CMSC 641 Design & Analysis of Algorithms Spring 2023
Section 01 Homework 11

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

1 Special Partition

Sometimes the special case of an NP-complete problem remains NP-complete. For example, the 3-Color problem restricted to planar graphs remains NP-complete. For this problem we consider the opposite situation: a special case of Partition that can be solved by a greedy algorithm. Recall that in the Partition problem, we are given n numbers a1, . . . , an. The question is whether there is partition of the indices {1, . . . , n} into sets A and B such that

$$\sum_{i\epsilon A} a_i = \sum_{j\epsilon B} a_j$$

Here, A and B forms a partition means A ∩ B = ∅ and A ∪ B = {1, . . . , n}. In this special case, all of the numbers are powers of 2. That is, the ai are chosen from {1, 2, 4, 8, 16, . . .}. Duplicates are allowed, so there maybe multiple 4's for example.

Assignment:

1. First prove this useful lemma: Lemma: Let x0, x1, . . . , xk ∈ N such that
   $x_0 2^0 + x_1 2^1 + x_2 2^2 + \cdots + x_k 2^k \geq 2^{k+1}$
   then there exists y0, y1, . . . , yk ∈ N such that each yi ≤ xi and
   $y_0 2^0 + y_1 2^1 + y_2 2^2 + \cdots + y_k 2^k = 2^{k+1}$
   The lemma says that if there is a way to combine lower powers of 2 to exceed 2^{k+1}, then there is a way to combine a subset of them to total exactly 2^{k+1}. Prove the lemma above using proof by induction.

A. <u>Proof of Induction using above Lemma</u>:

<u>Induction Hypothesis P(n):</u>
there is a way to combine lower powers of 2 to exceed 2^{k+1}, then there is a way to combine a subset of them to total exactly 2^{k+1}

<u>Base Case:</u>
When k=0,
$x_0 2^0 \geq 2^{0+1}$
$x_0 \geq 2$
we have $y_0 \leq x_0$
there should be atleast 2 total values of 2 . Therefore x_0 =1 in Set A and y_0=1 in Set B.
The base condition holds.

<u>Induction Case:</u>
Assume P(t) holds for all t<n. We want to show that P(n) holds.
Let t=3, then we have
$x_0 2^0 + x_1 2^1 + x_2 2^2 + x_3 2^3 \geq 2^{3+1}$

$x_0 2^0 + x_1 2^1 + x_2 2^2 + x_3 2^3 \geq 2^4$

We $y_i \leq x_i$ assuming $y_i = x_i$,

Consider these values x0=0,x1=4,x2=4,x3=1 as x0, x1, . . . , xk ∈ N

$0\ 2^0 + 4\ 2^1 + 4\ 2^2 + 1\ 2^3 \geq 2^4$

We have this condition $y_i \leq x_i$ assuming $y_i = x_i$,

To satisfy the set property of having the same sum in A & B,

y0=0,y1=2,y2=3,y3=0 as y0, y1, . . . , yk ∈ N such that each yi ≤ xi

$y_0 2^0 + y_1 2^1 + y_2 2^2 + y_3 2^3 = 2^{3+1}$

$0\ 2^0 + 2\ 2^1 + 3\ 2^2 + 0\ 2^3 = 2^4$

$2^4 = 2^4$

Therefore, with the help of the lemma given we can have the assignment which satisfies the given conditions.

This gives a feasible solution of assigning elements to set A & B such that there sums are equal.

By the Induction Hypothesis, we know that there is a way to combine lower powers of 2 to exceed 2^{k+1} , then there is a way to combine a subset of them to total exactly 2^{k+1} .

Thus,

For x0, x1, . . . , xk ∈ N

$$x_0 2^0 + x_1 2^1 + x_2 2^2 + \cdots + x_k 2^k \geq 2^{k+1}$$

then there exists y0, y1, . . . , yk ∈ N such that each yi ≤ xi and

$$y_0 2^0 + y_1 2^1 + y_2 2^2 + \cdots + y_k 2^k = 2^{k+1}$$

Therefore, P(n) holds.

