BU CLA CS 530: Analysis of Algorithms

Spring 1996

Handout 4

Parallel and Randomized Parallel Graph Algorithms

We consider two algorithms. The first algorithm, published in 1995, finds a maximal matching in a cubic graph on a CREW PRAM. It takes $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(n^3)$ processors. The second is a randomized algorithm to determine maximal independent sets (MIS) which are closely related to matchings. It runs on a EREW PRAM in $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(n+m)$ processors.

Throughout the handout $G = (V, E)$ denotes a graph where V are the vertices and E the edges. Furthermore $n = |V|$ is the number of vertices and $m = |E|$ the number of edges.

1 Maximal Matching in a Cubic Graph

1.1 Cubic Graphs

The algorithm operates on a subset of undirected graphs called cubic graphs. This notion is covered by Definition 1.

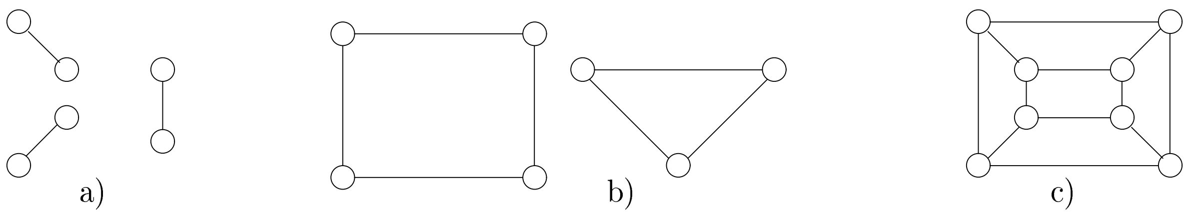
Definition 1 (Regular and Cubic Graphs) Let $G = (V, E)$ be an undirected graph.

(1) G is d-regular if every vertex $v \in V$ has exactly degree d.

(2) G is cubic if G is connected and G is 3-regular.

(3) G is at most cubic if G is connected and no vertex has degree > 3.

Example 2 Here are a few examples of d-regular graphs:



a) 1-regular graph; b) 2-regular graph; c) cubic graph (this one is actually called 'cube')

We see from the definitions that the number of edges is always linear in the number of vertices, *i.e.* $m = \mathcal{O}(n)$ for d-regular graphs. The following fact covers this relation more precisely.

Fact 3 *If G is d-regular then $m = d \cdot n/2$*

1.2 Matchings

We take a closer look at the notion of a matching. This corresponds to the definition on page 600 in [CLR].

Definition 4 (Matchings) Let $G = (V, E)$ be an undirected graph.

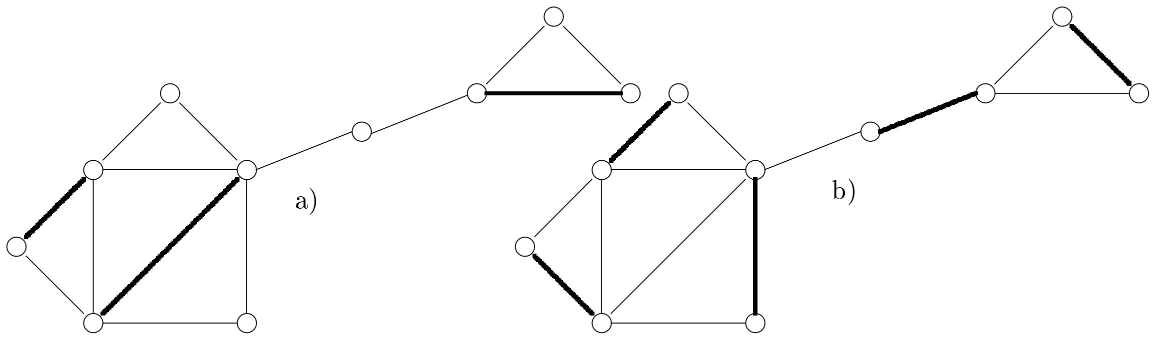
(1) $M \subseteq E$ is a matching if for every vertex $v \in V$ at most one edge in M is incident on v.

(2) $M \subseteq E$ is a maximal matching if M is a matching and for every matching $M' \supseteq M$ we have $M = M'$.

(3) $M \subseteq E$ is a maximum matching if M is a matching and for every matching M' we have $|M'| \leq |M|$.

