問題 1

Σ を文字の集合 $\{a, b\}$ とする．Σ 上の言語 $L \subseteq \Sigma^*$ に対して，$\Gamma(L)$ を以下のように定める．

$$\Gamma(L) = \{v \in \Sigma^* \mid \exists w \in \Sigma^*.(|v| = |w| \wedge vw \in L)\}.$$

ただし，$|x|$ は文字列 x の長さを表す．例えば，$L_1 = \{aa, ba, abb, abbb\}$ とすると，$\Gamma(L_1) = \{a, b, ab\}$ である．

以下の問いに答えよ．

(1) $L_2 = \{(ab)^n \mid n \geq 0\}$ とする．$\Gamma(L_2)$ を正規表現を用いて表せ．

(2) $L_3 = \{a^n b^n a^m b^m \mid n \geq 0, m \geq 0\}$ とする．$\Gamma(L_3)$ を生成する文脈自由文法を与えよ．

(3) $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ を決定性有限オートマトン，$L_{\mathcal{M}}$ を $\mathcal{M}$ が受理する言語とする．ただし，Q, δ, q_0, F は $\mathcal{M}$ の状態集合，遷移関数，初期状態，受理状態集合をそれぞれ表すものとする．遷移関数 $\delta : Q \times \Sigma \to Q$ は全域関数であると仮定してよい．$\Gamma(L_{\mathcal{M}})$ を受理する有限オートマトンを，簡単な説明とともに与えよ．

(4) 以下の命題が真ならば，L を生成する文脈自由文法から $\Gamma(L)$ を生成する文脈自由文法を構成する方法を簡単な説明とともに与えよ（文脈自由文法の代わりにプッシュダウンオートマトンを用いてもよい）．偽ならば，簡単な説明とともに反例を示せ．
命題：「いかなる文脈自由言語 $L \subseteq \Sigma^*$ についても，$\Gamma(L)$ は文脈自由言語である．」

Problem 1

Let Σ be the set $\{a, b\}$ of letters. For a language $L \subseteq \Sigma^*$ over Σ, we define $\Gamma(L)$ as follows.

$$\Gamma(L) = \{v \in \Sigma^* \mid \exists w \in \Sigma^*.(|v| = |w| \wedge vw \in L)\}.$$

Here, $|x|$ denotes the length of the string x. For example, if $L_1 = \{aa, ba, abb, abbb\}$, then $\Gamma(L_1) = \{a, b, ab\}$.

Answer the following questions.

(1) Let $L_2 = \{(ab)^n \mid n \geq 0\}$. Express $\Gamma(L_2)$ using a regular expression.

(2) Let $L_3 = \{a^n b^n a^m b^m \mid n \geq 0,\ m \geq 0\}$. Give a context-free grammar that generates $\Gamma(L_3)$.

(3) Let $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton, and let $L_{\mathcal{M}}$ be the language accepted by $\mathcal{M}$. Here, Q, δ, q_0, and F are the set of states, the transition function, the initial state, and the set of final states of $\mathcal{M}$, respectively. You may assume that the transition function $\delta : Q \times \Sigma \to Q$ is total. Give a finite automaton that accepts $\Gamma(L_{\mathcal{M}})$, with a brief explanation.

(4) If the proposition given below is true, then give how to construct a context-free grammar that generates $\Gamma(L)$, from a context-free grammar that generates L (you may use push-down automata instead of context-free grammars), with a brief explanation. Otherwise, give a counterexample, with a brief explanation.
Proposition: "For every context-free language $L \subseteq \Sigma^*$, $\Gamma(L)$ is a context-free language."