2. Devise and describe a greedy algorithm that finds a partition A and B of {1, 2, 3, . . . , n} such that

   $\sum_{i \epsilon A} a_i = \sum_{j \epsilon B} a_j$

   where each ai is a power of 2. Your algorithm should report either the sets A and B, or declare that no such partition is possible.

   Note: the fact that {a1, . . . , an} may contain repeated values is an important part of this problem. You should find the lemma above useful.

A. <u>Greedy Algorithm that finds a partition A and B:</u>
   - Let (1,2,…,n) be the given input
   - Initially the set A and set B will be empty, and their sum is 0.
   - Sort the given input in descending order
   - Now iterate the given input which is sorted in descending order:

     If both sets are empty, then add initial item in Set A and update its sum.

     After that,

     For each item:

If adding the element to Set A makes the total sum greater than Set B sum then add the element to Set B and update sum. Otherwise add the element to Set A(i.e. adding the element makes sum of Set A less than or equal to Set B sum) and update the sum.

- Repeat the above step for every item.
- If the sum of set A is equal to sum of Set B then partition is there, else partition is not possible.

3. State and briefly justify the running time of your algorithm.

A. According to our algorithm we first sort the elements in decreasing order. Assuming best sorting algorithm Mergesort it takes O(n log n).
Iterating over each input takes O(n) times as n elements are there.
Running time = O(n log n) + O(n)
= O(n log n)

4. Argue that any partition A and B produced by your algorithm does indeed have

$\sum_{i \epsilon A} a_i = \sum_{j \epsilon B} a_j$

A. Let S be the sum of all the numbers in the input set {a1, a2, ..., an}. The greedy algorithm first sorts the numbers in decreasing order, so if there exists a partition A and B with
$\sum_{i \epsilon A} a_i = \sum_{j \epsilon B} a_j$
then there must exist element in partition A where the largest power of 2 in A is greater than or equal to the largest power of 2 in B as our algorithm is defined like that. The algorithm doesn't add element in Set/Partition A unless the partition B doesn't get greater than or equal to sum of A.
This will ensure that the sum of both the partitions maintain to equal.
The above greedy algorithm will ensure that we indeed have sum of both the partitions equal i.e.
$\sum_{i \epsilon A} a_i = \sum_{j \epsilon B} a_j$

5. Argue that if your algorithm declares that no partition is possible, it is indeed correct.
Hint: it is easier to prove the contrapositive that if some partition A^{l} and B^{l} of the indices exist then your algorithm will report a partition A and B.

A. To show that if the algorithm declares that no partition is possible, it is indeed correct, we can use a proof by contradiction.
<u>Proof by contradiction:</u>

Suppose the algorithm declares that no partition is possible, but there exists some partition A^{l} and B^{l} such that its elements make the sum of Set A and Set B equal. This means that there is a way to partition the set {1, 2, ..., n} into two sets A and B such that the sum of the elements in A is equal to the sum of the elements in B i.e. $\sum_{i\epsilon A} a_i = \sum_{j\epsilon B} a_j$

Since our algorithm cannot find such a partition, this means that the greedy algorithm, did not consider all the elements. This one contradicts, as our greedy algorithm considers all the elements given. It has not considered the elements in Set $A^{|}$ and $B^{|}$ contradicts statement that the partition of Set $A^{|}$ and $B^{|}$ gives elements and Set A and B sum becomes equal. Therefore, our algorithm declares that no partition is possible, it is indeed correct.

2 Improving Approximations for Clique

In this problem, we will show that it is possible to improve the approximation factor for any algorithm that approximates the Clique problem. Much of the construction is given by the problem. You have to complete the tasks that are embedded below.

Let G = (V, E) be any undirected graph and let n = |V |. We construct a new graph $G^{(4)}$ that has n^4 vertices. Each vertex of $G^{(4)}$ is a 4-tuple of vertices in G:

$V^{(4)}$ = { (v1, v2, v3, v4) | vi ∈ V, for i = 1, 2, 3, 4 }.

If we used Cartesian product notation, $V^{(4)}$ = V × V × V × V .

Now, we define the set of edges for $G^{(4)}$. Given two vertices (u1, u2, u3, u4) and (v1, v2, v3, v4), we place an edge between these two vertices if for every i = 1, 2, 3, 4, either ui = vi or (ui , vi) ∈ E. Let's call this set of edges E(4). Then, we have fully defined $G^{(4)}$ by saying $G^{(4)}$ = ($V^{(4)}$, $E^{(4)}$).