(4) $M \subseteq E$ is a perfect matching if M is a matching and every vertex $v \in V$ is matched
(*i.e.* v is incident to an edge in M).

Example 5 Here are some examples of matchings.



a) maximal matching; b) perfect matching

Put differently, a matching is a subset of edges with disjoint adjacent vertices. We have to be careful to distinguish the notions of a maximal and a maximum matching. Intuitively we have a maximal matching if we can not add another edge to it without violating the disjoint vertices property. One could refer to this as a relative or local maximum. In contrast, a maximum matching is an absolute or global maximum, *i.e.* there is no other matching with more edges. All matchings are not necessarily unique, *e.g.* we could have more than one maximum matching in a given graph. The different types of matchings show an interesting property.

Lemma 6 *If M is a perfect matching in G then $|V|$ is even and M is a maximum matching. If M is a maximum matching then M is a maximal matching.*

The matchings can be ordered via $\subseteq$. In fact, if P_G is the class of perfect matchings over G, U_G the class of maximum matchings, A_G the class of maximal matchings and M_G the class of matchings we find:

$$P_G \subseteq U_G \subseteq A_G \subseteq M_G$$

Recall that if G is a bipartite graph we can find a maximum matching in $\mathcal{O}(VE)$ sequential time. This result is from [CLR] §27.3. The problem is reduced to a maxflow problem which is solved using the FORD-FULKERSON method. A more general result is stated by the following fact.

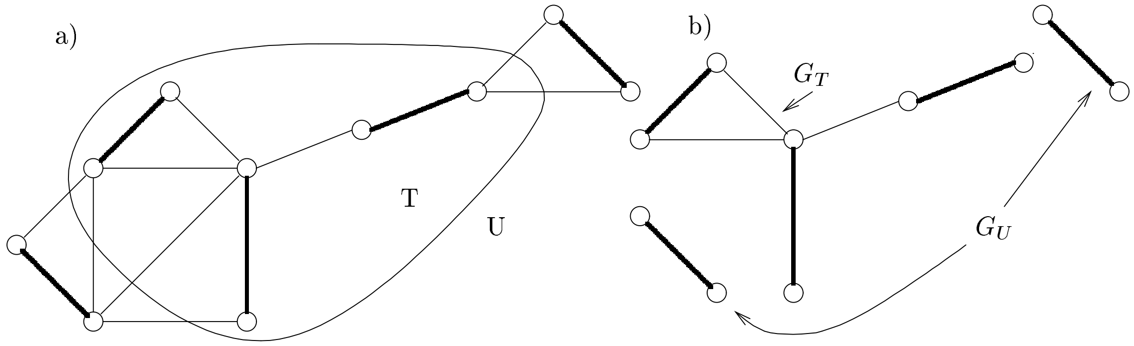
Fact 7 *There are sequential algorithms to construct a maximum matching in an arbitrary graph in polynomial time (e.g. $\mathcal{O}(V^2E)$).*

1.3 The PARALLEL-MAXIMAL-MATCHING Algorithm

The algorithm below makes use of the following lemma.

Lemma 8 *Let $G = (V, E)$ be an undirected graph and $T, U \subseteq V$ such that $T \cup U = V$ and $T \cap U = \varnothing$ (i.e. T, U is a partition of G). Now let $G_T = (T, E \cap (T \times T))$ and $G_U = (U, E \cap (U \times U))$ be the subgraphs induced by T and U.*
If M is a perfect matching in G_T and N is a maximum matching in G_U, then $M \cup N$ is a maximal matching in G.

Example 9 Here is the graph from Example 5 with a partition T, U.



a) Partitions T and U; b) Induced Subgraphs G_T and G_U

Lemma 8 states that we can easily construct a maximal matching if we have a perfect and a maximum matching in the induced subgraphs of a partition.

The time used by the algorithm below is mostly spent on finding a BFS tree in the graph. Lemma 10 gives an upper bound for this operation.

Lemma 10 *Let $G = (V, E)$ be a connected, undirected graph. A BFS tree of G can be constructed in $\mathcal{O}(\log^2 n)$ time using exactly n^3 processors on a CREW PRAM.*

Exercise 11 Prove Lemma 10. (Hint: Consider using matrix multiplication.)