問題2

相異なるn個の正整数からなる集合$P=\{x_1,x_2,\ldots,x_n\}$をm個の集合$P_1,P_2,\ldots,P_m$ $(1<m<n,$ $P=P_1\cup P_2\cup\ldots\cup P_m,$ $\forall i,j(i\neq j)$ $P_i\cap P_j=\emptyset)$ に分割することを考える．ここで，$\emptyset$は空集合を表す．このとき，集合列$\mathcal{Q}=[P_1,P_2,\ldots,P_m]$を$P$の分割とよぶ．以下では整数集合$S$，あるいは正整数が格納されたスタック$S$に対し，それに含まれる整数の和を$\|S\|$と表記する．なお，$S$が空集合あるいは空スタックである場合は$\|S\|=0$とする．

Pの分割$\mathcal{Q}=[P_1,P_2,\ldots,P_m]$に対し，$maxsum(\mathcal{Q})=\max_i\|P_i\|$と定義する．$P$の$m$個の集合への分割$\mathcal{Q}$として考えられるものすべての中で$maxsum(\mathcal{Q})$がとり得る最小値を$minmaxsum(P,m)$とおく．

以下の疑似コードは$minmaxsum(P,m)$の近似値を求めるアルゴリズムである．ただし以下で，$push(S,x)$はスタックSにxをプッシュする手続き，$pop(S)$はスタックSのトップ要素をポップし，ポップした要素を返す手続き，$top(S)$はスタックSのトップ要素を返す手続きである．なお，$top(S)$と$pop(S)$は同じSに対し同じ値を返すが，$top(S)$はスタックの状態を変更しない．

```
1:  approx_minmaxsum( 整数集合 P, 整数 m) {
2:    Q = [P_1, P_2, ..., P_m] ← P の m 個の集合への任意の分割;
3:    for (1 ≤ i ≤ m) {
4:      S_i ← 空スタック;
5:      foreach (x ∈ P_i) { push(S_i, x); }
6:    }
7:    while(1) {
8:      j ← argmax_i ||S_i||; /* j ← ||S_i|| が最大である i のひとつ */
9:      k ← argmin_i ||S_i||; /* k ← ||S_i|| が最小である i のひとつ */
10:     if (top(S_j) + ||S_k|| ≥ ||S_j||) break;
11:     push(S_k, pop(S_j));
12:   }
13:   return ||S_j||;
14: }
```

以下の問いに答えよ．

(1) $minmaxsum(\{3,4,5,6\},2)$を求めよ．

(2) $minmaxsum(P,m)\geq\|P\|/m$であることを示せ．

(3) 上の疑似コードにおいて，2行目でどのような分割$\mathcal{Q}$を選択したとしても，$approx_minmaxsum(P,m)\leq 2\cdot minmaxsum(P,m)$が成り立つことを示せ．

(4) 上の疑似コードにおいて，2行目でどのような分割$\mathcal{Q}$を選択したとしても，whileループの繰り返し回数が高々n回であることを示せ．

(5) 上記アルゴリズムの実行時間を$O(n\log m)$とするために必要なデータ構造およびその使用法を説明せよ．

Problem 2

We consider a division of a set of mutually distinct n positive integers $P = \{x_1, x_2, \ldots, x_n\}$ into m sets $P_1, P_2, \ldots, P_m$ ($1 < m < n$, $P = P_1 \cup P_2 \cup \ldots \cup P_m$, $\forall i, j (i \neq j)\ P_i \cap P_j = \emptyset$), where $\emptyset$ denotes an empty set. The set sequence $\mathcal{Q} = [P_1, P_2, \ldots, P_m]$ is called a division of P. We denote by $\|S\|$ the summation of all the integers in S if S is a set of integers or a stack consisting of integers. Note that $\|S\| = 0$ in case S is an empty set or an empty stack.

Let $maxsum(\mathcal{Q}) = \max_i \|P_i\|$ for a division $\mathcal{Q} = [P_1, P_2, \ldots, P_m]$ of P. Let $minmaxsum(P, m)$ denote the minimum value of $maxsum(\mathcal{Q})$ among all the possible divisions $\mathcal{Q}$ of P into m sets.