Task 1: Let C ⊆ V be a clique in G. Argue that $C^{(4)}$ = { (v1, v2, v3, v4) | vi ∈ C, for i = 1, 2, 3, 4 } must be a clique in $G^{(4)}$.

A. Given C ⊆ V a clique in G.
   Need to show that $C^{(4)}$ = { (v1, v2, v3, v4) | vi ∈ C, for i = 1, 2, 3, 4 } a clique in $G^{(4)}$.
   We define edges b/w 2 vertices (u1, u2, u3, u4) and (v1, v2, v3, v4) for $G^{(4)}$ as for every i = 1, 2, 3, 4, either ui = vi or (ui , vi) ∈ E.
   The vertices (v1, v2, v3, v4) must have projected edges to the C clique in G on the dimensional plane for $G^{(4)}$ as we have been given that C is a clique.
   There can be edges to other vertices also which are not in clique in G.
   As $C^{(4)}$ will have all edges to vertices of the Clique C in G.
   Therefore $C^{(4)}$ must be a clique in $G^{(4)}$.

Task 2: Let $C^{|}$ ⊆ $V^{(4)}$ be a clique in $G^{(4)}$. Argue that

C1 = { x | x = v1 for some (v1, v2, v3, v4) ∈ $C^{|}$ } must be a clique in G.

For $C^{|}$, we can similarly define

C2 = { x | x = v2 for some (v1, v2, v3, v4) ∈ $C^{|}$ }

C3 = { x | x = v3 for some (v1, v2, v3, v4) ∈ $C^{|}$ }

C4 = { x | x = v4 for some (v1, v2, v3, v4) ∈ $C^{|}$ }.

Using the same argument as for C1, each Ci must also be a clique in G.

A. Given $C^l \subseteq V^{(4)}$ a clique in $G^{(4)}$.
   Need to show that C1 = { x | x = v1 for some (v1, v2, v3, v4) ∈ C^l } must be a clique in G.
   Here one plane and the its corresponding vertex(v1 for C1) is fixed.
   v1 has every edge connected to the other vertices of the planes.
   We also know that (v1, v2, v3, v4) ∈ C^l,
   From task 1 we have proved that Given C ⊆ V a clique in G, $C^{(4)}$ = { (v1, v2, v3, v4) | vi ∈ C, for i = 1, 2, 3, 4 } is a clique in $G^{(4)}$.
   v1 is in C^l then, the vertex of C^l might also have clique in G, as we proved that in task 1.
   v1 itself is a vertex of C clique in G.
   Therefore, C1 = { x | x = v1 for some (v1, v2, v3, v4) ∈ C^l } must be a clique in G.
   Similarly these are also clique in C:

   C2 = { x | x = v2 for some (v1, v2, v3, v4) ∈ C^l }

   C3 = { x | x = v3 for some (v1, v2, v3, v4) ∈ C^l }

   C4 = { x | x = v4 for some (v1, v2, v3, v4) ∈ C^l }.

Task 3: Argue that $|C^l| \le |C1| \cdot |C2| \cdot |C3| \cdot |C4|$.

A. From task 2 we came to conclusion that C1,C2,C3,C4 are clique in G.
   They also have same number of vertices in G.
   C^l is a clique in $G^{(4)}$and have subset of vertices of $G^{(4)}$, $C^l \subseteq V^{(4)}$
   At most it largest clique in $G^{(4)}$ has
   $C^l \leq V^{(4)}$
   C1,C2,C3,C4 are clique in G and has equal vertices to C^l when combined together but that's the largest clique case for C^l
   $|C^l|$ some times may not have the largest clique but it will not be more than ($|C1| \cdot |C2| \cdot |C3| \cdot |C4|$).
   Therefore, $|C^l| \le |C1| \cdot |C2| \cdot |C3| \cdot |C4|$.

Suppose that the largest clique in G has k vertices and the largest clique in $G^{(4)}$ has k^l vertices.

Using the above, we can show a tight relationship between k and k^l .

Task 4: Show that $k^l \ge k^4$.