We can do even better than the $\mathcal{O}(n^3)$ stated in Lemma 10.

Fact 12 *The number of processors needed depends on the matrix multiplication algorithm used. Today's best algorithm needs $\mathcal{O}(n^{2.376})$ processors [Coppersmith and Winogard 1990].*

We can now look at the CREW algorithm to find a maximal matching in a cubic graph. The steps described here use the methods discussed earlier in the various lemmas and facts. Therefore the description is very general.

PRALLEL-MAXIMAL-MATCHING(G)

1. Construct a BFS-tree B from the graph G.
2. Assign BFS-level numbers to the vertices.
3. Construct a maximal matching M of B in two stages:
   (a) One for the vertices in the even levels, and
   (b) one for the vertices on the odd levels.
4. Let U be the unmatched vertices in B. Construct a maximum matching M' in G_U.
5. Return $M \cup M'$.

1.4 Analysis

Correctness:

The BFS tree obtained in step 1 covers all vertices in V because G is a connected, undirected graph.
The BFS numbering in step 2 assigns 0 to the root and the corresponding depth to the other nodes.
In step 3 the algorithm picks an arbitrary edge from those available at each level. It is important to note that an even numbered vertex has only odd numbered neighbors and vice versa. Therefore the two stage construction in step 3 avoids a concurrent write to the data structure. Hence it is not possible to pair a vertex more than once.
Observe also that in step 3.(b) the only candidates left are those that connect to leaves in the BFS tree.
Now U are the unmatched vertices in V. This allows us to look at the partition T, U where $T = V - U$. As M pairs all the vertices in T it is a perfect matching in G_T. Lemma 13 gives us more information about U.

Lemma 13 *After the two stage matching in step 3, the unmatched vertices are either isolated or form a path or a cycle of unmatched vertices.*

Proof: In the remaining graph G_U each vertex has a degree at most two. With this premise it is not possible to have components in G_U other than isolated nodes, paths or cycles. ∎
Although the description of the algorithm mentions only cubic graphs is is easy to see that it can be applied unchanged to at most cubic graphs. We get this result from Lemma 13.
It is straight forward to construct a maximum matching M' with these low degree components.
With the perfect matching M over G_T and the maximum matching M' over G_U we finally get a maximal matching according to Lemma 8.

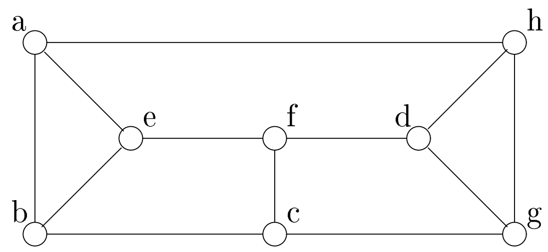
Time Estimate: If we look at the running time of the algorithm we find that the dominating part is the construction of the BFS-tree in step 1. According to Lemma 10 this can be done in $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(n^3)$ processors on a CREW PRAM.

Step 2 can be performed in $\mathcal{O}(\log n)$ time using a list ranking algorithm with pointer jumping with a processor for each node.

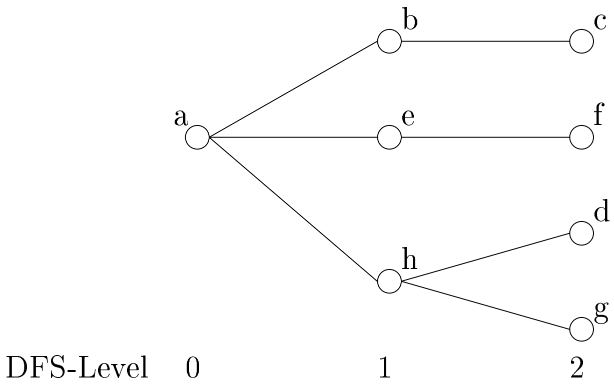
The obvious solution for step 3 takes only constant time as each node has it's own processor.
In Step 4 we construct a maximum matching in a graph consisting of isolated vertices, paths and/or cycles. This can be done in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n)$ processors on a EREW PRAM (again using listranking).

Therefore the whole algorithm runs in $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(n^3)$ processors on a CREW PRAM.

