MC558 - Projeto e análise de algoritmos II - Unicamp

Laboratório 5 - Galeria de arte

Erik Perillo, RA135582

5 de dezembro de 2016

1 Abordagem

O problema a se resolver é o de usar o mínimo número possível de guardas para cobrir regiões arbitrárias quadriculadas.

Pode-se reduzir o problema ao problema do emparelhamento máximo: Nós da bipartição são pontos onde deve-se ter um vigilante. Para cada dois pontos adjacentes, há um nó para cada lado da bipartição. As direções de adjacências garantem que o grafo resultante nunca tenha ciclos ímpares e, portanto, sempre há como montar uma bipartição e temos uma transformação injetora.

Achar um emparelhamento máximo nesse esquema é equivalente a achar o máximo número possível de quadrados que não há ninguém mas são vigiados por outro, o que dá então o número mínimo possível de guardas a se usar: é o número máximo (área da região) menos o número de quadrados poupados que conseguimos.

O pseudo algoritmo abaixo explica de forma abstraida a transformação: Primeiro da matriz das regiões para uma bipartição e depois para uma rede de fluxo.

Algorithm 1

```
1: procedure TRANSFORM(M)
2:     (S, T) ← bipartition(M)
3:     C ← s ∪ t
4:     for (x, y) in adjacencies(M) do
5:         (u, v) ← map_to_edge(M, x, y)
6:         if v ∈ S then
7:             swap(u, v)
8:         C ← C ∪ (s, u)
9:         C ← C ∪ (u, v)
10:        C ← C ∪ (v, t)
11:    G ← V(C)
12: return (G, C, s, t)
```

2 Complexidade

O grafo foi implementado como uma lista de adjacências. As regiões a serem vigiadas foram implementadas como uma matriz binária, com 1 se e só se o ponto (i, j) é uma região alfa. Seja V o número de vértices de um grafo e E o número de arestas do mesmo e l o número de linhas e c o número de colunas da matriz das regiões.

A linha 2 é simbólica pois na prática pode-se determinar qual o lado da bipartição de um nó só com valores booleanos. O *loop* da linha 4 é basicamente uma iteração sobre cada ponto da matriz, checando ao seu redor para adjacências. O mapeamento de um ponto para um nó da linha 5 é feito em tempo constante com uma matriz de mapeamento. As linhas 8 e 9 checam se as arestas já estão no grafo de capacidades antes de adicioná-las. Isso é feito em tempo constante com um vetor. Assim, a transformação leva tempo $O(lc)$. A rede retornada é a entrada do algoritmo `EDMONDS-KARP` que foi implementado como esperado com `BFS` e então leva tempo $O(VE^2)$ com V, E das capacidades. Para se obter a resposta do problema original a partir do fluxo máximo, simplesmente calcula-se: $n = |C| - 2 - |f|$, onde f, C são o fluxo e capacidade, respectivamente. O grafo de capacidades é construído de tal forma que ele tenha $lc + 2$ nós: pontos que não são adjacências simplesmente não se conectam a nada. Isso leva tempo constante. Assim, o tempo final do algoritmo é $O(lc + VE^2) = O(l^3c^3)$ pois $V, E = O(lc)$.