A. We know that the largest clique in G has k vertices and the largest clique in $G^{(4)}$ has k^l vertices.
   C ⊆ V a clique in G and $C^l \subseteq V^{(4)}$ a clique in $G^{(4)}$.
   Since the graph is connected with cartesian planar for $G^{(4)}$ there can be edge connected i.e. ui to vi in $G^{(4)}$ (as there are 4 times the vertex compared to G) but not in G as it is simple one dimensional graph.
   $G^{(4)}$ has many edges projected b/w planar so $G^{(4)}$ vertices are to power of 4 vertices to G.
   Since G has k vertices; $G^{(4)}$ has k^4 vertices atleast.
   Therefore $k^l \ge k^4$

Task 5: Show that $k^l \le k^4$.

A. $G^{(4)}$ will have 4 dimension projection of graph G.
G has k vertices and the largest clique in $G^{(4)}$ has k^l vertices.
From task 3 we have, $| C^l | \le |C1| \cdot |C2| \cdot |C3| \cdot |C4|$

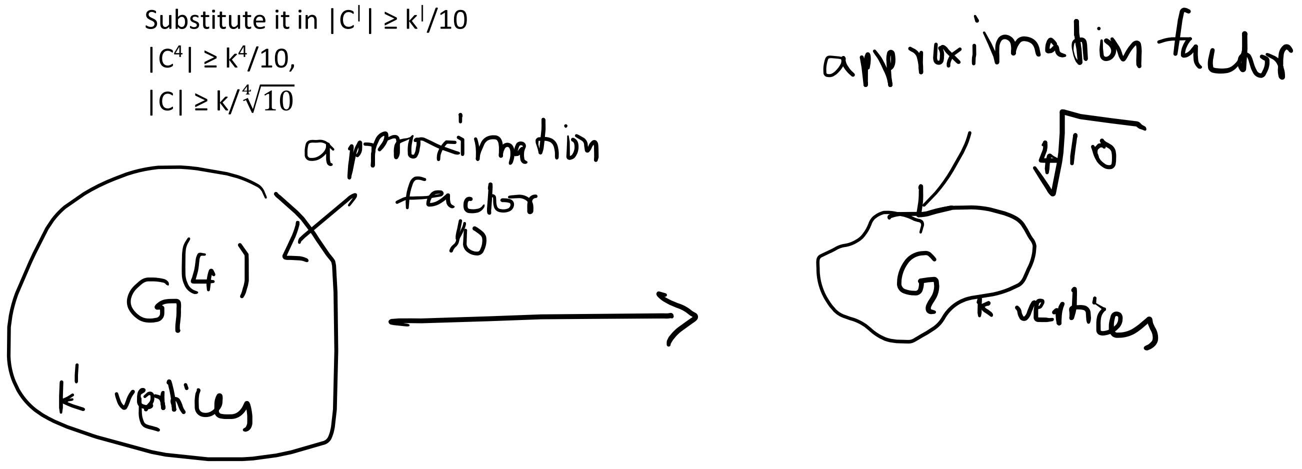
$k^l \le k \;.\; k \;.\; k \;.\; k$

$k^l \le k^4$

Thus, we can conclude that k^l is exactly equal to k^4 . Now suppose that you are given a clique C^l in $G^{(4)}$ along with the guarantee that C^l is a factor 10 approximation. That is $|C^l| \ge k^l/10$ where k^l is once again the size of the largest clique in $G^{(4)}$.

Task 6: Explain how to construct a clique $C \subseteq V$ in G so that $|C| \ge k/\sqrt[4]{10}$ where k is the size of the largest clique in G. Do explain why C must have at least $k/\sqrt[4]{10}$ vertices.

A. We can construct a clique $C \subseteq V$ in G so that $|C| \ge k/\sqrt[4]{10}$ where k is the size of the largest clique in G from $G^{(4)}$.
As we are known that $k^l = k^4$, which is given and also $G^{(4)}$ has approximation factor of 10 i.e.
$|C^l| \ge k^l/10$, clique C^l in $G^{(4)}$
We know that $C = (C^l)^4$
$k=(k^l)^4$
Substitute it in $|C^l| \ge k^l/10$
$|C^4| \ge k^4/10$,
$|C| \ge k/\sqrt[4]{10}$



So according to the construction of the graph G from Graph $G^{(4)}$
We are approximating the factor of 10 from $G^{(4)}$ to factor of $\sqrt[4]{10}$ for G.
$|C| \ge k/\sqrt[4]{10}$
Therefore, C has atleast $k/\sqrt[4]{10}$ vertices.

