分治

目录

CDQ分治

- 将所有问题分成左右两边，然后计算左边对右边的贡献或者右边对左边的贡献。

- 在线 -> 离线

例题

- D维数点 / 单点加矩形求和
- 加点(x,y)，查询ax+by最大值

CF 848C

- 给一个数列，定义一段的权值为每个不同的数值最后一次出现的位置减去第一次出现的位置和。

- 要求单点修改/求某一段的权值和。

UOJ 50 链式反应

- 求解微分方程f’(x)=p(x) f(x)^2 + 1幂级数的前n项。
- 也就是n * f[n]=sum 0<=i+j<n p[n-i-j-1] * f[i] * f[j]

线段树分治

- 加边/删边/求连通性 离线
- 01向量 加一个/删一个/求线性基
- 对每个物品求不包含它的背包
- 对所有的i,j,k求i到j不经过k的方案数

HNOI 城市建设

- 支持修改边权，求MST的和

Solution

- 给一个边集S，首先把S中的边设成+inf，然后删去不在MST中的边。
- 然后把S中的边设成-inf，然后把MST/S中的边缩起来。

二进制分组

- 解决强制在线问题 / 每次只能在末尾加
- 二项堆
- 加点(x,y)，查询ax+by最大值 (强制在线版本)

UOJ Unknown

- 有一个向量序列。
- 支持末尾加入一个元素 / 末尾删除一个元素 / 求[l,r]这一段与(x,y)最大的点积。

- Solution 0: 只有插入直接二进制分组
- Solution 1: 把版本树建出来，然后考虑点分治。
- 强制在线的一些想法：
  - 1. 重量平衡/Treap 每个点随机一个权值，如果比上一个块最小值小就合并
  - 2. 加buffer 建立线段树的结构，每一层如果最后两个节点都坏了，那么重构。
- 注意：我们可以把一些树上的问题进行这样的转化。

整体二分

- n*n的矩形Q次查询子矩形第k大的数
- 单点修改，区间第k大
- 区间整体插入一个数，求区间k大。

CF 102354 B

- 给a1, a2,…, an, b1, b2, …, bn 为 1到n的排列
- 求c[k]=max(|ai-bj|, gcd(i,j)=k)
- n<=10^5

决策单调性

CF 321E

- 你有一个n*n对称矩阵表示不熟悉程度。
- 你要将它1~n按顺序分成k组，每组的代价为两两之间不熟悉程度的和。
- n 4000 k 800

NOI 2009 诗人小G

- 有N个诗句需要被排版为若干行，顺序不能改变。
- 一行内可以有若干个诗句，相邻诗句之间有一个空格。
- 定义行标准长度L，每行的不协调度为|实际长度-L|^P，整首诗的不协调度就是每行不协调度之和。
- 任务是安排一种排版方案，使得整首诗的不协调度最小。
- P=1, 2, 3

CF 868F

- 将一个数列分成k段，每段的代价是相同的数字的对数。
- 求代价总和最小。

IOI 2013

- 题意大概是，有r*c的网格图，每次只能朝右下两个方向。
- 修改一条边的边权，然后查询某两点之间最短路。
- 要求时间复杂度O(c^2 logr log c)

IOI14 holiday

- 有n个城市，每个城市有一个权值，起点在s。
- 每一天你可以往左或者往右走一步，或者选择游览这个城市。
- 问d天能获得的最大权值和是多少?

带权二分

12互测 Tree

- 给你一个无向带权连通图，每条边是黑色或白色。让你求一棵最小权的恰好有need条白色边的生成树。

邮局(链)

- 一条路有n个村庄，你要建k个邮局，每个邮局到最近村庄距离最小。

Rikka with Matching

- n*m的网格，有边权，求k对匹配的最小和。
- n<=5

2018联考 林克卡特树

- 有一棵树，你要切掉k条边，使得每部分直径加起来最大。

广义的单调性

- Circular Edit distance

Yuhao Du Contest 5 L

- 加边，求互相可达点对。

Yuhao Du Contest 5 I

- 邮局(环版本)

