به نام خدا

علی غفاری ثابت - ۹۵۲۳۹۰۳

روش اول

توضیح کد:

۱- ابتدا تابع min_max را تعریف می کنیم. پارامتر های ورودی این تابع شامل لیستی از لیست ها، اندیس شروع و پایان و یک index است که برای مشخص کردن معیار پیدا کردن کوچکترین و بزرگترین عدد (x یا y) است. اگر بازه داده شده در ورودی شامل یک نقطه باشد یک زوج مرتب از ۲ تا x یا y (بسته به مقدار index) را بر می گرداند و اگر بازه داده شده شامل ۲ نقطه باشد x یا y آن ۲ نقطه را با یک عملیات مقایسه مرتب کرده و آن را بر می گرداند. در غیر این دو صورت بازه داده شده را از وسط به دو قسمت تقسیم می کند و max و min هر دو قسمت را پیدا می کند و از بین min های دو قسمت کوچکترین و از بین max های دو قسمت بزرگترین را به ترتیب گفته شده در قالب یک زوج مرتب بر می گرداند.

۲- سپس عدد n را از ورودی خوانده و تعداد n نقطه را از کاربر دریافت می کنیم.

۳- مقادیر minx و maxx را با index=0 و miny و maxy را با index=1 به کمک تابع min_max پیدا می کنیم و به عنوان معادله خطوط تشکیل دهنده کوچکترین مستطیل محدود کننده چاپ می کنیم.

پیچیدگی زمانی:

در تابع min_max مسئله هر بار به دو مسئله کوچکتر با طول $n/2$ تقسیم می شود و هر بار $O(1)$ عملیات انجام می شود پس پیچیدگی زمانی تابع min_max برابر است با:

$$T(n) = 2T(n/2) + c = \sum_{i=0}^{lg(n)-1} c2^i = c(2^{lg(n)} - 1) = c(n-1) = \theta(n)$$

پیچیدگی زمانی خواندن مختصات نقاط از ورودی نیز برابر $\theta(n)$ است. بنابراین پیچیدگی زمانی کل برنامه برابر $\theta(n)$ است.

پیچیدگی حافظه:

پیچیدگی حافظه به جز لیستی که برای ذخیره مختصات نقاط استفاده می شود و پیچیدگی حافظه آن $\theta(n)$ پیچیدگی حافظه سایر متغیر ها برابر $\theta(1)$ است. بنابراین پیچیدگی حافظه برنامه (بدون در نظر گرفتن حافظه stack که برای تابع بازگشتی min_max استفاده می شود.) برابر $\theta(n)$ است.

پیچیدگی حافظه stack نیز برابر ارتفاع درخت بازگشتی تابع min_max یعنی $\theta(lg(n))$ است.

تعداد عملیات مقایسه:

تعداد عملیات مقایسه نیز برای اعدادی که از توان ۲ هستن برابر است با: $(f(2) = 1)$

$$f(n) = 2f(n/2) + 2 = 1 * n/2 + \sum_{i=0}^{lg(n)-2} 2 * 2^i = 2 * (2^{lg(n)-1} - 1) + n/2 = 2 * (n/2 - 1) + n/2 = 3n/2 - 2$$

روش دوم

توضیح کد:

۱- ابتدا تابع max_min را تعریف می کنیم که دو عدد را به عنوان ورودی دریافت کرده و سپس این دو عدد را با انجام یک مقایسه به صورت مرتب شده در قالب یک زوج مرتب بر می گرداند.

۲- سپس عدد صحیح n را که نشان گر تعداد نقاط در صفحه است را از ورودی می خوانیم و سپس مختصات n نقطه را از ورودی دریافت و در یک لیست به نام vertices ذخیره می کنیم.

۳- x ها و y های دو نقطه اول را مرتب می کنیم و متغیر های minx, miny, maxx, maxy را مقدار دهی می کنیم.

۴- به کمک یک حلقه هر بار x, y دو نقطه بعدی را مرتب کرده و به ترتیب در tempx, tempy ذخیره می کنیم و عدد کوچکتر در هر یک را با مینیمم ها و اعداد بزرگتر را با ماکسیمم ها مقایسه کرده و با توجه به حاصل مقایسه در صورت نیاز مقدار ماکسیمم ها و مینیمم ها تغییر می کند. (مثلا اگر x بزرگتر از maxx بزرگتر بود maxx را برابر x قرار می دهیم.)

۵- در صورتی که n فرد باشد برای x, y راس آخر نیز این عملیات را انجام می دهیم. در این مرحله چون تنها یک نقطه باقی مانده است نیاز به مرتب کردن x, y نیست.

۶- در نهایت حاصل را به صورت معادله خطوط تشکیل دهنده کوچکترین مستطیل محدود کننده را چاپ می کنیم.

- تعداد نقاط ورودی (n) باید بیشتر از ۱ باشد زیرا آنگاه کوچکترین مستطیل احاطه گر معنا ندارد.

تحلیل زمانی:

پیچیدگی زمانی خواندن نقاط از ورودی و تک حلقه for $\theta(n)$ است و پیچیدگی سایر قسمت ها $\theta(1)$ است. در نتیجه پیچیدگی زمانی کل برنامه $\theta(n)$ است.

تحلیل حافظه:

به جز لیستی که برای دریافت نقاط استفاده می شود و پیچیدگی حافظه $\theta(n)$ دارد سایر اجزا و متغیر های برنامه پیچیدگی حافظه $\theta(1)$ دارند.