The following pseudo code shows an algorithm that computes an approximation of $minmaxsum(P, m)$. Below, $push(S, x)$ pushes x onto the stack S, $pop(S)$ pops the top element of the stack S and returns the popped element, and $top(S)$ returns the top element of the stack S. Note that $top(S)$ and $pop(S)$ return the same value for the same stack S, but $top(S)$ does not modify the stack.

```
 1: approx_minmaxsum(integer set P, integer m) {
 2:   Q = [P_1, P_2, ..., P_m] ← An arbitrary division of P into m sets;
 3:   for (1 ≤ i ≤ m) {
 4:     S_i ← an empty stack;
 5:     foreach (x ∈ P_i) { push(S_i, x); }
 6:   }
 7:   while(1) {
 8:     j ← argmax_i ||S_i||;  /* j ← one of the i's that maximize ||S_i|| */
 9:     k ← argmin_i ||S_i||;  /* k ← one of the i's that minimize ||S_i|| */
10:     if (top(S_j) + ||S_k|| ≥ ||S_j||) break;
11:     push(S_k, pop(S_j));
12:   }
13:   return ||S_j||;
14: }
```

Answer the following questions.

(1) Calculate $minmaxsum(\{3, 4, 5, 6\}, 2)$.

(2) Show $minmaxsum(P, m) \geq \|P\|/m$.

(3) Show that $approx\_minmaxsum(P, m) \leq 2 \cdot minmaxsum(P, m)$ holds, regardless of whatever division $\mathcal{Q}$ is chosen in line 2 of the above code.

(4) Show that the while loop in the above code will be repeated at most n times, regardless of whatever division $\mathcal{Q}$ is chosen in line 2.

(5) Describe data structures needed to make the above algorithm run in $O(n \log m)$ time, and explain how to use them.

問題 3

オペレーティングシステムにおいて，ページ置換アルゴリズムはページフォールトの回数を減らすように設計される．アルゴリズムを評価するには，「参照ストリング」と呼ばれる特定のメモリ参照列に対してアルゴリズムを実行し，ページフォールトの回数を数える．ここで，メモリ参照列は一連のページ番号によって表現される．

以下の問いに答えよ．

(1) 最適なページ置換アルゴリズムとして知られているのは，将来最も長い間使用されないページを置換するアルゴリズムである．3 個のページフレームが利用可能で，初期状態ではそれらのページフレームが空である場合に，以下の参照ストリングに対して最適なページ置換アルゴリズムを実行することで発生するページフォールトの回数を答えよ．

参照ストリング: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

(2) 最適なページ置換アルゴリズムは参照ストリングに関する未来の情報が必要になるという点で実装が困難である．その代替手段として，Least Recently Used (LRU) アルゴリズムは過去に最も長い間使用されていないページを置換する．3 個のページフレームが利用可能で，初期状態ではそれらのページフレームが空である場合に，問い (1) で使用した参照ストリングに対して LRU アルゴリズムを実行することで発生するページフォールトの回数を答えよ．

(3) ハードウェア機能としてのカウンタが利用でき，ページテーブルの各エントリにそのカウンタと関連づけられたフィールドが存在する場合に，以下の観点から LRU アルゴリズムの実装方法を述べよ．

- カウンタはいつ増加するか．
- カウンタの値はいつページテーブルのフィールドにコピーされるか．
- 置き換え対象のページはどのように選択されるか．

(4) 実用的な実装の観点から，LRU アルゴリズムの欠点を 1 つ説明せよ．

(5) ページテーブルの各エントリが参照ビットを持っているという仮定のもと，LRU の近似アルゴリズムの実装方法を述べよ．

Problem 3

In an operating system, a page replacement algorithm should be designed to reduce the number of page faults. To evaluate the algorithm, we count the number of page faults caused by running the algorithm on a particular string of memory references, called a "reference string". Here, each memory reference is represented by a page number.

Answer the following questions.

(1) An optimal page replacement algorithm replaces a page that will not be used in future for the longest period of time. Assuming that three page frames are available and they are empty in the initial state, give the number of page faults caused by running the optimal page replacement algorithm on the following reference string.

Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

(2) The optimal page replacement algorithm is difficult to implement, because it requires future knowledge of the reference string. As an alternative of the optimal page replacement algorithm, the Least Recently Used (LRU) algorithm replaces a page that has not been used for the longest period of time. Assuming that three page frames are available and they are empty in the initial state, give the number of page faults caused by running the LRU algorithm on the reference string used in question (1).

(3) Under the assumption that you can use a counter supported by hardware and it is associated with a field contained by each entry of the page table, describe how to implement the LRU algorithm from the following standpoints.