Example 14 To get a better intuition of how the algorithm works look at the following cubic graph:

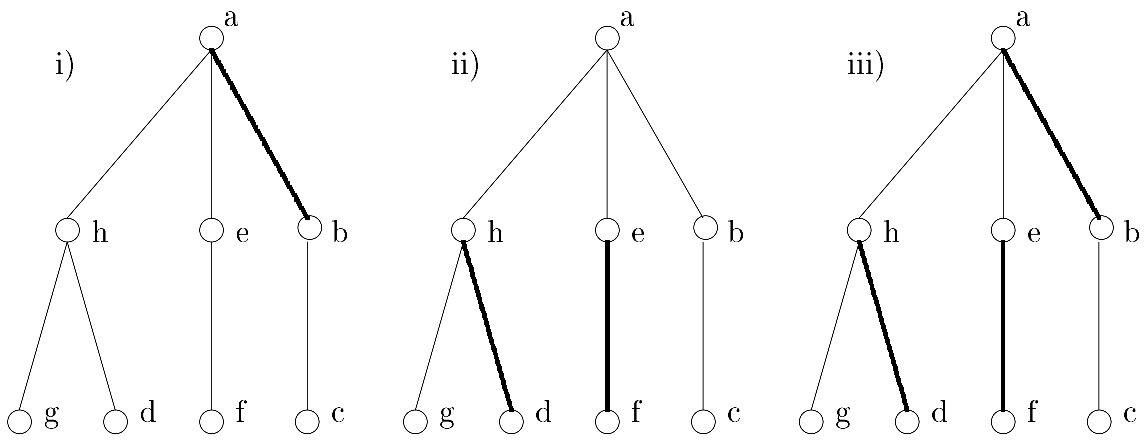


Starting at node a we get the following BFS-tree in step 1.

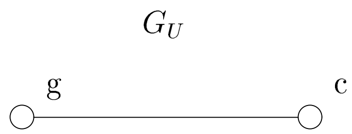


The level numbers start with 0 at the root and increase with the depth of the tree.

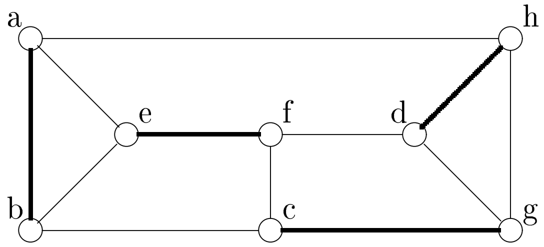
In step 3 we first match the nodes at the even levels and then the nodes at the odd levels. We have to do this to avoid concurrent writes. Here we chose an edge by marking the edge to the lowest order child.



The complete matching M is shown in iii). The only unmatched nodes after that step are the vertices c and g. Therefore the maximum matching M' is the only edge in G_U.



As a result we return $M \cup M'$.



This is in fact a perfect matching; but in general we are only guaranteed to obtain a maximal matching.

2 Maximal Independent Set Algorithm

We consider an algorithm to find a minimal independent set (MIS) in an arbitrary, undirected graph. It was developed by Luby in 1985. Our presentation follows the adaption given in [MR].

It uses randomization as well as parallelism. The expected running time is $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(m+n)$ processors on a EREW PRAM.