ZROJ 799 最短路

- 给你一个网格图。你可以往上下和右边走。
- 问对于第一列的每个点，到最后一列的点的距离和是多少。
- WH<=2e5

树分治

- 一般的图分治，是找到一个很小的集合，能将整个图分成均匀的几部分。
- 对于树，我们可以选择点分治/边分治/链分治。
- 边分治 - 3度化

点分治

- 求距离不超过k的点对。
- 统计有多少对点对之间构成了括号序列。
- 求距离为1, 2, …, n的点对。
- [SDOI]模式字符串 给一个模板串，求多少对点之间路径是t的重复。
- 树上的背包问题(连通子树/不相邻)。

点分树

- ZJOI 2007 Hide 修改一个点的黑白，求最远黑点距离。
- 单点修改，求距离x不超过r的点权和。
- ZJOI 2015 点权修改，求树的重心。

杂题

- LOJ 6145 给一棵树，支持查询 min i=l…r dis(i,x)
- CF 757G 给一棵树和排列p[i]，支持查询sum i=l..r dis(p[i],x)，支持修改交换p[x],p[x+1]，强制在线
- CF 936E 给一个n个格子的4连通图形，保证格子的补集也是4连通的，支持插入一个点，查询最近点。

杂题

- CSAcademy Round 10 Yury's Tree 给一棵带权树，要求支持查询一个点的价值，修改(x,y,z) 对于x子树的所有点u如果u到x的路径上的边权全都大于等于y，那么u价值加z。

- Luogu P5311 给定一棵n个点的树，每个节点有一个颜色。q询问从x出发，只经过编号在[l,r]中的点，所能到达的点的颜色种数。

- 给出一棵n个点的有根树，每个结点上有一个一次多项式。求每个结点到根的多项式乘积的和。

杂题

- 给两棵树T1, T2，求sum 1<=i,j<=n dis1(i,j)*dis2(i,j)
- CF 1010F 求包含根节点的大小为1, 2, …, n的连通子树的个数。

动态点分治

- 重量平衡树：
  - 带插入区间第k大，替罪羊树/Treap。
  - O(log n)插入，O(1)比较大小。
- 紫荆花之恋，支持插入叶子，维护点分树结构。
- 一个森林，支持加边，维护点分树结构。

平面图分治

- 对于平面图，由separator theorem，可以找到大小为O(sqrt(n))的点集，将图分成不超过2/3的部分。
- 由于问题比较复杂，所以我们一般碰到的是网格图或者outplanar graph。