- When is the counter incremented?
- When is the value of the counter copied to the page table field?
- How is the page to be replaced selected?

(4) Explain a drawback of the LRU algorithm from the practical implementation point of view.

(5) Under the assumption that each entry of the page table has a reference bit, describe how to implement an LRU-approximation algorithm.

問題4

コンピュータアーキテクチャに関する以下の問いに答えよ.

(1) 以下に示すC言語のプログラムをあるプロセッサA上でコンパイルし実行すると，出力(a)が得られた．同一のプログラムを別のプロセッサB上でコンパイルし実行すると出力(b)が得られた．プロセッサアーキテクチャの観点から，この違いが生じる理由を説明せよ.

```
#include <stdio.h>

union my_uni {
  int v;
  char arr[4];
};

int main(){
  union my_uni val = {0x12345678};
  int i;
  for(i=0; i<4; i++){
     printf("0x%x\n", val.arr[i]);
  }
  return 0;
}
```

出力 (a)	出力 (b)
0x12	0x78
0x34	0x56
0x56	0x34
0x78	0x12

(2) フォワーディング機構を持たないパイプラインプロセッサにおけるデータハザードと制御ハザードを具体的な例を用いて説明せよ.

(3) 全体で32キビバイト (32×2^{10} バイト) のデータを保持する，4ウェイ・セットアソシアティブ方式のキャッシュメモリを考える．このキャッシュメモリのアドレス幅は32ビット，キャッシュラインサイズ（ブロックサイズ）は64バイトである．このキャッシュメモリのキャッシュ・インデックスとタグのビット幅をそれぞれ求めよ．また，このキャッシュメモリのタグを保持するRAMの総容量（ビット数）を求めよ.

(4) 命令キャッシュとデータキャッシュを搭載するプロセッサを考える．命令キャッシュとデータキャッシュの両方でキャッシュミスが発生しない場合のこのプロセッサのCPI (cycles per instruction) を C とする．いずれかのキャッシュでミスする場合，ミス毎に P クロックサイクルのキャッシュミスペナルティが追加で必要となる．このプロセッサ上で，あるプログラムを実行したところ，全実行命令中のロード・ストア命令の割合が R_{ls} であった．また，そのプログラム実行の際の命令キャッシュのミス率は R_i，データキャッシュのミス率は R_d，IPC (instructions per cycle) は I であった．C，R_i，R_d，R_{ls}，P を用いて I を表せ.

Problem 4

Answer the following questions on computer architecture.

(1) When the following program in C language was compiled and executed on a processor A, the output (a) was obtained. When the same program was compiled and executed on a processor B, the output (b) was obtained. Explain the reason why the difference occurred from the viewpoint of processor architecture.

```
#include <stdio.h>

union my_uni {
  int v;
  char arr[4];
};

int main(){
  union my_uni val = {0x12345678};
  int i;
  for(i=0; i<4; i++){
     printf("0x%x\n", val.arr[i]);
  }
  return 0;
}
```

Output (a)	Output (b)
0x12	0x78
0x34	0x56
0x56	0x34
0x78	0x12

(2) Explain data-hazard and control-hazard on a pipeline processor with no forwarding mechanism, using a concrete example.

(3) Consider a 4-way set-associative cache memory that stores totally 32 kibibytes (32×2^{10} bytes) of data. The address width of the cache memory is 32 bits, and the cache line size (block size) is 64 bytes. Calculate the bit width of the cache index and that of a tag of the cache memory, respectively. Calculate also the total RAM capacity (the number of bits) for storing the tags of the cache memory.

(4) Consider a processor with an instruction cache and a data cache. Suppose that the CPI (cycles per instruction) of the processor is C when there is no cache miss on both the instruction and data caches. When there is a cache miss on any of the caches, a cache miss penalty of P clock cycles is additionally imposed. Suppose that when a program was executed on the processor, the ratio of the number of load/store instructions to the total number of executed instructions was R_{ls}. Suppose also that, for that program execution, the cache miss rate of the instruction cache was R_i, the cache miss rate of the data cache was R_d, and the IPC (instructions per cycle) of the processor was I. Express I in terms of C, R_i, R_d, R_{ls}, and P.