Suppose that some algorithm A can always find a clique in any graph and guarantee that the size of that clique is within a factor of 10 of the largest clique. Then, we can run A on $G^{(4)}$ and obtain C^{l} . Using your construction above, we can get a C that is within a factor of $\sqrt[4]{10}$ of k. Since $\sqrt[4]{10} \approx 1.78$, we have improved the approximation factor from 10 to about 1.78. The new algorithm is: given G, $G^{(4)}$, run A on $G^{(4)}$ to get C^{l} , construct C from C^{l} and output C.

Presumably there was nothing special about the factor 10 in your construction and you can turn an algorithm that achieves a factor r approximation of Clique into an algorithm that achieves a factor $\sqrt[4]{r}$ approximation. For example, if we repeated the process on the factor $\sqrt[4]{10}$ algorithm, we get a factor $\sqrt[4]{\sqrt[4]{10}} \approx 1.15$ algorithm. If we do this yet again, we get a factor $\sqrt[4]{\sqrt[4]{\sqrt[4]{10}}} \approx 1.04$ algorithm. We can get arbitrarily close to a factor of 1, but we pay a price in running time.

Task 7: Suppose that we have an algorithm A that produces a factor 10 approximation of Clique and that A runs in $\Theta(n^d)$ time on graphs with n vertices. Using the process above we can construct an algorithm that produces a factor $(1 + \in)$ approximation of Clique. Bound the running time of such an algorithm. Assume that it takes $\Theta(n^2)$ time to construct a graph with n vertices and that $d \geq 2$. Give the running time in terms of n, d and $\in$. Show your work.

A. Algorithm A with factor 10 approximation of Clique runs in $\Theta(n^d)$ time on graphs with n vertices.
We do the process as follows:
given G, $G^{(4)}$, run A on $G^{(4)}$ to get C^{l} , construct C from C^{l} and output C.
algorithm which produces a factor $(1 + \in)$ approximation of Clique, running time
Running time = time to construct a graph with n vertices + time to run a graph based on the approximation factor of $(1 + \in)$.
NP optimization problems with $(1 + \in)$ approximation factor, for every $\in > 0$, running time may have $1/\in$ in the exponent.
Running time = $\Theta(n^2) + \Theta(n^{\frac{1^d}{\in}})$
$= \Theta(n^2) + \Theta(n^{\frac{d}{\in}})$
$\Theta(n^{\frac{d}{\in}})$ this term is the dominating term.

Task 8: Using your calculations above, suppose that d = 2 and you want an algorithm that produces a factor 1.001 approximation of Clique. What is the running time of that algorithm? Give your answer as $\Theta(n^l)$ and provide an actual number for l. Show your work.

A. Given d=2 and approximation factor, $1 + \in = 1.001$
$\in = 0.001$
Running time = $\Theta(n^2) + \Theta(n^{\frac{d}{\in}})$
$= \Theta(n^2) + \Theta(n^{\frac{2}{0.001}})$
$= \Theta(n^2) + \Theta(n^{2000})$
This is exponentially very large and might not be practical in real world, l=2000

References:

https://archive.org/details/introduction-to-algorithms-by-thomas-h.-cormen-charles-e.-leiserson-ronald.pdf/page/427/mode/2up

https://archive.org/details/AlgorithmDesign1stEditionByJonKleinbergAndEvaTardos2005PDF/page/n201/mode/2up

16-NP-Completeness-post.pdf - Google Drive

20.1-Partition-post.pdf - Google Drive

21.1-Approximation-Algorithms-post.pdf - Google Drive

21.2-Approximating-TSP-post.pdf - Google Drive

22-Bin-Packing-post.pdf - Google Drive

23.2-Subset-Sum-post.pdf - Google Drive

23.1-Approximating-Set-Cover-v2-post.pdf - Google Drive