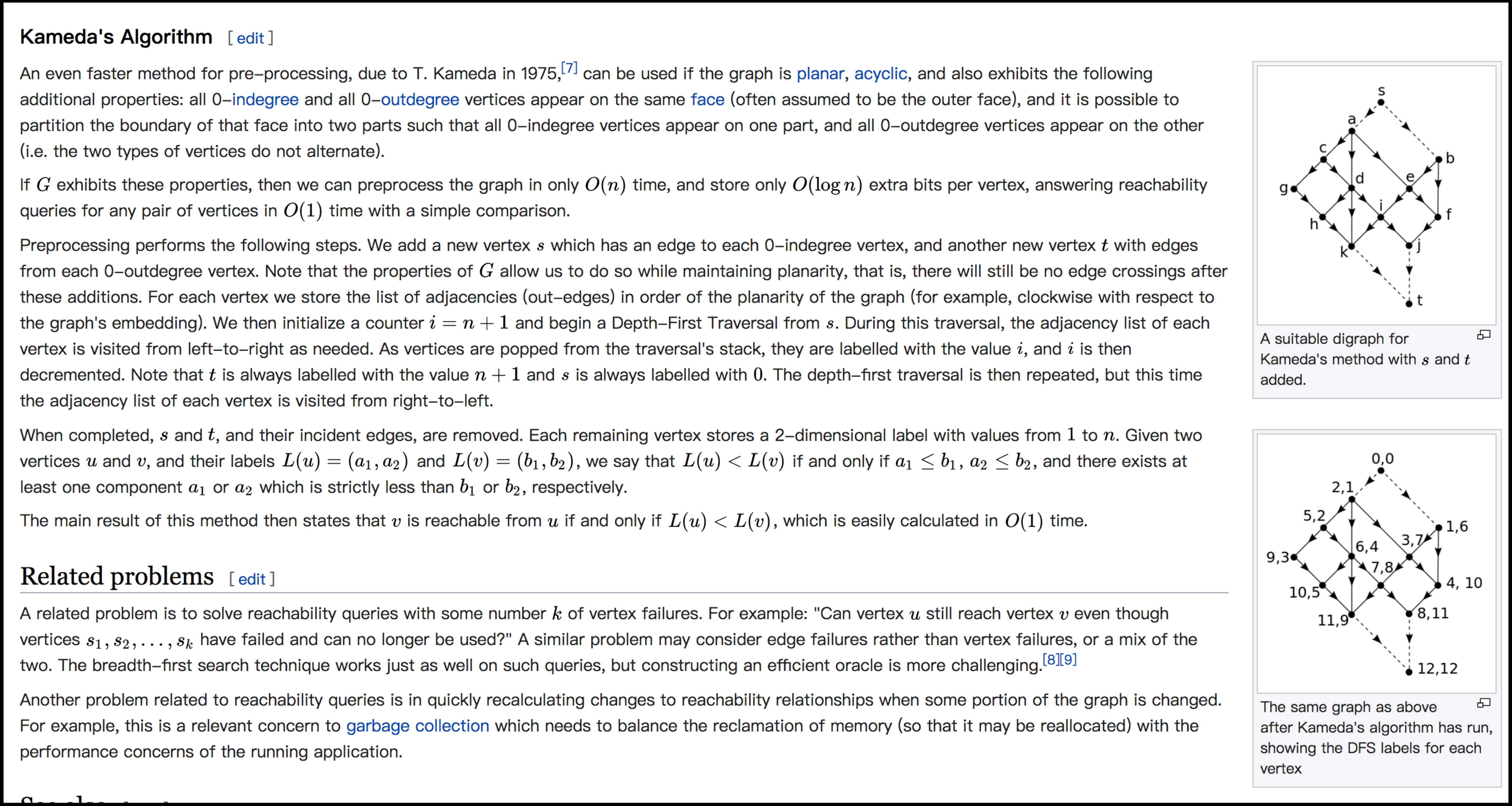
最短路

- outplanar graph: 对偶树分治
- 网格图：找短边分治

可达性

- CF 232E
- 给你一个n*m的网格图，每次只能往右或者往下走，q个询问两点之间之间是否可达。
- 注意可以做到O(nm log nm) - O(1)，自行查阅相关文献。

st planar graph



Kameda's Algorithm [edit]

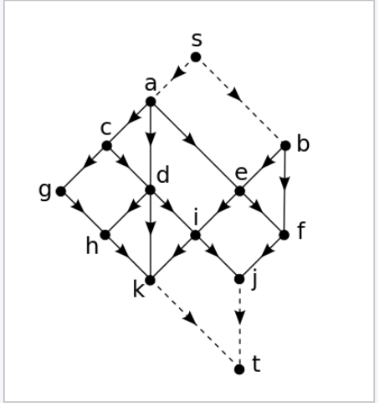
An even faster method for pre-processing, due to T. Kameda in 1975,[7] can be used if the graph is planar, acyclic, and also exhibits the following additional properties: all 0-indegree and all 0-outdegree vertices appear on the same face (often assumed to be the outer face), and it is possible to partition the boundary of that face into two parts such that all 0-indegree vertices appear on one part, and all 0-outdegree vertices appear on the other (i.e. the two types of vertices do not alternate).

If G exhibits these properties, then we can preprocess the graph in only $O(n)$ time, and store only $O(\log n)$ extra bits per vertex, answering reachability queries for any pair of vertices in $O(1)$ time with a simple comparison.