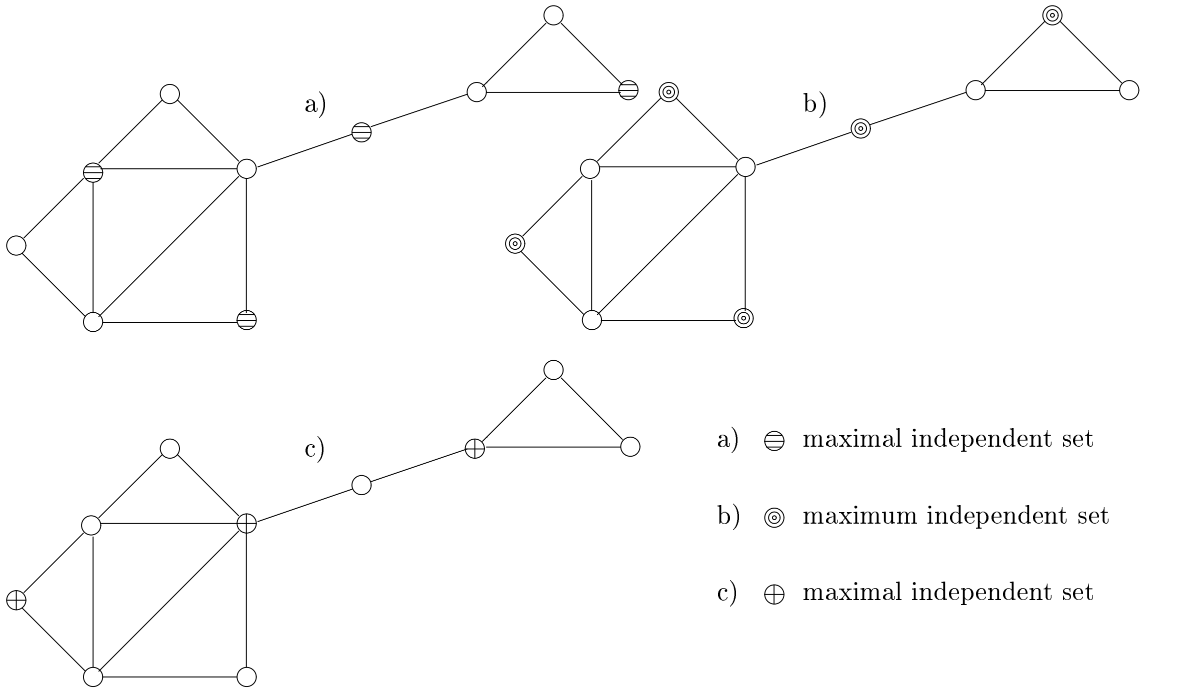
2.1 Independent Sets

Independent sets and vertex covers are dual notions.

Definition 15 (Independent Sets and Vertex Covers)

1. $I \subseteq V$ is an independent set if every edge $e \in E$ is incident on at most one vertex in I.
2. $I \subseteq V$ is an maximal independent set (MIS) if I is an independent set and for every independent set I' we have $I' \supseteq I \Rightarrow I' = I$.
3. $I \subseteq V$ is an maximum independent set if I is an independent set and for every independent set I' we have $|I'| \leq |I|$.
4. $C \subseteq V$ is a vertex cover if every edge $e \in E$ is incident to at least one vertex in C.
5. $C \subseteq V$ is a minimal vertex cover if C is a vertex cover and for every vertex cover C' we have $C' \subseteq C \Rightarrow C' = C$.
6. $C \subseteq V$ is a minimum vertex cover if C is a vertex cover and for every vertex cover C' we have $|C| \leq |C'|$.

Example 16 Some independent sets:



As we have ten vertices with degree ≥ 2 we can hope to get at most five vertices in a maximum independent set. Therefore b) is maximum.

There is a certain relationship between independent sets and vertex covers. The next fact illustrates this.

Fact 17 *Let $G = (V, E)$ be an undirected graph.*

1. *$I \subseteq V$ is an independent set iff $(V - I)$ is a vertex cover.*
2. *Finding a maximum independent set is NP-hard. ([CLR] problem 36-1)*
3. *Finding a minimum vertex cover is NP-hard. ([CLR] §36.5)*
4. *The complement of a maximum (resp. maximal) independent set is a minimum (resp. minimal) vertex cover.*

2.2 The Greedy-Mis Algorithm

When we are looking for a minimal independent set we can safely ignore neighbors of already selected nodes. Therefore we define a few auxiliary functions.

Definition 18 (Neighbors) Let $G = (V, E)$ be an undirected graph.

- For every $v \in V$ the function $\Gamma(v) := \{u|(u, v) \in E\}$ returns the set of neighbors of v.
- For every $S \subseteq V$ the function $\Gamma(S) := \cup_{v \in S}\Gamma(v)$ returns the neighborhood of S.
- The function $d(v) := |\Gamma(v)|$ is the degree of v.

Therefore $I \subseteq V$ is an independent set if we have $\Gamma(v) \cap I = \varnothing$ for every $v \in I$. We can easily come up with the following sequential greedy algorithm to find a MIS in a graph.

```
Greedy-Mis(G)
    I ← ∅
    for v ← 1 to n
    do if I ∩ Γ(v) = ∅
        then I ← I ∪ {v}
    return I
```

Exercise 19 Prove that if G is represented by a adjacency list then Greedy-Mis runs in $\mathcal{O}(m)$ time.

