入箱子的宽度 或者 这一行货架的剩余高度小于插入箱子的高度 的时候
            //就将新插入的箱子放在下一层
            bin[i].x=0;bin[i].y=currentShelfHeight;//保存插入的箱子的左下角坐标
            currentY+=currentShelfHeight;//更新
            currentX=this_width;//更新
            currentShelfHeight=this_height;//更新
        }else{
            //可以放在这一层货架
            bin[i].x=currentX;bin[i].y=currentY;//保存插入的箱子的左下角坐标
            currentX=currentX+this_width;//更新
            //当这一层货架没有物品的时候，将这一层货架的限高设置为自己的高度
            if(currentShelfHeight==NoLimit)currentShelfHeight=this_height;
        }
    }

    endTime = clock();
```

```
    //cout << "The run time is: " <<(double)(endTime - startTime)  << "ms"
<< endl;

    allHeight=currentY+currentShelfHeight;//更新货架总高度
    cout<<"最优高度:"<<optH<<endl;
    cout<<"总共高度:"<<allHeight<<endl;
    cout<<"ApRatio"<<allHeight/optH<<endl;
    //输出每个箱子的（x坐标，y坐标）：箱子宽度 箱子高度
    /*for(i=0;i<n;i++){
        cout<<i+1<<":("<<bin[i].x<<","<<bin[i].y<<"):"<<bin[i].width<<" "
<<bin[i].height<<endl;
    }*/

    return 0;
}
```

- 生成数据算法

```
#include <algorithm>
#include <iostream>
#include <fstream>
#include <time.h>
#define cuts 5
using namespace std;

struct Node{
    double height;
    double width;
};

int main(){
    int i,n;
    double allHeight=0;//最终输出的总高度
    cout<<"输入n必须为"<<cuts+1<<"的倍数:";
    cin>>n;

    struct Node bin[n];

    srand((unsigned)time(NULL));
    double last_width=1;//维护一个需要切割的矩形，初始宽度为1
    double last_height=0;//维护一个需要切割的矩形，高度需要随机生成

    double h;
```

```
    i=0;
    int times=cuts;//初始化times为cuts，使得一开始就需要重新生成一个随机矩形
    while(i<n){
        if(times==cuts){//当切割五次后需要重新生成矩形
            if(i!=0){//除了第一次，其他情况需要将切割后剩余的矩形也保存
                bin[i].height=last_height;
                bin[i].width=last_width;
                i++;
                if(i==n)break;
            }

            times=0;//置为0
            last_height=(double)rand()/RAND_MAX;//随机生成高度
            while(last_height<0.5)last_height=(double)rand()/RAND_MAX;//使得高度必须>0.5
            last_width=1;
            allHeight+=last_height;//更新高度
        }

        h=(double)rand()/RAND_MAX;//随机生成一个切割的比例
        //cout<<times<<":h="<<h<<endl;

        if(times%2==0){//第一次竖切，横切竖切依次进行
            if((double)rand()/RAND_MAX>0.5){//随机选择保留的部分
                //写入矩形数组
                bin[i].height=last_height;
                bin[i].width=last_width*(1-h);
                //再维护剩余的高度和宽度
                last_width=last_width*h;
            }else{
                //写入矩形数组
                bin[i].height=last_height;
                bin[i].width=last_width*h;
                //再维护剩余的高度和宽度
                last_width=last_width*(1-h);
            }
        }else{
            if((double)rand()/RAND_MAX>0.5){//随机选择保留的部分
                //写入矩形数组
                bin[i].width=last_width;
                bin[i].height=last_height*(1-h);
                //再维护剩余的高度和宽度
                last_height=last_height*h;
```

```
            }else{
                //写入矩形数组
                bin[i].width=last_width;
                bin[i].height=last_height*h;
                //再维护剩余的高度和宽度
                last_height=last_height*(1-h);
            }
        }


        //如果需要每个箱子width<height
//          for(int j=i;j<i+12;j++){
//
if(bin[j].width>bin[j].height)swap(bin[j].width,bin[j].height);
//          }
        //随机翻转每个箱子
        if((double)rand()/RAND_MAX>0.5)swap(bin[i].width,bin[i].height);

        i++;
        times++;
    }
    //写入文件
    ofstream ofs;
    ofs.open("test.txt", ios::out);
    ofs<<n<<endl;

    for(i=0;i<n;i++){
        ofs<<bin[i].height<<" "<<bin[i].width<<endl;
        //cout<<bin[i].height<<" "<<bin[i].width<<endl;
    }
    ofs<<allHeight<<endl;
    return 0;
}
```

- SkyLine

```
from greedypacker import skyline,item,guillotine
import sys
import math
import time
S=skyline.Skyline(100000,sys.maxsize,heuristic='bottom_left') #尽量放在最左下角
#start=time.time()
with open ('test.txt','r') as file:
    n=int(file.readline())
```

```
    for _ in range(n):
        line = file.readline().strip()  #去掉换行符
        h,w = map(float,line.split())    #取出小矩形的高度和宽度
        h=math.floor(10000*h)     #转成整型，10000是减小约去的误差
        w=math.floor(10000*w)
        S.insert(item.Item(w,h))     #online算法
    optH=math.ceil(10000*float(file.readline()))#转成整型，10000是减小约去的误差
#end=time.time()

#print(optH)
#print(S.filledheight)
a=(float)(S.filledheight)    #填充的高度
b=(float)(optH)       #最优的高度
print(a/b)
#print(1000*(end-start))
```

- Skyline的函数类文件请见code文件夹