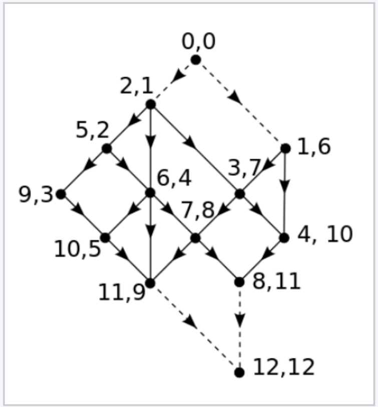
Preprocessing performs the following steps. We add a new vertex s which has an edge to each 0-indegree vertex, and another new vertex t with edges from each 0-outdegree vertex. Note that the properties of G allow us to do so while maintaining planarity, that is, there will still be no edge crossings after these additions. For each vertex we store the list of adjacencies (out-edges) in order of the planarity of the graph (for example, clockwise with respect to the graph's embedding). We then initialize a counter $i = n + 1$ and begin a Depth-First Traversal from s. During this traversal, the adjacency list of each vertex is visited from left-to-right as needed. As vertices are popped from the traversal's stack, they are labelled with the value i, and i is then decremented. Note that t is always labelled with the value $n + 1$ and s is always labelled with 0. The depth-first traversal is then repeated, but this time the adjacency list of each vertex is visited from right-to-left.

When completed, s and t, and their incident edges, are removed. Each remaining vertex stores a 2-dimensional label with values from 1 to n. Given two vertices u and v, and their labels $L(u) = (a_1, a_2)$ and $L(v) = (b_1, b_2)$, we say that $L(u) < L(v)$ if and only if $a_1 \leq b_1$, $a_2 \leq b_2$, and there exists at least one component a_1 or a_2 which is strictly less than b_1 or b_2, respectively.

The main result of this method then states that v is reachable from u if and only if $L(u) < L(v)$, which is easily calculated in $O(1)$ time.



A suitable digraph for Kameda's method with s and t added.



The same graph as above after Kameda's algorithm has run, showing the DFS labels for each vertex

Related problems [edit]

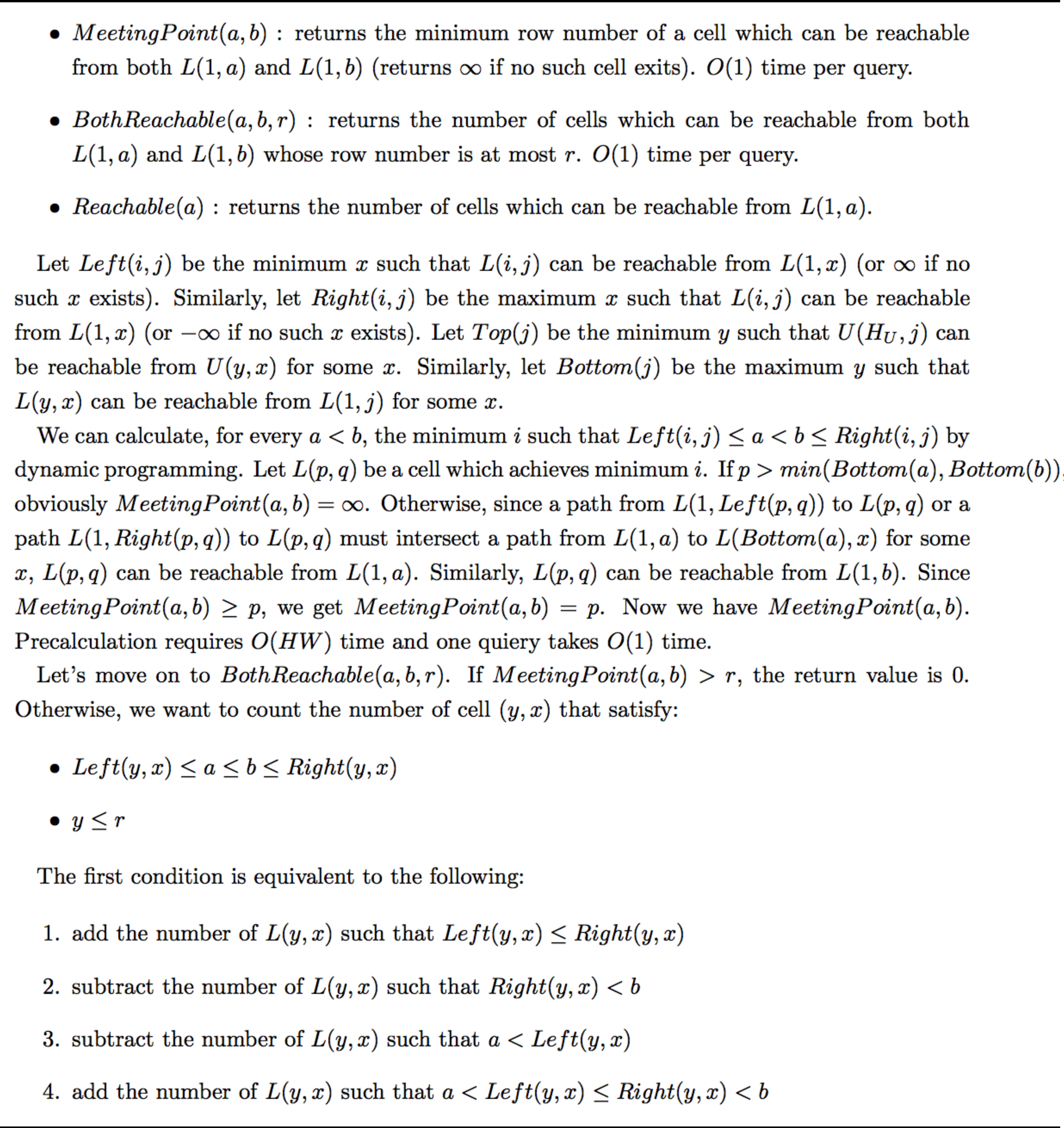
A related problem is to solve reachability queries with some number k of vertex failures. For example: "Can vertex u still reach vertex v even though vertices $s_1, s_2, \ldots, s_k$ have failed and can no longer be used?" A similar problem may consider edge failures rather than vertex failures, or a mix of the two. The breadth-first search technique works just as well on such queries, but constructing an efficient oracle is more challenging.[8][9]