The result of the Greedy-Mis is called lexicographically first MIS because in each iteration the next node in order is chosen. It is considered to be impossible to achieve the same result in parallel unless the parallel algorithm degenerates to a sequential one. However we can find parallel algorithms that give us an arbitrary maximal independent set.

Exercise 20 Describe an EREW algorithm to verify whether a given $I \subseteq V$ is a MIS in $\mathcal{O}(\log\ m)$ time using $\mathcal{O}(m/\log\ m)$ processors.

Now we are ready to look at the parallel randomized algorithm. It is similar to the greedy version in the way that it chooses a set S from the input graph and adds it to the solution. It uses $\mathcal{O}(n+m)$ processors on a EREW PRAM machine.
The instructions in the main loop can be subdivided into five functional groups. The three parallel-for statements (lines 4-9, 10-12, 13-15), the expansion of the solution set (line 16) and the reduction of the input set (lines 17 and 18). Randomization takes place in line 8 where the function RANDOM is supposed to return a real number between it's two arguments. Nodes with a lower degree have a better chance to be accepted as candidates for the solution than those with higher degrees. But note that in line 12 where collisions are resolved we favor the nodes with higher degree. It turned out to work better than the "obvious" solution of choosing the node with lower degree. Ties in this step are broken arbitrarily.

PARALLEL-MIS(G)
1. $I \leftarrow \varnothing$
2. **repeat**
3. $\quad S \leftarrow \varnothing$
4. $\quad$ **for** all $v \in V$ in parallel
5. $\quad$ **do if** $d(v) = 0$
6. $\quad\quad$ **then** add v to I
7. $\quad\quad\quad$ delete v from V
8. $\quad\quad$ **else if** RANDOM(0,1) $< \frac{1}{2\ d(v)}$
9. $\quad\quad\quad$ **then** mark v
10. $\quad$ **for** all $(u,v) \in E$ in parallel
11. $\quad$ **do if** both u and v are marked
12. $\quad\quad$ **then** unmark the vertex with lower degree in $\{u,v\}$
13. $\quad$ **for** all $v \in V$ in parallel
14. $\quad$ **do if** v is marked
15. $\quad\quad$ **then** $S \leftarrow S \cup \{v\}$
16. $\quad I \leftarrow I \cup S$
17. $\quad V \leftarrow V - (S \cup \Gamma(S))$
18. $\quad E \leftarrow E - \{(u,v) \in V \times V \mid u \in S \cup \Gamma(S)\}$
19. **until** $V = \varnothing$
20. **return** I

Exercise 21 Show that each iteration of PARALLEL-MIS can be implemented with running time $\mathcal{O}(\log\ n)$ using a EREW PRAM with $\mathcal{O}(m+n)$ processors.

Lemma 22 *The expected number of iterations of* PARALLEL-MIS *is* $\mathcal{O}(\log\ n)$.

Proof Idea: The proof is shown in detail in [MR]. It uses the fact that the expected number of edges removed in line 18 is at least a constant fraction of the current set of edges.

Theorem 23 PARALLEL-MIS *can be run on a EREW PRAM in expected time* $\mathcal{O}(\log^2 n)$ *using* $\mathcal{O}(m+n)$ *processors.*

Proof Idea: This follows from the results of Exercise 21 and Lemma 22.

It is also possible to derandomize this algorithm (as shown in [MR]). The deterministic version runs in polylogarithmic time (*i.e.* $\mathcal{O}(\log^c n)$) and uses $\mathcal{O}(n^c)$ processors where c is a constant.

The randomized algorithm may run infinitely long with a very low probability if the implementation of the floating point numbers (*i.e.* real numbers) is correct. If floats are implemented with fixed size variables the algorithm may never terminate in any case. For example if we have a fully connected graph where $|V| = 10^{40}$ the probability to mark a node in line 9 is smaller than the precision of a normal float in C ($1.175494351 \times 10^{-38}$).

Material in this handout is partly drawn from:

- Article on Cubic Graphs in ACM Computing Surveys Vol.27 No.4 December 95 by Greenlaw and Petreshi
- [MR] Chapter 12.3 in Randomized Algorithms by Motwani and Raghavan
- [CLR] Introduction to Algorithms by Cormen, Leiserson and Rivest

---

Alexander Sorg

Created: April 18, 1996