Another problem related to reachability queries is in quickly recalculating changes to reachability relationships when some portion of the graph is changed. For example, this is a relevant concern to garbage collection which needs to balance the reclamation of memory (so that it may be reallocated) with the performance concerns of the running application.

AGC 28 Reachable Cells

- 给一个n*m的网格图，求可达的点对的权值和。

贴点题解防止忘了咋做



- $MeetingPoint(a,b)$: returns the minimum row number of a cell which can be reachable from both $L(1,a)$ and $L(1,b)$ (returns ∞ if no such cell exits). $O(1)$ time per query.
- $BothReachable(a,b,r)$: returns the number of cells which can be reachable from both $L(1,a)$ and $L(1,b)$ whose row number is at most r. $O(1)$ time per query.
- $Reachable(a)$: returns the number of cells which can be reachable from $L(1,a)$.

Let $Left(i,j)$ be the minimum x such that $L(i,j)$ can be reachable from $L(1,x)$ (or ∞ if no such x exists). Similarly, let $Right(i,j)$ be the maximum x such that $L(i,j)$ can be reachable from $L(1,x)$ (or $-\infty$ if no such x exists). Let $Top(j)$ be the minimum y such that $U(H_U,j)$ can be reachable from $U(y,x)$ for some x. Similarly, let $Bottom(j)$ be the maximum y such that $L(y,x)$ can be reachable from $L(1,j)$ for some x.

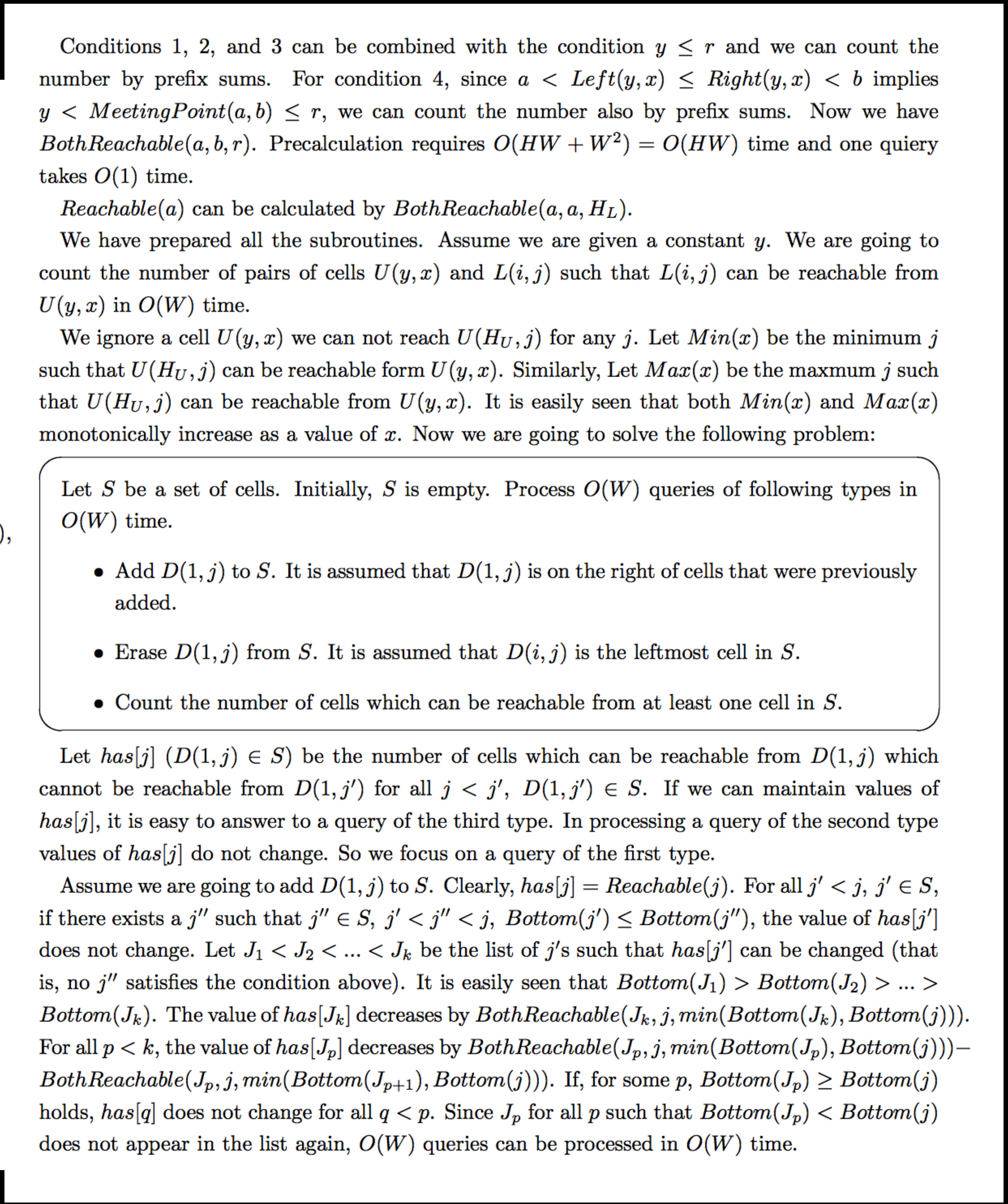
We can calculate, for every $a<b$, the minimum i such that $Left(i,j) \le a < b \le Right(i,j)$ by dynamic programming. Let $L(p,q)$ be a cell which achieves minimum i. If $p > min(Bottom(a), Bottom(b))$, obviously $MeetingPoint(a,b)=\infty$. Otherwise, since a path from $L(1, Left(p,q))$ to $L(p,q)$ or a path $L(1, Right(p,q))$ to $L(p,q)$ must intersect a path from $L(1,a)$ to $L(Bottom(a),x)$ for some x, $L(p,q)$ can be reachable from $L(1,a)$. Similarly, $L(p,q)$ can be reachable from $L(1,b)$. Since $MeetingPoint(a,b) \ge p$, we get $MeetingPoint(a,b)=p$. Now we have $MeetingPoint(a,b)$. Precalculation requires $O(HW)$ time and one quiery takes $O(1)$ time.

Let's move on to $BothReachable(a,b,r)$. If $MeetingPoint(a,b) > r$, the return value is 0. Otherwise, we want to count the number of cell (y,x) that satisfy:

- $Left(y,x) \le a \le b \le Right(y,x)$
- $y \le r$

The first condition is equivalent to the following:

1. add the number of $L(y,x)$ such that $Left(y,x) \le Right(y,x)$
2. subtract the number of $L(y,x)$ such that $Right(y,x) < b$
3. subtract the number of $L(y,x)$ such that $a < Left(y,x)$
4. add the number of $L(y,x)$ such that $a < Left(y,x) \le Right(y,x) < b$



Conditions 1, 2, and 3 can be combined with the condition $y \le r$ and we can count the number by prefix sums. For condition 4, since $a < Left(y,x) \le Right(y,x) < b$ implies $y < MeetingPoint(a,b) \le r$, we can count the number also by prefix sums. Now we have $BothReachable(a,b,r)$. Precalculation requires $O(HW + W^2) = O(HW)$ time and one quiery takes $O(1)$ time.

$Reachable(a)$ can be calculated by $BothReachable(a,a,H_L)$.

We have prepared all the subroutines. Assume we are given a constant y. We are going to count the number of pairs of cells $U(y,x)$ and $L(i,j)$ such that $L(i,j)$ can be reachable from $U(y,x)$ in $O(W)$ time.

We ignore a cell $U(y,x)$ we can not reach $U(H_U,j)$ for any j. Let $Min(x)$ be the minimum j such that $U(H_U,j)$ can be reachable form $U(y,x)$. Similarly, Let $Max(x)$ be the maxmum j such that $U(H_U,j)$ can be reachable from $U(y,x)$. It is easily seen that both $Min(x)$ and $Max(x)$ monotonically increase as a value of x. Now we are going to solve the following problem:

> Let S be a set of cells. Initially, S is empty. Process $O(W)$ queries of following types in $O(W)$ time.
>
> - Add $D(1,j)$ to S. It is assumed that $D(1,j)$ is on the right of cells that were previously added.
> - Erase $D(1,j)$ from S. It is assumed that $D(i,j)$ is the leftmost cell in S.
> - Count the number of cells which can be reachable from at least one cell in S.

Let $has[j]$ $(D(1,j) \in S)$ be the number of cells which can be reachable from $D(1,j)$ which cannot be reachable from $D(1,j')$ for all $j < j'$, $D(1,j') \in S$. If we can maintain values of $has[j]$, it is easy to answer to a query of the third type. In processing a query of the second type values of $has[j]$ do not change. So we focus on a query of the first type.

Assume we are going to add $D(1,j)$ to S. Clearly, $has[j] = Reachable(j)$. For all $j' < j$, $j' \in S$, if there exists a j'' such that $j'' \in S$, $j' < j'' < j$, $Bottom(j') \le Bottom(j'')$, the value of $has[j']$ does not change. Let $J_1 < J_2 < ... < J_k$ be the list of j's such that $has[j']$ can be changed (that is, no j'' satisfies the condition above). It is easily seen that $Bottom(J_1) > Bottom(J_2) > ... > Bottom(J_k)$. The value of $has[J_k]$ decreases by $BothReachable(J_k, j, min(Bottom(J_k), Bottom(j)))$. For all $p < k$, the value of $has[J_p]$ decreases by $BothReachable(J_p, j, min(Bottom(J_p), Bottom(j))) - BothReachable(J_p, j, min(Bottom(J_{p+1}), Bottom(j)))$. If, for some p, $Bottom(J_p) \ge Bottom(j)$ holds, $has[q]$ does not change for all $q < p$. Since J_p for all p such that $Bottom(J_p) < Bottom(j)$ does not appear in the list again, $O(W)$ queries can be processed in $O(W)$ time.

杂题

- n*m的网格，求1个数不超过r个的矩形。
- r<=12